8

Biological Inspiration for Computing

Chapters 4-7 address ways in which computer science and engineering can assist in the pursuit of a broadly defined research agenda in biology. This chapter suggests how insights from the biological sciences may have a positive impact on certain research areas in computing, although the impact of this reversed direction is at present much more speculative.[1]

8.1 THE IMPACT OF BIOLOGY ON COMPUTING

8.1.1 Biology and Computing: Promise and Skepticism

Today's computer systems are highly complex and often fragile. Although they provide high degrees of functionality to their users, many of today's systems are also subject to catastrophic failure, difficult to maintain, and full of vulnerabilities to outside attack. An important goal of computing is to be able to build systems that can function with high degrees of autonomy, robustly handle data with large amounts of noise, configure themselves automatically into networks (and reconfigure themselves when parts are damaged or destroyed), rapidly process large amounts of data in a massively parallel fashion, learn from their environment with minimal human intervention, and "evolve" to become better adapted to what they are supposed to do.

There is little doubt that such computer systems with these properties would be highly desirable. Although the development of such systems is an active area of computer science research today (indeed, the Internet itself is an example of a system that is capable of operating without centralized authority and reconfiguring itself when parts are damaged), computer science researchers are working to develop new such systems, and the prospect of looking outside the existing computer science toolbox for new types of hardware, software, algorithms, or something entirely different (and unknown) is increasingly attractive.

One possible area of research focuses on a set of techniques inspired by the biological sciences, because biological organisms often exhibit properties that would be desirable in computer systems.

[1]A popularized account of biological inspiration for computing is N. Forbes, *Imitation of Life: How Biology Is Inspiring Computing,* MIT Press, Cambridge, MA, 2004.

They function with high degrees of autonomy. Some biological entities—such as neurons in a brain—can configure themselves automatically into networks (and reconfigure themselves to some degree when parts are damaged or destroyed). Sensory systems rapidly pick out salient features buried in large amounts of data. Many animals learn from their environment and become better adapted to what they are supposed to do. All biological organisms have mechanisms for self-repair, and all multicellular organisms grow from an initial state that is much less phenotypically complex than their final states.

Carver Mead once noted that "engineers would be foolish to ignore the lessons of a billion years of evolution." The solutions that nature has evolved to difficult engineering problems are, in many cases, far beyond present-day engineering capability. For example, the human brain is not fast enough to process all of the raw sensory data detected by the optic or auditory nerves into meaningful information. To reduce processing load, the brain uses a strategy we know as "attention" that focuses on certain parts of the available information and discards other parts. Such a strategy might well be useful for an artificial machine processing a large visual field. Studies of the way in which humans limit their attention has led to computational models of the strategy of shifting attention. Such models of biological systems are worth studying even if they appear intuitively less capable than computation, if only for the fact that no machine systems exist that can function as autonomously as a housefly or an ant.

On the other hand, biological organisms operate within a set of constraints that may limit their suitability as sources of inspiration for computing. Perhaps the most important constraint is the fact that biological organisms emerge from natural selection and the evolutionary process. Because selection pressures are multidimensional, biological systems must be multifunctional. For example, a biological system may be able to move, but it has also evolved to be able to feed itself, to reproduce, and to defend itself. The list of desirable functions in a biological system is long, and successfully mimicking biology for one particular function requires the ability to separate nonrelevant parts of the system that do not contribute to the desired function. Furthermore, because biological systems are multifunctional, they cannot be optimized for any one function. That is, their design always represents a compromise between competing goals. Organisms must be adequately (rather than optimally) adapted to their environments. (The notion of "optimal design" is also somewhat problematic in the context of stochastic real-world environments.) Also, optimal adaptation to any one environment is likely to disadvantage an organism in a significantly different environment, and so adequately adapted organisms tend to be more robust across a range of environments.

The evolutionary process constrains biological solutions as well. For example, biological systems inevitably include vestiges of genetic products and organs that are irrelevant to the organism in its current existence. Thus, biological adaptation to a given environment depends not only on the circumstances of the environment but also on its entire evolutionary history—a fact that may well obscure the fundamental mechanisms and principles in play that are relevant to the specific environment of interest. (This point is a specific instantiation of a more general phenomenon, which is that our understanding of biological phenomena will often be inadequate to provide detailed guidance in engineering a computational device or artifact.)

A corollary notion is that nature may evolve different biological mechanisms to solve a given problem. All of these mechanisms may enable the organism to survive and even to prosper in its environment, but it is far from clear how well these mechanisms work relative to one another.[2] Thus, which one of many biological instantiations is the most appropriate model to mimic remains an important question.

Finally, there are only a few examples of successful biologically inspired computing innovations. Thus, the jury is still out on the ultimate value of biology for computing. Rather than biology being helpful across the board to all of computing, the committee believes that biology's primary relevance (at least in the short term) is likely to be to specific problem areas within computing that are poorly

[2]For example, fish and squid use different mechanisms to propel themselves through the water. Which mechanism is better under what circumstances and for what engineered artifacts is a question for research to answer.

understood, or for which the relevant underlying technologies are too complex or unwieldy, and in providing approaches that will address parts of a solution (as described in Section 8.1.2). Nevertheless, the potential benefits that biology might offer to certain problem areas in computing are large, and it is worth exploring different approaches to exploit these benefits; this is the focus of Sections 8.2 to 8.4.

8.1.2 The Meaning of Biological Inspiration

What does it mean for something to be biologically inspired? It is helpful to consider several possible interpretations. One interpretation is that significant progress in computing can occur *only* through the application of principles derived from the study of biology. This interpretation, offered largely as a strawman, is absurd—there are many ways in which computing can progress without the application of biologically derived principles.

A second, somewhat less grandiose and more reasonable interpretation is that significant progress in computing *can* occur through the application of principles derived from the study of biology. That is, a biological system may operate according to principles that have applicability to nonbiological computing problems. By studying the biological system, one may be able to derive or understand the relevant principles and use them to help solve a nonbiological problem. It is this interpretation—that biology is relevant to computing only when principles emerge directly from a study of biological phenomena—that underlies many claims of biological relevance or irrelevance to computing.

A third interpretation is that certain aspects of biology are analogous to aspects of computing, which means that insights from biology are relevant to aspects of computing. This is the case, for instance, when a set of principles or paradigms turns out to have strong applicability both to a biological system or systems and to interesting problems in computing. These principles or paradigms may have had their intellectual origin in the study of a biological or a nonbiological system.

When their origin is in a biological system, this interpretation reduces to the second interpretation above. What makes the case of an origin in a nonbiological system interesting is that the principles in question may be more manifestly obvious in a biological context than in a nonbiological context. That is, the principles and their application may most easily be seen and appreciated in a biological context, even if they did not initially originate in a biological context. Moreover, the biological context may also provide a source of language, concepts, and metaphors that are useful in talking about a nonbiological problem or phenomenon.

For this report, the term "inspiration" will be used in its broadest sense, that is, the third interpretation above, but there are three other points to keep in mind:

- Biological inspiration does not mean that the weaknesses of biology must be adopted along with the strengths. In some cases, it may be possible to overcome problems found in the actual biological system when the principles underlying them are implemented in engineered artifacts.
- As noted in Chapter 1, even when biology cannot provide insight into potential computing solutions, the drive to solve biological problems can still inspire interesting, relevant, and intellectually challenging research in computing—so biology can serve as a useful and challenging problem domain for computing.[3]

[3]For example, IBM used the problem of protein folding to motivate the development of the BlueGene/L supercomputer. Specifically, the problem was formulated in terms of obtaining a microscopic view of the thermodynamics and kinetics of the dynamic protein-folding process over longer time scales than have previously been possible. Because this project involved both computer architecture and the exploration of algorithmic alternatives, the applications architecture was structured in such a way that subject experts in molecular simulation could work on their applications without having to deal with the complexity of the parallel communications environment required by the underlying machine architecture (see BlueGene/L Team, "An Overview

- Incomplete (and sometimes even incorrect) biological understandings help to inspire different and useful approaches to computing problems. Important and valuable insights into possible ways to solve a current problem have been derived from biological models that were incomplete (as in the case of evolutionary programming) or even inaccurate (as in the case of immunologically based computer security).

On the other hand, it must be understood that the use of a biological metaphor to inspire new approaches to computing does not necessarily imply that the biological side is well understood, whether or not the metaphor leads to progress in computing. That is, even if a biological metaphor is applicable and relevant to a computing problem, this does not mean that the corresponding biological phenomena can necessarily be understood in computational terms.

For example, although researchers use the term "genetic algorithms" to describe a class of algorithms using operators that have a similar flavor to evolutionary genetic operators such as mutation or recombination to search a solution space stochastically, the definition and implementation of these genetic operators does not imply a fundamental understanding of biological evolutionary processes. Similarly, although the field of "artificial neural networks" is an information-processing paradigm inspired by the parallel processing capabilities and structure of nerve tissue, and it attempts to mimic learning in biology by learning to adjust "synaptic" connections between artificial processing elements, the extent to which an artificial neural network reflects real neural systems may be tenuous.

8.1.3 Multiple Roles: Biology for Computing Insight

Biological inspiration can play many different roles in computing, and confusion about this multiplicity of meanings accounts for a wide spectrum of belief about the value of biology for developing better computer systems and improved performance of computational tasks. One point of view is that only a detailed "ground-up" understanding of a biological system can result in such advances, and because such understanding is available for only a very small number of biological systems (and "very small" is arguably zero), the potential relevance of biology for computing is small, at least in the near term.

A more expansive view of biology's value for computing acknowledges that detailed understanding is the key for a maximal application of biology to computing, but also holds that biological metaphors, analogies, examples, and phenomenological insights may suggest new and interesting ways of thinking about computational problems that might not have been imagined without the involvement of biology.[4] From this perspective, what matters is performance of a task rather than simulation of what a biological system actually does, though one would not necessarily expect initial performance models

of the BlueGene/L Supercomputer," presented at Supercomputing Conference, November 2002, available at http://sc-2002.org/paperpdfs/pap.pap207.pdf). Other obvious problems inspired by biology include computer vision and artificial intelligence. It is also interesting to note this historical precedent of biological problems being the domain in which major suites of statistical tools were developed. For instance, Galton invented regression analysis (correlation tests) to study the relation of phenotypes between parents and progeny (see F. Galton, *Natural Inheritance*, 5th Edition, Macmillan and Company, New York, 1894). Pearson invented the chi-square and other discrete tests to study the distribution of different morphs in natural populations (see K. Pearson, "Mathematical Contributions to the Theory of Evolution, VIII. On the Inheritance of Characters Not Capable of Exact Quantitative Measurement," *Philosophical Transactions of the Royal Society of London*, *Series A* 195:79-150, 1900). R.A. Fisher invented analysis of variance to study the partitioning of different effects in inheritance (see R. Fisher, "The Correlation Between Relatives on the Supposition of Mendelian Inheritance," *Transactions of the Royal Society of Edinburgh* 52:399-433, 1918).

[4]An analogy might be drawn to the history of superconducting materials. A mix of quantum principles, phenomenology, and trained experience has led to superconducting materials with ever-higher transition temperatures. (Indeed, the discovery of superconducting materials preceded quantum mechanics by more than a decade.)

based on biological systems to function more effectively than models constructed using more traditional techniques.

One of biology's most important roles is that it can serve as an existence proof of performance—that some desirable behavior is possible. The reasoning is that if a biological system can do something interesting, why can't an artificial system to the same thing? Birds fly, so why shouldn't people or constructed artifacts be able to fly? Many biological behaviors and functions would be desirable in a computing context, and biological systems that exhibit such behavior demonstrate that this behavior is possible.[5]

Existence proofs are important in engineering. For example, in the view of many nuclear scientists associated with the Manhattan Project, the information that was most critical to the Soviet development effort was not a secret gained through espionage, but rather the fact that a nuclear explosion was possible at all—and that fact was reported in every major newspaper in the world.[6] In other words, it is one thing to work toward a goal that may well be impossible to achieve and an entirely different psychological matter to work toward a goal whose achievement is known—with certainty—to be possible.

An example of using a biological metaphor for understanding some dimension of computing relates to computer security. From many centuries of observation, it is well known that an ecology based on a monoculture is highly vulnerable to threats that are introduced from the outside. With this insight in mind, many expert observers have used the term "monoculture" to describe the present-day security environment for desktop computers in which one vendor dominates the operating system market. This report does not take a position on whether such a characterization is necessarily accurate,[7] but the point is that the metaphor, used in this manner, can determine the terms of discussion and thus provide a useful way of looking at the issue.

Despite its conceptual value, an existence proof does not speak directly to how to build the artifact so that it does the same thing. That is, existence proofs do not necessarily provide insight about construction or creation. Diversity as a strategy for survival does not necessarily indicate how much or what kinds of diversity would be helpful in any given instance. Similarly, aerodynamics is a body of theory that explains the flight of birds, and also enables human beings to design airplanes, but a study of birds did not lead to the airplane. For construction or creations, a deeper understanding of biology is required. Knowing what kind of deeper understanding is possible potentially leads to at least three additional roles for biology:

- *Biology as source of principles*. Nature builds systems out of the same atoms that are available to human engineers. If a biological system can demonstrate a particular functionality, it is because that system is built according to principles that enable such functionality. The hope is that upon close examination, the physical, mathematical, and information-processing principles underlying the interesting biological functionality can be applied through human engineering to realize a better artificial system. Note also that in some cases, the actual principles underlying some biological functionality may be difficult to discern. However, plausibility counts for a great deal here, and biology may well provide inspiration for engineered artifacts if human beings propose a set of plausible principles that govern the behavior of interest in an actual organism, even if those principles, as articulated, turn out not to have a biological instantiation in that organism. (Note that in this domain the division between "applying

[5]An accessible and more extended discussion of these ideas can be found in J. Benyus, *Biomimicry: Innovation Inspired by Nature*, William Morrow, New York, 1997.

[6]D. Holloway, *Stalin and the Bomb: The Soviet Union and Atomic Energy, 1939-1956*, Yale University Press, New Haven, 1994.

[7]For example, it may be that even though the number of operating system platforms is small compared to the number of desktop computers in use, different computer configurations and different operational practices might introduce sufficient diversity to mitigate any system-wide instabilities. Furthermore, replication has many other advantages in the computer context, such as easier interoperability.

biological principles to information processing" and "understanding biological information processing" is least meaningful.)

• *Biology as implementer of mechanism*. Nature also implements mechanisms to effect certain functions. For example, a biological organism may implement an algorithm that could be the basis of a solution to a computing problem of interest to people. Or, it may implement an architecture or a way to organize and design the structural and dynamic relationships between elements in a complex system, knowledge of which might greatly improve the design of an engineered artifact. In this category are the neural network architecture as inspired by the activation model of dendrites and axons in the brain, evolutionary computation as driven by genomic changes and selection pressures, and the use of electroactive polymers as actuator mechanisms for robots, inspired by the operation of animal muscles (rather than, for example, gears). (Note that implementations of biological mechanisms tend to be easier to identify and extract for later use when they involve physical observables—and so mechanisms underlying sensors and locomotion have had some nontrivial successes in their application to engineered artifacts.)

• *Biology as physical substrate for computing.* Computation can be regarded as an abstract or a physically instantiated form. In the abstract, it is divorced from anything tangible. But all real-world computation requires hardware—a device of some kind, whether artificial or biological—and given that biological organisms are functional physical devices, it makes sense to consider how engineered artifacts might have biological components. For example, biology may provide parts that can be integrated into engineered devices. Thus, a sensitive chemical detection system might use a silk moth as the sensor for chemicals in the air and thus instrument the moth to appropriate readouts. Or a small animal might be used as the locomotive platform for carrying a useful payload (e.g., a camera), and its movements might be teleoperated through electrodes implanted in the animal by a human being viewing the images sent back by a camera.

These three different roles are closely connected to the level(s) of abstraction appropriate for thinking about biological systems. For some systems and phenomena of interest, a very "bottom-up" perspective is warranted. In the same way that one needs to know how to use transistors to build a logic gate for a silicon-based computer, one needs to know how neurons in the brain encode information in order to understand how a neural implant or prosthetic device might be constructed. For other systems and phenomena, architecture provides the appropriate level of abstraction. In this case, understanding how parts of a system are interconnected, the nature of the information that is passed between them, and the responses of those parts to such information flows may be sufficient.

Another way of viewing these three roles is to focus on the differences between computational content, computational representation, and computational hardware. Consider, for example, a catenary curve—the shape that a cable suspended at both ends takes when subjected to gravity.

• The computational content is specified by a differential equation and the appropriate boundary conditions. Although the solution is not directly apparent from the differential equation, the differential equation implies a specific curve that represents the answer.

• The computational representation refers to how the computation is actually represented—in digital form (as bits in a computer), in analog form (as voltages in an analog computer), in neural form (as how a calculus student would solve the problem), or in physical form (as the string or cable being represented).

• The computational hardware refers to the physical device used to solve the equation—the digital computer, the analog computer, the human being, or the cable itself.

These three categories correspond roughly and loosely to the three categories described above: content as source of principles, representation as implementer of mechanism, and hardware as physical substrate. The remaining sections of this chapter describe some biological inspirations for work in computing.

8.2 EXAMPLES OF BIOLOGY AS A SOURCE OF PRINCIPLES FOR COMPUTING

8.2.1 Swarm Intelligence and Particle Swarm Optimization

Swarm intelligence is a property of systems of nonintelligent, independently acting robots that exhibit collectively intelligent behavior in an environment that the robots do sense and can alter.[8] One form of swarm intelligence is particle swarm optimization, based on the flocking of birds.[9]

The canonical example of flocking behavior is a flight of birds wheeling through the sky, or a school of fish darting through a coral reef. Somehow, myriad not-very-bright individuals manage to move, turn, and respond to their surroundings as if they were as a single, fluid organism. Moreover, they seem to do so collectively, without a leader: biologists armed with high-speed video cameras have shown that the natural assumption—that each flock or school has a single, dominant individual that always initiates each turn just a fraction of a second before the others follow—is simply not true.

The first known explanation of the leaderless, collective quality of flocking or schooling behavior emerged in 1986. This explanation used swarms of simulated creatures—"boids"—that could form surprisingly realistic flocks if each one simply sought to maintain an optimum distance from its neighbors. The steering rules of the so-called Reynolds simulation were simple:[10]

- *Separation:* steer to avoid crowding local flock mates.
- *Alignment:* steer toward the average heading of local flock mates.
- *Cohesion:* steer toward the average position of local flock mates.

These rules were entirely local, referring only to what an individual boid could see and do in its immediate vicinity;[11] none of them said, "Form a flock." Yet the flocks formed every time, regardless of the starting positions of the boids. These flocks were able to fly around obstacles in a very fluid and natural manner. Sometimes the flock would even break into subflocks that flowed around both sides of an obstacle, rejoining on the other side as if the boids had planned it all along. In one run, a boid accidentally hit a pole, fluttered around for a moment, and then darted forward to rejoin the flock as it moved on.

Today, the Reynolds simulation is regarded as one of the best and most evocative demonstrations of *emergent behavior*, in which complex global behavior arises from the interaction of simple local rules. The approach embodied in the simple-rule/complex-behavior approach has become a widely used technique in computer animation—which was Reynolds' primary interest in the first place.[12]

[8]T. White, "Swarm Intelligence: A Gentle Introduction with Applications," PowerPoint presentation, available at http://www.sce.carleton.ca/netmanage/tony/swarm-presentation/tsld001.htm.

[9]Bird flocks are an example of complex, adaptive systems. Among the many other examples that scientists have studied are the world economy, brains, rain forests, traffic jams, corporations, and the prehistoric Anasazi civilization of the Four Corners area. Complex adaptive systems are similar in structure and behavior even if they differ in their superficial manifestations. For example, complex adaptive systems are massively parallel and involve many quasi-independent "agents" interacting at once. (An agent might be a single firm in an economy, a single driver on a crowded freeway, and so on.) Each of them is adaptive, meaning that the agents that constitute them are constantly responding and adapting to each other. And each of them is decentralized, meaning that no one agent is in charge. Instead, a complex system's overall behavior tends to emerge spontaneously from myriad low-level interactions.

[10]C.W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics* 21(4):25-34, 1987, available at http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/ and http://www.red3d.com/cwr/papers/1987/SIGGRAPH87.pdf. An updated discussion, with many pictures and references to modern applications, can be found in C.W. Reynolds, "Boids: Background and Update," 2001, available at http://www.red3d.com/cwr/boids/.

[11]More precisely, each boid had global information about the physical layout of its environment, including any obstacles, but it had no information about its flock mates, except for those that happened to come within a certain distance that defined its local neighborhood.

[12]The first Hollywood film to use a version of Reynolds' boids software was Tim Burton's *Batman Returns* (1992), which featured swarms of animated bats and flocks of animated penguins. Since then it has been used in films such as *The Lion King* (1994) and many others (see http://www.red3d.com/cwr/boids/).

A second simulation of flocking behavior, developed in 1990, employed the Reynolds' rules (though they were independently developed) and also incorporated the influence of "dynamic forces" on the behavior of the simulated creatures.[13] These dynamic forces would allow the creatures to be attracted toward a convenient roosting point, say, or a particularly rich cornfield. As a result, the flock would turn and head in the direction of a cornfield as soon as it was placed into view, with various subgroups swinging out and in again until finally the whole group had landed right on target.

