

GRAPH-BASED CODES

NIGEL BOSTON

ABSTRACT. This is a mini-course on graph-based codes, given at the Center for Theoretical Sciences at Taipei, Taiwan, July 12-15, 2004. In practical coding theory, the main challenge has been to find codes with rates close to channel capacity and with efficient encoding and decoding algorithms. Only recently was this accomplished, theoretically for the binary erasure channel but only experimentally for the binary symmetric and AWGN channels. The classes of codes are related to graphs with message passing along edges providing the decoding. Analyzing them and designing them leads to interesting mathematics, much of it still unresolved.

0. Caveat

It is important for the mathematician approaching this material to come at it with the right mindset. As mathematicians we often seek to impose what sort of problem we wish to tackle. Here in dealing with the fundamental problems that real-life channels throw up, we do not enjoy that luxury. In particular, note:

(1) Decoding algorithms matter. In fact, here they become the crucial issue, and rather than skip them as engineering rather than math, they should be viewed as a springboard towards new and interesting mathematical challenges.

(2) Low density parity-check (LDPC) and similar codes are currently a very hot topic. Every new ISIT meeting or issue of IEEE Transactions in Information Theory typically has more on this topic than any other (with the exception of special issues, and with space-time codes, list decoding, and quantum error-correcting codes running it close).

(3) The minimum distance of a code does not play as crucial a role as it does in classical coding theory. It is just one measure of how good a code is and not the best one at that. A few low-weight codewords may be tolerable. On the other hand, whereas low-complexity bounded distance decoders abound, a large minimum distance does not guarantee transmission at rates close to capacity.

1. History

In 1948 Shannon set the bar [21]. He showed that each communication channel has a capacity, i.e. a rate below but not above which reliable transmission is possible. He proved the existence of codes of rates arbitrarily close to capacity for which probability of error using Maximum Likelihood Decoding (MLD) goes

The author thanks the CTS and Winnie Li for their kind hospitality during his visit to Taiwan and NSF DMS-0300321 for support in completing this report.

to zero (in fact exponentially fast) as block length goes to infinity. This, however, was non-constructive, and so the hunt was then on to design codes with efficient encoding and decoding algorithms and which approach the channel capacity. For a random linear code of length n , it takes time and space of about $O(n^2)$ to encode (using the generator matrix) which is not so bad, but the real problem is that MLD decoding takes $O(2^n)$ operations (all codes below are binary). For LDPC codes we shall see that the belief propagation algorithm reduces this to $O(n)$. In fact these codes are typically so long in practice, e.g. $n = 50000$, that the n^2 in encoding becomes the bigger issue, for which various ad hoc solutions exist.

The interesting thing is that solutions (full and partial) to the above problem have been repeatedly rediscovered and it is only in recent years that a uniform description for these has been discovered. This is the theory of graphical models (aka Bayesian networks) and the corresponding codes are called graph-based codes. These have taken the coding theory world by storm, and now form the main topic of modern practical coding theory [18]. It should be noted, however, that unlike other topics in mathematical coding theory the decoding algorithms are of paramount importance.

The first answer to Shannon's challenge was presented in Gallager's 1960 Ph.D. thesis [9], which first defined LDPC codes and proposed iterative decoding methods, but with lack of computational power then they were impractical and their value went largely unrecognized at the time, except in the USSR with Zyablov-Pinsker [30] and Margulis [15]. In 1981, Tanner [25] rediscovered LDPC codes and generalized them by introducing the notion of a Tanner graph. None of these, even in simulation, came close to capacity, and so it was a major shock in 1993 (or a year later, when experts finally acknowledged the correctness of their results) when Berrou, Glavieux, and Thitimajshima [4] presented the new powerful class of Turbo codes with decoding algorithms that came close to Shannon's limit.

When this development finally settled in, Wiberg, Loeliger, and Koetter [28] responded by generalizing Tanner graphs to factor graphs by including state variables, allowing the theory to subsume Viterbi decoding [27],[8]. It was soon realised that the sum-product algorithm being used was a version of Pearl's belief propagation algorithm [16]. In fact the sum-product algorithm, operating by message passing in a factor graph, subsumes algorithms used in artificial intelligence, statistical mechanics, and signal processing. This theme of rediscovery again arose when in 1999 MacKay [12] rediscovered and revitalized Gallager's codes, with extensive simulations and actual proofs that LDPC codes are very good.

Attention shifted to which graphs produce the best codes, with Sipser and Spielman [24] in the theoretical computer science community rediscovering LDPC codes and realising how code performance relates to expansion properties of the underlying graph (something the Russians mentioned earlier had realised but also was temporarily unlearned). In time, LDPC codes were overtaking Turbo codes, one advantage of LDPC codes being that efficient encoders and decoders could be designed for them. Luby, Mitzenmacher, Shokrollahi, Spielman, and Stemann [11] designed one such for the binary erasure channel (BEC), performed a rigorous analysis, and noted how graphs with irregular degree distributions perform much better than the state-of-the-art regular graphs. This was further analyzed [17] using techniques of density evolution to obtain optimal degree distributions that for the BEC provably answer Shannon's challenge and indeed in practice perform much better than and have implementation advantages over Turbo codes. There is, for instance, an en-

semble of rate $1/2$ irregular LDPC codes for the AWGN channel that gets within 0.0045 dB of Shannon capacity [5].

As noted earlier, the literature (IEEE Trans.) and conference proceedings are currently awash with papers on LDPC codes and related issues with fundamental issues being whether the results for the BEC can be extended to other channels, the optimization of irregular LDPC codes for channels of interest, their implementation in VLSI, and their combination with other technologies such as MIMO (multiple-input multiple-output) systems. Time will tell as to which of the current flood of papers is of greatest importance but one fascinating development is the discovery by Koetter and Vontobel [10] of the relevance of graph covers and the introduction of the fundamental polytope. The point here is that these decoding algorithms operate locally and so cannot distinguish between a graph and some cover of it, thus yielding pseudo-codewords responsible for suboptimal behavior of the algorithms - the fundamental polytope tightly characterizes these pseudo-codewords.

2. Overview of Lectures

After this history and overview, we define what is meant by a channel and outline Shannon's results on the existence of a channel capacity limiting the possible rates of reliable information transmission, setting us our goal. There are now three aspects of graph-based codes to be discussed. We begin with trellises and give the BCJR [2] or two-way or forward-backward algorithm for trellises, explaining its efficiency compared with MLD. Then we introduce Tanner graphs and describe the belief propagation algorithm for the corresponding codes. At this point, the usefulness of sparse graph codes, specifically LDPC codes, becomes clear from a complexity standpoint. The third aspect, namely factor graphs, combines the trellis and Tanner graph approaches and there is a very general sum-product algorithm for general graph-based codes.

We next focus on the simplest channel, the BEC, where belief propagation amounts to a very straightforward algorithm, permitting detailed analysis by techniques such as density evolution. This leads to capacity-achieving ensembles of LDPC codes thereby answering Shannon's challenge, and we deduce the degree distributions of the corresponding irregular graphs. We then discuss the role of expander graphs in this area, their strengths [15],[19] and weaknesses [14]. Finally, graph covers and the fundamental polytope are discussed.

For a mathematician knowing the basics of classical coding theory and wanting to learn more, the voluminous (468 pages!) online notes by Richardson and Urbanke [18] give an exhaustive (and exhausting) description of these developments. The two survey papers by Shokrollahi [22],[23] give a very readable introduction for mathematicians. There are other good introductions to the field, many referenced below, although these are often written for engineers or theoretical computer scientists. The reader might try, for instance, Sason's course notes [20]. MacKay's website [13] has plenty of useful examples, theory, and software, whereas the website of the company Flarion Technologies [7] displays another side of LDPC codes, namely their industrial applications.

