Low-density parity-check codes: 
Design and decoding

Sarah J. Johnson *
Steven R. Weller †
School of Electrical Engineering and Computer Science
University of Newcastle
Callaghan, NSW 2308, Australia
email: {sarah,steve}@ee.newcastle.edu.au

Chapter in Wiley Encyclopedia of Telecommunications (J. Proakis, Ed.),
Technical Report EE02041
June 20, 2002

Abstract

In this chapter low-density parity-check (LDPC) codes are presented. Our main
focus is the design of LDPC codes and the algorithms used to decode them. The
chapter begins by introducing parity-check equations and codes defined by their
parity-check matrices. To introduce iterative decoding a hard-decision iterative algo-
rithm is discussed before the soft-decision iterative sum-product decoding algorithm
is presented. Following sections present the relationship between graph-based repre-
sentations of codes and the development of both the theoretical understanding and
implementation of iterative decoders, as well as the current methods for design of
LDPC codes. The chapter concludes with the connections of this work to other topics
and future directions in the area.

Keywords

Low-density parity-check codes, error correction, iterative decoding, sum-product algo-
rithm, min-sum algorithm, a posteriori decoding, Tanner graph, factor graph, graphical
models, Viterbi algorithm, soft-decision decoding, message-passing decoders

---

*Work supported by a CSIRO Telecommunications & Industrial Physics postgraduate scholarship and the Centre for Integrated Dynamics and Control (CIDAC).
†Work supported in part by Bell Laboratories Australia, Lucent Technologies, as well as the Australian Research Council under Linkage Project Grant LP0211210, and the Centre for Integrated Dynamics and Control (CIDAC).
1 Introduction

The publication of Claude Shannon’s 1948 paper, *A mathematical theory of communication* [1], marked the beginning of coding theory. In his paper, Shannon established that every communication channel has associated with it a number called the channel *capacity*. He proved that arbitrarily reliable communication is possible even through channels which corrupt the data sent over them, but only if information is transmitted at a rate less than the channel capacity.

In the simplest case, transmitted messages consist of strings of 0’s and 1’s, and errors introduced by the channel consist of bit inversions: 0 → 1 and 1 → 0. The essential idea of forward error control coding is to augment messages to produce codewords containing deliberately introduced redundancy, or *check* bits. With care, these check bits can be added in such a way that codewords are sufficiently distinct from one another so that the transmitted message can be correctly inferred at the receiver, even when some bits in the codeword are corrupted during transmission over the channel.

While Shannon’s noisy channel coding theorem establishes the existence of capacity-approaching codes, it provides no explicit guidance as to how the codes should be chosen, nor how messages can be recovered from the noise-corrupted channel output. The challenges to communicating reliably at rates close to the Shannon limit are therefore twofold: the first is to design sets of suitably distinct codewords, and the second is to devise methods for extracting estimates of transmitted messages from the output of a noise-contaminated channel, and to do so without excessive decoder complexity.

In this chapter, we consider code design and decoding for a family of error correction codes known as *low-density parity-check (LDPC)* block codes. In the simplest form of a parity-check code, a single parity-check equation provides for the detection, but not correction, of a single bit inversion in a received codeword. To permit correction of errors induced by channel noise, additional parity checks can be added at the expense of a decrease in the rate of transmission. Low-density parity-check codes are a special case of such codes. Here “low-density” refers to the sparsity of the parity-check matrix characterizing the code. Each parity-check equation checks few message bits, and each message bit is involved in only a few parity-check equations. A delicate balance exists in the construction of appropriate parity-check matrices, since excessive sparsity leads to uselessly weak codes.

First presented by Gallager in his 1962 thesis [2], [3], low-density parity-check codes are capable of performance extraordinarily close to the Shannon limit when appropriately decoded. The key to extracting maximal benefit from LDPC codes is *soft decision* decoding, which starts with a more subtle model for channel-induced errors than simple bit inversions. Rather than requiring that the receiver initially make *hard decisions* at the channel output, and so insisting that each received bit be assessed as either 0 or 1, whatever is the more likely, soft decision decoders use knowledge of the channel noise statistics to feed probabilistic (or “soft”) information on received bits into the decoder.

