People & Ideas in Theoretical Computer Science

edited by
C. S. Calude

Springer
Getting Educated

Subjects fall of surprises and surprises, I mean. How do decades and exposures to conscious algorithmic, the mind, they purpose to know life, experience. After more years, and the matching begins the open book of knowledge and potential. In the world of the great stories and destinations of the day, and, after a flow of experiences that are more of the works of nature, "know yourself."

By the Principle:

Understanding what you are good at and what you like to do and choose success.

The point that's been a great deal of work and writing, there to develop the theories and algorithms. I have been writing books and short stories with a goal of marketing to a wider audience. If I had been born a few years earlier, I would have been a different person.

If I have been born a few years earlier, I would have been a different person. I have been able to work in the great research and development field with a number of mentors who have supported me in the pursuit of my goals. I have been able to continue my education and develop my research in the field of computer science. I believe that the education that I have received has contributed to my success.

Richard M. Karp

Richard M. Karp was born in Boston, Massachusetts in 1933 and was educated at Harvard University. He received his bachelor's degree in mathematics and physics from Harvard in 1954 and his Ph.D. in mathematics from the Massachusetts Institute of Technology in 1957. He has been a professor at Stanford University since 1962.
found that I actually had to work in order to earn good grades, and that there were many students who equalled or surpassed my ability. I discovered that my writing ability was no better than workmanlike, and that laboratory science was definitely not the field for me.

The early mathematics courses were easy enough, but in the second half of my junior year I faced a tougher challenge in a course based on Halmos' monograph "Finite-Dimensional Vector Spaces," which developed linear algebra from the point of view of operator theory. The course coincided with the ending of an ill-fated sophomore romance. My unhappy love life shattered my morale, and for a time I retreated from the world, spending my afternoons at the Boylston Chess Club in Boston. As a result I spent virtually no time on mathematics, failed to master the Halmos text, and did poorly in the course.

In my senior year I greatly enjoyed a course in probability theory from Hartley Rogers, who encouraged me very strongly to pursue a mathematical career. On the other hand, I felt undermatched in Ahlfors' graduate course in Complex Analysis, where the students included a future Nobel Prize winner, a future Fields Medalist and a mathematical prodigy who was taking the course as a freshman.

By the middle of my senior year I had concluded that a career in pure mathematics was not for me. Being reluctant to leave Cambridge, and even more reluctant to work for a living, I decided to become a Ph.D. student at the Harvard Computation Lab, where Howard Aiken had built the Mark I and Mark IV computers. A faculty member at the Lab advised me to drop Prof. Ahlfors' Complex Analysis course immediately in favor of a solid introductory course in Accounting. Being an obedient young man I accepted this advice and approached Prof. Ahlfors for permission to drop the course. At first he was reluctant, but when he heard that I was giving up his course for one in Accounting he must have decided that there was no hope for me, as he gave his consent at once. At that point the die was cast - I was not to become a pure mathematician.

The Computation Lab

When I entered the Comp Lab in 1955 there were no models for a curriculum in the subject that today is called computer science. The young faculty offered courses in numerical analysis, switching theory, data processing, computational linguistics and operations research, and outside the Lab I took a variety of courses in applied mathematics, electrical engineering, probability and statistics. My performance was spotty, but I seemed to have a special feel for those topics that involved probability and discrete mathematics, and my successes in those areas produced a feeling of confidence. Sputnik in 1957 led to boom times in technical fields, and summer jobs became plentiful. Productive summers with M.I.T. Lincoln Lab and General Electric further fortified my sense that I might amount to something after all. A major turning point was Tony Oettinger's numerical analysis seminar in the fall of 1957, where I had the opportunity to give some talks, and discovered the pleasure of teaching.

Don Knuth 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 Hungarian Algorithm for solving the Assignment Problem. I was fascinated by the elegant simplicity with which the algorithm converged inexorably upon the optimal solution, performing no arithmetic operations except addition and subtraction.

