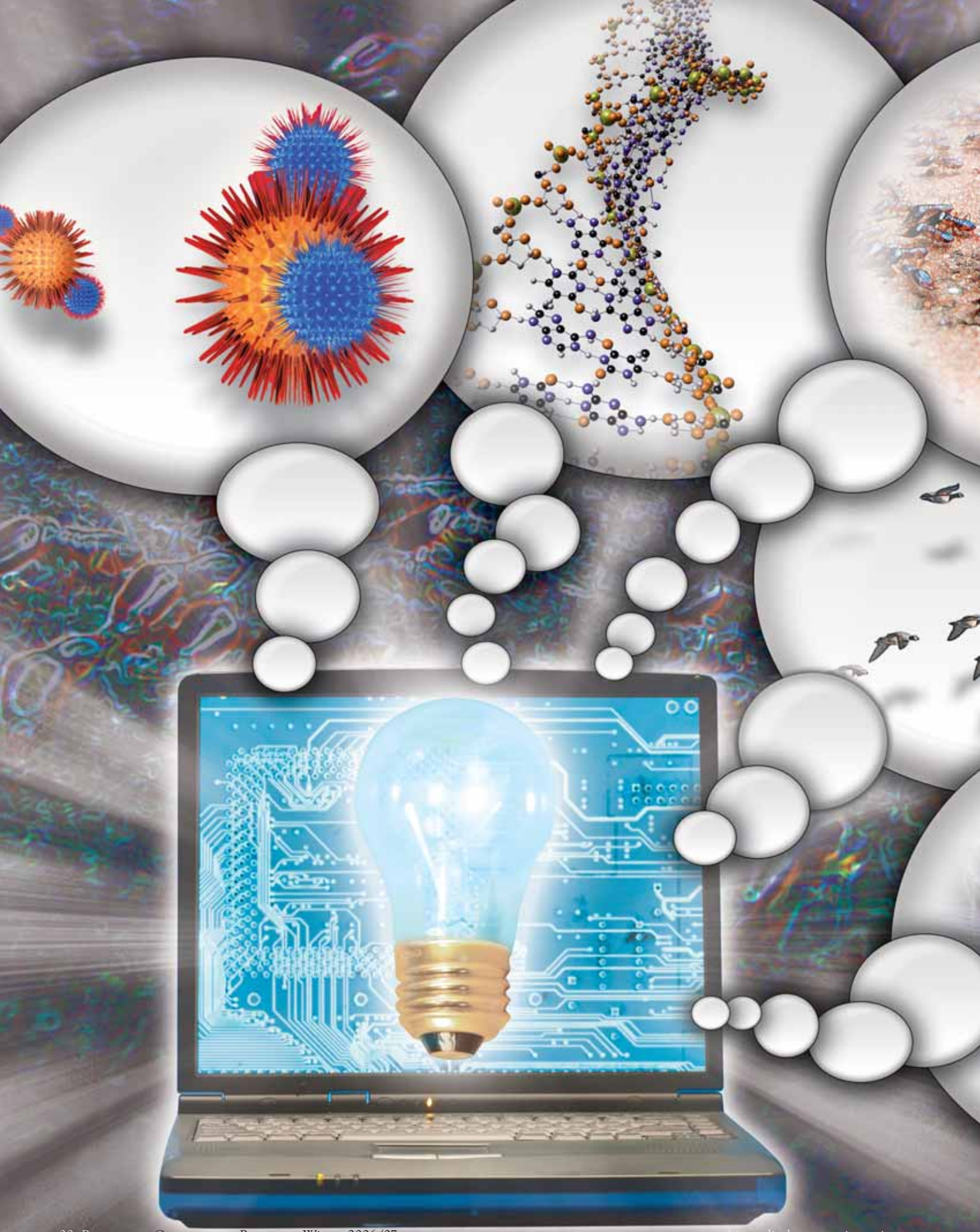


Biologically Inspired Computation

BY MEREDITH ALEXANDER KUNZ

Algorithms That Mimic Nature's Tricks

Today's computers are fast, cheap, and effective at crunching numbers—yet they still cannot equal the versatility of even the smallest of Earth's life forms when it comes to processing information. "Somehow living systems are able to do all the things that are very hard to engineer in computers," says **Melanie Mitchell, PhD**, professor of computer science at Portland State University. To name just a few of these biological feats, Mitchell cites "pattern recognition, self repair, robustness to certain kinds of noise, and adaptation to changing environments."

