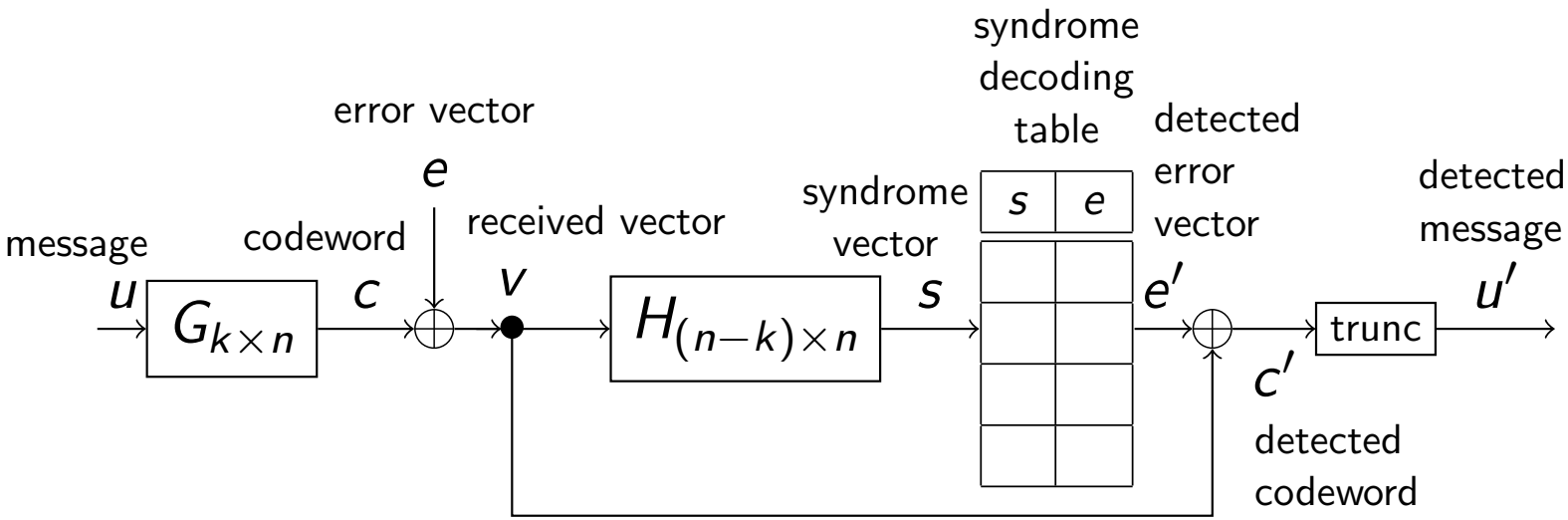
Binary Golay codes, CRC codes, LDPC codes

Coding Technology

Illés Horváth

2025/09/26

The binary linear coding scheme



Perfect binary codes

A binary code is perfect if it can correct t errors and the Hamming bound holds with equality:

$$\sum_{i=0}^{t} \binom{n}{i} = 2^{n-k}.$$

We have already seen two types of perfect binary codes:

- the $n\times$ repeater codes (for odd values of n) that can correct $t = \lfloor \frac{n-1}{2} \rfloor$ errors;
- Hamming codes can correct $t = 1$ error.

Are there any more perfect codes?

Theorem (Tietäväinen)

Apart from the above codes, the only perfect binary linear code is the $C(23, 12)$ Golay code that can correct $t = 3$ errors.

(No proof.)

Binary Golay codes

Golay designed two pairs of codes, a pair of binary codes and a pair of ternary codes (with digits $0, 1$ and 2). We only consider the binary Golay codes now.

The binary Golay codes are two specific codes:

- the $C(23, 12)$ perfect binary Golay code, and
- the $C(24, 12)$ extended binary Golay code.

Similar to Hamming codes, the $C(24, 12)$ code can be obtained from the $C(23, 12)$ code by adding a parity bit, and the $C(23, 12)$ code can be obtained from the $C(24, 12)$ by puncturing a bit.

Golay codes are highly symmetric, and there are several different ways to design them (resulting in the same code).

Binary Golay codes – Turyn's construction

The following is Turyn's construction, deriving the $C(24, 12)$ Golay code from the $C(8, 4)$ extended Hamming code.

Consider the following two matrices (both generate codes equivalent to the $C(8, 4)$ extended Hamming code):

$$G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \quad G_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The generator matrix G of the $C(24, 12)$ Golay code has the following block form (all blocks are 4×8):

$$G = \begin{bmatrix} G_1 & 0 & G_1 \\ 0 & G_1 & G_1 \\ G_2 & G_2 & G_2 \end{bmatrix}$$

Binary Golay codes – Turyn's construction

Theorem

For the $C(24, 12)$ code with generator G,

$$d_{\min} = \min_{c \neq 0} w(c) = 8.$$

Proof 1. List all 2^{12} codewords and check that every nonzero codeword c has $w(c) \geq 8$.

Proof 2. We explore the properties of this code step by step.

Proving $\min_{c \neq 0} w(c) = 8$ will be done in two main steps:

1. For every codeword c, $w(c)$ is divisible by 4.
2. There is no codeword c with $w(c) = 4$.

Binary Golay codes – Turyn's construction

- The only common codewords for G_1 and G_2 are (00000000) and (11111111).
- All other codewords in both codes have weight 4.
- If a is a codeword from the first code and b a codeword from the second code, then a and b are orthogonal.
- G_1 is self-dual ($G_1 G_1^T = 0$), so any two rows of G_1 are orthogonal. Same for G_2.
- Any two codewords of G_1 are orthogonal. Same for G_2.
- Any two rows of G are orthogonal.
- Any two codewords of G are orthogonal.
- If c and c' are orthogonal vectors with $4|w(c)$ and $4|w(c')$, then $4|w(c+c')$.
- Any row of G has weight divisible by 4.
- Any codeword of G has weight divisible by 4.

