The Renaissance of Gallager's Low-Density Parity-Check Codes

Tom Richardson, Flarion Technologies
Rüdiger Urbanke, EPFL

ABSTRACT

LDPC codes were invented in 1960 by R. Gallager. They were largely ignored until the discovery of turbo codes in 1993. Since then, LDPC codes have experienced a renaissance and are now one of the most intensely studied areas in coding. In this article we review the basic structure of LDPC codes and the iterative algorithms that are used to decode them. We also briefly consider the state of the art of LDPC design.

INTRODUCTION

The year 1993 marks an important paradigm shift in the realm of communications — C. Berrou, A. Glavieux, and P. Thitimajshima presented their approach to error correcting coding, termed *turbo coding* [1], to the world. Coding theory, after 50 years of steady but slow progress, had suddenly made a giant leap: the new technique, with complexity only a small factor larger than that of standard coding schemes like convolutional codes, enables performance approaching the Shannon capacity of the additive white Gaussian noise channel within a fraction of a dB. This was staggering progress: the power required for reliable transmission with convolutional codes, compared to Shannon's limit, is typically *twice* as large (3 dB more). The (communications) world was in commotion.

As the research community turned its attention to turbo codes, it slowly emerged [2, 3] that the original seed for this revolution had been planted much earlier and, since then, lain largely dormant. In 1960 R. Gallager completed his Ph.D. thesis [4], *Low-Density Parity-Check Codes*. In this remarkable thesis Gallager introduced at least two lasting concepts: a powerful bounding technique to assess the maximum-likelihood performance of coding systems, and *low-density parity-check* (LDPC) codes together with their associated *iterative decoding algorithm*. The latter concept is the topic of this article.

Recent years have seen significant advances in our understanding of, and ability to design, iterative coding systems (see the article by Benedetto *et al.* in this issue and [5, 6]). Moreover, it is now apparent that virtually all aspects of the communications chain can be included in the iterative processing framework: source coding, channel coding, modulation, equalization, multiple access, transmission via multiple antennas, and so on — all of these areas are currently being attacked by iterative signal processing techniques.

Iterative decoding of LDPC codes is the archetypal example for illustrating the principles of iterative signal processing. This is mainly due to the simplicity of LDPC codes, which renders the key ideas particularly transparent. For the same reason, most advances in our understanding of and ability to analyze iterative systems have taken place in the context of LDPC codes. Not surprisingly, from this ability to predict and analyze the system also grows unprecedented performance. Recent design improvements and hardware realizations of LDPC codes, exploiting their underlying simplicity, have produced coding systems that match or outperform turbo codes while requiring lower complexity. Figure 1 presents simulation results for rate 1/2 LDPC codes with information block lengths $k = 512$, 1024, 2048, and 4096, respectively. The decoder is a software implementation of belief propagation. These codes represent state-of-the-art designs. For all of these codes the Tanner graph has only 3.3 edges per codeword bit, implying very low decoding complexity. Observe the dramatic improvement in the performance for increasing block lengths. To give some further evidence of the promise of LDPC codes, it has been shown that, in the limit of infinite block lengths, reliable communication within 0.0045 dB of the Shannon limit is possible, and simulations of a block length 10^7 code performed within 0.04 dB of the Shannon limit at a bit error rate of 10^{-6} [7].

So what is it that makes iterative signal processing such an attractive candidate for much of communications, and how does it differ from the classical approach?

