Madhu Sudan Receives Nevanlinna Prize

Meena Mahajan and Priti Shankar

The Nevanlinna Prize is awarded every four years by the International Congress of Mathematicians (since 1982), to young researchers who have made outstanding contributions in mathematical aspects of information science. The 2002 Nevanlinna Prize has been awarded to Dr. Madhu Sudan, from the Electrical Engineering and Computer Science Department, MIT, USA, for his work in the areas of probabilistically checkable proofs PCPs, non-approximibility of optimization problems, and error-correcting codes. In this article, we describe these areas and Madhu Sudan's important contributions.

Efficient Computation

Computational complexity is concerned with the inherent hardness of problems and addresses questions of the form: How fast can a given problem be solved on any computer? Or, what problems can be solved on a computer given a particular time limit or a space limit? Typically, a problem is posed as a *decision problem* i.e. one where the solution consists of answering with a Yes or a No. For example, in the primality problem, the input is a number n, and the question is to decide whether n is prime. All the inputs for which the answer is yes form a set; this set completely describes the problem. An algorithm solves such a problem if it correctly answers any question of the form "Is x in the corresponding set?".

An important parameter describing the efficiency of an algorithm is the number of steps it takes as a function of the input length. An algorithm is said to be efficient, or tractable, or feasible, if the number of steps it takes on an input of length N is bounded by some polynomial in N, say N^c for some constant c. Problems with feasible algorithms form the complexity class P. For instance, for the primality problem, the number n can be represented in base a with $\log_a n$ digits. So for any integer a greater than 1, the length of the representation is roughly $\log n$. Recently Agrawal, Kayal and Saxena showed that the primality of n can be decided within $(\log n)^{12}$ steps. Thus primality testing is feasible and in P.

Several important naturally occurring problems are not known to be in P. Consider for instance, the set HAM of Hamiltonian graphs; these are graphs that have a Hamiltonian cycle, i.e. a cycle touching every vertex exactly once. No efficient algorithm is known for deciding if a graph is in HAM. However HAM has an important property; it has solutions that are *efficiently verifiable*. That is, if a graph G is alleged to be Hamiltonian, and a proof in the form of an alleged Hamiltonian cycle C is given, an algorithm verifying this proof would have to check that (1) all edges of C are in G, and (2) C touches every vertex exactly once. Clearly, each of these tasks can be performed in time polynomial in the length of the graph description. Many other problems not known to be in P have this property of efficiently verifiable solutions. Such problems constitute the class *NP*. (*NP* stands for non-deterministic polynomial time).

It is clear that all of P is in NP. A central question in the theory of computational complexity is whether *P* equals *NP*. The belief that it is not so is pretty deep seated, though a formal proof is not yet available.

Some problems within NP enjoy a special status: if any of them is in P, all of NP collapses into P. These problems are called the NP-complete problems. HAM is such a problem. Formally, a problem Π_1 is said to be NP-complete if it is in NP and if for every problem Π_2 in *NP* there is a polynomial time algorithm that transforms an instance x of Π_2 into an instance y of Π_1 such that $x \in \Pi_2 \Leftrightarrow y \in \Pi_1$. It follows that all NP-complete problems are computationally equivalent (instances of one can be transformed into instances of any other).

Generalising the Concept of a Proof

Since sets in *NP* have efficient verification algorithms, for every set A in NP, we can imagine a setting with two agents: an all-powerful *prover*, and a polynomial-time-bounded *verifier*. On input x, the following *interaction* takes place between these agents.

Prover's role: Claim that $x \in$A. Provide a proof y, of length $|y|$ bounded by a polynomial in the input length $|x|$.

Verifier's role: Check that y is a valid proof of membership for x.

The interaction has the property that if $x \in A$, then there is a y that the prover can supply which will convince the verifier, but for $x \notin A$, no y that the prover supplies will convince the verifier.

An interesting question was investigated by Goldwasser, Micali and Rackoff, and independently by Babai and Moran: What if the verifier is allowed to use randomness and ask for parts of the proof in a way that the prover is not able to anticipate? This may require more than one round of questions and answers between the prover and the verifier. For instance, to verify Hamiltonicity, the prover may have a cycle C in mind. The verifier asks what are the neighbours of a randomly chosen vertex v on this cycle, then simplifies the graph, and then asks for the neighbours of some other vertex. After a few rounds, without having asked about neighbours of all vertices, the verifier may see for himself that the given information can be extended to produce a Hamiltonian cycle. If however the graph is not in HAM, after a few rounds the verifier may have pushed the prover into a corner.

