The Power and Limits of Algorithms

Richard Manning Karp

It is a great honor to address you today. I would like to tell you about my background and education, and about the events and decisions that have shaped my career as a computer scientist. I belong to the first generation that came to maturity after the invention of the digital computer. My school years coincided with enormous growth in the support of science in the United States. I have been able to work in two great universities and a great industrial research laboratory. No previous generation had access to such opportunities. If I had been born a few years earlier I would have had a very different, and undoubtedly less satisfying, career.

My Family

My family lived in Dorchester, a section of Boston, Massachusetts. Figure 1 shows a typical street in Dorchester in wintertime that is a quite similar to the street where my family lived. The city was divided into ethnic neighborhoods, and nearly everyone in our neighborhood was Jewish, as I am. We rented a small apartment, and I slept in the same room with my three younger siblings.

My mother was the strongest influence on my development. She went to work after graduating from high school, but eventually earned a Harvard degree at age 57 after taking evening courses for many years. Her top priority was the education of her children and she took great pride in my achievements at school. We had little money, and she taught us not to value wealth as an end in itself.

My father was a graduate of Harvard University. He was a humble and gracious man who was widely admired in our community. He had wanted to go to medical school but could not afford it financially, and became a school mathematics teacher instead, rising to the position of head of a school district. Teachers were poorly paid, so to support our family he took on a variety of extra jobs such as tutoring private students, working in a grocery store, and selling encyclopedias.

I have fond memories of visiting his classroom as a youngster. I remember that he was able to draw a nearly perfect circle on the blackboard freehand. It is probably no accident that two of my siblings, my brother David and my sister Carolyn as well as I, became teachers. David is a Sociology professor and a Carolyn is a teacher of English. We were following in our father's footsteps.

My parents have passed away, but I am delighted that my Aunt Phyllis, my mother's youngest sister, is here today. She was my babysitter, playmate and mentor, and taught me to read and ride a bicycle.

My wife Diana and son Jeremy cannot be here today, but I would like to acknowledge how they have enriched my life and contributed to my well-being. Diana is a docent at the San Francisco Museum of Modern Art, and Jeremy is an honor student at Brandeis University, majoring in Economics and Psychology.

Childhood Memories

I was a rather meek and quiet child and spent a great deal of time reading. I particularly remember books about ancient Greece, the travels of Marco Polo, cowboys and Indians, and the American Revolutionary War. I liked poetry and enjoyed reading it aloud. I was a great fan of the local professional baseball team, the Boston Red Sox. I enjoyed swimming in the ocean and riding my bicycle. I also worked in my grandfather's neighborhood grocery store, making deliveries to customers, carrying the groceries on a sled in the wintertime.

I excelled in academic work, but because I started school early and then skipped a grade I was much younger than my classmates and lagged behind them in manual skills such as penmanship, art and carpentry. Lack of confidence in my manual skills has continued throughout my life, and I have always had a resistance to doing laboratory work.

At school my favorite subjects were Latin and mathematics. I enjoyed learning Latin grammar because of its logical structure. At age ten I appeared several times on a radio show called the Boston Quiz Kids, where my strong point was solving mathematics puzzles. At the age of 13 I was introduced to plane geometry and was wonderstruck by the power and elegance of mathematical proofs. I recall feigning sickness in order to stay home from school and solve geometry problems.

Becoming a Scientist

I entered Harvard College at a time when physics was the most glamorous scientific field, but I found that I had no special aptitude for it, and mathematics came to me much more easily. But as I advanced in mathematics I was overshadowed by a few exceptional mathematics students, including a future Fields Medalist and a future Nobel Prize winner. This was somewhat discouraging but, at the same time, I received strong encouragement from Prof. Hartley Rogers, whose course in probability I had taken. I was particularly attracted to the applications of discrete mathematics (finite mathematics) and discrete probability to the application of problems of decision-making, ranging from baseball strategy to gambling systems to game theory. Such topics were not in the mainstream of mathematics at that time, and were of little interest to most of my mathematics professors. They have become much more prominent in recent years.

The great potential of computers was becoming clear by this time, and so I decided to pursue a Ph.D. at the Harvard Computation Lab. Sputnik in 1957 led to boom times in technical fields. I was able to obtain interesting summer jobs, had a few successes, and began to feel that I might amount to something.