Consider the case of error correcting coding. Much of the effort in coding over the last 50 years has focused on the construction of highly structured codes with large minimum distance. The structure keeps the decoding complexity manageable, while large minimum distance is supposed to guarantee good performance. This approach, however, is not without its drawbacks. First, ironically, it seems that finding structured codes with large minimum distance is a harder problem than the problem we started out with (reliably communicating close to capacity)! Finding good codes in general is, in a sense, trivial: randomly chosen codes are good with high probability. More generally, one can easily construct good codes provided one admits sufficient *description complexity* into the definition of the code. This conflicts, however, with the goal of finding highly structured codes that have simple decodings. Second, close to capacity, minimum distance is only a poor surrogate for the performance measure of real interest. Iterative coding systems take an entirely different approach. The basic idea is the following. Codes are constructed so that the relationship between their bits (the structure of their redundancy) is *locally* simple, admitting simple *local decoding*. The local descriptions of the codes are interconnected in a complex (e.g., random-like) manner, introducing long-range relationships between the bits. Relatively high global description complexity is thereby introduced in the interconnection between the simple local structures. Iterative decoding proceeds by performing the simple local decodings and then exchanging the results, passing messages between locales across the "complex" interconnection. The locales repeat their simple decodings, taking into account the new information provided to them from other locales. Usually, one uses a graph to represent this process. *Locales* are nodes in the graph, and interconnections are represented as edges. Thus, description complexity is introduced without adding computational complexity per se, but rather as wiring or routing complexity.

Complexity in iterative decoding has three parts: first, there is the complexity of the local computations; second, the complexity of the interconnection (i.e., the routing of information); third, the number of times the local computations need to be repeated, usually referred to as the number of iterations. All of these are practically manageable.

The above principle generalizes to a much wider setting. The method of iterative signal processing is to express the dependencies among the various components in the (communications) system using a suitably chosen graph and to distribute signal processing analogously: information is sent along the edges of the graph and compiled (processed) locally at each node of the graph. In order to facilitate the subsequent iterative processing, one tries to keep the graph as sparse (low-density) as possible. Although such an approach is easily seen to be suboptimal in general, it is usually quite close to optimal and has an excellent complexity vs. performance trade-off. In this way, one can often design systems that are inherently very complex and completely unamenable to a direct maximum likelihood approach. The general framework of graphical representations is known as the *factor graph* approach [8].

What catapulted iterative coding systems, and error correcting in particular, into the limelight was the empirical observation that, for those codes for which this graphical representation is sufficiently sparse, this suboptimal iterative algorithm performs remarkably close to an optimal maximum likelihood decoder. Furthermore, there is an abundance of codes that are "good," allowing reliable transmission close to capacity, and have sparse graphical representations, rendering them amenable to an iterative decoding approach. It is fortunate that sparseness helps performance (of iterative schemes), since sparseness also reduces complexity.

In the sequel we will concentrate on the LDPC (de)coding problem. We will first describe the graphical visualization, which for the case of LDPC codes is known as *Tanner graphs*, named after Michael Tanner, one of a handful of researchers who worked on iterative methods prior to the discovery of turbo codes. We will then describe the two major classes of iterative decoding algorithms commonly used — one class containing all *flipping* algorithms and the second class, somewhat more complex, containing all *message-passing* algorithms. It is noteworthy that the most powerful message-passing algorithm is also known as the *belief propagation* algorithm and that standard turbo decoding is an instance of belief propagation. We will conclude with some remarks on the encoding problem and a short state-of-the-art section addressing hardware implementation.

For readers interested in technical aspects of iterative coding systems beyond what we can present in this very brief overview, we highly recommend the February 2001 *IEEE Transactions on Information Theory* special issue on codes on graphs and iterative algorithms.