The final ingredient in implementing soft decision decoders with acceptable decoder complexity are *iterative* schemes which handle the soft information in an efficient manner. Soft iterative decoders for LDPC codes make essential use of *graphs* to represent codes, passing probabilistic messages along the edges of the graph. The use of graphs for iterative decoding can be traced to Gallager, although for over 30 years barely a handful of researchers pursued the consequences of Gallager’s work. This situation changed dramatically with the independent rediscovery of LDPC codes by several researchers in the mid-1990s, and graph-based representations of codes are now an integral feature in the development of both the theoretical understanding and implementation of iterative...
In this chapter, we begin by introducing parity-checks and codes defined by their parity-check matrices. To introduce iterative decoding we present in Section 3 a hard-decision iterative algorithm which is not very powerful, but suggestive of how graph-based iterative decoding algorithms work. The soft-decision iterative decoding algorithm for LDPC codes known as sum-product decoding is presented in Section 4. Section 5 presents the relationship between the codes and the decoding algorithm as expressed in the graphical representation of LDPC codes and Section 6 considers the design of LDPC codes. The chapter concludes with the connections of this work to other topics and future directions in the area.

2 Low-density parity-check codes

2.1 Parity-check codes

The simplest possible error detection scheme is the single parity check, which involves the addition of a single extra bit to a binary message. Whether this parity bit should be a 0 or a 1 depends on whether even or odd parity is being used. In even parity, the additional bit added to each message ensures an even number of 1s in each transmitted codeword. For example, since the 7-bit ASCII code for the letter S is 1010011, a parity bit is added as the eighth bit. If even parity is being used, the value of the parity bit is 0 to form the codeword 10100110.

More formally, for the 7-bit ASCII plus even parity code we define a codeword $c$ to have the following structure:

$$c = c_1 c_2 c_3 c_4 c_5 c_6 c_8,$$

where each $c_i$ is either 0 or 1, and every codeword satisfies the constraint

$$c_1 \oplus c_2 \oplus c_3 \oplus c_4 \oplus c_5 \oplus c_6 \oplus c_7 \oplus c_8 = 0. \quad (1)$$

Here the symbol $\oplus$ represents modulo-2 addition, which is equal to 1 if the ordinary sum is odd and 0 if the ordinary sum is even. While the inversion of a single bit due to channel noise can easily be detected with a single parity check—as (1) is no longer satisfied by the noise-corrupted codeword—this code is not sufficiently powerful to indicate which bit, or bits, were inverted. Moreover, since any even number of bit inversions produces a word satisfying the constraint (1), any even numbers of errors go undetected by this simple code.

One measure of the ability of a code to detect errors is the minimum distance of the code. The Hamming distance between two codewords is defined as the number of bit positions in which they differ. For example the codewords 10100110 and 10000111 differ in positions 3 and 8 so the Hamming distance between them is 2. The minimum distance of a code, $d_{\text{min}}$, is defined as the smallest Hamming distance between any pair of codewords in the code. For the even parity code $d_{\text{min}} = 2$, so the corruption of two bits in a codeword can result in another valid codeword and will consequently not be detected.

Detecting more than a single bit error calls for increased redundancy in the form of additional parity checks. To illustrate, suppose we define a codeword $c$ to have the following structure:

$$c = c_1 c_2 c_3 c_4 c_5 c_6,$$
where each \( c_i \) is either 0 or 1, and \( c \) is constrained by three parity-check equations:

\[
\begin{align*}
    c_1 &\oplus c_2 &\oplus c_4 = 0 \\
    c_2 &\oplus c_3 &\oplus c_5 = 0 \\
    c_1 &\oplus c_2 &\oplus c_3 &\oplus c_6 = 0
\end{align*}
\]  

\[
[H] = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
c_3 \\
c_4 \\
c_5 \\
c_6
\end{bmatrix} = \begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}.
\]  

(2)

In matrix form we have that \( c = [c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6] \) is a codeword if and only if it satisfies the constraint

\[
Hc^T = 0,
\]  

(3)

where the parity-check matrix, \( H \), contains the set of parity-check equations which define the code. To generate the codeword for a given message, the code constraints can be rewritten in the form

\[
\begin{align*}
    c_4 &= c_1 \oplus c_2 \\
    c_5 &= c_2 \oplus c_3 \\
    c_6 &= c_1 \oplus c_2 \oplus c_3
\end{align*}
\]  

\[
\begin{bmatrix}
c_4 \\
1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1
\end{bmatrix} = \begin{bmatrix}
c_1 & c_2 & c_3 & c_4 & c_5 & c_6
\end{bmatrix}
\]  