Binary Golay codes – Turyn's construction

Due to the block structure of G, all codewords have the form

$$c = (a + b|a' + b|a + a' + b)$$

where a and a' are codewords from the first code (G_1), and b is a codeword from the second code (G_2).

The vectors $a + b, a' + b, a + a' + b$ all have even weight; if $w(c) = 4$, then this is possible only of one of the three vectors has 0 weight.

- if $w(a + b) = 0$, then $a = b$, but the only common codewords are (00000000) and (11111111), so either $c = (0|a'|a')$ or $c = (0|a' + 1|a')$, and for both cases, $w(c) \neq 4$.
- Same if $w(a' + b) = 0$.
- if $w(a + a' + b) = 0$, then $c = (a'|a|0)$;
  - if $w(a') = 0$, then $a = b$, and we get back the first case;
  - if $w(a) = 0$, then $a' = b$, and we get back the second case;
  - if neither $w(a')$ and $w(a)$ are 0, then $w(c) \geq 8$.

Binary Golay codes – Turyn's construction

Now we have

$$d_{\min} = \min_{c \neq 0} w(c) = 8$$

for the $C(24, 12)$ binary Golay code.

G_1 and G_2 are both self-dual, and so is G:

$$G \cdot G^T = \begin{bmatrix} G_1 & 0 & G_1 \\ 0 & G_1 & G_1 \\ G_2 & G_2 & G_2 \end{bmatrix} \cdot \begin{bmatrix} G_1^T & 0 & G_2^T \\ 0 & G_1^T & G_2^T \\ G_1^T & G_1^T & G_2^T \end{bmatrix} =$$

$$= \begin{bmatrix} 2G_1G_1^T & G_1G_1^T & 2G_1G_2^T \\ G_1G_1^T & 2G_1G_1^T & 2G_1G_2^T \\ 2G_2G_1^T & 2G_2G_1^T & 3G_2G_2^T \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Binary Golay codes – Turyn's construction

An equivalent generator of G, in systematic form (only the 1's are shown, other elements are 0):

$$G = \left[\begin{array}{cccccccccccc|cccccccccccc}
1 & & & & & & & & & & & & & & 1 & 1 & 1 & 1 & 1 & & & 1 \\
 & 1 & & & & & & & & & & & & 1 & & 1 & 1 & 1 & 1 & 1 & 1 & \\
 & & 1 & & & & & & & & & & & & 1 & & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & 1 & & & & & & & & & 1 & & & 1 & & 1 & 1 & 1 & 1 & 1 & \\
 & & & & 1 & & & & & & & & 1 & 1 & & & 1 & & 1 & 1 & 1 & & 1 \\
 & & & & & 1 & & & & & & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & \\
 & & & & & & 1 & & & & & & 1 & 1 & 1 & 1 & & & 1 & & & 1 & 1 \\
 & & & & & & & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & & 1 & & & \\
 & & & & & & & & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & & 1 & & 1 \\
 & & & & & & & & & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & & 1 & 1 \\
 & & & & & & & & & & 1 & & & & 1 & 1 & 1 & 1 & 1 & & & & \\
 & & & & & & & & & & & 1 & & 1 & & 1 & & 1 & & 1 & 1 & 1 & \\
 & & & & & & & & & & & & 1 & 1 & 1 & & 1 & & 1 & 1 & & & 1
\end{array}\right]$$

This is the $C(24, 12)$ extended binary Golay code.

Binary Golay codes – Turyn's construction

To obtain the $C(23, 12)$ perfect binary Golay code, we puncture the last bit from the $C(24, 12)$ code. Then $d_{\min}$ changes from 8 to 7, so the resulting code can still correct $\lfloor \frac{7-1}{2} \rfloor = 3$ errors.

What remains is to check the Hamming bound with $t = 3$:

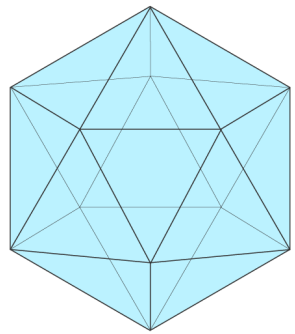
$$\binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 1 + 23 + 253 + 1771 = 2048 = 2^{23-12}$$

which holds, so the $C(23, 12)$ binary Golay code is perfect.

Binary Golay codes – icosahedron construction

Another possible construction for the $C(24, 12)$ binary Golay code is the following.

Let N be the adjacency matrix of the graph of an icosahedron (regular polyhedron with 12 vertices).



Theorem

Let J denote the 12×12 all-1 matrix. The code with 12×24 generator matrix

$$G = \left[I | J - N \right]$$

is equivalent to the $C(24, 12)$ Golay code.

(No proof.)

Cyclic Redundancy Check (CRC)

The Cyclic Redundancy Check (CRC) generalizes the addition of a parity bit. It is used almost exclusively for error detection (but not correction).

The CRC code adds r bits to the original message vector. For a fixed message length k and r, CRC is a systematic, linear $C(k + r, k)$ code.

However, it is often used for varying message lengths (with the same r), so instead of the generator matrix, we give a general coding scheme applicable for any message length.

CRC coding scheme

The CRC code that adds r bits to the message has a parameter: a nonzero binary vector $d \in \{0,1\}^r$.

Coding is equivalent to binary polynomial division (with remainder term). We make a binary polynomial out of both the message vector, then divide the message polynomial by the parameter polynomial, and the added r bits correspond to the remainder polynomial.

CRC coding scheme

Example. Let $u = (10010110)$ and $d = (011)$.