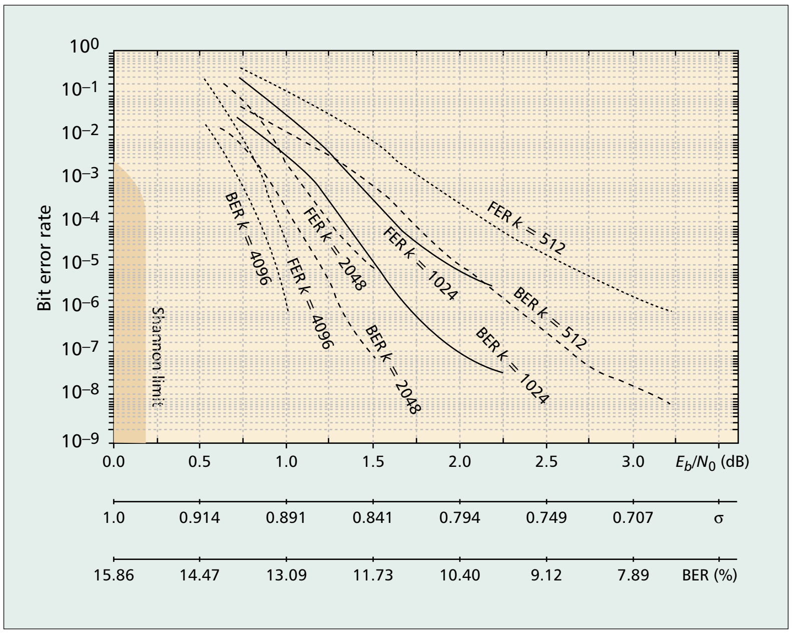

Figure 1. *Frame error rate (FER) and bit error rate (BER) curves for rate 1/2 LPDC codes over the additive white Gaussian noise channel under belief propagation decoding. The performance improves significantly with increasing block length. The second axis shows the performance as a function of the standard deviation σ of the noise, assuming that the signal has unit energy. The third axis, finally, is labeled by the corresponding raw input bit error probability.*

$$H := \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & \mathbf{0} & 0 & 0 \\ 2 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 3 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \mathbf{1} & 1 & 1 \\ 4 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & \mathbf{1} & 1 & 1 \\ 5 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & \mathbf{0} & 1 & 1 \end{array}$$

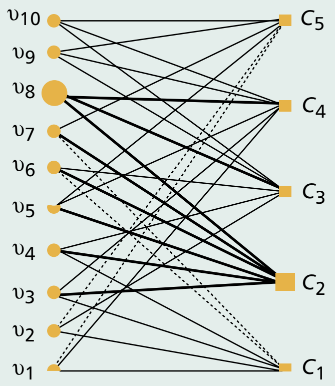

Figure 2. *A parity-check matrix **H** and the corresponding Tanner graph. To illustrate the relationship more clearly, column 8 and row 2 of **H** are shown in bold. The corresponding variable node and check node as well as the attached edges are emphasized as well.*

Graphical Representation of LDPC Codes: Tanner Graphs

Tanner graphs of LDPC codes are bipartite graphs with variable nodes on one side and constraint nodes on the other. Each variable node corresponds to a bit, and each constraint node corresponds to one parity-check constraint on the bits defining a codeword. A parity-check constraint applies to a certain subset of the codeword bits, those that participate in the constraint. The parity-check is satisfied if the XOR of the participating bits is 0 or, equivalently, the modulo 2 sum of the participating bits is 0. Edges in the graph attach variable nodes to constraint nodes indicating that the bit associated with the variable node participates in the parity-check constraint associated with the constraint node. A bit sequence associated with the variable nodes is a codeword if and only if all of the parity-checks are satisfied.

The Tanner graph representation of the LDPC code closely mirrors the more standard parity-check matrix representation of a code. In this latter description the code is represented as the set of all binary solutions $x = (x_1, x_2, \ldots, x_n)$ to a simple linear algebraic equation $Hx^T = 0^T$. The elements of the parity-check matrix are 0s and 1s, and all arithmetic is modulo 2, that is, multiplication of x by a row of H means taking the XOR of the bits in x corresponding to the 1s in the row of H. The connection between the parity-check matrix representation and the Tanner graph is straightforward, and is illustrated in Fig. 2 by means of an example. The elements of x are in one-to-one correspondence with the variable nodes in the Tanner graph. Thus, the *variable* nodes correspond to the *columns* of H. The parity checks on x are in one-to-one correspondence with the constraint nodes in the Tanner graph. Thus, the *constraint* nodes corresponds to the rows of H. The *edges* in the Tanner graph correspond to the 1s in H, that is, the entry in the ith row and jth column of H is a 1 if and only if the ith constraint node is connected to the jth variable node in the Tanner graph.