(4)

where bits \( c_1, c_2 \), and \( c_3 \) contain the 3-bit message, and parity-check bits \( c_4, c_5 \) and \( c_6 \) are calculated from the message. Thus for example the message 110 produces parity-check bits \( c_4 = 1 \oplus 1 = 0, c_5 = 1 \oplus 0 = 1 \) and \( c_6 = 1 \oplus 1 \oplus 0 = 0 \), and hence the codeword 110010. The matrix \( G \) is the generator matrix of the code. Substituting each of the \( 2^3 = 8 \) distinct messages \( c_1 c_2 c_3 = 000, 001, \ldots, 111 \) into equation (4) yields the following set of codewords:

\[
\begin{align*}
    &000000 \ 001011 \ 010111 \ 011100 \\
    &100101 \ 101100 \ 110010 \ 111001.
\end{align*}
\]  

(5)

The reception of a word which is not in this set of codewords can be detected using the parity-check constraint equation (3). Suppose for example that the word \( r = 101011 \) is received from the channel. Substitution into equation (3) gives

\[
Hr^T = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix} = \begin{bmatrix}
1 \\
0 \\
1
\end{bmatrix}
\]  

(6)

which is nonzero and so the word 101011 is not a codeword of our code.

To go further and correct the error requires that the decoder determine the codeword most likely to have been sent. Since it is reasonable to assume that a small number of errors is more likely to occur than a large number, the required codeword is the one closest in Hamming distance to the received word. By comparison of the received word \( r = 101011 \) with each of the codewords in (5), the closest codeword is \( c = 001011 \), which is at Hamming distance 1 from \( r \). The minimum distance of this code is 3, so a single bit error always results in a word closer to the codeword which was sent than any other codeword, and
hence can always be corrected. In general, for a code with minimum distance $d_{\text{min}}$, $e$ bit errors can always be corrected by choosing the closest codeword whenever

$$e \leq \lfloor (d_{\text{min}} - 1)/2 \rfloor,$$

(7)

where $[x]$ is the largest integer that is at most $x$.

Error correction by direct search is feasible only when the number of distinct codewords is small. For codes with thousands of bits in a codeword, it becomes far too computationally expensive to directly compare the received word with every codeword in the code, and numerous ingenious methods have been devised, including choosing codes which are cyclic or, as presented in this chapter, devising iterative methods to decode the received word.

### 2.2 low-density codes

LDPC codes are parity-check codes with the requirement that $H$ is “low-density” so that the vast majority of entries are zero. A parity-check matrix is regular if each code bit is contained in a fixed number, $w_c$, of parity checks and each parity-check equation contains a fixed number, $w_r$, of code bits. If an LDPC code is described by a regular parity-check matrix it is called a $(w_c, w_r)$-regular LDPC code otherwise it is an irregular LDPC code.

Importantly, an error correction code can be described by more than one parity-check matrix, where $H$ is a valid parity-check matrix for a code provided that (3) holds for all codewords in the code. Two parity-check matrices for the same code need not even have the same number of rows; what is required is that the rank over GF(2) of both be the same, since the number of message bits in a binary code is

$$k = n - \text{rank}_2(H).$$

(8)

where $\text{rank}_2(H)$ is the number of rows in $H$ which are linearly dependent over GF(2). To illustrate we give a regular parity-check matrix for the code of (2) with $w_c = 2$, $w_r = 3$ and $\text{rank}_2(H) = 3$ which satisfies (3):

$$H = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1
\end{bmatrix}.$$  

(9)

A Tanner graph is a graphical representation of $H$ which facilitates iterative decoding of the code. The Tanner graph consists of two sets of vertices: bit vertices (or bit nodes), and parity-check vertices (or check nodes) where there is a parity-check vertex for every parity-check equation in $H$ and a bit vertex for every codeword bit. Each parity-check vertex is connected by an edge to the bit vertices corresponding to the code bits included in that parity-check equation. The Tanner graph of the parity-check matrix (9) is shown in Fig. 1. As the number of edges leaving the bit vertices must equal the number of edges leaving the parity-check vertices it follows that for a regular code,

$$m \cdot w_r = n \cdot w_c$$

(10)

A cycle in a Tanner graph is a sequence of connected vertices which start and end at the same vertex in the graph, and which contain other vertices no more than once. The length
of a cycle is the number of edges it contains, and the \textit{girth} of a graph is the size of its smallest cycle. A cycle of size 6 is shown in bold in Fig. 1.

Traditionally, the parity-check matrices of LDPC codes have been defined pseudo-randomly subject to the requirement that $H$ be sparse, and code construction of binary LDPC codes involves randomly assigning a small number of the values in an all-zero matrix to be 1. The lack of any obvious algebraic structure in randomly constructed LDPC codes sets them apart from traditional parity-check codes. The properties of LDPC codes are often considered in terms of the ensemble performance of all possible codes that meet a given set of constraints, reminiscent of the methods used by Shannon in proving his noisy channel coding theorem. Recent research has considered the design of LDPC codes with specific properties, such as large girth, and we describe in Sections 5 and 6 methods to design LDPC codes. For sum-product decoding however, no additional structure beyond a sparse parity-check matrix is required and in the following two sections we present decoding algorithms requiring only the existence of a sparse $H$.