These two models are direct ancestors of the particle swarm optimization (PSO) algorithm, first published in 1995.[14] The algorithm substitutes a mathematical function for the original roosts and cornfields, and employs a conceptual swarm of bird-like particles that swoop down on the function's maximum value, even when the function has many local maxima that might confound more standard optimization algorithms.

The essential innovation of the PSO algorithm is to scatter particles at random locations throughout a multidimensional phase space that represents all the arguments to the function to be maximized. Then the algorithm sets the particles in motion. Each particle evaluates the function as it flies through phase space and keeps trying to turn back toward the best value that it has found so far. However, it is attracted even more toward the best value that any of its neighboring particles have found. So it inexorably begins to move in that direction—albeit with a little built-in randomness that allows it to explore other values of the function along the way. The upshot is that the particles quickly form a flock that flows toward a point that is one of the highest function values available, if not *the* highest.

The PSO algorithm is appealing for both its simplicity—the key steps can be written in just a few lines of computer code—and its effectiveness. In the original publication of the PSO algorithm, the algorithm was applied to a variety of neural network problems, and it was found to be a very efficient way to choose the optimum set of connection weights for the network.[15] Since then, the basic technique has been refined and extended to systems that have discrete variables, say, or that change with time. It also has been applied to a wide variety of engineering problems,[16] such as the automatic adjustment of power systems.[17]

The PSO algorithm is biologically inspired in the sense that it is a plausible account of bird flocking behavior. However, it is not known whether birds, in fact, use the PSO algorithm to fly in formation.

Swarm algorithms have the virtues of simplicity and robustness, not to mention an ability to function without the need for centralized control. For this reason, they may find their most important applications in, say, self-healing and self-organizing communications networks or in electrical power networks that could protect themselves from line faults and reroute current around a broken link "on the fly."[18]

On the other hand, simple rules are not automatically good. Witness army ants, which are such obsessive self-organizers that the members of an isolated group will often form a "circular mill," follow-

[13]F.H. Heppner and U. Grenander, "A Stochastic Nonlinear Model for Coordinated Bird Flocks," *The Ubiquity of Chaos*, S. Krasner, ed., AAAS Publications, Washington, DC, 1990.

[14]J. Kennedy and R.C. Eberhart, "Particle Swarm Optimization," pp. 1942-1948 in *Proceedings of the IEEE International Conference on Neural Networks*, IEEE Service Center, Piscataway, NJ, 1995; R. Eberhart, Y. Shi, and J. Kennedy, *Swarm Intelligence*, Morgan Kaufman, San Francisco, CA, 2001.

[15]See Section 8.3.3.2 for further discussion.

[16]A good sense of current activity in the field can be gleaned from the programs and talks at the 2003 IEEE Swarm Intelligence Symposium, April 24-26, 2003, available at http://www.computelligence.org/sis/index.html. Extensive references to PSO can be found at "Welcome to Particle Swarm Central," 2003, available at http://www.particleswarm.info. This site also contains a number of links to online tutorials and downloadable PSO code.

[17]K.Y. Lee and M.A. El-Sharkawi, eds., *Modern Heuristic Optimization Techniques with Applications to Power Systems*, John Wiley and IEEE Press, New York, March 2003.

[18]E. Bonabeau, "Swarm Intelligence," presented at the O'Reilly Emerging Technology Conference, April 22-25, 2005, Santa Clara, CA. Powerpoint presentation available at http://conferences.oreillynet.com/presentations/et2003/Bonabeau_eric.ppt.

ing one another around and around and around until they die from starvation.[19] Such blind-leading-the-blind behaviors are an ever-present possibility in swarm intelligence; the trick is to find simple rules that minimize the chances of that happening.

A closely related challenge is to find ways of designing emergent behavior, so that the swarm will produce predictable and desirable results. Today, swarm algorithms are based on the loose and imprecise specification of a relatively small number of parameters—but it is almost certainly true that engineered artifacts that exhibit complex designed behavior will require the tight specification of many parameters.

This point is perhaps most obvious in the cooperative construction problem, where the rule sets that produce interesting, complex structures are actually very rare; most self-organized structures look more like random blobs.[20] The same problem is common to all collective behaviors; finding the right rules is still largely a matter of trial and error—not least because it is in the very nature of emergence for a simple-seeming change in the rules to produce a huge change in the outcome. Thus, in their efforts to find the right rules, researchers may well seek to develop procedures that will find in the right rules rather than trying to find them directly themselves. This point is discussed further in Section 8.3.1.

8.2.2 Robotics 1: The Subsumption Architecture

One approach to robotic design is based on the notion that complex and highly capable systems are inherently expensive, and hence fewer can be built. Instead, this approach asserts the superiority of using large numbers of individually smaller, less capable, and inexpensive systems.[21] In 1989, Brooks and Flynn suggested that "gnat robots" might be fabricated using silicon micromachining to fabricate freely movable structures onto silicon wafers. Such an approach potentially allows sensors, actuators, and electronics to be embedded on the same silicon substrate. This arrangement is the basis for Brooks' subsumption architecture, in which low-level functionality can be used as building blocks for higher-level functionality.

Robots fabricated in this manner could be produced by the thousands, just as integrated circuits are produced today—and thus become an inexpensive, disposable system that does its work and need not be retrieved. For applications such as exploration in hostile environments, the elimination of a retrieval requirement is a significant cost savings.

To the best of the committee's knowledge, no self-propelled robots or other operational systems have been built using this approach. Indeed, experience suggests that the actual result of applying the swarm principle is that one highly capable robot is not replaced by many robots of lesser capability, but rather *one* such robot. This suggests that real-world applications are likely to depend on the ability to fabricate many small robots inexpensively.

A key challenge is thus to develop ways of assembling microrobots that are analogous to chip fabrication production lines. One step toward meeting this challenge has been instantiated in a concept known as "smart dust," for which actual prototypes have been developed. Smart dust is a concept for a

[19]B. Hölldobler and E.O. Wilson, *The Ants*, Belknap Press of Harvard University Press, Cambridge, MA, 1990, pp. 585-586. In a famous account published in 1921, the entomologist William Beebe described a mill he saw in the Amazonian rain forest that measured some 360 meters across, with each ant taking about $2^1/_2$ hours to complete a circuit. They kept at it for at least 2 days, stumbling along through an ever-accumulating litter of dead bodies, until a few workers finally straggled far enough from the trail to break the cycle. And from there, recalled Beebe, the group resolutely marched off into the forest. See W. Beebe, *Edge of the Forest*, Henry Holt and Company, New York, 1921.

[20]But then, so do most insect nests. Honeycombs, wasps' nests, and other famous examples are the exception rather than the rule.

[21]R.A. Brooks and A.M. Flynn, "Fast, Cheap and Out of Control: A Robot Invasion of the Solar System," *Journal of the British Interplanetary Society* 42:478-485, 1989.

highly distributed sensor system.[22] Each dust mote has sensors, processors, and wireless communications capabilities and is light enough to be carried by air currents. Sensors could monitor the immediate environment for light, sound, temperature, magnetic or electric fields, acceleration, pressure, humidity, selected chemicals, and other kinds of information, and the motes, when interrogated, would send the data over kilometer-scale ranges to a central base station, as well as communicate with local neighbors.

This architecture was the basis of an experiment that sought to track vehicles with an unmanned aerial vehicle (UAV)-delivered sensor network.[23] The prototype sensors were approximately a cubic inch in volume and contained magnetic sensors for detecting vehicles (at ranges of about 10 meters), a microprocessor, radio-frequency communications, and a battery or solar cell for power. With six to eight air-delivered sensor motes landed diagonally across a road at about 5-meter intervals, the sensor network was able to detect and track vehicles passing through the network, store the information, and then transfer vehicle track information from the ground network to the interrogating UAV and then to the base camp.

The subsumption architecture also asserts that this robust behavior can emerge from the bottom up.[24] For example, in considering the problem of an autonomously functioning vehicle (i.e., one that drives itself), a series of layers can be defined that

- Avoid contact with objects (whether the objects move or are stationary),
- Wander aimlessly around without hitting things, and
- Explore the world by seeing places in the distance that look reachable and heading for them.

Any given level contains as a subset (subsumes) the lower levels of competence, and each level can be built as a completely separate component and added to existing layers to achieve higher levels of competence. In particular, a level 0 machine would be built that simply avoided contact with objects. A level 1 machine could be built by adding another control layer that monitors data paths in the level 0 layer and inserts data onto the level 0 data paths, thereby subsuming the normal data flow of level 0. More complex behavior is thus built on top of simpler behaviors.

Brooks claims that the subsumption architecture is capable of accounting for the behavior of insects, such as a house fly, using a combination of simple machines with no central control, no shared representation, slow switching rates, and low-bandwidth communication. This results in robust and reliable behavior despite its limited sensing capability and an unpredictable environment, because individual behaviors can compensate for each others' failures, resulting in coherent and emergent behavior despite the limitations of the component behaviors. A number of robots have been built using subsumption architectures. Of particular note is Hannibal,[25] a hexapod with more than 100 physical sensors and 1,500 augmented finite-state machines grouped into several dozen behaviors split over eight on-board computers.[26]

8.2.3 Robotics 2: Bacterium-inspired Chemotaxis in Robots[27]

The problem of locating gradient sources and tracking them over time is an important problem in many real-world contexts. For example, fires cause temperature gradients in their immediate vicinity;

[22]See, for example, http://robotics.eecs.berkeley.edu/~pister/SmartDust/.

[23]See http://robotics.eecs.berkeley.edu/~pister/29Palms0103/.

[24]R.A. Brooks and A.M. Flynn, "Fast, Cheap and Out of Control," 1989.

[25]C. Ferrell, "Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators," Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1993.

[26]A finite-state machine is a machine with a finite number of internal states that transitions from one state to another on the basis of a specified function. That is, the argument of the function is the machine's previous state, and the function's output is its new state. An augmented finite-state machine is a finite-state machine augmented with a timer that forces a transition after a certain time.

[27]Material in Section 8.2.3 is based on excerpts from A. Dhariwal, G.S. Sukhatme, and A.A.G. Requicha, "Bacterium-inspired Robots for Environmental Monitoring," *International Conference on Robotics and Automation*, New Orleans, LA, April 2004.

chemical spills lead to chemical concentration gradients in the soil and/or water; ecosystems host gradients of light, salinity, and pH. In many cases, the source intensity of these gradients varies with time (e.g., because of movement of the source), and there may be multiple sources for any given characteristic (e.g., two fires causing a complex temperature gradient).

Autonomous detection, location, and tracking of gradient sources would be very helpful for those trying to study or respond to the environment. Using robots, an environmental scientist might need to find the source(s) of a given toxic chemical, whereas a firefighter might need to locate the source(s) of a fire in order to extinguish it.

Noting that other approaches for locating and tracking gradient sources were primarily useful in static or quasi-static environments, and inspired by biological studies of how bacteria are attracted to gradient sources of nutrition, Dhariwal et al.[28] sought to develop a strategy for finding gradient sources that worked well with sources that are small, weak, mobile, or time-varying in intensity. Specifically, their algorithm is based on the repetition of a straight-line run for a certain time, followed by a random change in direction that sets up the direction for a new run. If the bacterium senses a higher concentration in its immediate environment, the run length is longer. Thus, although the bacterium still undergoes a random walk, it is a random walk biased in the direction of the gradient source.

This algorithm is also well suited for implementation in a simple robot. That is, only the last sensor reading must be stored, and so memory requirements are lower. Because only one computation has to be done (a comparison between the present and the previous sensor reading), processing requirements are minimal.

Dhariwal et al. compared the performance of this algorithm with a simple gradient descent algorithm. They found that for single, weak sources, the simple gradient algorithm displayed better performance. However, the bacterium-inspired algorithm displayed better performance in locating and tracking multiple and/or dissipative sources and in covering the entire area in which the gradient can be found.

8.2.4 Self-healing Systems

In the past few years, the term "self-healing" has become a fashionable object of study and interest in the academic and research computer science communities[29] and in the marketing materials of information technology (IT) companies such as IBM,[30] Microsoft,[31] Sun,[32] and HP.[33] Despite (or because of?) this level of interest, there is no commonly accepted definition of "self-healing" or agreement of what functionality it encompasses or requires.

[28]A. Dhariwal, G.S. Sukhatme, and A.A.G. Requicha, "Bacterium-inspired Robots for Environmental Monitoring," *IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 25-30, 2004, available at http://www-lmr.usc.edu/~lmr/publications/Icra04bact.pdf.

[29]Workshop on Self-healing, Adaptive and Self-managed Systems (SHAMAN), June 23, 2002, available at http://www.cse.psu.edu/~yyzhang/shaman/proc.html; ICSE 2003 Workshop on Software Architectures for Dependable Systems, May 2003 (for more information, see http://www.cs.kent.ac.uk/events/conf/2003/wads/); David Garlan, Self-healing Systems Course, #17-811, Carnegie Mellon University seminar, Spring 2003 (for more information see http://www-2.cs.cmu.edu/~garlan/17811/); D. Garlan, J. Kramer, and A. Wolf, eds., *Proceedings of the First Workshop on Self-healing Systems*, ACM Press, New York, 2002.

[30]M. Hamblen, "IBM to Boost Self-healing Capabilities in Tivoli Line," *Computerworld*, April 4, 2003, available at http://www.computerworld.com/softwaretopics/software/story/0,10801,80050,00.html.

[31]"Windows 2000 Professional: Most Reliable Windows Ever," December 5, 2000, available at http://www.microsoft.com/windows2000/professional/evaluation/business/overview/reliable/default.asp.

[32]"Sun and Raytheon Create Open, Adaptive, Self-healing Architecture for DD 21," available at http://wwws.sun.com/software/jini/news/Jini-Raytheon.pdf.

[33]"HP Delivers Self-healing and Virtual Server Software to Advance the Adaptive Enterprise," press release, May 6, 2003, available at http://www.hp.com/hpinfo/newsroom/press/2003/030506c.html.

In fact, many of the techniques described as self-healing are familiar to the decades-old hardware field of reliable systems, also known as fault tolerance or high availability. These techniques, such as fault detection, fault masking, and fault tolerance, are in common use when designing hardware to improve the reliability and availability of large systems. This is most likely because hardware designers, unlike software programmers, long ago accepted the unavoidable reality that components of their designs will fail at some point. (It also helps immeasurably that hardware failures are often easier to characterize than software failures.) In areas with extremely high demands for reliability, such as aerospace or power plants, these fault-tolerance techniques have become quite sophisticated, as have mechanisms for testing system operation. The oldest and most accepted use of the term self-healing is found in networking;[34] networks from the original ARPANET (and even the public switched telecommunications network) to modern peer-to-peer embedded networks are self-healing in the sense that traffic is routed around unresponsive nodes.

In contrast, until quite recently, software quality has focused on producing bug-free products, by an intensive effort of careful design, code review, and extensive prerelease testing. However, when bugs do occur, software typically has no ability to detect or react to them, or to continue to operate. This was a workable strategy for much of the history of modern software, but the continuing rise of the complexity of software applications has made formal review or correctness proofs inadequate to provide minimum levels of reliability.[35]

This rise in complexity and the resulting rise in human cost of configuration and maintenance of software applications has spurred interest in self-healing, hoping to shift much of the burden of this configuration and maintenance back to the software. The idea is that, like its biologically analogous namesake, a self-healing system would detect the presence of nonfunctioning (or, more challengingly, malfunctioning) components and initiate some response to continue proper overall functionality, preferably without any centralized or external force (such as a system administrator) required. The most common implementation today seems to be one of reconfiguration: if a fault is detected, a spare hardware component is brought into play. This is "healing" only in the loosest sense, although it certainly is a valid fault tolerance technique. However, it doesn't translate well to software-only failures.

None of the systems that describe themselves as self-healing (such as Microsoft Windows 2000, IBM DB/2, or Sun's Jini) seem to actually employ biological principles, other than in the grossest sense of having redundancy. However, one research project that is inspired very explicitly by biology is Swarm at the University of Virginia.[36] The Swarm programming model defines units as individual cells, which can both reproduce through cellular division and die. Additionally, they can emit signals at various strengths and respond to the aggregate strength of signals in the environment. For example, a system set to grow to a certain size would start with a single cell that emitted a small amount of signal and with a program set to reproduce if the aggregate signal was at a certain threshold. Until the total amount of signal exceeded that threshold, the cells would continue to divide, but they would stop once the threshold was exceeded. If cells were to fail or otherwise be deleted, other cells would respond by dividing again to bring the signal back to the threshold. This is indeed a primitive form of self-healing. However, this programming model is unlikely to catch on for complex tasks without significant higher-level abstractions available.

---

[34]W.D. Grover, "The Self-healing Network: A Fast Distributed Restoration Technique for Networks Using Digital Cross-connect Machines," *Proceedings of the IEEE Global Telecommunications Conference*, Tokyo, 1987, pp. 1090-1095.

[35]In his lecture on receiving the ACM Turing Award in 1980, C.A.R. Hoare said, "There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies." Lecture available at http://www.braithwaite-lee.com/opinions/p75-hoare.pdf.

[36]G. Selvin, D. Evans, and L. Davidson, "A Biologically Inspired Programming Model for Self-healing Systems," *Proceedings of the First Workshop on Self-Healing Systems*, November 2002, available at http://www-2.cs.cmu.edu/~garlan/woss02/.

8.2.5 Immunology and Computer Security[37]

The mammalian immune system is an information processor—this is clear from its ability to distinguish between self and nonself. (Section 5.4.4.3 provides a brief introduction to the immune system.) Some have thus been drawn to the architecture of the immune system as a paradigm of information processing that might be useful in solving a variety of different computational problems. Immunological approaches have been proposed for solving problems in computer security, semantic classification and query, document and e-mail classification, collaborative filtering problem, and optimization.[38] This section concentrates on computer security applications.

8.2.5.1 Why Immunology Might Be Relevant

Computer and network security is intended to keep external threats at bay, and this remains an intellectually challenging problem of the highest order. It is useful to describe two general approaches to such security problems. The first, widely in use today, is based on the notion of what might be called environmental control—the idea that by adequately controlling the environment in which a computer or network functions, better security can be obtained. The computer or network environment is defined broadly, to include security policy (who should have what rights and privileges), resources (e.g., programs that provide users with computing or communications capability), and system configuration. In support of this approach, a number of reports[39] cite security problems that arise from flaws in security policy, bugs in programs, and configuration errors and argue that correcting these flaws, bugs, and errors will result in greater security.

A complementary approach is to take as a given the inability to control the computing or network environment.[40] This approach is based on the idea that computer security can result from the use of system design principles that are more appropriate for the imperfect, uncontrolled, and open environments in which most computers and networks currently exist. Note that there is nothing mutually exclusive about the two approaches—both could be used in the design of an effective overall approach to system or network security.

For inspiration in addressing problems in computer security, some researchers have considered the immune system and the unpredictable and largely hostile environment in which it functions.[41] That is, the unpredictable pathogens to which the immune system must respond are analogous to some of the threats that computer systems face, and the principles underlying the operation of the immune system may provide new approaches to computer security.

8.2.5.2 Some Possible Applications of Immunology-based Computer Security

A variety of loose analogies between computer security and immunology are intuitively obvious, and there is clearly at least a superficial conceptual connection between the protection afforded to

[37]The discussion in Section 8.2.5 owes much to Stephanie Forrest of the University of New Mexico.

[38]For a view of the immune system as information processor, see S. Forrest and S. Hofmeyr, "Immunology as Information Processing," *Design Principles for Immune Systems and Other Distributed Autonomous Systems*, L.A. Segal and I.R. Cohen, eds., Oxford University Press, 2000. For an overview of various applications of an immunological computing paradigm, see www.hpl.hp.com/personal/ Steve_Cayzer/downloads/030213ais.ppt and references therein.

[39]National Research Council, *Cybersecurity Today and Tomorrow: Pay Now or Pay Later*, National Academy Press, Washington, DC, 2002.

[40]This discussion is based on A. Somayaji, S. Hofmeyr, and S. Forrest, "Principles of a Computer Immune System," *Proceedings of the 1997 Workshop on New Security Paradigms*, ACM Press, Langdale, UK, 1998, pp. 75-82.

[41]One of the first papers to suggest that self-nonself discrimination, as used by the immune system might be useful in computer security was by S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself Discrimination in a Computer," *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 202-212. This paper focused mainly on the issue of protection against computer viruses but set the stage for a great deal of subsequent work.

human beings by the immune system and computer security. The following examples are adapted from Somayaji et al.:[42]

• *Protecting active processes on a single host*. For this application, a computer running multiple processes might be conceptualized as a multicellular organism (in which each process is analogous to a cell). An adaptive immune system could be a detector process that queried other processes to see whether they were functioning normally. If not (i.e., if the detector process found "nonself" in its probes), the adaptive system could slow, suspend, kill, or restart the misbehaving process. One approach to detection (positive detection) is based on the establishment of a profile of observed normal behaviors and using that profile to notice when a program behaves abnormally.[43]

• *Protecting a network of computers.* For this application, each computer in a network might be conceptualized as a cell in an individual. Each process would still be considered as a cell, but now an individual is a network of computers. (Another possible analogy for the network of computers is that each computer represents a single organism and population-level protections are achieved by the collective group through independence, diversity, and sharing of information.) An adaptive detector process could be implemented as described above, with the added feature that these detectors could migrate between computers, thereby enabling all computers on the network to benefit from the detection of a problem on one of them.