In hopes of designing algorithms and software to do what nature does with such ease, computational scientists have scrutinized biological systems looking for useful ideas and inspiration. They do this because biological systems stand as proof that incredibly complex information processing systems are possible. The brain, the immune system, genes, and social insects are the envy of engineers, who've yet to achieve anything close to what these biological systems do on their own. Indeed, according to **Carver Mead, PhD**, microelectronics pioneer and Caltech professor emeritus, "engineers would be foolish to ignore the lessons of a billion years of evolution."

Nature has created brains that can process data, and spot patterns and learn from them; generated new life forms that evolve to fit their environments; produced organisms capable of working in concert as colonies and swarms; and formed organized systems to fight disease and repair damage. Each of these is a jumping-off point for a type of biologically inspired computation.

In the cases highlighted here, biology serves as a metaphor, inspiring new approaches, rather than as a mechanism for computation itself (though there are some scientists at work at creating logic gates and other computational tools within biomolecules and even living cells). Biological phenomena are not copied point-by-point in some sort of computational translation—the digital imitation is far more abstract.

Mimicking nature's tricks has yielded some very creative means of digital problem solving. Some such problems are computational, such as the classic "traveling salesmen" problem (minimizing the cost of a visit to N different cities). But bio-inspired computation has also helped improve designs for circuits, racecars and robots; predict moving targets like patterns in the stock market; and come full circle—from biological inspiration back to biological applications such as modeling immune systems; helping understand the visual cortex; and identifying protein segments.

Neural Networks: If the Brain Can Do It, Why Can't Computers?

The brain is a remarkable computational tool that has fascinated computer scientists for decades. **Risto Miikkulainen, PhD**, professor of computer science at the University of Texas, Austin, names speech recognition, accurate motion, and vision as just a handful of things that the brain can accomplish that, so far, computational science still struggles to achieve. "It looks like the standard [computational] approaches that we have are not going to get us there. Therefore, we should look at how the brain is doing it," says Miikkulainen, who specializes in brain-inspired artificial neural networks.

Today's computers are based on concepts first proposed in the 1940s by Hungarian-born polymath John von Neumann, who developed early high-speed calculating machines that processed information according to a set of serial instructions. By contrast, rather than following a series of directions one by one, the human brain's 100 billion neurons handle massive amounts of information in parallel.

Mathematician Walter Pitts and psychiatrist Warren McCullough conceptualized artificial neural networks in 1943 and the field expanded in the 1950s and 1960s, when "perceptrons"—the earliest artificial neurons that could process information—were developed.

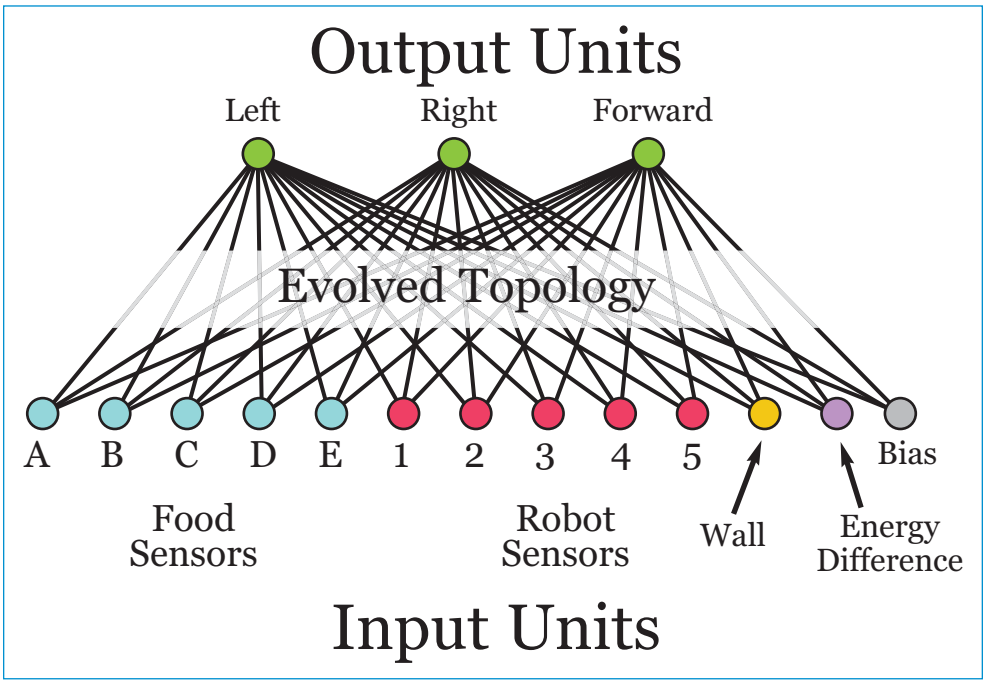
Artificial neural networks are networks of perceptrons loosely analogous to networks of biological neurons. An activation in the "input layer" flows through weighted connections to the neurons in the "hidden layer(s)," and on through similar connections to the "output layer." Each neuron calculates a weighted sum of its inputs and generates an output that is a nonlinear function of the sum. The weights are adjusted according to a learning rule local to each neuron.