3 \textbf{Iterative decoding}

To illustrate the process of iterative decoding a \textit{bit-flipping} algorithm is presented, based on an initial hard decision (0 or 1) assessment of each received bit. An essential part of iterative decoding is the passing of messages between the nodes of the Tanner graph of the code. For the bit-flipping algorithm the messages are simple: a bit node sends a message to each of the check nodes to which it is connected declaring if it is a 1 or a 0, and each check node sends a message to each of the bit nodes to which it is connected, declaring whether the parity check is satisfied or not. The sum-product algorithm for LDPC codes operates similarly but with more complicated messages.

The bit-flipping decoding algorithm is as follows:

\textbf{Step 1. Initialization:} Each bit node is assigned the bit value received from the channel, and sends messages to the check nodes to which it is connected indicating this value.

\textbf{Step 2. Parity update:} Using the messages from the bit nodes, each check node calculates whether or not its parity-check equation is satisfied. If all parity checks are satisfied the algorithm terminates, otherwise each check node sends messages to the bit nodes to which it is connected indicating if the parity check is satisfied or not.

\textbf{Step 3. Bit update:} If the majority of the messages received by each bit node are “not satisfied,” the bit node flips its current value, otherwise the value is retained. If the maximum number of allowed iterations is reached the algorithm terminates and a failure to converge is reported, otherwise each bit node sends new messages to the check nodes to which it is connected, indicating its value and the algorithm returns to Step 2.

To illustrate the operation of the bit-flipping decoder, we take the code of (9) and again assume that the codeword $c = 001011$ is sent, and the word $r = 101011$ is received from the channel. The steps required to decode this received word are shown in Fig. 2. In Step 1 the bit values are initialized to be 1, 0, 1, 0, 1, and 1, respectively, and messages are sent to the check nodes indicating these values. In Step 2 each parity check is satisfied only if an even number of the bits included in the parity check are 1. For the first and third check nodes this is not the case, and so they send “not satisfied” messages to the bits to which
they are connected. In Step 3 the first bit has the majority of its messages indicating “not satisfied” and so flips its value from 1 to 0. Step 2 is repeated and since now all four parity-checks are satisfied, the algorithm halts and returns $c = 001011$ as the decoded codeword. The received word has therefore been correctly decoded without requiring an explicit search over all possible codewords.

The existence of cycles in the Tanner graph of a code reduces the effectiveness of the iterative decoding process. To illustrate the detrimental effect of a 4-cycle we adjust the code of the previous example to obtain the new code shown in Fig. 3. A valid codeword for this code is 001001 but again we assume that the first bit is corrupted, so that $r = 101001$ is received from the channel. The steps involved to decode this received word using the bit-flipping algorithm are shown in Fig. 3. In Step 1 the initial bit values are 1, 0, 1, 0, 0, and 1, respectively, and messages are sent to the check nodes indicating these values. Step 2 reveals that the first and second parity checks are not satisfied. In Step 3 both the first and second bits have the majority of their messages indicating “not satisfied” and so both flip their bit values. When Step 2 is repeated we see that the first and second parity checks are again not satisfied. Further iterations at this point simply cause the first two bits to flip their values in such a way that one of them is always incorrect; the algorithm fails to converge. As a result of the 4-cycle each of the first two codeword bits are involved in the same two parity checks, and so when neither of the parity checks are satisfied, it is not possible to determine which bit is causing the error.

4 Sum-product decoding

The sum-product decoding algorithm, also called belief propagation decoding, was first introduced by Gallager in his 1962 thesis where he applied it to the decoding of pseudo-randomly constructed LDPC codes. For block lengths of $10^7$, highly optimized irregular LDPC codes decoded with the sum-product algorithm are now known to be capable of approaching the Shannon limit to within hundreds of a decibel on the binary-input additive white Gaussian noise (AWGN) channel. In the early 1960s, however, limited computing resources prevented Gallager from demonstrating the capabilities of iteratively decoded LDPC codes for blocklengths longer than around 500, and for over 30 years his work was ignored by all but a handful of researchers. It was only re-discovered by several researches in the wake of turbo decoding [4], which has subsequently been realized as an instance of the sum-product algorithm.

