

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Elettronica

Communication techniques based on SC-LDPC codes with sliding window decoding and noiseless feedback channel

Tecniche di comunicazione basate su codici SC-LDPC con decodifica a finestra scorrevole e canale di feedback privo di rumore

Advisor:
Dr. Massimo Battaglioni

Candidate:
Mauro Michele Maria Costantino

Academic Year 2025/2026

Abstract

SC-LDPC codes are commonly employed with SWD, which allow handling the complexity of the decoding process that, in its full version, is often unfeasible due to the long frame lengths. This approach, however, remains susceptible to error propagation, which can severely degrade the overall performance. Therefore, in the present work we propose techniques for both reliable decoding error detection and effective mitigation of the error propagation under the assumption that a noiseless feedback channel is available. We first present two decoder error detection methods that, rather than relying on empirically tuned criteria based on LLRs, are based on the number of decoding iterations and the concatenation with an outer CRC code, respectively. Once errors are detected, it is then possible to either send a resynchronization request or initiate selective retransmissions, as a form of type-I HARQ protocol. We further study the effects of the type-I HARQ-based protocol, by developing a finite-state analytical model that describes the interaction between retransmissions and the evolution of the decoding wave, thus enabling a prediction of the block error rate across window positions. Even though these strategies introduce transmission overhead and cause a slight rate loss, Monte Carlo simulations over both AWGN and Rayleigh fading channels demonstrate that they significantly improve error-rate performance compared to standard sliding window decoding.

Sommario

I codici SC-LDPC vengono comunemente decodificati tramite SWD, grazie al quale è possibile gestire la complessità del processo di decodifica che, a rigore, risulta essere spesso computazionalmente intrattabile a causa delle elevate lunghezze di blocco. Questo approccio, tuttavia, è vulnerabile al fenomeno della propagazione degli errori, che può compromettere in modo significativo le prestazioni complessive del sistema. Pertanto, nel presente lavoro vengono proposte tecniche che consentano sia di individuare errori di decodifica e sia di mitigare in maniera efficace la propagazione degli errori, assumendo che si abbia accesso ad un canale di feedback privo di rumore. In prima istanza, vengono presentati due metodi di identificazione degli errori di decodifica che, invece di vertere su parametri basati su LLRs e ottimizzati empiricamente, si fondano, rispettivamente, sul numero di iterazioni del processo di decodifica e sulla concatenazione con un codice CRC esterno. Riguardo le tecniche di mitigazione, invece, si sono studiati gli effetti di una risincronizzazione tra trasmettitore e ricevitore, e di un protoccolo basato su ritrasmissioni selettive come forma di HARQ di tipo I. Gli effetti di tale protocollo vengono inoltre ulteriormente approfonditi, sviluppando un modello analitico che descrive l'interazione tra le ritrasmissioni e l'evoluzione dell'onda di decodifica, permettendo inoltre predizioni accurate del tasso di errore sul blocco al variare della posizione della finestra di decodifica. Tramite simulazioni Monte Carlo, inoltre, viene validata l'efficacia delle tecniche proposte. L'analisi delle tecniche di rilevazione mostra che il criterio basato sulle iterazioni è più efficace di quanto già presente in letteratura, mentre il codice CRC esterno si comporta quasi da rilevatore ideale. Le tecniche di mitigazione proposte, invece, introducono significativi miglioramenti al tasso di errore rispetto allo SWD standard e ad altre tecniche di mitigazione prese come riferimento. Inoltre, la robustezza delle tecniche presentate in questo lavoro viene ulteriormente dimostrata da confronti eseguiti sul canale di Rayleigh che, soprattutto in contesti di slow fading, rappresenta uno scenario particolarmente sfavorevole per le comunicazioni.

Contents

Introduction 1

1 Spatially-Coupled LDPC codes and decoding algorithms 3
- 1.1 Linear block codes . . . 3
  - 1.1.1 LDPC codes . . . 6
- 1.2 Convolutional codes . . . 9
  - 1.2.1 SC-LDPC codes . . . 12

2 Iterative decoding algorithms 17
- 2.1 Sum-Product Algorithm . . . 18
- 2.2 Sliding window decoding . . . 23
  - 2.2.1 Complexity and latency analysis . . . 25

3 Error propagation detection and mitigation 29
- 3.1 Detection techniques . . . 29
  - 3.1.1 Iteration-based threshold . . . 30
  - 3.1.2 CRC concatenation . . . 30
- 3.2 Mitigation strategies . . . 32
  - 3.2.1 Encoder resynchronization . . . 32
  - 3.2.2 Type-I HARQ . . . 34
- 3.3 Coding rate and throughput analysis . . . 47

4 Numerical results 51
- 4.1 Model validation . . . 52
- 4.2 Error detection effectiveness . . . 54
- 4.3 Error rate performance . . . 57

5 Conclusions 67

Introduction

Low-Density Parity-Check (LDPC) codes have become the standard in the telecommunications industry thanks to their excellent error-correction capabilities combined with limited decoding complexity. These codes are currently employed in Ethernet 10GBASE-T, form an integral part of the standard Wi-Fi IEEE 802.11, play a fundamental role in the new standard Wi-Fi 7 IEEE 802.11be [1] and are also used for the 5G data channel [2]. Moreover, as reported in [3] and [4], they are also widely employed in the field of satellite communications.

Spatially-Coupled LDPC (SC-LDPC) codes, introduced for the first time as convolutional LDPC codes in [5], further extend the capabilities of LDPC block codes, leading to exceptional asymptotic performance. In fact, through a process known as spatial coupling, these codes exhibit the so-called threshold saturation phenomenon, whereby the Belief Propagation (BP) decoding threshold approaches the Maximum A Posteriori (MAP) threshold of the underlying LDPC block code ensemble [6], [7]. Therefore, SC-LDPC codes can operate remarkably close to Shannon capacity while maintaining a relatively simple decoding structure, thus representing a very promising candidate for the upcoming new standards.

However, despite these benefits, full parallel decoding of SC-LDPC over the entire frame length is computationally expensive and impractical for most real-time applications. In order to address this limitation, Sliding Window Decoder (SWD) was proposed in [8] as an efficient strategy that processes only a portion of the code's Tanner graph [9] at each decoding step. Sliding window decoders can achieve significantly reduced latency, lower memory usage, and scalable hardware implementation. However, as will be clarified later, they are susceptible to decoder error propagation because, due to the structure of the decoder, the decoding of a given window also depends on the decoding of a certain number of previous windows, i.e., if decoding errors occur in a target block they can propagate to future blocks, leading to an undesired degradation in performance. Therefore, enhancing the resilience of the SWD against error propagation is still a critical research challenge in order to employ SC-LDPC codes in continuous transmission scenarios.

This motivates the present work, in which we focus on both reliable decoding error detection and effective mitigation of decoder error propagation. In particular, to address the problem of error detection, we propose a lightweight and heuristic method based on both the number of decoding iterations and a partial syndrome check (extending the idea introduced by us in [10]), and an alternative error detection method based on the concatenation of the SC-LDPC code with an outer Cyclic

Redundancy Check (CRC) code. Regarding the latter, we also show that several concatenation strategies are possible and analyze their performance.

Then, as for error mitigation, we explore two complementary strategies to limit error propagation:

- *Adaptive resynchronization*: When a mitigation request is sent, the decoder issues a resynchronization request to the transmitter over a noiseless binary feedback channel. Then, the transmitter rolls back its buffer, fixes some symbols to a predetermined pattern, and retransmits all blocks following the erroneous one, allowing transmission to resume as if a new frame had been initiated, thus preventing error propagation in subsequent decoding windows;
- *HARQ-based retransmission*: We also consider a type-I Hybrid Automatic Repeat reQuest (HARQ) strategy, where a mitigation request triggers a retransmission of all affected blocks. The combination of HARQ and SWD requires careful handling of soft information and proper management of the decoder state.

These proposed mitigation strategies are adaptive and therefore rely on error detection techniques. As will be shown later, while they assume the availability of a binary noiseless feedback channel and some buffering at the transmitter, they significantly reduce the decoding error rate, at the cost of some rate and/or throughput reduction. It is important to emphasize that, although the underlying techniques are known for block codes, their incorporation into SWD of SC-LDPC codes is far from straightforward, since the convolutional memory and inter-block coupling alter their behavior and make the combined use of a CRC and resynchronization/retransmission significantly more challenging compared with block-code scenario.

The document is organized as follows. Chapter 1 introduces the concepts behind both block and convolutional error-correcting codes, focusing on LDPC and SC-LDPC codes. Chapter 2 introduces the main decoding algorithms for LDPC codes and, focusing on the Sum-Product Algorithm (SPA), the structure of SWD is then analyzed and the impact on both the complexity and the latency of the overall decoding process is studied. Chapter 3 describes in detail the proposed error detection and error mitigation strategies. In particular, a stochastic model describing the evolution of the SWD reliability across different positions for CRC-based type-I HARQ strategy is also presented. Then, the same Chapter describes the impact of the proposed strategies on both the coding rate and the throughput. In Chapter 4 we provide a numerical analysis of the detection and mitigation performance. Moreover, we study the impact of the proposed strategies in different communication scenarios, considering an Additive White Gaussian Noise (AWGN) channel and a Rayleigh channel. Finally, Chapter 5 concludes the present work.

Chapter 1

Spatially-Coupled LDPC codes and decoding algorithms

This chapter introduces the basic concepts of both block and convolutional codes and, more specifically, describes the structure of LDPC and SC-LDPC codes.

1.1 Linear block codes

Let us define an *information sequence* as a continuous sequence of binary symbols over GF(2) generated by an information source. In *block coding*, such an information sequence is segmented into messages of fixed length k. So, for a fixed k, there are 2^k distinct messages $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$. Then, at channel encoder, every sequence is mapped into a longer binary sequence $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, called *codeword*, according to a certain encoding rule. In particular, a *block code* (n, k) is actually formed by the set of 2^k distinct codewords. The channel encoder, therefore, maps each k-bit message into a longer $n-k$ codeword, thus introducing $n-k$ *redundant bits*. Although these redundant bits carry no new information, they allow the receiver to *detect* and *correct* errors caused by the transmission on a noisy channel.

Another meaningful parameter used to describe a block code, and a code in general, is the *code rate*, that can be interpreted as the average number of information bits carried by every symbol of the encoded sequence. So, for a block code, the rate is defined as

$$R = \frac{k}{n}. \tag{1.1}$$

Block codes can be divided into two main categories, *linear* and *non-linear* codes, even though the latter are not widely employed in modern standards. Therefore, the following definition of linear block codes is reported from [11]:

Definition 1. *A binary block code of length n with 2^k codewords is called an (n, k) linear block code if and only if its 2^k codewords form a k-dimensional subspace of the vector space V of all the n-tuples over GF(2).*

Hence, since a (n, k) linear block code (denoted, in the following, as $\mathcal{C}$) defines a k-dimensional vectorial subspace of all the n-tuples over GF(2), there exist k *linearly independent* codewords $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$ such that every $\mathbf{c} \in \mathcal{C}$ can be expressed as a

linear combination of these k linearly independent codewords that, therefore, form a *basis* $\mathcal{B}_C$ of $\mathcal{C}$. In other words, given an information sequence $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$, it is possible to obtain the corresponding codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ using the aforementioned basis with the k message bits of $\mathbf{u}$ as the coefficients of the following linear combination:

$$\mathbf{c} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \cdots + u_{k-1}\mathbf{g}_{k-1}. \tag{1.2}$$

Arranging the basis of $\mathcal{C}$ in a matrix, we obtain

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \cdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{1.3}$$

then, the equation (1.2) can be rewritten as

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{G}. \tag{1.4}$$

Therefore, $\mathbf{G}_{[k,n]}$ is called a *generator matrix* of the (n, k) linear block code $\mathcal{C}$. In general, $\mathcal{C}$ has more than one basis, so the generator matrix of a given (n, k) linear block code is not unique and any choice of a basis of $\mathcal{C}$ gives a valid generator matrix.

From definition 1, $\mathcal{C}$ is a vectorial subspace of the n-tuples over GF(2), thus it is possible to define its *null* (or *dual*) *space* $\mathcal{C}_d$ as an $(n-k)$-dimensional subspace of V given by:

$$\mathcal{C}_d = \{\mathbf{m_s} \in V : \langle \mathbf{m_s}, \mathbf{c} \rangle = 0 \ \forall \mathbf{c} \in \mathcal{C}\}, \tag{1.5}$$

where $\langle \mathbf{m_s}, \mathbf{c} \rangle$ denotes the inner product of $\mathbf{m_s}$ and $\mathbf{c}$. Consequently, $\mathcal{C}_d$ can in turn be seen as a $(n, n-k)$ linear block code, called the *dual code* of $\mathcal{C}$. As done for $\mathcal{C}$, we can define a basis $\mathcal{B}_d$ of $\mathcal{C}_d$ formed by $n-k$ linearly independent codewords in $\mathcal{C}_d$. Denoting these codewords as $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k-1}$, and arranging them into a matrix, we obtain

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \cdots \\ \mathbf{h}_{n-k-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & \cdots & h_{n-k-1,n-1}. \end{bmatrix} \tag{1.6}$$

$\mathbf{H}$ is a generator matrix of the dual code $\mathcal{C}_d$ of the code $\mathcal{C}$. From the definition of $\mathbf{G}$ and $\mathbf{H}$ follows that $\mathbf{G} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{O}$, where $\mathbf{O}$ is a $k \times (n-k)$ zero matrix and $^{\mathrm{T}}$ denotes the matrix transposition. Given these relationships, it follows that $\mathbf{c} \in V$ is a valid codeword of $\mathcal{C}$ if and only if $\mathbf{c} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}$, which can also be expressed as

$$\mathcal{C} = \left\{\mathbf{c} \in V : \mathbf{c} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}\right\}. \tag{1.7}$$

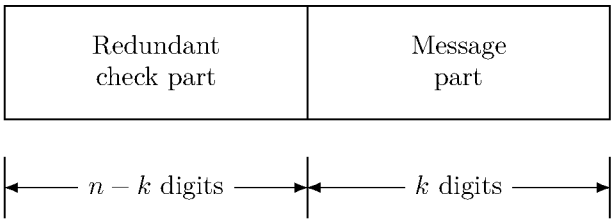


Figure 1.1: Structure of a codeword of a linear systematic code.