2. Channels and Capacity

Definition. A *communication channel* is a triple consisting of an input alphabet A , an output alphabet B , and for each pair $a \in A, b \in B$ a real number $P(b | a)$ between 0 and 1. $P(b | a)$ should be thought of as the probability that b is received, given that a is transmitted. In particular, we impose the constraint $\sum_{b \in B} P(b | a) = 1$, reflecting the fact that if a is transmitted, something is received.

You might complain that in practice it may be that nothing is received, i.e. we have an erasure. This however is modeled by the *binary erasure channel* (BEC) by making erasure one of the output symbols. Here, $A = \{0, 1\}$ (throughout these talks we focus on binary codes), $B = \{0, 1, e\}$, and $P(b | a) = 1 - p$ if $b = a$, $= p$ if $b = e$, else $= 0$, where p is the probability of an erasure. The BEC is the simplest channel to work with and appears in many real-life situations, where feedback channels do not exist, such as satellite links, rendering retransmission infeasible, and where one sender serves many recipients, such as a multicast scenario.

Another basic channel is the *binary symmetric channel* (BSC), where the input and output alphabets are both $\{0, 1\}$ and each bit is flipped with probability p . Thus, $P(b | a) = 1 - p$ if $b = a$, else $= p$. The theory of the BSC turns out to be much more challenging than that of the BEC.

Finally (although many other interesting channels, in particular nonsymmetric ones have been devised; see e.g. Sason's notes [20]), there is the *additive white Gaussian noise* (AWGN) channel. Here, $A \subseteq \mathbf{R}, B = \mathbf{R}$, and

$$P(b | a) = \exp(-(b - a)^2 / 2\sigma^2) / (\sqrt{2\pi}\sigma)$$

meaning that the error (noise) is normally distributed with mean 0 and standard deviation σ .

All three channels above have practical significance, but there are advantages to working with general channels, and indeed universal decoding where we seek algorithms that work well regardless of the underlying channel assumption, which may be unknown, is a dream that has undergone much investigation. Since we wish to transmit more than one symbol, we consider *memoryless channels* defined by the channel output at some instant depending only on the channel input at that instant, so that if $\mathbf{a} = a_1 \dots a_n$ is transmitted and $\mathbf{b} = b_1 \dots b_n$ received, then $P(\mathbf{b} | \mathbf{a}) = \prod_{i=1}^n P(b_i | a_i)$.

For a general discrete-input, memoryless channel, Shannon [21] in 1948 associated a number called its capacity, using information theory, a subject he simultaneously developed. For the BEC, its capacity is $1 - p$. For the BSC, its capacity is $1 + p \log_2 p + (1 - p) \log_2 (1 - p)$. For the AWGN, it is $W \log_2 (1 + S/N)$, where W is the bandwidth of the channel in Hz, S is the signal power in Watts, and N is the total noise power of the channel in Watts. The quantity S/N is called signal-to-noise ratio and is often expressed in decibels, such that 10^k is $10k$ dB. Shannon showed that if maximum likelihood decoding (MLD) (see below) is used, then the decoding error probability for any code of rate greater than the channel capacity approaches 1, and that for any rate below capacity, there are codes of that rate for which the decoding error probability decreases to zero exponentially fast in the block-length as the latter goes to infinity.

Since, however, random codes were used, Shannon's result left open the question of how to construct explicit ensembles of codes approaching capacity, with

efficient encoding and decoding algorithms. MLD for the BSC, for instance, is NP-complete [3]. For the BEC it is polynomial-time but still inefficient [6]. Shannon's arguments show that there are sequences of linear codes with rates approaching capacity and MLD error probability approaching zero. Using linear codes, encoding is polynomial-time but it is a major open question to find sequences of capacity-achieving linear codes for which MLD for the BSC is polynomial-time. We explain below how to solve this problem for the BEC using the sum-product decoding algorithm.

3. Trellises and the Forward-Backward Algorithm

Definition. A *trellis* is a directed graph in which every node is at a well-defined depth with respect to a beginning node.

These depths will also be called times, so that in Figure 1, the horizontal is a time axis. This figure gives a trellis associated to the $[7, 4, 2]$ -code with parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

meaning that the code consists of the 2^4 length 7 binary vectors x satisfying $Hx = 0$. The code is recovered from the trellis by following all possible paths through the trellis. In fact this is a minimal trellis, and given a code it is easy to construct a corresponding minimal trellis. The most important invariants of the trellis are its maximum width, here 4, called its *state complexity* σ , and its *constraint length* $\log_2(\sigma)$.

Figure 2 gives another example of a trellis, this time attached to the $[4, 3, 2]$ -code C consisting of all vectors of length 4 with even weight. It has state complexity 2 and so constraint length 1. We consider in detail the problem of decoding using this code and see how useful the trellis approach is for this, with the state complexity determining its efficiency.

Suppose a codeword is transmitted over the BSC with bit-error probability $p = 0.1$ and $y = 0010$ received. Let us compute the most likely 2nd component \hat{x}_2 of the transmitted codeword x . The probability that $x_2 = 0$ given y is

$$P(x_2 = 0 | y) = \sum_{x \in C, x_2=0} \frac{P(y | x)P(x)}{P(y)} \quad (1)$$

since by Bayes rule, each term on the righthandside gives the posterior probability that x is the input, given that y is the output.

We compute that

$$\sum_{x \in C, x_2=0} P(y | x)P(x) = (3p(1 - p)^3 + p^3(1 - p))/8 = 0.02745$$

since $P(x) = 1/8$ (there being 8 codewords) and since of the 4 codewords in C with $x_2 = 0$, namely 0000, 1001, 1010, 0011, one (1001) agrees with y in only one place whereas the other three agree in exactly three places.

Likewise, we compute that

$$\sum_{x \in C, x_2=1} P(y | x)P(x) = (3p^3(1-p) + p(1-p)^3)/8 = 0.00945$$

since of the 4 codewords in C with $x_2 = 1$, namely 1100, 1111, 0101, 0110, only one (0110) agrees in three places, the rest in only one.

Overall, $P(y)$ is the sum over all x in C of the $P(y | x)$, so $P(y) = 0.02745 + 0.00945 = 0.0369$. We now return to equation (1) and compute that $P(x_2 = 0 | y) = 0.02745/0.0369 = 0.744$ and likewise $P(x_2 = 1 | y) = 0.00945/0.0369 = 0.256$. Then Maximum-Posterior (MAP) decoding tells us that most likely $\hat{x}_2 = 0$ since $0.744 > 0.256$. The problem is that this is simply too much work, particularly for large codes where about $n2^k$ operations will be needed.

The BCJR (or two-way or forward-backward) algorithm on the associated trellis works much better. Given received word $y = 0010$, we make a forward computation as given in Figure 3 and then a backward computation given in Figure 4. To compute e.g. $P(x_2 = 0 | y)$, we put these computations together as in Figure 5, focusing just on the 2nd portion of the trellis. The zeros are on the horizontal edges and multiplying the node and edge values and adding gives:

$$P(x_2 = 0 | y) = q(0.45 \times 0.45 \times 0.09 + 0.05 \times 0.45 \times 0.41) = 0.744$$

Likewise using the diagonals,

$$P(x_2 = 1 | y) = q(0.45 \times 0.05 \times 0.41 + 0.05 \times 0.05 \times 0.09) = 0.256$$

Here q is a normalizing constant to make sure that the two above probabilities add up to 1 (in fact $q = 27.1$).