• *Protecting a network of disposable computers*. This application is similar to that described above, with the addition that when an anomaly is detected, the problematic machine can be isolated, rebooted, or shut down. If the true source of the anomaly were outside the network, a detector process or system could stand in for the victimized machine, doing battle with the malicious host and potentially sacrificing itself for the good of the network. Note that this application requires that hosts be more or less interchangeable—otherwise the network could not afford the loss of a single host.

8.2.5.3 Immunological Design Principles for Computer Security

The immune system exhibits a number of characteristics—one might call them design principles—that could reasonably describe how effective computer security mechanisms might operate in a computer system or network. (As in Section 5.4.4.3, "immune system" is understood to mean the *adaptive* immune system.) For example, the immune system is:[44]

• *Distributed*, in the sense that it has no central point of control. Instead, the components of the immune system interact locally to mount responses to foreign pathogens (e.g., pathogen detectors [lymphocytes] operate locally to flag the presence of pathogens). By contrast, a computer system based on centralized control is vulnerable to "decapitation"—a successful attack on the point(s) of centralized control renders the system entirely useless.[45]

• *Diverse*, in the sense that because of the ways in which pathogen detectors are produced, each individual human being can detect a somewhat different set of pathogens—a diversity that protects

[42]A. Somayaji, S. Hofmeyr, and S. Forrest, "Principles of a Computer Immune System," *Proceedings of the 1997 Workshop on New Security Paradigms,* ACM Press, Langdale, UK, 1998, pp. 75-82.

[43]An alternative approach is to use a randomly generated detector or set of detectors, living for a limited amount of time, after which it would be replaced by another detector. Detectors that proved particularly useful during their lifetimes (e.g., by detecting new anomalies) could be given a longer life span or allowed to spawn related processes. This approach has been used by Forrest et al. in the development of a network intrusion detection system known as LISYS.

[44]This discussion of the immune system is based on S. Forrest and S. Hofmeyr, "Immunology as Information Processing," *Design Principles for Immune Systems and Other Distributed Autonomous Systems*, L.A. Segal and I.R. Cohen, eds., Oxford University Press, New York, 2001.

[45]A distributed, mobile agent architecture for security was also proposed in M. Crosbie and G. Spafford, "Active Defense of a Computer System Using Autonomous Agents," Technical Report 95-008, Department of Computer Science, Purdue University, 1995.

the species as a whole. By contrast, computer system monoculture (i.e., lack of diversity) implies that systems share vulnerabilities, and a successful attack on one system is likely to succeed on other systems as well.[46]

- *Autonomous*, in the sense that it classifies and eliminates pathogens and repairs itself by replacing damaged cells without the benefit of any centralized control mechanism. Given the growing security burden placed on today's computer systems and networks, it will be increasingly desirable for these system and networks to manage security problems with minimal human intervention.
- *Tolerant of error*, in the sense that some mistakes in identification of pathogens (false positives or false negatives) are not generally fatal and do not cause immune system collapse, although they can cause lingering autoimmune disease. Such tolerance is in part the result of a multilayered design of the immune system, in which multiple, independently architected layers of defense ("defense in depth") operate to provide levels of protection that are not achievable by any single mechanism.[47] Computer systems are often not so tolerant, and small errors or problems in some part of a system can lead to significant malfunctions.
- *Dynamic*, in the sense that pathogen detectors are continually being produced to replace those that are (routinely) destroyed. These detectors, circulated through the body, provide whole-body protection and may be somewhat different in each new generation (in that they respond to different pathogens). Because these detectors turn over, the immune system has a greater potential coverage. By contrast, protection against computer viruses, for example, is based on the notion that all threat viruses are known—and most antiviral systems are unable to cope with a new virus for which no signature is known.
- *Capable* of remembering (adaptable), in the sense that the immune system can learn about new pathogens and "remember" how it coped with one pathogen in order to respond more effectively to a future encounter with the same or a similar pathogen. It can also "forget" about nonself entities that are incorporated into the body (e.g., food gets turned into body parts). Computer systems must also adapt to new environments, as for example, when new software is added legitimately, as well as identify new threats.
- *Imperfect*, in the sense that individual pathogen detectors do not identify pathogens perfectly, but rather respond to a variety of pathogens. Greater specificity is obtained through redundant detection of a pathogen using different detector types. By contrast, computer security systems that look for precise signatures of intruders (e.g., viruses) are easily circumvented.
- *Redundant*, in the sense that multiple and different immune system detectors can recognize a pathogen. Pathogens generally contain many parts, called epitopes, that are recognized by immune system detectors; thus, failure to recognize one epitope is not fatal because many others are available for recognition.
- *Homeostatic*, in the sense that the immune system can be regarded as one mechanism through which the human body seeks to maintain a stable internal state despite a changing environment. A computer system can be designed to autonomously monitor its own activities, routinely making small corrections to maintain itself in a "normal" state, even in the face of wide variations in inputs, such as those caused by intruders.[48]

At a deeper level, it is instructive to ask whether the particular methods by which the immune system achieves these characteristics (implements these design principles) have potential relevance to computer security. To address this issue, deeper and more detailed immunological knowledge is necessary, but some work has been done in this area and is described below.

[46]For more discussion of this point, see Computer Science and Telecommunications Board, National Research Council, *Computers at Risk: Safe Computing in the Information Age*, National Academy Press, Washington, DC, 1991.

[47]This point suggests that detection mechanisms are biased to be more tolerant of false negatives than false positives, because threats that are unaffected by one layer (i.e., false negatives) might well be intercepted by another.

[48]A. Somayaji and S. Forrest, "Automated Response Using System Call Delays," *Journal of Computer Security* 6:151-180, 1998.

8.2.5.4 An Example: Immunology and Intruder Detection

To detect pathogens, the immune system generates detectors that can bind to pathogens, and only to pathogens (i.e., do not bind to self). (A detector binding to a pathogen is the marker of a detection event.) To vastly simplify a complex process, the immune system first generates detectors at random. Through a process known as tolerization, detectors that bind to self are destroyed, leaving only detectors that bind to nonself at the end; these detectors are called mature. Mature detectors are released throughout the body; if they do not bind to a nonself entity in some period of time (several days?), they are destroyed (self-destruct?). Those that do bind to nonself entities are regarded as activated detectors. However, an activated detector must receive a second, independent signal (created by the binding of another type of detector to the same pathogen costimulation) to become capable of surviving for a long period of time. These long-term survivors are memory detectors that enable subsequent immune responses to be generated much more rapidly and are the basis for long-term immunity. (Memory detectors have lifetimes that range from days to the lifetime of an organism, and the underlying mechanisms governing their lifetimes are not well understood.)

In the context of computer security, Forrest and Hofmeyr have described models for network intrusion detection and virus detection.[49] In the network intruder detection example, self is defined through a set of "normal" connections in a local area network. Each connection is defined by a triplet consisting of the addresses of the two parties in communication with each other and the port over which they communicate (a total of 49 bits), and the set of all triplets (normal triplets) generated during a training period represents, by definition, normal operation of the network.

When the network operates outside the training period, the intrusion detection system generates random detector strings that are 49 bits in length. Matches are declared according to an "*r*-contiguous-bit" rule—a match is deemed to exist if a random detector string matches some normal triplet in at least *r* contiguous bit positions. In this phase (the maturation phase), detector strings that match some normal triplet are eliminated, leaving only mature detectors that have not matched any normal triplet.

Mature detectors—which might match an abnormal triplet that arises as the result of a network intrusion—are then exposed to the nontraining network operation. If a mature detector matches some triplet found in the nontraining network operation, such a match is potentially a sign of network intrusion (which would be indicated by an unusual pair of systems communicating over an unusual port). If a mature detector does not match any such triplet in a given period of time, it too is eliminated.[50] The remaining detectors—activated detectors—are now fully capable of signaling the presence of abnormal triplets.

However, as a further guard against false positives, the system invoked a mechanism inspired by immunological *costimulation*. Costimulation reduces the likelihood that a pathogen will be indicated when there is no pathogen. After negative selection of lymphocytes occurs, the remaining now-mature lymphocytes are likely to bind to nonself entities encountered. However, before the lymphocytes are "promoted" to memory cells, they must be activated by a costimulatory signal indicating that the substances to which they bind are in fact pathogens. This costimulatory signal is generated independently and reduces the incidence of pathogen detectors that are overly sensitive (and hence the likelihood of autoimmune reactions).

The intrusion detection system implements a costimulatory mechanism as the requirement of a human confirmation of behavior flagged as potentially anomalous—that is, it presents matches signaled by an activated detector to a human operator for confirmation. If the system receives human confirmation within a fixed amount of time, the activated detector responsible for the warning is made

[49]S. Forrest and S. Hofmeyr, "Immunology as Information Processing," *Design Principles for Immune Systems and Other Distributed Autonomous Systems*, L.A. Segal and I.R. Cohen, eds., Oxford University Press, New York, 2001.

[50]In fact, the mature detector is eliminated if it does not exceed some parametrically set threshold (the activation threshold) for the number of matches to abnormal triplets.

into a memory detector (with an indefinite lifetime and a subsequent activation threshold of 1). However, if human confirmation is not forthcoming, the detector responsible is eliminated.

An intrusion detection product based on this approach was introduced in early 2003.[51] The real-world success of this product remains to be seen.

8.2.5.5 Interesting Questions and Challenges

8.2.5.5.1 Definition of Self Any paradigm for computer security that is based on the differentiation of self from nonself must imply some operational definition of self that represents normal and benign operation. It is clear that a good definition is matched to the signature of the threat being defended against, and hence the designer must be able to answer the question, "How would I know my system were under attack?" Thus, self might be definable in terms of memory access patterns on a single host, TCP/IP packets entering and leaving a single host, the collective behavior of a local network of computers, network traffic through a router, instruction sequences in an executing or stored program, sequences of system calls, user behavior patterns, or keyboard typing patterns.[52]

At the same time, computer security must account for the fact that "self" on a computer system, even one that has not been subject to threat or intrusion, changes over time. New users are added, new software is added, and files are created, deleted, and modified in the course of normal activity, even though all such activities may also occur in the course of an attack. That is, the notion of self must be dynamically modifiable.

These points suggest that better insights into characterizing threat signatures dynamically would be helpful if immunological approaches are to be used to enhance computer security.

8.2.5.5.2 More Immunological Mechanisms Another intellectual challenge is to incorporate more of what is known about immunology into computer security. Thus, it is interesting to consider how a number of immunological mechanisms known today might be useful in making the analogy closer, using the functions and design principles of these specific mechanisms within the general context of an immunologically based approach to computer security. One such mechanism is antigen processing and the major histocompatibility complex (MHC). Some pathogens have the ability to "hide" within cells generally recognized as self. Because lymphocytes can detect antigens only by binding to them, they are unable to detect pathogens inside friendly cells. Molecules from the MHC have the ability to bring key parts of such pathogens to the surface of those cells, thereby enabling the lymphocytes to detect them. Moreover, each individual has a different set of MHC molecules; hence the kinds of hidden pathogens that can be brought to a cell's surface are different for different individuals, providing an important immunological diversity in the population as a whole.

An analogous mechanism was implemented in the intrusion detection system described above. Just as certain pathogens are able to hide within cell interiors to avoid detection, the use of detectors that can match a number of subsets of nonself patterns (so that fewer detectors are needed) implies that there exist some nonself patterns for which no detectors can be generated. In other words, a detector capable of matching such nonself patterns would also match some patterns found in self. Furthermore, as the number of nonself patterns that can be recognized by a single detector increases, the number of problematic nonself patterns also increases. Because they result from the structure of the set of self patterns, dynamic change in the detectors cannot find them.

A solution that proved to be effective at reducing the overall number of holes (i.e., gaps in coverage) is multirepresentation—different representations are used for different detectors. One way of achieving

[51]See http://www.sanasecurity.com.

[52]S. Forrest, S.A. Hofmeyr, and A. Somayaji, "Computer Immunology," *Communications of the ACM* 40(10):88-96, 1997.

this is for each detector to have a randomly generated permutation rule, according to which all data path triples are permuted before being matched against the detector. This effectively changes the structure of the self set for each detector, with the result that different detectors will be subject to different holes. Consequently, where one detector fails to detect a nonself triple, another may succeed. Multirepresentation was particularly effective at reducing the number of holes when the nonself patterns were similar to self patterns. To deal with this problem, the bits in a given triplet of connection triplets were randomly permuted before presentation to detectors, just as the specific MHC molecules that are operating to bring pathogens to the surface are probabilistically determined (with respect to an averaging over the population).

8.2.5.6 Some Possible Difficulties with an Immunological Approach

Although these analogies have appeal, it remains to be seen how far they can be pushed. Given that the immune system is a very complex entity whose operation is not fully understood, a bottom-up development of a computer security system based on the immune system is not possible today. The human immune system has evolved to its present state due to many evolutionary accidents as well as the constraints imposed by biology and chemistry—much of which is likely to be artifactual and mostly irrelevant to the underlying principles that the system embodies and also to the design of a computer security system. Further, the immune system is oriented toward problems of survival. By contrast, computer security is traditionally concerned with confidentiality, accountability, and trustworthiness—and the relevance of immunological processes to confidentiality and accountability is entirely unclear today.

8.2.6 Amorphous Computing

An area of research known as amorphous computing seeks to understand how to obtain "coherent behavior from the cooperation of large numbers of unreliable parts that are interconnected in unknown, irregular, and time-varying ways."[53] This work, inspired by observations of cooperative and self-organizing biological phenomena, seeks to identify the engineering principles that can be used to observe, control, organize, and exploit the behavior of cooperating multitudes for human purposes such as the design of engineered artifacts.

An individual entity in a collection of cooperating multitudes has the following characteristics:

- It is inexpensive, in the sense that it is easy to create large numbers of them. For all practical purposes, each entity is identical to every other one.
- It is locally guided or programmed. That is, the guidance or programming is carried by the entity "on-board" rather than being resident elsewhere in the overall system. As a consequence of fabrication, the guidance or programming aboard any given entity is identical to that aboard every other entity.
- It communicates with nearby entities, but in a stochastic manner without the need for precise interconnections and testing. Note also that the ability to function in a stochastically connective environment implies that the overall macrosystem is robust in the face of damaged or nonoperational components. Furthermore, by eliminating the need for precision interconnections, these entities can reduce the enormous costs usually associated with interconnection in traditional forms of assembly, costs that are generally higher than those associated with individual elements.
- It interacts with its environment locally, so that the entity is directly knowledgeable about some aspect of its immediate environment but not about anything more global. To the extent that an individual entity gains global knowledge about the environment, it is as the result of a self-organizing process that develops such information and transmits it to all entities in the system. Similarly, any on-board effectors affect only the immediate environment.

[53]See http://www.swiss.ai.mit.edu/projects/amorphous/.

These characteristics are easily obtained by biology and are increasingly true for certain artifacts that result from today's chip fabrication technologies. A metaphor with some resonance is that of "paintable" computers—a paint that can be applied to a surface, in which are suspended millions of computing and MEMS-like entities that communicate with each other and interact with the surface on which they are painted. (MEMS is an acronym for microelectromechanical systems.)

The vision presented by Abelson et al.[54] is that smart materials may reduce the need for strength and precision in mechanical and electrical apparatus, through the application of computation. For example, coating a building or a bridge with "smart paint" may enable it to report on traffic loads and wind loads, to monitor the integrity of the structure, to resist buckling, or to heal small cracks by shifting material around. A different kind of smart paint may make it possible to build clean rooms with "active skins," lined with cilia that can push dirt and dust into a corner for removal. Still other paints may enable walls that sense vibration or actively cancel noise. "Smart dust," with light sensors in each particle, could be spread over a wide area to recognize shadows or other traffic passing overhead.

In short, the hope is to create systems with unprecedented responsiveness to their environment. Abelson et al. further argue that the study of amorphous computing has implications for software design in a more general sense. Specifically, a software problem has long been recognized—the dependence of greater functionality of software on increasingly complex software packages and systems. Today, software is mostly developed "by hand," and each line is individually coded. One obtains a high degree of detailed control in this manner, but reliably abstracting the higher-level behavior of a software system so developed is highly problematic. Principles of amorphous computing may enable a more top-down specification of systems that more closely tracks how humans define the functionality they wish to obtain from software.

Amorphous computing may be applicable to fabrication as well. For example, consider amorphous computing entities that are capable of some mechanical interaction with the substrate on which they are painted (e.g., they might expand or contract in certain directions). Nagpal has demonstrated the feasibility of an amorphous computing substrate that is capable of pattern formation (Box 8.1); if the entities making up this formation have the mechanical property described, it is conceivable that they might be able to warp a sheet onto which they were painted into a three-dimensional structure.

It is also conceivable that the vision described in amorphous computing and other approaches to that area could be extended so that appropriately configured microentities could be programmed to self-assemble into useful physical structures on the nanoscale. These structures might be useful to end users in and of themselves, or might serve as nanofabrication machinery that could construct other structures useful to the end user. In particular, the large macromolecules involved in the biochemistry of life—specifically protein molecules—demonstrate the ability to configure themselves into structures, and some research seeks to co-opt biochemical machinery to assemble structures designed for entirely human purposes (as described in Section 8.4.3).

8.3 BIOLOGY AS IMPLEMENTER OF MECHANISMS FOR COMPUTING

8.3.1 Evolutionary Computation[55]

8.3.1.1 What Is Evolutionary Computation?

Evolutionary computation is inspired by genetics and evolutionary events.[56] Given a particular problem for which a solution is desired, evolutionary computation requires three components:

[54] H. Abelson, T.F. Knight, G.J. Sussman, et al., "Amorphous Computing," available at http://www.swiss.ai.mit.edu/projects/amorphous/white-paper/amorph-new/amorph-new.html.

[55] The discussion in Section 8.3.1 owes much to Melanie Mitchell, now at Portland State University in Oregon.

[56] Evolutionary computation is a generic name for techniques that are based loosely on evolutionary principles. There are a number of variants, including evolutionary programming, evolution strategies, genetic programming, and genetic algorithms, which have somewhat different emphases but share the generic approach.

Box 8.1
Pattern Formation Using Identical Autonomous Agents

In a 2001 Ph.D. thesis, Nagpal describes a language for instructing a sheet of identically programmed, flexible, and randomly but densely distributed autonomous agents ("cells") to assemble themselves into a predetermined global shape, using only local interactions. A wide variety of global shapes and patterns can be synthesized (patterns including flat layered shapes, all plane Euclidean constructions, and a variety of tessellation patterns) using only local interactions between identically programmed deformable cells. That is, the global shape results from a coordinated set of local shape changes in individual cells. Despite being programmed identically, each cell deforms in its own individualized manner, depending on the behavior and state of its neighbors. (The governing structural metaphor is that of epithelial cells, which generate a wide variety of structures: skin, capillaries, and many embryonic structures (gut, neural tube) through the coordinated effect of many cells changing their individual shape.)

The global shape is specified as a folding construction on a continuous sheet, using a small set of axioms, simple initial conditions (edges and corners of the sheet), and two types of folds. From an engineering standpoint, the significance of global shape description is that a process that is inherently local can be harnessed to produce a shape of known configuration. This differs significantly from approaches based on cellular automata, in which the local-to-global relationship is not well understood and there is no framework for constructing local rules to obtain any desired pattern (and patterns "emerge" in a non-obvious way from the local interactions).

In this formalism, the specific global shape desired uniquely determines the program executed by all cells. The cellular program is based on several (biologically inspired) primitives for interacting with the cell's local environment. A cell can change the local environment in ways that create the equivalent of chemical gradients, query its local neighborhood and collect information about the state of local companions (e.g., collect neighboring values of a gradient), broadcast messages to all the cells in its local neighborhood, invert its polarity, connect with neighbors in physical contact to establish communication (thus allowing multiple layers of the sheet to act as a single fused layer), and fold itself along a particular orientation by calling the local fold within its program with two arguments: a pair of neighbors and a cell surface.

Each cell has limited resources and reliability. All cells execute the same program and differ only in a small amount of local dynamic state. The cell program does not rely on regular cell placement, global coordinates, or synchronous operation. Robustness against a small amount of random cell death is achieved by depending on large and dense cell populations, using average behavior rather than individual behavior, trading off precision for reliability, and avoiding any centralized control. Further, global coordinates are not required, because cells are able to "discover" positional information. An average cell neighborhood of 15 is sufficient to reliably self-assemble complex shapes and geometric patterns on randomly distributed cells.

SOURCE: R. Nagpal, "Programmable Self-assembly: Constructing Global Shape Using Biologically-inspired Local Interactions and Origami Mathematics," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, June 2001.

• A population of candidate solutions to the problem. For example, these candidate solutions may be a sequence of amino acids that can fold into some protein, a computer program, some encoding of the design for something, or some set of rules in a production system.

• A "fitness" metric by which the "goodness" of a candidate solution can be evaluated. For example, if the program was intended to model the output of some designer circuit, the fitness metric might be based on the performance of a candidate program acting on a test case. That is, given the test case, the fitness metric would be the deviation of the output of the program from a known, appropriate answer. Programs that minimized this deviation would be more fit.