This means that $\mathcal{C}$ is also uniquely specified by the matrix $\mathbf{H}_{[n-k\times n]}$, which is called the *parity-check* matrix of $\mathcal{C}$.

Sometimes, it is also desirable to have codewords in the format shown in Figure 1.1, that is, divided into the message part and the redundant check part. In particular, the former consists of the k unaltered information digits, while the latter consists of the $n-k$ parity-check digits. A linear block code with this structure is referred to as a *linear systematic code* and it is completely specified by the following $k \times n$ generator matrix:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \underbrace{\begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} & 1 & 0 & \cdots & 0 \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & \cdots & 1 \end{bmatrix}}_{\mathbf{P}\text{ matrix} \qquad k\times k\text{ identity matrix }\mathbf{I}_k}. \tag{1.8}$$

This particular structure is called *standard form* of $\mathbf{G}$, that can be therefore written as $\mathbf{G} = [\mathbf{P}\ \mathbf{I}_k]$, where $\mathbf{P}$ is a $k \times (n-k)$ submatrix with entries over GF(2), and $\mathbf{I}_k$ is a $k \times k$ identity matrix. Note that, given a generator matrix $\mathbf{G}'$ of an (n,k) linear block code $\mathcal{C}'$ not in standard form, it is always possible to obtain a *combinatory equivalent* matrix $\mathbf{G}$ in the standard form performing elementary operations on the rows of $\mathbf{G}'$ and, if necessary, taking columns permutations. The (n,k) linear block code $\mathcal{C}$ generated by $\mathbf{G}$ is then a systematic code and it is called a *combinatory equivalent code* of $\mathbf{G}'$. Even though, in general, the codewords of $\mathcal{C}$ and $\mathcal{C}'$ may differ from the arrangement of code bits, two combinatory equivalent (n,k) linear block codes give the same error performance.

Furthermore, if a generator matrix of a (n,k) linear block code $\mathcal{C}$ is given in the standard form of (1.8), its corresponding parity-check matrix in standard form is given by:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & \ldots & 0 & p_{0,0} & p_{1,0} & \cdots & p_{k-1,0} \\ 0 & 1 & \ldots & 0 & p_{0,1} & p_{1,1} & \cdots & p_{k-1,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & p_{0,n-k-1} & p_{1,n-k-1} & \cdots & p_{k-1,n-k-1} \end{bmatrix} \tag{1.9}$$

which can be expressed as $\mathbf{H} = \left[\mathbf{I}_{n-k}\ \mathbf{P}^{\mathrm{T}}\right]$.

In order to understand the concept of *error detection*, some other elements must be introduced. Suppose that a codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathcal{C}$ is transmitted over a binary-input, binary-output channel (e.g., Binary Symmetric Channel (BSC)) and let $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ be the hard-decision received vector at the input of the channel decoder. Due to the channel noise, $\mathbf{r}$ and $\mathbf{c}$ may differ in some places so, given a vector $\mathbf{e}$ defined as

$$\begin{aligned}\mathbf{e} &= \mathbf{c} + \mathbf{r} \\ &= (e_0, e_1, \ldots, e_{n-1}) \\ &= (c_0 + r_0, c_1 + r_1, \ldots, c_{n-1} + r_{n-1}),\end{aligned} \tag{1.10}$$

we have that $e_j = 1$ for $c_j \neq r_j$ and $e_j = 0$ for $c_j = r_j$. In other words, $\mathbf{e}$ expresses the *error pattern* introduced by the channel. Therefore, the received vector $\mathbf{r}$ can be expressed as

$$\mathbf{r} = \mathbf{c} + \mathbf{e}. \tag{1.11}$$

However, the receiver knows neither the transmitted codeword $\mathbf{c}$ nor the introduced error pattern $\mathbf{e}$ and the decoder must, first of all, determine whether there are transmission errors in $\mathbf{r}$.

To check if $\mathbf{r}$ contains transmission errors, the decoder computes the following $(n-k)$-tuple over GF(2)

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^{\mathrm{T}}, \tag{1.12}$$

which is called the *syndrome* of $\mathbf{r}$. Taking into account (1.7), the decoder essentially computes

$$\begin{aligned}\mathbf{s} &= (\mathbf{c} + \mathbf{r}) \cdot \mathbf{H}^{\mathrm{T}} \\ &= \mathbf{e} \cdot \mathbf{H}^{\mathrm{T}}.\end{aligned} \tag{1.13}$$

Therefore, recalling that $\mathbf{r} \in V$ is a valid codeword of $\mathcal{C}$ if and only if $\mathbf{r} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}$, $\mathbf{r}$ is a codeword if and only if $\mathbf{s} = \mathbf{0}$. Note that $\mathbf{s} = \mathbf{0}$ does not necessarily imply that r is error-free. In fact, since $\mathcal{C}$ is a vectorial subspace, if the error pattern $\mathbf{e}$ is identical to a nonzero codeword of $\mathcal{C}$, $\mathbf{r}$ is still a valid codeword. In this case, however, accepting $\mathbf{r}$ as the transmitted codeword yields a *decoding error* since $\mathbf{r} \neq \mathbf{c}$. Thus, there are $2^k - 1$ *undetectable error patterns.*

1.1.1 LDPC codes

LDPC codes are a class of linear block codes presented by Robert Gallager in [12] and mostly ignored for the 35 years that followed due to the computational complexity of the proposed decoder. A fundamental exception is the work of Tanner in 1981 [9], where a graphical representation of LDPC codes, called *Tanner graph*,

was introduced. The revival of LDPC codes, then, began in 1995 with the publication of [13], a paper that highlighted the decoding complexity advantages achievable exploiting the structure of the parity-check matrix of those codes.

In general, LDPC codes are linear block codes defined as the null space of a *sparse* parity-check matrix $\mathbf{H}$, i.e., a binary matrix that has a low density of 1s. This structure permits effective iterative decoding, since the complexity of iterative decoders is (also) related to the number of 1s within the parity-check matrix. In other words, LDPC codes have made it operatively possible to use codes with very high values of n that approach Shannon's limit.

Furthermore, the $\mathbf{H}$ of an LDPC code can be easily represented with a sparse *Tanner graph*.

Let us consider, for example, the following parity-check matrix of a block code

$$\mathbf{H} = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{matrix} \\ \left[\begin{matrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{matrix}\right] \end{array} \begin{matrix} c_1 \\ c_2, \\ c_3 \end{matrix} \tag{1.14}$$

from which two sets of nodes can be identified:

- **Variable Nodes (VN)**, denoted as v_j, whose number is equal to the number of columns of $\mathbf{H}$, i.e., the number of symbols in a codeword.
- **Check Nodes (CN)**, denoted as c_i, whose number is equal to the number of rows of $\mathbf{H}$, that, if $\mathbf{H}$ has full-rank, is equal to the number of redundancy symbols introduced by the code itself.

These sets define a bipartite graph drawn as follows: the check node c_i is connected to the variable node v_j whenever $h_{i,j} = 1$.

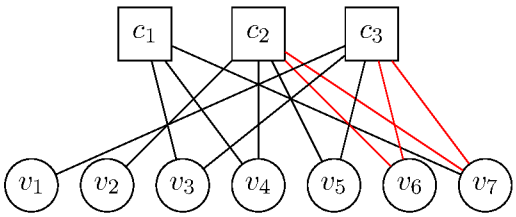


Figure 1.2: Tanner graph of the code described by (1.14).

The Tanner graph of an LDPC code is directly involved in the decoding process because, as will become clear later, every iteration is basically split into two semi-iterations, in which every node acts as an operating processor and every edge conveys information from a given node to its neighbors. Furthermore, the effectiveness of the iterative decoders depend on some structural properties of the Tanner graph. More specifically, in order to mitigate convergence issues in the decoding process, it is important to avoid short *cycles*, i.e., it is important to prevent the information

sent by a node to its neighbors from returning to the node itself too early. In terms of parity-check matrix, this imposes a structural property on $\mathbf{H}$: no two rows (or two columns) should have more than one position in common that contains a nonzero element. This property is called *row-column constraint*.

The *length* of a cycle is given by the number of edges that form the cycle and a cycle of length l is called an l-cycle. The minimum cycle length in a bipartite graph is called the graph's *girth*. Obviously, the shortest possible cycle is a 4-cycle. Moreover, based on the above considerations, the need for a sparse parity-check matrix is also evident: at high densities, many short cycles will exist, thus severely degrading the performance of iterative decoders.

Depending on the *weights* of the rows and the columns of $\mathbf{H}$, i.e., the number of 1s in each row and column of the parity-check matrix, LDPC codes can be classified as either *regular* or *irregular*. In the former, every row and every column of $\mathbf{H}$ has a constant weight, denoted as w_r and w_c, respectively; in the latter, the weights vary depending on the considered row or column. It is worth mentioning that there also exist codes that are regular only in the rows or the columns, i.e., codes with fixed w_r and variable w_c and vice versa. Generally speaking, regular LDPC codes do not exhibit *error floor*, that is, a flattening of the Bit Error Rate (BER) curve at high values of Signal-to-Noise Ratio (SNR), but their *waterfall region*, i.e., the range of SNR values in which the BER curve is steeper, is typically far from Shannon's limit, thus making these codes unsuitable for many applications with strict power constraints. Irregular LDPC codes, on the other hand, come closer to Shannon's limit in the waterfall region, but suffer from error floor, being typically unsuitable for applications that require very low BERs.

In [14], a *protograph*-based construction of (J, K)-regular LDPC codes, i.e., codes where every variable node is connected to J check nodes and every check node is connected to K variable nodes, is illustrated. In particular, a protograph with rate $R = 1 - n_c/n_v$ is a bipartite graph (V, C, E) that connects a set of n_v variable nodes $V = \{v_1, v_2, \ldots, v_{n_v}\}$ to a set of n_c check nodes $C = \{c_1, c_2, \ldots, c_{n_c}\}$ with a set of edges E. The protograph can be represented by its *base matrix* $\mathbf{B}_{[n_c \times n_v]}$, where the element $B_{i,j}$ represents the number of edges that connect the check node c_i to the variable node v_j. Note that a protograph may have many edges connecting a check node and a variable node, i.e., $\mathbf{B}$ can have entries greater than 1. Given this structure, it is possible to obtain the parity-check matrix $\mathbf{H}$ of an LDPC code with $n = Mn_v$, with M called the *lifting factor*. Moreover, the matrix $\mathbf{H}_{[Mn_c \times Mn_v]}$ can be obtained by substituting every nonzero element of $\mathbf{B}$ with a sum of $B_{i,j}$ $M \times M$ non-overlapping permutation matrices, and every null element of $\mathbf{B}$ with an $M \times M$ null matrix. For example, the protograph shown in Figure 1.3 is described by the following base matrix

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{1.15}$$

and, based on the above, the parity-check matrix corresponding to an M-lifting of B is therefore given by

$$\mathbf{H} = \begin{bmatrix} \Pi_{1,1} & \Pi_{1,2} & \Pi_{1,3} \\ \Pi_{2,1} & \Pi_{2,2} & \Pi_{2,3} \end{bmatrix} \tag{1.16}$$

where $\Pi_{i,j}$ is an $M \times M$ permutation matrix. It is important to highlight that, since a block LDPC code is defined as the null space of a parity-check matrix $\mathbf{H}$, a protograph identifies an *ensemble* of LDPC codes, formed by all the parity-check matrices that can be obtained by lifting the same base matrix $\mathbf{B}$.

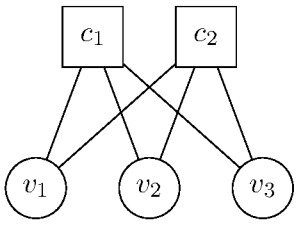


Figure 1.3: Protograph of a (3, 6)-regular LDPC code.

1.2 Convolutional codes

As reported in [15], a *convolutional code* is generated by passing an information sequence through a linear finite-state shift register. In general, the shift register consists of K stages of k bits each, and n modulo-2 adders, as shown in the example of Figure 1.4. For each k-bit input sequence, the encoder produces an n-bit output sequence, thus the code rate is given by $R = k/n$, consistent with the definition of code rate for block codes. In total, the shift register consists of $L = Kk$ delay elements, where K is called the *constraint length* of the code. Due to the structure of the shift register, the n output bits generated at a given time instant are not only a function of the k input bits at the same time instant, but also of the bits in the previous $(K-1)k$ stages. Therefore, the shift register is a finite-state machine with $2^{(K-1)k}$ possible states.

