2
IRA Codes

Irregular Repeat-Accumulate were introduced by Hui Jin, Khandekar & McEliece [JKM2000] in 2000, providing a set of codes that use linear-time encoding and iterative decoding while communicating reliably at rates close to channel capacity. The authors proved these codes can achieve channel capacity for the binary erasure channel and show remarkably good performance for the AWGN channel. IRA codes are a variation of the LDPC codes, as they work by adding parity bits to very large blocks, and are based on the same concepts.

LDPC codes are usually characterized by their parity check matrices — and their duals, the generator matrices — which can be described as regular or irregular. A regular matrix has constant row and column weight, i.e. regular matrices have an equal number of non-zero elements in every row and an equal number of non-zero elements in every column.

The parity-check matrices can be seen as the representation of a bipartite graph, which we call the Tanner graph (after R. Michael Tanner). A bipartite graph is composed of

- two disjoint sets of nodes;
- edges that join only nodes in different sets.

The two disjoint sets are labeled

variable-nodes whose values are given by the symbols that compose a codeword;

check-nodes whose values are given by the sum of the variable-nodes at the other ends of its incident edges;

In a Tanner Graph the edges are undirected, hence the convenience of describing it as a matrix. In a given code's parity-check matrix **H** each column identifies the edges leaving the column's corresponding variable-node, and the rows describe the connections from the perspective of their corresponding

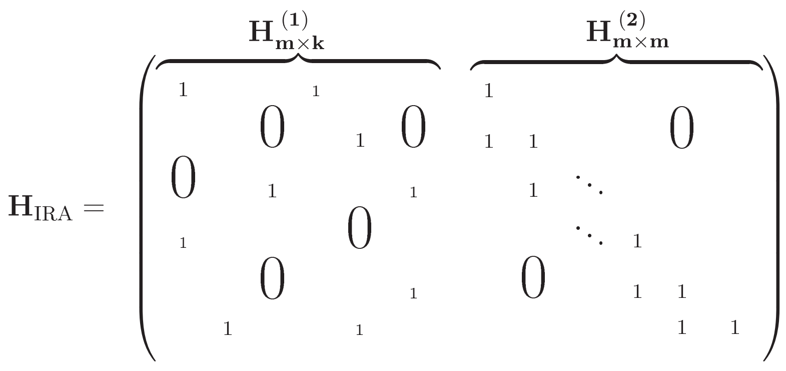

$\mathbf{H}_{\mathbf{m\times k}}^{(1)}$
$\mathbf{H}_{\mathbf{m\times m}}^{(2)}$
$\mathbf{H}_{\mathrm{IRA}} =$

Figure 2.1: An illustration of an IRA code's parity-check matrix, with a random region ($H^{(1)}_{m\times k}$) and a staircase-shaped region $H^{(1)}_{m\times m}$

check-nodes. Each edge in the graph is represented by making $H_{i,j} = 1$, where i and j are the indices to the two nodes being joined.[1]

The codes are said to be low-density because each check-node is joined with relatively few variable-nodes and vice-versa. Therefore, the code's description is a matrix containing relatively few non-zero elements. Such a matrix is termed a sparse matrix and can be represented in a very simple manner by keeping track only of the positions (and values, which are not implicit in the non-binary case) of the few non null elements. While classic LDPC parity-check matrices don't necessarily follow any visible pattern, the non-systematic part of IRA matrices (i.e. the columns that multiply the non-systematic, or parity, bits when obtaining a block's syndrome) is composed of *ones* in the main diagonal and the subdiagonal, with zero entries elsewhere, as illustrated by Figure 2.1. Due to this characteristic format, IRA codes are also called *Staircase Codes*.

In spite of the mathematical convenience of using matrices to represent sparse graphs, the concepts presented in this work require a graph oriented language. Therefore, instead of column and row weights we define a node's degree.