We first add a single 1 bit to the beginning of d, and add r 0 bits to the end of u.

The corresponding extended vectors are (10010110000) and (1011).

The corresponding binary polynomials are

$$u(x) = 1 \cdot x^{10} + 0 \cdot x^9 + 0 \cdot x^8 + 1 \cdot x^7 + 0 \cdot x^6 + 1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3,$$
$$d(x) = 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1.$$

The polynomials go from highest to lowest degree.

(For some other situations where we convert vectors to polynomials, the order of coefficients might be reversed; pay attention to each specific application!)

Polynomial division

We want to divide the polynomial $u(x)$ by $d(x)$. How to do that?

The general setup for polynomial division is

$$u(x) = q(x)d(x) + r(x),$$

where $r(x)$ is the remainder term with $\deg r(x) < \deg d(x)$.

$q(x)$ is computed step-by-step, first matching the highest degree term of $u(x)$, then proceeding to lower degree terms.

Polynomial division

Example. For the previous $u(x)$ and $d(x)$, we first have

$$x^{10} + x^7 + x^5 + x^4 = (x^3 + x + 1)(x^7 + \dots);$$

to match the x^{10} term on both sides, then polynomial division goes as follows:

$$\begin{aligned}
&x^{10} + x^7 + x^5 + x^4 - (x^3 + x + 1) \cdot x^7 = x^8 + x^5 + x^4, \\
&x^8 + x^5 + x^4 - (x^3 + x + 1) \cdot x^5 = x^6 + x^4 \\
&x^6 + x^4 - (x^3 + x + 1) \cdot x^3 = x^3 \\
&x^3 - (x^3 + x + 1) \cdot 1 = x + 1.
\end{aligned}$$

The end result is

$$x^{10} + x^7 + x^5 + x^4 + x^3 = (x^7 + x^5 + x^3 + 1)(x^3 + x + 1) + (x + 1)$$

Polynomial division

This means that for $u(x) = x^{10} + x^7 + x^5 + x^4 + x^3$ and $d(x) = x^3 + x + 1$,

$$u(x) = q(x)d(x) + r(x),$$

where

$$q(x) = x^7 + x^5 + x^3 + 1 \qquad r(x) = x + 1,$$

then $r(x)$ is converted back to a binary vector of length 3:

$$1 + x \quad \rightarrow (011),$$

and the codeword is

$$(10010110011).$$

Only $r(x)$ is relevant, $q(x)$ is not.

Fast polynomial division

Polynomial division can be done fast:

1	0	0	1	0	1	1	0	0	0	0
1	0	1	1							
0	0	1	0	0	1	1	0	0	0	0
	$\rightarrow$	1	0	1	1					
0	0	0	0	1	0	1	0	0	0	0
			$\rightarrow$	1	0	1	1			
0	0	0	0	0	0	0	1	0	0	0
						$\rightarrow$	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1

$$v = (u|d) = (10010110|011)$$

Error detection

Example. Assume the codeword $c = (10010110011)$ was sent through the channel, and the error vector is $e = (00010000000)$. Then the received vector is

$$v = c + e = (10000110011).$$

How do we check if there were any errors?

For the received vector v, we execute the exact same polynomial division, and check whether the remainder term is 0 or not.

Error detection

1	0	0	0	0	1	1	0	0	0	0
1	0	1	1							
0	0	1	1	0	1	1	0	0	0	0
	$\rightarrow$	1	0	1	1					
0	0	0	1	1	0	1	0	0	0	0
		$\rightarrow$	1	0	1	1				
0	0	0	0	1	1	0	0	0	0	0
			$\rightarrow$	1	0	1	1			
0	0	0	0	0	1	1	0	0	0	0
				$\rightarrow$	1	0	1	1		
0	0	0	0	0	0	1	1	1	0	0
					$\rightarrow$	1	0	1	1	
0	0	0	0	0	0	0	1	0	1	0
						$\rightarrow$	1	0	1	1
0	0	0	0	0	0	0	0	0	0	1

The remainder vector is not 0 $\rightarrow$ error detected!

Error detection

Theorem

The CRC code can always detect ≥ 1 error for any value of k.

Proof. Codewords correspond to polynomials which are multiples of $d(x)$.

If there was a codeword with weight 1, then it would correspond to a polynomial with a single term x^i.

But the divisor polynomial $d(x)$ always has more than 1 terms (x^r, plus whatever lower order terms correspond to the vector d), and x^i can never be a multiple of a polynomial with more than 1 terms. (By the way, this is the reason the vector d was assumed nonzero.)

Error detection

Depending on the value of k and the polynomial $d(x)$, codewords with weight 2 are possible. For example,

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

We do not strive to determine $d_{\min}$ for every possible choice of message size k and parameter vector d, but in general, the situation is the following: for a fixed vector d, as k is increased, $d_{\min}$ will decrease and eventually reach $d_{\min} = 2$.

While according to our usual definition of error detection, CRC codes can detect only 1 error, they have a different type of error detection property which is often useful.

Error detection

Theorem

The CRC code can detect multiple errors as long as all errors are within a block of r consecutive bits.

Proof. Any such error vector can be written as

$$x^i + \cdots + x^j = x^j(x^{i-j} + \dots),$$

where $i - j \leq r - 1$.

A polynomial of the form $x^d + \dots$ can divide neither the term x^j, nor the term $(x^{i-j} + \dots)$.

For the BSC channel model, errors could be anywhere (they are as likely to be close or far from each other), but for certain applications, 'batch errors' could happen, and CRC codes are good for detecting those.

Low-density parity-check (LDPC) codes

Next we will discuss LDPC codes, but only the very basics. Some further related ideas will be addressed briefly, but not in full detail.