Convolutional codes, as block codes, can be described by their generator matrix. However, in general, the generator matrix of a convolutional code is semi-infinite since the input sequence is semi-infinite itself. For this reason, it is possible to represent a convolutional code in a functionally equivalent way using a set of n Kk-dimensional vectors, one for every modulo-2 adder in the encoder, such that a 1 in the j-th position of the vector $\mathbf{g}_i$ indicates that the j-th stage in the shift register is connected to the i-th modulo-2 adder.

For example, let us consider the binary convolutional encoder shown in Figure 1.5, with constraint length $K = 3$, $k = 1$, and $n = 3$. Based on the above considerations, the shift register can therefore be described by the following set of vectors

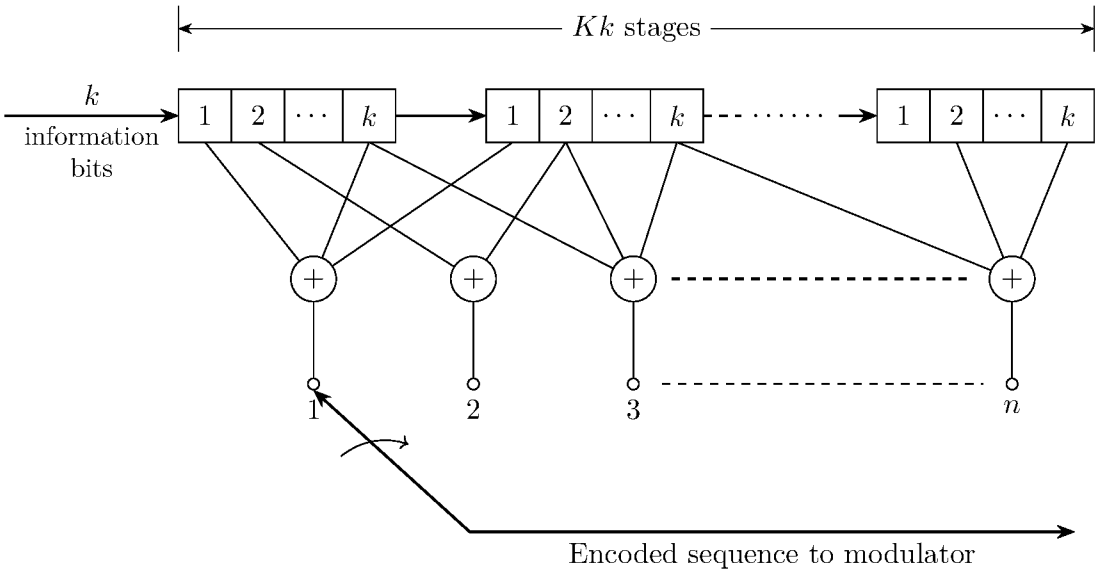


Figure 1.4: Schematic representation of a convolutional encoder [15].

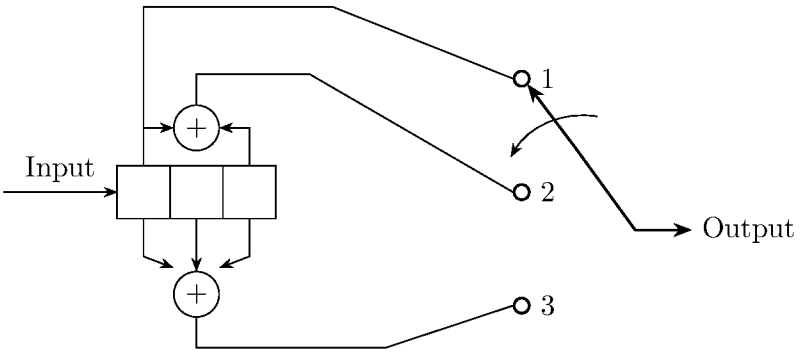


Figure 1.5: Example of a convolutional encoder with $K = 3$, $k = 1$, $n = 3$ [15].

$$\begin{aligned} \mathbf{g}_1 &= [100] \\ \mathbf{g}_2 &= [101] \\ \mathbf{g}_3 &= [111] \end{aligned} \tag{1.17}$$

that can also be expressed in their octal form as $(4, 5, 7)$. These *generators* correspond to the n impulsive responses from the encoder input to the three outputs. Therefore, denoting as $\mathbf{c}^{(i)}$ the i-th output of the shift register and as $\mathbf{u}$ the input sequence, the following equations are valid

$$\begin{aligned} \mathbf{c}^{(1)} &= \mathbf{u} * \mathbf{g}_1 \\ \mathbf{c}^{(2)} &= \mathbf{u} * \mathbf{g}_2 \\ \mathbf{c}^{(3)} &= \mathbf{u} * \mathbf{g}_3 \end{aligned} \tag{1.18}$$

where $*$ denotes the *convolution* operation. The corresponding code sequence $\mathbf{c}$ is thus obtained by interleaving the above vectors as

$$\mathbf{c} = \left(c_1^{(1)}, c_1^{(2)}, c_1^{(3)}, c_2^{(1)}, c_2^{(2)}, c_2^{(3)}, \dots\right). \tag{1.19}$$

Let us define the D transform[1] of $\mathbf{u}$ as

$$u(D) = \sum_{i=0}^{\infty} u_i D^i \tag{1.20}$$

and the transfer function of the three impulse responses as

$$\begin{aligned} g_1(D) &= 1 \\ g_2(D) &= 1 + D^2 \\ g_3(D) &= 1 + D + D^2. \end{aligned} \tag{1.21}$$

Therefore, since the convolution operation is equivalent to the multiplication in the transform domain, the output transforms are given by

$$\begin{aligned} c^{(1)} &= u(D)g_1(D) \\ c^{(2)} &= u(D)g_2(D) \\ c^{(3)} &= u(D)g_3(D). \end{aligned} \tag{1.22}$$

In order to obtain the D transform of the serialized output sequence $\mathbf{c}(D)$, it is necessary to account for the multiplexing of the $n = 3$ encoder outputs onto the channel. Since each input symbol at index t produces $n = 3$ code symbols, the i-th sub-stream $c^{(i)}$ is mapped to the indices $3t + (i - 1)$ within the serialized output sequence. This means that successive symbols of any given sub-stream $c^{(i)}$ are separated by $n = 3$ positions in the multiplexed output sequence, and its D transform must then be evaluated at D^3, thus obtaining

$$\mathbf{c}(D) = c^{(1)}(D^3) + Dc^{(2)}(D^3) + D^2c^{(3)}(D^3). \tag{1.23}$$

Furthermore, these results can be expressed in a more compact representation defining

$$\mathbf{G}(D) = \begin{bmatrix} g_1(D) & g_2(D) & g_3(D) \end{bmatrix} \tag{1.24}$$

from which it follows that

$$\mathbf{c}(D) = \mathbf{u}(D) \cdot \mathbf{G}(D) \tag{1.25}$$

where $\mathbf{G}(D)$ is a $k \times n$ *generator matrix in the transform domain*, whose elements are polynomials in D with a maximum degree of $K - 1$.

[1] Transform domain where D represents a unitary delay introduced by a memory element in the shift register. By substituting $D = z^{-1}$, the D transform becomes the z transform.

1.2.1 SC-LDPC codes

As reported in [16], SC-LDPC codes can be viewed as a type of convolutional LDPC codes, since, as will be explained later, spatial coupling is equivalent to introducing memory into the encoding process. Moreover, the Tanner graph of SC-LDPC codes differ from the graph of a regular LDPC by slight *structured irregularities* that essentially merge in a single design the desirable features of regular and irregular codes:

- they enable the use of iterative decoders whose lowest SNR value for reliable operation, i.e., their *threshold*, approaches Shannon's limit, thus achieving excellent performance in the waterfall region (a characteristic feature of irregular codes).
- Their minimum distance grows linearly with block length, thus promising the elimination of an error floor (a characteristic feature of linear codes).

The primary interest in employing these type of codes in modern communication systems lies in the fact that, as shown in [17], SC-LDPC codes benefit from the phenomenon of *threshold saturation*. Because of this, a BP decoder, despite relying on message passing and being only *locally* optimum, achieves a threshold approaching that of a *globally* optimum, but computationally infeasible, MAP decoder.

Convolutional LDPC codes were first introduced by Jimenez-Feltstrom and Zigangirov in [5]. They are characterized by a semi-infinite parity-check matrix, obtained by repeating several times a (transposed) matrix, called *syndrome former*, which will hereafter be denoted as $\mathbf{H}_S^{\mathrm{T}}$. Specifically, $\mathbf{H}_S^{\mathrm{T}}$ is formed by a sequence of parity-check matrices of LDPC block codes that are, therefore, mutually interconnected. Thus, denoting m_s as the maximum distance between a couple of interconnected blocks, i.e., the *syndrome former memory order*, the parity-check matrix of a generic SC-LDPC code can be represented in the following form

$$\mathbf{H}_{[1,L]} = \begin{bmatrix} \mathbf{H}_0(1) & & & & \\ \vdots & \mathbf{H}_0(2) & & & \\ \mathbf{H}_{m_s}(1) & \vdots & \ddots & & \\ & \mathbf{H}_{m_s}(2) & & \mathbf{H}_0(L) & \\ & & \ddots & \vdots & \\ & & & \mathbf{H}_{m_s}(L) & \end{bmatrix} \tag{1.26}$$

where $[1, L]$ denotes that the SC-LDPC was *terminated* at the time instant $t = L$, i.e., the syndrome former matrix $\mathbf{H}_S^{\mathrm{T}} = \begin{bmatrix} \mathbf{H}_0(t) & \dots & \mathbf{H}_{m_s}(t) \end{bmatrix}^{\mathrm{T}}$ was replicated diagonally L times. As can be inferred form the notation used, the blocks of the syndrome former matrix are, in general, Time-Varying (TV). Although Time-Invariant (TI) versions also exist, i.e., such that $\mathbf{H}_i(t_1) = \mathbf{H}_i(t_2)\ \forall i \in [0, m_s], \forall t_1, t_2 \in [1, L]$, they generally offer slightly inferior performance than their above counterparts and, for

this reason, we will focus on TV codes. In the following, the number of columns of $\mathbf{H}_S^{\mathrm{T}}$ and the vertical shift between two of its replicas will be denoted with a and c, respectively. Having established these notations, it is possible to define the *asymptotic rate* of the code, given by

$$R_\infty = 1 - \frac{c}{a} \tag{1.27}$$

that, as the name suggests, would be the rate of the code obtained by considering an infinite number of replicas of the syndrome former matrix. This value differs from the actual rate of a terminated code, which is given by

$$R = 1 - \frac{c}{a}\left(\frac{L+m_s}{L}\right) = R_\infty - \frac{c}{a}\left(\frac{m_s}{L}\right) \tag{1.28}$$

where the first term is the asymptotic rate defined above and the second term $\Delta R = \frac{c}{a}\left(\frac{m_s}{L}\right)$ explicitly quantifies the *rate loss* introduced by code termination.

Based on these definitions, one can estimate the minimum number of replicas required to construct a terminated code whose performance is comparable to that of its unterminated counterpart; indeed, $\mathbf{H}_S^{\mathrm{T}}$ should be repeated a number of times such that $R \approx R_\infty$.

As with block codes, [18] introduced a code construction based on the concept of *convolutional protographs* that can be obtained by connecting, or *spatially coupling*, a sequence of disjoint block protograph together in a chain. Spatial coupling essentially introduces memory into the code design, i.e., transitioning from a block code to a convolutional code, and is achieved with *edge spreading* operations.

In order to understand the concept of *edge spreading*, consider replicating a block protograph with a variable nodes and c check nodes as an infinite sequence of disjoint graphs, each one associated with a time index t. Given a variable node v_j with $j = 1, 2, \ldots, a$ connected with a certain check node c_i with $i = 1, 2, \ldots, c$ via $B_{i,j}$ edges in each protograph, it is possible to *spread* (connect) the edges emanating from node v_j at time t arbitrarily over the m_s+1 check nodes c_i at times $t, t+1, \ldots, t+m_s$, where $m_s > 0$ denotes the *coupling width* of the graph or memory of the code. Therefore, a convolutional protograph can be constructed by applying edge spreading to variable nodes at all time instants. More specifically, denoting with $b = \gcd(J, K)$, there exist positive integers J', K' such that $J = bJ'$ and $K = bK'$ with $\gcd(J', K') = 1$. Then, the base matrix of a (J, K)-regular protograph-based SC-LDPC ensemble with coupling width $m_s = b - 1$ can be defined as

$$\mathbf{B}_{[-\infty,\infty]} = \begin{bmatrix} \ddots & \ddots & & \ddots & & & \\ & \mathbf{B}_{m_s} & \mathbf{B}_{m_s-1} & \cdots & \mathbf{B}_0 & & \\ & & \mathbf{B}_{m_s} & \mathbf{B}_{m_s-1} & \cdots & \mathbf{B}_0 & \\ & & & \ddots & \ddots & & \ddots \end{bmatrix}, \tag{1.29}$$

where the submatrices $\mathbf{B}_i$, with $i = 0, 1, \ldots, m_s$, are identical $J' \times K'$ matrices, and the notation $[-\infty, \infty]$ indicates that the code was not terminated. A terminated convolutional protograph with coupling width m_s and *coupling length* $L > 0$ can be obtained as the subgraph of the convolutional protograph induced by the variable nodes over time instants $t = 0, 1, \ldots, L-1$, which is equivalent to applying the edge spreading to L disjoint copies of a block protograph, where connections are allowed at the right boundary to wc check nodes in sections $t = L, L+1, \ldots, L+m_s-1$. Therefore, there are a variable nodes and c check nodes at each time instant $t = 0, 1, \ldots, L-1$ and c additional check nodes at each time instant $t = L, L+1, \ldots, L+m_s-1$. Denoting n_v and n_c as the total number of variable nodes and check nodes, respectively, the $n_c \times n_v = (L+m_s)c \times La$ base matrix $\mathbf{B}_{[0,L-1]}$ corresponding to the terminated convolutional protograph can be expressed as

$$\mathbf{B}_{[0,L-1]} = \begin{bmatrix} \mathbf{B}_0 & & & \\ \mathbf{B}_1 & \mathbf{B}_0 & & \\ \vdots & \mathbf{B}_1 & \ddots & \\ \mathbf{B}_{m_s} & \vdots & \ddots & \mathbf{B}_0 \\ & \mathbf{B}_{m_s} & & \mathbf{B}_1 \\ & & \ddots & \vdots \\ & & & \mathbf{B}_{m_s} \end{bmatrix}, \tag{1.30}$$

and can be also obtained by truncating the base matrix in (1.29). Moreover, terminating a convolutional protograph has two effects on the resulting subgraph:

- The graph has a *structured irregularity* because the variable nodes at each time index have the same number of edge connections as the original block protograph, while the first wc and last wc check nodes have a reduced number of edge connections.