"Somehow living systems are able to do all the things that are very hard to engineer in computers," says Melanie Mitchell, such as "pattern recognition, self repair, robustness to certain kinds of noise, and adaptation to changing environments."

A neural network learns by running test cases and measuring the network's solution against the one desired. Errors are sent backward (using "back-propagation") through hidden layers in the network, all the while readjusting the weights it gives to various connections. By training the network with numerous examples—for instance, of how to recognize letters written in handwriting—artificial neurons can classify never before seen inputs, even when confronted with incomplete or noisy inputs.

Miikkulainen sees four major branches of neural network research today. "The field has matured tremendously, which means it also has splintered into subfields," he says. Some neural network researchers, working directly with biologists and animal data, are interested in biological applications, such as understanding the way the visual cortex works. Then there are those focused on the mathematical underpinnings of neural networks, pursuing how nonlinear statistics can improve performance. A third area centers on high-level cognitive science, looking at topics like natural language processing—how children (and possibly computers, someday) learn to speak.

A fourth grouping of researchers—in the engineering world—have discovered a wide range of applications for neural networks. For instance, they can help recognize unanticipated patterns in the stock



In "neuroevolution," a neural network evolves by genetic algorithms as it learns. In this diagram, inputs record sensor data and other information available to a moving robot, while outputs are the robot's actions. In between the two are hidden nodes containing weighted connections. As the network learns, it adds connections and hidden nodes and varies the weights of the connections, enabling the network to produce new, more accurate outputs. In this case, the robot learns from its surroundings whether to turn left, right, or move forward. Image courtesy of Risto Miikkulainen and Ken Stanley.

market, identify faces for law enforcement, or classify credit applicants as a good bet. "It's being used as a standard toolbox for solving really hard problems in engineering," Miikkulainen reports.

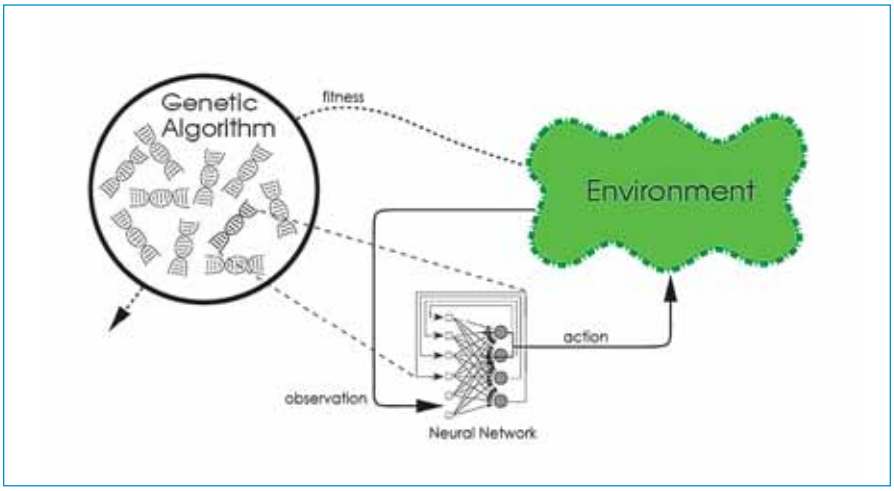
In addition to back-propagation, several other new approaches to neural network learning have emerged in recent years. One of the most promising is "neuroevolution," which uses a learning process inspired by evolution to construct the network. A large population of neural networks represents various answers to a problem; the top performers are selected and recombined to create even better networks. Miikkulainen has evolved neural networks to control a complex rocket and to warn automobiles of collision dangers. To understand how, we must delve deeper into evolutionary computation.