We hope that the previous discussion has made clear why LDPC codes are parity-check codes. It is also easy to explain where the term low-density comes from. For many linear codes the number of 0s and 1s in the matrix H are approximately the same. For LDPC codes the number of 1s is very small compared to the number of 0s: The matrix H has a *low density* of 1s. Equivalently, the Tanner graph of the code has a low density of edges. The importance of this cannot be overstated. The complexity of message-passing decoding of LDPC codes depends directly on this edge density so that a concerned designer of LDPC codes will try to keep this density *as low as possible*. How low can this density be made? Assume we would like to transmit at a fraction $1 - \delta$ of capacity, where δ is some small positive constant. It is then known [4] that the number of ones in the parity-check matrix has to scale at least like $n\ln(1/\delta)$ as a function of δ, as δ approaches zero, where n is the block length of the code. There is empirical evidence that such a slow growth is indeed achievable. As we discuss in some more detail in the next section, iterative decoders work by performing several (simple) decoding rounds, and the required *number* of such rounds is conjectured to grow like $1/\delta$. Therefore, the *total* complexity of iterative decoders *per bit* is conjectured to scale like $1/\delta \ln (1/\delta)$, see [9]. This is in stark contrast to classical coding systems which typically suffer from an *exponential* increase in complexity as a function of $1/\delta$ as we are approaching capacity. This explains why iterative coding systems are such a promising choice for reliable transmission close to Shannon capacity.

Iterative Decoding of LDPC Codes

The Tanner graph captures the dependency structure of the various bits. The iterative decoding algorithms we will discuss work directly on this bipartite graph. Iterative algorithms for LDPC codes come in two distinct flavors, message-passing and flipping algorithms, both introduced by Gallager [4].

Message-Passing Decoders

In *message-passing* decoders, messages are exchanged along the edges of the graph, and computations are performed at the nodes, as shown in Fig. 3. Each message represents an estimate of the bit associated with the edge carrying the message. These decoders can be understood by focusing on one bit as follows. Suppose the bits of an LDPC codeword are transmitted over a communications channel and, during transmission, some of them are corrupted so that a 1 becomes a 0 or vice versa. Each bit node in the decoder gets to see the bit that arrived at the receiver corresponding to the one that was transmitted from the equivalent node at the transmitter. Imagine that the node would like to know if that bit is in error or not and therefore asks all of its neighboring check nodes what they think the bit's value should be. Each neighboring check node then asks their other neighbors what their bit values are and sends back to the original bit node the modulo 2 sum of those values.

The bit node now has several opinions as to the bit's correct value and must somehow reconcile these opinions; it could, for example, take a majority vote. The above procedure constitutes *one round* or *iteration* of message passing.

In order to improve performance, a further such round can be performed. In more detail, assume that those bit nodes that send their observed bits to the checks to be summed first ask their check node neighbors what they think is their correct bit value. Those check nodes could then query their other variable node neighbors and forwarded their modulo 2 sum as an opinion. With more information now available, the bit nodes would have a better chance of communicating the correct value to the check nodes, and the opinions returned to the original node would therefore have a better chance of being correct. This gathering of opinions could obviously be extended through multiple iterations; typically, many iteration rounds are performed.

In actual decoding all nodes decode concurrently. Each node gathers opinions from all its neighbors and forwards to each neighbor an opinion formed by combining the opinions of the other neighbors. This is the source of the term *message passing*. The process continues until either a set of bits is found that satisfies all checks or time runs out. The message passing may proceed asynchronously. Note that with LDPC codes, convergence to a codeword is easy to detect since one need only verify that the parity checks are satisfied.

In the above example, opinions about a bit were expressed as a binary value for that bit. Much better decoding is possible if the opinions are expressed as probabilities. If the transmitted bits are received in error with probability p, the probability that the observed bit is correct is $1 - p$. If a check node forms a modulo 2 sum of k of these bits, the probability that the sum is correct is $(1+(1 - 2p)^k)/2$. Thus, the opinions returned from the check nodes have a different probability of being correct than those coming from the channel. If the bit nodes properly take these probabilities into account when combining the opinions, better performance results. In the *belief propagation* algorithm the nodes assume that all incoming probabilities are independent and then combine them by applying the rules of probability.