- A rate loss is incurred due to the check nodes at the right boundary of the subgraph.

Specifically, the rate of the terminated convolutional protograph with $n_v = La$ variable nodes and $n_c = (L+m_s)c$ check nodes is given by $R = 1 - n_c/n_v$ which, as expected, coincides with equation (1.28).

Example 1. *Let us consider the* $(3, 6)$ *ensemble, where each VN has degree 3 and each CN has degree 6, therefore corresponding to the uncoupled base matrix* $\mathbf{B} = \begin{bmatrix} 3 & 3 \end{bmatrix}$. *In order to obtain an SC version with memory* $m_s = 2$, *the component matrices can be defined as* $\mathbf{B}_0 = \mathbf{B}_1 = \mathbf{B}_2 = \begin{bmatrix} 1 & 1 \end{bmatrix}$, *such that* $\mathbf{B} = \mathbf{B}_0 + \mathbf{B}_1 + \mathbf{B}_2$. *As described above, given, for instance, a coupling length* $L = 6$, *the full convolutional*

base matrix $\mathbf{B}_{\mathrm{SC}}$ *is given by*

$$\mathbf{B}_{\mathrm{SC}} = \begin{bmatrix} \mathbf{B}_0 & & & & & \\ \mathbf{B}_1 & \mathbf{B}_0 & & & & \\ \mathbf{B}_2 & \mathbf{B}_1 & \mathbf{B}_0 & & & \\ & \mathbf{B}_2 & \mathbf{B}_1 & \mathbf{B}_0 & & \\ & & \mathbf{B}_2 & \mathbf{B}_1 & \mathbf{B}_0 & \\ & & & \mathbf{B}_2 & \mathbf{B}_1 & \mathbf{B}_0 \\ & & & & \mathbf{B}_2 & \mathbf{B}_1 \\ & & & & & \mathbf{B}_2 \end{bmatrix}. \tag{1.31}$$

Applying a lifting factor of M*, the parity-check matrix will be of size* $(L+m_s)c \times La = 8M \times 12M$*. For a certain* L*, the design rate is*

$$R_L = R_\infty - \frac{(L+m_s)c}{La} = \frac{1}{2} - \frac{1}{L}, \tag{1.32}$$

that approaches $R_\infty = 1/2$ *as* $L \to \infty$*.*

In [17], the *threshold saturation* phenomenon is described and proved. Denoting the thresholds of a BP decoder and a MAP decoder by ϵ_{BP} and ϵ_{MAP}, respectively, Kudekar, Richardson and Urbanke demonstrated the following relation

$$\lim_{m_s \to +\infty} \lim_{L \to +\infty} \epsilon^{\mathrm{BP}}(J, K, L, m_s) = \lim_{m_s \to +\infty} \lim_{L \to +\infty} \epsilon^{\mathrm{MAP}}(J, K, L, m_s) = \epsilon^{\mathrm{MAP}}(J, K). \tag{1.33}$$

Therefore, with a sufficient coupling length, SC-LDPC codes allow the use of practical iterative decoders (BP-based) that approaches the performance of globally optimum MAP-based decoders.

Chapter 2

Iterative decoding algorithms

This chapter analyzes iterative decoding algorithms, mainly focusing on SPA, and characterizes SC-LDPC codes' sliding window decoders. Then, a comparison is made in terms of complexity and latency between the full SPA and the SWD.

The terms *iterative decoding* and *message passing decoding* are often used interchangeably to refer to a class of low-complexity decoders that work in a distributed fashion to decode a received codeword in a concatenated coding scheme. In particular, LDPC codes can be seen as a generalized collection of many Single Parity Check (SPC) codes, concatenated through an interleaver to a collection of Repetition (REP) codes. This structure, represented graphically in Figure 2.1, allows the application of the *turbo principle* presented in [19], which consists in iteratively exchanging soft information between the decoders of the *component codes* (in this case, the SPC and REP codes) to improve the decoding performance.

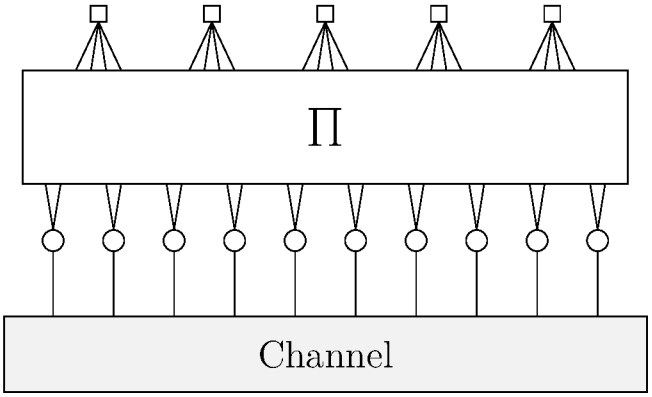


Figure 2.1: Schematic representation of a message passing decoder for an LDPC code.

The description of the message-passing rule, however, requires the introduction of *extrinsic information*, which is the information that a decoder of a component code can provide to the other decoders, excluding the information that it has received from them. In other words, the extrinsic information is the *new* information that a decoder can provide to the others, which is not already known by them. The iterative decoding process consists in iteratively exchanging extrinsic information between the decoders of the component codes until a stopping criterion is met or

the maximum number of iterations is reached.

2.1 Sum-Product Algorithm

The SPA was introduced by Gallager in [12], and it is also sometimes referred to as the BP algorithm. The algorithm is identical for all memoryless channels so its description will be general.

The optimal criterion underlying the SPA decoder is symbol-wise MAP, and we are therefore interested in computing the A Posteriori Probability (APP) of a specific bit in the transmitted codeword $\mathbf{x} = [x_0, x_1, \ldots, x_n]$, given the received sequence $\mathbf{y} = [y_0, y_1, \ldots, y_n]$, i.e. $P(x_i|\mathbf{y})$. Specifically, the SPA computes the estimation of the log-APP ratio, also called Log-Likelihood Ratio (LLR), of the i-th bit, defined as

$$L(x_i|\mathbf{y}) = \log \frac{\Pr(x_i = 0|\mathbf{y})}{\Pr(x_i = 1|\mathbf{y})}. \tag{2.1}$$

In the following L_j will denote the LLR information that the j-th variable node receives from the channel, while $L_{j\to i}$ and $L_{i\to j}$ will denote the LLR information that the j-th variable node sends to the i-th check node and that the i-th check node sends to the j-th variable node, respectively. Moreover, we will denote the set of connections to variable node v_j as the check nodes included in $\mathcal{N}(j) = \{i \mid \mathbf{H}_{i,j} = 1\}$, and the set of connections to check node c_i as the variable nodes included in $\mathcal{M}(i) = \{j \mid \mathbf{H}_{i,j} = 1\}$.

The SPA is essentially based on the MAP decoders of REP codes (for VNs updates) and SPC codes (for CNs updates), and their detailed derivation can be found in [11]. From Equation (2.1), it is clear that the estimation of the *value* of x_i is represented by the sign of $L(x_i|\mathbf{y})$, and that the *reliability*, or confidence, of the decision is represented by the magnitude of $L(x_i|\mathbf{y})$.

As shown in Figure 2.2a, the decoder is initialized by setting all VN messages $L_{j\to i}$ equal to

$$L_j = L(x_j|y_j) = \log \frac{\Pr(x_j = 0|y_j)}{\Pr(x_j = 1|y_j)}, \tag{2.2}$$

where y_j represents the channel value that was actually received. Therefore, we consider the two following special cases, assuming that $\mathbf{x}$ is a Binary Phase Shift Keying (BPSK)-modulated sequence:

AWGN channel. The j-th received sample is

$$y_j = x_j + n_j, \tag{2.3}$$

where the n_j are noise samples that are independent and normally distributed as $\mathcal{N}(0, \sigma^2)$. It is possible to show that

$$\Pr(x_j = x|y_j) = \left[1 + \exp(-2y_j x/\sigma^2)\right]^{-1} \tag{2.4}$$

where $x = \pm 1$ and, from this, that

$$L_j = L(x_j|y_j) = 2y_j/\sigma^2. \tag{2.5}$$

Rayleigh channel. In this case, the j-th received sample is given by

$$y_j = \alpha_j x_j + n_j, \tag{2.6}$$

where $\{\alpha_j\}$ are independent Rayleigh random variables with unity variance, i.e., they are modeled as complex Gaussian random variable with zero mean and unit variance. The real and imaginary parts of each sample are independent and identically distributed as $\mathcal{N}(0, 1/2)$. The channel transition probability is therefore

$$\Pr(x_j = x|y_j) = \left[1 + \exp(-2\alpha_j y_j x/\sigma^2)\right], \tag{2.7}$$

and the variable nodes are initialized by

$$L_j = L(v_j|y_j) = 2\Re\left\{\alpha_j y_j\right\}/\sigma^2. \tag{2.8}$$

Algorithm 1: Sum-Product Algorithm

1. **Initialization**: $\forall j \in \{1, \ldots, n\}$, initialize L_j according to (2.2) for the appropriate channel model. Then, $\forall i \in \mathcal{N}(j)$ set

$$L_{j \to i} = L_j. \tag{2.9}$$

2. **CN update**: Compute outgoing CN messages $L_{i \to j}$, i.e, $\forall j \in \mathcal{M}(i)$ and $\forall i \in \{1, \ldots, n-k\}$, set

$$L_{i \to j} = 2 \tanh^{-1} \left(\prod_{j' \in \mathcal{M}(i) \setminus \{j\}} \tanh \left(\frac{L_{j' \to i}}{2} \right) \right). \tag{2.10}$$

   This step is shown diagrammatically in Figure 2.2b, and each CN operates as a SPC decoder.

3. **VN update**: Compute outgoing VN messages $L_{j \to i}$, i.e., $\forall i \in \mathcal{N}(j)$ and $\forall j \in \{1, \ldots, n\}$, set

$$L_{j \to i} = L_j + \sum_{i' \in \mathcal{N}(j) \setminus \{i\}} L_{i' \to j}. \tag{2.11}$$

   This step is shown diagrammatically in Figure 2.2a, and each VN operates as a REP decoder.

4. **LLR total**: $\forall j \in \{1, \ldots, n\}$, compute

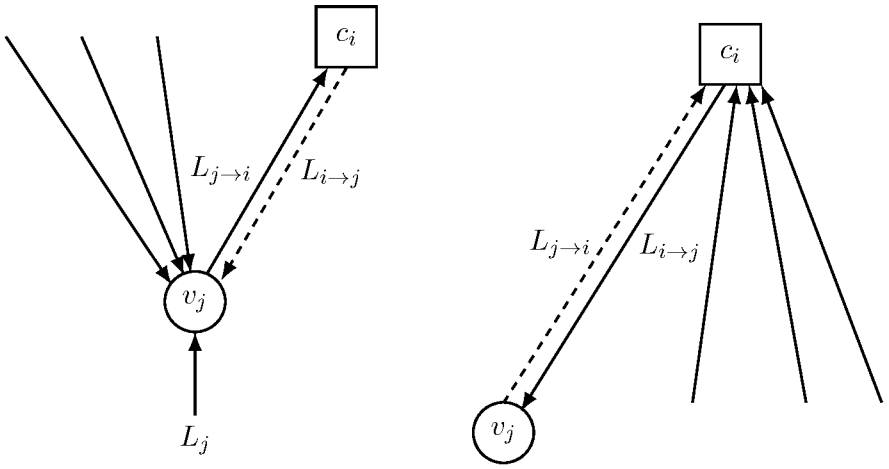
$$L_j^{\text{total}} = L_j + \sum_{i \in \mathcal{N}(j)} L_{i \to j}. \tag{2.12}$$

5. **Stopping criteria**: $\forall j \in \{1, \ldots, n\}$, set

$$\hat{v_j} = \begin{cases} 1 & \text{if } L_j^{\text{total}} < 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2.13}$$

   to obtain $\hat{\mathbf{v}}$. If $\hat{\mathbf{v}}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$ or the iterations equal the maximum limit, stop; else, go to Step 2.