- A mechanism (or mechanisms) by which changes to the candidate solutions can be introduced—portions of different candidate solutions are exchanged, for example, or modified in some small random way.[57]

With these components in place, an evolutionary process takes place. The set of new solutions is evaluated for fitness—those with lower fitness scores are thrown out and those with higher scores are retained. This mutation process is iterated many times, and the result at the end is (supposed to be) a solution that is much better than anything in the starting set.

Initially demonstrated on the solving of what might be called "toy" problems, evolutionary techniques have been used in a variety of business applications, including scheduling and production optimization, image processing, engine design, and drug design. Evolutionary computation has also achieved results that are in some sense competitive with human-developed solutions to quite substantive problems. Competitiveness has a number of possible measures, among them results that are comparable to those produced by a succession of human researchers working on a well-defined problem over a period of years, a result that is equivalent to a previously patented or patentable invention, a result that is publishable in its own right (i.e., independent of its origins), or a result that wins or ranks highly in a judged competition involving human contestants.[58]

Evolutionary computation has demonstrated successes according to all of these measures. For example, there are at least 21 instances in which evolutionary techniques have led to artifacts related to previously patented inventions.[59] Eleven of these infringe on previously issued patents, and ten duplicate the functionality of previously patented inventions in a non-infringing way. Also, while some of the relevant patents were issued many years ago (as early as 1917), others were issued as recently as 2001. Some of the inventions created by evolutionary processes include the ladder filter, the crossover filter, a second-derivative controller, a NAND circuit, a PID (proportional, integrative, and derivative) controller, a mixed analog-digital variable capacitor circuit, a voltage-current conversion circuit, and a cubic function generator. They have also created a soccer-playing program that won its first two games in the Robo Cup 1997 competition and another that ranked in the middle of the field of 34 human-written programs in the Robo Cup 1998 competition, four different algorithms for the transmembrane segment identification problem for proteins, and a variety of quantum computing algorithms, and have rediscovered the Campbell ladder topology for low-pass and high-pass filters.

Evolutionary computation also poses intellectual challenges, as described in the next several sections.

8.3.1.2 Suitability of Problems for Evolutionary Computation[60]

Whether or not an evolutionary approach will be successful in solving a given problem is not yet fully understood. Although many components of a full theory of evolutionary algorithms have been worked out, there are critical gaps that remain open questions.

It is known that the relationship between the representation of a problem, genetic operators, and the objective function is the primary determinant of the performance of an evolutionary algorithm. For any optimization problem, there is always a representation or a genetic operator that makes the optima easy to find with an evolutionary algorithm.[61] In addition, evolutionary algorithms are no better or worse

[57]In biology, "crossover" refers to the process in which chromosomal material is exchanged between chromosomes during cell duplication. The exchanged chromosomal material is analogous to portions of the different candidate solutions. "Mutations" are genetic changes induced as the result of random environmental events.

[58]See http://www.genetic-programming.org.

[59]See http://www.genetic-programming.com/humancompetitive.html. More information on these accomplishments can be found in J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Series in Genetic Programming,* Volume 5, Springer, New York, 2005.

[60]Lee Altenberg of the University of Hawaii was a major contributor to Section 8.3.1.2.

[61]G.E. Liepins and M.D. Vose, "Representational Issues in Genetic Optimization," *Journal of Experimental and Theoretical Artificial Intelligence* 2(2):101-115, 1990.

than any other search algorithm over the space of all problems.[62] Therefore, problem-specific knowledge must be incorporated either implicitly or explicitly in an evolutionary algorithm for it to perform well. Finally, evolutionary algorithms are dynamical systems, and the systems properties necessary to make them good search algorithms are well characterized.[63]

The primary question that remains to tie together the above is the following: How—and when—can knowledge about the problem be translated into representations and genetic operators that produce an evolutionary algorithm with good performance?

In the absence of this critical link in the theory of evolutionary algorithms, the approach taken by designers resorts to the empirical: try it out and see if it works. Evolutionary approaches provide the greatest advantage over other methods in cases where it is not understood how to construct answers from "first principles" (i.e., logico-deductive procedures), but where approximate solutions can be refined by variation and testing. Such problems can be characterized as "difficult inverse problems," where the inverse refers to finding inputs that produce desired outputs of the system in question.

Moreover, evolutionary techniques tend to work best on problems involving relatively large search spaces and large numbers of variables that are not well understood. Evolutionary algorithms have been able to construct and adapt complex neural networks that are intractable analytically or for which derivative-based back-propagation is inapplicable. Genetic programming has produced complex circuits that infringe on patented inventions. By contrast, problems involving small search spaces can usually be searched systematically, and search spaces being well understood generally means that special-purpose heuristics are available. (For example, the Traveling Salesman Problem is reasonably well understood, and there are very good special heuristics for solving that problem.)

For problems in which evolutionary techniques are unable to find global optima, they may nevertheless find very good approximations that are robust to wide-ranging initial conditions. Thus, the solutions generated may be adequate to the task at hand. For this reason, evolutionary techniques may also be better when data are very noisy or in the presence of a varying fitness function: the algorithm may rapidly produce approximate solutions that track the changing environment, just as evolving species can track environmental changes. (An example of a problem calling for a varying fitness function might be a robot that must learn, online, in a dynamic environment, where the task facing the robot changes over time.)

8.3.1.3 Correctness of a Solution

One of the most challenging aspects of evolutionary computation is evaluating the correctness of a solution derived through evolutionary means. Because evolutionary solutions are cumulative, in the sense that they build on previous solutions, the design process does not have an opportunity to develop solutions that are clean and elegantly designed from first principles. Human inspection of a solution so derived is unlikely to yield much insight. Thus, essentially the only way known today to assess the correctness of such a solution is to subject it to extensive testing. Rather than a human being understanding how the solution achieves its goals, the proposed solution convinces a human being that it will do so.

Note, however, that ascertaining the correctness of any large computational artifact (e.g., a complex software system or a VLSI chip) depends to a large degree on testing. Of course, because the thought and decision-making processes of human beings are not available to public inspection, it is only by observing a human being in action that one develops confidence in the designer's ability to perform

[62]D.H. Wolpert and W.G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation* 1(1):67-82, 1997, available at http://citeseer.ist.psu.edu/wolpert96no.html.

[63]L. Altenberg, "Open Problems in the Spectral Analysis of Evolutionary Dynamics," pp. 73-102 in *Frontiers of Evolutionary Computation*, A. Menon, ed., Genetic Algorithms and Evolutionary Computation Series, Volume 11, Kluwer Academic Publishers, Boston, MA, 2004.

appropriately under certain circumstances. Thus, in the limit of increasing complexity, testing an evolutionary solution may resemble the Turing test. (In the Turing test, an outside observer is asked to distinguish between a human being's answers to a set of questions and a computer's answers. The computer is said to have passed the Turing test if the outside observer is unable to distinguish between the two.)

8.3.1.4 Solution Representation

In biological organisms, the genetic code of DNA is subject to changes (e.g., mutation), and the impact of these changes becomes manifest as the new mutated code is involved in the reproductive process. That is, the particular DNA sequence of an organism can be said to be biology's representation of a "solution" to the problem of adapting the organism to a particular set of evolutionary selective pressures.

From the standpoint of someone solving a problem with techniques from evolutionary computation, the question arises as to the analogue of DNA. More formally, how is a solution to a computational problem to be represented?

In general, the solution to a computational problem is an algorithm. However, an algorithm can be represented in many different ways. Just as data can be represented as lists of numbers or in graphical form, computer programs (which embed algorithms) can be represented as "source code" that is readable by human beings or as "object code"—the raw ones and zeros of binary computation.

If candidate solutions are to be computer programs, one might imagine that their machine language representation is an obvious possible representation. However, changing a machine language program one bit at a time, at random, is highly likely to prevent the (modified) program from running at all (because previously valid op-codes will be turned into invalid ones), and a nonrunning program is useless. The same comments apply to the source code of a program. By randomly changing characters in the source code file, the most likely result is a program that will not compile and therefore cannot be evaluated in any meaningful way. Thus, attempting to evolve a binary program or the source code of a program would likely result in an extraordinarily slow rate of evolution.

A more robust way to conduct this process is to impose the constraint that the program must be executable. Thus, one might insist that the source code of a program be *syntactically* correct but not place any limits whatsoever on its semantics (on what it does). For example, statements in a program can be represented as combinations of functions with various numbers of arguments, and the only requirement for syntactic correctness is that a function have the right number of arguments.[64] Changes to the program can be effected by changing the functions and the specific arguments to the functions. The result, by definition, is a program that is still syntactically correct, still runs, but does not necessarily do what is desirable. A typical initial program is then created by randomly generating a parse tree. A population of such parse trees is then subject to crossovers that exchange different parts of the various parse trees, or mutations that replace one argument or function with a new argument or function.

8.3.1.5 Selection of Primitives

Closely related to the issue of representations is the question of the appropriate semantic primitives (i.e., the smallest meaningful unit that can be changed). For example, in the representation of programs as parse trees, the relevant primitives are functions with arguments, and the efficacy of a genetic algorithm is strongly dependent on the particular set of functions that the evolutionary process can manipulate.

[64]This approach is based on parse trees, a way of representing statements in computer programs. See J.R. Koza, *Genetic Programming: On the Programming of Computers by the Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

To illustrate, any computable function can in principle be built from the appropriate combination of Boolean operators (AND, OR, and NOT). But these functions operate at too low a level to build the kind of hierarchical structures needed to do anything complicated. It is for this reason that high-level programming languages have emerged that are not based on these operators directly. Such languages allow the creation of many other kinds of structure. For example, a program intended to undertake financial analysis might benefit from an operator or function that would allow finding the average stock price for the previous month. If its task were to evolve a program for financial analysis, such functions might be included in the set of primitives from which an evolutionary process might draw.

One important aspect of the evolutionary approach is the ability to evolve new operators or new functions that can be used subsequently. In some instances, new structures can emerge spontaneously that are more or less stable; more frequently, it is possible to insert rules that will prevent such structures from changing. Alternatively, functions can be defined automatically—the environment provides slots for function and the ability to call on those function (even if they are no-ops), and the subsequent evolutionary process fills in those spaces with functions.[65]

8.3.1.6 More Evolutionary Mechanisms

The model described above is a very crude model of evolution, incorporating only a few bare essential features. However, biologists have characterized other features of evolution. Two of the most important with possible application to computing are coevolution and development; these are discussed below. Other aspects of evolution, such as diploid behavior and sexual selection, do not at this stage provide obvious new approaches to computing.

8.3.1.6.1 Coevolution Coevolution refers to the biological phenomenon in which two or more species interact as they evolve. For example, a host may be susceptible to infection by a parasite. The host evolves some defenses against the parasite, which in turn stimulates the parasite to evolve ways in which to penetrate or circumvent those defenses. In coevolution, other species—which are also evolving—constitute part of the environment in which a given species is embedded.

One application of coevolution to evolutionary programming is to allow the evolution of testing data simultaneously with the solution. Doing so enables the program to account for a wider range of input. In this case, one fitness function is required for the program to evaluate how well it performs against a given set of test data, while a different fitness function is needed for the test data to evaluate how well it breaks the program.[66]

8.3.1.6.2 Development Development refers to the phenomenon in which biological complexity is shaped by growth within the organism (what might be called maturation) and the action of environmental forces on the organism. It is very difficult to create significant complexity using genetic mechanisms alone. Thus, one intellectual thrust in evolutionary computation focuses on the creation of developmental mechanisms that can be evolved to better create their own complexity. For example, evolutionary techniques can be used to evolve neural networks (see Section 8.3.3.2). In designing neural networks, the problems involve various issues related to the topology and configuration of the network. However, a grammar can be used to generate structures of interest. (A grammar is a formal system of rules that can be used to generate far larger structures.) Grammars can evolve as well, with the fitness function being the complexity of the structures it can generate.

[65]J.R. Koza, *Genetic Programming, 11: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA, 1994.

[66]D. Hillis, "Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure," *Physica D* 42(1-3):228-234, 1990.

Box 8.2
Genetic Programming in Animation

In the world of computer graphics and animation, it can be difficult to build virtual creatures that behave in a realistic manner and simultaneously remain under the user's direct control. For example, directly controlling the positions and angles of moving objects such as limbs can result in detailed behavioral control, but likely at the expense of achieving physically plausible motions. On the other hand, providing a realistic, physics-based environment in which the relevant dynamics are simulated can result in a higher degree of realism, but will likely make it difficult to achieve the desired behavior, especially as the entities involved become more complex.

One way to manage the complexity of control is to optimize the behavior of the creature against some fitness function. Using evolutionary techniques, it is possible to fabricate creatures that behave realistically without understanding the procedures or parameters used to generate them. Different fitness functions can represent different modes of movement (e.g., swimming, walking, jumping, following a source). This approach forces the user to sacrifice some detailed control, but there is also considerable gain in automating the creation of complexity—and the user still influences the outcome by specifying the fitness function.

For purposes of animation, a creature is determined by its physical morphology (e.g., size, shape, number of legs) and the neural system for controlling the relevant muscle forces (the neural system involves sensors that tell the creature about its immediate environment, effectors that cause motion [analogous to muscles], and neurons that retain some memory of its previous states). Both morphology and neural system can be evolved, resulting in a succession of increasingly "fit" creatures that move realistically in a given mode.

In Sims' work, a developmental process was used to generate the creatures and their control systems. The use of such a process allowed similar components, including their local neural circuitry, to be defined once and then replicated, instead of requiring each to be separately specified. Thus, a coded representation—a genotype—of a creature was established that uniquely defined the phenotype of that creature—its morphology and neural system. By evolving the genotype, different phenotypes emerged.

SOURCE: Adapted from K. Sims, "Evolving Virtual Creatures," *Computer Graphics*, Annual Conference Series (SIGGRAPH '94 Proceedings), July 1994, pp. 15-22.

In this case, the goal is to evolve a neural network that has the potential to learn things, rather than evolving the things themselves that are the object of learning. In the case of a robotic brain, it is too difficult to anticipate all of the possibilities that might face the robot, and thus it is impossible to develop a fitness function that fully reflects this diversity. By giving the brain the ability to learn and reason, one can circumvent this difficulty, and as long as one can develop a fitness function for how well the brain has learned over some period, evolutionary techniques can be used to evolve a robotic brain. (Note that the indirect nature of this approach makes it doubly difficult to understand what is going on.)

An example of such work is that of Sims (Box 8.2).

8.3.1.7 Behavior of Evolutionary Processes

Today, those working in evolutionary computation are not able to predict, in general, how long it will take to evolve some desired solution or determine a priori how large an initial population size should be, how rapidly mutations should occur, or how often genetic crossovers should take place. Obviously, all of these parameters have some potential impact on the rate of evolution and how effective a solution might be. Yet how they should be set and their possible relationship to the nature of a given problem are, in general, not known, although some intuitions exist in this area.

For example, variation in a species results from mutations (involving random changes to a genome) and crossovers (involving exchanges of different parts of existing genomes). One hypothesis is that crossovers result in changes that are much more rapid than those driven by mutation. The argument in favor of this is that genomic exchange is in some sense enabling an organism to build on stable substructures. On the other hand, it may be that evolutionary solutions cannot make good use of existing substructures or that crossover is incapable of integrating existing substructures.

If it is true that evolutionary change is more rapid with crossovers than with mutations, this suggests that programs designed to evolve genetic programs may wish to emphasize crossover in their processes for introducing variation.

8.3.2 Robotics 3: Energy and Compliance Management

Biological systems provide an existence proof that self-effected motion is possible. Furthermore, compared to the locomotion made possible by human engineering, biological mechanisms capable of locomotion appear to be energetically efficient, possible in a wide variety of physical environments, and often small in size.

Given these characteristics, it is not unreasonable to ask what lessons biology might hold for the design of engineered systems for locomotion. For example, one reason that biological systems are energetically efficient is that they are not rigid, but rather compliant, and often have mechanisms for energy recovery. That is, these mechanisms store kinetic energy that might otherwise be dissipated, much as a braking electric car can store in batteries the kinetic energy associated with slowing down. A kangaroo employs such a mechanism in its tail, which acts as a spring that compresses as the kangaroo lands from one jump and then assists the kangaroo in pushing off for the next jump. Full has argued that leg locomotion can be described as a point mass attached to a spring and finds that the ratio of relative leg stiffness[67] to body mass is more or less constant across legged animals spanning a wide range of size.[68] In this context, leg musculature functions not just as a source of power but also as an actuator, a springy "strut" that participates in energy absorption, storage, and return.

A second example is that many-legged animals demonstrate an inherent dynamic stability. Contrary to expectations that locomotion would require complex neural control feedback mechanisms, the structure of the leg itself and its inherent multifunctionality provide a key aspect of the control of the system and the combination of stability and forward momentum needed for locomotion. Indeed, analysis of many-legged animals reveals that this inherent stability arises from the production of large lateral and opposing leg forces when the legs are moving. Modeling these forces as a spring between opposing legs reveals that the system is highly stable against perturbations—and the leg assembly is capable of stabilizing itself without any equivalent of neural reflexes at all. Thus, the animal does not need to devote expensive neurological processing to the supervision of locomotive tasks.

Raibert was one of the pioneers of robotics engineering based on physics-inspired control laws—one for height, one for pitch, and one for speed. A fundamental insight was that running animals make use of dynamic stability—a running animal moving forward is out of balance, but legs move forward in rhythm to break its fall. To model this phenomenon, a one-legged "animal" (the "Planar One-legged Hopper") was created. It consisted of a mechanized pogo stick with a three-part control system—one controlling forward running speed, one controlling body attitude, and one controlling hopping height. Stepping motion was not programmed explicitly, but rather emerged under the constraints of balance

[67]Relative leg stiffness is the weight-normalized, size-normalized spring constant of the leg.

[68]R. Blickhan and R.J. Full, "Similarity in Multilegged Locomotion: Bouncing Like a Monopode," *Journal of Comparative Physiology* 173:509-517, 1993; T.M. Kubow and R.J. Full, "The Role of the Mechanical System in Control: A Hypothesis of Self-stabilization in Hexapedal Runners," *Philosophical Transactions of the Royal Society of London B* 354:849-862, 1999; A. Altendorfer et al., "RHex: A Biologically Inspired Hexapod Runner," *Journal of Autonomous Robots* 11:207-213, 2002.

and controlled travel.[69] With this basic unit, a two-legged running animal (the Planar Biped) could be modeled as a body with two pogo sticks working 180° out of phase.[70] A four-legged animal could consist of two two-legged pairs working in opposition (left front and right rear, for example).[71]

Since Raibert's pioneering work, these insights have been applicable to the design of other artificial legged locomotion devices. For example, an autonomous hexapod named "RHex" has a motor associated with each leg, each of which is springy and is able to turn on its central axis. This design enables RHex to have self-correcting reflexes that enable it to respond to obstacles without computational control. Another family of six-legged robots, called the SPRAWL family, is cockroaches. Each leg, driven by a piston, acts as a spring that enables SPRAWL robots to bounce over objects in their path without feedback from the environment. Analysis of the force pattern exerted by the legs closely matches that exerted by a running cockroach.

Other robots are intended to manipulate objects into precise orientations. The traditional way to build such robots is to build them rigidly, with limb motion effected through motors and gear assemblies to increase torque. However, gear assemblies are inherently imprecise, because their very motion requires some degree of play where the gears meet (i.e., some nonzero compliance). In practice, the effect of compliance in the gears introduces a noise function that greatly complicates the prediction of how a limb will move given a certain motor input, and puts limits on the precision with which the final orientation can be known.

One solution to this problem is to use "direct-drive" motors placed at every joint, thus eliminating the gears entirely.[72] Another solution is based on the deliberate introduction of compliance into a gear assembly. This solution is based on the observation that humans can effect precise positioning without precision in their joints. In particular, natural joints are often based on ball-and-socket mechanisms even when they are intended to exhibit 1 degree of freedom. Soft tissue around and in the ball joint introduces compressive compliance in the joint, allowing it to absorb impact and automatically maintain a degree of tightness in the joint.

In the robot context, Pratt et al. inserted a spring mechanism into a limb joint so that the response lags the input.[73] This spring adds a large but known compliance in series into the joint (so-called series elasticity) that is much larger than the unknown compliance of the gears; thus, the gear compliance can safely be ignored in the prediction of final position. Entirely apart from the increased ease of prediction, the introduction of series elasticity enables a local response to any sudden changes in loading—during which time the motors involved can build up torque to handle that load. Other benefits include shock tolerance, lower reflected inertia, more accurate and stable force control, less damage during inadvertent contact, and energy storage.

8.3.3 Neuroscience and Computing

Natural brains demonstrate an alternative to the traditional von Neumann computing architecture (i.e., a fully serial information processor); thus, it is natural to consider possible lessons of neuroscience for computer design. These lessons occur at varying levels of detail.

[69]See http://www.ai.mit.edu/projects/leglab/robots/2D_hopper/2D_hopper.html; see also M.H. Raibert and H.B. Brown, Jr., "Experiments in Balance with a 2D One-legged Hopping Machine," *ASME Journal of Dynamic Systems, Measurement, and Control* 106:75-81, 1984.