My Ph.D. dissertation was based on the idea that the flow of control in a computer program can be represented by a directed graph, and that graph theory algorithms can be used to analyze programs. In a loose sense this work was a precursor of the later development of the field of code optimization. Tony Oettinger was my supervisor, and my other readers were Ken Iverson and Gerry Salton.

The Comp Lab's old boy network was already operative by the time I finished my dissertation, and Fred Brooks, who had preceded my by two years at the Lab and had become a major player at IBM, set me up with a wide range of interviews. I accepted a position in the Mathematical Sciences Department within IBM's Research Division.

Nirvana on the Hudson

In January, 1959 I reported for work at the Lamb Estate, a former sanitarium for wealthy alcoholics that was the temporary home of the fledgling IBM Research Division. There was a diverse group of applied mathematicians under the direction of Herman Goldstine, John von Neumann's long-time collaborator, and an exciting group under Nat Rochester, doing what would today be called cognitive science. The atmosphere was informal; a high point of each day was the lunchtime frisbee game on the vast lawns that surrounded the Lamb Estate.

I was assigned to work on algorithms for logic circuit design under the direction of the topologist Paul Roth, who had made fundamental contributions to the subject. It was this work that first brought me up against the harsh realities of combinatorial explosions. While some of the algorithms our group devised scaled well with increasing problem size, others were essentially enumerative, and their running time escalated exponentially as the number of variables increased.

To my great good fortune IBM Research became a mecca for combinatorial mathematicians during the early sixties. Although computers were primitive by today's standards, they could already be used to solve logistical problems of significant size. Splendid algorithms for linear programming and network flow problems had been discovered, and the field of combinatorial algorithms was in a stage of rapid development. In the summer of 1960 the leaders of the field came together at the Lamb Estate for an extended period. Among the visitors were Richard Bellman, George Dantzig, Merrill Flood, Ray Fulkerson, Ralph Gomory, Alan Hoffman, Ed Moore, Herb Ryser, Al Tucker and Marco Schutzenberger. Soon thereafter IBM brought in Ralph Gomory and Alan Hoffman to build a combinatorics research group.
Alan Hoffman became my mentor. He is a virtuoso at linear algebra, linear programming and algebraic graph theory, and has a talent for explaining mathematical ideas with clarity, precision and enthusiasm. Although my interests were more algorithmic than his, his style of exposition became a model for my own. I gained an understanding of the theory of linear programming and network flows, and came to appreciate how special their structure was compared to nastier problems such as integer programming and the traveling-salesman problem. During this period Mike Held and I developed the 1-tree heuristic, which remains the best method of computing a tight lower bound on the cost of an optimal traveling-salesman tour. With a bit of help from Alan Hoffman and Phil Wolfe we realized that our heuristic was a special case of an old method called Lagrangian relaxation; this connection motivated many other researchers to apply Lagrangian relaxation to difficult combinatorial optimization problems.

I also visited the National Bureau of Standards to work with Jack Edmonds on network flow problems. We pointed out, perhaps for the first time, the distinction between a strongly polynomial algorithm, whose running time (assuming unit-time arithmetic operations) is bounded by a polynomial in the dimension of the input data, and a polynomial-time algorithm, whose running time is bounded by a polynomial in the number of bits of input data. We gave the first strongly polynomial algorithm for the max-flow problem. For the min-cost flow problem we introduced a scaling technique that yielded a polynomial-time algorithm, but we were unable to find a strongly polynomial algorithm. The first such algorithm was obtained by Eva Tardos in the early 80's.

A few years before our collaboration began Edmonds had published a magnificent paper entitled "Paths, Trees and Flowers" which gave an algorithm for constructing a matching of maximum cardinality in any given graph. The paper began by introducing the concept of a "good algorithm," Edmonds' term for what is now called a polynomial-time algorithm. He showed that his algorithm for matching was a good one, and, even more importantly, raised the possibility that, for some combinatorial optimization problems, a good algorithm might not exist. This discussion by Edmonds was probably my first exposure to the idea that some standard combinatorial optimization problem might be intractable in principle, although I later learned that Alan Cobham and Michael Rabin had thought along similar lines, and that the possibility had been discussed extensively in Soviet circles.