The fact that the answers agree with those obtained by direct calculation is easily checked. Consider $P(x_2 = 0 | y)$. The edge value $a = P(x_2 = 0)P(y_2 = 0 | x_2 = 0)$, the forward values $f_0 = P(x_1 = 0)P(y_1 = 0 | x_1 = 0)$, $f_1 = P(x_1 = 1)P(y_1 = 0 | x_1 = 1)$, and the backward values $b_0 = P(y_4 = 0 | x_4 = 0)P(x_3 = 0)P(y_3 = 1 | x_3 = 0) + P(y_4 = 0 | x_4 = 1)P(x_3 = 1)P(y_3 = 1 | x_3 = 1)$, $b_1 = P(y_4 = 0 | x_4 = 0)P(x_3 = 1)P(y_3 = 1 | x_3 = 1) + P(y_4 = 0 | x_4 = 1)P(x_3 = 0)P(y_3 = 1 | x_3 = 0)$. Then $f_0ab_0 + f_1ab_1$ runs through

$$\sum_{c \in C, c_2=0} \prod_{i=1}^3 P(x_i = c_i) \prod P(y_i = r_i | x_i = c_i)$$

where $r_1 = 0, r_2 = 0, r_3 = 1, r_4 = 0$ denotes the received word. Since the channel is memoryless, this is $\sum_{x \in C, x_2=0} P(x)P(y | x)$. If the trellis is simple, i.e. has small state complexity/constraint length, then the forward-backward algorithm will be considerably quicker than the term-by-term approach given first.

Analyzing this further, one sees that the time savings come essentially from using the distributive law, namely $ab + ac = a(b + c)$ where the first expression requires two multiplications and an addition whereas the second only one multiplication and one addition. Note that sum distributes over min (leading to the ‘‘min-sum’’ algorithm, analogous to the sum-product algorithm), namely $a + \min(b, c) = \min(a, b) + \min(a, c)$. What we have ended up with is a comparison

of the likelihood that $x_2 = 0$ versus that $x_2 = 1$, given the received word y . Much of what follows is a massive generalization of this observation.

It is worth noting that there is a variant on trellises, namely *tail-biting trellises*, where the time axis is a circle rather than a line and codewords are given by the paths that go round once meeting themselves. Improper paths that go around more than once before closing are like the pseudo-codewords of section 10. The point is that whereas regular trellises are constrained to be thin at each end and so wide in the middle, tail-biting trellises tend to have more uniform width. For instance, whereas the $[24, 12, 8]$ -binary Golay code requires a trellis of constraint length at least 8, it has a tail-biting trellis of constraint length 4.

4. Tanner Graphs

Suppose that a binary linear code C has parity-check matrix H , i.e. $C = \{x \mid Hx = 0\}$. Suppose that C has length n and that H is an $r \times n$ matrix over \mathbf{F}_2 . The rate of C is then at least $(n - r)/n$ (some of the checks may be redundant).

Definition. The *Tanner graph* associated to H is a bipartite graph constructed by taking as its vertices two kinds of nodes, namely n message nodes (one for each component x_i) and r check nodes (one for each parity check), and by an edge going from a message node v to a check node c only when c involves v .

Note that the same code can have different realizations as a Tanner graph (or indeed as a trellis) by choosing different parity-check matrices for it.

For example, for the parity check matrix of the $[7, 4, 2]$ -code given in the last section, C is the common solution of $x_1 + x_2 + x_3 = 0$, $x_1 + x_4 + x_5 = 0$, $x_1 + x_6 + x_7 = 0$ - call these c_1, c_2, c_3 . The associated Tanner graph is given in Figure 6. It is cycle-free, which will turn out to be a desirable property for belief propagation to allow independence assumptions, but this is actually a rare occurrence. Most good codes, for instance the $[7, 4, 3]$ -Hamming code, have no cycle-free realizations, although we can reasonably approximate cycle-free Tanner graphs in practice by ensuring the graph has some desirable properties, e.g. large girth, expansion properties, etc. This will be explored later.

One important property is that the graph be sparse since as with the trellises computations will take place along those edges. This is typically an informal notion, but if we wish to be precise, a sequence of $r \times n$ matrices is called sparse if rn tends to infinity and the number of nonzero elements in these matrices is always less than $k \max(r, n)$ for some k .

Definition. The codes constructed by taking sparse matrices as their parity-check matrices are called *Low Density Parity-Check* (LDPC) Codes, the central topic of study in these lectures. An (s, t) *regular LDPC code* is a linear code with a parity-check matrix H that has s ones in every column and t ones in every row.

If H is an $r \times n$ parity-check matrix of an (s, t) regular LDPC code, the total number of ones $rt = ns$. The rate of the code is at least $(n - r)/n = 1 - s/t$, so for instance a $(3, 6)$ regular LDPC code has rate $\geq 1/2$ (typically not much

larger). Irregular LDPC codes have parity-check matrices with variable numbers of ones in their rows/columns and correspondingly the vertices of their Tanner graphs have irregular degree distributions. They have overtaken regular LDPC codes in importance since they can be designed to come closer to capacity.

Gallager’s construction of regular LDPC codes proceeds as follows. Suppose we want a $(3, 6)$ regular LDPC code. Suppose n is a multiple of 6. Make the first row 11111100...0 of length n , the second row 0000011111100...0, and so on, for the first $n/6$ rows. To get the next $n/6$ rows apply a random permutation of degree n to the columns of the above $n/6 \times n$ matrix, and a second random permutation yields the final $n/6$ rows. This is the so-called random construction of an LDPC code. In section 9 below we discuss the pros and cons of efforts at non-random construction.

5. Belief Propagation

Message-passing algorithms form a class of algorithms on Tanner graphs where in one round “messages” are passed from message nodes to check nodes and from check nodes to message nodes. In belief propagation, the “messages” are actually probabilities (or “beliefs”) meant to represent whether the i th message node “believes” it is a 0 or a 1. At each stage, each node decides what it is to send to an adjacent node by using all the information that the other adjacent nodes have sent it.

We begin with an example (from Wolf’s excellent presentation [29]). Consider the 9×12 parity-check matrix

$$H = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

There are three ones per column and four ones per row and so this produces a $(3, 4)$ regular LDPC code. It’s a $[12, 3, 4]$ -code, whose codewords are

$$\begin{aligned} &000000000000, 000011100001, 010101100101, 010110000100, \\ &101001111011, 101010011010, 111100011110, 111111111111 \end{aligned}$$

Suppose a codeword x is transmitted and the channel initially yields the following probabilities that $x_i = 1$, $1 \leq i \leq 12$; 0.9, 0.5, 0.4, 0.3, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9. The i th message node of the Tanner graph gets this i th initial estimate and then broadcasts it to its adjacent check nodes. Each check node then makes new estimates based on what it has received from other message nodes and sends these estimates back to the message nodes.

How does it make this estimate? Take the first check c_1 given by $x_3 + x_6 + x_7 + x_8 = 0$. This check node receives the initial estimates p_3, p_6, p_7, p_8 from the 3rd,

6th, 7th, and 8th message nodes respectively. It computes new estimates

$$p'_3 = p_6(1 - p_7)(1 - p_8) + p_7(1 - p_6)(1 - p_8) + p_8(1 - p_6)(1 - p_7) + p_6p_7p_8$$

and likewise for p'_6, p'_7, p'_8 . This formula is explained below (see (3)).

Then p'_3 is sent back to the 3rd message node, p'_6 to the 6th message node, and so on. This happens simultaneously from all check nodes to their adjacent message nodes (this parallelism is a definite implementation advantage over Turbo codes). Thus the 3rd message node receives revised estimates from check nodes c_1, c_5, c_9 , and makes a further refined estimate itself.

How does it make the refinement? Suppose it receives p_A, p_B, p_C from those three check nodes. The estimate it sends back to A is simply $kp_3p_Bp_C$, to B is $kp_3p_Ap_C$, and to C is $kp_3p_Ap_B$, where p_3 is the initial channel estimate and k is a normalizing constant. To compute k we solve simultaneously $p' = kp_Ap_Bp_C, 1-p' = k(1-p_A)(1-p_B)(1-p_C)$ to get $k = 1/(p_Ap_Bp_C + (1-p_A)(1-p_B)(1-p_C))$. This formula is explained below (see (2)).