The sum-product algorithm can be thought of as being similar to the bit-flipping algorithm described in the previous section, but with the messages representing each decision (check met, or bit value equal to 1) now probabilistic values represented by log-likelihood ratios. Whereas with bit flipping decoding an initial hard decision is made on the signal form the channel, what is actually received is a string of real values where the sign of the received value represents a binary decision, 0 if positive and 1 if negative, and the magnitude of the received value is a measure of the confidence in that decision. A shortcoming of using only hard decisions when decoding is that the information relating to the confidence of the signal, the soft information, is discarded. Soft decision decoders such as the sum-product decoder make use of the soft received information, together with knowledge of the channel properties, to obtain probabilistic expressions for the transmitted signal.

For a binary signal, if $p$ is the probability of a 1, then $1 - p$ is the probability of a 0.
which is represented as a log-likelihood ratio (LLR) by

$$\text{LLR}(p) = \log_e \frac{1 - p}{p}. \quad (11)$$

The sign of LLR($p$) is the hard decision and the magnitude $|\text{LLR}(p)|$ is the reliability of this decision. One benefit of the logarithmic representation of probabilities is that whereas probabilities need to be multiplied, log-likelihood ratios need only be added reducing implementation complexity.

The aim of sum-product decoding is to compute the a posteriori probability (APP) for each codeword bit, $P_i = P\{c_i = 1|N\}$, which is the probability that the $i$th codeword bit is a 1 conditional on the event $N$ that all parity-check constraints are satisfied. The intrinsic or a priori probability, $P_i^{\text{int}}$, is the original bit probability independent of knowledge of the code constraints, and the extrinsic probability $P_i^{\text{ext}}$ represents what has been learnt from the event $N$.

The sum-product algorithm iteratively computes an approximation of the APP values of the code bits. The approximations are exact if the code is cycle free. Extrinsic information gained from the parity-check constraints in one iteration is used as a priori information for the subsequent iteration. The extrinsic bit information obtained from a parity-check constraint is independent of the a priori value for that bit at the start of the iteration. The extrinsic information provided in subsequent iterations remains independent of the original a priori probability until that information is returned via a cycle.

To compute the extrinsic probability of a code bit $i$ from the $j$th parity-check equation we determine the probability that the parity-check equation is satisfied if bit $i$ is assumed to be a 1, which is the probability that an odd number of the other codeword bits are a 1,

$$P_{i,j} = \frac{1}{2} + \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2P_{i'}^{\text{int}}). \quad (12)$$

The notation $B_j$ represents the set of column locations of the bits in the $j$th parity-check equation of the code considered. Similarly, $A_i$ is the set of row locations of the parity-check equations which check on the $i$th code bit of the code. To put (12) into log-likelihood notation we note that

$$\tanh \left( \frac{1}{2} \log_e \left( \frac{1 - p}{p} \right) \right) = 1 - 2p,$$

to give

$$\text{LLR}(P_{i,j}^{\text{ext}}) = \log_e \left( \frac{1 + \prod_{i' \in B_j, i' \neq i} \tanh(\text{LLR}(P_{i'}^{\text{int}})/2)}{1 - \prod_{i' \in B_j, i' \neq i} \tanh(\text{LLR}(P_{i'}^{\text{int}})/2)} \right).$$

The LLR of the estimated APP of the $i$th bit at each iteration is then simply

$$\text{LLR}(P_i) = \text{LLR}(P_i^{\text{int}}) + \sum_{j \in A_i} \text{LLR}(P_{i,j}^{\text{ext}}).$$

Step 1. Initialization: The initial message sent from bit node $i$ to the check node $j$ is the LLR of the (soft) received signal $y_i$ given knowledge of the channel properties. For an AWGN channel with signal-to-noise ratio $E_b/N_0$ this is:

$$L_{i,j} = R_i = 4y_i \frac{E_b}{N_0}. \quad (13)$$
Step 2. **Check-to-bit:** The extrinsic LLR from check node $j$ to bit node $i$ is the probability that parity-check $j$ is satisfied if bit $i$ is assumed to be a 1:

$$E_{i,j} = \log_e \left( \frac{1 + \prod_{i' \in B_j, i' \neq i} \tanh(L_{i',j}/2)}{1 - \prod_{i' \in B_j, i' \neq i} \tanh(L_{i',j}/2)} \right).$$  \hspace{1cm} (14)

Step 3. **Codeword test:** The combined LLR is the sum of the extrinsic LLRs and the original LLR calculated in Step 1:

$$L_i = \sum_{j \in A_i} E_{i,j} + R_i. \hspace{1cm} (15)$$

For each bit a hard decision is made

$$z_i = \begin{cases} 1, & L_i \leq 0 \\ 0, & L_i > 0. \end{cases}$$

If $z = [z_1, \ldots, z_n]$ is a valid codeword ($Hz^T = 0$), or if the maximum number of allowed iterations have been completed, the algorithm terminates.