[70]See http://www.ai.mit.edu/projects/leglab/robots/2D_biped/2D_biped.html; see also J. Hodgins and M.H. Raibert, "Planar Biped Goes Head Over Heels," Proceedings ASME Winter Annual Meeting, Boston, December 1987.

[71]See http://www.ai.mit.edu/projects/leglab/robots/quadruped/quadruped.html; see also M.H. Raibert, "Four-legged Running with One-legged Algorithms," pp. 311-315 in *Second International Symposium on Robotics Research*, H. Hanafusa and H. Inoue, eds., MIT Press, Cambridge, MA, 1985.

[72]H. Asada and T. Kanade, "Design of a Direct-Drive Mechanical Arm," *ASME Journal of Vibration, Stress, and Reliability in Design* 105(3):312-316, 1983.

[73]G.A. Pratt, M.M. Williamson, P. Dillworth, J. Pratt, K. Ulland, and A. Wright, "Stiffness Isn't Everything," preprints of the Fourth International Symposium on Experimental Robotics, ISER '95, Stanford, CA, June 30-July 2, 1995.

8.3.3.1 Neuroscience and Architecture in Broad Strokes

The most general lesson is that much of human cognition depends on the ability to ignore most of the information made available by the senses.[74] That is, a very high fraction of the raw information that is accessible through sight, sound, and so on does not participate directly in the human's cognitive processes. Human and mammalian cognition is based on an architecture that involves a flexible, but low-capacity, working memory and attentional selection mechanisms that place events and objects into working memory where they become available for cognitive processing.[75]

This approach of selective attention stands in sharp contrast to traditional algorithms that are designed with the goal of seeking optimal solutions and based on the use of as much information about the problem domain as possible. The architecture of biological computation has generally evolved with a different purpose—the adequate management of a complex, changing, and potentially dangerous environment in real time (where "adequate" means "provides for survival").

This architecture is based on a two-track processing arrangement—a very flexible, albeit slow system that implements consciousness, awareness, and cognition but attends to only few things, and a large number of online, fast-acting, sensory-motor systems that bypass attention and awareness (e.g., eye movements, head and hand movements, posture adjustments, and other reflex and reflex-like responses).

Koch et al. have investigated the utility of such a strategy in multiple contexts: (1) a saliency-based visual attention mechanism that selects highly "salient" location in natural images for further processing;[76] (2) a competitive, two-person video game in which an algorithm that focuses on a restricted portion of the playing field outperforms an "optimal" player when a temporal limitation is imposed on the duration of each move;[77] and (3) an algorithm that rapidly solves the NP-complete bin-packing problem under most conditions.[78]

8.3.3.2 Neural Networks

Biology affords an alternative computing model that (1) appears well suited for many ill-posed problems constrained by uncertainty, which is the problem set for which digital machines to date have been reasonably ineffective; and (2) provides an existence proof that slow and noisy circuits can undertake very rapid computations of a certain class. Furthermore, it provides huge numbers of working examples. Although the mechanisms underlying nerve tissue computation are not well understood despite many decades of study, the fact remains that biology has found incredibly good solutions to many engineering problems, and these approaches may well serve to inform practical solutions for engineering problems posed by human beings. Indeed, although biological tissue is not naturally suited for information processing as understood in traditional terms, the fact that biological tissue can do information processing suggests that the underlying architectural principles must be powerful indeed.

Neural networks are among the most successful of biology-inspired computational systems and are modeled on the massively parallel architecture of the brain—and on the brain's inherent ability to learn

[74] C. Koch, "What Can Neurobiology Teach Computer Engineers?" January 31, 2001, unpublished paper, available at http://www7.nationalacademies.org/compbio_wrkshps/Christof_Koch_Position_Paper.doc.

[75] F. Crick and C. Koch, "Consciousness and Neuroscience," *Cerebral Cortex* 8(2):97-107, 1998.

[76] F. Crick and C. Koch, "Consciousness and Neuroscience," *Cerebral Cortex* 8(2):97-107, 1998; L. Itti and C. Koch, "A Saliency-based Search Mechanism for Overt and Covert Shifts of Visual Attention," *Vision Research* 40(10-12):1489-1506, 2000; L. Itti and C. Koch, "Target Detection Using Saliency-based Attention," *Search and Target Acquisition*, RTO Meeting Proceedings 45, NATO, RTO-MP-45, 2000; L. Itti, C. Koch, and E. Niebur, "A Model of Saliency-based Visual Attention for Rapid Scene Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 20:1254-1259, 1998.

[77] J.G. Billock, "Attentional Control of Complex Systems," Ph.D. Thesis, 2001, available at http://sunoptics.caltech.edu/~billgr/thesis/thesiscolor.pdf.

[78] J.G. Billock, D. Psaltis, and C. Koch, "The Match Fit Algorithm: A Testbed for the Computational Motivation of Attention," *International Conference on Computational Science* 2: 208-216, 2001.

from experience.[79] A neural network is a network of nodes and links.[80] The nodes, or units, are very simple processors that correspond to neurons—the brain's electrically active cells—and are usually organized in layers, while the links, or connections, are node-to-node data channels that correspond to synapses—the junctions that convey nerve impulses from one neuron to the next. Each node has an activation level that corresponds to a neuron's rate of firing off nerve impulses, while each link has a numeric weight that corresponds to the strength or efficiency of a synapse.

Digital "activation energy" patterns are presented to the network via the "input layer."[81] From the input layer, the activation surges through the various intermediate layers automatically, with the flow being shaped and channeled by the connection strengths in much the same way that the flow of nerve impulses in the brain is shaped by synapses. Once everything has settled down, the answer can be read out from the pattern of activation on a set of designated output nodes in the final layer.

This computation-by-network architecture is where parallelism is relevant:[82] all of the nodes are active at once, and the activation can travel on any number of paths simultaneously. It is also the basis of the system's ability to learn: since the flow of activation (and, thus, the computation) is shaped by the connection weights, it can be *re*shaped by changing the weights according to some form of learning rule. How the connection weights are modified in response to the input patterns is the content of the learning rule. This seems similar in some ways to what happens in the cerebral cortex, where knowledge and experience are encoded as subtle changes in the synaptic strengths. Likewise in a neural network: with very few exceptions, it will always contain some sort of built-in mechanism that can adjust the weights to improve its performance.

These brain-like characteristics give neural networks some decided advantages over traditional algorithms in certain contexts and problem types. Because they *can* learn, for example, the networks can be trained to recognize patterns and compute functions for which no rigorous algorithms are known, simply by being shown examples. ("This is a letter *B*: **B**. So is this: B.") Often, in fact, they can generalize from the training examples well enough to recognize patterns they've never seen before. And their parallel architecture helps them keep on doing so even in the face of noisy or incomplete data or, for that matter, faulty components. The multiple data streams can do a lot to compensate for whatever is missing.

Training a neural network generally involves the use of a large number of individual runs to determine the best solution (i.e., a specific set of connection weights that enables the network to do its job).[83] Most learning rules have a parameter that controls the rate of convergence between the current solution and the global minimum and another that controls the degree to which the network will ignore local minima. Once the network is trained to demonstrate satisfactory performance, it can be presented with other data.[84] With new data, the network no longer invokes the learning rule, and the connection weights remain constant.

[79]Note that neural networks are only one approach to the general problem of machine learning. A second general approach involves what is called statistical learning techniques, so called because they are techniques for the estimation of unknown probabilistic distributions based on data. These techniques have not, as a rule, been derived from the consideration of biological systems.

[80]Useful online tutorials can be found at http://neuralnetworks.ai-depot.com/3-Minutes/ and http://www.colinfahey.com/2003apr20_neuron/2003apr20_neuron.htm.

[81]Some of this discussion is adapted from http://www.cs.wisc.edu/~bolo/shipyard/neural/neural.html.

[82]Note, however, that this does not represent parallelism on the scale of the brain, where the neurons are numbered in the hundreds of billions, if not trillions. The number of units in a neural network is more likely to be measured in the dozens. In practice, moreover, these networks are usually simulated on ordinary, serial computers—although for specific applications they can also be implemented as specialized microchips. (See the online tutorial at http://www.particle.kth.se/~lindsey/HardwareNNWCourse/home.html.) Still, the parallelism is there in principle.

[83]Some of this is adapted from http://www.cs.wisc.edu/~bolo/shipyard/neural/neural.html.

[84]Note that it is possible to "overtrain" a neural network, which means that the network cannot respond properly to anything but the training data. (This might correspond to rote memorization.) Obviously, such a network is not particularly useful.

Neural networks are most useful for problems that are not amenable to computational approaches and are constrained by strict assumptions of normality, linearity, variable independence, and so on.[85] That is, they work well in classifying objects, capturing associations, and discovering regularities within a set of patterns where the volume, number of variables, or diversity of the data is very great; when the relationships between variables are vaguely understood or the relationships are difficult to describe adequately with conventional approaches; or when the problems in question are ill-posed and involve high degrees of uncertainty.[86] In addition, they are well suited for problems that are subject to distortions in the input data.

Neural networks have been applied to a large number of real-world problems of high complexity, including the following.[87]

• *Optical character recognition*. Commercial OCR (optical character recognition) software packages have incorporated neural network technology since the mid-1980s, when it significantly increased their ability to recognize unfamiliar fonts and noisy, degraded documents such as faxes.[88] Today, OCR systems typically use a mix of neural network and rule-based approaches.

• *Finance and marketing*. Neural networks' ability to detect unanticipated patterns has made them a favored tool for analyzing market trends, predicting risky loans, detecting credit card fraud, managing risk, and many other such tasks in the financial sector.[89]

• *Security and law enforcement*. Neural networks' pattern-detection ability has likewise made them a useful tool for fingerprint matching, face identification, and surveillance applications.[90]

• *Robot navigation*. Neural networks' ability to extract relevant features from noisy sensor data can help autonomous robots do a better job of avoiding obstacles.[91]

• *Detection of medical phenomena*. A variety of health-related indices (e.g., a combination of heart rate, levels of various substances in the blood, respiration rate) can be monitored. The onset of a particular medical condition could be associated with a very complex (e.g., nonlinear and interactive) combination of changes on a subset of the variables being monitored. Neural networks have been used to recognize this predictive pattern so that the appropriate treatment can be prescribed.

• *Stock market prediction*. Fluctuation of stock prices and stock indices is another example of a complex, multidimensional, but in some circumstances at least partially deterministic phenomenon. Neural networks are being used by many technical analysts to make predictions about stock prices based on a large number of factors such as past performance of other stocks and various economic indicators.

• *Credit assignment*. A variety of pieces of information are usually known about an applicant for a loan. For instance, the applicant's age, education, occupation, and many other facts may be available. After training a neural network on historical data, neural network analysis can identify the most relevant characteristics and use them to classify applicants as good or bad credit risks.

[85]This material adapted from http://cfei.geomatics.ucalgary.ca/matlab/ann.html.

[86]See http://www.cs.wisc.edu/~bolo/shipyard/neural/neural.html.

[87]See http://www.emsl.pnl.gov:2080/proj/neuron/neural/what.html; see also http://neuralnetworks.ai-depot.com/Applications.html. Examples in the list below for the topics "detection of medical phenomena" through "engine management" are taken from http://www.statsoftinc.com/textbook/stneunet.html#apps.

[88]See http://www.scansoft.com/omnipage/ocr/. At the time, the state of the art in commercial OCR software was the rule-based approach, in which a system broke each character image into simple features and then identified the letters by reasoning about curves, lines, and such. This approach worked well—but only if the fonts were known and the text was very clean.

[89]See http://neuralnetworks.ai-depot.com/Applications.html; see also http://www.nd.com/ and http://www.walkrich.com/value_investing/howdo.htm.

[90]See http://www.neurodynamics.com/.

[91]See http://ai-depot.com/BotNavigation/Obstacle-Introduction.html.

• *Monitoring the condition of machinery*. Neural networks can be instrumental in cutting costs by bringing additional expertise to scheduling the preventive maintenance of machines. A neural network can be trained to distinguish between the sounds a machine makes when it is running normally ("false alarms") versus those it makes when it is on the verge of a problem. After this training period, the expertise of the network can be used to warn a technician of an upcoming breakdown, before it occurs and causes costly unforeseen "downtime."

• *Engine management*. Neural networks have been used to analyze the input of sensors from an engine. The neural network controls the various parameters within which the engine functions, in order to achieve a particular goal, such as minimizing fuel consumption.

8.3.3.3 Neurally Inspired Sensors

One of the first attempts to draw on the principles underlying biological sensors occurred in the mid-1980s, when researchers such as Carver Mead and his coworkers at Caltech made their first attempts to create artificial retinas using VLSI technology,[92] with hoped-for applications that ranged from artificial eyes for the blind to better sensors for robots. A second, more recent example of a neurally inspired sensor is the computational sensor of Brajovic and Kanade.[93] Many approaches toward improving machine vision have been based on better cameras with higher resolution and sensitivity, new sensors such as uncooled infrared cameras, and new recognition algorithms. But standard vision systems typically have high latency (a long time between registration of the image on the vision system's sensors and image recognition), induced by the requirements of transferring large amounts of data from the sensor to the processor and processing those large amounts of data quickly. In addition, latency increases more or less linearly with image size. Standard vision systems can also be very sensitive to small details in the appearance of an object in sensor images. A number of processor-based algorithms have been developed that adjust for such variations, but they are often complex and ad hoc, and hence unreliable.

The computational sensor approach borrows biological architectural principles to use low-latency processing and top-down sensory adaptation as techniques for speeding up vision processes. Computational sensors are (usually) VLSI circuits that include on-chip processing elements tightly coupled with on-chip sensors, exploit unique optical design or geometrical arrangement of elements, and use the physics of the underlying material for computation. The integration of sensor and processor elements on a VLSI chip enables latency to be reduced by a considerable factor and provides opportunities for fast processor-sensor feedback in service of top-down adaptation—and computational sensors have produced an order-of-magnitude improvement in sensing and information processing itself, such as range sensing, sorting, high-dynamic range imaging, and display.

8.3.4 Ant Algorithms

Ant colonies depend on workers that can collectively build nests, find food, and carry out a multitude of other complex tasks while having little or no intelligence of their own. Further, they must do so without the benefit of a leader to organize their efforts. They also continue to do so even in the face of outside disruptions, or the failure and death of individual members, thereby exhibiting a high degree of flexibility and robustness.

[92]M.A. Sivilotti, M.A. Mahowald, and C. Mead, "Real-time Visual Computations Using Analog CMOS Processing Arrays," pp. 295-312 in *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*, P. Losleben, ed., MIT Press, Cambridge, MA, 1987.

[93]V. Brajovic, "Computational Sensor for Global Operations in Vision," Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1996.

8.3.4.1 Ant Colony Optimization

Entomologists have devoted a great deal of research to figuring out how the social insects achieve these feats.[94] Their answers, in turn, have led computer scientists to devise a variety of "ant algorithms," all of which attempt to capture some of those same qualities of bottom-up self-organization, flexibility, and robustness.[95] Ant algorithms are an example of agent-based models—a broad class of simulations that began to emerge in the early 1990s as researchers tried to model complex adaptive systems on a computer. The idea was to represent different agents with variables that weren't just numbers, as they would be in conventional econometric models, but complex data structures that could respond and adapt to one another—rather like agents in the real world. (In practice, each agent could be modeled as an expert system, a neural network, or any number of other ways.)

The first ant-based optimization—the Ant Colony Optimization algorithm—was created in the early 1990s.[96] The algorithm is based on observations of ant foraging, something that ants do with high efficiency. Imagine that worker ants wandering far from the nest come across a rich food source. Each ant carrying food back to the nest marks her trail by laying pheromone on the ground. When another randomly moving ant encounters this previously marked trail, it will follow it with high probability and reinforce the trail with its own pheromone. This behavior is thus characterized by a positive feedback loop in which the probability with which an ant chooses a given trail increases with the number of ants that previously chose the same trail.

Because the first ant to reach the nest will be the one whose path just happens to be the shortest, there will be a period of time during which the shortest path is the only path to the nest. This fact provides a "seed" around which further pheromone depositions can occur and collectively converge on a path that is one of the shortest possible.

The paradigmatic application of this algorithm is the Traveling Salesman Problem. A salesman is assigned to visit a specified list of cities, going through each of them once and only once before returning to his starting point. In what sequence should he visit them so as to minimize his total distance?

What makes the Traveling Salesman Problem difficult is that there seems to be no guaranteed way of finding the absolute shortest path other than to check every possible sequence, and the number of such sequences grows explosively as the number of cities increases, quickly outstripping the computational ability of any computer imaginable.[97] As a result, practical programmers have had to give up on

[94]See, for example, E.O. Wilson and B. Hölldobler, *The Ants,* Belknap Press of Harvard University Press, Cambridge, MA, 1990.

[95]Overviews can be found in E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, 1999; E. Bonabeau, "Swarm Intelligence," presented at the O'Reilly Emerging Technology Conference, available at http://conferences.oreillynet.com/presentations/et2003/Bonabeau_eric.ppt; and E. Bonabeau and G. Theraulez, "Swarm Smarts," *Scientific American* 282(3):72-79, 2000.

[96]M. Dorigo, "Optimization, Learning, and Natural Algorithms," Ph.D. Dissertation, Politecnico di Milano, Italy, 1992; M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: An Autocatalytic Optimizing Process," Technical Report No. 91-016 Revised, Politecnico di Milano, Italy, 1991; M. Dorigo, V. Maniezzo, and A. Colorni, "Positive Feedback as a Search Strategy," Technical Report No. 91-016, Politecnico di Milano, Italy, 1991 (later published as M. Dorigo et al., "The Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B* 26(1):29-41, 1996, available at ftp://iridia.ulb.ac.be/pub/mdorigo/journals/IJ.10-SMC96.pdf.); M. Dorigo, T. Stützle, and G. Di Caro, eds., *Future Generation Computer Systems* (Special Issues on Ant Algorithms) 16(8), 2000. Dorigo maintains a Web page on ant colony optimization, including an extensive bibliography (with many papers downloadable), plus links to tutorials and software, available at http://iridia.ulb.ac.be/~mdorigo/ACO/about.html.

[97]If there are N cities in the list, then the number of possible routes is on the order of $N!$—that is, $N \times (N - 1) \times (N - 2). \ldots \times 2 \times 1$. (There are N choices of a place to start, $N - 1$ choices of a city to visit next, $N - 2$ choices to visit after that, and so on.) This is nothing much to worry about for small numbers: 10 cities yield only 10! = 3.628 million paths, which a personal computer could examine fairly quickly, but 20 cities would yield about 2.4×10^{18} paths—a (very fast) computer that examined one path per nanosecond would take more than 77 years to get through all of them; and 30 cities ($30! = 2.65 \times 10^{32}$) would keep that same computer busy for 8 quadrillion years. In computer science, this is a classic example of an NP-complete problem. An NP-complete problem is both NP (i.e., verifiable in nondeterministic polynomial time) and NP-hard (any other NP problem can be translated into this problem). In an NP-complete problem, the number of computations required to solve it grows faster than any power of its size. ("Verifiable in nondeterministic polynomial time" means that a proposed solution to this problem can be verified in polynomial time on a computer that can execute different instructions depending on its input. Polynomial time means a time that is proportional to some power of the problem's size.)

finding the *best* solution to the Traveling Salesman Problem and its relatives, and instead look for algorithms that find an *acceptable* solution in an acceptable amount of time. Many such algorithms have been developed over the years, and the Ant Colony Optimization algorithm has proved to rank among the best—especially after Dorigo and his colleagues introduced several refinements during the 1990s to improve its scaling behavior.[98]

Variations of the algorithm have also been developed for practical applications such as vehicle routing, scheduling, routing of traffic through a data network, or the design of connections between components on a microchip, and the scheduling of special orders in a factory.[99] The technique is particularly useful in such cases because it allows for very rapid *re*routing in the face of unexpected disruptions in the network. Among the successful commercial applications are plant scheduling for the consumer products giant Unilever; truck routing for the Italian oil company Pina Petroli; supply chain optimization and control for the French industrial gas supplier Air Liquide; and network routing for British Telecom, France Telecom, and MCI.[100]

8.3.4.2 Other Ant Algorithms

Ant algorithms are based on two essential principles: (1) self-organization, in which global behavior arises from a myriad of low-level interactions, and (2) stigmergy, in which the individuals interact with one another indirectly using the environment as an intermediary.[101] That is, one individual changes its surroundings (e.g., by laying a pheromone trail), and other individuals then react to those changes at a later time. As researchers have looked to other ant colony behaviors for inspiration, moreover, those same two principles turn up again and again.[102] For example:

- *Sorting behavior*. Certain species of ants apparently have an instinct to keep their surroundings clean; if dead ants are scattered through the nest at random, the workers will immediately begin moving all the corpses into neat little piles (albeit piles in random locations). These ants likewise seem to have an instinct for keeping the brood chambers well organized; if workers are presented with a random jumble of ants-to-be, they will quickly see to it that the eggs and micropupae are in the center, while the larger and more developed pupae and larvae are toward the outside where they have more room. Simulated ants can produce much the same results by following a simple local rule: pick up any item that is isolated—that is, any item that has no others like it in the neighborhood—and drop it whenever many of those items are encountered. Picking things up and then dropping them modifies the environment, while the constant shifting causes the piles and/or broods to self-organize fairly rapidly.