Upon receiving my Ph.D., I was fortunate to obtain a position in Mathematical Sciences at IBM Research, where I was given great freedom to pursue my research in collaboration with excellent colleagues. My mathematical education was inadequate at that point, and I benefited greatly from conversations with my mentor at IBM, the mathematician Alan Hoffman. He taught me a great deal about combinatorial mathematics, showed confidence in my ability, and shaped my approach to teaching and research through the example of his deep enjoyment of beautiful mathematics and his zest for communicating it. I also became acquainted with a whole school of wonderful mathematicians, at places like the Rand Corporation, Princeton University and the National Bureau of Standards, who pioneered the application of discrete mathematics to problems of decision-making.

During my years at IBM I taught a number of evening courses at universities in New York City. I discovered that I loved everything about teaching—the mastery of new material in preparing a lecture, the contact with students—and the idea began to grow on

me that I was destined to be a professor.

A Feeling for Algorithms

I also came to realize that my life's work would be the study of algorithms. An algorithm is a systematic computational procedure for solving a specified problem. The most familiar algorithms are perhaps the ones for arithmetic that we learned in school. Algorithms are everywhere. At the core of every information processing application there is an algorithm. Whether it's processing search queries or routing messages through a network or conducting auctions over the Internet, algorithms are at the heart of it. At the core of all the cryptographic protocols that underlie e-commerce, there's an algorithm for encrypting data, which depends in turn on an algorithm for testing whether a number is prime. That algorithm can be written down in a few lines, but it's very subtle and without it we wouldn't be able to conduct business over the Internet the way we do.

Don Knuth, a great computer scientist who received the Kyoto Prize in 1996, has called attention to a breed of people who derive great aesthetic pleasure from contemplating the structure of computational processes. I still recall the exact moment when I realized that I was such a person. It was when a fellow student, Bill Eastman, showed me the so-called Hungarian Algorithm for solving the Assignment Problem, a classic problem of matching workers with jobs. I was fascinated by the elegant simplicity with which the algorithm converged inexorably upon the solution, using no arithmetic operations except addition and subtraction.

The first requirement of an algorithm is that it should be correct. It should do what it's supposed to do. Beyond that, it should be efficient. The primary way we measure the efficiency or cost of an algorithm is by the time it takes to execute—roughly speaking, the number of basic steps. I've spent a lot of my time devising algorithms that are efficient by that measure.

My first attempts to develop an algorithm came as a schoolboy. My father was a school principal and every fall before school started he had to create a schedule of classes that satisfied a variety of constraints. Certain classes could only be taught in certain rooms and at certain times. Certain pairs of classes could not conflict with one another. Certain teachers were unavailable on certain days, and so on. He got out a stack of index cards

representing the different classes, laid them out on the kitchen table, and started shuffling them around to contract a schedule. I was not able to offer much help. Finding a schedule that met all these conditions, among the vast number of possible schedules, was like searching for a needle in a haystack. The only method I could find to attack my father's scheduling problem was trial-and-error search.

Puzzle-like problems of this kind, which involve searching through a vast set of patterns or arrangements to find one that satisfies a set of conditions, arise in every scientific field and in every domain of human activity. They are essential tools for the efficient exploitation of society's resources. Examples include scheduling jobs in a factory, arranging components and wires on a computer chip, routing messages in the Internet or electricity in a power grid, and sequencing the human genome. My work contributes to all these fields both by creating useful algorithms for such combinatorial search problems and by clarifying their fundamental limitations.

Complexity

Some combinatorial search problems are surprisingly easy to solve. For example an efficient algorithm to compute the shortest route from a starting point to a destination in a road network underlies the satellite navigation systems in our cars. But most combinatorial search problems seem to elude such easy solutions, and to require exponentially growing computational resources as the description of the problem becomes more complex.

In general, combinatorial problems like the class-scheduling problem, which arise in commerce, in physical sciences, in engineering, and even in the humanities and social sciences, are very difficult to solve. There are a vast number of ways the objects can be arranged—assigning classes to time slots, for example—but only a few of these arrangements meet all the requirements. So the question is, are there shortcuts? Some problems have clever, efficient algorithms, but for most of the problems that come up, there seems to be no radical shortcut. You can solve small examples, but as the size grows—for example, the number of classes to schedule gets bigger—the running time of even the best algorithm tends to explode. That is, it roughly doubles every time you increase the size of the problems just a little bit.