Step 4. **Bit-to-check:** The message sent by each bit node to the check nodes to which it is connected is similar to (15), except that bit $i$ sends to check node $j$ a LLR calculated without using the information from check node $j$:

$$L_{i,j} = \sum_{j' \in A_i, j' \neq j} E_{i,j'} + R_i. \hspace{1cm} (16)$$

Return to Step 2.

The application of equations (14) and (16) to the code in (9) is demonstrated in Fig. 4. The extrinsic information passed from a check node to a bit node is independent of the probability value for that bit. The extrinsic information from the check nodes is then used as a priori information for the bit nodes in the subsequent iteration.

To illustrate the power of sum-product decoding we revisit the example of Fig. 2 where the codeword sent is 0 0 1 0 1 1. Suppose that the channel is AWGN with $E_b/N_0 = 1.25$ and the received signal is $y = -0.1 \ 0.5 \ -0.8 \ 1.0 \ -0.7 \ 0.5$. There are now two bits in error if the hard decision of the signal is considered, bits 1 and 6. Fig. 5 illustrates the operation of the sum-product decoding algorithm as described in equations (13)–(16) to decode this received signal. The existence on an exact termination rule for the sum-product algorithm (is $Hz^T = 0$) has two important benefits; the first is that a failure to converge is always detected and the second that extra iterations are not uselessly applied once a solution has been found.

There are variations to the sum-product algorithm presented here. The min-sum algorithm for example simplifies the calculation of (14) by recognizing that the term corresponding to the smallest $L_{i,j}$ dominates the product term and so the product can be approximated by a minimum; the resulting algorithm thus requires calculation of only minimums and additions. An alternative approach to bridging the gap between the error performance of sum-product decoding and maximum-likelihood (ML) decoding finishes each iteration of sum-product decoding with ordered statistic decoding, with the algorithm terminating when a fixed number of iterations have returned the same codeword [5].
5 Codes, graphs and cycles

The relationship between LDPC codes and their decoding is closely associated with the graph-based representations of the codes. The most obvious example of this is the link between the existence of cycles in the Tanner graph of the code to both the analysis and performance of sum-product decoding of the code. In his work, Gallager used a graphical representation of the bit and parity-check sets of regular LDPC codes, to describe the application of iterative APP decoding. The systematic study of codes on graphs however is largely due to Tanner who, in 1981, extended the single parity-check constraints of Gallager’s LDPC codes to arbitrary linear code constraints, foresaw the advantages for very large-scale integration (VLSI) implementations of iterative decoders, and formalized the use of bipartite graphs for describing families of codes [6]. In doing so, Tanner also founded the topic of algebraic methods for constructing graphs suitable for sum-product decoding.

By proving the convergence of the sum-product algorithm for codes whose graphs are free of cycles, Tanner was also the first to formally recognize the importance of cycle-free graphs in the context of iterative decoding. The effect of cycles on the practical performance of LDPC codes was demonstrated by simulation experiments when LDPC codes were rediscovered by MacKay and Neal [7] (among others) in the mid-1990s, and the beneficial effects of using graphs free of short cycles were shown [8]. Given the detrimental effects of cycles on the convergence of iterative decoders, it is natural to seek strong codes whose Tanner graphs are free of cycles. An important negative result in this direction was established by Etzion et al. [9], who showed that for linear codes of rate \( k/n \geq 0.5 \) that can be represented by a Tanner graph without cycles, the minimum distance is at most 2.

As the existence of cycles in a graph makes analysis of the decoding algorithm difficult, most analysis considers the asymptotic performance of infinitely many decoding iterations on a graph with asymptotically unbounded girth. This analysis provides thresholds to the performance of LDPC codes with sum-product decoding. Also, as we will see in the following section, this process can be used to select LDPC code parameters which improve the threshold values, a process which works well even if the resulting code does contain cycles.

To date very little analysis has been presented regarding the convergence of iterative decoding methods on graphs with cycles, and the majority of the work in this area can be found in [10]. Gallager suggested that the dependencies introduced by cycles have a relatively minor effect and tend to cancel each other out somewhat. This “seems to work” philosophy has underlined the performance of sum-product decoding on graphs with cycles for much of the (short) history of the topic. It is only recently that exact analysis on the expected performance of codes with cycles has emerged. The recent paper of Di et al. [11], uses finite length analysis to give the exact average bit and block error probabilities for a regular ensemble of LDPC codes over the binary erasure channel when decoded iteratively, however there is as yet no such analysis for irregular codes or more general channel models.