Definition 2 (node degree) *The degree of an information-node is this node's number of outgoing edges, i.e. the number of check-nodes joined with it. The reciprocal defines the degree of a check-node.* ◇

Although check-nodes are joined with information and parity nodes (see Figure 2.2), in IRA codes its degree refers only to the number of neighboring information-nodes (see more details in A).

[1]Throughout this document, we will say the variable-node and the check-node in opposite ends of an edge are *joined*, rather than *connected*. This distinction will become relevant when discussing graph construction.

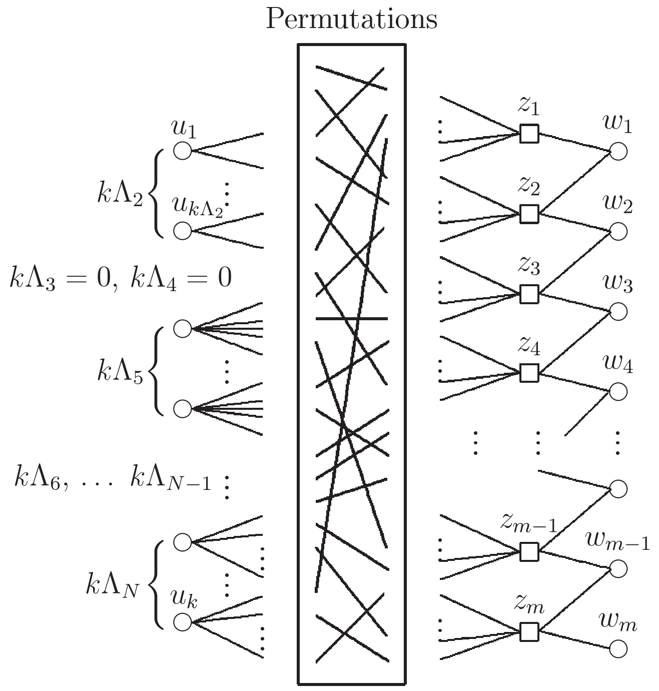


Figure 2.2: An example of a tanner graph where $\Lambda_3, \Lambda_4 = 0$. Circles (◯) represent variable-nodes, squares (□) are check-nodes and parity variable-nodes are to the left of the check-nodes.

IRA codes are best visually represented by a slightly modified bipartite Tanner Graph, as displayed in Figure 2.2. The information variable-nodes (representing vector $\mathbf{u}$ with systematic bits) are represented by the circles to the left, the check-nodes ($\mathbf{z}$) are displayed as squares and the parity variable-nodes (the vector of non-systematic bits $\mathbf{w}$) are represented by the circles to the right. The degree distribution $\boldsymbol{\Lambda}$ describes the fraction of information-nodes of degree $i \in \{1, 2 \ldots, N\}$, while all check-nodes have the same degree a. There is an equal number of parity variable-nodes and check-nodes. Each check-node z_j, $j = \{1, \ldots, m\}$ is joined with parity-nodes w_{j-1} and w_j and the parity-nodes are set to values such that that all variable-nodes joined with each check-node sum to 0 (mod 2), see (2-1) and (2-2).

2.1 Encoding

The constraint on the parity-nodes can be stated as

$$z_j = w_j + w_{j-1} + \sum_{l \in \mathcal{U}_j} u_l, \tag{2-1}$$

$$z_j = 0\ \forall j, \text{ where } j \in \mathbb{N} \text{ and } 1 \leq j \leq m. \tag{2-2}$$

The "+" sign denotes mod 2 sum, u_l is the value of the l^{th} information-node and $\mathcal{U}_j$ denotes all information-nodes joined with z_j.

For convenience we set $w_0 = 0$, and equation (2-1) with constraint (2-2) can be rewritten as

$$\begin{aligned} w_1 &= \sum_{l \in \mathcal{U}_1} u_l, \\ w_2 &= w_1 + \sum_{l \in \mathcal{U}_2} u_l, \\ &\vdots \\ w_j &= w_{j-1} + \sum_{l \in \mathcal{U}_j} u_l. \end{aligned} \tag{2-3}$$

The name Irregular Repeat-Accumulate comes from the encoding process using an irregular repetition of the information-nodes' values, which are accumulated in the order defined by the edges arriving at the check-nodes to obtain each value in $\mathbf{w}$.