The process is then iterated. Let us run the example. Using $p_3 = 0.4, p_6 = 0.9, p_7 = 0.9, p_8 = 0.9$, we compute at c_1 that $p'_3 = 0.756$. Since c_5 involves the 1st, 3rd, 8th, and 11th message nodes and the initial estimates for the three other than the 3rd are again 0.9, check node c_5 also sends $p'_3 = 0.756$ back to the 3rd message node. Finally, c_9 involves the 2nd, 3rd, 9th, and 10th message nodes, and it computes $p'_3 = 0.5$.

Thus the 3rd message node receives from c_1, c_5, c_9 the estimates 0.756, 0.756, 0.5 respectively. What does it send back? The products of the channel estimate 0.4 and the other two estimates give respectively 0.1512, 0.1512, 0.2286. These are normalized using $k = 3.1692$ to 0.4792, 0.4792, 0.7245.

After about 22 or more rounds of this, every message node carries an estimate negligibly lower than 1.000 and we conclude that the all ones codeword was transmitted. (In fact a hard decision after 4 rounds, comparing the estimates with 0.5, gives the same conclusion.) We now give the general method.

Definition. Given a binary random variable x , $P(x = 0)/P(x = 1)$ will be called the *likelihood* of x , denoted $L(x)$. Likewise, the *conditional likelihood* of x given y will refer to $P(x = 0 | y)/P(x = 1 | y)$, denoted $L(x | y)$.

Likelihoods allow us to make hard decisions on x by seeing if $L(x)$ is greater than or less than 1. Since additions are easier to handle than multiplications, the best algorithms often employ *log-likelihoods*, i.e. $\log L(x), \log L(x | y)$, i.e. these are the “beliefs” passed along edges.

If y_1, \dots, y_d are independent random variables, then

$$\log L(x | y_1, \dots, y_d) = \sum_{i=1}^d \log L(x | y_i) \quad (2)$$

There is a small trick now to computing $\log L(x_1 + \dots + x_d | y_1, \dots, y_d)$, where the x_i and y_i are binary random variables. Namely we use the easily checked fact that

$$2P(x_1 + x_2 = 0 | y_1, y_2) - 1 = (2P(x_1 = 0 | y_1) - 1)(2P(x_2 = 0 | y_2) - 1)$$

yielding by induction that

$$2P(x_1 + \dots + x_d = 0 \mid y_1, \dots, y_d) - 1 = \prod_{i=1}^d (2P(x_i = 0 \mid y_i) - 1)$$

Letting $L_i = L(x_i \mid y_i) = P(x_i = 0 \mid y_i)/(1 - P(x_i = 0 \mid y_i))$, we get that $P(x_i = 0 \mid y_i) = L_i/(1 + L_i)$, whence $2P(x_i = 0 \mid y_i) - 1 = (L_i - 1)/(L_i + 1)$. Combining with the previous paragraph gives

$$2P(x_1 + \dots + x_d = 0 \mid y_1, \dots, y_d) - 1 = \prod_{i=1}^d (L_i - 1)/(L_i + 1)$$

which, if we set $\ell_i = \log L_i$, the conditional log-likelihood, yields

$$2P(x_1 + \dots + x_d = 0 \mid y_1, \dots, y_d) - 1 = \prod_{i=1}^d \tanh(\ell_i/2)$$

whence

$$\log L(x_1 + \dots + x_d = 0 \mid y_1, \dots, y_d) = \log\left(\frac{\prod_{i=1}^d \tanh(\ell_i/2)}{1 - \prod_{i=1}^d \tanh(\ell_i/2)}\right) \quad (3)$$

So our example should be viewed as successive refinement of log-likelihoods from $(\ell_1, \dots, \ell_{12}) \mapsto (\ell'_1, \dots, \ell'_{12})$. Let $\phi(x) = (x - 1)/(x + 1)$. By the above, $\phi(L(x_6 + x_7 + x_8 \mid y)) = \phi(L(x_6 \mid y))\phi(L(x_7 \mid y))\phi(L(x_8 \mid y))$ and since $x_6 + x_7 + x_8$ should equal x_3 , this gives us our new estimate ℓ'_3 for the log-likelihood of x_3 given y . This is where the formula $2p'_3 - 1 = (2p_6 - 1)(2p_7 - 1)(2p_8 - 1)$ comes from. In terms of how the log-likelihoods evolve, we have

$$\phi(\exp(\ell'_3)) = \phi(\exp(\ell_6))\phi(\exp(\ell_7))\phi(\exp(\ell_8))$$

If $x > 0$, set $\gamma_2(x) = \log \tanh(x/2)$. Then we have

$$\ell'_3 = \gamma_2^{-1}(\gamma_2(\ell_6) + \gamma_2(\ell_7) + \gamma_2(\ell_8))$$

Looking at the graph of γ_2 , $\gamma_2(x)$ is very large for x close to 0 and then drops dramatically to 0 as x increases, and so the righthandside is approximated by $\min(\ell_6, \ell_7, \ell_8)$, which can be used to create an approximate min-sum algorithm.

The main belief propagation algorithm now follows the example given at the start, only with log-likelihoods rather than probabilities being the messages sent. In round zero, message nodes receive their log-likelihoods conditioned on observed values. For example, for a BSC with probability p of bits flipping, if a 0 is observed at some bit, then the initial log-likelihood at that message node is $\log((1 - p)/p)$, whereas if a 1 is observed, the initial value is $\log(p/(1 - p))$. In subsequent rounds, a check node c sends to an adjacent message node v a likelihood computed by (3) above. A message node v sends to an adjacent check node c its log-likelihood conditioned on its observed value and on the incoming log-likelihoods from adjacent check nodes other than c , by using (2) above. This is iterated for some maximal

number of rounds or until the passed likelihoods are close to certainty, whichever is first.

Since \tanh can take on negative values, for which \log is not defined, we let $\gamma : [-\infty, \infty] \rightarrow \{\pm 1\} \times [0, \infty]$ be given by $\gamma(x) = (\text{sgn}(x), -\log \tanh(|x|/2))$. Since γ is bijective, there exists an inverse function γ^{-1} . Then (3) is equivalent to

$$m_{cv}^{(\ell)} = \gamma^{-1}\left(\sum_{i=1}^d \gamma(m_{v_i c}^{(\ell-1)})\right) \quad (4)$$

where $m_{cv}^{(\ell)}$ is the message passed from check node c to message node v in the ℓ th round, v_1, \dots, v_d are the message nodes other than v adjacent to c , and $m_{v_i c}^{(\ell-1)}$ is the message passed from message node v_i to check node c in the $(\ell - 1)$ th round.

Certainty is when a log-likelihood of $+\infty$ (certainty of a 0) or of $-\infty$ (certainty of a 1) is received at a message node. The algorithm can run for a given number of rounds or until certainty or near-certainty (given by some threshold) is obtained.

The decoding complexity of a round of the algorithm is proportional to the number of edges in the Tanner graph. This is why sparsity of the parity-check matrix is of paramount importance, and explains the value of LDPC codes. For example, for an (s, t) regular LDPC code, the Tanner graph has sn edges.

6. Factor Graphs and Generalized Distributive Laws

Wiberg and others [28] extended Tanner graphs to include invisible nodes, or “states”, thus creating a marriage of trellises and Tanner graphs. We call these factor graphs. Figure 7 shows the factor graph corresponding to the trellis given in Figure 1, attached to a $[7, 4, 2]$ -code. The states are represented by open circles of different sizes, the check nodes represent the constraints governing a particular trellis section, only certain pairs of neighboring states are allowed, and each allowed pair uniquely determines an associated code symbol. Note that the factor graph associated to a trellis is always cycle-free.