Computation Trees

A convenient way to visualize decoding is to trace back the origins of a particular message in the decoding and unwrap the Tanner graph in the process. Consider a particular variable-to-check-node message in the kth iteration of the decoding process. That message is a function of the channel observation associated with the variable node and the check-to-variable-node messages arriving at the variable node along other edges in the same iteration. Those messages, coming from neighboring check nodes, in turn are functions of variable-to-check-node messages in the $(k - 1)$th iteration along other edges attached to those neighboring check nodes. By unwrapping the Tanner graph to capture this traceback while ignoring any repetitions of nodes or edges that might occur, one arrives at the *computation tree* depicted in Fig. 4.

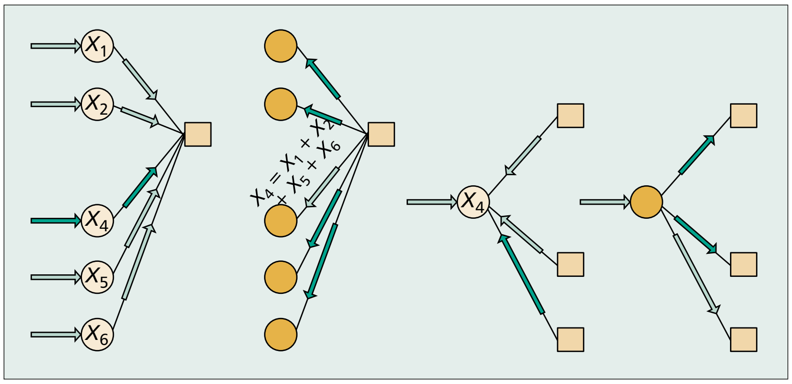

Figure 3. *The principle of a message-passing decoder. Messages represent an estimate of the bit associated with the edge carrying the message. Nodes query neighboring nodes to collect opinions, process this information, and forward their current estimate to their neighbors.*

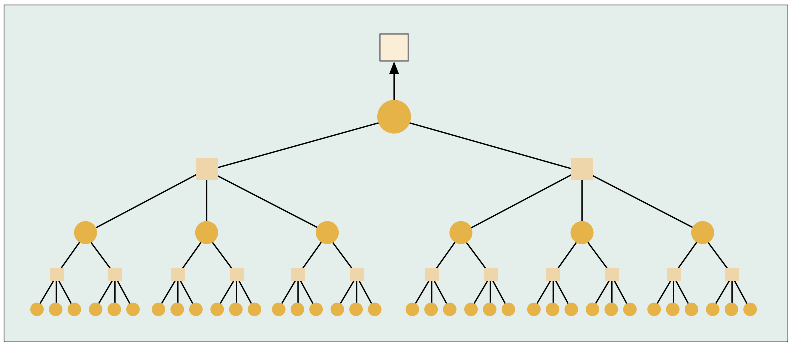

Figure 4. *A computation tree for a variable-to-check message in the third iteration. Information flows up the tree.*

Now, in the case of the belief propagation decoder the meaning of the upgoing message from the root of the computation tree is easy to describe: it represents the probability distribution of the bit associated with the root variable node conditioned on the entire computation tree provided there are no repetitions in the tree.

In practice, computation trees will have repetitions, so the belief propagation decoder will be suboptimal in general. Computation trees arising from larger Tanner graphs will tend to have fewer repetitions in them. This largely accounts for the significant improvement in performance of these codes as the block length increases.

Virtually all other practical message-passing decoders can be viewed as approximations of belief propagation, and the messages try to convey some estimate of the posterior distribution of the bits. Although it is less obvious, the same reasoning applies to turbo codes.

Flipping Algorithms

There is a second distinct class of decoding algorithms that is often of interest for very-high-speed applications, such as optical networking. This class of algorithms is known as flipping algorithms; see [10, 11] for some analytical results. They generally have even lower complexity than message-passing algorithms, albeit at the cost of somewhat worse performance.