IBM had a strong group in formal models of computation under the leadership of Cal Elgot. Through my contacts with that group I became aware of developments in automata theory, formal languages and mathematical logic, and followed the work of pioneers of complexity theory such as Rabin, McNaughton and Yamada, Hartmanis and Stearns, and Blum. Michael Rabin paid an extended visit to the group and became my guide to these subjects. From Hartley Rogers' splendid book "Theory of Recursive Functions and Effective Computability" I became aware of the importance of reducibilities in recursive function theory, but the idea of using subrecursive reducibilities to classify combinatorial problems did not yet occur to me.

My own work on formal models centered around parallel computation. Ray Miller, Shmuel Winograd and I did work that foreshadowed the theory of systolic algorithms. Miller and I introduced the parallel program schema as a model of asynchronous parallel computation; in the course of this work we introduced vector addition systems and initiated the study of related decision problems. The most notorious of these was the reachability problem, which after many false tries was proved to be decidable through the efforts of several researchers, culminating in a 1982 paper by Rao Kosaraju.

A Zest for Teaching

I moved to Berkeley at the end of 1968 in order to lead a more rounded life than the somewhat isolated suburban environment of IBM could provide. One aspect of this was the desire to be more involved with students. My father was a junior high school mathematics teacher, and I have fond memories of visiting his classroom as a youngster. He was undoubtedly the role model responsible for my attraction to teaching. I have been involved in teaching throughout my career and have always enjoyed it. Recently Greg Sorkin, a former student, reminded me of some thoughts on teaching that I wrote up for the Berkeley students in 1988. I include them here.

Thoughts on Teaching

Preparation

Follow the Boy Scout motto: Be Prepared!

Never select material that doesn't interest you. Boredom is deadly and contagious. If the standard syllabus is boring, then disregard it and pick material you like.

Figure out your notation and terminology in advance. Know exactly where you're going, and plan in detail what you are going to write on the board.

Check out the trivial details. They are more likely to hang you up than the major points.

Make sure you understand the intuition behind the technical results you are presenting, and figure out how to convey that intuition.

Debug your assignments and exams. They are just as important as the lectures.

Don't teach straight out of a textbook or from old notes. Recreate the material afresh, even if you're giving the course for the tenth time.

Structuring the Material

Ideally, each lecture should be a cohesive unit, with a small number of clearly discernible major points.

In organizing your lecture, use the principles of structured design: top-down organization, modularity, information hiding, etc.
Make sure the students have a road map of the material that is coming up.

Conducting the Lecture

Take a few minutes before each class to get relaxed.
Start each lecture with a brief review.
Go through the material at a moderate but steady pace. Don’t worry about covering enough material. It will happen automatically if you don’t waste time.
Write lots on the board; it helps the students’ comprehension and keeps you from going too fast. Print, even if your handwriting is very clear. Cultivate the skill of talking and writing at the same time.
Talk loud enough and write big enough.
Maintain eye contact with the class.
Develop a sense of how much intensity the students can take. Use humor for a change of pace when the intensity gets too high.
Be willing to share your own experiences and opinions with the students, but steer clear of ego trips.
Make it clear that questions are welcome, and treat them with respect. In answering questions, never obfuscate, mystify or evade in order to avoid showing your ignorance. It’s very healthy for you and the students if they find out that you’re fallible.

Be flexible, but don’t lose control of the general direction of the lecture, and don’t be afraid to cut off unproductive discussion. You’re in charge; it’s not a democracy.

If you sense from questions or class reaction that you’re not getting through, back up and explain the material a different way. The better prepared you are, the better you will be able to improvise.

Try the scribe system, in which the students take turns writing up the lectures and typesetting the notes.

Start on time and end on time.