Evolutionary Computation: Algorithms and Programs with Answers that Adapt

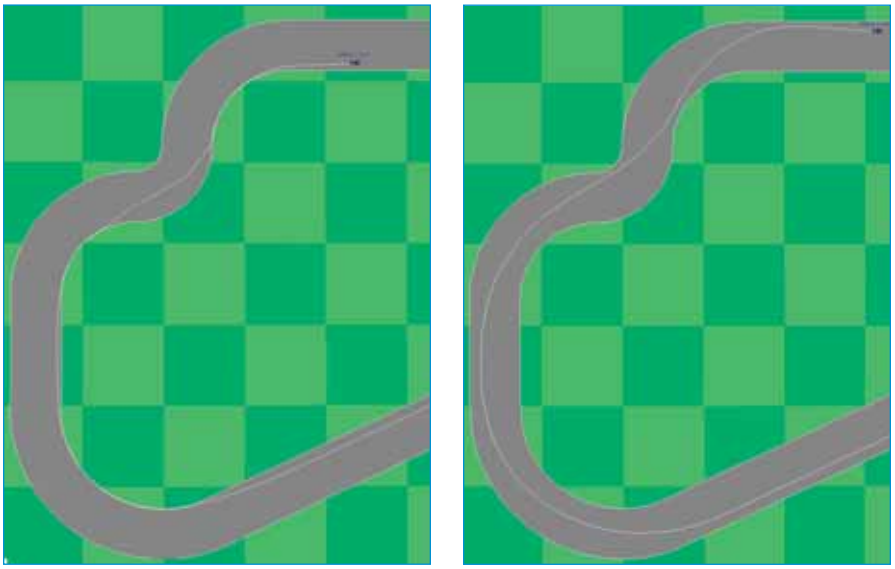
The central idea behind evolutionary computation is that any possible solution, or part of a solution, can be treated as a "chromosome," where digital bits of information are analogous to genetic material. First, programmers create a group of potential answers. These then perform a task, are evaluated using a pre-defined fitness metric, and the best are "mated" in a process that borrows from the theory of genetic change: Digital information mimics mutation, crossover, and inversion of genes to give birth to a new generation. The process repeats, and the most fit solution survives for the next generation.

Evolutionary computing is sometimes the best approach when approximate solutions can be refined by variation and testing. In computer science, the most commonly used evolutionary tool is the genetic algorithm, which was pioneered in the 1960s and 1970s by **John Holland, PhD**, professor of psychology, electrical engineering, and computer science at the University of Michigan. This computational method targets optimization and search problems. Numerous patented inventions have been designed by genetic algorithms, including several electronic circuits. "Genetic algorithms will design circuits in ways that are very different from the way an electrical engineer might design them, but they are very successful," says Mitchell. In the world of biology, genetic algorithms have helped design new proteins and identify protein segments.

Peter Bentley, PhD, a computer scientist who leads a digital biology research group at University College London, has explored innovative genetic applications. One is a racing car setup that allowed his group's virtual Formula One car to outperform its competitors in a computerized race. Running simulations, his team evaluated 68 variables in the car's design and evolved a model



Neuroevolution draws from the strengths of both neural networks and genetic algorithms. Connection weights are broken down into "chromosomes," which are represented numerically. Initially, a random population of networks is created and tested on a particular problem; in each generation, the genetic algorithm selects top performers to "mate" and create a new generation with different "chromosomes." Hundreds of networks are tried until a solution emerges. Image courtesy of Risto Miikkulainen and Tino Gomez.