[98]The algorithm and its refinements are discussed at length in Chapter 2 of E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, 1999.

[99]Many of their key papers are available for downloading at M. Dorigo, "Ant Colony Optimization," 2003, available at http://iridia.ulb.ac.be/~mdorigo/ACO/about.html.

[100]E. Bonabeau, "Swarm Intelligence," presented at the O'Reilly Emerging Technology Conference, 2003, April 22-25, 2003, Santa Clara, CA, available at http://conferences.oreillynet.com/presentations/et2003/Bonabeau_eric.ppt.

[101]E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, 1999.

[102]Among the most notable of these investigators have been entomologist Guy Theraulaz of the French National Center for Scientific Research (CNRS) and telecommunications engineer Eric Bonabeau, formally of France Telecom. Bonabeau, in particular, has been among the most active in the promotion and commercialization of ant algorithms, first as head of European operations for the Santa Fe-based BiosGroup and since 2000 as head of his own company, Icosystem, Inc., of Cambridge, Massachusetts. Details of the various ant behaviors under study, and the algorithms drawn from them, can be found in E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, 1999; E. Bonabeau and G. Theraulez, "Swarm Smarts," *Scientific American* 282(3):72-79, 2000; and E. Bonabeau, "Swarm Intelligence," presented at the O'Reilly Emerging Technology Conference, 2003, available at http://conferences.oreillynet.com/presentations/et2003/Bonabeau_eric.ppt.

• *Division of labor*. In order to gather food, maintain the nest, defend against predators, and so on, a colony has to allocate many different tasks among many different ants simultaneously—again, without the benefit of central planning or individual intelligence. In many cases this is done by a physical caste system, so that workers do certain jobs, soldiers do others, and so on. Yet ants will often allocate tasks even within a single caste. A simple mechanism that reproduces this behavior is to give each individual a response threshold for each task: once the stimuli associated with that task pass the threshold—imagine the smell of accumulating garbage—the individual gets to work. The result is that individuals with higher and higher thresholds keep pitching in until the stimuli are under control, leaving everyone else free to engage in tasks for which *they* have low thresholds.

• *Cooperative transport*. If a single ant encounters a food item that's too big for her to carry alone (e.g., a dead cockroach), she will recruit nest mates via pheromones to help. Now, however, without a leader or brains, they somehow have to start pulling in the same direction. A simple, two-part rule that reproduces the observed behavior is (1) if the object is already moving in the direction you're pulling, keep pulling, and (2) it's not moving at all, or is moving in a different direction, reorient yourself at random and start pulling *that* way. The result is a sequence in which the ants start out pulling their burden from every direction at once, to no effect—until suddenly, when enough ants just happen to line up by accident, a kind of phase transition sets in and the load begins to move.

• *Cooperative construction*. Many species of social insects can build structures of astonishing complexity: witness the vast, hexagonal combs of the honeybee or the multilayered, intricately swirling nests of the paper wasp. And yet again, they manage to do so without the benefit of central planning or individual intelligence. One way to account for such behavior in simulated insects is to equip each individual with a collection of local rules: in situation 1, take action A; in situation 2, take action B; and so on. For a wasp carrying a load of wood pulp, say, such a rule might be, "If you're surrounded by three walls, then deposit the pulp." In general, each insect will modify the environment encountered by the others, and the structure will organize itself in much the same way that the proteins comprising a virus particle assemble themselves inside an infected cell.

Ant algorithms are conceptually similar to the particle swarm optimization algorithm described in Section 8.2.1. However, at least in the case of the Ant Colony Optimization algorithm, it is known that ants really use the algorithm described. For this reason, this algorithm was placed in the category of biologically inspired mechanisms (rather than principles).

8.4 BIOLOGY AS PHYSICAL SUBSTRATE FOR COMPUTING

8.4.1 Biomolecular Computing

The idea of constructing computer components from single molecules or atoms is the logical, if distant, end point of the seemingly inexorable miniaturization of chips and has been foreseen at least since Richard Feynman's lecture "There's Plenty of Room at the Bottom" in 1959.[103] Molecular computing would have significant advantages, most obviously minuscule size of the resulting component, but also a potentially low marginal cost per component and extreme energy efficiency. However, the technology for the precision placing of single atoms or molecules on a large scale is still in its infancy.

However, there is a significant shortcut available: to use biological molecules, including DNA, RNA, and various enzymes, as instruments to perform computational tasks. The sophisticated functions of DNA and related molecules, coupled with the existing technological infrastructure for synthesizing, manipulating, and analyzing them found in molecular biology laboratories, make it feasible to employ them as a universal set of computing components. Also, because the code of DNA is essentially

[103]R.P. Feynman, "There's Plenty of Room at the Bottom," American Physical Society, December 29, 1959; available at http://www.zyvex.com/nanotech/feynman.html.

a digital code, particular strands of DNA can be used to code information, and in particular, joinings and other recombinations of these strands can be used to represent putative solutions to certain computational problems.

This idea is known variously as DNA computation, molecular computation, and biomolecular computation (BMC). The use of DNA as a computational system had been discussed theoretically by T. Head in 1987,[104] but the idea leapt into prominence with Len Adleman's publication in 1994 of a working experiment (Box 8.3) that solved a seven-node instance of the Hamiltonian path problem, an NP-complete problem that is a special case of the Traveling Salesman Problem.[105]

8.4.1.1 Description

Early attention has focused on DNA because its properties are extremely attractive as a basis for a computational system. First, it offers a digital abstraction: the value of a piece of DNA can be precisely *and only* A, G, T, or C. This abstraction is of course quite familiar to the digital abstractions of 0 and 1. Second, the Watson-Crick complementarity of the bases (A with T, G with C) allows matching operations, conceptually similar to "if" clauses in programming. Third, DNA's construction as a string allows a number of useful operations such as insertion, concatenation, deletion, and appending. Next, billions of years of evolution have provided a large set of enzymes and other molecules that perform those operations, some in very specific circumstances. Finally, the last few decades of progress in molecular biology have created a laboratory and instrument infrastructure for the manipulation and analysis of DNA, such as the custom synthesis of sequences of DNA, chips that can detect the presence of individual sequences, and techniques such as polymerase chain reaction (PCR) that can amplify existing sequences. Without such an infrastructure (importantly including the existence of a body of trained laboratory technicians), the use of DNA for computation would be entirely theoretical.

Biomolecular computing provides a number of advantages that make it quite attractive as a potential base for computation. Most obvious are its information density, about 10^{21} bits per gram (billions of times more dense than magnetic tape), and its massive parallelism, 10^{15} or 10^{16} operations per second.[106] Less immediately apparent, but of equal potential importance, is its energy efficiency: it uses approximately 10^{-19} joules per operation, close to the information theoretic limit (compared to 10^{-9} joules per operation for silicon).

One class of biomolecular computing generates witness molecules for all possible solutions to a problem and then uses molecular selection to sift out molecules that represent solutions to the problem at hand. This was the basic architecture developed by Adleman (described in Box 8.3), and with an exponential amount of witness material, this approach can theoretically solve NP-complete problems. Short sequences of DNA (or RNA) are used to represent data, and these are combined to form longer strands, each of which represents a potential solution. Obtaining the particular DNA strand that represents the solution is thus based on laboratory processes that extract the proper DNA strand, and these laboratory processes are based on the existence of an algorithm that can distinguish between correct and incorrect solutions.

A further important step was taken in 2001 by Benenson et al., who developed a programmable finite automaton comprising DNA and DNA-manipulating enzymes that solves certain computational problems autonomously.[107] In particular, the automaton's "hardware" consisted of a restriction nu-

[104]T. Head, "Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors," *Bulletin of Mathematical Biology* 49(6):737-759, 1987.

[105]L.M. Adleman, "Molecular Computation of Solutions to Combinatorial Problems," *Science* 266(5187):1021-1024, 1994.

[106]It is only the fact of massive parallelism that makes biological computing at all feasible, because biological switching speeds are diffusion-limited and quite slow.

[107]Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro, "Programmable and Autonomous Computing Machine Made of Biomolecules," *Nature* 414(6862):430-434, 2001.

Box 8.3
Adleman and DNA Computing

Adleman used the tools of molecular biology to solve an instance of the directed Hamiltonian path problem. A small graph was encoded in molecules of DNA, and the "operations" of the computation were performed with standard protocols and enzymes. This experiment demonstrates the feasibility of carrying out computations at the molecular level.

The Hamiltonian path problem is based on finding a special path through an arbitrarily connected set of nodes (i.e., an arbitrary directed graph). (The adjective "directed" means that the connections between nodes are unidirectional, so that a path from A to B does not mean necessarily that another connection from B to A exists.) This path (the Hamiltonian path) is special in the sense that beginning with a specified entering node and ending with a specified exiting node, a continuous path exists that enters and exits every other node once and only once. Hamiltonian paths do not necessarily exist for a given directed graph, and their existence may depend on an appropriate specific choice of entering and exiting nodes.

All known algorithms for determining whether an arbitrary directed graph with designated vertices has a Hamiltonian path exhibit worst-case exponential complexity, which means that there are some directed graphs with a small number of nodes for which this determination would take an impractical amount of computing time.

One method for determining if a Hamiltonian path exists is illustrated in the first column of the table below.

Step	Algorithmic Step	Biological Equivalent
0	Establish directed graph notation as problem representation.	Encode each node and directed node-to-node path as a specific DNA sequence.
1	Generate all possible paths through the graph.	Combine large amounts of these DNA sequences, and with a sufficiently large quantity, the probability that all possible paths will be generated is essentially unity. (In general, these various combinations will be in length several multiples of a single sequence.)
2	Keep only those paths that begin with a specified starting and ending node.	Use polymerase chain reaction (PCR) that amplifies only those molecules encoding paths that begin and end with the specified nodes.
3	If the graph has *n* nodes, then keep only those paths that enter exactly *n* nodes.	Separate only those sequences from step 2 that have the correct length (corresponding to the number of nodes in the graph).
4	Keep only those paths that enter all of the nodes of the graph at least once.	Separate the sequences from step 3 that have a subsequence corresponding to each and every node.
5	If any paths remain, say, "Yes, a Hamiltonian path exists"; otherwise, say "No."	Use PCR amplification on the output of step 4, what remains after step 5 represents the solution to the problem.

SOURCE: Adapted from L.M. Adleman, "Molecular Computation of Solutions to Combinatorial Problems," *Science* 266(5187):1021-1024, 1994.

clease and ligase, while the software and input were encoded by double-stranded DNA. Programming was implemented by choosing appropriate software molecules. The automaton processed the input molecule through a cascade of restriction, hybridization, and ligation cycles, producing a detectable output molecule encoding the automaton's computational result. However, a finite-state automaton is not Turing-complete, and the actual demonstration of a Turing-complete biomolecular machine with a set of primitives sufficient for universal computation has yet to be shown experimentally.[108]

Since Adleman's initial publication, researchers have explored many variants of the basic biological approach. One such variant is the use of RNA, which simplifies the process of removing invalid sequences. In this variant, RNA is used for the solution sequences and DNA is used to represent an element of an invalid solution. Thus, any potential solution that was invalid would be represented by a DNA-RNA hybridized double strand. A single enzyme, ribonuclease H, destroys all DNA-RNA hybridized pairs, leaving only valid solutions. This is significantly simpler than the use of many, potentially noncompatible enzymes necessary to mark and destroy the appropriate DNA-DNA hybrids in the traditional method. (In developing an algorithm based on RNA computing for solving a certain chess problem, Cukras et al.[109] found that although the algorithm was able to recover many more correct solutions than would be expected at random, the persistence of errors continued to present the most significant challenge.)

Other variants of the process seek to automate or simplify the management of stages of the reactions. In the original experiments, the DNA reactions took place in solution in test tubes or other containers, with stages of the process controlled by humans—for example, by introducing new enzymes, changing the temperature (perhaps to break chemical bonds), or mixing DNA solutions. Some of these steps can be automated through the use of laboratory robotics. In some variants, DNA strands are chemically anchored to various types of beads; these beads can be designed with different properties, such as being magnetic or electrically charged, allowing the manipulation of the DNA strands through the application of electromagnetic fields. Another solution is to use microfluidic technologies, which consist of MEMS devices that operate as valves and pumps; a properly designed system of pipettes and microfluidic devices offers significant advantages by automating tasks and reducing the total volume of materials required.[110]

Still another variant is to restrict the chemical operations to a surface, rather than to a three-dimensional volume.[111] In this approach, DNA sequences, perhaps representing all of the solution space of an NP problem, would be chemically attached to a surface. Challenges in this approach include the attachment chemistry, addressing particular strands on the surface, and determining whether chemical attachment interferes with DNA hybridization and enzymatic reactions.

A second class of biomolecular computing begins with an input and a program represented in a molecular form and evolves the program in a number of steps to process the input to produce an output. In this approach, the complexity of the problem does not manifest itself in the number of starting molecules, but rather in the form of the rules provided and the amount of time or number of steps needed to fully evaluate a particular problem and input. For example, in the programmed mutagenesis method, DNA molecules that represent rewrite rules are combined with DNA molecules that encode input data and program. When the combined mixture of these DNA molecules is thermally cycled in the

[108]However, Rothemund has provided a highly detailed description of a Turing-complete DNA computer. See P.W.K. Rothemund, "A DNA and Restriction Enzyme Implementation of Turing Machines," pp. 75-119 in *DNA Based Computers: Proceedings of a DIMACS Workshop,* Vol. 27, R.J. Lipton and E.B. Baum eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Princeton, NJ, 1996.

[109]A.R. Cukras, D. Faulhammer, R.J. Lipton, and L.F. Landweber, "Chess Games: A Model for RNA Based Computation," *Biosystems* 52(1-3):35-45, 1999.

[110]A. Gehani and J.H. Reif, "Micro-Flow Bio-Molecular Computation," *BioSystems* 52(1-3):197-216, October 1999.

[111]L.M. Smith, R.M. Corn, A.E. Condon, M.G. Lagally, A.G. Frutos, Q. Liu, and A.J. Thiel, "A Surface-based Approach to DNA Computation," *Journal of Computational Biology* 5(2):255-267, 1998.

presence of DNA polymerase and DNA ligase, the rewrite rules cause new DNA molecules to be produced that represent intermediate states in a computation. These new DNA molecules can be a very general function of the beginning mixture of DNA molecules, and a DNA encoding has been discovered that permits such a system to theoretically implement arbitrary computation.

8.4.1.2 Potential Application Domains

The field of biomolecular computing is still composed of theory and tentative laboratory steps; we are years away from commercial activity. The results of laboratory experiments are proofs of concept; as yet, no biomolecular computer has outperformed an electronic computer.

Biomolecular computing is, in principle, well suited for problems that involve "brute force" solutions, in which candidate solutions can be tested individually to see if they are correct. As noted above, the main application pursued for the first decade of biomolecular computing work is the exhaustive solution of NP-complete problems. While this has been successful for small numbers of nodes (up to 20), the fact that it requires exponential volumes of DNA most likely limits the further development of NP-solving systems (see below for further discussion).

Biomolecular computation also has potential value in the field of cryptography. For example, DNA, with its incredible information density, could serve as an ideal one-time pad, as a tiny sample could provide petabytes of data suitable for use for encryption (as long as it was suitably random). More generally, biomolecules could serve as components of a larger computational system, possibly serving alongside traditional silicon-based semiconductors. For this, and indeed any biomolecular computing system, a challenge is the transformation of information from digital representation into biomolecules and back again. Traditional molecular biological engineering has provided a number of tools for synthesizing DNA sequences and reading them out; however, these tend to be fairly lengthy processes. Recent advances in DNA chips show the potential for more efficient biodigital interfaces. For example, photosensitive chips will synthesize given sequences of DNA based on optical inputs and, similarly, will produce optical signals in the presence of certain sequences. These optical signals are two-dimensional arrays of intensities that can be read by digital image-processing hardware and software. Other approaches for output include the inclusion of fluorescent materials in the DNA molecules or other additives that can be detected with the use of microscopy.

A potential component role for biomolecules is as memory. Whereas biomolecular computation must compete against rapidly improving and increasingly parallel optoelectronic technologies for computation, biomolecular memory is many orders of magnitude superior to conventional magnetic implementations in terms of density. Although DNA memory is unlikely to be used as the rapid-access read-write memory of modern computers, its density makes it useful for "black-box" applications that write a great deal of data, but read only on rare occasions (a fact that would usually tend to increase the acceptable retrieval time).

One such implementation would use DNA as the storage medium of an associative database. A DNA strand would encode the information of a specific record, with sequences on that strand representing attributes of the record and a unique index. Query strings would be composed of the complement of the desired attribute. Although individual lookups would be slow (limited by the speed of DNA chemistry), the total amount of information stored would be enormous and the queries would execute in parallel over the entire database. In contrast, conventional electronic computer implementations of associative memory require linear time with the size of the database.

Such a DNA database might be most useful as a set of tools to manipulate, retrieve, or analyze existing biological or chemical substances. For example, special-purpose DNA computers might search through databases of genetic material. In this model, a large library of genetic material (perhaps representing DNA sequences of various biological lineages, or of criminals) would be stored in its original DNA form, rather than as an electronic digital representation. Biomolecular computers would generate appropriate strands representing a query (matching a sequence found in a new organism, or at a crime scene) and, in massively parallel fashion, identify potential matches. This idea could even be extended to queries of proteins or chemicals, if the appropriate query strand of DNA can be generated.

A separate approach to biomolecular memory uses changes in the sequence of individual strands to represent bits. Certain enzymes known as site-specific recombinases (SSRs) can (among a set of other potential modifications) reverse the sequence of the bases between two marker sequences; repeated application of such an enzyme would flip the sequence back and forth, representing 0 and 1. In this implementation, a single bit requires a long series of bases; research aims at attaining the far more dense use of single bases as bits (in fact, as two bits, since each base can have four values).

8.4.1.3 Challenges

Biomolecular computing faces some significant challenges to adoption beyond the laboratory. The most cited barrier is the exponential doubling of the volume of DNA required to perform exhaustive search of NP-complete problems, such as done by Adleman (Section 8.4.1.1). That is, while the number of different DNA sequences required grows linearly with the number of directed paths in a graph, the volume of those DNA sequences needed to solve a given problem is exponential in the problem's size (in this case, the number of nodes in the graph). Put differently, for the problems to which DNA computing is applicable, a problem that can be solved in exponential time on silicon-based von Neumann computers is replaced by one that can be solved with exponential increases of mass. It is thus an open question today about what kinds of problems can be solved practically using DNA computing. For example, Hartmanis reports that the amount of DNA necessary to replicate Adleman's experiment for a 200-node problem would exceed the weight of the Earth.[112]

While this is a valid concern, standard computers have been widely accepted despite their inability to solve NP-complete problems in a timely fashion. To the best understanding of computer science today, NP-complete problems are fundamentally challenging, and so it ought to be no surprise that even new models of computation struggle with them. Nevertheless some breakthrough may provide subexponential scaling for biomolecular-based exhaustive search.

A second concern involves the time-consuming and expensive laboratory techniques necessary to set up and read out the answer from an experiment—in essence, the input-output problem for biomolecular computing. While DNA reactions themselves offer staggering parallelism (although in fact they take about an hour), the bottleneck may be the time it takes for trained humans to undertake the experiment. Adleman's experiment required about 7 days of laboratory work. And although DNA synthesis itself is cheap, some of the enzymes used in Adleman's experiments cost 10,000 times as much as gold,[113] suggesting that scaling up significantly may not be feasible on economic grounds.

Related to this is the fact that DNA computation is not error-free. Synthesis of sequences can introduce errors; strands of DNA that are close to being complements—but not quite—may still hybridize; point mutations may occur; sheer chance may allow strands of DNA to escape enzymatic destruction; and so forth. Although comparatively high error rates can be acceptable in laboratory environments, this is far more problematic for computation. The problem can be ameliorated partly by the use of techniques familiar to communications protocols, including error-correcting codes and careful design of the code words used in computation, so as to maximize the information distance between any pair. This last example is a good case of computer science and biological cooperation: the distance between a pair of code words composed of a series of bases is a product of both its information content and its biochemical properties. Word design is currently an active area of DNA computation research.

[112]J. Hartmanis, "On the Weight of Computations," *Bulletin of the European Association for Theoretical Computer Science* 55:136-138, 1995.

[113]A.L. Delcher, L. Hood, and R.M. Karp, "Report on the DNA/Biomolecular Computing Workshop (June 6-7, 1996)," National Science Foundation, NSF 97-168, 1998, available at http://www.nsf.gov/pubs/1998/nsf97168/nsf97168.htm.

A related problem is the lack of programmability of current models. Even if experimental verification of Turing-complete biomolecular computing can be achieved, individual runs must still be carefully tuned to a specific instance of a specific problem, much like the hardwiring of the first generation of electronic computers. Worse yet, the sequences of biomolecules synthesized for a particular biomolecular computation are usually consumed or destroyed during the computation. For a replication of the experiment, even with the same dataset, much of the entire process of setup must be repeated. If a different dataset or a different "program" is run, then other steps must be included, such as designing the set of sequences to be used as "words" of the computation and determining the set of enzymes and concentration levels necessary to correctly identify, mark, destroy, and read out the appropriate strands of nucleic acids. The ability to formulate a problem of any generality in terms that map onto a set of chemical processing lab procedures is likely an essential aspect of DNA computing, but it is not at all clear today how such formulations can occur in general.