Supposing that small probability of error can be tolerated; that is, since the verifier is now a randomized algorithm, we expect the following behaviour:

Completeness: If $x \in$A, then there is a way for the prover to answer questions, which will convince the verifier with high probability.

Soundness: If $x \notin A$, no matter how the prover answers queries, the verifier will remain unconvinced with high probability.

The class of languages for which membership can be verified this way is called IP (for interactive proofs). Very surprisingly, it was shown to be much larger than NP – it equals the class of problems for which polynomial *space* is allowed. This class contains problems for which polynomial time proofs are not known. Evidently randomness makes a huge difference.

Things became even more interesting when Ben-Or and Goldwasser investigated the scenario where multiple all-powerful provers (isolated from one another) are allowed and the verifier is allowed to make random queries to any of them, but still be restricted to polynomial time and allowed a small probability of error. Babai, Fortnow and Lund showed that this class, called MIP, is much larger than IP. The proofs in this class could be exponentially large, i.e. deterministic verifying takes exponential time (hence the class is called NEXP). In another model proposed by Fortnow, Rompel and Sipser, the prover tabulates the whole proof (which may be exponential in size) ahead of the verifier queries, and the verifier looks at the proof in only polynomially many places based on random coin tosses before accepting or rejecting it. This class is called Oracle Interactive Proofs (OIP), and was shown to be the same as MIP.

The term *Probabilistically Checkable Proofs* was used by Arora and Safra to describe the class OIP. The class PCP (r, q) consists of all languages L for which there is a randomized polynomial time verifier V which, on every input of length n, makes at most $r(n)$ coin tosses and looks at a tabulated proof in at most $q(n)$ bits. Furthermore, for each string $x \in L$, there exists a proof such that the verifier (working under the conditions described earlier) accepts with probability 1, and for each $x \notin L$, any purported proof is accepted with a probability at most 0.5. We can immediately see that PCP(0,poly) = NP (where poly is the class of polynomial functions). By the preceding discussions, it follows that PCP(poly,poly) = NEXP. This last inequality seems counter-intuitive: a proof is probed in k places, where k is only logarithmic in the total proof size. The natural queston that arose here was: Is there an analogue for NP? After a series of impressive results by a galaxy of researchers, Arora, Lund, Motwani, Sudan and Szegedy provided the crowning finale:

$$\mathrm{NP} = \mathrm{PCP}(O(\log n), c).$$

What this amounts to is the following: for any NP set A (and for no other set), for any $x \in A$, there is a way of writing a 'proof of membership' such that a randomized verifier, making $\log|x|$ random coin tosses and then probing this proof in just c places, will accept the proof to be valid. Note that c is a constant depending only on A and independent of x. On the other hand, for $x \notin A$, no matter what 'proof' is presented, a randomized verifier making $\log|x|$ random coin tosses and then probing this proof in just c places will, with probability at least half, find an error!

Connections to Non-approximability

Many natural problems are *optimization problems*: find the *largest*, or the *smallest*, subset of a given set with some desired property. All of them have corresponding decision versions too: For example, given a graph G, finding the smallest vertex cover VC in it is an optimization problem. (A vertex cover is a set of vertices such that every edge has at least one endpoint in this set.) The corresponding decision version asks, given a graph G and a number k, whether G has a vertex cover of size at most k. Similarly, finding the largest clique (a set of vertices all of which are adjacent to each other) is an optimization problem CLQ, and the decision version asks if G has a clique of size at least k. The decision versions of both VC and CLQ are NP-complete; hence no feasible algorithm is known for finding the optimal solution for these problems.

While all NP-complete decision problems are polynomially equivalent, research over the last three decades suggestes that NP-hard optimization problems can differ vastly if we are interested in computing approximate solutions. For instance, for Vertex Cover(VC), there is an efficient algorithm A which, on input graph G, produces a vertex cover of size k such that

$$\text{min-VC}\,(\mathrm{G}) \leq k \leq 2\ \text{min-VC}\,(G)$$

This algorithm is said to achieve an approximation factor $f=2$. Many NP-complete problems have a constant-factor approximation algorithm, i.e. an algorithm which, for some fixed constant $c>1$, approximates the optimal solution to within a factor c in polynomial time. Such problems are said to be in the class APX (APproXimable). However, for CLQ, no efficient approximation algorithm with a constant-factor approxmation is known.