The Professorial Life

The move to Berkeley marked the end of my scientific apprenticeship. At IBM I had enjoyed the mentorship of Alan Hoffman and Michael Rabin, and the opportunity to work with a host of other experienced colleagues. At Berkeley I worked mainly with students and young visiting scientists, and I was expected to serve as their mentor. The move also caused a sudden leap in my professional visibility. From 1968 onward I have been steadily besieged with requests to write letters of reference, do editorial work and serve on committees. With the advent of e-mail, the flow of requests has become a deluge; I offer my sincere apologies to any readers to whom I have been e-brusque.

A professor’s life is a juggling act. The responsibilities of teaching, research, advising, committee work, professional service and grantsmanship add up to a full agenda. Fortunately, I have always heeded the advice of my former colleague Beresford Parlett: “If it’s not worth doing, it’s not worth doing well.”

Parlett’s wise counsel has helped me resist administrative responsibilities, but there was one period when I could not avoid them. Berkeley in the 60’s was a cauldron of political controversy, and the mood of dissent extended to computer science. The faculty were divided as to whether to maintain computer science in its traditional home within Electrical Engineering, or to establish it as a separate department in Letters and Science. In 1967 the administration decided to do both, and I was one of several faculty hired into the newly formed Computer Science Department. The two-department arrangement was awkward administratively and only exacerbated the tensions between the two groups. In 1972 the administration decreed that the two groups of computer science faculty should be combined into a Computer Science Division within the Department of Electrical Engineering and Computer Sciences. As the faculty member least tinged with partisanship I emerged as the compromise candidate to head the new unit, and I somewhat reluctantly took the job for two years. I expected a period of turmoil, but once the merger was a fait accompli the tensions dissipated and harmony reigned. Much of the credit should go to Tom Everhart, the department chair at the time and later the Chancellor of Caltech. Tom nurtured the new unit and respected its need for autonomy.

Once the political disputes had healed Berkeley was poised to become a great center for computer science. In theoretical computer science the merger created a strong group of faculty, anchored by Manuel Blum, Mike Harrison, Gene Lawler and myself. Berkeley became a mecca for outstanding graduate students, and has remained so to this day. It is one of the handful of places that have consistently had a thriving community of theory students, 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, a deep researcher, a charismatic teacher, and the best research adviser in all of computer science.

Over the years at Berkeley I supervised thirty-five Ph.D. students. I have made it a rule never to assign a thesis problem, but to work together with each student to develop a direction that is significant and fits the student’s abilities and interests. Each relationship with a thesis student is unique. Some students are highly independent and merely need an occasional sounding board. Others welcome collaboration, and in those cases the thesis may become a joint effort. Some students have an inborn sense of how to do research, while others learn the craft slowly, and only gradually develop confidence in their ability. My greatest satisfaction has come from working with these late bloomers, many of whom have gone on to successful research careers.

NP-Completeness

In 1971 I read Steve Cook’s paper “The Complexity of Theorem-Proving Procedures,” in which he proved that every set of strings accepted in polynomial time by a nondeterministic Turing machine is polynomial-time reducible to SAT
Levin, in the Soviet Union, had independently been working along the same lines as Cook and myself, and had obtained similar results.

The early work on \( NP \)-completeness had the great advantage of putting computational complexity theory in touch with the real world by propagating to workers in many fields the fundamental idea that computational problems of interest to them may be intractable, and that the question of their intractability can be linked to central questions in complexity theory. Christos Papadimitriou has pointed out that in some disciplines the term \( NP \)-completeness has been used loosely as a synonym for computational difficulty. He mentions, for example, that Diffie and Hellman, in their seminal paper on public-key cryptography, cited \( NP \)-completeness as a motivation for posing the existence of one-way functions and trapdoor functions, even though it can be shown that, when translated into a decision problem, the problem of inverting a one-way function lies in \( NP \cap co-NP \), and thus is unlikely to be \( NP \)-complete.

The study of \( NP \)-completeness is more or less independent of the details of the abstract machine that is used as a model of computation. In this respect it differs markedly from much of the earlier work in complexity theory, which had been concerned with special models such as one- or two-tape Turing machines and with lower levels of complexity such as linear time or quadratic time. For better or for worse, from the birth of \( NP \)-completeness onward, complexity theory has been mainly concerned with properties that are invariant under reasonable changes in the abstract machine and distortions of the time complexity measure by polynomial factors. The concepts of reducibility and completeness have played a central role in the effort to characterize complexity classes.