The advantages of this recursive method over the typical encoding of LDPC codes are not obvious at this point. A brief discussion on the systematic encoding of classic LDPC codes will help to explain this advantage.

Let

$$\mathbf{H_{m\times n}} = [\mathbf{H^{(1)}_{m\times k}} \mid \mathbf{H^{(2)}_{m\times m}}], \tag{2-4}$$

be the parity-check matrix of a classic LDPC code. Classic LDPC parity-check matrices are randomly generated sparse matrices with no remarkable distinction to be made between its systematic and non-systematic sub-matrices $\mathbf{H^{(1)}}$ and $\mathbf{H^{(2)}}$. Let the matrix $\mathbf{A}$ be a linear transformation of $\mathbf{H}$ that contains an identity matrix in its non-systematic part as in (2-5).

$$\begin{aligned} \mathbf{A_{m\times n}} &= \left(\mathbf{H^{(2)}_{m\times m}}\right)^{-1} \mathbf{H} \\ &= [\mathbf{A^{(1)}_{m\times k}} \mid \mathbf{I_{m\times m}}]. \end{aligned} \tag{2-5}$$

From the matrix $\mathbf{A}$ it becomes easy to find the generator matrix $\mathbf{G}$, as shown in (2-6).

$$\mathbf{G}^{\mathrm{T}} = \begin{bmatrix} \mathbf{I_{k\times k}} \\ -- \\ \mathbf{A^{(1)}_{m\times k}} \end{bmatrix} \tag{2-6}$$

The encoding is then obtained by linear transformation of the message $\mathbf{u}$ through the generator matrix.

$$\begin{aligned} \mathbf{c_{1\times n}} &= \mathbf{u_{1\times k} G_{k\times n}} \\ &= [\mathbf{u_{1\times k}} \mid \mathbf{w_{1\times m}}] \end{aligned} \tag{2-7}$$

Since $\mathbf{A}$ is obtained from a linear transformation of the matrix $\mathbf{H}$, both matrices define the same subspace of $\mathrm{GF}\{2\}^n$. It can also be verified from (2-5) and (2-6) that $\mathbf{A}\mathbf{G}^{\mathrm{T}} = \mathbf{A}^{(1)} + \mathbf{A}^{(1)} = 0$, which means that $\mathbf{A}$ and $\mathbf{G}$ define orthogonal subspaces. In other words, $\mathbf{G}$ is the algebraic dual space of $\mathbf{A}$ [SBW2003, Ch. 2].

Since $\mathbf{A}$ and $\mathbf{H}$ define the same subspace, $\mathbf{H}$ also defines a dual space to $\mathbf{G}$ and is a valid parity-check matrix for the same code. While the matrix $\mathbf{A}$ defines the same code as $\mathbf{H}$, it does not describe a good graph for decoding under the message passing algorithm, which will be explained in section 2.2.3 and is needed only to obtain a generator matrix for systematic encoding.

The problem with using the generator matrix $\mathbf{G}$ for encoding, as in (2-7), is that the inverse of a sparse matrix — such as $\mathbf{H}$ — is often not sparse, i.e. the number of non-null elements in $\mathbf{A}^{(1)}$ grows more than linearly with the code's block-length, which increases the number of operations per parity bit. IRA encoding, in contrast, has a fixed number of $a + 1$ operations per parity bit (where a is the check-nodes' degree), as seen in equation (2-3).

2.2 Decoding

The decoding of a received vector $\mathbf{r}$ is a conceptually simple task, which consists of finding the most likely codeword ($\hat{\mathbf{c}}$) based on the Euclidean distance between the hypothetical transmitted vector and the received vector. In other words, that would be

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}} P\left(\mathbf{c} \mid \mathbf{y}\right), \quad \forall\, \mathbf{c} \in \mathcal{C}, \tag{2-8}$$

which is the definition of a maximum *a posteriori* (MAP) decoding.

