from *SIAM News*, Volume 34, Number 10

Coding Theory Meets Theoretical Computer Science

By Sara Robinson

Theoretical computer scientists have been plying their skills in coding theory since the late 1980s, focusing primarily on the design of faster decoding algorithms. Over the last few years, techniques have begun to flow in the other direction as well, with error-correcting codes proving to be useful tools for the solution of problems in theoretical computer science.

Ample evidence of the flows in both directions appeared in the research presented at this year's IEEE Symposium on Foundations of Computer Science (FOCS), held in Las Vegas in October.

Some of the results presented at the conference shed light on the use of computer scientists' combinatorial tools to construct codes with desirable algorithmic properties. Other results apply coding theory techniques to problems of interest to computer scientists, such as proving lower bounds for algorithms and devising tools for generating random numbers.

Coding theory has proved to be such a treasure trove of interesting problems and clever tools that many theoretical computer scientists with no background in the subject are eager to learn about it.

Madhu Sudan, who has given one-week workshops in coding theory around the country, gave a two-hour tutorial on the subject at the FOCS conference. In addition, a few computer science departments have begun to offer courses in coding theory for theoretical computer scientists; Sudan is teaching such a course at the Massachusetts Institute of Technology.

For now, however, these courses are describing what is still a nascent area of research: "The more you think about [the connections], the more you understand, but there is more to understand," says Luca Trevisan, a professor of computer science at the University of California, Berkeley.

Efficient Algorithms for Coding Theory

The recent results have created a new role for the theory of reliable communication in the presence of noise, founded in the late 1940s by Claude Shannon. Another mathematician, Richard Hamming, developed some of the combinatorial underpinnings of error-correcting codes, a major piece of coding theory, at about the same time.

An error-correcting code is a way of adding redundancy to information so that it can be recovered even if some of it is corrupted in transmission. Formally, an error-correcting code is defined as a collection of strings of length n over a finite alphabet, called "code words." Information is transformed into a sequence of code words via an "encoder" algorithm, and then sent across a noisy channel.

At the other end is a "decoder," an algorithm that takes an n-tuple as input, identifying it—if it's not one of the code words—with the most likely code word. Clearly, a good property for a code to have is that most pairs of code words in the collection disagree on many coordinates.

The rate R of a code, defined by

$$R = \frac{\log_2\left(|C|\right)}{n},$$

is a measure of the amount of information the code transmits per bit over the channel. Shannon showed that every channel has a capacity C_0, which is the maximum rate of information it can reliably transmit.

Shannon's main theorem says that there exist sequences of error-correcting codes with rates approaching the capacity of the channel, such that the probability the data will not be correctly decoded goes to zero exponentially in n. The theorem, whose proof is based on a set of randomly chosen code words, shows that random codes are likely to disagree on enough coordinates to make decoding possible with a very small probability of error.

Being close to channel capacity and having a small probability of error, however, aren't enough to make a code useful. It must also be possible to encode and decode data quickly.

Over the years, isolated groups of coding theory researchers looked at the computational aspects of error-correcting codes, beginning with a paper by Robert Gallager of MIT in the 1960s. But only with the technological revolution of the 1990s did computational issues come to the forefront of code design. During that time, French engineers devised Turbo codes, which have extremely fast (linear-time) encoding and decoding algorithms and, in practice, correct a large fraction of errors. At the time, however, rigorous analysis of the performance of these algorithms eluded researchers.

Naturally, the new algorithmic focus of coding theory caught the attention of the theoretical computer science research community. In 1997, Sudan devised an algorithmic paradigm for "list-decoding," a notion that dates back to the 1950s. A list-decoding algorithm reports a small list of code words, rather than just one code word, that could potentially be decodings of a corrupted transmission. Such a list, along with some extra knowledge about the message being transmitted, typically allows for the efficient recovery of the actual code word being transmitted. Usually, list-decoding algorithms recover from many more errors than standard decoding algorithms.

At about the same time, Michael Sipser and Daniel Spielman of MIT used expander graphs, a combinatorial tool widely used in theoretical computer science, to construct codes whose decoding efficiency is intricately intertwined with their ability to error-correct. The (linear-time) decoding algorithm for "Expander codes" is an essential part of the proof that the codes successfully correct errors.

While Expander codes were significant from a theoretical standpoint, they were not practical, since their error-correction capabilities are not competitive with those of existing codes. A new type of code, designed by Michael Luby, Michael Mitzenmacher, Amin Shok-rollahi, and Spielman, who was then at the International Computer Science Institute in Berkeley, used general bipartite graphs instead of expanders. The distribution of the degrees of the vertices in the graphs used to construct these "Tornado codes" determines the fraction of erasure errors they can correct. Because randomized algorithms are used to construct the codes, they have desirable error-correction properties only with high probability. By contrast, Expander codes, along with the algebraic codes developed in the 1950s, such as Reed–Solomon codes, consistently correct a certain fraction of errors.

The development of Tornado codes started a revolution in code design that carried over to the traditional coding theory community. It brought with it techniques that made possible rigorous analysis of Turbo codes and other codes used frequently in practice. At the same time, Expander and Tornado codes revived connections between coding theory and combinatorial graph theory, a connection that researchers continue to explore.