For a general $(N-K) \times N$ binary matrix H, we expect that the number of 1's in H is about $\frac{1}{2}(N-K)N$. For example this is the case for the Hamming code.

LDPC codes (also known as Gallager codes) utilize very large but low-density parity-check matrices, where

- N and K typically range from 10^3 to 10^7, and
- the number of 1's in each row is small, typically below 10.

As a consequence, the total number of 1's in H is proportional to $(N-K)$.

LDPC codes

Example.

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

LDPC codes

As usual, the main challenge is decoding; for the received vector v, we want to find the closest codeword c.

Codewords satisfy $cH^T = 0$; a possible interpretation for this is that the rows of H correspond to linear constraints on the bits of the codeword.

The Tanner graph is defined as a bipartite graph; one class of vertices (called bits) correspond to the bits of the codeword, and the other class (called checks) corresponds to the rows of H.

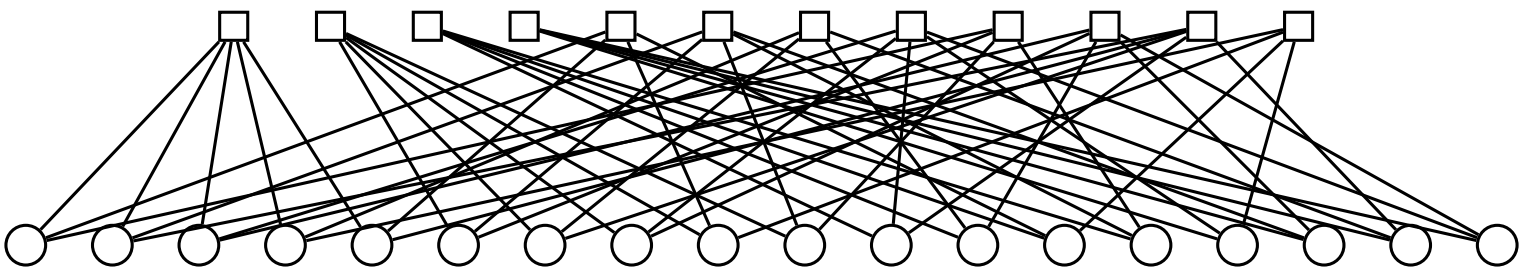
A bit is connected to a check if the corresponding element of H is 1.

LDPC codes

Example.

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

The Tanner graph has two classes of vertices: $N - K$ checks (boxes) and N bits (circles).



Bit-flipping algorithm

Decoding: for the received vector v, we want to find the closest codeword c.

This is done by the bit-flipping algorithm:

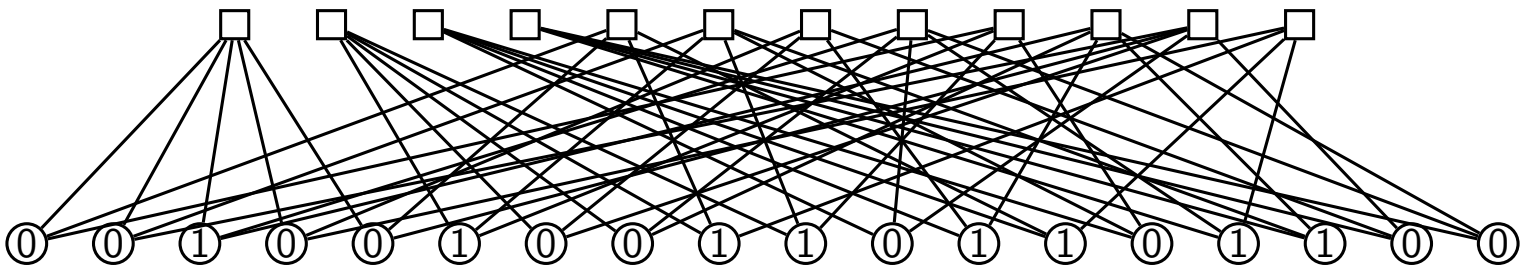
1. Initialization: put the bits of the received vector into the bit nodes.
2. For each check node, compute the sum of its neighboring bit nodes and put the result in the check node (green for 0, red for 1).
3. For each bit node, count how many neighboring checks are red.
4. For the bit with the most red check neighbors, flip the value of the bit.
5. Repeat from step 2 until all checks are green.
6. Read the detected codeword from the bit nodes.

Bit-flipping algorithm

Example. Assume

$$c = (0110010011011100100),$$
$$e = (0100000000000001000),$$
$$v = c \oplus e = (0010010011011101100).$$

We initialize the bit nodes with the bits of v.