At IBM during the 1960s I worked on a number of applied combinatorial problems in computer design and operations research and became painfully aware of combinatorial explosions in computation. Stimulated by conversations with Jack Edmonds, a visionary researcher at the National Bureau of Standards, I began to wonder whether such exponential growth in running time might be inherently unavoidable.

I was also following developments in computational complexity theory, the branch of computer science that studies the fundamental, inherent limitations on the efficiency of computation. Just as physics has fundamental laws such as conservation of energy, the laws of thermodynamics and the Heisenberg uncertainty principle, computational complexity theory strives to establish fundamental, inherent limitations on the efficiency of computation. It was natural to ask whether computational complexity theory could explain combinatorial explosions in the running time of algorithms, but the time was not yet ripe for me to attack this question.

Becoming a Professor

In 1968 I was offered a professorship at the University of California at Berkeley. It had been a great privilege to be a researcher at IBM, and my work there had been crucial for my development as a computer scientist and mathematician, but now it was time to fulfill my destiny as a teacher.

Berkeley in the late 1960s was at the forefront of social change. The Vietnam War was at its height. Soon after I arrived, there were major protests against the Cambodian invasion. I remember teaching some classes in my home because protesting students had closed the campus, and paying bail to release a faculty colleague who was arrested for participating in an illegal protest march against the war.

Computer Science at Berkeley was also in turmoil at that time. The Electrical Engineering Department was home to a number of computer scientists, but several had split off to form a new department to serve the College of Letters and Science. As the rivalry between the two departments was unhealthy both academically and socially, the administration decided to fold the two groups into a single unit within Electrical Engineering. I was not very experienced in academic affairs having just moved to academia a few years before, but since I hadn't taken sides in the controversy I was chosen to lead

this new unit. I wasn't a born administrator and I didn't stay in the job very long, but I played some role in quieting things down and getting people to accept the arrangement and start building something together.

Once the politics settled down Berkeley was poised to build a great computer science department. We have consistently had a thriving community of students of theoretical computer science, and there has always been a spirit of cooperation and enthusiasm among them. A major reason for the success of theory at Berkeley has been Manuel Blum. Manuel was my closest colleague at Berkeley for twenty-five years. Through his example I learned to have respect for all my students, not merely the strongest ones. I learned from him that the best way to teach is to lead students to make their own discoveries, and I learned to build my research around broad unifying concepts, rather than mere technical advances.

The Joy of Teaching

Teaching has been one of the greatest pleasures in my career, and possibly the most important way in which I have influenced the lives of others. I greatly enjoy the process of mastering a body of material and shaping it into a coherent lecture; it is the primary means by which I learn. Lecturing permits me to express my personality, share my experiences and opinions, and connect with each new generation of students.

I greatly value the opportunity to serve as a mentor and role model for talented research students. I have had 41 Ph.D. students. They have gone on to become leading professors and industrial researchers, found companies, and lead professional societies. They include the inventor of a famous algorithm for linear programming, the creator of new links between the fields of economics and computer science, the developer of a radical method of distributing data over the Internet, the designer of mechanisms that allow the resources of a computer network to be shared efficiently, and several who have made key contributions to computational molecular biology and served as my co-workers and mentors in that subject. I stay in touch with most of my former students, and enjoy witnessing the growth of their families.

My students asked me to formulate some principles of advice for pursuing their careers as theorists. This is what I told them. Understand what you are good at and what

you like to do, and choose accordingly. In the words of Socrates, “Know thyself.”Disregard the fashions of the day and search for new areas of research that are about to become important. To find exciting problems, look to the interfaces between disciplines. Believe in yourself and trust your own taste and judgment. Understand that progress comes slowly but that it will come with time. Enjoy the process of creation, even when progress is slow. Enjoy the company of your colleagues and be generous in sharing ideas. Perhaps the most important of these considerations is that if you are truly born to be a researcher you will enjoy what you are doing even on the days when you are not making progress, simply because of the beauty of the subject.