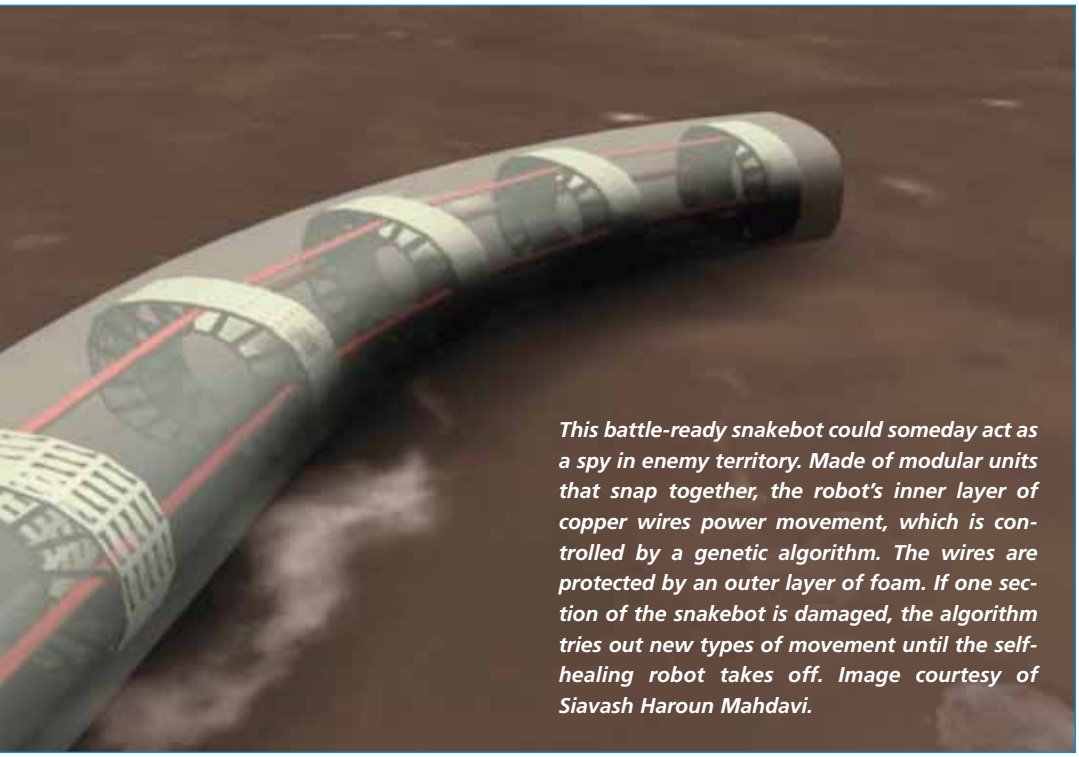
Evolved neural networks learn "intelligent driving" in the auto collision prevention work by Risto Miikkulainen and his colleagues. The first image above shows the robotic car's early attempt to drive a twisty course. The car hugs the inside of the turn, which minimizes driving distance but slows the car's speed considerably. After some 400 generations, the network has learned to steer outward when entering and exiting a turn, maintaining maximum speed while at the same time avoiding the course's walls. Images courtesy of Risto Miikkulainen, Nate Kohl and Ken Stanley.

that beat out those created by human experts. Another project is a self-repairing robot: Bentley and UCL PhD candidate **Siavash Haroun Mahdavi** created a snake-like machine for remote reconnaissance missions. "Digital chromosomes" control the snakebot's muscle wires, effectively acting as a brain to manage its movement. If the robot is damaged, genetic algorithms try different genetic combinations, mating the top performers in each generation until the snakebot begins to go forward.

In the 1980s and 1990s, the field of evolutionary computing grew with contributions from researchers such as Stanford consulting professor **John Koza, PhD**, who formulated genetic programming, an effort that allows computers to generate and run new programs "automatically." Genetic programming divides a program into its basic parts, represented by a "parse tree," where program pieces subdivide at nodes and become separate branches. The branches are exchanged for one another in a process similar to genetic mutation and combination, and a computer then runs the whole program to test its worth. Like other evolutionary computing strategies, genetic programming can help control robots, design circuits, and solve problems in molecular biology. A debate continues in the field as to the relative value of genetic programming compared with other evolutionary methods, according to Mitchell. Bentley comments that genetic programming surged in popularity for a time, only to subside again after computational scientists tested its limits.