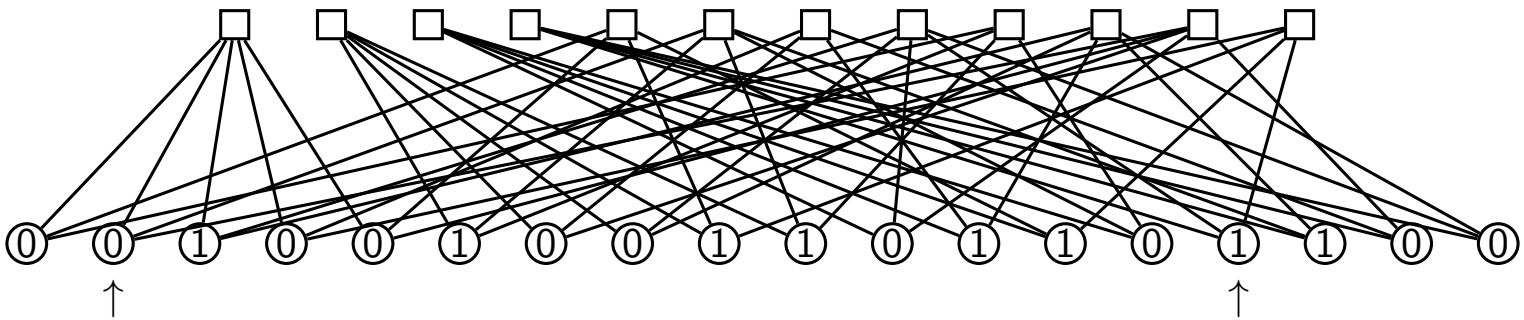


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Then

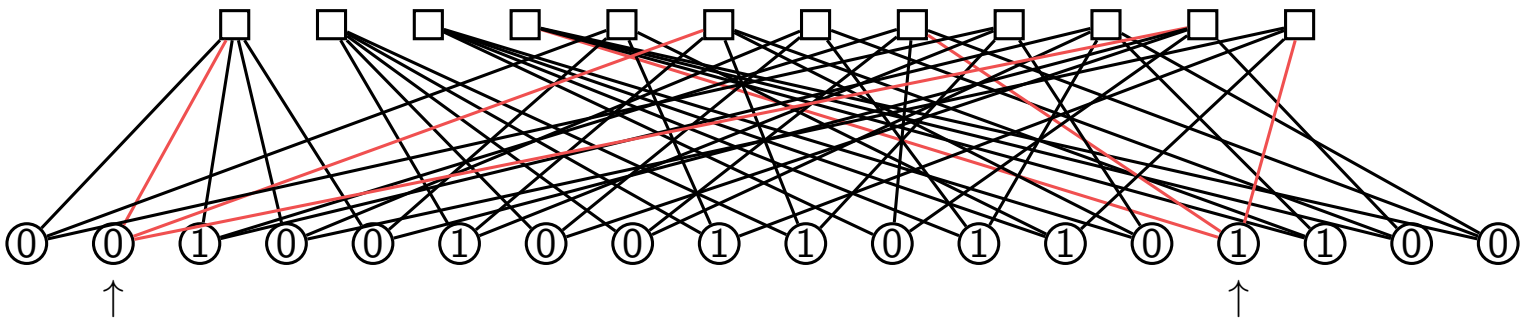


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Then

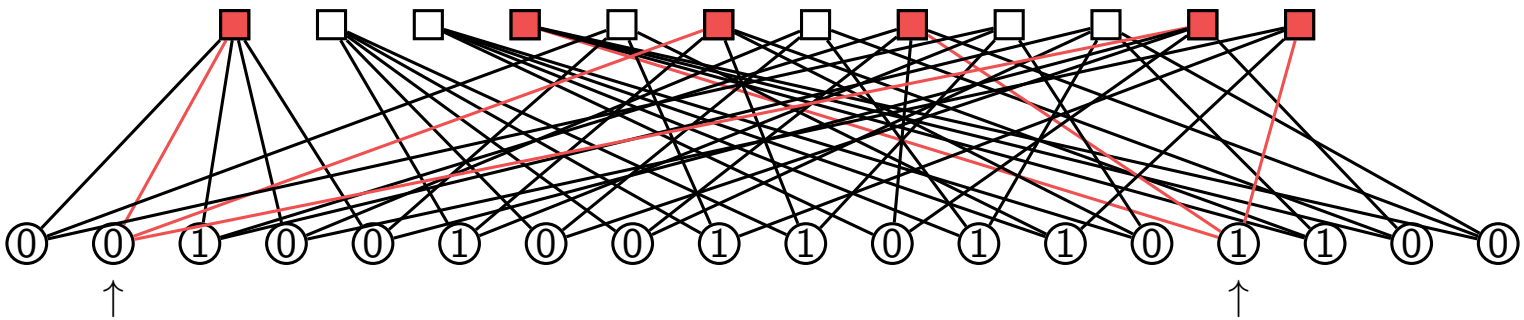


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011100100),$$
$$e = (0100000000000001000),$$
$$v = c \oplus e = (0010010011011101100).$$

The errors cause some checks to become red.

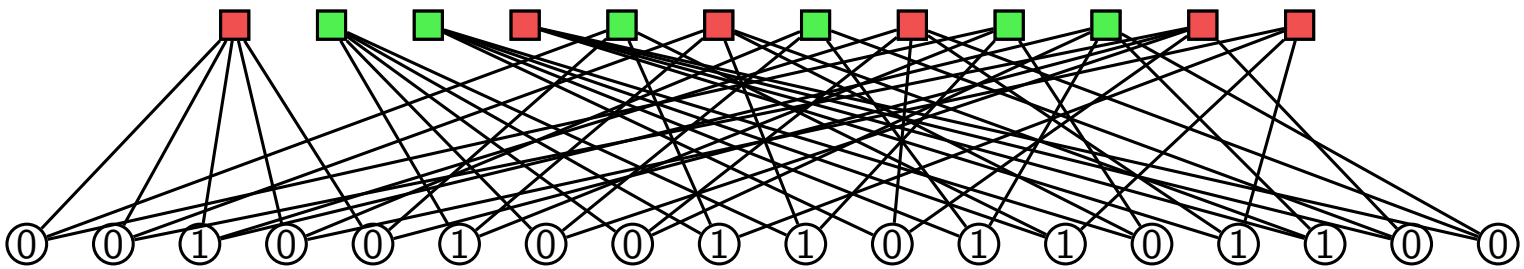


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011100100),$$
$$e = (0100000000000001000),$$
$$v = c \oplus e = (0010010011011101100).$$

The receiver only sees the red and green check nodes.