NP-Completeness

In 1971 a paper by Stephen Cook, a former Berkeley colleague, led me to think that computational complexity theory could be brought to bear on the question of whether combinatorial explosions in computation are inevitable. Cook defined a class of computational problems, now called NP that includes versions of a vast number of combinatorial computing problems. This class includes many famous problems, some of which have resisted efficient solution for 200 years going back to the time of the great mathematician Gauss. Cook showed that a certain problem in mathematical logic known as the Satisfiability Problem was as hard as any problem in this class. My experience with combinatorial search problems suggested to me that many well-known combinatorial search problems were just as hard as Satisfiability. Starting from the Satisfiability Problem I used a technique called “reduction” to build up a set of problems that are all of equivalent difficulty, and as hard as any problem in the vast class NP which includes virtually all the combinatorial search problems that we encounter. The problems that have this property of being the hardest in the class NP are called NP-complete. To show that a problem is NP-complete, all one needs to do is to show that, if you had an efficient algorithm for solving the new problem, you could also use that algorithm to solve some other problem already known to be NP-complete. Using this method of reducing one problem to another, I showed that 21 classic problems, arising in fields such as engineering, mathematics, natural science, biology and commerce, having no overt similarity in their descriptions, are NP-complete. The NP-complete problems are equivalent in the sense that if you could solve

any one of them efficiently, you could solve all of them efficiently. By now thousands of problems have been proven to be NP-complete by means of the reduction technique. Many of the reductions that specialists in the theory now produce are far more intricate than the ones I originally came up with. New reductions and NP-completeness proofs are published every day. Figure 2 illustrates one example: We see a solution to a particular instance of a NP-complete problem. The figure shows a large square into which non-overlapping rectangles of various sizes are packed. It is easy to verify that the solution is correct, but the NP-complete problem is not to verify a solution but to construct one: given the small rectangles, to find a way if possible to fit them into the big square without overlapping. So, it is a fundamental characteristic of the complexity class NP that solutions once given are easy to check, but may be very hard to find.

The next three cartoons illustrate three ways that a worker might explain to his boss that he has failed to solve an NP-complete problem. In the first one (Fig. 3), being ignorant of NP-completeness, he takes personal responsibility. He says, “I can't find an efficient algorithm. I guess I'm just too dumb.”

For the second one (Fig. 4), he would have to know that P is unequal to NP, in other words, that NP-complete problems are difficult. This is currently an open mathematical problem. He would say in this case, “I can't find an efficient algorithm because no such algorithm is possible. ”

The third one represents our current state of knowledge (Fig. 5). The empirical evidence of multitudes of failed attempts strongly indicates that we will never find an efficient algorithm to solve every instance of an NP-complete problem, but theoretical mathematical questions are resolved by proofs, not by empirical evidence, and we seem to be very far from proving that this is the case.

There have been many advances related to NP-completeness over the years. For example, when I first worked on certain problems, I was unable to classify them. One of these problems is to test whether a given number is prime. But, just few years ago, it was proven this problem is actually in the class P. So, there is an efficient way of solving it.

Another development is based on a beautiful theorem developed around 1990 called the PCP theorem (PCP stands for Probabilistically Checkable Proof). The theorem leads to reductions showing that certain problems are not only hard to solve exactly if P is

unequal to NP, but also are hard to solve even approximately.

Unfortunately, at this time, because we can not resolve the problem of P vs. NP, the field of complexity theory is filled with conditional theorems of the form, if P is unequal to NP, then some conclusion follows. Here are two examples: if P is unequal to NP, then cryptographic protocols that are useful for information transmission cannot be broken; if P is unequal to NP, then certain problems are hard to solve approximately. So it is crucial that we settle this question of deciding whether these two complexity classes are equal. The problem of deciding whether the NP-complete problems (and hence all problems in NP) are efficiently solvable is known as the P vs. NP problem. The P vs. NP problem has taken its place among the most prominent open questions in mathematics. It is one of the seven problems for whose solution the Clay Mathematics Institute has offered a million-dollar reward. Among those select problems it may well have the greatest philosophical significance, since it is equivalent to asking whether finding a proof is harder than checking a proof, and hence whether finding proofs inherently requires creative inspiration, as we expect. NP-completeness is the most important bridge between the abstract theory of computational complexity and the practical business of solving combinatorial problems. About 6,000 papers published each year have the term “NP-complete” in their title, abstract or list of keywords. NP-completeness is the one topic from theoretical computer science that every computer science undergraduate must learn.