The idea is that in computer science and engineering many algorithms deal with complicated “global” functions of many variables that can be efficiently computed by exploiting the way that the global function factors into a product of simple “local” functions. For example, suppose

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3)f_D(x_3, x_4)f_E(x_3, x_5)$$

Figure 8 shows how the dependencies can be represented by a bipartite graph with function nodes and variable nodes. We shall describe factor graphs that not only encode the factorization of the global function but, when cycle-free, yield an algorithm, the sum-product algorithm, for computing marginal functions. In the coding theory world, this was realised by Aji and McEliece [1] presented at Ulm in 1997, where corridor discussions among the so-called “Ulm group” crystallized the notion of factor graph, but the approach has broad application to Markov random fields, Bayesian networks (expertise in which was recently claimed to be an essential part of Microsoft’s competitive advantage by Bill Gates and which MIT Technology Review recently named one of the top ten emerging technologies), etc.

These techniques are used e.g. in drug discovery, foreign-language translation, and microchip manufacturing.

Given global function $g(x_1, \dots, x_n)$, the marginal functions are defined by

$$g_i(x_i) := \sum g(x_1, \dots, x_n), \quad 1 \leq i \leq n$$

meaning the sum over all variables other than x_i . So, for instance, using the distributive law,

$$g_1(x_1) = f_A(x_1) \sum_{x_2, x_3} (f_B(x_2) f_C(x_1, x_2, x_3)) \left(\sum_{x_4} f_D(x_3, x_4) \right) \left(\sum_{x_5} f_E(x_3, x_5) \right)$$

Figure 9 shows how to represent this factorization and then add “not-sums” on the edges. When the factor graph is a tree, as here, by making x_i the root, the factor graph gives a message-passing algorithm for computing $g_i(x_i)$, namely:

1. At a variable node x_i , take the product of expressions formed at the descendants of x_i .

2. At a function node f , take the product of f with expressions formed at the descendants of f ; then perform the not-sum over the parent of f .

In coding theory, if we transmit $\mathbf{x} = x_1 \dots x_n$ over a memoryless channel and observe $\mathbf{y} = y_1 \dots y_n$, then the a posteriori joint probability distribution of $\{x_1, \dots, x_n\}$ is proportional to

$$P(x_1, \dots, x_n) = \prod_{A \in G} P_A(x_A) \prod_{i=1}^n P(y_i | x_i)$$

so computing it amounts to a sum-product algorithm on a corresponding factor graph. As a simple example, consider the parity-check matrix H of the $[7, 4, 2]$ -code C considered above. Let f be the characteristic function of C , so that $f(x) = 1$ if $Hx = 0$ and is 0 otherwise. Letting $\chi(x, y, z) = 1$ if $x + y + z = 0$ and be 0 otherwise, the global function f factors into local functions as

$$f(x_1, \dots, x_7) = \chi(x_1, x_2, x_3) \chi(x_1, x_4, x_5) \chi(x_1, x_6, x_7)$$

7. Density Evolution on the Binary Erasure Channel

The following example is taken from [18]. Let

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

This is the parity-check matrix of a $(3, 6)$ regular LDPC code C of length 20 and rate $1/2$. Suppose that over a binary erasure channel the received word is

$$y = (e, e, 0, 0, 0, e, 0, 1, 1, 0, e, 0, 0, 0, 0, e, e, 0, e, 0)$$

For any subset S of $\{1, \dots, 20\}$ let H_S denote the $10 \times |S|$ matrix comprised of the k th column for each $k \in S$ and x_S denote the vector of length $|S|$ consisting of all x_k for $k \in S$. Let E denote the set of erased bits, so here $\{1, 2, 6, 11, 16, 17, 19\}$ and \bar{E} its complement, i.e. the received bits. Since $Hx = 0$ for $x \in C$, $H_E x_E + H_{\bar{E}} x_{\bar{E}} = 0$ and so $H_E x_E = H_{\bar{E}} x_{\bar{E}}$, with this latter syndrome s calculable from the given received bits. In the example, we obtain:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_6 \\ x_{11} \\ x_{16} \\ x_{17} \\ x_{19} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

The 6th, 7th, and 8th rows have just one 1 in them, which allows us to find that $x_{11} = 0, x_6 = 1, x_1 = 1$ respectively. These then become received bits and we get a simpler equation (eliminating those rows too):

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_2 \\ x_{16} \\ x_{17} \\ x_{19} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Now the first two and last rows have a single 1 in them, leading immediately to $x_{16} = 1, x_{19} = 0$. Continuing we obtain $x_2 = 1, x_{17} = 0$.

In this way, the whole codeword is recovered without any need for Gaussian elimination. This works so long as we always are left with a row involving exactly one 1. For general linear codes this fails with high probability but for sparse H the opposite is true (see stopping sets below). Let us describe the above procedure in terms of the Tanner graph.

In effect, we begin by removing all message nodes where 0 or 1 is received, leaving just the erased bits. Their values are transmitted to adjacent check nodes (to compute the syndrome s) and then those edges removed. We find all check nodes of degree 1 in the remaining subgraph and then pass the value of each to its unique adjacent message node, which thereby becomes non-erased. We now iterate with these newly non-erased message nodes getting removed, and so on. We now show that this algorithm is the same as belief propagation on the BEC.

Let us assume without loss of generality that the all zero codeword is transmitted. Then a 1 will never be received at any bit and so the initial log-likelihoods are

$m_v = +\infty$ if 0 is received at v , else is 0 (since an erased bit is equally likely to come from a 0 or a 1 according to the BEC model). The update equations tell us the following (noting that $\gamma_2(+\infty) = 0$ and $\gamma_2(0) = +\infty$. If v is not erased, then m_{vc} is always $+\infty$. If v is erased, then $m_{vc} = +\infty$ if and only if there is a check node adjacent to v other than c which sent $+\infty$ to v in the previous round. $m_{cv} = +\infty$ if and only if all the message nodes adjacent to c excluding v are not erased, else it equals 0.

Thus, the messages sent are binary ($+\infty$ or 0) and by agreeing to delete messages and edges once they carry a $+\infty$, i.e. certainty, then we recover the algorithm given above. We can also very effectively analyze the algorithm by considering the proportion of edges carrying a 0 and seeing if this proportion approaches zero over time (in which case our certainty approaches 100 percent. This is density evolution [17],[18].

A *stopping set* in the graph is a set of message nodes such that the graph induced by these message nodes has the property that no check node has degree one. The above algorithm for the BEC stops prematurely, i.e. without recovering all the message nodes, if and only if this subgraph has a stopping set. Unions of stopping sets are stopping sets, and so every finite graph contains a unique maximal stopping set (possibly empty). The size of a stopping set actually provides a lower bound for the minimum distance of the code.

As for density evolution, let λ_d be the probability that an edge is connected to a message node of degree d and ρ_d be the probability that an edge is connected to a check node of degree d . The generating functions $\lambda(x) = \sum_d \lambda_d x^{d-1}$, $\rho(x) = \sum_d \rho_d x^{d-1}$ are critical in evaluating an LDPC code. For an (s, t) regular LDPC code, $\lambda(x) = x^{s-1}$, $\rho(x) = x^{t-1}$, but it will turn out that more complicated $\lambda(x)$, $\rho(x)$ are desirable, explaining the value of irregular LDPC codes.