After my 1972 paper I did little further work on \( NP \)-completeness proofs. Some colleagues have suggested that I had disdain for such results once the general direction had been established, but the real reason is that I am not particularly adept at proving refined \( NP \)-completeness results, and did not care to compete with the virtuosi of the subject who came along in the 70's.

**Dealing with \( NP \)-Hard Problems**

There is ample circumstantial evidence, but no absolute proof, that the worst-case running time of every algorithm for solving an \( NP \)-hard optimization problem must grow exponentially with the size of the instance. Since \( NP \)-hard problems arise frequently in a wide range of applications they cannot be ignored; some means must be found to deal with them.

The most fully developed theoretical approach to dealing with \( NP \)-hard problems is based on the concept of a polynomial-time approximation algorithm. An \( NP \)-hard minimization problem is said to be \( \tau \)-approximable if there is a polynomial-time algorithm which, on all instances, produces a feasible solution whose cost is at most \( \tau \) times the cost of an optimal solution. A similar definition holds for maximization problems.

\( NP \)-hard problems differ greatly in their degree of approximability. The minimum makespan problem in scheduling theory and the knapsack problem are
probabilistic, but the probabilistic choices are internal to the algorithm, and no assumptions about the distribution of input data are required.

As I stated in a 1991 survey paper, "Randomization is an extremely important tool for the construction of algorithms. There are two principal types of advantages that randomized algorithms often have. First, often the execution time or space requirement of a randomized algorithm is smaller than that of the best deterministic algorithm that we know of for the same problem. But even more strikingly, if we look at the various randomized algorithms that have been invented, we find that invariably they are extremely simple to understand and to implement; often, the introduction of randomization suffices to convert a simple and naive deterministic algorithm with bad worst-case behavior into a randomized algorithm that performs well with high probability on every possible input."

Inspired by Rabin’s paper and by the randomized primality test of Solovay and Strassen, I became a convert to the study of randomized algorithms. With various colleagues I have worked on randomized algorithms for reachability in graphs, enumeration and reliability problems, Monte Carlo estimation, pattern matching, construction of perfect matchings in graphs, and load balancing in parallel backtrack and branch-and-bound computations. We have also investigated randomized algorithms for a variety of on-line problems and have made a general investigation of the power of randomization in the setting of on-line algorithms. I take special pride in the fact that two of my former students, Rajeev Motwani and Prabhakar Raghavan, wrote the first textbook devoted to randomized algorithms.

In a 1982 paper Les Valiant suggested the problem of finding a maximal independent set of vertices in a graph as an example of a computationally trivial problem that appears hard to parallelize. Avi Wigderson and I showed that the problem can be parallelized, and in fact lies in the class $NC$ of problems solvable deterministically in polylog time using a polynomial-bounded number of processors. It was fairly easy to construct a randomized parallel algorithm of this type, and the harder challenge was to convert the randomized algorithm to a deterministic one. We achieved this by a technique that uses balanced incomplete block designs to replace random sampling by deterministic sampling. This was one of the first examples of derandomization - the elimination of random choices from a randomized algorithm. Later, Mike Luby and Noga Alon found simpler ways, also based on derandomization, to place the problem in $NC$.

In 1985 Nick Pippenger, Mike Sipser and I gave a rather general method of reducing the failure probability of a randomized algorithm exponentially at the cost of a slight increase in its running time. Our original construction, which is based on expander graphs, has been refined by several researchers, and these refinements constitute an important way of reducing the number of random bits needed to ensure that a randomized algorithm achieves a specified probability of success.
A spirited debate

In the field of computer science, the concept of "complexity" is often debated. Some argue that complexity is a fundamental property of information, while others believe it is a social construct. The debate continues as researchers explore the boundaries of what can be computed efficiently.

The Complexity Year

The Complexity Year is an annual event where experts from various fields gather to discuss the latest developments in complexity theory. This year's event focused on the implications of recent breakthroughs in quantum computing and their potential impact on cryptography.