It is important to emphasize that the SPA assumes that the messages passed are statistically independent throughout the decoding process. Even assuming that the y_j are independent, that assumption would hold true only if the Tanner graph possessed no cycles. However, for a graph of girth g, the independence assumption is true only up to the $(g/2)$-th iteration, after which messages start to loop back on themselves in the graph's cycles. If, on the other hand, the codes had graphs without cycles, the algorithm would converge in a finite number of iterations; nevertheless, it has been proved that LDPC codes with such characteristics have a rather small



(a) VN decoder (REP code decoder). (b) CN decoder (SPC code decoder).

Figure 2.2: Schematic representation of the variable node decoder (a) and the check node decoder (b) [11].

minimum distance and therefore very poor performance in terms of BER ([15]).

In terms of computational complexity, the update equation (2.10) for Step 2 is quite heavy due to the presence of tanh and $\tanh^{-1}$ functions. For applications with strict complexity constraints it is possible to use reduced-complexity algorithms that *approximate* the SPA, such as the Min-Sum (MS) algorithm presented in [20]. First of all, factor $L_{j\to i}$ into its bit value and bit reliability, i.e. into its sign and magnitude, respectively:

$$\begin{aligned} L_{j\to i} &= \alpha_{ji}\beta_{ji}, \\ \alpha_{ji} &= \operatorname{sign}(L_{j\to i}), \\ \beta_{ji} &= |L_{j\to i}|, \end{aligned} \tag{2.14}$$

so that (2.10) can be rewritten as

$$\prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \tanh\left(\frac{L_{j'\to j}}{2}\right) = \prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \alpha_{j'i} \cdot \prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \tanh\left(\frac{\beta_{j'i}}{2}\right) \tag{2.15}$$

and the Step 2 of the SPA can therefore be expressed in the following form

$$\begin{aligned} L_{i\to j} &= \prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \alpha_{j'i} \cdot 2\tanh^{-1}\left(\prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \tanh\left(\frac{\beta_{j'i}}{2}\right)\right) \\ &= \prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \alpha_{j'i} \cdot 2\tanh^{-1}\log^{-1}\log\left(\prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \tanh\left(\frac{\beta_{j'i}}{2}\right)\right) \\ &= \prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \alpha_{j'i} \cdot 2\tanh^{-1}\log^{-1}\sum_{j'\in\mathcal{M}(i)\setminus\{j\}} \log\left(\tanh\left(\frac{\beta_{j'i}}{2}\right)\right) \end{aligned}$$

$$= \prod_{j' \in \mathcal{M}(i) \setminus \{j\}} \alpha_{j'i} \cdot \phi \left(\sum_{j' \in \mathcal{M}(i) \setminus \{j\}} \phi \left(\beta_{j'i} \right) \right) \tag{2.16}$$

where $\phi(x)$, defined as,

$$\phi(x) = -\log \left[\tanh \frac{x}{2} \right] = \log \left(\frac{e^x + 1}{e^x - 1} \right) \tag{2.17}$$

is a decreasing function such that $\phi^{-1}(x) = \phi(x)$ when $x > 0$ that can be implemented using a look-up table.

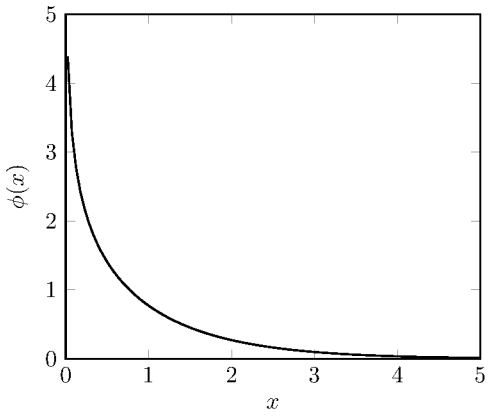


Figure 2.3: Plot of the $\phi(x)$ function.

Therefore, as shown in Figure 2.3, the largest term in the sum of Equation 2.16 corresponds to the smallest β_{ji}. So, assuming that this term dominates the sum,

$$\begin{aligned} \phi \left(\sum_{j' \in \mathcal{M}(i) \setminus \{j\}} \phi \left(\beta_{j'i} \right) \right) &\approx \phi \left(\phi \left(\min_{j' \in \mathcal{M}(i) \setminus \{j\}} \beta_{j'i} \right) \right) \\ &= \min_{j' \in \mathcal{M}(i) \setminus \{j\}} \beta_{j'i}. \end{aligned} \tag{2.18}$$

Thus, the MS algorithm is simply the SPA with Step 2 replaced by

$$L_{i \to j} = \prod_{j' \in \mathcal{M}(i) \setminus \{j\}} \alpha_{j'i} \cdot \min_{j' \in \mathcal{M}(i) \setminus \{j\}} \beta_{j'i}. \tag{2.19}$$

In [11], it was shown that the amount of performance degradation introduced by the MS is a function of the code and the channel, and it is caused by the fact that for the first iteration of a MS decoder the extrinsic information passed from a CN to a VN is, on average, larger in magnitude than the extrinsic information that would have been passed by an SPA decoder. A possible adjustment for this behavior consists in introducing an attenuation factor $0 < c_{\text{atten}} < 1$ of such extrinsic information before

it is passed to a VN. Therefore, the resulting update equation is

$$L_{i\to j} = \prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \alpha_{j'i} \cdot c_{\text{atten}} \cdot \min_{j'\in\mathcal{M}(i)\setminus\{j\}} \beta_{j'i}. \tag{2.20}$$

Alternatively, in [21] was presented a variation of the MS decoder based on the introduction of an offset $c_{\text{offset}} > 0$ to be subtracted from each β_{ji} (with the constraint that the result must be non-negative). Thus, the offset-MS algorithm computes

$$L_{i\to j} = \prod_{j'\in\mathcal{M}(i)\setminus\{j\}} \alpha_{j'i} \cdot \max\left\{\min_{j'\in\mathcal{M}(i)\setminus\{j\}} \beta_{j'i} - c_{\text{offset}}, 0\right\}. \tag{2.21}$$

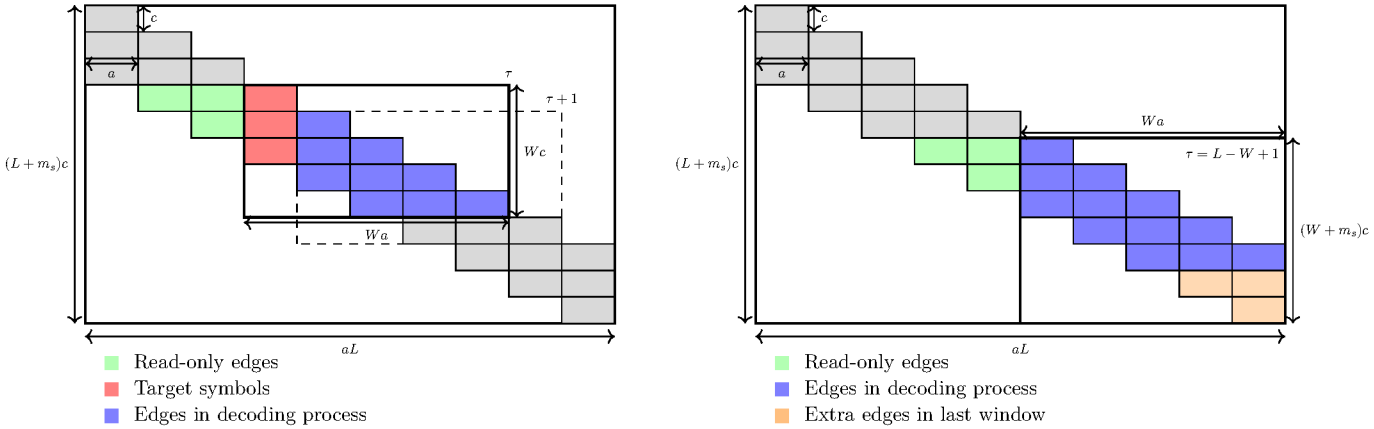
2.2 Sliding window decoding

From the description of Algorithm 1, and as it will be shown later, it is clear that the latency of SPA strongly depends on the dimensions of $\mathbf{H}$. Therefore, the straightforward application of the SPA on SC-LDPC codes is unfeasible. For instance, in order to even start the decoding process, it is necessary to wait until all symbols have been received, which is clearly extremely inefficient for theoretically infinite parity-check matrices. For this reason, in [8] was presented a SWD based on the SPA.

The SWD applies the SPA only on a small portion of the parity-check matrix, i.e. a *window*, at a time. The decoding windows, in particular, must include a number of variable and check nodes that is related to the memory of the SC-LDPC code. Indeed, the maximum number of parity-check equations involving a certain variable node of a protograph-based SC-LDPC code is $(m_s+1)J'M = (m_s+1)c$, while the parity-check equation associated with a certain check node involves a maximum of $(m_s+1)K'M = (m_s+1)a$ variable nodes. Therefore, denoting W as the number of blocks of the syndrome-former matrix ($\mathbf{H}_n(t)$ in equation (1.26)) considered within a window, the SWD applies the SPA on $[Wc \times Wa]$ sized portions of $\mathbf{H}$. As a rule of thumb, it is generally assumed that $W = \rho(m_s+1)$ with $5 \leq \rho \leq 10$ because it was empirically shown that the performance are similar to those achieved by higher values of W, thus reducing the overall latency.

As shown in Figure 2.4a, the decoding of each position τ leads to the decision of the leftmost a symbols included in the window, i.e., the *target symbols*, highlighted in red. Moreover, after decoding the target symbols at position τ, the window shifts by a columns to the right and c rows to the bottom within the parity-check matrix. However, in order to consider *complete* parity-check equations at each position, all nodes connected with the target symbols must be taken into account, including those belonging to the m_s positions preceding the one under consideration.

For this reason, additional $m_s \cdot a$ variable nodes and $m_s \cdot c$ check nodes must be considered in the decoding window and, as reported in Figure 2.4, those nodes are in a *read-only* state, which means that their LLRs are sent to the nodes involved in the decoding process, but are not updated anymore.



(a) SWD at the fourth decoding instance.

(b) Early termination of SWD.

Figure 2.4: Schematic representation of a sliding window decoder for an SC-LDPC code.

The present work builds upon the SWD presented in [22], where the following modifications to the original schedule were proposed:

- *Early termination of windowed decoding*: in general, when $\tau = L - W + 1$, the right side of the window meets the right boundary of the parity-check matrix. In the conventional SWD, the decoding window slides out from the parity-check matrix until it processes the last sub-block as a target, progressively reducing the number of variable and check nodes included in the decoding window. Instead, it is proposed to stop sliding the window when it reaches $\tau = L - W + 1$, and to process all the symbols inside the window as target symbols. As shown in Figure 2.4b, in the last decoding window is therefore necessary to include additional $m_s \cdot c$ check nodes.

- *Message reuse*: in [23], the nodes connected to the previous target symbols are initialized with their *total* LLRs (calculated in Equation (2.12)). However, if the previous target symbols are erroneous, the errors propagate to the current window, thus altering the decoding of current target symbols. In [22] was shown that the *extrinsic* LLRs (calculated in Equation (2.11)) represent a better choice. Therefore, at every window shift, the nodes connected to the previous target symbols, i.e. the nodes included in both the current and the previous decoding window, are initialized with the extrinsic LLRs, while the leftmost a nodes, i.e. the nodes that enter the decoding window, are initialized to *channel*'s LLRs, according to Equation (2.2).

From this point on, all decoding algorithms based on Algorithm 1 will be presented in pseudocode to highlight their implementation details. Therefore, the detailed description of the SWD is provided in Algorithm 2, where $\tau_0 = \max\{0, (\tau - m_s)a\}$.