Let p_i be the probability that the message passed from a message node to a check node at round i of the algorithm is 0. Let q_i be the probability that the message passed from a check node to a message node at round i is 0. A message from a message node v to a check node c is 0 if and only if v was erased and all the messages coming from the neighboring check nodes other than c are 0, which is q_i^{d-1} (assuming independence and that the degree of v is d) and so $p_{i+1} = pq_i^{d-1}$. The check node c sends message ∞ to the message node v if and only if all the neighboring message nodes except for v send a message to ∞ to c the previous round, and so $q_i = 1 - (1 - p_i)^{d-1}$, where here d is the degree of c . Putting these together,

$$p_{i+1} = p\lambda(1 - \rho(1 - p_i)) \quad (5)$$

This is the critical equation. Suppose e.g. we have a $(3, 6)$ regular LDPC code, so that $\lambda(x) = x^2$, $\rho(x) = x^5$. Consider $p_0 = p = 0.4$; we successively compute p_1, p_2, \dots to be 0.3402, 0.3062, 0.2818, 0.2617, 0.2438, 0.2266, ... If, however, we start with $p_0 = p = 0.45$, then we successively get 0.4058, 0.3858, 0.3748, 0.3681, 0.3639, 0.3612, ... In the first case, the numbers tend to 0, whereas the latter tend to 0.3554. The first will lead to successful decoding, the second unsuccessful. Further experimentation shows that the cut-off in behavior is at $p = 0.42944$. This is illustrated by the graphs in Figure 10 for $p = 0.4, 0.42944, 0.45$.

This value of p is called the *threshold* and can be determined graphically or by the following theorem. It has also been observed experimentally in simulations to

be the correct cut-off point.

Theorem. [18] Suppose we have an (s, t) regular LDPC code. Let σ be the unique positive real root of the polynomial $((s-1)(t-1)-1)x^{t-2} - \sum_{i=0}^{t-3} x^i$. Then the threshold equals $(1-\sigma)/(1-\sigma^{t-1})^{s-1}$.

Proof The process will converge to a fixed point, i.e. where $p\lambda(1-\rho(1-x)) = x$. Consider this equation as giving p as a function of x , namely $g(x) = x/(\lambda(1-\rho(1-x))) = x/(1-(1-x)^{t-1})^{s-1}$. Note that $g(x) \geq x$ with equality only at $x = 1$. The threshold then is given by the minimum of $g(x)$ over the range $x \in [0, 1]$. Working with $g(1-x)$ and taking its derivative gives polynomial equation

$$((s-1)(t-1)-1)x^{t-1} - (s-1)(t-1)x^{t-2} + 1 = 0$$

By Descartes' rule of signs, this equation has at most two positive real roots, one of which is $x = 1$. Dividing by $x-1$ yields the polynomial equation $((s-1)(t-1)-1)x^{t-2} - \sum_{i=0}^{t-3} x^i = 0$, which by the above has at most one positive real root. Since $g(x)$ has a pole at $x = 0$ and $g(1) = 1$, the minimum of $g(x)$ on $[0, 1]$ occurs in the interior, so the positive real root does exist. Plugging it back into g we obtain the threshold value of p .

Thus, for example, with $s = 3, t = 6$ we get polynomial $9x^4 - x^3 - x^2 - x - 1$, which has unique positive real root $\sigma = 0.73943$ leading to $(1-\sigma)/(1-\sigma^5)^2 = 0.42944$.

In general (5) yields the following condition for successful decoding:

$$p\lambda(1-\rho(1-x)) < x \text{ for } x \in (0, p] \quad (6)$$

This is sufficient since if it holds, then by (5), $p = p_0 > p_1 > p_2 > \dots$ These must approach a limit ≥ 0 , which is then a solution of $p\lambda(1-\rho(1-x)) = x$. By (6), that can only be 0.

Let \mathcal{E} denote the total number of edges of the Tanner graph. Then there are $\lambda_i \mathcal{E}/i$ message nodes of degree i , whence the total number of message nodes $n = \sum_i \lambda_i \mathcal{E}/i = \mathcal{E} \int_0^1 \lambda(x) dx$. Likewise, the number of check nodes $r = \mathcal{E} \int_0^1 \rho(x) dx$. Thus, r/n equals $\int_0^1 \rho(x) dx / \int_0^1 \lambda(x) dx$ and so the lower bound for rate $1 - r/n$ can be read off $\lambda(x)$ and $\rho(x)$.

8. Capacity-Achieving Ensembles of Codes

It can be shown that LDPC codes get arbitrarily close to capacity on the BEC but it is not known if the same holds for arbitrary channels, although the evidence [5] certainly suggests that they do for the BSC and AWGN channels. We call an LDPC code *ϵ -close to capacity* for channel \mathcal{C} of capacity $Cap(\mathcal{C})$ with respect to some message-passing algorithm if the rate of the code is $\geq Cap(\mathcal{C}) - \epsilon$ and if the algorithm corrects errors over that channel with arbitrarily high probability. A sequence of degree distributions $(\lambda^{(n)}(x), \rho^{(n)}(x))$ is called *capacity-achieving* over \mathcal{C} if for any ϵ there exists n_0 such that for all $n \geq n_0$ an LDPC code with degree distribution $(\lambda^{(n)}(x), \rho^{(n)}(x))$ is ϵ -close to capacity.

Major Open Question. Is there a channel other than the BEC and a message-passing algorithm for which there exists a capacity-achieving sequence?

Here is how the question is answered for the BEC with erasure probability p [17]. Let $\epsilon > 0$ be given and D be the ceiling of $1/\epsilon$. Let $H(D)$ denote the harmonic sum $\sum_{i=1}^D 1/i$. Set $\lambda(x) = (1/H(D)) \sum_{i=1}^D x^i/i$ and $\rho(x) = \exp(\mu(x-1))$, where $\mu = H(D)/p$ (actually not a polynomial but it can be arbitrarily well approximated by polynomials). Note that these are legitimate degree distributions since $\lambda(1) = 1, \rho(1) = 1$ and their coefficients lie between 0 and 1.

Since $\lambda(x) < (-1/H(D)) \log(1-x)$ (by completing the power series),

$$p\lambda(1 - \rho(1 - x)) < (-p/H(D)) \log \rho(1 - x) = (\mu p/H(D))x = x$$

establishing (6). The rate of these codes is at least $1 - r/n = 1 - \int_0^1 \rho(x) dx / \int_0^1 \lambda(x) dx = 1 - p(1 + 1/D)(1 - e^{-\mu})$, which exceeds $1 - p(1 + \epsilon)$. Recalling that the BEC has capacity $1 - p$, this shows that the LDPC codes with the above degree distributions (called Tornado codes for commercial appeal) are ϵ -close to capacity (and by (6) the belief propagation algorithm corrects errors arbitrarily well). We conclude that:

Theorem. [17] For the binary erasure channel, there exists a family of LDPC codes whose rates approach capacity and which can be decoded by belief propagation with arbitrarily small probability of error, i.e. Shannon's challenge is answered for the BEC.

For other channels belief propagation can be quite complicated and so sometimes a discretized version is employed with the messages made binary, as they are with the BEC.

9. Graphs of Large Girth, Expander Graphs and LDPC Codes

Attention shifts to what sorts of Tanner graphs are best for decoding. After ℓ rounds message-passing travels at most 2ℓ edges from any node and so stays within the ball of radius 2ℓ about that node. If for all the vertices these balls are trees, then the graph has *girth* (i.e. smallest cycle length) $> 2\ell$ but in practice it is enough that this holds for most nodes. Theoretical computer science has arguments to show that the actual behavior of the algorithm is sharply concentrated around its expectation.

For fixed ℓ and large enough n, r , for random bipartite graphs the neighborhood of depth ℓ of most of the message nodes is a tree. If (6) holds, then for any $\epsilon > 0$, there exists n_0 such that for all $n \geq n_0$, the BEC algorithm given reduces the number of erased message nodes below size ϵn . To get further, we introduce the following.

Definition. A bipartite graph with n message nodes is an (α, β) -*expander* if for any subset S of the message nodes of size $\leq \alpha n$ the number of neighbors of S is $\geq \beta a_S |S|$, where a_S is the average degree of the nodes in S .

One can show that if the Tanner graph is an $(\epsilon, 1/2)$ -expander, then the BEC algorithm recovers any set of ϵn or fewer erasures, and so we are done.