In a paper presented at the recent FOCS conference, Noga Alon of Tel-Aviv University, and Alexander Lubotsky and Avi Wigderson of the Hebrew University, showed a connection between the classic semi-direct product groups and a graph construction called the "zig-zag product," which provides a way of building good expanders. The authors describe their paper as "another step in the long chain of work attempting to understand and construct expander graphs," which have many applications in computer science theory, including code construction.

Another paper at the conference, by Venkatesan Guruswami and Piotr Indyk of MIT, described the use of expander graphs to construct codes. They have been able to construct both unique and list-decodable codes that achieve either rates similar to those of existing codes with faster encoding and decoding algorithms, or better rates with similar error-correction capabilities.

Codes Applied to Lower Bounds and Randomness

Even before ideas from graph theory were applied to code construction, tools from coding theory were beginning to play a role in a few results in theoretical computer science, particularly in determining lower bounds on the time required by algorithms for certain tasks. In a paper published in *SIAM Journal on Computing* in 1989, Nader Bshouty used elementary coding theory to find a lower bound for the classic problem of matrix multiplication. Using other techniques, Markus Blaser improved on that bound two years ago.

At the recent FOCS conference, Amir Shpilka of the Hebrew University returned to coding theory to produce a significant improvement on Blaser's result for matrices over finite fields. In his proof, Shpilka shows that the intermediate computations of the algorithm essentially define a code with many code words, such that the code words differ in many coordinates. Standard techniques from coding theory show that any such code must have long code words, which translates in Shpilka's work to the requirement that the algorithm have many intermediate steps in its computation, and thus include many multiplications.

Most recently, error-correcting codes have proved useful for generating sources of randomness. Although the world is full of randomness, it's hard to find usable sources of truly random numbers. For computational purposes, then, theorists have devised tools that stretch small, truly random numbers into larger numbers that are computationally indistinguishable from random.

A connection between codes and such tools became evident two years ago in a paper by Sudan, Trevisan (then at Columbia), and Salil Vadhan (then at MIT). They showed that error-correcting codes can be used to construct a new type of pseudo-random number generator, or PRG, the tool typically used in computers to generate long random numbers for randomized algorithms. A PRG takes short strings of truly random bits as input and, by way of a hard-to-compute function, stretches them into much longer strings that, by any efficient test, are indistinguishable from random strings.

In the Sudan–Trevisan–Vadhan construction, error-correcting codes were used to "amplify" a function's hardness: Suppose that a finite function, viewed as a vector of values, forms a code word of a nice code. If the function is easy to compute on randomly chosen inputs, then, by error-correction, its value on every input can be determined exactly, and the function is easy to compute on all inputs.

The contrapositive says that if a function is a code word and is hard to compute on some input, then it is hard to compute on most inputs. From a function that is hard to compute on most inputs, it is possible, in turn, to derive a string that is computationally indistinguishable from a random string.

Later that year, Trevisan showed that codes could be used to convert a PRG into a tool of a different type for obtaining long random strings. The new tool, known as an "extractor," takes as input a string from a source of weakly random bits (bits with some minimum entropy), along with a much shorter string of truly random bits. Its output is a distribution statistically indistinguishable from uniform. Ideally, the length of the output is close to the minimum entropy of the weakly random source.

At a conference last summer, Amnon Ta-Shma of Tel-Aviv University and David Zuckerman of the University of Texas at Austin showed how extractors can be used to build codes with good list-decoding properties. And, at the recent FOCS conference Ta-Shma and Zucker-man, with Shmuel Safra of Tel-Aviv University, described the construction of extractors from Reed–Muller codes, one of a class of well-known, commonly used error-correcting codes based on polynomials.

In the Ta-Shma–Zuckerman–Safra construction, the extractor uses the weakly random source to choose multivariate polynomials, the encoding mechanism for Reed–Muller codes. The truly random source is used to choose a series of points on which the polynomials can be evaluated so that the result is statistically close to uniform.

Certain properties of the codes translate into corresponding properties for the extractors, Zuckerman points out. The minimum distance between code words for the Reed–Muller codes, for instance, has a simple inverse relationship to the collision probability of the output of the extractor.

Another FOCS paper built on this construction, substantially improving the parameters of the extractor. This paper—by Ronen

Shaltiel of Hebrew University and Christopher Umans of Microsoft Research—also shows how Reed–Muller codes can be used to build a new type of pseudo-random number generator.

The recent flurry of research connecting codes and sources of randomness, on top of the earlier links between codes and combinatorial objects, seems to hint at the possibility of a broader theory, which researchers are only beginning to elucidate. "It's certainly at a point where codes are indispensable as a tool," says Umans, who considers any further comments at this point premature.

But for Sudan, the connections found so far suggest that something deeper is going on. "For some mysterious reason, every notion from extremal combinatorics that computer scientists have gotten entangled with turned out be some form of error-correcting code," he says. "In many cases, the mere realization that a problem can be related to coding theory was the solution."

Sara Robinson is a freelance writer and part-time journalist-in-residence at the Mathematical Sciences Research Institute at Berkeley.