Algorithm 2: Sliding Window Decoder for SC-LDPC Codes

Input: Parity-check matrix $\mathbf{H}$, received LLR vector $\mathbf{y}$, window size W, maximum iterations $I_{\max}$
Output: Decoded codeword $\hat{\mathbf{x}}$

Initialize $\tau \leftarrow 0$
Initialize decoded codeword $\hat{\mathbf{x}} \leftarrow \mathbf{0}$
Set read-only symbols positions: $[\tau_0, \tau a - 1]$
while $\tau \leq L - W$ **do**
  if $\tau = L - W$ **then**
    Extract $\mathbf{H}_{[\tau c,(L+m_s)c-1],[\tau_0,La-1]}$
    Set target symbols: $[\tau a, La - 1]$
  else
    Extract $\mathbf{H}_{[\tau c,(\tau+W)c-1],[\tau_0,(\tau+W)a-1]}$
    Set target symbols: $[\tau a, (\tau + 1)a - 1]$
  end if
  Extract LLR subvector $\mathbf{y}_{[\tau a,(\tau+W)a-1]}$
  for iteration $I = 1$ to $I_{\max}$ **do**
    Run a conventional BP iteration (without updating the LLRs of read-only symbols)
  end for
  Update decoded values: $\hat{\mathbf{x}}_{[\tau a,(\tau+1)a-1]}$
  Store final extrinsic LLRs for read-only symbols
  Increment $\tau \leftarrow \tau + 1$
end while
Update decoded values: $\hat{\mathbf{x}}_{[(L-W+1)a,La-1]}$
Return $\hat{\mathbf{x}}$

2.2.1 Complexity and latency analysis

As explained above, the main challenge in terms of computational complexity is the implementation of Equation (2.10), i.e. the implementation of the check-node update. In [24], an efficient serial implementation of the SPA has been proposed, which significantly reduces the computational load of the aforementioned operations. So, referring to this serial implementation, and denoting d_c and d_v as the average weight of the columns and the rows of $\mathbf{H}$, the computational load of check-node update, variable-node update and parity-check (C_{LSI}, C_{RSI} and C_{PC}, respectively) can be written as

$$\begin{cases} C_{\text{LSI}} = (n-k)(6d_c - 2) \\ C_{\text{RSI}} = n(2d_v + 1) \\ C_{\text{PC}} = d_c(n-k). \end{cases} \tag{2.22}$$

By taking into account that $d_v = (1-R)d_c$ and $(n-k) = n(1-R)$, the average number of operations required to decode a frame with SPA is given by

$$\begin{aligned} C_{\text{SPA}} &= I_{\text{ave}} n \left[9d_c(1-R) + 2R - 1\right] \\ &= I_{\text{ave}} La \left[9d_c(1-R) + 2R - 1\right]. \end{aligned} \tag{2.23}$$

Considering that the SWD with early termination applies the SPA on $L-W+1$ frames of length $n' = Wa$, denoting I'_{ave} as the average number of SPA iterations required to decode a single window, the average number of operations needed to decode a frame with SWD can be expressed as

$$C_{\text{SWD}} = I'_{\text{ave}}(L-W+1)a\left[9d_c(1-R) + 2R - 1\right]. \tag{2.24}$$

If we set $C_{\text{SWD}} = C_{\text{SPA}}$, we get

$$I'_{\text{ave}} = I_{\text{ave}} \frac{L}{W(L-W+1)}, \quad I_{\text{ave}} = I'_{\text{ave}} \frac{W(L-W+1)}{L}. \tag{2.25}$$

In order to analyze the gain in terms of latency provided by the SWD described above, let us first consider the latency introduced by the conventional SWD, i.e. the decoder that shifts the window by L positions until it processes the last a symbols as target symbols. For decoding the target symbols at a certain window position τ, the overall latency can be expressed as

$$\Gamma^{\tau}_{\text{WD}} = T^{\tau}_R(W) + T^{\tau}_D(W), \tag{2.26}$$

where $T^{\tau}_R(W)$ is the time required to receive Wa symbols at position τ and $T^{\tau}_D(W)$ is the time required to decode the target symbols. Note that if a stopping criterion is applied (for instance, the one introduced in Step 5 of Algorithm 1), $\Gamma^{\tau}_{\text{WD}}$ is also a function of τ. Therefore, to simplify the analysis, we will consider the worst-case scenario, i.e. the one in which the maximum number of iterations is reached at each window position, thus deriving an upper bound to the actual latency of the decoder with a proper stopping rule. Denoting T_F and T_D as the time taken to receive all symbols in a frame and the time taken to decode the frame using the SPA, respectively, the following relations hold:

$$T^{\tau}_R(W) = \begin{cases} \dfrac{Wa}{La} T_F = \dfrac{W}{L} T_F & \text{if } 1 \le \tau \le L-W+1 \\ \dfrac{(L-\tau+1)a}{La} T_F = \left(\dfrac{L-\tau+1}{L}\right) T_F & \text{if } L-W+1 < \tau \le L \end{cases} \tag{2.27}$$

and, similarly,

$$T_D^{\tau}(W) = \begin{cases} \dfrac{Wa}{La} T_D = \dfrac{W}{L} T_D & \text{if } 1 \leq \tau \leq L - W + 1 \\ \dfrac{(L-\tau+1)a}{La} T_D = \left(\dfrac{L-\tau+1}{L} \right) T_D & \text{if } L - W + 1 < \tau \leq L. \end{cases} \tag{2.28}$$

So, under the same processing power, the latency introduced by the decoding of a single window of the SWD is related to that of the SPA as

$$\Gamma_{\text{WD}}^{\tau} = \begin{cases} \dfrac{W}{L} \Gamma_{\text{SPA}} & \text{if } 1 \leq \tau \leq L - W + 1 \\ \left(\dfrac{L-\tau+1}{L} \right) \Gamma_{\text{SPA}} & \text{if } L - W + 1 < \tau \leq L. \end{cases} \tag{2.29}$$

Therefore, the latency of the sliding window is reduced by a factor W/L compared with the SPA for the first $L-W+1$ sub-blocks of target symbols, while it is reduced by a factor $(L-\tau+1)/L$ for the following sub-blocks.

As for the decoding latency of the entire transmission, let us assume, for simplicity, that the reception of a sub-block and the decoding for the corresponding window do not occur at the same time. Under this assumption, the following equation is taken from [22]

$$\Gamma_{\text{WD}}^{F} = T_F + \sum_{\tau=1}^{L-W+1} \frac{W}{L} T_D + \sum_{i_\tau=1}^{W-1} \left(\frac{W - i_\tau}{L} \right) T_D. \tag{2.30}$$

Considering the early-terminated SWD, it is possible to achieve significant latency improvements. Indeed, taking into account that the window does not slide any further from $\tau = L - W + 1$, the overall latency can be expressed as

$$\Gamma_{\text{etWD}}^{F} = T_F + \sum_{\tau=1}^{L-W+1} \frac{W}{L} T_D. \tag{2.31}$$

Moreover, after the reception of the first Wa symbols, the adopted decoding schedule produces new decoded symbols after receiving only a fraction of the symbols required by the block decoder. The average number of symbols that the early-terminated SWD requires to produce new decoded symbols can be expressed as

$$\overline{\Gamma} = \frac{Wa + (L-W)a}{L - W + 1} = \frac{L}{L - W + 1} a. \tag{2.32}$$

However, the latency gain introduced by the SWD necessarily implies a slight loss in performance. Indeed, Figure 2.5 shows the BER achieved by applying Algorithm 2

to a protograph-based SC-LDPC code characterized by $m_s = 2$ and $R \approx 1/2$, under different values of W. As can be seen in the Figure, window sizes $W \geq 5(m_s + 1)$ yield to marginal gains in terms of performance; for this reason, hereafter it will be always assumed that $W = 5(m_s + 1)$.

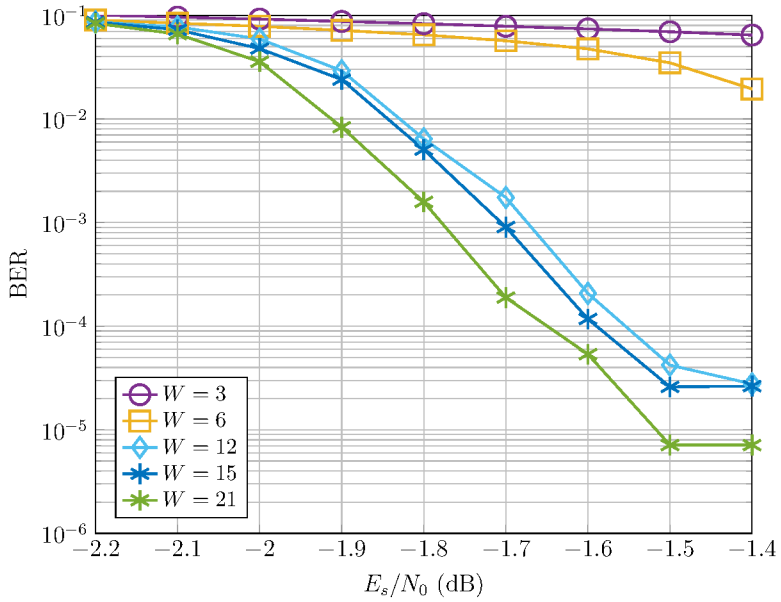


Figure 2.5: BER vs SNR per coded bit for a protograph-based SC-LDPC code, with $L = 500$, $R \approx 1/2$, under Algorithm 2 and different window sizes W

Chapter 3

Error propagation detection and mitigation

A drawback of SWD is potential *decoder error propagation*. Indeed, due to the structure of the decoder, the decoding of a given window also depends on the decoding of a certain number of previous windows. This means that if decoding errors occur in a target block they can propagate to future blocks. In [25], it was also shown that this phenomenon is most noticeable when operating close to capacity and when W is kept small, and it may persist indefinitely as $L \to \infty$. However, Algorithm 2 does not include any strategy to address this problem.

The first section of this chapter discusses some of the methods currently available in the literature to detect error propagation. In particular, it presents two original techniques: a lightweight and heuristic method based on both the number of decoding iterations and a partial syndrome check [10], and an alternative error detection method based on an outer CRC code, which is introduced in this work.

The second section presents two new adaptive decoding strategies that improve the decoder's performance by mitigating the error propagation phenomenon. Specifically, the former decoding strategy is based on resynchronization requests and the latter relies on selective retransmissions as a special form of the type-I HARQ protocol. Moreover, for the type-I HARQ protocol, we also develop a finite-state theoretical model to characterize the interaction between retransmissions and the evolution of the decoding wave across subsequent window positions.

Then, in the last section, we evaluate the impact of the proposed strategies on both the coding rate of the SC-LDPC code and the overall throughput of the transmission.

3.1 Detection techniques

In order to mitigate effectively error propagation, it is first crucial to identify the beginning of an error chain.

Recalling that the magnitude of LLRs represents the reliability of their related messages, following [25], [26], [27], error propagation can be detected by monitoring the *average* magnitude of the LLRs over a subset of symbols and comparing it to a threshold. Moreover, denoting the LLR of the k-th symbol within the τ-th window

position by LLR_k^τ, $k \in [0, Wa-1]$, a mitigation technique is triggered whenever, for some τ,

$$\frac{1}{k_{\text{end}} - k_0 + 1} \sum_{k=k_0}^{k_{\text{end}}} |\text{LLR}_k^\tau| \leq \eta, \tag{3.1}$$

where η is a predetermined threshold, and the indices k_0 and k_{end} define the range of symbols within the decoding window used for the detection. In [25], [27] these are set to $k_0 = 0$ and $k_{\text{end}} = a - 1$, i.e., detection is based on the target symbols. However, in [26], it is mentioned that relying only on the first a symbols within the decoding window to detect error propagation might be sub-optimal. The main problem with this approach lies in determining the optimal value of η because, as it will be clear in Chapter 4, it heavily depends on both the code used and the channel conditions. Thus, the optimal value of η must be determined empirically by performing several simulations, therefore introducing a significant overhead to the whole scheme.

3.1.1 Iteration-based threshold

To address the aforementioned issues with the LLR-averaging method, in [10] we proposed to raise a flag (i.e., send a mitigation request) whenever some iteration threshold J is reached and the parity-check equations under evaluation remain unsatisfied.

Moreover, denoting the index of the last parity-check equation to be checked within the τ-th decoding window by φ, we set

$$\varphi = \begin{cases} (L + m_s)c - 1 & \text{if } \tau = L - W + 1 \\ \\ (\tau + m_s + 1)c - 1 & \text{if } \tau < L - W + 1 \end{cases} \tag{3.2}$$

and, after each iteration, we check if the parity-check equations in the rows $[\tau c, \varphi]$ are satisfied.

Later on, it will be shown that setting $J = I_{\max}$ typically yields better detection performance compared to the LLR-averaging method. This is achieved without requiring further tuning or simulations, and introduces a negligible overhead compared to the overall decoding process.

3.1.2 CRC concatenation

CRC is a checksum algorithm used to detect inconsistencies in data widely employed in many technical standards, such as Ethernet (IEEE 802.3), USB (IEEE 802.3), 5G NR ([2]). CRCs are so called because they add *redundancy* following an algorithm based on systematic *cyclic codes* and they became popular thanks to their implementation simplicity and their good error detection capabilities. Furthermore, since

the check value has a fixed length, the function that generates it is often used as a non-cryptographic hash function.

In particular, the checksum is obtained by dividing modulo two the input data, i.e., the *dividend*, by the *generator polynomial*, i.e., the *divisor*, defined by the used CRC-algorithm. Then, the resulting *remainder* is appended directly to the actual data, and the receiver computes the CRC over the whole data and checks whether the result is 0. If the result is 0, the receiver assumes that the data is correct, otherwise it assumes that an error has occurred during transmission.

Therefore, the proposed error detection method is based on the concatenation of an outer CRC code with SC-LDPC codes. Assuming systematic SC-LDPC encoding, each spatial position contains a transmitted code symbols, divided into $a - c$ systematic symbols and c parity symbols. Unlike block code scenarios, where the CRC is typically applied to the entire codeword, the structure of SC-LDPC codes allows for flexibility in selecting the number of systematic symbols p included in each outer CRC codeword. This flexibility opens up multiple design options, each one with a different impact on the overall code rate and performance of the concatenated scheme:

1. *Partial coverage*: the $a - c$ systematic symbols within a given spatial position are partitioned into K_p parts, individually protected by a separate CRC. Then, assuming that K_p divides $a - c$, every CRC covers

$$p = \frac{a - c}{K_p}$$

   symbols;

2. *Single-position coverage*: a single CRC protects all the $a-c$ systematic symbols associated with a given spatial position, so that

$$p = a - c;$$

3. *Extended coverage*: one CRC protects the systematic symbols of K_e consecutive spatial positions, for some integer K_e, thus yielding

$$p = K_e(a - c).$$

The error rate performance curves shown in Chapter 4 compare these CRC design options. Therefore, supported by the numerical results, hereafter we will adopt the single position coverage option. In terms of error detection, we assume that a mitigation request is issued whenever the receiver observes F consecutive CRC failures, where F is a design parameter that can be tuned to balance the trade-off between detection performance and the throughput of the overall system.