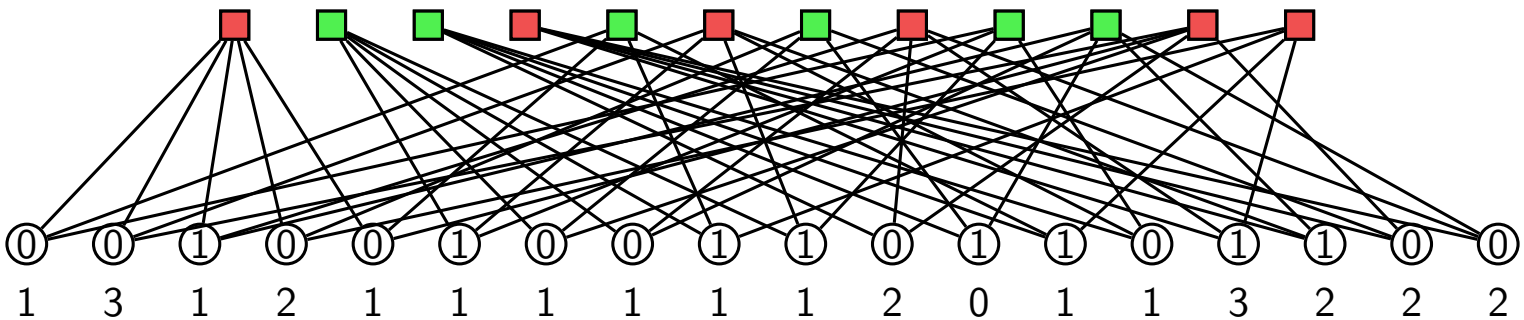


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011 00100),$$
$$e = (0100000000000 1000),$$
$$v = c \oplus e = (0010010011011 01100).$$

For each bit node, count its red neighbors.

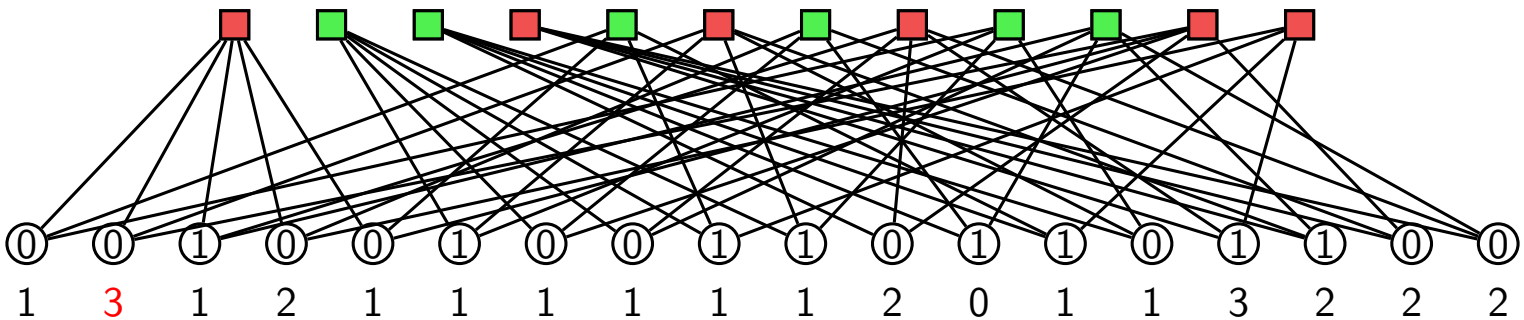


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011 00100),$$
$$e = (0100000000000 1000),$$
$$v = c \oplus e = (0010010011011 01100).$$

Select the maximum (either maximum in case of a tie).

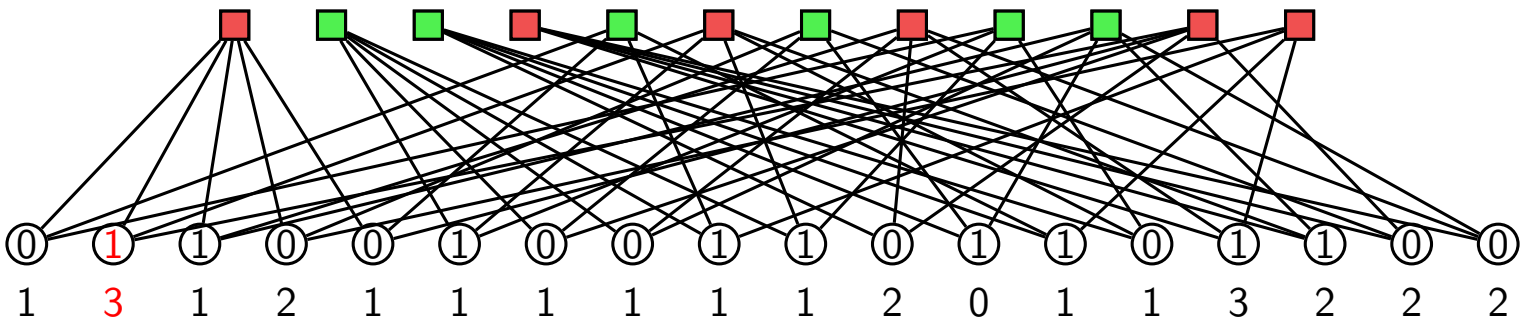


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Flip the corresponding bit.