Besides cycles, Sipser and Spielman [12] showed that the expansion of the graph is a significant factor in the application of iterative decoding. Using only a simple hard decision decoding algorithm they proved that a fixed fraction of errors in an LDPC code can be corrected in linear time provided that the Tanner graph of the code is a good enough expander. That is, any subset \( S \) of bit vertices of size \( m \) or less is connected to at least \( \epsilon |S| \) constraint vertices, for some defined \( m \) and \( \epsilon \).
6 Designing LDPC codes

For the most part, LDPC codes are designed by first choosing the required blocklength and node degree distributions, then pseudo-randomly constructing a parity-check matrix, or graph, with these properties. A generator matrix for the code can then be found using elementary matrix row operations [8]. Gallager, for example, considered the ensemble of all \((w_r, w_c)\)-regular matrices with rows divided into \(w_c\) submatrices, the first containing \(w_r\) copies of the identity matrix and with subsequent submatrices being random column permutations of the first. Using ensembles of matrices defined in this way Gallager was able to find the maximum crossover probability of the binary symmetric channel (BSC) for which LDPC codes could be used to transmit information reliably using a simple hard decision decoding algorithm.

Luby et al. extended the class of LDPC ensembles to those with irregular node degrees and showed that irregular codes are capable of outperforming regular codes [13]. In extending Gallagers analysis to irregular ensembles, Luby et al. introduced tools based on linear programming for designing irregular codes ensembles for which the maximum allowed crossover probability of the binary symmetric channel is optimized [14]. Resulting from this work are the so-called “tornado codes,” a family of codes that approach the capacity of the erasure channel and can be encoded and decoded in linear time.

Richardson and Urbanke extended the work of Luby et al. to any binary input memoryless channel and to soft decision message passing decoding [15]. They determined the capacity of message passing decoders applied to LDPC code ensembles by a method called density evolution. For sum-product decoding density evolution makes it possible to determine the corresponding capacity to any degree of accuracy and hence determine the ensemble with node degree distribution which gives the best capacity. Once a code ensemble has been chosen a code from that ensemble is realized pseudo-randomly. By carefully choosing a code from an optimized ensemble Chung et al. have demonstrated the best performance to date of LDPC codes in terms of approaching the Shannon limit [16].

A recent development in the design of LDPC codes is the introduction of algebraic LDPC codes, the most promising of which are the finite geometry codes proposed by Lucas et al. [17] which are cyclic and described by sparse 4-cycle free graphs. An important outcome of this work with finite geometry codes was the demonstration that highly redundant parity-check matrices can lead to very good iterative decoding performances without the need for very long blocklengths. While the probability of a random graph having a highly redundant parity-check matrix is vanishingly small, the field of combinatorial designs offers a rich source of algebraic constructions for matrices which are both sparse and redundant. In particular, there has been much recent interest in balanced incomplete block designs (BIBDs) to produce sparse matrices for LDPC codes which are 4-cycle free. For codes with greater girth, generalized quadrangle designs give the maximum possible girth for a graph with given diameter [18]. Both generalized quadrangles and BIBDs are subsets of the more general class of combinatorial structures called partial geometries, a possible source of further good algebraic LDPC codes [19].

In comparison with more traditional forms of error correcting codes, the minimum distance of LDPC codes plays a substantially reduced role. There are two reasons for this: firstly, the lack of any obvious algebraic structure in pseudo-randomly constructed LDPC codes makes the calculation of minimum distance infeasible for long codes, and most analyses focus on the average distance function for an ensemble of LDPC codes. Secondly, the absence of conspicuous flattening of the bit-error rate (BER) curve at moderate to high signal-to-noise ratios (the so-called “error floor”) strongly suggests that minimum distance
properties are simply not as important for LDPC codes as for traditional codes. Indeed, it has recently been established that to achieve capacity on the binary erasure channel when using irregular LDPC codes, the codes cannot have large minimum distances [20].

7 Connections and future directions

Following the rediscovery of Gallager’s iterative LDPC decoding algorithm in the mid-1990s, the notion of an iterative algorithm operating on a graph has been generalized and is now capable of unifying a wide range of apparently different algorithms from the domains of digital communications, signal processing, and even artificial intelligence.

An important generalization of Tanner graphs was presented by Wiberg in his 1996 PhD thesis [21]. Wiberg introduced state variables into the graphical framework, thereby establishing a connection between codes on graphs and the trellis complexity of codes, and was the first to observe that on cycle-free graphs, the sum-product (respectively, min-sum) algorithm performs APP (respectively, ML) decoding.