True MAP decoding could be achieved by comparing the received vector (or block) $\mathbf{r}$ with all possible codewords, a computationally burdensome task, whose complexity grows exponentially with the block-length n. It would be more time efficient to make use of the code's tree-like structure and solve this problem using marginalization and factor graphs.

The function to be marginalized in this case would be the conditional probability

$$P\left(\mathbf{C} = \mathbf{c}^{(j)} \mid \{\mathbf{c}^{(j)} \in \mathcal{C}\},\, \mathbf{y}\right), \quad \mathbf{c}^{(j)} = \left(c_1^{(j)}, \ldots, c_n^{(j)}\right). \tag{2-9}$$

It should be noted that the domain of the received vector $\mathbf{y}$ depends on the channel under consideration (BEC, BSC, AWGN), and $|\mathcal{C}| = 2^k$.

To express the marginalized function, let us introduce the notation

$$g(x_i) = \sum_{\sim\{x_i\}} g(\mathbf{x}) \triangleq \sum_{x_1} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} g(x_1, x_2, \ldots, x_n).$$

We call $g(x_i)$ the summary of $g(\mathbf{x})$ with respect to x_i. Using this notation, a clean expression for the marginal probabilities of the received bits can be written as

$$P\left(C_i = x \mid \{\mathbf{C} \in \mathcal{C}\}, \mathbf{y}\right) = \sum_{\sim\{x_i\}} P\left(\mathbf{C} = \mathbf{x} \mid \{\mathbf{C} \in \mathcal{C}\}, \mathbf{y}\right). \tag{2-10}$$

These function marginalization procedures involve a number of operations that may grow exponentially with the number of variables.

Definition 3 ($\mathcal{O}(\cdot)$ notation) *An algorithm has complexity $\mathcal{O}(n)$ when its number of operations is upper bounded by $\kappa \cdot n$, where κ is a constant.* ◇

Thus, the brute force computation of the expression in (2-10) requires 2^k operations per bit, because there are 2^k valid codewords in $\mathcal{C}$. More generally, we can state $\mathcal{O}\left(|\mathcal{X}|^k\right)$, where $\mathcal{X}$ is the set in which the variable x is defined and $|\mathcal{X}|$ is the cardinality of this set.

We next present the theory behind Gallager's iterative decoding algorithm and the algorithm per se.

2.2.1 Factor Graphs

The practical impossibility of MAP decoding suggests the use of the code's structure to approximate a MAP estimate for the marginal probabilities of each bit in the codeword. This is called the "MPF" (Marginalize Product of Functions) problem. In the context of decoding LDPC codes, the solution to this problem is usually termed Belief Propagation, Message Passing Algorithm or Sum-Product Algorithm. The three names are often treated as synonyms, although we can observe they stress different properties. The Sum-Product Algorithm is the theoretical base for decoding using Bayes' rule and marginalization (in contrast with the Min-Sum algorithm), the Message Passing Algorithm is what is used in practice for iterative decoding; finally, Belief Propagation is used to evoke the use of soft decision, where the degree of certainty (a-posteriori probabilities) on a variable-node's value is more important than the presumed value per se.

Definition 4 (Factor Graph) *The* factor graph *is a bipartite graph expressing the structure of a function's factorization. There is one set of factor nodes* *and one set of variable-nodes. Each factor node represents one of the factor functions and is joined with one or more of its parameter variable.* ◇

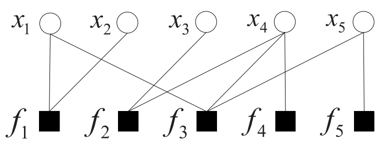
The Tanner Graph is therefore a factor graph where the factor nodes are the check-nodes. When we expand a graph from one variable in a tree-like manner we gain valuable insight on the expression to compute its marginals, as can be seen in Example 1 next.