3.2 Mitigation strategies

In recent years, several mitigation strategies have been proposed to address the error propagation problem in SC-LDPC decoding. For instance, in [26], the authors propose different adaptive strategies: increasing the number of decoding iterations when the confidence of the decision is low; reprocessing problematic sections by shifting the decoding window backwards; and shifting the decoding window relying on the estimated location of the decoding wave. Moreover, [28], [29], have explored the idea of using an adaptive window size, i.e., increasing W when error propagation is detected. However, these strategies, while effective, require dynamic adjustments to the decoding schedule, therefore complicating the implementation of the decoder at the receiver side.

In [30], CN doping was proposed, that introduces low-degree CNs at periodic positions in order to facilitate reliable decoding wave initiation. This method, however, requires structural modifications to the code and introduces some rate loss. VNs doping [28], instead, periodically introduces some known bits in the codeword, which mitigate error propagation without altering the code structure but, again, reducing the code rate. Adaptive doping strategies [25], [27] refine this approach by introducing doped bits only after a mitigation request, based on Equation (3.1), is raised. In particular, after a VN doping request, the next block of bits entering the far end of the decoding window is fixed to 0 at the encoder. As shown in Figure 3.1, VN doping requires access to a binary noiseless feedback channel to send mitigation requests and involve the monitoring and computation of average LLRs. Moreover, adaptive doping typically requires tolerating a certain number of decoding errors to maintain continuous transmission.

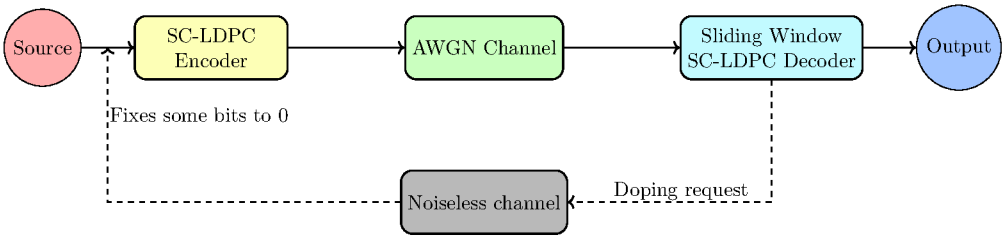


Figure 3.1: Adaptive doping scheme.

Figure 3.2, instead, shows a simplified communication chain where error detection is handled by an outer CRC and recovery relies on resynchronization or HARQ, highlighting that the feedback intervenes midway between the source and the CRC encoder.

3.2.1 Encoder resynchronization

This technique can be paired with any of the detection methods described in the previous section (e.g., the iteration-based threshold method or the CRC concate-

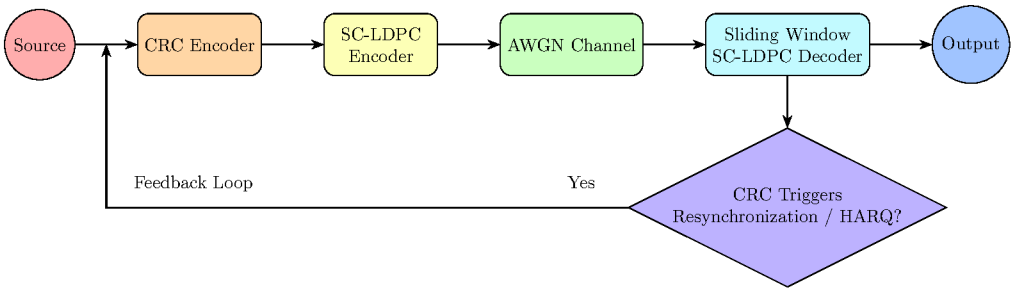
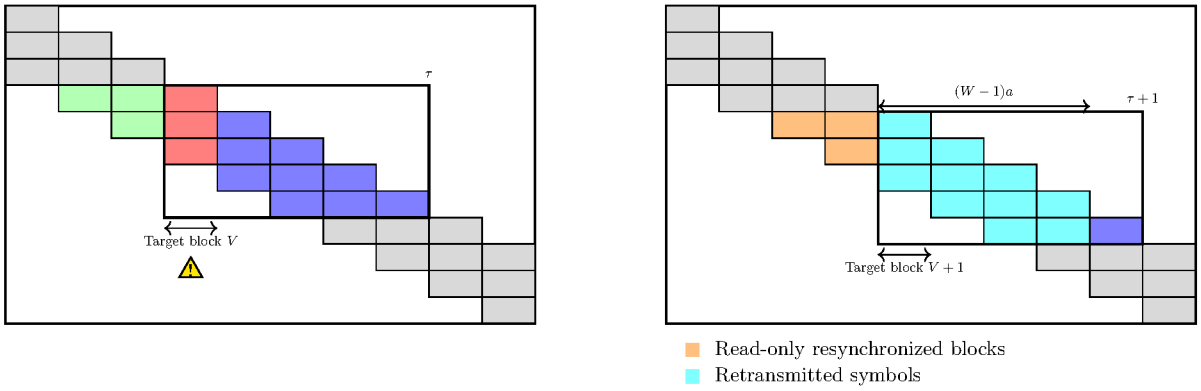


Figure 3.2: CRC-based detection and mitigation scheme.



(a) Detecting an error block.

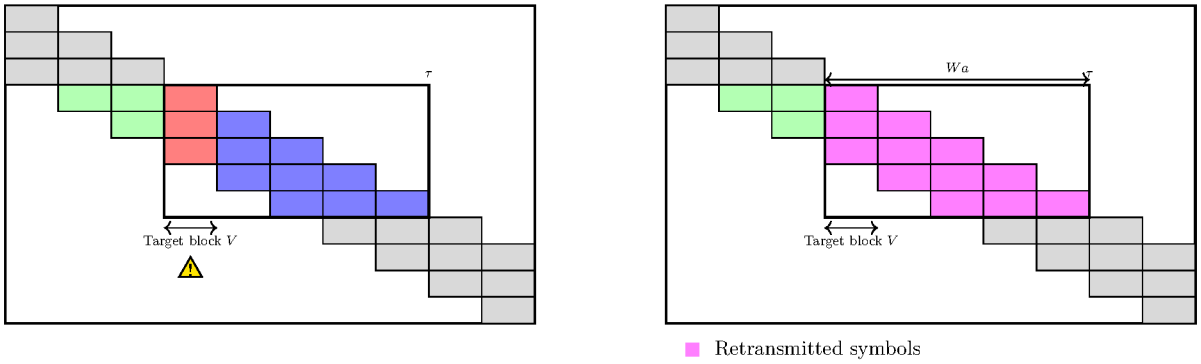
(b) Performing resynchronization.

Figure 3.3: Sliding window decoder: (a) error detection in the V-block; (b) implementing resynchronization in the $(V+1)$-th block.

nation method). Therefore, defining a certain function `detectPropagation()` and a flag **mitigationRequest**, the proposed adaptive resynchronization strategy is described in Algorithm 3. Whenever the flag is raised, the decoding window slides as in Algorithm 2, but a resynchronization request, containing the index V of the target block, is sent to the transmitter over a noiseless feedback channel. Then, the encoder handles the resynchronization process by taking the following steps:

1. It buffers back to the $\max\{0, (V - m_s + 1)\}$-th block;
2. It sets all the symbols in the blocks with indexes in the range $[\max\{0, (V - m_s + 1)\}, V]$ to "0" (or any pre-agreed arbitrary pattern of fixed and known symbols);
3. It resumes the transmission to the receiver from the $(V+1)$-th block.

Thus, after the retransmission of the $W-1$ blocks with indexes in the range $[V+1, V+W-1]$, the decoder decodes the $(V+1)$-th block as usual, except that the LLRs of the read-only symbols are set to their clipping values ℓ. In this way, the decoder is able to break the error chain relying on the high reliability of the fixed symbols, without requiring any structural modification to the code design.



(a) Detecting an error block.

(b) Performing retransmission.

Figure 3.4: Sliding window decoder: (a) error detection in the V-block; (b) implementing type-I HARQ.

Figure 3.3 illustrates the process from the decoder's perspective. In particular, Figure 3.3a shows the issuing of a mitigation request after the detection of an error, while Figure 3.3b depicts the implementation of the resynchronization process during the decoding of the $(V + 1)$-th block.

It is worth underlining that the proposed adaptive resynchronization and LLR-based adaptive doping differ profoundly in their objectives. In fact, the latter employs no retransmissions and typically is paired with small values of η in order to trigger doping only when the LLRs are extremely unreliable. In other words, doping aims to preserve continuous transmission and to prevent the decoder from entering a catastrophic error propagation state that persists until the end of a long frame, limiting severe BER degradation tolerating a minimal rate loss. On the other hand, adaptive resynchronization employs some retransmissions and paired, for instance, with more sensitive CRC conditions, seeks to detect also random block errors, therefore aiming to minimize BER and Block Error Rate (BLER) at the expense of reduced throughput.

3.2.2 Type-I HARQ

As remarked in the previous section, the proposed adaptive resynchronization strategy requires the use of a noiseless feedback channel to send mitigation requests. The use of a feedback channel, however, opens up the possibility of implementing mitigation strategies based on retransmissions, such as the type-I HARQ protocol. In this scenario, assuming that the error detection method is based on CRC concatenation, when the receiver observes F consecutive CRC failures, instead of sending a resynchronization request, it issues a request for the full retransmission of Wa symbols. These correspond to the symbols within the window that most recently produced a target block whose CRC check failed. In particular, this window contains the a target symbols originally protected by the CRC and the subsequent $W - 1$ blocks,

Algorithm 3: Sliding Window Decoder with Adaptive Resynchronization for SC-LDPC Codes

Input: Parity-check matrix $\mathbf{H}$, received LLR vector $\mathbf{y}$, window size W, maximum iterations $I_{\max}$, detection method `detectPropagation()`
Output: Decoded codeword $\hat{\mathbf{x}}$

Initialize $\tau \leftarrow 0$
Initialize decoded codeword $\hat{\mathbf{x}} \leftarrow \mathbf{0}$
Initialize **mitigationRequest** $\leftarrow$ **false**
while $\tau \leq L - W$ **do**
  if $\tau = L - W$ **then**
    Extract $\mathbf{H}_{[\tau c,(L+m_s)c-1],[\tau_0,La-1]}$
    Set target symbols: $[\tau a, La - 1]$
  else
    Set $\varphi \leftarrow (\tau + m_s + 1) \cdot c - 1$
    Extract $\mathbf{H}_{[\tau c,(\tau+W)c-1],[\tau_0,(\tau+W)a-1]}$
    Set target symbols: $[\tau a, (\tau + 1)a - 1]$
  end if
  Extract LLR subvector $\mathbf{y}_{[\tau a,(\tau+W)a-1]}$
  Identify read-only symbols: $[\tau_0, \tau a - 1]$
  if mitigationRequest = true then
    Send a resynchronization request to the transmitter
    Set LLRs of read-only symbols to ℓ
  end if
  for iteration $I = 1$ to $I_{\max}$ **do**
    Run BP iteration (keep read-only LLRs fixed)
  end for
  Update decoded values: $\hat{\mathbf{x}}_{[\tau a,(\tau+1)a-1]}$
  if `detectPropagation()` = **true then**
    Set **mitigationRequest** $\leftarrow$ **true**
  else
    Set **mitigationRequest** $\leftarrow$ **false**
  end if
  Store extrinsic LLRs for read-only symbols
  Increment $\tau \leftarrow \tau + 1$
end while
Update decoded values for last symbols: $\hat{\mathbf{x}}_{[(L-W+1)a,La-1]}$
Return $\hat{\mathbf{x}}$

along with their associated parity checks. As shown in Figure 3.4, the symbols within the current decoding window are retransmitted, thus allowing the decoder to make a fresh attempt at correcting the erroneous target block, without relying on the LLRs from the previous transmission. We denote the maximum allowed number of retransmissions of the same window by γ.

This process can be viewed as a form of the type-I HARQ protocol because the receiver does not combine the information from multiple transmissions (i.e., no chase combining), nor does it request additional redundancy (type-II HARQ). In fact, the

original transmission is discarded, and the receiver relies solely on the information from the most recent retransmission attempt. The proposed HARQ protocol, considering $F = 1$, i.e., issuing a retransmission request after a single CRC failure, is described in Algorithm 4.

It is also important to point out that the receiver control can be implemented as a simple fixed three-state finite-state machine per window position: DECODE $\rightarrow$ CHECKCRC $\rightarrow$ (ADVANCE if the CRC is satisfied or the retransmission limit is reached; REQUEST-RETX otherwise). Moreover, the number of states and possible state transitions are fixed and independent of the code design and channel conditions. From a hardware perspective, the HARQ mechanism can be implemented by adding two small buffers to the baseline decoder: a retransmission counter $K \in [0, \gamma]$ of $\lceil \log_2(\gamma + 1) \rceil$ bits (where γ is typically small), and a one-bit request flag. Recalling that type-I HARQ essentially overwrites the current window LLRs, no storage of past receptions is needed and the overall memory requirements of the decoder remain unchanged. Comparing Algorithm 4 with Algorithm 2, it can be seen that the proposed strategy reuse the same datapath as the baseline SWD (window submatrix extraction and BP iterations). Therefore, a retransmission corresponds to re-entering the standard decoding loop rather than handling a special decoding state. Overall, the additional implementation complexity is limited to a simple control logic and a one-bit feedback request, while the decoder datapath and memory structure remain identical to those of the baseline SWD.