The reason flipping algorithms have lower complexity is that, in effect, each edge outgoing from a node carries the same message. Further-

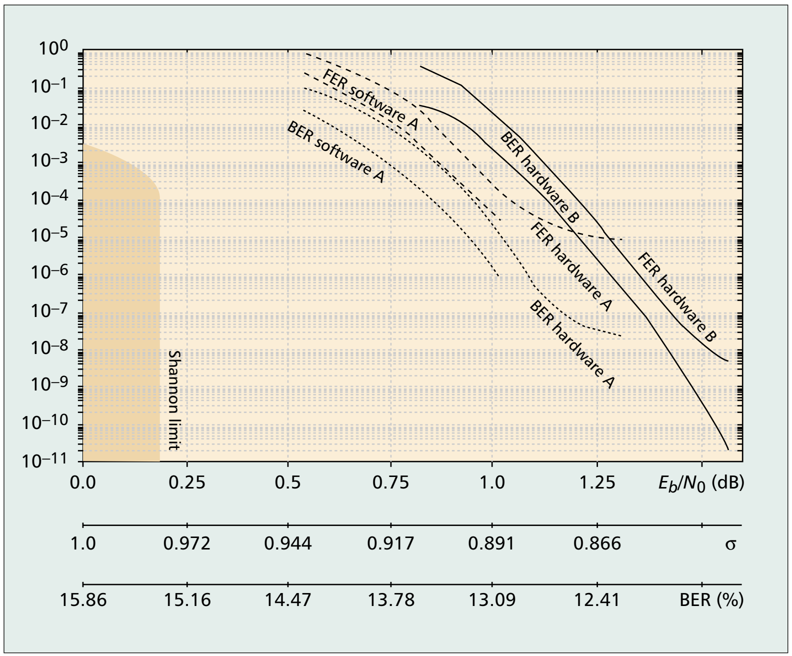

Figure 5. *FER and BER curves for rate 1/2 LDPC codes over the AWGN decoded using belief propagation and a 5-bit message passing algorithm. Code B was designed for deeper error floor, illustrating the trade-off.*

more, although it is possible to incorporate soft information, bit flipping usually operates on hard decisions: the information exchanged between neighboring nodes in each iteration is a single bit. The basic idea of flipping is that each bit, corresponding to a variable node assumes a value, either 0 or 1, and, at certain times, decides whether to flip itself (i.e., change its value from a 1 to a 0 or vice versa). That decision depends on the state of the neighboring check nodes under the current values. If enough of the neighboring checks are unsatisfied, the bit is flipped. The notion of "enough" may be time-varying, and it may also depend on soft information available about the bit. Thus, under flipping, variable nodes inform their neighboring check nodes of their current value, and the check nodes then return to their neighbors the parity of their values. The underlying assumption is that bits that are wrong will tend to have more unsatisfied neighbors than bits that are correct.

For the above reasons, flipping algorithms are attractive candidates for first-generation hardware implementations for those applications that require extremely high throughputs. Since the flipping algorithm only concerns the processing at the *receiver*, later generations of the hardware implementation can be changed to better performing (but higher complexity) message-passing decoders, keeping the code itself unchanged.

ENCODING

Turbo codes are usually defined by their *encoding* process. LDPC codes, on the other hand, are defined from the perspective of their *decoding* process. Roughly speaking, encoding LDPC codes proceeds as follows. First, the information bits are placed on certain variable nodes. Second, the values for the remaining variables are determined so that all of the parity check constraints are satisfied. Given a Tanner graph, the problem of designing an encoder boils down to selecting the nodes on which to place the information bits and preparing a method to efficiently solve for the values of the other nodes. On the surface it appears that encoding an LDPC code will involve solving the parity-check equation and that encoding complexity will therefore be rather large: quadratic in the block length. In actuality, however, efficient encoding of LDPC codes is always feasible, and the encoding complexity is a small fraction of the decoding complexity.

First, general techniques that exploit the sparseness of the parity-check matrix have been developed to address this problem [12]. Another approach is to impose structure on the Tanner graph so that encoding is transparent and simple. Repeat-accumulate codes are a good example of such structured graphs, [13]. Further examples are codes whose Tanner graphs possess some regular structure [14].