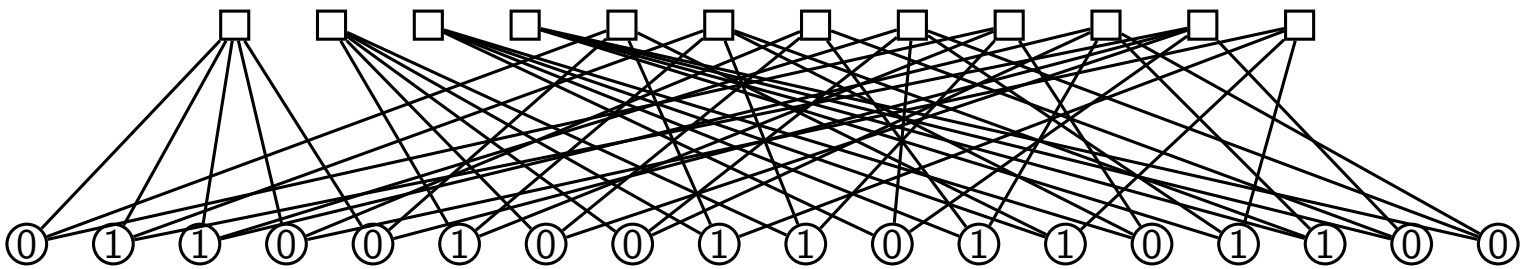


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

And repeat.