Example 1 — Simple function factorization

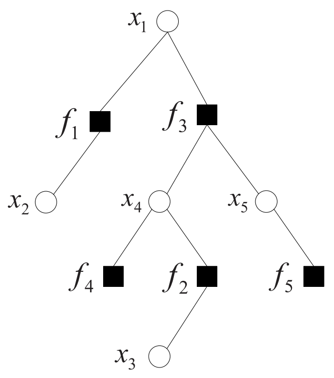
▶ Given a simple function $f(\mathbf{x}) = f(x_1, x_2, x_3, x_4, x_5)$ that can be factored as

$$f(\mathbf{x}) = f_1(x_1, x_2)f_2(x_3, x_4)f_3(x_1, x_4, x_5)f_4(x_4)f_5(x_5),$$

we can draw its factor graph and develop its expression tree to observe how the expression for the marginals can be factored. The connections are unchanged in both structures (see Figure 2.3), what differs is the hierarchical evidence of the variable chosen as the root and the logical division of tiers.



2.3(a): A factor graph in its usual bipartite form



2.3(b): The same graph's expression tree

Figure 2.3: bipartite factor graph and expression tree

Marginalizing $f(x_1)$ from a brute force approach would demand $\mathcal{O}(|\mathcal{X}|^4)$ operations. Luckily the expression tree suggests that we factor the expression splitting it into two sub-trees.

$$f(x_1) = \left(\sum_{x_2} f_1(x_1, x_2)\right)\left(\sum_{\sim\{x_1,x_2\}} f_2(x_3, x_4)f_3(x_1, x_4, x_5)f_4(x_4)f_5(x_5)\right) \quad (2\text{-}11)$$

This reduces the the computational effort to $\mathcal{O}(|\mathcal{X}|) + \mathcal{O}(|\mathcal{X}|^3) \approx \mathcal{O}(|\mathcal{X}|^3)$, a very modest gain. We can further improve our performance if we factor the expression in (2-11) once more, now choosing as root of the new sub-tree the variable-node with the highest degree, that is, x_4. This recursive procedure gives us the best possible factorization.

$$\begin{aligned} f(x_1) = &\left(\sum_{x_2} f_1(x_1,x_2)\right) \times \\ &\times \left(\sum_{x_4} f_4(x_4)\Big(\sum_{x_3} f_2(x_3,x_4)\Big)\Big(\sum_{x_5} f_3(x_1,x_4,x_5) f_5(x_5)\Big)\right), \end{aligned} \tag{2-12}$$

which demands $\mathcal{O}(|\mathcal{X}|) + \mathcal{O}(|\mathcal{X}|^2) + \mathcal{O}(|\mathcal{X}|^2) \approx \mathcal{O}(|\mathcal{X}|^2)$ operations. These approximations neglect the differences in the costs of the operations involved, but those costs are not expected to exceed the advantages of factorization.

◀

It should be noted that, theoretically, such factorization can only be accomplished if the factor graph is a bipartite tree or a set of trees. A factor graph is tree-like when there is only one path connecting two nodes, i.e. there are no cycles, which implies in many of the variable-nodes (the *leafs*) having degree one. This tree-like structure is not found in most known codes and, more surprisingly, actual tree-like codes are not desirable either. It can be proven (lemma 2.24 in [MCT2006]) that tree-like graphs have at least $\frac{2r-1}{2}n$ weight-two codewords, leading to bad distance properties.

2.2.2 The Sum-Product Algorithm (SPA)

In the case of the message passing algorithm, we want to factor the marginal probabilities of each received bit, respecting the constraints on the codeword. The constraints, represented in the graph by the check-nodes, apply to small groups of variable-nodes instead of the block as a whole. Therefore the check-nodes are factors of the membership function $\mathbb{1}(\cdot)$ defined next.