NP-completeness has drawn the attention of many scientific fields to the fundamental limits of computation. Beyond its precise technical meaning, NP-completeness has become a metaphor for phenomena such as chaos in dynamical systems, bounded rationality in economics, the impossibility of certain ideal voting systems, the inability of artificial intelligence to simulate the human brain, and genetic indeterminism.

The work on NP-completeness contributed to raising the visibility and respectability of computational complexity theory within the pure mathematics community. In 1975 complexity theory was the topic of my series of Hermann Weyl lectures at the Institute for Advanced Study in Princeton, and a few years later the Institute established a thriving program in this subject. In 1985 I was co-organizer with the Fields Medalist Stephen Smale of a yearlong program in Computational Complexity at the Mathematical Sciences Research Institute in Berkeley.

Even when a combinatorial search problem is found to be NP-complete, there

may be ways to deal with it in practice. Suppose, for example, that you are planning a driving trip that will take you to many cities. You know the driving time between each pair of cities, and you want to visit the cities in an order that will minimize your total driving time. This is an instance of a general NP-complete problem called the traveling-salesman problem. Because the problem is NP-complete we cannot expect an efficient algorithm that will solve all cases, but it still may be possible to find the minimum-time solution for a particular instance, such as the one shown in Figure 6.

And, if not, a simple trial-and-error approach may give a solution that is satisfactory, although not optimal. In practice, we often use so-called heuristic algorithms, which run fast and give acceptable solutions to typical examples, even though there are no general guarantees on their performance. Figure 6 shows a heuristic solution to an example with 101 cities. Such heuristic algorithms often perform quite well on the instances that arise in practice, but we have no theoretical explanation of why this is the case. Sometimes we can prove that a heuristic algorithm succeeds on most instances, or even on almost all instances, but this does not imply that it will perform well on the examples that occur in practice. The mysterious success of some heuristic algorithms remains a challenge to be explained in the future.

Computational Biology

In 1990 the Human Genome Project was started, with the ambitious goal of determining the genetic makeup of the human species. Our genetic endowment consists mainly of the DNA in our chromosomes. A DNA molecule is a long, double-stranded chain of nucleotides, chemical units of four types: A, C, T and G. Thus, a fundamental goal was to determine the sequence of the human genome: the three billion nucleotides within the chromosomes. Using advanced technology it was possible to determine, with a small error rate, the sequences of millions of random fragments, each containing several hundred nucleotides. Figure 7 shows signals that come from a sequencing machine that automatically reads short DNA sequences.

I was immediately attracted to the combinatorial problem of reconstructing a long DNA sequence from noisy measurements of these millions of fragments. A first step in this direction was the so-called physical mapping problem of determining the positions of tens

of thousands of landmark sequences scattered along the genome. During the 1990s I worked with small teams of students to develop algorithms and software for solving the physical mapping problem. Our approach was very successful from a scientific point of view, but had little influence, as we were not sufficiently connected with the leading centers where the data for assembling the genome sequence was being gathered.

The first draft sequences of the human genome were announced in 2000, and since then the genomes of many organisms have been sequenced, but sequencing is only the first step toward understanding how living cells perform their functions. It still remains to determine the genes within these sequences, to understand how the genes direct the production of biomolecules such as RNA and protein, and to infer the complex networks of biomolecules, such as the networks indicated in Figure 8, that act in concert to control how cells replicate, repair themselves, maintain their structure, communicate with other cells, respond to their environments and perform their specialized functions within the body. Many questions suggest themselves. How is it that liver cells, blood cells and skin cells function very differently even though they contain the same genes? Why do cancer cells behave differently from normal cells? Which variations in our genome are correlated with disease? How can we understand the complex regulatory networks that control the functioning of cells and systems such as the immune systems? This is, of course, the area in which my fellow laureate, Professor Pawson has done fundamental work. I did not aspire to be a biologist like Professor Pawson but tried to learn enough biology to provide mathematical support for the investigation of data that allows such questions to be addressed. My current work is concerned in particular with inferring how differences between individuals at the genomic level affect their susceptibility to disease.