As noted earlier, if there is no cycle of length $\leq 2\ell$, then the independence assumption is valid for ℓ rounds of belief propagation. In particular, for these rounds density evolution describes exactly the expected behavior of the density functions. The girth of a graph is clearly even for bipartite graphs and can be upper bounded in the following way:

Theorem. Consider an (s, t) regular LDPC code with block length $n = rt/s$. Let $\alpha = (s - 1)(t - 1)$. If the girth $c \equiv 2 \pmod{4}$, then $c \leq 4 \log_\alpha r + 2$.

Proof Starting at a check node, count the number of check nodes reached after at most $(c - 2)/4$ steps. These are all distinct, yielding the inequality

$$1 + t(s - 1) + t(s - 1)\alpha + \dots + t(s - 1)\alpha^{(c-2)/4-1} \leq r$$

Replacing t by $t - 1$ in the lefthandside gives

$$1 + \alpha + \alpha^2 + \dots + \alpha^{(c-2)/4} = (\alpha^{(c-2)/4+1} - 1)/(\alpha - 1) \leq r$$

from which the result follows.

Note also that a word of weight d leads to a cycle of length at most $2d$ in the Tanner graph, but not vice versa (leading to pseudo-codewords as studied in the next section). Thus the girth of a graph is akin to the minimum distance of a code, and just as codes can withstand having a few low-weight codewords, a few short cycles don't necessarily hurt the decoding. Ramanujan graphs, which have large girth in an asymptotic sense, are now considered, their importance to LDPC codes having been recognized as long ago as Margulis' paper [15].

Definition. A *Ramanujan graph* is a finite, connected k -regular graph such that $\mu_1 \leq 2\sqrt{k - 1}$ where μ_1 is the largest nontrivial eigenvalue of the adjacency matrix of the graph.

There are many ways to construct Ramanujan graphs, the following method being due to Margulis. Let q be an odd prime and $PGL_2(\mathbf{F}_q)$ the quotient group of 2×2 invertible matrices over \mathbf{F}_q modulo scalars. This group has order $q^3 - q$. The projective special linear group $PSL_2(\mathbf{F}_q)$ coming from determinant one matrices has index 2 in $PGL_2(\mathbf{F}_q)$. Suppose now $p < q$ are primes that are $1 \pmod{4}$ with p a quadratic nonresidue mod q . By Jacobi, $p = a_0^2 + a_1^2 + a_2^2 + a_3^2$ has exactly $p + 1$ solutions with a_0 odd and greater than zero and a_j even for $j = 1, 2, 3$. Let $i \in \mathbf{F}_q$ satisfy $i^2 = -1$. For each of the $p + 1$ solutions, define a matrix

$$\begin{pmatrix} a_0 + ia_1 & a_2 + ia_3 \\ -a_2 + ia_3 & a_0 - ia_1 \end{pmatrix}$$

Let X be the set of all such matrices.

The Cayley graph of $PGL_2(\mathbf{F}_q)$ with respect to X is then a $p+1$ -regular bipartite Ramanujan graph with $q^3 - q$ vertices and girth $c \geq 4 \log_p q - \log_p 4$. We build a corresponding LDPC code as follows.

Let the columns of the parity-check matrix be indexed by two copies of $V := PSL_2(\mathbf{F}_q)$. Since p is a nonresidue mod q and $\det(A) = p$ for all $A \in X$, the coset $VA \neq V$ and so $PGL_2(\mathbf{F}_q) = V \cup VA$. Index the rows of the parity-check matrix by the elements of VA . For every column, put a one in the position of the rows $vA_1, \dots, vA_{(p+1)/2}$ for v in the left half of the matrix and put a one in the position of the rows of $vA_1^{-1}, \dots, vA_{(p+1)/2}^{-1}$ for v in the right half of the matrix, where X consists of $A_1, \dots, A_{(p+1)/2}$ and their inverses. This forms a $((p+1)/2, p+1)$ regular LDPC code with blocklength $q^3 - q$ and rate $1/2$.

For example, Rosenthal and Vontobel consider in detail [19] the case $p = 5, q = 17$, yielding a $(3, 6)$ regular LDPC code of blocklength 4896 and in fact rank 2474. The girth was computed to be 12. [19] presented promising performance results. The parity-check matrix in fact has several redundant rows and so rate higher than $1/2$ but decodes as well as a random code of weight $1/2$. The catch, though, is that the code has words of weight 24 and below a block error probability of 10^{-5} this ruins its performance [14].

[14] also considered other LDPC codes constructed by Margulis' techniques in [19] and found similar problems. There is an ongoing battle between advocates of random and non-random (algebraic) constructions of LDPC codes. Random code-constructions carry an uncertainty about the properties of any one chosen code and are harder to implement (in particular in terms of encoding, which we have barely touched upon here), but non-random constructions carry the risk that the extra structure permits the existence of low-weight codewords or "near-codewords".

A (w, v) *near-codeword* of a code with parity-check matrix H is a vector c with weight w such that Hc has weight v . Near-codewords with both small v and relatively small w tend to be error states from which the sum-product decoding algorithm cannot escape. So, for instance, [14] found for Margulis' codes with $p = 5, q = 11$ numerous $(12, 4)$ and $(14, 4)$ near-codewords, causing an error floor of about 10^{-6} .

10. Graph Covers and the Fundamental Polytope

Our decoding algorithm sometimes gets stuck on fake codewords, called pseudo-codewords. This is because, as noted earlier, every codeword produces a cycle but not necessarily vice versa. A tight characterization of this decoding failure was recently given by Koetter and Vontobel [10] in terms of graph covers. The point is that since the sum-product algorithm operates locally, it cannot distinguish between the Tanner graph and covers of the graph.

Consider, for example, the $[3, 0]$ -code with parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Its Tanner graph G is given in Figure 11. While strange to consider a zero-rate code, this example is ideal for investigating the problems with iterative decoding. Under an optimal decision rule, the decoding algorithm must output the all zero codeword independently of the initial log-likelihoods ℓ . On the other hand, a simple experiment reveals that the behavior of the decoding algorithm does indeed depend on ℓ .

Suppose $\mathbf{y} = y_1 y_2 y_3$ is received and let the initial log-likelihood at the three message nodes be ℓ_1, ℓ_2, ℓ_3 so that $\ell = (\ell_1, \ell_2, \ell_3)$. Fixing $\ell_1 = 0.013$, [10] found that as ℓ_2, ℓ_3 range from -5 to 5 , for the region in Figure 12 the algorithm failed to converge after 100 iterations. A closer experimental study showed that the region of convergence to the zero word is described well by $\ell_1 + \ell_2 + \ell_3 \geq 0$.

Consider now Figure 13. This graph G_H is a cubic cover of G . Every node of G is repeated three times and local adjacency relationships preserved. Also in Figure 13 is indicated a codeword for the cubic cover G_H that does not arise on G . Any locally operating message-passing algorithm will take into account all possible codewords on all possible covers of G . The codeword in Figure 13 is “closer” to y than the all zero codeword in a region that would correspond to a virtually present all one codeword.

It seems a formidable task to characterize all possible codewords introduced by the union of all finite covers of any degree (whose number grows faster than exponentially in the degree), but there is an elegant description, thanks to the fundamental polytope, which we now describe.

If G is a Tanner graph for a code C of length n , then a degree m cover \hat{G} is a Tanner graph of a code \hat{C} of length mn . Any codeword in C lifts to a codeword in \hat{C} and conversely, given a codeword \hat{c} in \hat{C} , we define

$$\omega_i(\hat{c}) := |\{j : \hat{c}_{i,j} = 1\}|/m$$

and set $\omega(\hat{c}) = (\omega_1(\hat{c}), \dots, \omega_n(\hat{c}))$.