Definition 5 (Membership Function)

$$\mathbb{1}(expression) = \begin{cases} 1, & \textit{if expression is true,} \\ 0, & \textit{otherwise} \end{cases} \tag{2-13}$$

Since the membership function $\mathbb{1}(\mathbf{x} \in \mathcal{C})$ is true only when the codeword $\mathbf{x}$ has an all-zero syndrome, we can see it as a product of membership functions for each check-node. We denote $\mathcal{C}^{(i)}$ as the set of blocks that satisfy the constraints imposed by check-node z_i, such that $\mathcal{C} = \bigcap_{i=1}^{m} \mathcal{C}^{(i)}$ and

$$\mathbb{1}\left(\mathbf{x} \in \mathcal{C}\right) = \prod_{i=1}^{m} \mathbb{1}\left(\mathbf{x} \in \mathcal{C}^{(i)}\right). \tag{2-14}$$

The factorization of the graph requires a notation for the subsets of nodes specified by the neighborhoods of the nodes in the graph as explained below.

Definition 6 (Node Neighborhood) *The notation $\mathcal{N}_m$ is used to denote the set of variable-nodes in the* neighborhood *of z_m, i.e. the set of all variable-nodes joined to the check-node z_m. The neighborhood of a variable-node c_n is denoted by $\mathcal{M}_n$. Also,*

$$\mathcal{M}_{n,m} = \mathcal{M}_n \setminus z_m$$
$$\mathcal{N}_{m,n} = \mathcal{N}_m \setminus c_n$$

◇

Bayes' theorem allows us to write an expression for the a posteriori probability of a codeword,

$$\begin{aligned} P\left(\mathbf{C} = \mathbf{x} \mid \{\mathbf{C} \in \mathcal{C}\}, \mathbf{y}\right) &= \frac{P\left(\{\mathbf{C} \in \mathcal{C}\} \mid \mathbf{C} = \mathbf{x}, \mathbf{y}\right) \times P\left(\mathbf{C} = \mathbf{x} \mid \mathbf{y}\right)}{P\left(\mathbf{C} \in \mathcal{C} \mid \mathbf{y}\right)} \\ &= \frac{\mathbb{1}\left(\mathbf{x} \in \mathcal{C}\right) P\left(\mathbf{C} = \mathbf{x} \mid \mathbf{y}\right)}{P\left(\mathbf{C} \in \mathcal{C} \mid \mathbf{y}\right)}, \end{aligned} \tag{2-15}$$

where $P\left(\mathbf{C} \in \mathcal{C} \mid \mathbf{y}\right) = \sum_{\mathbf{x}' \in \mathcal{X}^n} \mathbb{1}\left(\mathbf{x}' \in \mathcal{C}\right) P\left(\mathbf{C} = \mathbf{x}' \mid \mathbf{r}\right)$ is a function of $\mathbf{r}$ whose value does not depend on $\mathbf{x}$. If we focus on a single variable-node, the expression becomes

$$\begin{aligned} &P\left(C_j = x \mid \left\{\mathbf{C} \in \bigcap\nolimits_{i \in \mathcal{M}_j} \mathcal{C}^{(i)}\right\}, \mathbf{y}\right) \\ &= \frac{P\left(\left\{\mathbf{C} \in \bigcap\nolimits_{i \in \mathcal{M}_j} \mathcal{C}^{(i)}\right\} \mid C_j = x, \mathbf{y}\right) \times P\left(C_j = x \mid \mathbf{y}\right)}{P\left(\mathbf{C} \in \bigcap\nolimits_{i \in \mathcal{M}_j} \mathcal{C}^{(i)} \mid \mathbf{y}\right)} \end{aligned} \tag{2-16}$$

where $P\left(\left\{\mathbf{C} \in \bigcap\nolimits_{i \in \mathcal{M}_j} \mathcal{C}^{(i)}\right\} \mid C_j = x, \mathbf{y}\right)$ can be factored, and $P\left(\mathbf{C} \in \bigcap\nolimits_{i \in \mathcal{M}_j} \mathcal{C}^{(i)} \mid \mathbf{y}\right)$ is a normalization constant. The complexity involved in computing (2-16) can be reduced if we define a vector $\mathbf{x}_{(\mathcal{N}_{i,j})}$ containing only the elements $\{x_{i'} \mid \forall\, i \in \mathcal{N}_{i,j}\}$, and $\sum_{\{\mathbf{x}_{(\mathcal{N}_{i,j})}\}}[\ldots]$ a summation over all instances of such vector.