"Genetic algorithms will design circuits in ways that are very different from the way an electrical engineer might design them, but they are very successful," says Mitchell.

Evolutionary computing has also spawned the subfield of coevolution. Two populations—one of candidate solutions, one of problems—mimic the typical biological roles of real life forms, such as a host and parasite. Hosts change to fight off parasites better, while at the same time, parasites attempt to overcome hosts' ever-evolving defenses. "The solutions you get to problems are often much more robust," comments Mitchell, who is using coevolution methods to explore nanoscale computing architectures. In her project, each host is a candidate cellular automaton that is part of an array of similar automata capable of simple processing tasks. The automata collectively perform computations without centralized control. One sample computation is to synchronize the entire array of processors from any given starting point. The parasites are inputs. When the hosts and parasites coevolve, an evolutionary arms race ensues, with better and better hosts being faced with increasingly challenging parasites. At the end of the process, the resulting hosts are arrays in which the processors can synchronize successfully and robustly in the absence of any central controller. Mitchell is investigating the usefulness of coevolutionary computing in accomplishing other tasks that require such decentralized, distributed computing architectures.



This battle-ready snakebot could someday act as a spy in enemy territory. Made of modular units that snap together, the robot's inner layer of copper wires power movement, which is controlled by a genetic algorithm. The wires are protected by an outer layer of foam. If one section of the snakebot is damaged, the algorithm tries out new types of movement until the self-healing robot takes off. Image courtesy of Siavash Haroun Mahdavi.

Ant Algorithms and Swarm Intelligence: Finding the Right Path

While the big picture of evolution intrigues some computer scientists, others take inspiration from the way ants so quickly line up for a glob of peanut butter on a kitchen counter. These insects' ability to find the shortest path to a destination piqued the interest of **Marco Dorigo, PhD**, research director of the IRIDIA lab at the Université Libre de Bruxelles. Dorigo realized that the computational implications were tremendous: Numerous optimization problems could be attacked using ant-inspired techniques. He developed the ant colony optimization metaheuristic, a computational tool that harnesses the power of ants' movement.

When ants search for food, they leave a trail of pheromones directing other ants. Dorigo's ant colony optimization employs digital ants using pheromone-like signals to tell each other which paths they are try-

ing out. Ants travel back and forth, and the ones with the shortest paths return fastest, spreading their signals to the next group of ant followers. Soon, the paths begin to converge on the most efficient trails to the desired destinations.

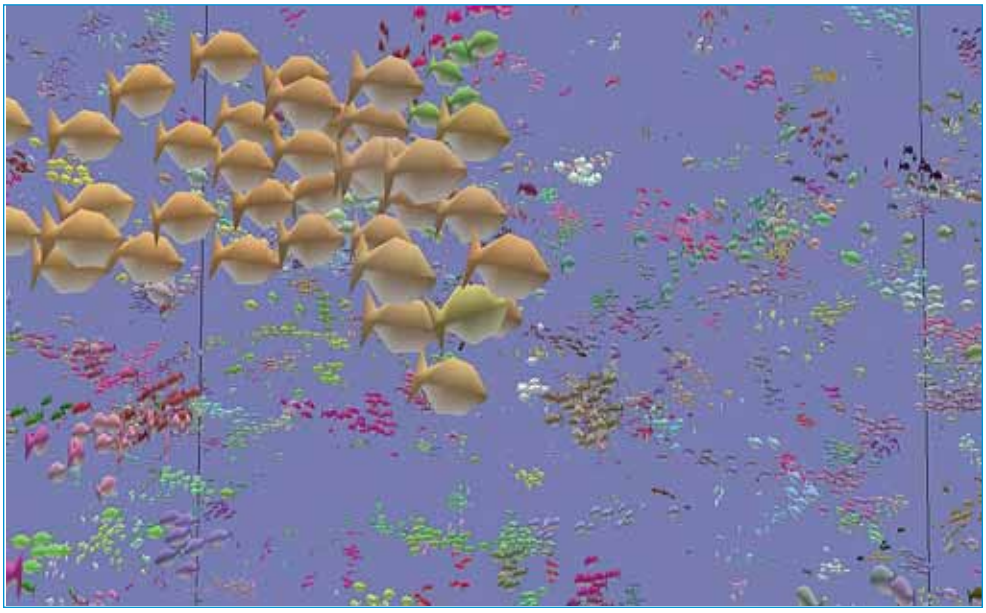
This technique is perfect for solving extremely complex (NP-hard) combinatorial optimization problems, of which the classic "traveling salesman" problem is the best known example. Commercial applications—such as how to optimize trucking routes for a distribution company—are starting to appear. "We use the artificial ants moving on particular mathematical representation of the problem, and they find very good solutions," says Dorigo. At least two companies, the Swiss AntOptima and the American Icosystem, base business applications on ant colony optimization. Their products target task scheduling, vehicle routing and telecommunications network routing. Newer ant algorithms can also assist in data mining, an application that stems from ants' ability to locate and cluster small objects, Dorigo explains.