In earlier work that set the stage for the PCP result mentioned above, Feige, Goldwasser, Lovasz, Safra and Szegedy established a major breakthrough, namely a connection between PCPs and non-approximability of optimization problems. Building on this, Arora and Safra showed that if CLQ can be approximated to any constant factor, then NP = P. Arora, Lund, Motwani, Sudan and Szegedy, while proving the PCP result, showed a somewhat different non-approximability result. An optimisation problem is said to have a polynomial time approximation scheme, PTAS, if there is an efficient algorithm which, for any input instance and any constant ε, finds a solution that is within $(1+\varepsilon)$ of the optimum. (Naturally, the run time will depend on ε.) Clearly, PTAS is contained in APX. The ALMSS result shows that several problems in APX are unlikely to be in PTAS. Further, there is a complexity class of optimisation problems called MaxSNP, defined by Papadimitriou and Yannakakis in 1991, which contains many problems in PTAS and APX. The PCP theorem implies that unless P=NP, no MaxSNP-complete problem is in PTAS. In other words, for every Max-SNP problem π, there is a threshold ε such that unless P=NP, π cannot be efficiently approximated to a factor better than $1+\varepsilon$.

Madhu Sudan had significant contributions to the development of these proofs. His doctoral thesis addressed several problems that formed essential ingredients in proving the PCP theorem. One of these was whether a given tabulated function corresponds to a low degree polynomial and self correcting a function that is close to a low degree polynomial. (This, turned out to be related to the problem of decoding certain codes.) Subsequently, he as well as other researchers have significantly sharpened the parameters of the PCP theorem, obtaining stronger non-approximability results. He has also made immense contributions to algorithmic coding theory: improved decoding algorithms, and newer applications of codes to theoretical computer science. We now describe Madhu Sudan's work in coding theory.

The Setting in Coding Theory

There are three entities in coding theory – the *sender*, the *receiver*, and a *noisy channel* over which the sender sends messages to the receiver. A *message* $\mathbf{m}$ is an element of a finite (usually large) message space M and is composed of a sequence of k symbols over a finite alphabet Σ. It is encoded using an *encoding function E* into a *codeword* of length n symbols, where n is fixed and is called the *block length* of the code. The space of all words being transmitted over the channel is the set of all n-tuples over Σ designated by Σ^n. The *code* is just the image $\{E(\mathbf{m}) \mid \mathbf{m} \in M\}$ of the encoding function. The number of coordinates in which any two n-tuples differ is called the *Hamming distance* between them. The minimum of the Hamming distances between all pairs of codewords is called the *minimum distance* of the code. One desirable property of a code is that it have large minimum distance. Another is that it be easily decodable. Thus what we want is an efficient algorithm that transforms a possibly corrupted codeword back to what was probably the transmitted message.

Linear Codes and List Decoding

A special subclass of codes, the linear codes, have been widely used in practice, principally because they possess efficient decoding algorithms. An (n, k, d) q-ary linear code is a k-dimensional subspace of an n-dimensional vector space with minimum distance d. In particular, a t-error correcting code over an alphabet F_q is a set $C \subseteq F_q^{\,n}$ such that for any received vector $\mathbf{r}$ there is at most one vector $\mathbf{c} \in C$ that lies at a Hamming distance of t from $\mathbf{r}$. Hamming showed that for a code to be able to correct t errors, its minimum distance should be at least $2t+1$. There are several algorithms available that perform this kind of unique decoding for various classes of linear codes. A question that naturally arises is: What does the decoder do when $t' > t$ errors occur? In an alternate notion of decoding proposed in the late 1950's, the decoder could output a *list* of codewords at distance t' from the received vector, the actual transmitted codeword being present in this list. This kind of decoding is called *list decoding*. The size of the list that the decoder outputs cannot be too large, as then the decoding will not be efficient. The notion of a *list decoding radius* is important here. A ball of Hamming radius t around any vector **v** contains all vectors at Hamming distance at most t from **v**. The list decoding radius defines a number e such that there are at most a polynomial number of codewords present in a ball with Hamming radius e around a received vector. It was shown that for an (n, k, d) block code this number of codewords is polynomial in n as long as the radius e satisfies $e< n- \sqrt{n(n-d)}$ So the question to be addressed was: Is there a polynomial time algorithm, which given a received vector, outputs all codewords within a radius e satisfying this bound? The first such algorithm was given by Sudan for a very important class of linear codes: the Reed–Solomon codes, briefly described below. These are used extensively for error correction on compact discs and for improving the reliability of deep space communication.