STATE OF THE ART

Although we have not discussed them in this article, there are by now fairly sophisticated tools for designing LDPC codes. A design consists, in general, of two components:
- The choice of structure
- The placement of the edges

By *structure* we mean the broad description of the Tanner graph: the degrees of the various nodes, restrictions on their interconnections, whether variable nodes are punctured, and so on. Gallager considered only *regular* graphs where all variable nodes and all check nodes have the same degree. Better performance can be realized by allowing *irregular* codes, containing nodes of various degrees. A further improvement can be achieved by introducing more elaborate structures (i.e., constraining the interconnections on a broad level). We have introduced a framework, called *multi-edge type LDPC*, for this purpose. These advanced structures allow a finer and wider control of key features, such as very low error floors, better interaction with overall iterative schemes, or the construction of efficient LDPC codes of very low rate. The simulation results presented in this article employ multi-edge type LDPC designs.

There is a typical trade-off that one encounters when designing LDPC codes. This trade-off is illustrated in Fig. 5. For a given structure, complexity, and block length, one can either push the waterfall portion of the error curve as far as possible toward the Shannon limit, or one can back off slightly and aim for very low error floors. The performance curve of the code with $k = 4096$ from Fig. 1 is repeated here, labeled code A. Its BER curve is fairly close to the Shannon limit, but it exhibits a floor around 10^{-7}. A hardware simulation, performed on the Flarion FPGA implementation using a 5-bit approximation of belief propagation, of the same code is shown as well. It performs about 0.1 dB worse. This difference is due to quantization of the input and in the decoder and a lower maximum iterations cap. As a second example, the performance curve of an alternative code, labeled code B, is given. It performs about 0.2 dB worse at high BER values but has a much deeper floor, around 10^{-11}.

The second part of the design process, the placement of edges, requires some care to get the best possible performance. Moreover, hardware complexity can depend strongly on how this is done. In general, it is possible to produce a very-low-complexity LDPC decoder implementation that supports the best performing designs. Producing those designs, though, requires some expertise.

The processing required for message-passing decoding LDPC codes is highly parallelizable and flexible. Each message-passing operation performed at a node depends on other nodes only through the messages that arrive at that node. Moreover, message updating need not be synchronized. Consequently, there is great freedom to distribute in time and space the computation required to decode LDPC codes. Turbo codes are also decoded using belief propagation, but their structure does not admit the same level of atomization.

Practical Implementations

Turbo codes have already made their presence felt in practical applications and are part of third-generation (3G) wireless standards (see the article by C. Berrou in this issue). In this game LDPC codes (re)appeared slightly late and thus missed their inclusion into current standards. (During the final preparations of this article, however, an LDPC based solution was adopted for the latest DVB satellite communications standard.) It is possible that the current delay in the deployment of 3G systems will allow the design and introduction of LDPC-based coding systems. Several hardware implementations for a wide array of applications already exist, and we mention in the sequel those of which we are aware. (See also the articles by Yeo *et al.* and Eleftheriou *et al.* in this issue.)

- Flarion Technologies has implemented a programmable LDPC decoder. The hardware supports a wide range of LDPC designs. The message-passing algorithm used is a 5-bit approximation of belief propagation. A version of the decoder is working in the Flarion wireless system. Encoding is done on a DSP and requires less than one clock cycle per codeword bit.
- Digital Fountain was founded by a group of researchers working on LDPC codes. This group contributed key results in the analysis of LDPC codes for the erasure channel [15]. The company is using LDPC-like codes to provide efficient and reliable content delivery over the Internet.
- Lucent Technologies has implemented an LDPC code for optical networking. The device runs at 10 Gb/s throughput with a coding rate of 0.93 and target error performance 10^{-15}.
- LDPC codes have been proposed to the Consultative Committee for Space Data Systems (CCSDS) by the Jet Propulsion Laboratory.
- Agere has also implemented an LDPC code for optical networking. Howland and Blanksby implemented a fully parallel rate 1/2 block length 1024 LDPC code to demonstrate the potential of LDPC codes for low-power high-speed applications. The device has a 1 Gb/s throughput while consuming only 200 mW of power.
- The storage industry has also shown serious interest in incorporating LDPC codes into future-generation devices (see the article, and references therein, by Dholakia *et al.* in this issue), but no products have been announced to date. Only time will tell.