Definition. A *pseudo-codeword* of C is any such $\omega(\hat{c})$ for any finite cover \hat{C} of C .

For instance, the codeword \hat{c} in Figure 13 yields pseudo-codeword $(2/3, 2/3, 2/3)$.

Theorem. Let a Tanner graph G_δ be given, consisting of a single parity-check node of degree δ and δ variable nodes, and C_δ be the corresponding code. Let \mathcal{P}_δ denote the set of pseudo-codewords $\omega(\hat{c})$ taken over the union of all covers of G_δ of all degrees. The closure of \mathcal{P} is the polytope

$$\overline{\mathcal{P}_\delta} = \{\omega \in \mathbf{R}^n : \omega = \mathbf{x}P_\delta, \mathbf{x} \in \mathbf{R}^{2^{\delta-1}}, 0 \leq x_i \leq 1, \sum_i x_i = 1\}$$

where the $2^{\delta-1} \times \delta$ matrix P_δ contains all binary even weight vectors of the form $(\sum_{i=1}^m c_i)/m$.

Let a Tanner graph G have check nodes f_1, \dots, f_r and message nodes c_1, \dots, c_n . Let \mathcal{P} be the set of pseudo-codewords taken over the union of all covers of G of all degrees. The closure of \mathcal{P} is the polytope

$$\overline{\mathcal{P}} = \{\omega \in \mathbf{R}^n : \omega_{\Gamma(f_i)} \in \overline{\mathcal{P}_\delta(f_i)}, 1 \leq i \leq r\}$$

$\overline{\mathcal{P}}$ is a convex body entirely inside the positive orthant, with one corner located at the origin. As an example of how this can be used, consider the $(3, 5)$ regular

LDPC code constructed in [26]. This is a very nice [155, 64, 20] binary linear code. Its parity-check matrix is 93×155 but because of redundant rows its rate is actually higher than $1 - 93/155$, namely $64/155 = 0.4129$. The underlying graph has girth 8, which together with the relatively large minimum distance 20 (28 is the largest known for a [155, 64]-code) makes it an outstanding candidate for iterative decoding. However, one easily finds a pseudo-codeword of small pseudo-weight and the automorphism group of the graph then yields at least 155 pseudo-codewords of this pseudo-weight (an example of MacKay-Postol's observation of the potential weaknesses of non-random LDPC code constructions - see the last section). Thus the large minimum distance of the code itself is largely irrelevant for the performance of iterative decoding.

A finer understanding of the suboptimal behavior of iterative decoding has in the last few sections been attributed to stopping sets, near-codewords, and the fundamental polytope. How are these interrelated? While the notion of stopping set is well-suited to the BEC, it does not work well with the AWGN channel. While the notion of near-codewords helps understand potential problems in the design of iteratively decodable codes, it is not as refined a notion as the fundamental polytope. Any near-codeword can be completed into a pseudo-codeword, giving a precise measure of the effect of the near-codeword.

BIBLIOGRAPHY

- [1] S.M.Aji and R.J.McEliece, The generalized distributive law, *IEEE Trans. Inform. Theory*, 46, 325–343, 2000.
- [2] L.R.Bahl, J.Cocke, F.Jelinek, and J.Raviv, Optimal decoding of linear codes for minimizing symbol error rate, *IEEE Trans. Inform. Theory*, 20, 284–287, 1974.
- [3] E.Berlekamp, R.McEliece, and H.van Tilborg, On the inherent intractability of certain coding problems, *IEEE Trans. Inform. Theory*, 24, 384–386, 1978.
- [4] C.Berrou, A. Glavieux, and P.Thitimajshima, Near Shannon limit error-correcting coding and decoding, *Proceedings of ICC '93*, 1064–1070, 1993.
- [5] S.-Y.Chung, G.D.Forney, Jr., T.J.Richardson, and R.Urbanke, On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit, *IEEE Comm. Letters*, 5, 58–60, 2001.
- [6] P.Elias, Coding for two noisy channels, *Information Theory*, 3rd London Symposium, 61–76, 1955.
- [7] Flarion Technologies, Vector-LDPC Coding Solution, www.flarion.com/products/vector.asp
- [8] G.D.Forney, Jr., The Viterbi algorithm, *Proc. IEEE*, 61, 268–278, 1973.
- [9] R.G.Gallager, *Low Density Parity-Check Codes*, MIT Press, Cambridge, MA 1963.
- [10] R.Koetter and P.Vontobel, Graph-covers and iterative decoding of finite length codes, *Turbo conference*, Brest, 2003.
- [11] M.Luby, M.Mitzenmacher, M.A.Shokrollahi, D.Spielman, and V.Stemann, Practical loss-resilient codes, *Proc. 29th Annual ACM Symposium on Theory of Computing*, 150–159, 1997.
- [12] D.J.C.MacKay, Good error correcting codes based on very sparse matrices, *IEEE Trans. Inform. Theory*, 45, 399–431, 1999.
- [13] D.J.C.MacKay, Gallager code resources,

www.inference.phy.cam.ac.uk/mackay/CodesFiles.html

- [14] D.J.C.MacKay and M.S.Postol, Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes, *Electronic Notes in Theoretical Computer Science*, 74, 2003.
- [15] G.A.Margulis, Explicit constructions of graphs without short cycles and low density codes, *Combinatorica*, 2, 71–78, 1982.
- [16] J.Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, Inc., 1988.
- [17] T.Richardson, A.Shokrollahi, and R.Urbanke, Design of capacity-approaching irregular low-density parity-check codes, *IEEE Trans. Inform. Theory*, 47, 619–637, 2001.
- [18] T.Richardson and R.Urbanke, *Modern Coding Theory, Lecture Notes*, <http://lthcwww.epfl.ch/papers/ics.ps>
- [19] J.Rosenthal and P.Vontobel, Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis, *Proc. 38th Annual Allerton Conference on Communication, Control, and Computing*, 248–257, 2000.
- [20] I.Sason, Codes on graphs and iterative decoding algorithms, www.ee.technion.ac.il/people/sason/slides_codes_on_graphs.pdf
- [21] C.E.Shannon, *A Mathematical Theory of Communication*, *Bell System Technical Journal*, vol. 27, 379–423, 1948 (Part I), 623–656 (Part II).
- [22] A.Shokrollahi, Codes and graphs, *Proc. of STACS 2000, Lecture Notes in Computer Science 1770*, 1–12, 2000.
- [23] A.Shokrollahi, LDPC codes: an introduction, www.ipm.ac.ir/IPM/homepage/Amin2.pdf
- [24] M.Sipser and D.Spielman, Expander codes, *IEEE Trans. Inform. Theory*, 42, 1710–1722, 1996.
- [25] R.M.Tanner, A recursive approach to low complexity codes, *IEEE Trans. Inform. Theory*, 27, 533–547, 1981.
- [26] R.M.Tanner, D.Sridhara, and T.Fuja, A class of group-structured LDPC codes, *Proc. of ICSTA 2001, Ambleside, England*, 2001.
- [27] A.J.Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Trans. Inform. Theory*, IT-13, 260–269, 1967.
- [28] N.Wiberg, H.-A.Loeliger, and R.Koetter, Codes and iterative decoding on general graphs, *European Transactions in Telecommunication*, 6, 513–525, 1995.
- [29] J.K.Wolf, A tutorial on low density parity check (LDPC) codes, ece-classweb.ucsd.edu/ece154c/LDPC.ppt, 2004.
- [30] V.V.Zyablov and M.S.Pinsker, Estimation of error-correction complexity of Gallager low-density codes, *Probl. Inform. Transm.*, 11, 18–28, 1976.

DEPARTMENTS OF MATHEMATICS, ELECTRICAL AND COMPUTER ENGINEERING, AND COMPUTER SCIENCES, UNIVERSITY OF WISCONSIN, MADISON, WI 53706, USA

E-mail address: `boston@math.wisc.edu`