In an even more general setting, the role of the Tanner graph is taken by a factor graph [22]. Central to the unification of message-passing algorithms via factor graphs is the recognition that many computationally efficient signal processing algorithms exploit the way in which a global cost function acting on many variables can be factorized into the product of simpler local functions, each of which operates on a subset of the variables. In this setting a (bipartite) factor graph encodes the factorization of the global cost function, with each local function node connected by edges only to those variable nodes associated with its arguments.

The sum-product algorithm operating on a factor graph uses message-passing to solve the marginalize product-of-functions (MPF) problem which lies at the heart of many signal processing problems. In addition to the iterative decoding of LDPC codes, specific instances of the sum-product algorithm operating on suitably defined factor graphs include the forward/backward algorithm (also known as the Bahl–Cocke–Jelinek–Raviv algorithm [23] or APP decoding algorithm), the Viterbi algorithm, the Kalman filter, Pearl’s belief propagation algorithm for Bayesian networks, and the iterative decoding of “turbo codes,” or parallel concatenated convolutional codes.

For high-performance applications, LDPC codes are naturally seen as competitors to turbo codes. LDPC codes are capable of outperforming turbo codes for block lengths greater than around 10^5, and the error floors of LDPC codes at BERs below about 10^-5 are typically much less pronounced than those of turbo codes. Moreover, the inherent parallelism of the sum-product decoding algorithm is more readily exploited with LDPC codes than their turbo counterparts, where block interleavers pose formidable challenges to achieving high throughput [24]. Despite these impressive advantages, LDPC codes lag behind turbo codes in real-world applications. The exceptional simulation performance of the original turbo codes [4, 25] generated intense interest in these codes, and variants of them were subsequently incorporated into proposals for third-generation (3G) wireless systems such as the Third Generation Partnership Project (3GPP), a global consortium of standards-setting organizations [26]. Whatever performance advantages of very long LDPC codes over turbo codes there may be, the invention of turbo codes some three years prior to the (re-)discovery of LDPC codes has given them a distinct advantage in wireless communications, where blocklengths of at most several thousand are typical, and where compliance with global standards is paramount.

One serious shortcoming of LDPC codes is their potentially high encoding complexity,
which is in general quadratic in the blocklength, and compares poorly with the linear time encoding of turbo codes. Finding computationally efficient encoders is therefore critical for LDPC codes to be considered as serious contenders for replacing turbo codes in future generations of forward error correction devices. Several approaches have been suggested, including the manipulation of the parity-check matrix to establish that while the complexity is, strictly speaking, quadratic, the actual number of encoding operations grows essentially linearly with blocklength. For some irregular LDPC codes whose degree distributions have been optimized to allow transmission near to capacity, the encoding complexity can be shown to be truly linear in blocklength [5]. A very different approach to the encoding complexity problem is to employ cyclic, or quasi-cyclic, codes as LDPC codes, as encoding can be achieved in linear time using simple feedback shift registers [27].

While encoding complexity needs to be addressed for practical applications, there are two (inter-related) issues likely to dominate future theoretical investigations of LDPC codes. The first of these is

- Finding exact expressions for bit and block erasure probability for the LDPC ensembles under ML decoding. Comparing the expressions for the iterative decoding of the LDPC ensembles with the ones for the ML decoding of LDPC ensembles and the ones for the ML of standard random ensembles it will then be possible to assess how much loss id due to the structure of the codes and how much loss is due to the suboptimum decoding.

- decoding on graphs with cycles

References


Figure 1: Tanner graph representation of the parity-check matrix in (9). A 6-cycle is shown in bold.

Figure 2: Bit-flipping decoding of the received word $r = 101011$. Each sub-figure indicates the decision made at each step of the decoding algorithm based on the messages from the previous step. A cross ($\times$) represents that the parity check is not satisfied while a tick ($\checkmark$) indicates that it is satisfied. For the messages, a dashed arrow corresponds to the messages “bit = 0” or “check not satisfied,” while a solid arrow corresponds to “bit = 1” or “check satisfied”.
Figure 3: Bit-flipping decoding of the received word \( r = 101001 \). A 4-cycle is shown in bold in the first sub-figure.

Figure 4: An example of the messages for Sum-product decoding. Calculation of the extrinsic message sent to bit 1 depends on messages from bits 2 and 4 but not from bit 1. Similarly, the message sent to check 1 is independent of the message just received from it.
Figure 5: The operation of sum-product decoding on the code from (9) when the codeword [001011] sent through an AWGN channel with $E_b/N_0 = 1.25$ returns the vector $[-0.525 - 45 - 3.525]$. 