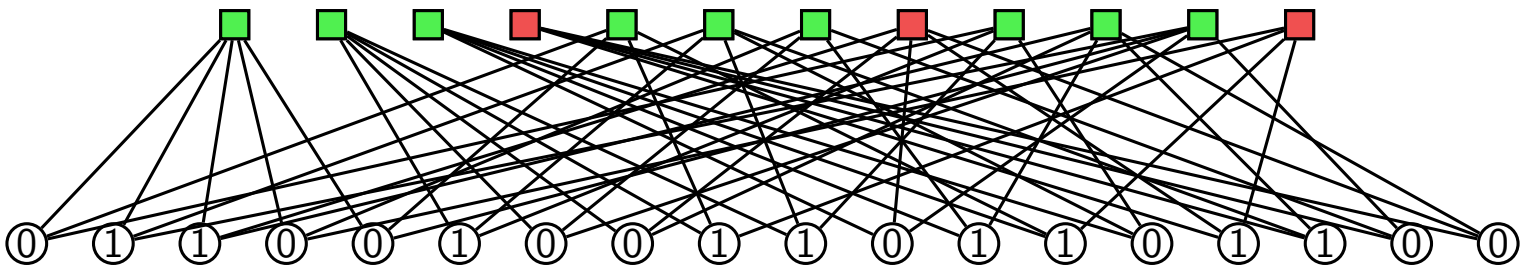


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Then

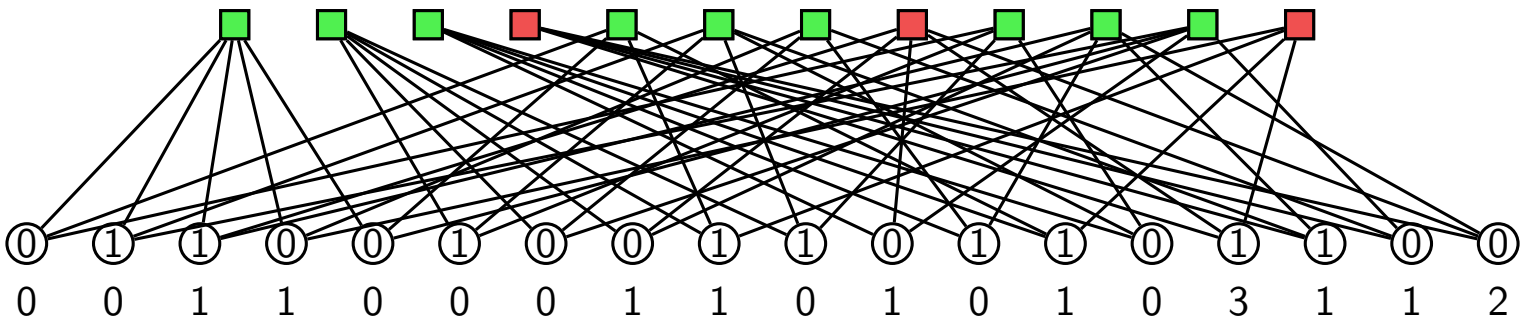


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Then

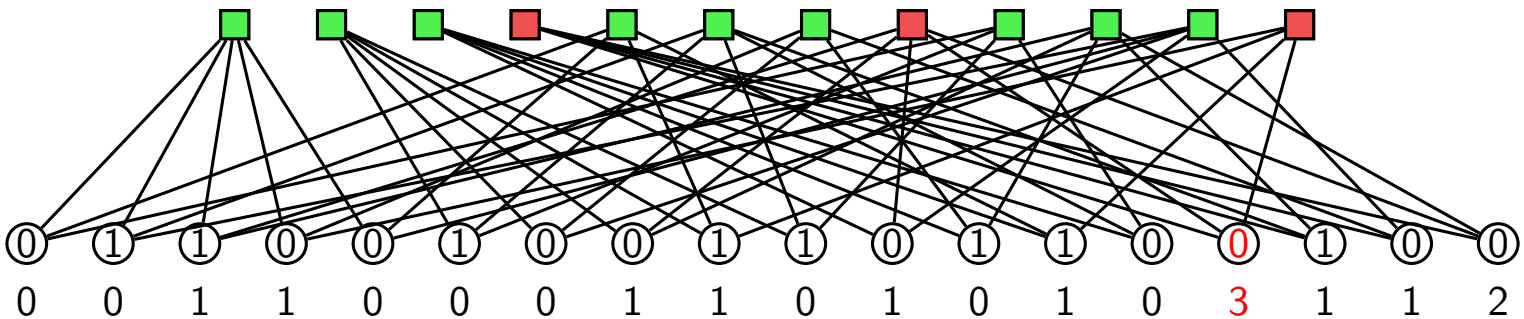


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Then

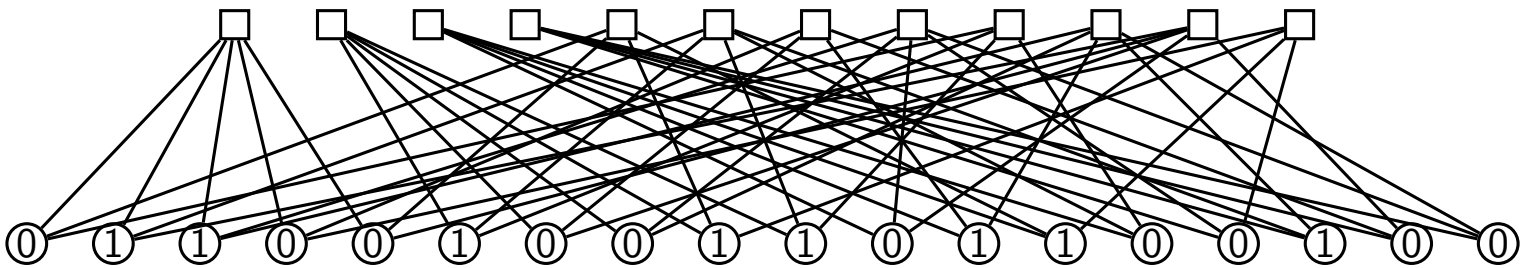


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Then

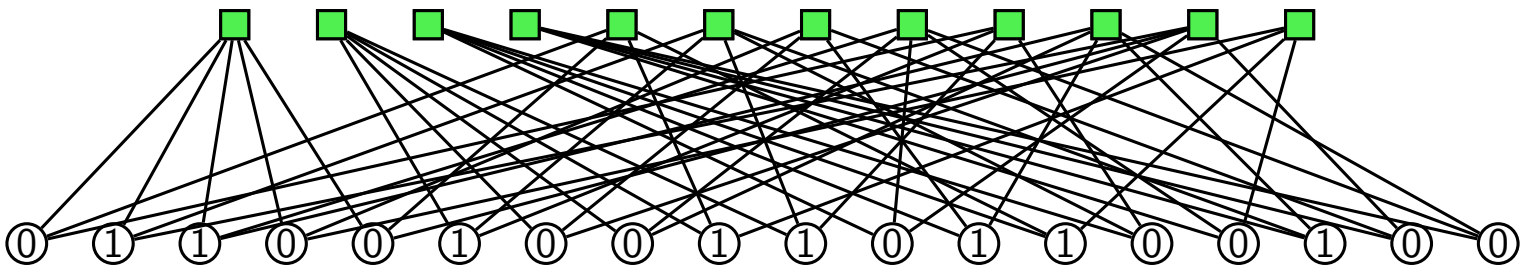


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011001 00),$$
$$e = (0100000000000010 00),$$
$$v = c \oplus e = (0010010011011011 00).$$

Then

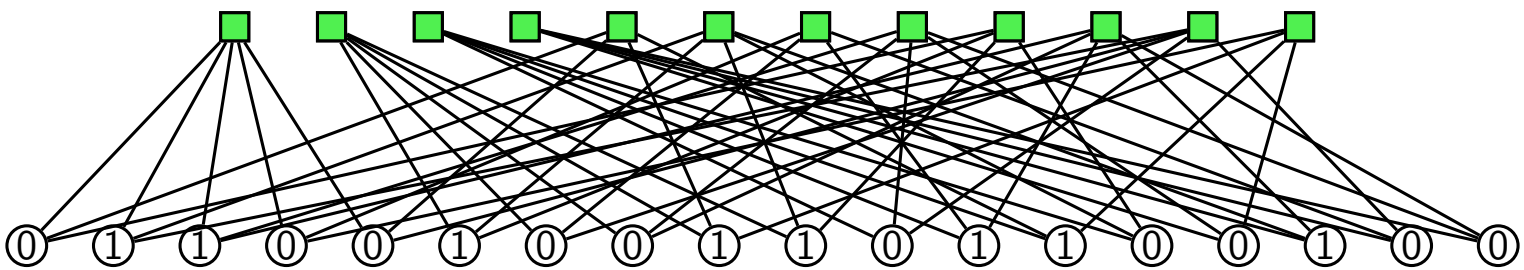


Bit-flipping algorithm

Example. Assume

$$c = (0110010011011100100),$$
$$e = (0100000000000001000),$$
$$v = c \oplus e = (0010010011011101100).$$

Then



Finished: $c' = (0110010011011100100)$.

Properties of LDPC codes

Properties of LDPC codes (intuitive explanations only, no theorems and proofs).

Why do LDPC codes work? The general idea is that few errors change only few checks (since the matrix is low-density), and the probability that the affected checks overlap is small, so errors can be easily identified.

LDPC codes work best when H is large, and its structure is somewhat 'random'. This brings a challenge: how to design a large but random-looking matrix? Usually it is done by a scalable block structure.

Apart from being 'random-looking', what kind of low-density matrices are good candidates for H with strong error correction capabilities?

Soft decoding

The bit-flipping algorithm is reasonably efficient, but actually there is an even better decoding algorithm; we discuss some of the ideas related to that next.

So far, we were doing hard decoding, where the received vector is decoded to a single codeword with minimal Hamming-distance. The motivation for minimal Hamming-distance is that that corresponds to the error vector with highest probability.

Soft decoding, on the other hand, calculates conditional probabilities for multiple possible codewords, or for each bit of the codeword separately.

Theoretically soft decoding could lead to wrong decoding, but in practice, the probability of this is negligible.

Soft decoding

For LDPC decoding, initially each check is treated as an independent parity check bit. Then for each bit node, the probability that it is a 1 is estimated in several rounds, improving the estimate with each iteration by factoring in the information from nearby check nodes and bit nodes.

Computations for the bit nodes can be run parallel within a round.

A main component of the algorithm is how the information from nearby nodes is calculated to update probabilities. The approach used is known as belief propagation or sum–product message passing.