The implications of these new findings are significant, as they could revolutionize the way we approach security and privacy in the digital age. However, the complexity of these developments also presents new challenges, particularly in terms of practical implementation and security.

As the debate continues, it is clear that understanding complexity is crucial for the future of technology. The Complexity Year serves as a platform for researchers to share their insights and collaborate on solutions to these complex problems.
Computational Molecular Biology

In the second half of this century molecular biology has been one of the most rapidly developing fields of science. Fundamental discoveries in the 50's and 60's identified DNA as the carrier of the hereditary information that an organism passes on to its offspring, determined the double helical structure of DNA, and illuminated the processes of transcription and translation by which genes within the DNA direct the production of proteins, which mediate the chemical processes of the cell. The connections between these processes and digital computation are striking: the information within DNA molecules is encoded in discrete form as a long sequence of chemical subunits of four types, and the genes within these molecules can be thought of as programs which are activated under specific conditions. Technology for manipulating genes has led to many applications to agriculture and medicine, and nowadays one can hardly pick up a newspaper without reading about the isolation of a gene, the discovery of a new drug, the sequencing of yet another microbe, or new insights into the course of evolution.

In 1963 the Mathematical Sciences Department at IBM decided to look into the applications of mathematics to biology and medicine, and I visited the Cornell Medical Center in New York City and the M.D. Anderson Hospital in Houston, looking for a suitable research problem. Nothing came of this venture, except that I ran across the work of the geneticist Seymour Benzer, in which he invented the concept of an interval graph in connection with his studies of the arrangement of genes on chromosomes; this was one of the earliest connections between discrete mathematics and genetics.

Over the next decades I was an avid reader of popular literature about molecular biology and genetics, but it was not until 1991 that I began to think seriously about applying my knowledge of algorithms to those fields. By then the Human Genome Project had come into existence, and it was evident that combinatorial algorithms would play a central role in the daunting task of putting the three billion symbols in the human genome into their proper order. The databases of DNA and protein sequences, genetic maps and physical maps had begun to grow, and to be used as indispensable research tools. My friend and colleague Gene Lawler and my former student Dan Gusfield, as well as several Berkeley graduate students, were working closely with the genome group at the Lawrence Berkeley Laboratory up the hill from the Berkeley campus, and I began to attend their seminars, as well as Terry Speed’s seminar on the statistical aspects of mapping and sequencing.

To get started in computational biology I decided to tackle the problem of physical mapping of DNA molecules. We can view a DNA molecule as a very long sequence of symbols from the alphabet \( \{A, C, T, G\} \). Scattered along the molecule are features distinguished by the occurrence of particular short DNA sequences. The goal of physical mapping is to determine the locations of these features, which can then be used as reference points for locating the positions of genes and other interesting regions of the DNA. The map is inferred from the fingerprints of clones; a clone is a segment of the DNA molecule being mapped, and the fingerprint gives partial information about the presence or absence of features on the clone. The problem of determining the arrangement of the clones and the features along the DNA molecule is a challenging combinatorial puzzle, complicated by the fact that the fingerprint data may be noisy and incomplete.

Beginning around 1991 my students and I developed computer programs to solve a number of versions of the physical mapping problem, but at first we lacked the close connections with the Human Genome Project that would enable us to have a real impact. In 1994, through my friends Maria Klawe and Nick Pippenger at the University of British Columbia, I made contact with a group of computer scientists and biologists who were meeting from time to time at the University of Washington to discuss computational problems in genomics. In addition to Maria and Nick, the group included the computer scientists Larry Ruzzo and Martin Tompa, as well as the geneticist Maynard Olson and the computational biologist Phil Green. I found the meetings very useful, and realized that the University of Washington was a hotbed of activity in the application of computational methods to molecular biology and genetics.

During the early 90's the University of California had a rich pension fund but a lean operating budget. In order to solve its financial problems the University offered a series of attractive early retirement offers to its older and more expensive faculty. Although I knew that it would not be easy to leave Berkeley after twenty-five years, I succumbed to the third of these offers.