I have been fortunate to work with gifted younger scientists who began as my students but became my mentors. By sharing their biological knowledge with me they help me formulate the key algorithmic problems that must be resolved to reveal the secrets of the cell. Here is a quote from a distinguished biologist at University of Washington, Garrett Odell laying down a challenge, which many of us are now trying to address. He says, “We can approach understanding how the whole genome works by breaking it down into groups of genes that interact strongly with each other. Once researchers identify and understand these network modules, the next step will be to figure out the interactions within networks of networks, and so on until we eventually understand how the whole genome works, many

years from now."

The Computational Lens

At this point, I would like to offer a speculation about the future direction of computer science. Computer science has been called a "science of the artificial" because it deals in large part with entities such as computers, programs and algorithms that are devised and specified by humans rather than occurring in the natural world. However, many processes that occur in nature also lie within the province of computer science, since they can be regarded as computational. Many such examples occur in biological systems. Each of the following processes can be regarded as to some extent a systematic process of computation: learning by networks of neurons; the process by which the immune system marshals its defenses against an invading microbe, or by which cells acquire specialized functions during embryonic development; the collective behavior of animal communities, such as the flocking of birds or the self-organization of ant colonies; and the evolution of species. So, at each of these various levels of biology, we can ask, "To what extent is nature computing?" "To what extent are the processes algorithmic?" "Is the self-organization of ant colonies algorithmic in nature? Can we understand it as an algorithm? (Fig. 9)"

Do biological cells compute (Fig. 10)? Does the immune system compute (Fig. 11)? Does the brain compute (Fig. 12)? In this latter regard, the most interesting question is the following. Is the brain a machine? If we could elucidate every biochemical reaction occurring in the brain, would we be able to explain consciousness? I personally don't believe that this is the case.

The computational lens that I applied to biological processes can also be focused on man-made artifacts. Consider the Internet. It was never completely designed or specified. Rather, the Internet emerged as a complex system, simultaneously computational, social and economic, through the uncoordinated actions of many independent agents. For this reason, it is perhaps best studied empirically as an evolving organism, rather than an artificially constructed process with a single creator. It is more like a market or society than a well-specified mathematical object. My colleague, Christos Papadimitriou, speaking of the Internet and the Worldwide Web, said, "For the first time, we had to approach an artifact with the same puzzlement with which the pioneers of other sciences had to

approach the universe, the cell, the brain and the market."

The study of the Internet as an emergent computational system also raises questions for the social sciences. Minutely detailed data is becoming available about Internet-based social networks: networks of Web-based interactions linking individuals, organizations and communities. Many questions can be addressed. How do ideas, opinions, innovations and technologies spread? How can we identify coherent communities and sub-communities? How can we understand and exploit the "six degrees of separation" phenomenon?

Several other fields can be viewed through the computational lens. The behavior of computational devices at a sub-atomic level needs to be framed in terms of quantum mechanics, and leads to the abstract mathematical model of a quantum computer. The possibility of someday building quantum computer is tantalizing because, in principle, the qubit, the basic unit of information in a quantum computer, is much richer in content than the bit, the basic unit of information in a classical computer, which takes only the values of 0 and 1. So, there is a great interest in determining whether we can harness the potential power of a quantum computer. Although the technology for building a quantum computer has not yet been realized, the computational power of this model is being investigated, and can be used to define experiments that test the validity of the standard theory of quantum mechanics. And so, in connection with trying to build quantum computers much more probing experiments can be defined, which could possibly reveal flaws in the standard theory of quantum mechanics. As my colleague Umesh Vazirani puts it, "Quantum computing is as much about testing quantum physics as it is about building powerful computers."

Another area where there are intense links between the theory of computation and physical science is statistical physics. Statistical physics is based on probabilistic models of the interactions of large assemblages of atoms, molecules or molecular spins. The field studies such phenomena as the freezing of water and the magnetization of metals. Statistical physics finds common ground with probabilistic models in computer science involving the interactions of large numbers of combinatorial variables and constraints, or the interactions of many agents communicating and negotiating over the Internet.