For simplicity, we introduce the intermediate variable $\gamma_{i,j}(x)$, the probability of check z_i being satisfied when $c_j = x$, its value is given by

$$\begin{aligned}\gamma_{i,j}(x) &= P\left(\left\{\mathbf{C} \in \mathcal{C}^{(i)}\right\} \mid C_j = x, \mathbf{y}\right) \\ &= \sum_{\{\mathbf{x}_{(\mathcal{N}_{i,j})}\}} \mathbb{1}\left(\textstyle\sum_{j' \in \mathcal{N}_{i,j}} x_{j'} = x\right) \prod_{j' \in \mathcal{N}_{i,j}} P\left(C_{j'} = x_{j'} \mid y_{j'}\right),\end{aligned} \tag{2-17}$$

where the summation takes place over all possible combinations of bits in the positions of $\mathbf{c}$ defined by the check z_i. We reduced the number of operations in (2-17) by only including those variable-nodes in the neighborhood of the check-nodes that are adjacent to c_j.

The computation of $\gamma_{i,j}(x)$ is usually referred as the *horizontal step*. And, assuming independence of the parity-checks, we replace the first conditional probability in the numerator of (2-16) by $\prod_{i \in \mathcal{M}_j} \gamma_{i,j}(x)$. To compute equation (2-16), we now use the expression

$$P\left(C_j = x \mid \left\{\mathbf{C} \in \textstyle\bigcap_{i \in \mathcal{M}_j} \mathcal{C}^{(i)}\right\}, \mathbf{y}\right) = \alpha_j P\left(C_j = x \mid \mathbf{y}\right) \prod_{i \in \mathcal{M}_j} \gamma_{i,j}(x), \tag{2-18}$$

where

$$\alpha_j = \left(P\left(C_j = 0 \mid \mathbf{y}\right) \prod_{i \in \mathcal{M}_j} \gamma_{i,j}(0) + P\left(C_j = 1 \mid \mathbf{y}\right) \prod_{i \in \mathcal{M}_j} \gamma_{i,j}(1)\right)^{-1} \tag{2-19}$$

replaces the denominator in (2-16). This part of the algorithm is called the *vertical step*.

The process described above applies to all the variable-nodes in the graph. This gives not the MAP estimation for the transmitted codeword, but the MAP estimation of the probability of each bit given the channel *a posteriori* probabilities of each bit in the received vector and the graph properties. The output probabilities can be used as the input for another computation until a hard decision on the estimated probability vector gives a valid codeword.

2.2.3 Message-Passing

The Sum-Product Algorithm, as detailed in Section 2.2.2, works by using the whole received vector for the estimation of the emitted codeword. Still, computing (2-17) and (2-18) respectively, for every variable-node is unpractical. The Message-Passing Algorithm allows a faster computation of the SPA by creating intermediate variables that act as the *messages*. At each iteration while doing the computations described in the horizontal step, all messages flow only once from the variable-nodes to the check-nodes, and then back to the variable-nodes at the vertical step.