Finally, the most significant challenge is the high bar that DNA computation will have to surpass to gain wide acceptance. Moore's law is expected to continue unabated for at least a decade, resulting in petaflop machines by 2015. Additionally, biomolecular computation is not the only radical technique in town; quantum computation, various other applications of nanotechnology, analog computing, and other contenders may turn out to offer more favorable performance, programmability, or convenience.

These challenges are quite significant and possibly decisive. Len Adleman himself was pessimistic about the prospect of general computation in a 2002 paper: "Despite our successes, and those of others, in the absence of technical breakthroughs, optimism regarding the creation of a molecular computer capable of competing with electronic computers on classical computational problems in not warranted."[114] Of course, such breakthroughs may yet occur, and this possibility warrants some level of continued research.

8.4.1.4 Future Directions

While it was DNA's resemblance to the tape of a Turing machine that inspired Adleman to investigate the possibility, this model has not yet been pursued experimentally. Nor is it likely that it would have practical computing utility—a Turing machine is extraordinarily slow even executing simple algorithms.

A very different approach would involve single molecules of DNA (or RNA or another biomolecule) acting as the memory of a single process, while enzymes performed the computation by splicing and copying sequences of bases. Although this has been discussed theoretically, it has not yet been shown in an experiment. This model would be best used for massively parallel applications, since the individual operations on DNA are still quite slow compared to electronic components, but it would offer massive improvements of density and energy efficiency over traditional computers.

In a slightly different approach, enzymes that operate on DNA sequences are used as logic gates, such as XOR, AND, or NOT. DNA strands are data, and the enzymes, by reacting to the presence of certain sequences, modify the DNA or generate new strands. Thus, using fairly traditional digital logic design techniques, assemblies of logic gates can be constructed. The resulting circuits will operate in exactly the same manner as traditional silicon electronic-based circuits, but at the energy efficiency and size of molecules.[115]

Even if it turns out that biomolecular computation is a dead end, the research that went into it will not be for naught: the laboratory techniques, enabling technologies, and deeper understanding of

[114]R.S. Braich, C. Johnson, P.W.K. Rothemund, D. Hwang, N. Chelyapov, and L.M. Adleman, "Solution of a 20-Variable 3-SAT Problem on a DNA Computer," *Science* 296(5567):499-502, 2002.

[115]M.N. Stojanovic, T.E. Mitchell, and D. Stefanovic, "Deoxyribozyme-based Logic Gates," *Journal of the American Chemical Society* 124(14):3555-3561, 2002.

biomolecular processes will be valuable. Already, commercial spinoff technologies are available: based on Adleman's research, a company in Japan developed a way to synthesize 10,000 DNA sequences to rapidly search for the presence of genes related to cancer.[116] Also, biologist Laura Landweber's research into biomolecular computation at Princeton has provided insights for her research on DNA and RNA mechanisms in living organisms. For example, her and Lila Kari's analysis of the DNA manipulations that occur in some protozoa is based on techniques of formal languages from computer science, showing that the cellular operations performed by these protozoa are actually Turing-complete. The use of formal computer science theory, in other words, has proven a useful tool for the analysis of natural genetic processes.

8.4.2 Synthetic Biology

As a field of inquiry, the goal of biology—reductionist or otherwise—has been to catalog the diversity of life and to understand how it came about and how it works. These goals emphasize the importance of observation and understanding. Synthetic biology, in contrast, is a new subfield of biology with different intent: based on biological understanding, synthetic biology seeks to modify living systems and create new ones.

Synthetic biology encompasses a wide variety of projects, definitions, and goals and thus is difficult to define precisely. It usually involves the creation of novel biological functions, such as custom metabolic or genetic networks, novel amino acids and proteins, and even entire cells. For example, a synthetic biology project may seek to modify *Escherichia coli* to fluoresce in the presence of TNT, creating in effect a new organism that can be used for human purposes.[117] In one sense, this is a mirror image of natural selection: adding new features to lineages not through mutation and blind adaptation to an environment, but through planned design and forethought. Synthetic biology shares some similarities with recombinant genetic engineering, a common approach that involves transplanting a gene from one organism into the genome of another. However, synthetic biology does not restrict itself to using actual genes found in organisms; it considers the set of all possible genes. In effect, synthetic biology involves writing DNA, not merely reading it.

One basic motivation of this field is that creating artificial cells, or introducing novel biological functions, challenges our understanding of biology and requires significant new insight. In this view, only by reproducing life can we demonstrate that we fully understand it; this is the ultimate acid test for our theories of biology. It is precisely analogous to early synthetic chemistry: only by the successful synthesis of a substance would a theory of its composition be verified.[118]

More broadly, some synthetic biology researchers see created life as an opportunity to explore wider conceptions of life beyond the examples provided by nature. For example, what are the physical limitations of biological systems?[119] Are other self-replicating molecular information systems possible? Are there general principles of biochemical organization? These inquires may help researchers to understand how life began on Earth, as well as the possibility of life in extraterrestrial environments.[120]

Finally, synthetic biology has the potential to contribute significantly to technology, offering in many ways a new industrial revolution. In this view, chemical synthesis, detection, and modification could all be done by creating a microbe with the desired characteristics. This holds the promise of new methods for energy production, environmental cleanup, pharmaceutical synthesis, pathogen detection

[116]*Business Week*, "Len Adleman: Tapping DNA Power for Computers," January 4, 2002.

[117]L.L. Looger, M.W. Dwyer, J.J. Smith, and H.W. Hellinga, "Computational Design of Receptor and Sensor Proteins with Novel Functions," *Nature* 423(6936):185-190, 2003.

[118]S.A. Benner, "Act Natural," *Nature* 421:118, 2003.

[119]D. Endy, quoted in L. Clark, "Writing DNA: First Synthetic Biology Conference Held at MIT," available at http://web.mit.edu/be/news/synth_bio.htm.

[120]J.W. Szostak, D.P. Bartel, and P.L. Luisi, "Synthesizing Life," *Nature* 409(6818):387-390, 2001.

and neutralization, biomaterials synthesis, or any task that can be done by biochemistry. This is essentially a form of nanotechnology, in which the already existing mechanisms of biology are employed to operate on structures at the molecular scale.

However, all of these goals will require a different set of approaches and techniques than traditional biology or any natural science provides. While synthetic biology employs many of the same techniques and tools as systems biology—simulation, computer models of genetic networks, gene sequencing and identification, massively parallel experiments—it is more of an engineering discipline than a purely natural science.

8.4.2.1 An Engineering Approach to Building Living Systems

Although as a viewpoint it is not shared by all synthetic biology researchers, a common desire is to invent an engineering discipline wherein biological systems are both the raw materials and the desired end products. Engineering—particularly, electronics design—is an appropriate discipline to draw on, because no other design field has experience with constructing systems composed of millions or even billions of components. The engineering design approaches of abstraction, modularity, protocols, and standards are necessary to manage the complexity of the biomolecular reality.

One important piece of establishing an engineering discipline of building living systems is to create a library of well-defined, well-understood parts that can serve as components in larger designs. A team led by Tom Knight and Drew Endy at the Massachusetts Institute of Technology (MIT) have created the MIT Registry of Standard Biological Parts, also known as BioBricks, to meet this need.[121] An entry in the registry is a sequence of DNA that will code for a piece of genetic or metabolic mechanism. Each entry has a set of inputs (given concentrations or transcription rates of certain molecules) and a similar set of outputs.

The goal of such a library is to provide a set of components for would-be synthetic biology designers, where the parts are interchangeable, components can be composed into larger assemblies and easily be shared between separate researchers, and work can build on previous success by incorporating existing components. Taken together, these attributes allow the designers to design in ignorance of the underlying biological complexity.

These BioBricks contain DNA sequences at either end that are recognized by specific restriction enzymes (i.e., enzymes that will cut DNA at a target sequence); thus, by adding the appropriate enzymes, a selected DNA section can be spliced. When two or more BioBricks sequences are ligated together, the same restriction sequences will flank the ends of the DNA sequence, allowing the researcher to treat the composite as a single component. BioBricks are in the early stages of research still, and the final product will likely be substantially different in construction.

8.4.2.2 Cellular Logic Gates

Of particular interest to synthetic biologists are modifications to cellular machinery that simulate the operations of classical electronic logic gates, such as AND, NOT, XOR, and so forth. These are valuable for many reasons, including the fact that that their availability in biological systems would mean that researchers could draw on a wide range of existing design experience from electronic circuits. Such logic gates are especially powerful because they increase the ability of designers to build more sophisticated control and reactivity into engineered biological systems. Finally, it is the hope of some researchers that, just as modern electronic computers are composed of many millions of logical gates, a new generation of biological computers could be composed of logic gates embedded in cells.

[121]T. Knight, "Idempotent Vector Design for Standard Assembly of Biobricks," available at http://docs.syntheticbiology.org/biobricks.pdf.

Researchers have begun to construct cellular logic gates in which signals are represented by protein concentrations rather than electrical voltages, with the intent of developing primitives for digital computing on a biological substrate and control of biological metabolic and genetic networks. In other words, the logic gate is an abstraction of an underlying technology (based on silicon or on cellular biology): once the abstraction is available, the designer can more or less forget about the underlying technology.

A biological logic gate uses intracellular chemical mechanisms, such as the genetic regulatory network, metabolic networks, or signaling systems to organize and control biological processes, just as electronic mechanisms are used to control electronic processes.

Any logic gate is fundamentally nonlinear, in the sense that it must be able to produce two levels of output (zero and one), depending on the input(s), in a manner that is highly insensitive to noise (hence, subsequent computations based on the output of that gate are not sensitive to noise at the input). That is, variations in the input levels that are smaller than the difference between 1 and 0 must not be significant to the output of the gate.

Once a logic gate is created, all of the digital logic design principles and tools developed for use in the electronic domain are in principle applicable to the construction of systems involving cellular logic.

A basic construct in digital logic is the inverting gate. Knight et al.[122] describe a cellular inverter consisting of an "output" protein Z and an "input" protein A that serves as a repressor for Z. Thus, when A is present, the cellular inverter does not produce Z, and when A is not present, the inverter does produce Z. One implementation of this inverter is a genetic unit with a binding site for A (an operator), a site on the DNA at which RNA polymerase binds to start transcription of Z (a promoter), and a structural gene that codes for the production of Z.

Protein Z is produced when RNA polymerase binds to the promoter site. However, if A binds to the operator site, it prevents (represses) the binding of RNA polymerase to the promoter site. Thus, if proteins have a finite lifetime, the concentration of Z varies inversely with the concentration of A. To turn this behavior into digital form, it is necessary for the cellular inverter to provide low gain for concentrations of A that are very high and very low, and high gain for intermediate concentrations of A.

Overall gain can be increased by providing multiple copies of the structural gene to be controlled by a single operator binding site. Where high and low concentrations call for low gain, a combination of multiple steps or associations into a single pathway (e.g., the mitogen-activated protein [MAP]-kinase pathway, which consists of many switches that turn on successively) can be used to generate a much sharper nonlinear response for the system as a whole than can be obtained from a single step.

Once this inverter is available, any logic gate can be constructed from combinations of inverters.[123] For example, a NAND gate can be constructed from two inverters that have different input repressors (e.g., A1 and A2) but the same output protein Z, which will be produced unless both A1 and A2 are present. On the other hand, cellular logic and electronic logic differ in that cellular logic circuits are more inherently asynchronous because signal propagation in cellular logic circuits is based on diffusion of proteins, which makes both synchronization and high speed very hard to achieve. In addition, because these diffusion processes are, by definition, not channeled in the same way that electrical signals are confined to wires, a different protein must be used for each unique signal. Therefore, the number of proteins required to implement a circuit is proportional to the complexity of the circuit. Using different proteins means that their physical and chemical properties are different, thus complicating the design and requiring that explicit steps be taken to ensure that the signal ranges for coupled gates are appropriately matched.

[122]T.F. Knight and G.J. Sussman, "Cellular Gate Technology," *Unconventional Models of Computation*, C. Calude, J. Casti, and M.J. Dinneen, eds., Springer, Auckland, New Zealand, 1998.

[123]In general, the availability of an inverter is not sufficient to compute all Boolean functions—an AND or an OR function is also needed. In this particular case, however, the implementing technology permits inverters to be placed side by side to form NOT-AND (NAND) gates.

Cellular circuits capable of logic operations have been demonstrated. For example, Elowitz and Leibler designed and implemented a three-gene network that produced oscillations in protein concentration.[124] The implemented network worked in only a fraction of the cells but did, in fact, oscillate. Gardner et al. built a genetic latch that acted as a toggle between two different stable states of gene expression.[125] They demonstrated that different implementations of the general designs yielded more or less stable switches with differing variances of concentration in the stable states. While both of these applications demonstrate the ability to design a simple behavior into a cell, they also demonstrate the difficulty in implementing these circuits experimentally and meeting design specifications.

In a step toward clinical application of this type of work,[126] Benenson et al. developed a molecular computer that could sense its immediate environment for the presence of several mRNA species of disease-related genes associated with models of lung and prostate cancer and, upon detecting all of these mRNA species, release a short DNA molecule modeled on an anticancer drug.[127] Benenson et al. suggest that this approach might be applied in vivo to biochemical sensing, genetic engineering, and medical diagnosis and treatment.

8.4.2.3 Broader Views of Synthetic Biology

While cellular logic emphasizes the biological network as a substrate for digital computing, synthetic biology can also use analog computing. To support analog computing, the biomolecular networks involved would be sensitive to small changes in concentrations of substances of interest. For example, a microbe altered by synthetic biology research might fluoresce with an intensity proportional to the concentration of a pollutant. Such analog computing is in one sense closer to the actual functionality of existing biomolecular networks (although of course there are many digital elements in such networks as well), but is more alien to the existing engineering approaches borrowed from electronic systems.

For purposes of understanding existing biology, one approach inspired by synthetic biology is to strip down and clean up genomes for maximal clarity and comprehensibility. For example, Drew Endy's group at MIT is cleaning the genome of the T7 bacteriophage, removing all unnecessary sequences, editing it so that genes are contiguous, and so on.[128] Such an organism would be easier to understand than the wild genotype, although such editing would obscure the evolutionary history of the genome.

While synthetic biology stresses the power of hand-designing biological functions, evolution and selection may have their place. Ron Weiss's group at Princeton University has experimented with using artificial selection as a way to achieve desired behavior.[129] This approach can be combined with engineering approaches, using evolution as a final stage to eliminate unstable or faulty designs.

The most extreme goal of synthetic biology is to generate entirely synthetic living cells. In principle, these cells need have no chemical or structural similarity to natural cells. Indeed, achieving an understanding of the range of potential structures that can be considered living cells will represent a profound step forward in biology. This goal is discussed further in Section 9.3.

---

[124] M.B. Elowitz and S. Leibler, "A Synthetic Oscillatory Network of Transcriptional Regulators," *Nature* 403(6767):335-338, 2000.

[125] T.S. Gardner, C.R. Cantor, and J.J. Collins, "Construction of a Genetic Toggle Switch in *Escherichia coli*," *Nature* 403(6767):339-342, 2000.

[126] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An Autonomous Molecular Computer for Logical Control of Gene Expression," *Nature* 429(6990):423-429, 2004.

[127] In fact, the molecular computer—analogous to a process control computer—is designed to release a suppressor molecule that inhibits action of the drug-like molecule.

[128] W.W. Gibbs, "Synthetic Life," *Scientific American* 290(5):74-81, 2004.

[129] Y. Yokobayashi, C.H. Collins, J.R. Leadbetter, R. Weiss, and F.H. Arnold, "Evolutionary Design of Genetic Circuits and Cell-Cell Communications," *Advances in Complex Systems*, World Scientific, 2003.

8.4.2.4 Applications

While significant from a research view, synthetic biology also has practical applications. A strong driver of this is the rapidly falling cost of custom DNA synthesis. For a few dollars per base pair in 2004, laboratories can synthesize an arbitrary sequence of DNA;[130] these prices are expected to fall by orders of magnitude over the next decade. This not only has enabled research into constructing new genes, but also offers the promise of cost-effective use of synthetic biology for commercial or industrial applications. Once a new lineage is created, of course, organisms can self-replicate in the appropriate environment, implying extremely low marginal cost.

Cells can be abstracted as chemical factories controlled by a host of process control computers. If the programming of these process control computers can be manipulated, or new processes introduced, it is—in principle—possible to co-opt the functional behavior of cells to perform tasks of engineering or industrial interest. Natural biology creates cells that are capable of sensing and actuating functions: cells can generate motion and light, for example, and respond to light or to the presence of chemicals in the environment. Natural cells also produce a variety of enzymes and proteins with a variety of catalytic and structural functions. If logic functions can be realized through cellular engineering, cellular computing offers the promise of a seamlessly integrated approach to process control computing.

Synthetic or modified cells could lead to more rational biosynthesis of a variety of useful organic compounds, including proteins, small molecules, or any substance that is too costly or difficult to synthesize by ordinary bench chemistry. Some of this is already being done by cloning and gene transfection (e.g., in yeast, plants, and many organisms), but synthetic biology would allow finer control, increased accuracy, and the ability to customize such processes in terms of quantity, precise molecular characteristics, and chemical pathways, even when the desired characteristics are not available in nature.

8.4.2.5 Challenges

Synthetic biology brings the techniques and metaphor of electronic design to modify biomolecular networks. However, in many ways, these networks do not behave like electronic networks, and the nature of biological systems provides a number of challenges for synthetic biology researchers in attempting to build reliable and predictable systems.

A key challenge is the stochastic and noisy nature of biological systems, especially at the molecular scale. This noise can lead to random variation in the concentration of molecular species; systems that require a precise concentration will likely work only intermittently. Additionally, as the mechanisms of synthetic biology are embedded in the genome of living creatures, mutation or imperfect replication can alter the inserted gene sequences, possibly disabling them or causing them to operate in unforeseen ways.

Unlike actual electronic systems, the components of biomolecular networks are not connected by physical wires that direct a signal to a precise location; the many molecules that are the inputs and outputs of these processes share a physical space and can commingle throughout the cell. It is therefore difficult to isolate signals and prevent cross-talk, in which signals intended for one recipient are received by another. This physical location sharing also means that it is more difficult to control the timing of the propagation of signals; again, unlike electronics, which typically rely on a clock to precisely synchronize signals, these biomolecular signals are asynchronous and may arrive at varying speeds. Finally, the signals may not arrive, or may arrive in an attenuated fashion.[131]

[130] One firm claims to be able to provide DNA sequences as long as 40,000 base pairs. See http://www.blueheronbio.com/genemaker/synthesis.html. Others suggest that sequences in the 100 base pair range are the longest that can be synthesized today without significant error in most of the resulting strands.

[131] R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali, "Genetic Circuit Building Blocks for Cellular Computation, Communications, and Signal Processing," *Natural Computing* 2:47-84, 2003.

Aside from the technical challenges of achieving the desired results of synthetic biology projects, there are significant concerns about the misuse or unintended consequences of even successful work. Of major concern is the potential negative effect on the environment or the human population if modified or created organisms became unmanaged, through escape from a laboratory, mutation, or any other vector. This is especially a concern for organisms, such as those intended to detect or treat pollutants, that are designed to work in the open environment. Such a release could occur as a result of an accident, in which case the organism would have been intended to be safe but may enter an environment in which it could pose a threat. More worrisome, an organism could be engineered using the techniques of synthetic biology, but with malicious intent, and then released into the environment. The answer to such concerns must include elements of government regulation, public health policy, public safety, and security. Some researchers have suggested that synthetic biology needs an "Asilomar" conference, by analogy to the conference in 1975 that established the ground rules for genetic engineering.[132]

Some technical approaches to answer these concerns are possible, however. These include "bar-coding" engineered organisms, that is, including a defined marker sequence of DNA in their genome (or in every inserted sequence) that uniquely identifies the modification or organism. More ambitiously, modified organisms could be designed to use molecules incompatible with natural metabolic pathways, such as right-handed amino acids or left-handed sugars.[133]

8.4.3 Nanofabrication and DNA Self-Assembly[134]

Nanofabrication draws from many fields, including computer science, biology, materials science, mathematics, chemistry, bioengineering, biochemistry, and biophysics. Nanofabrication seeks to apply modern biotechnological methodologies to produce new materials, analytic devices, self-assembling structures, and computational components from both naturally occurring and artificially synthesized biological molecules such as DNA, RNA, peptide nucleic acids (PNAs), proteins, and enzymes. Examples include the creation of sensors from DNA-binding proteins for the detection of trace amounts of arsenic and lead in ground waters, the development of nonsocial DNA cascade switches that can be used to identify single molecular events, and the fabrication of novel materials with unique optical, electronic, rheological, and selective transport properties.

8.4.3.1 Rationale

Scientists and engineers wish to be able to controllably generate complex two- and three-dimensional structures at scales from 10^{-6} to 10^{-9} meters; the resulting structures could have applications in extremely high-density electronic circuit components, information storage, biomedical devices, or nanoscale machines. Although some techniques exist today for constructing structures at such tiny scales, such as optical lithography or individual atomic placement, in general they have drawbacks of cost, time, or limited feature size.