The theory of computation is also finding common ground with economics. One area of overlap is economic mechanism design. This area has been stimulated by the

development of economic mechanisms for conducting transactions over the Internet. An economic mechanism is an algorithm whose inputs come from economic agents with private data and selfish interests. One example is the design of on-line auctions. The goal in economic mechanism design is to induce the participants, via incentives, to state their objectives truthfully and respond in ways that achieve the designer' s goals, such as social welfare or profit maximization. Another emerging area where the theory of computation meets economics is computational game theory. Classical game theory is the study of rational behavior in situations of conflict, and assumes perfect rationality. Computational game theory based on complexity theory injects a new ingredient, by postulating that the strategic behavior of the participants is limited by computational complexity, so that they may not be able to achieve perfect rationality because of computational limitations.

The brief descriptions I have given illustrate how viewing physical, social or engineered systems through the lens of their computational capabilities or requirements provides new insights and ways of thinking, and promises to move computer science closer to the center of scientific discourse. I expect much of my future work will be in this direction.

Let me close with a personal statement. Being a professor at a research university is, for me at least, the best job in the world. It provides great personal autonomy, the opportunity to serve as a mentor and role model for talented students, and an environment that encourages and supports work at the frontiers of emerging areas of science and technology. I am fortunate to have come along at a time when such a career path was available to me.