In 1988 I had been part of a group of Berkeley faculty who helped establish ICSI, an international computer science research institute at Berkeley. Mike Luby, Lenore Blum and I built up a theoretical computer science group at ICSI which attracted outstanding postdocs and visitors from around the world. For the first year of my ‘retirement’ I based myself at ICSI, but in 1995 I moved to the University of Washington. I was attracted by the congenial atmosphere and strong colleagues in the computer science department at UW, and by the strength and depth of the activity in molecular biotechnology, led by Lee Hood. Hood saw the sequencing of genomes as merely a first step towards the era of functional genomics, in which the complex regulatory networks that control the functioning of cells and systems such as the immune system would be understood through a combination of large-scale automated experimentation and subtle algorithms. I decided that nothing else I might work on could be more important than computational biology and its application to functional genomics.

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? Although each gene codes for a protein, complex regulatory networks within the cell determine which proteins are actually produced, and in what abundance. These networks control the rate at which each gene is transcribed into messenger RNA and the rate at which each species of messenger RNA is translated into protein. These rates depend on the environment of the cell, the abundance of different proteins within the cell and the presence of mutated genes within the cell. Newly developed technologies make it possible to take a detailed snapshot of a cell, showing the rates of transcription of thousands of genes and the levels of large numbers of proteins. In model organisms
such as yeast we also have the ability to disrupt individual genes and observe how the effects of those disruptions propagate through the cell. The problem of characterizing the regulatory networks by performing strategically chosen disruption experiments and analyzing the resulting snapshots of the cell will be the focus of much of my future work. I expect to draw on the existing knowledge in statistical clustering theory and computational learning theory, and will need to advance the state of the art in these fields in order to succeed.

With the help of outstanding mentors I have enjoyed learning the rudiments of molecular biology. I have found that the basic logic of experimentation in molecular biology can, to some extent, be codified in abstract terms, and I have discovered that the task of inferring the structure of genomes and regulatory networks can lead to interesting combinatorial problems. With enough simplifying assumptions these problems can be made quite clean and elegant, but only at the cost of disregarding the inherent noisiness of experimental data, which is an essential aspect of the inference task. Combinatorial optimization is often useful, but unless the objective function is chosen carefully the optimal solution may not be the true one. Typically, the truth emerges in stages through an interplay between computation and experimentation, in which inconsistencies in experimental data are discovered through computation and corrected by further experimentation.

Conclusion

Being a professor at a research university is the best job in the world. It provides a degree of personal autonomy that no other profession can match, 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 has been available.

Solomon Marcus

Professor Solomon Marcus is affiliated to the Department of Mathematics, University of Bucharest, Romania, where he was successively, student 1945, instructor 1951, assistant professor 1955, associate professor 1964, professor 1966, and emeritus professor 1991. Research in set theory, real analysis, general topology, where he is quoted mainly for his results related to measure vs. Baire category, differentiation, Darboux property, Jensen convexity, quasicontinuity, determinant and stationary sets, symmetry of sets and functions, Riemann integrability in topological spaces, Hamel bases. In the late fifties and early sixties Professor Marcus became one of the initiators of mathematical and computational linguistics, proposing algebraic, logical and set-theoretic models for some fundamental linguistic categories in phonology, morphology and syntax. Later, he extended his interest to poetry, being one of the founders of mathematical poetics. Then, he became active in the semiotics of artificial languages (including also programming languages). Quoted by more than a thousand authors, he gave invited lectures and had temporary positions in most European countries, in U.S.A., Canada, Brazil, New Zealand and other countries. He is a member of the editorial boards of several journals of mathematics, computer science, linguistics, poetics and semiotics. Among his former students and pupils one can find many well known names in the fields of mathematics and computer science.

Bridging Linguistics and Computer Science, via Mathematics

From Poetry to Mathematical Analysis

When I was fifteen, I was fascinated by poetry. Mathematics was still a territory remaining to be discovered (school mathematics seems to be still today rather a