Biotechnology offers many advantages over such techniques; in particular, the molecular precision and specificity of the enzymatic biochemical pathways employed in biotechnology can often surpass what can be accomplished by other chemical or physical methods. This is especially true in the area of nanoscale self-assembly. Consider the following quote from M.J. Frechet, a chemistry professor at the

[132]D. Ferber, "Synthetic Biology: Microbes Made to Order," *Science* 303(5655):158-161, 2004.

[133]O. Morton, "Life, Reinvented," *Wired* 13.01, 2005.

[134]Section 8.4.3 draws heavily from T.H. LaBean, "Introduction to Self-Assembling DNA Nanostructures for Computation and Nanofabrication," *World Scientific*, CBGI, 2001; E. Winfree, "Algorithmic Self-Assembly of DNA: Theoretical Motivations and 2D Assembly Experiments," *Journal of Biomolecular Structure and Dynamics* 11(2):263-270, 2000; J.H. Reif, T.H. LaBean, and N.C. Seeman, "Challenges and Applications for Self-Assembled DNA Nanostructures," pp. 173-198 in *Proceedings of the Sixth International Workshop on DNA-Based Computers*, A. Condon and G. Rozenberg, eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Springer-Verlag, Berlin, 2001.

University of California, Berkeley, who is a leader in the area of the synthesis and control of molecular architectures on the nanometer scale:[135]

> While most common organic molecules—"small molecules"—have sizes well below one nanometer, macromolecules such as proteins or synthetic polymers have sizes in the nanometer range. Within this size range, it is generally very difficult to control the 3-D structure of the molecules. Nature has learned how to achieve this with proteins and DNA, but most other large synthetic macromolecules have little shape persistence and precise functional group placement is difficult.

It is this fine control of nanoscale architecture exhibited in proteins, membranes, and nucleic acids that researchers hope to harness with these applied biotechnologies, and the goal of research into "self-assembly" is to develop techniques that can create structures at a molecular scale with a minimum of manual intervention.

Self-assembly, also known as bottom-up construction, is a method of fabrication that relies on chemicals forming larger structures without centralized or external control.[136] Because of its ability to run in parallel and at molecular scales, self-assembly is considered to be a potentially important technique for constructing submicron devices such as future electronic circuit components.

Since the role of DNA and related molecules in biology is to generate complicated three-dimensional macromolecules such as proteins, DNA is a natural candidate for a system of self-assembly. Researchers have investigated the potential of using DNA as a medium for self-assembling structures at the nanometer scale. DNA has many characteristics that make it an excellent candidate for creating arbitrary components: its three-dimensional shape is well understood (in contrast to most proteins, which have poorly understood folding behavior); it is a digital, information-encoding molecule, allowing for arbitrary customization of sequence; and it, with a set of easily accessible enzymes, is designed for self-replication. Box 8.4 describes some key enabling technologies for DNA self-assembly.

One important focus of DNA self-assembly research draws on the theory of Wang tiles, a mathematical theory of tiling first laid out in 1961.[137] Wang tiles are polygons with colored edges, and they must be laid out in a pattern such that the edges of any two neighbors are the same color. Later, Berger established three important properties of tiling: the question of whether a given set of tiles could cover an area was undecidable; aperiodic sets of tiles could cover an area; and tiling could simulate a universal Turing machine,[138] and thus was a full computational system.[139]

The core of DNA self-assembly is based on constructing special forms of DNA in which strands cross over between multiple double helices, creating strong two-dimensional structures known as DNA tiles. These tiles can be composed of a variety of combinations of spacing and interconnecting patterns; the most common, called DX and TX tiles, contain two or three double helices (i.e., four or six strands), although other structures are being investigated as well. Ends of the single strands, sequences of unhybridized bases, stick out from the edges of the tile, and are known as "sticky ends" (or "pads") because of their ability to hybridize—stick to—other pads. Pads can be designed to attach to the sticky ends of other tiles. By careful design of the base sequence of these pads, tiles can be designed to connect only with specific other tiles that complement their base sequence.

The congruence between Wang tiles and DNA tiles with sticky ends is straightforward: the sticky ends are designed so that they will bond only to complementary sticky ends on other tiles, just as Wang tiles must be aligned by color of edge. The exciting result of combining Wang tiles with DNA tiles is that DNA tiles have also been shown to be Turing-complete and thus a potential mechanism for computing.

[135]See http://www.cchem.berkeley.edu.

[136]See, for example, G.M. Whitesides et al., "Molecular Self-Assembly and Nanochemistry—A Chemical Strategy for the Synthesis of Nanostructures," *Science* 254(5036):1312-1319, 1991.

[137]H. Wang, "Proving Theorems by Pattern Recognition," *Bell System Technical Journal* 40:1-41, 1961.

[138]A universal Turing machine is an abstract model of computer execution and storage with the ability to perform any computation that any computer can perform.

[139]R. Berger, "The Undecidability of the Domino Problem," *Memoirs of the American Mathematical Society* 66:1-72, 1966.

Box 8.4
Enabling Technologies for DNA Self-replication

DNA Surface Arrays

Current DNA array technologies based on spotting techniques or photolithography extend down to pixel sizes on the order of 1 micron.[1] Examples of these arrays are those produced by Affymetrix and Nanogen.[2] The creation of DNA arrays on the nanometer scale require new types of non-photolithographic fabrication technologies, and a number of methods utilizing scanning probe microscopic techniques and self-assembled systems have been reported.

DNA Microchannels

The separation and analysis of DNA by electrophoresis is one of the driving technologies of the entire genomics area. The miniaturization of these analysis technologies with micron-sized fluidic channels has been vigorously pursued with the end goal of creating "lab on a chip" devices. Examples are the products of Caliper Technologies and Aclara Biosciences.[3] The next generation of these devices will target the manipulation of single DNA molecules through nanometer-sized channels. Attempts to make such channels both lithographically and with carbon nanotubes have been reported.

DNA Attachment and Enzyme Chemistry

Robust attachment of DNA, RNA, and PNA onto surfaces and nanostructures is an absolute necessity for the construction of nanoscale objects—both to planar surfaces and to nanoparticles. The primary strategy is to use modified oligonucleotides (e.g., thiol, amine-containing derivatives) that can be reacted either chemically or enzymatically. The manipulation of DNA sequences by enzymatic activity has the potential to be a very sequence-specific methodology for the fabrication of DNA nanostructures.[4]

DNA-modified Nanoparticles

Nanoscale objects that incorporate DNA molecules have been used successfully to create biosensor materials. In one example, the DNA is attached to a nanometer-sized gold particle, and then the nucleic acid is used to provide biological functionality,while the optical properties of the gold nanoparticles are used to report particle-particle interactions.[5] Semiconductor particles can also be used, and recently the attachment of DNA to dendrimers or polypeptide nanoscale particles has been exploited for both sensing and drug delivery.[6]

DNA Code Design

To successfully self-assemble nucleic acid nanostructures by hybridization, the DNA sequences (often referred to as DNA words) must be "well behaved" (i.e., they must not interact with incorrect sequences). The creation of large sets of well behaved DNA molecules is important not only for DNA materials research by also for large-scale DNA array analysis. An example of the work in this area is the DNA word design by Professor Anne Condon at the University of British Columbia.[7]

DNA and RNA Secondary Structure

The secondary structure of nucleic acid objects beyond simple DNA Watson-Crick duplex formation, whether they are simple single strands of RNA or the complex multiple junctions of Ned Seeman, have to be understood by a combination of experimental methods and computer modeling. The incorporation of nucleic acid structures that include mismatches (e.g., bulges, hairpins) will most likely be an important piece of the self-assembly process of DNA nanoscale objects.[8]

Multistrand DNA Nanostructures and Arrays

The creation of three-dimensional objects with multistrand DNA structures has been pursued for many years by researchers such as Ned Seeman at New York University. Computer scientists such as Erik Winfree at the California Institute of Technology and John Reif at Duke University have been using the assembly of these nanostructures to create mosaics and tile arrays on surfaces. The application of computer science concepts to "program" the self-assembly of materials is the eventual goal. Since single-stranded RNA forms many biologically functional structures, researchers are also pursuing the use of RNA as well as DNA for these self-assembling systems.[9]

[1]A.C. Pease, D. Solas, E.J. Sullivan, M.T. Cronin, C.P. Holmes, and S.P.A. Fodor, "Light-generated Oligonucleotide Arrays for Rapid DNA Sequence Analysis," *Proceedings of the National Academy of Sciences* 91(11):5022-5026, 1994.

[2]See http://www.affymetrix.com and http://www.nanogen.com.

[3]See http://www.caliper.com; and http://www.alcara.com.

[4]A.G. Frutos, A.E. Condon, L.M. Smith, and R.M. Corn, "Enzymatic Ligation Reactions of DNA 'Words' on Surfaces for DNA Computing," *Journal of the American Chemical Society* 120 (40):10277-10282, 1998. Also, Q. Liu, L. Wang. A.G. Frutos, A.E. Condon, R.M. Corn, and L.M. Smith, "DNA Computing on Surfaces," *Nature* 403:175-179, 2000.

[5]C.A. Mirkin, R.L. Letsinger, R.C. Mucic, and J.J. Storhoff, "A DNA-based Method for Rationally Assembling Nanoparticles into Macroscopic Materials," *Nature* 382(6592):607-609, 1996; T.A. Taton, C.A. Mirkin, and R.L. Letsinger, "Scanometric DNA Array Detection with Nanoparticle Probes," *Science* 289(5485):1757-1760, 2000.

[6]F. Zeng and S.C. Zimmerman, "Dendrimers in Supramolecular Chemistry: From Molecular Recognition to Self-Assembly," *Chemical Review* 97(5):1681-1713, 1997; M.S. Shchepinov, K.U. Mir, J.K. Elder, M.D. Frank-Kamenetskii, and E.M. Southern, "Oligonucleotide Dendrimers: Stable Nano-structures," *Nucleic Acids Research* 27(15):3035-3041, 1999.

[7]A. Maranthe, A.E. Condon, and R.M. Corn, "On Combinatorial Word Design," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 54:75-90, 2000.

[8]C. Mao, T. LaBean, J.H. Reif, and N.C. Seeman, "Logical Computation Using Algorithmic Self-Assembly of DNA Triple Crossover Molecules," *Nature* 407(6803):493-496, 2000.

[9]E. Winfree, F. Liu, L.A. Wenzler, and N.C. Seeman, "Design and Self-Assembly of Two-Dimensional DNA Crystals," *Nature* 394(6693):539-544, 1998.

Given a set of tiles with the appropriate pads, any arbitrary pattern of tiles can be created. Simple, periodic patterns have been successfully fabricated and formed from a variety of different DNA tiles,[140] and large superstructures involving these systems and containing tens of thousands of tiles have been observed. However, nonperiodic structures are more generally useful (e.g., for circuit layouts), and larger tile sets with more complicated association rules are currently being developed for the assembly of such patterns.

The design of the pads is a critical element of DNA self-assembly. Since the sticky ends are composed of a sequence of bases, the set of different possible sticky ends is very large. However, there are physical constraints that restrict the sequences chosen; pads and their complements should be sufficiently different from other matched pairs, as to avoid unintended hybridization; they should avoid palindromes, and so on.[141] Most importantly, the entire set of pads must be designed so as to produce the desired overall assembly.

The process of DNA self-assembly requires two steps: the first is the creation of the tiles, by mixing input strands of DNA together; then, the tiles are placed in solution and the temperature is lowered slowly until the tiles' pads connect and the overall structure takes form. This process of annealing can take from several seconds to hours.

[140]C. Mao, "The Emergence of Complexity: Lessons from DNA," *PLoS Biology* 2(12):e431, 2004, available at http://www.plosbiology.org/archive/1545-7885/2/12/pdf/10.1371_journal.pbio.0020431-S.pdf.

[141]T.H. LaBean, "Introduction to Self-Assembling DNA Nanostructures for Computation and Nanofabrication," *Computational Biology and Genome Informatics*, J.T.L. Wang et al., eds., World Scientific, Singapore, 2003.

Once the structure is completed, a number of methods can be used to obtain the output if necessary. The first is to image the resulting structure, for example, with an atomic force microscope or transmission electron microscope. In some cases, the structure by itself is visible; in others, tiles can be made distinguishable by reflectivity or the presence of extra atoms such as gold or fluorescents possibly added to a turn of the strand that extends out of the plane. Second, with the use of certain tiles, a "reporter" strand of DNA can be included in such a way that when all the tiles are connected, the single reporter strand winds through all of them. Once the tiling structure completes assembly, that strand can then be isolated and sequenced by PCR or another technique to determine the ordering of the tiles.

8.4.3.2 Applications

DNA self-assembly has a wide range of potential applications, drawing on its ability to create arbitrary, programmable structures. Self-assembled structures can encode data (especially array data such as images); act as a layout foundation for nanoscale structures such as circuits; work as part of a molecular machine; and perform computations.

Since a tiled assembly can be programmed to form in an arbitrary pattern, it is potentially a useful way to store data or designs. In one dimension, this can be accomplished by synthesizing a sequence of DNA bases that encode the data; then, in the self-assembly step, tiles join to the input strand, extending the encoding into the second dimension. This two-dimensional striped assembly can be inspected visually using microscopy, enabling a useful way to read out data. To store two-dimensional data, the input strand is designed with a number of hairpin turns so that the strand weaves across every other line of the assembly; the tiles then attach between adjacent turns of the input strand. The resulting assembly can encode any two-dimensional pattern, and in principle this approach could be extended to three dimensions.

This approach can also be used to create a foundation for nanometer-scale electronic circuits. For this application, the DNA tiles would contain some extra materials, such as tiny gold beads, possibly in a strand fragment that extended above the plain of the tile. After the tiles have formed the desired configuration, chemical deposition would be used to coat the gold beads, increasing their size, until they merge and form a wire. Box 8.5 describes a fantasy regarding a potential application to circuit fabrication.

DNA has been used as a scaffold for the fabrication of nanoscale devices.[142] In crystalline form, DNA has enabled the precise and closely spaced placement of gold nanoparticles (at distances of 10-20 angstroms). Gold nanoparticles might function as a single-electron storage device for one bit, and other nanoparticles might be able to hold information as well (e.g., in the form of electric charge or spin). At one bit per nanoparticle, the information density would be on the order of 10^{13} to 10^{14} bits per square centimeter.

Computation through self-assembly is an attractive alternative to traditional exhaustive search DNA computation. Although traditional DNA computation, such as performed by Adleman, required a linear number of steps with the input size, in algorithmic self-assembly, the computation occurs in a single step. In current experiments with self-assembly, a series of tiles are provided as input, and computation tiles and output tiles form into position around the input. For example, in an experiment that used DNA tiles to calculate cumulative XOR, input tiles represented the Boolean values of four inputs, while output tiles, designed such that a tile representing the value 0 would connect to two identical inputs, and a tile representing the value of 1 would connect to two dissimilar inputs, formed alongside the input tiles. Then, the reporter strand is ligated, extracted, and amplified to read out the answer.[143]

[142] S. Xiao, F. Liu, A.E. Rosen, J.F. Hainfeld, N.C. Seeman, K. Musier-Forsyth, and R.A. Kiehl, "Assembly of Nanoparticle Arrays by DNA Scaffolding," *Journal of Nanoparticle Research* 4:313-317, 2002.

[143] C. Mao, T.H. LaBean, J.H. Reif, and N.C. Seeman, "Logical Computation Using Algorithmic Self-assembly of DNA Triple-crossover Molecules," *Nature* 407:493-496, 2000.

Box 8.5
A Fantasy of Circuit Fabrication

Consider:

> . . . a fantasy of nanoscale circuit fabrication in a future technology. Imagine a family of primitive molecular-electronic components, such as conductors, diodes, and switches, is available from generic parts suppliers. Perhaps we have bottles of these common components in the freezer. . . . Suppose we have a circuit to implement. The first stage of the construction begins with the circuit and builds a layout incorporating the sizes of the components and the ways they might interact. Next, the layout is analyzed to determine how to construct a scaffold. Each branch is compiled into a collagen strut that links only to its selected targets. The struts are labeled so that they bind only to the appropriate electrical component molecules. For each strut, the DNA sequence to make that kind of strut is assembled, and a protocol is produced to insert the DNA into an appropriate cell. These various custom parts are then synthesized by the transformed cells.
>
> Finally, we create an appropriate mixture of these custom scaffold parts and generic electrical parts. Specially programmed worker cells are added to the mixture to implement the circuit edifice we want. The worker cells have complex programs, developed through amorphous computing technology. The programs control how the workers perform their particular task of assembling the appropriate components in the appropriate patterns. With a bit of sugar (to pay for their labor), the workers construct copies of our circuit we then collect, test, and package for use.

SOURCE: H. Abelson, R. Weiss, D. Allen, D. Coore, C. Hanson, G. Homsy, T.F. Knight, Jr., et al., "Amorphous Computing," *Communications of the ACM* 43(5):74-82, 2000.

This approach has two main drawbacks: the speed of individual assemblies, and the error rate. First, the DNA reactions can take minutes or hours, and so any individual computation by self-assembly will likely be substantially slower than using a traditional computer. The potential for self-assembly is that, like exhaustive DNA computation, it can occur in parallel, with a parallelism factor as high as 10^{18}. In the XOR experiment, researchers observed an error rate of 2 to 5 percent. Certainly, this rate may be lowered as experience is gained in designing laboratory procedures and assembly methods; however, the error rate is likely to remain higher than that for electronic computers. For certain classes of problems, an ultraparallel though unreliable approach may be an effective way to compute a solution.

8.4.3.3 Prospects

So far, DNA self-assembly has been demonstrated successfully in the laboratory, constructing relatively simple patterns (e.g., alternating bands, or the encoding of a binary string) that are visible through microscopy. It has also been used successfully for simple computations such as counting, XOR, and addition.

Moving forward, laboratory techniques must improve in sophistication to handle the more complex assemblies and reactions that will accompany large-scale computations or designs. Along with progress in the lab, further theoretical developments are possible in developing algorithms for constructing arbitrary aperiodic patterns.

Although so far DNA self-assembly has used only naturally occurring variants of DNA, a possible improvement is to employ alternative chemistries, such as peptide nucleic acid, an artificial form of DNA in which the backbone has peptide links in place of the phosphate that occurs in natural DNA.

Also, a wide variety of potential geometries exists for crossover tiles. There have been experiments with a so-called 4×4 tile, where the sticky ends extend at right angles.

DNA also has the property that its length scale can bridge the gap between molecular systems and microelectronics components. If the issues of surface attachment chemistry, secondary structure, and self-assembly can be worked out, hybrid DNA-silicon nanostructures may be feasible, and a DNA-controlled field effect transistor is one possible choice for a first structure to fabricate. Some other specific near-term objectives for research in DNA self-assembly include the creation of highly regular DNA nanoparticles and the creation of programmable DNA self-assembling systems. For the cell regulatory systems and enzymatic pathways, some specific near-term objectives include the creation of sets of coupled protein-DNA interactions or genes, the simulation and emulation of kinase phosphor-relay systems, and the creation of networks of interconnecting nanostructures with unique enzyme communication paths.

To be adopted successfully as an industrial technology, however, DNA self-assembly faces challenges similar to solution-based exhaustive search DNA computing: a high error rate, the need to run new laboratory procedures for each computation, and the increasing capability of non-DNA technologies to operate at nanoscales. For example, while it is likely true that current lithography technology has limits, various improvements already demonstrated in laboratories such as extreme ultraviolet lithography, halo implants, and laser-assisted direct imprint techniques can achieve feature sizes of 10 nm, comparable to a single DNA tile. Some other targets might be the ability to fabricate biopolymers such as oligonucleotides and polypeptides as long as 10,000 bases for the creation of molecular control systems and the creation of biochemical and hybrid biomolecular-inorganic systems that can be self-assembled into larger nanoscale objects in a programmable fashion.

8.4.3.4 Hybrid Systems

A hybrid system is one that is assembled from both biological and nonbiological parts. Hybrid systems have many applications, including biosensors, measurement devices, mechanisms, and prosthetic devices.

Biological sensors, or biosensors, probe the environment for specific molecules or targets through chemical, biochemical, or biological assays. Such devices consist of a biological detection element attuned to the target and a transduction mechanism to translate a detection event into a quantifiable electronic or optical signal for analysis. For example, antennae from a living silkworm moth have been used as an olfactory sensor connected to a robot.[144] Such antennae are much more sensitive than artificial gas sensors, in this case to moth pheromones. A mobile robot, so equipped, has been shown to be able to follow a pheromone plume much as a male silkworm moth does. When a silkworm moth's antennae are stimulated by the presence of pheromones, the moth's nervous system activities alternate between active and inactive states in a pattern consistent with the activity pattern of neck motor neurons that guide the moth's direction of motion. In the robot, the silkworm moth's antennae are connected to an electrical interface, and a signal generated by the right (left) antenna results in a "turn right" ("turn left") command. This suggests that such signals may play an important role in controlling the pheromone-oriented zigzag walking of a silkworm moth.

[144]Y. Kuwana et al., "Synthesis of the Pheromone-oriented Behaviour of Silkworm Moths by a Mobile Robot with Moth Antennae as Pheromone Sensors," *Biosensors and Bioelectronics* 14:195-202, 1999.