Let $\mu_j(x) = P\left(C_j = x \mid \mathbf{y}\right)$ be the a posteriori bit probability of variable-node c_j. We define the *messages* in terms of the log-likelihood ratios $\ell(C_j)$ of the two possible outcomes of each bit from the received vectors as given by (2-20). This practice takes advantage of the use of a binary alphabet to reduce the number of computations per iteration of the algorithm.

$$\ell(C_j) = \ln\frac{\mu_j(0)}{\mu_j(1)}. \tag{2-20}$$

From (2-20), we have

$$\mu_j(0) = \frac{e^{\ell(C_j)}}{e^{\ell(C_j)}+1} \tag{2-21}$$

$$\mu_j(1) = \frac{1}{e^{\ell(C_j)}+1}. \tag{2-22}$$

Using the values obtained from (2-20) as messages we can find an expression for the log-likelihood ratio of a check-node as a function of the log-likelihood ratios of its adjacent variable-nodes. In other words,

$$f : \{\ell(C_j),\, j \in \mathcal{N}_{i,n'}\} \mapsto \ell\left(\sum_{j\in\mathcal{N}_{i,n'}} C_j\right). \tag{2-23}$$

We start with the simpler case of finding

$$\ell(C_1 \oplus C_2) = \ln\left(\frac{\mu_1(1)\mu_2(1)+\mu_1(0)\mu_2(0)}{\mu_1(0)\mu_2(1)+\mu_1(1)\mu_2(0)}\right). \tag{2-24}$$

Taking (2-21) and (2-22) into (2-24) yields

$$\ell(C_1 \oplus C_2) = \ln\left(\frac{1+e^{\ell(C_1)+\ell(C_2)}}{e^{\ell(C_1)}+e^{\ell(C_2)}}\right) \tag{2-25}$$

$$= \ln\left(\frac{(1+e^{\ell(C_1)})(1+e^{\ell(C_2)})+(1-e^{\ell(C_1)})(1-e^{\ell(C_2)})}{(1+e^{\ell(C_1)})(1+e^{\ell(C_2)})-(1-e^{\ell(C_1)})(1-e^{\ell(C_2)})}\right) \tag{2-26}$$

Then, considering the hyperbolic tangent function,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2-27}$$

and replacing the following patterns found in (2-26)

$$\ln\left(\frac{1+x}{1-x}\right) = 2\tanh^{-1} x \quad ; \quad \tanh\frac{x}{2} = \frac{e^x-1}{e^x+1}\ , \tag{2-28}$$

we arrive at the fundamental equation of log-likelihood algebra

$$\ell(C_1 \oplus C_2) = 2\tanh^{-1}\left(\tanh\frac{\ell(C_1)}{2}\tanh\frac{\ell(C_2)}{2}\right), \tag{2-29}$$

which can be used recursively to address the log-likelihood ratio involving a mod 2 sum of multiple variables in (2-23), that is

$$\ell\left(\sum_{j\in\mathcal{N}_{i,n'}} C_j\right) = 2\tanh^{-1}\left(\prod_{j\in\mathcal{N}_{i,n'}} \tanh\left(\frac{\ell(C_j)}{2}\right)\right) \tag{2-30}$$

From (2-20) and (2-28), we can verify that $\tanh\left(\frac{\ell(C_j)}{2}\right) = \mu_j(0) - \mu_j(1)$, so we will conveniently denote this value $\delta\mu_j$. At each decoder iteration, the messages are sent from the variable-nodes (c_j, $j \in \{1,\ldots,n\}$) to their adjacent check-nodes (z_i, $i \in \{1,\ldots,m\}$), making

$$\begin{aligned}\delta\gamma_i &\triangleq \gamma_i(0) - \gamma_i(1)\\ &= \tanh\left(\frac{\ell\left(z_i\right)}{2}\right) = \prod_{j\in\mathcal{N}_i}\tanh\left(\frac{\ell(C_j)}{2}\right)\\ &= \prod_{j\in\mathcal{N}_i}\delta\mu_j.\end{aligned} \tag{2-31}$$

This is equivalent to executing the *horizontal step* in equation (2-17) for all variable-nodes at once, since $\delta\gamma_{i,j} = \delta\gamma_i/\delta\mu_j$. The new messages are than obtained as in (2-17), and the process repeats itself with each variable-node sending messages with its *extrinsic information* along the graph for new computations. A compact description of the message passing algorithm can be found in Appendix B.