But computer science's interest in insects does not stop there. Other "swarm intelligence" work deals with the movement of a swarm of insects or flock of birds, defining very simple rules that determine how small creatures can move successfully in formation. Computer graphics expert **Craig Reynolds** defined such rules for his "Boids" simulation of the behavior of a bird flock. When each member follows the rules of separation (steer to avoid crowding local flockmates), alignment (steer towards the average heading of local flockmates), and cohesion (steer to move toward the average position of local flockmates), the artificial birds flock together across an obstacle-filled space. Graphics artists have used this concept to create real-looking armies and herds in movies such as *The Lord of the Rings*.

Taking this approach further, Dorigo has developed a group of "Swarm-bots"—simple robots that work together using the rules of swarm intelligence to get around and accomplish tasks—such as moving a heavy object—by working together. Such inventions could eventually aid the military in war zones or map areas inaccessible or dangerous to human beings, like disaster areas, the ocean floor, or even another planet.

Swarm intelligence may even

Ants' ability to find the shortest path between two destinations has inspired computer scientists to find new ways of solving optimization problems. Ant-like thinking has been captured in ant colony optimization algorithms pioneered by Marco Dorigo.



Craig Reynolds used the principles of swarm intelligence to simulate these schooling fish. In this demo—part of Reynold's PSCrowd graphics project—10,000 autonomous fish interact at 60 frames per second on a Sony PLAYSTATION3. Image courtesy of Craig Reynolds.

Marco Dorigo's "Swarm-bots" are small, simple robots programmed to cooperate according to the rules of swarm intelligence. Here, the bots pull together to negotiate a set of stairs—a task that one could not effectively do on its own because of its size, but that it can accomplish when linked to others in a "swarm." Image courtesy of Marco Dorigo

record your next CD. Bentley and colleague **Tim Blackwell, PhD**, of University College London wrote a program that mimics insects swarming around musical notes, improvising a jazz tune with a real musician. Swarm Music software divides music into three dimensions, pitch, loudness and note duration, and then releases a swarm of digital insect-like "particles" around the 3D space to seek out the musician's notes. According to the inventors, the resulting riffs sound human in origin.

Someday, swarms of nanobots—the tiny robots that have served as a science fiction villain in books like Michael Crichton's *Prey*—may become a reality. In the last two years, engineers at NASA have tested a robot prototype for nanobots that would form "autonomous nanotechnology swarms" (ANTS), moving of their own accord and repairing themselves as needed. Scientists hope to miniaturize the bots for use in space exploration.

Artificial Immune Systems to Protect PCs and Personal Data

Computers are notoriously prone to viruses—programs that invade and incapacitate them. Every year in the United States alone, viruses, hacking, spyware and other computer crimes cost businesses $67 billion, according to a 2006 FBI survey. The FBI also reported that some 52 percent of computer users experienced intrusion into their computer systems.

But what if computer scientists dealing with such attacks could mimic the way humans fight off disease? That is the inspiration behind artificial immune systems, a field that took shape in the mid-1980s and has grown considerably since.