Since the flipping algorithm only concerns the processing at the receiver, later generations of the hardware implementation can be changed to better performing (but higher complexity) message-passing decoders, keeping the code itself unchanged.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding," *Proc. ICC '93*, Geneva, Switzerland, May 1993, pp. 1064–70.

[2] D. J. C. MacKay and R. M. Neal, "Good Codes Based on Very Sparse Matrices," *Cryptography and Coding. 5th IMA Conf.* C. Boyd, Ed., *LNCS*, no. 1025, Berlin: Springer, 1995, pp. 100–11.

[3] N. Wiberg, "Codes and Decoding on General Graphs," Ph.D. thesis, Linköping Unive., S-581 83, Linköping, Sweden, 1996.

[4] R. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Info. Theory*, vol. 7, Jan. 1962, pp. 21–28.

[5] T. Richardson and R. Urbanke, "The Capacity of Low-Density Parity Check Codes Under Message-Passing Decoding," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 599–618.

[6] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 619–37.

[7] S. Chung *et al.*, "On the Design of Low-Density Parity-Check Codes Within 0.0045 db of the Shannon Limit," *IEEE Commun. Lett.*, vol. 5, Feb. 2001, pp. 58–60.

[8] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 498–519.

[9] A. Khandekar and R. J. McEliece, "On the Complexity of Reliable Communication on the Erasure Channel," *Proc. Int'l. Symp. Info. Theory*, Washington, DC, 2001, p. 1.

[10] V. Zyablov and M. Pinsker, "Estimation of the Error-Correction Complexity of Gallager Low-Density Codes," *Problemy Peredachi Informatsii*, vol. 11, Jan. 1975, pp. 23–26.

[11] M. Sipser and D. Spielman, "Expander Codes," *IEEE Trans. Info. Theory*, vol. 42, Nov. 1996, pp. 1710–22.

[12] T. Richardson and R. Urbanke, "Efficient Encoding of Low-Density Parity-Check Codes," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 638–56.

[13] D. Divsalar, H. Jin, and R. McEliece, "Coding Theorems for 'Turbo-Like' Codes," *Proc. 1998 Allerton Conf.*, 1998, pp. 201–10.

[14] J. Hu *et al.*, "A General Class of LDPC Finite Geometry Codes and Their Performance," *IEEE Int'l. Symp. Info. Theory*, Lausanne, Switzerland, June 30–July 5, 2002, p. 309.

[15] M. Luby *et al.*, "Efficient Erasure Correcting Codes," *IEEE Trans. Info. Theory*, vol. 47, Feb. 2001, pp. 569–84.

Biographies

RÜDIGER L. URBANKE (ruediger.urbanke@epfl.ch) received the Diplomingenieur degree from the Vienna Institute of Technology, Austria, in 1990, and M.S. and Ph.D. degrees in electrical engineering from Washington University, St. Louis, Missouri, in 1992 and 1995, respectively. From 1995 to 1999, he held a position at the Mathematics of Communications Department at Bell Laboratories. Since November 1999, he has been on the faculty of the Swiss Federal Institute of Technology, Lausanne. He is a recipient of a Fulbright Scholarship and since October 2000 has been an Associate Editor of *IEEE Transactions on Information Theory*. He is a co-recipient of the IEEE Information Theory Society 2002 Best Paper Award.

THOMAS J. RICHARDSON (tjr@flarion.com) is currently chief scientist at Flarion Technologies. From 1990 to 2000, he was a member of the Mathematical Sciences Research Center in Bell Laboratories. He is a co-recipient of the IEEE Information Theory Society 2002 Best Paper Award. He received a Ph.D. in electrical engineering from the Massachusetts Institute of Technology in 1990, and Bachelor's and Master's degrees, also in electrical engineering, from the University of Toronto.