Reed–Solomon Codes

The Reed–Solomon codes have parameters $(n, k+1, n-k)$. Thus for these, the list decoding radius e is at most $n=\sqrt{kn}$ Sudan provided a polynomial time algorithm to list decode up to a radius $n=\sqrt{kn}$ The paper left a gap between performance and the combinatorial bound, which was subsequently closed in a paper by V Guruswami and Sudan in 1999. The work of Guruswami and Sudan exploited previously known insights that the decoding problem was really a 'curve fitting' problem. A Reed–Solomon code of dimension $k+1$ is specified by distinct elements x_1, $x_2 \dots x_n$ in F_q and consists of the evaluations of all polynomials of degree at most k on the points $x_1, x_2 \dots x_n$. So the list decoding problem for an $(n, k+1, d)$ code over F_q could be stated as: Given n-dimensional vectors **x** and **y**, find all univariate polynomials of degree at most k, with coefficients in F_q, such that the Hamming distance between $p(\mathbf{x})$ and **y** is less than or equal to e, where $p(\mathbf{x})= \langle p(x_1), p(x_2) \dots p(x_n)\rangle$. (In other words, the evaluations of the polynomial differ from the vector **y** in at most e points.)

There exist very clever traditional algorithms for the unique decoding problem, that interpolate **y** as a rational function of **x**. That is, they find $a(x)$ and $b(x)$ such that for all $i \in \{1, 2, \dots n\}$, $a(x_i)= y_i\, b(x_i)$. The problem is that rational functions have limited capability to extract the message when the number of errors is large. In particular if there exist two codewords that are equidistant from the vector **y** we will have two polynomials $p_1(x)$ and $p_2(x)$ such that for half the points $p_1(x_i)= y_i$ and for the other half $p_2(x_i)= y_i$. One way to get around this is to define the bivariate polynomial $Q(x,y) = (y-p_1(x))(y-p_2(x))$ which is 0 on all points (x_i, y_i) and which can be found by solving a linear system. The candidate polynomials are roots of $Q(x, y)$.

Sudan saw that this was a solution to the list decoding problem for Reed–Solomon codes. His algorithm has two phases: In the first phase it finds a polynomial $Q(x, y)$ in two variables which 'fits' the points (x_i, y_i), where fitting implies that the polynomial is 0 at all those points. In the second phase, it finds all small degree roots of $Q(x, y)$. Thus the decoding algorithm reduces to finding the bivariate polynomial and factoring it. The factors give a list of polynomials that include the candidates for the output of the list decoding algorithm. Guruswami and Sudan used more sophisticated arguments to show that the combinatorial bound could actually be met. The algorithm was further refined to work for *soft decision decoding*, where the received symbols look like a combination of several field elements with varying 'degrees of confidence'. Generalizations to algebraic geometry codes were also included in their work. The appearance of the remarkable result on algebraic list decoding was followed by a series of papers extending the concept of list decoding to several other classes of codes.

Madhu Sudan's work showed deep and unexpected connections between coding theory and theoretical computer science. It has opened up new avenues for research, and facilitated viewpoints that have greatly improved our understanding of problems in both areas.

Suggested Reading

[1] Madhu Sudan, Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems, ACM Distinguished Theses Series, *Lecture Notes in Computer Science*, Vol. 1001. Springer, 1996.

[2] Madhu Sudan, Algorithmic Issues in Coding Theory, Proceedings of the 17th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (Kharagpur, India), *Lecture Notes in Computer Science*, Vol. 1346 pp. 184–199. Springer 1997.

[3] Madhu Sudan, List Decoding: Algorithms and Applications, *SIGACT News*, (ACM Special Interest Group on Automata and Computability Theory), Vol. 31, 2000.

[4] Several course notes are available on Madhu Sudan's home page at http://theory.lcs.mit.edu/~madhu/

Meena Mahajan and Priti Shankar, Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India.

K. S. Krishnan with his family