Computer science professor **Stephanie Forrest, PhD**, of the University of New Mexico, helped launch the field. "The thing I seized on was how the immune system performs computation in interesting ways," she says. She draws from the basic underpinnings of immunology to create systems that can identify an intrusion.

Although the immune system is not usually thought of in the context of computational abilities, it is truly remarkable in this respect. "The immune system is a very sophisticated pattern recognizer. It notices unusual peptides in the body," Forrest says. Using computational tricks like parallel processing, combinatorics and memory, it responds to foreign antigens by seeking to eliminate them.

Mimicking biology, Forrest has developed ways for computers to differentiate between their own "cells" (self) and harmful intruders (non-self) by tagging a computer's program content with "digital peptides." The peptides consist of a call pattern—lines of code sent between software and the computer's operating system—that allows the computer to recognize its own materials. A computer equipped with Forrest's defenses houses databases of normal signals for each program it contains. Lacking the all-important recognizable call patterns, intruding material is identifiable by lymphocyte-like detectors, which then isolate it. After discovering an anomaly, Forrest's security system slows the computer down, giving programmers a chance to find and fix problems before the whole computer or network is overwhelmed. A San Mateo, California-based company, Sana Security, offers a commercial product based on the fundamentals of this research.

A host of other applications follow the same line of thinking. For instance, a pattern-recognition approach can detect and isolate unusual loan applications in a pool of standard-looking paperwork, Forrest says.

A former Forrest student, Yale postdoc **Fernando Esponda, PhD**, is drawing upon immunology to develop "negative databases," The idea stems from the immune system's ability to respond to pathogens precisely because they lack known markers. Immune cells keep a negative database: Non-self is defined by what is not in the "self" database. Esponda asked whether other data sets could be searched "negatively." For instance, he wondered, "Can I, instead of storing all of the entries in my personal address book, efficiently store a representation of all of the entries that are not in my address book? The answer is yes," Esponda says, if all the entries are of finite, given length. This unexpected application could have uses in fields where information security is paramount.

Forrest calls this a promising approach for privately collecting information. "For example, people might be more willing to tell you what sexually disease they *don't* have," she says, which could give epidemiologists a fairly good sense of how many people do indeed have a specific disease. Another potential application could allow two banks to communicate

The human immune system inspired NASA scientists to create software that helps find faults in complex machines. Their algorithm, part of the Multi-level Immune Learning Detection (MILD) software tool under development at NASA Ames Research Center in California's Silicon Valley, uses sensor data to track patterns of system faults. Using the concept that an anomaly in these patterns signals a problem—the same notion that drives the human immune response—scientists hope to spot trouble in spacecraft, aircraft, and other large machines early. Scientists tested the MILD software in this aircraft flight simulator at NASA Ames. The software, still under development, was created by NASA scientist Kalmanje Krishnakumar and MILD co-investigator Dipankar Dasgupta of the University of Memphis. Image courtesy of NASA Ames Research Center.

about their transactions with each other without giving away the private information (names, dollar amounts or other data) contained in their records.

While negative databases are far from the nitty-gritty of immunology itself, they are an excellent example of how a biological system can provide a wholly different philosophical approach—one that specialists in the field might never have uncovered without a little help from the natural world.

What's Next for Biologically Inspired Computation?

Biologically inspired computation shows that computers need not be rigid logic boxes. Instead, software learns. Computer programs evolve. Artificial insects find their way—and guide humans. Machines heal.

The overarching challenge that remains is to create computational techniques so flexible that they can respond to the complicated real-world situations that nature holds in store.

In a sense, these advances are not surprising: Computer programming has steadily gained complexity ever since the construction of the first von Neumann machine. And as people have learned more about biology, it makes perfect sense to apply this knowledge to computation. Biologically-based computational strategies are an effort to catch up with those evolved by nature over eons.

Computational scientists will undoubtedly continue to seek biological inspiration for improving their work. The overarching challenge that remains is to create computational techniques so flexible that they can respond to the complicated real-world situations that nature holds in store. The motivation is clear: From global warming to drug-resistant bacteria, from the hardship of modern warfare to rising cancer rates, humans have a lot of problems to solve, and today, our tool of choice is the computer. □