Fig. 1

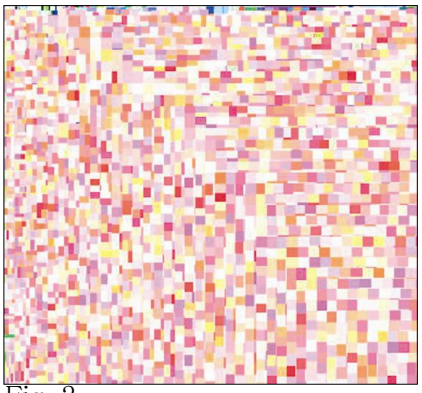

Fig. 2

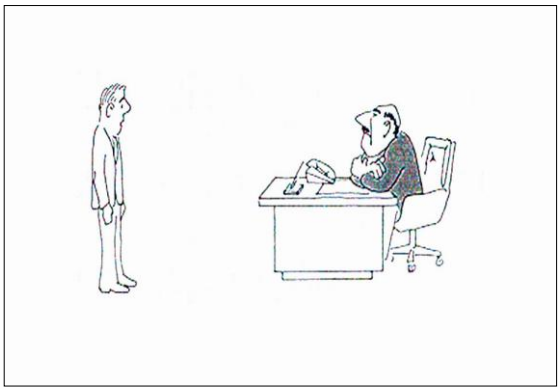

Fig. 3

Fig. 4

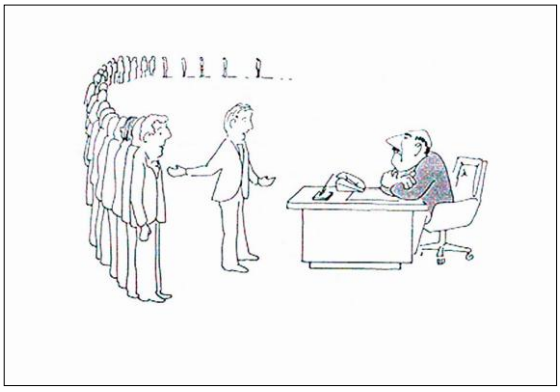

Fig. 5

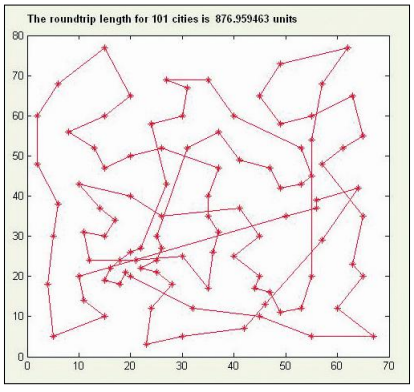

Fig. 6

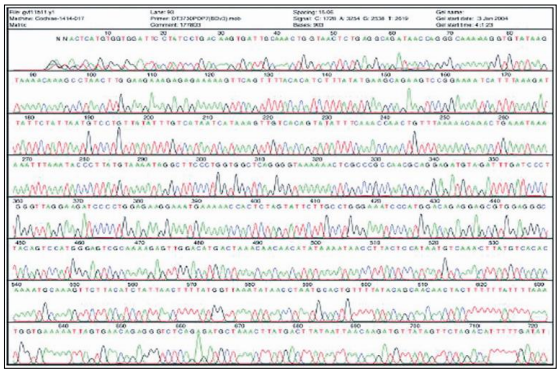

Fig. 7

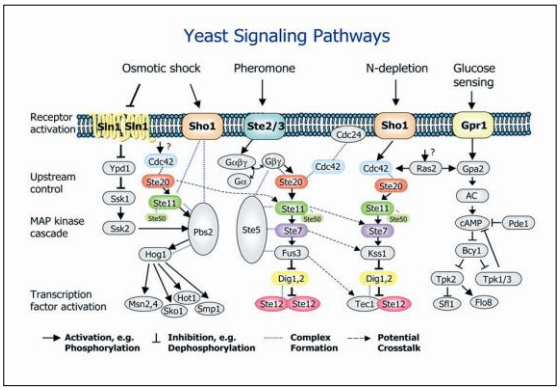

Fig. 8

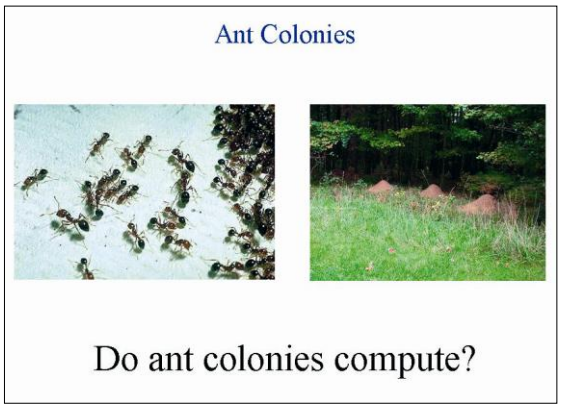

Fig. 9

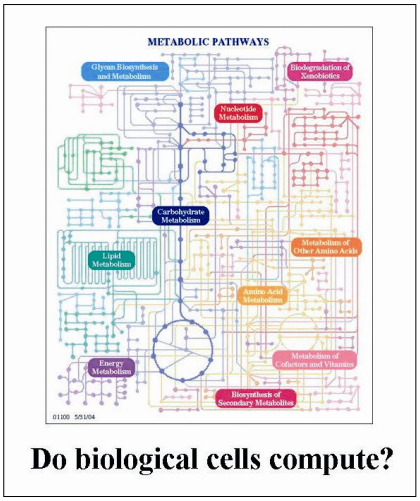

Fig. 10

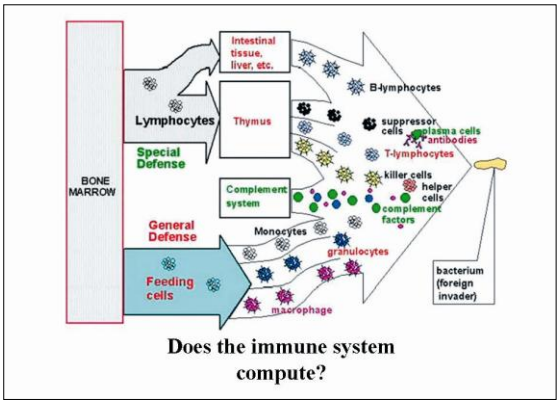

Fig. 11

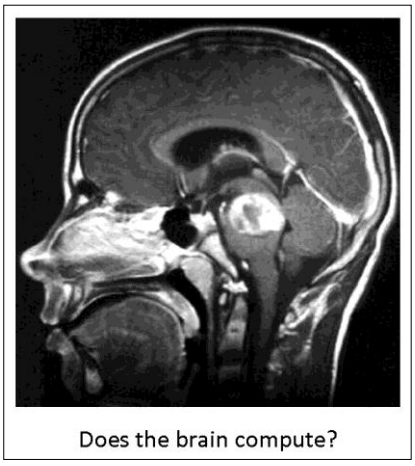

Fig. 12