Theoretical modeling of CRC-based type-I HARQ

According to the previous discussion, we will consider type-I HARQ without soft combining. At position τ, the decoder attempts to decode the target block $\hat{\mathbf{x}}_{[\tau a,(\tau+1)a-1]}$ and checks its integrity through the CRC. If the CRC fails, the receiver sends a retransmission request to the transmitter (assuming $F = 1$). Then, upon receiving the retransmitted blocks corresponding to the current window, the decoder makes a fresh attempt to decode the target block. Recalling that the maximum number of retransmissions is γ, the maximum number of decoding attempts per window is $\gamma + 1$.

Let $K \in [0, \gamma]$ represent the index of the retransmission attempt, where $K = 0$ corresponds to the original transmission. Moreover, define $S_{\tau,K} \in \{0, 1\}$ as the CRC outcome at the K-th decoding attempt of the τ-th window, where $S_{\tau,K} = 1$ indicates a successful CRC check and $S_{\tau,K} = 0$ indicates a failure. A successful CRC check ends the HARQ procedure, but it does not guarantee that the decoded block is correct, as undetected errors may still occur.

Therefore, the number of decoding attempts spent at window position τ can be

Algorithm 4: Sliding Window Decoder for SC-LDPC Codes with type-I HARQ

Input: Parity-check matrix $\mathbf{H}$, received LLR vector $\mathbf{y}$, window size W, maximum iterations $I_{\max}$, maximum retransmissions γ, CRC check function `checkCRC()`
Output: Decoded codeword $\hat{\mathbf{x}}$

Initialize $\tau \leftarrow 0$
Initialize decoded codeword $\hat{\mathbf{x}} \leftarrow \mathbf{0}$
Initialize the counter of retransmissions $K \leftarrow 0$
while $\tau \leq L - W$ **do**
  if $\tau = L - W$ **then**
    Extract $\mathbf{H}_{[\tau c,(L+m_s)c-1],[\tau_0,La-1]}$
    Set target symbols: $[\tau a, La - 1]$
  else
    Extract $\mathbf{H}_{[\tau c,(\tau+W)c-1],[\tau_0,(\tau+W)a-1]}$
    Set target symbols: $[\tau a, (\tau + 1)a - 1]$
  end if
  Extract LLR subvector $\mathbf{y}_{[\tau a,(\tau+W)a-1]}$
  Set read-only symbols positions: $[\tau_0, \tau a - 1]$
  for iteration $I = 1$ to $I_{\max}$ **do**
    Run a conventional BP iteration (without updating the LLRs of read-only symbols)
  end for
  Update decoded values: $\hat{\mathbf{x}}_{[\tau a,(\tau+1)a-1]}$
  if `checkCRC`$(\hat{\mathbf{x}}_{[\tau a,(\tau+1)a-1]})$ = **true or** $K = \gamma$ **then**
    Store final extrinsic LLRs for read-only symbols
    Increment $\tau \leftarrow \tau + 1$
    $K \leftarrow 0$
  else
    if $K < \gamma$ **then**
      Ask for retransmission of symbols $\mathbf{y}_{[\tau a,(\tau+W)a-1]}$
      Set the value of the LLRs from the symbols included in the decoding window to the values of $\mathbf{y}_{[\tau a,(\tau+W)a-1]}$
      $K \leftarrow K+1$
    end if
  end if
end while
Update decoded values: $\hat{\mathbf{x}}_{[(L-W+1)a,La-1]}$
Return $\hat{\mathbf{x}}$

expressed as

$$N_\tau = \begin{cases} 1 + \min_{K\in[0,\gamma]}\{K : S_{\tau,K} = 1\}, & \sigma_\tau = 1, \\ \gamma + 1, & \sigma_\tau = 0 \end{cases} \tag{3.3}$$

where $N_\tau = \gamma + 1$ means that the maximum number of retransmissions has been reached without observing a successful CRC, i.e., all the $\gamma + 1$ decoding attempts

have failed.

Then, let us define

$$\sigma_\tau = \begin{cases} 1, & \text{if } \exists K \in [0,\gamma] : S_{\tau,K} = 1 \\ 0, & \text{if } \nexists K \in [0,\gamma] : S_{\tau,K} = 1 \end{cases}, \tag{3.4}$$

where $\sigma_\tau = 1$ indicates that the window is resolved within the retransmission budget, and $\sigma_\tau = 0$ indicates that the budget is exhausted.

In order to model the variability of decoding conditions across different window positions, we introduce a finite-valued state variable

$$X_\tau \in [1, Q], \tag{3.5}$$

that summarizes how previous window outcomes affect the probability of resolving the current window within the retransmission budget. It is possible to adopt several finite-state representations depending on the desired modeling accuracy and complexity. For instance, in the following, the state is defined from the history of retransmmission budget exhaustion events. Since the aforementioned history is directly observable, this definition does not require to introduce empirically selected or unobservable decoding difficulty levels.

Note that the state-dependant parameters are assumed to depend on X_τ, but not separately on the absolute window position τ. We call this assumption the finite state approximation.

Let us first specialize the model to the case $\gamma = 1$ considered in the numerical validation. In the following, let

$$\mu_x \triangleq \Pr(S_{\tau,0} = 1 \mid X_\tau = x) \tag{3.6}$$

denote the probability that the first decding attempt at a certain window position in state x produces a successful CRC check. Assuming $\gamma = 1$, the retransmission budget is exhausted only if both attempts fail, and we can therefore express the budget exhaustion probability f_x and the window resolution probability s_x as

$$f_x \triangleq \Pr(\sigma_\tau = 0 \mid X_\tau = x) = \kappa_x (1 - \mu_x)^2 \tag{3.7}$$

$$s_x \triangleq 1 - f_x. \tag{3.8}$$

In the definitions above, $(1-\mu_x)^2$ represents the budget exhaustion probability that would be obtained if, conditioned on $X_\tau = x$, the two attempts were independent (and each failed with probability $1-\mu_x$), while the factor κ_x rescales this probability to the actual value f_x. Indeed, even though the retransmitted symbols experience independent channel realizations, the two decoding outcomes are not independent when conditioned only on X_τ. In fact, the state X_τ records only the budget exhaustion outcomes of the preceding m_s position, not including the actual read-only

information inherited by the current window. This means that the same value of X_τ can correspond to decoder conditions characterized by different failure probabilities, and these conditions are shared by the two attempts at the same window position.

Taking into account that retransmission budget exhaustion necessarily implies failure of the first attempt, $0 \leq f_x \leq 1 - \mu_x$. Hence, for $\mu_x < 1$,

$$0 \leq \kappa_x \leq \frac{1}{1 - \mu_x},$$

where κ_x can be equivalently written as

$$\kappa_x = \frac{\Pr(S_{\tau,1} = 0 \mid S_{\tau,0} = 0, X_\tau = x)}{\Pr(S_{\tau,0} = 0 \mid X_\tau = x)}. \tag{3.9}$$

Therefore, $\kappa_x = 1$ represents conditionally independent and statistically identical attempts, while $\kappa_x > 1$ denotes that the probability of exhausting the retransmission budget is larger than that predicted by the independent-attempt reference. Moreover, if $\mu_x = 1$, $f_x = 0$ and κ_x is undefined. In the finite-state model, f_x is used directly and, in the following, $f_x^{\text{ind}} = (1 - \mu_x)^2$ will define the independent-attempt reference.

Since $N_\tau \in \{1, 2\}$ for $\gamma = 1$, its conditional mean can be exactly computed as

$$m_x = \mathbb{E}\left[N_\tau \mid X_\tau = x\right] = 2 - \mu_x. \tag{3.10}$$

It is worth noticing that the equation above does not require the two decoding attempts to be independent. Moreover, in the finite-state model, the transistion probabilities and the BLER do not require conditional independence between the two attempts, as well.

As discussed before, the decoding window advances to the next position only when a successful CRC is observed, i.e., when $\sigma_\tau = 1$, or when the retransmission budget is exhausted. So, let us define

$$T_{xy}^{(1)} = \Pr(X_{\tau+1} = y \mid X_\tau = x, \sigma_\tau = 1), \tag{3.11}$$

$$T_{xy}^{(0)} = \Pr(X_{\tau+1} = y \mid X_\tau = x, \sigma_\tau = 0). \tag{3.12}$$

$$\tag{3.13}$$

Both $T_{xy}^{(1)}$ and $T_{xy}^{(0)}$ are $Q \times Q$ stochastic matrices whose (x, y) entry represents the probability of transitioning from state x at window position τ to state y at position $\tau + 1$, conditioned on the outcome of the decoding attempts at position τ and, in particular, the following holds:

$$\sum_{y=1}^{Q} T_{xy}^{(1)} = 1, \quad \sum_{y=1}^{Q} T_{xy}^{(0)} = 1, \quad \forall x \in [1, Q]. \tag{3.14}$$

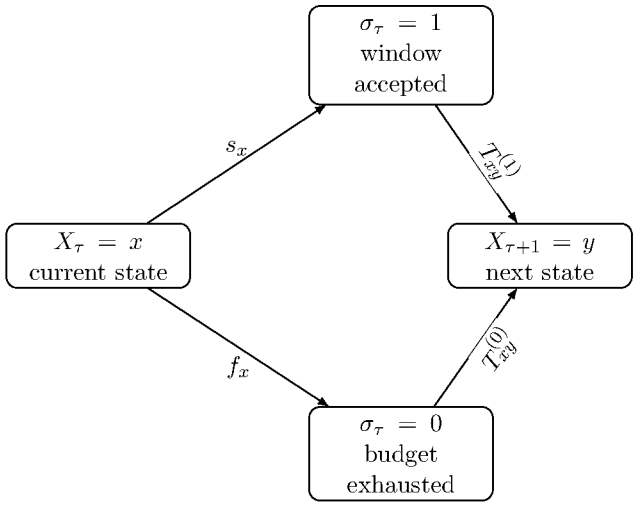


Figure 3.5: Finite-state model of CRC-based type-I HARQ with SWD. The hidden state X_τ determines the first-attempt success probability μ_x. The window is accepted with probability $s_x = 1 - f_x$, while the retransmission budget is exhausted with probability f_x. The next-state distribution follows $T^{(1)}$ or $T^{(0)}$, respectively.

By the law of total probability, the overall state transition probability from state x at position τ to state y at position $\tau + 1$ can be expressed as

$$\Pr(X_{\tau+1} = y \mid X_\tau = x) = s_x T_{xy}^{(1)} + f_x T_{xy}^{(0)}. \tag{3.15}$$

Therefore, defined $\mathbf{D}_s = \mathrm{diag}(s_1, s_2, \ldots, s_Q)$ and $\mathbf{D}_f = \mathrm{diag}(f_1, f_2, \ldots, f_Q)$, under this finite-state approximation, the transition matrix of $\{X_\tau\}$ is given by

$$\mathbf{P} = \mathbf{D}_s T^{(1)} + \mathbf{D}_f T^{(0)}. \tag{3.16}$$

Denoting $\boldsymbol{\pi}_\tau$ as the distribution of X_τ, the evolution of the state distribution across window positions satisfies

$$\boldsymbol{\pi}_{\tau+1} = \boldsymbol{\pi}_\tau \mathbf{P}. \tag{3.17}$$

The resulting model is summarized in Figure 3.5.

Observable states

As stated before, the state variable X_τ can be defined in several ways. In this work, we adopt a simple observable specification that is based on the history of retransmission budget exhaustion events. Define

$$U_\tau = 1 - \sigma_\tau, \tag{3.18}$$

such that $U_\tau = 1$ indicates that the retransmission budget has been exhausted at position τ, while $U_\tau = 0$ denotes that the target block has been accepted within

the available steps. Therefore, taking into account that a target block delivered after retransmission budget exhaustion remains among the read-only blocks for the following m_s, the state at position τ can be defined through

$$\mathbf{Z}_\tau = (U_{\tau-m_s}, \ldots, U_{\tau-1}) \in \{0,1\}^{m_s}. \tag{3.19}$$

For $\mathbf{Z}_0$, we set $U_\tau = 0$ for $\tau < 0$, which corresponds to the absence of any budget exhaustion event before the first window position. Each binary vector in $\{0,1\}^{m_s}$ is then associated with a value of X_τ. Therefore, the number of states is fixed as

$$Q = 2^{m_s}, \tag{3.20}$$

and, instead of being selected empirically, is determined by the code memory. Recalling that retransmitting the active window also replaces the channel observations of blocks that remain involved in up to $W-1$ subsequent windows, the resulting influence can extend beyond the m_s positions represented by $\mathbf{Z}_\tau$. Moreover, the state does not record whether an accepted window succeded at its first attempt or after a retransmission, nor the history of channel observations used in later overlapping windows. Thus, the choice of a binary history of m_s positions is motivated by the read-only span but is still an observable finite-state specification rather than a complete representation of the decoder state, that aims only to capture whether target blocks delivered after budget exhaustion are still present in the read-only buffer, and remains therefore a protocol-level approximation of the whole decoding state.

After processing the τ-th window, the state is then updated according to

$$\mathbf{Z}_{\tau+1} = (U_{\tau-m_s+1}, \ldots, U_{\tau-1}, 1-\sigma_\tau) \in \{0,1\}^{m_s}. \tag{3.21}$$

Then, conditioned on σ_τ, the next state is essentially deterministic, and the transition matrices $T^{(1)}$ and $T^{(0)}$ contain a single non-zero entry per row and do not require any numerical estimation.

For the code considered in Chapter 4, $m_s = 2$ and $Q = 4$. The states are then labeled as follows

$$1 \leftrightarrow (0,0), \quad 2 \leftrightarrow (0,1), \quad 3 \leftrightarrow (1,0), \quad 4 \leftrightarrow (1,1). \tag{3.22}$$

Whith this labeling, the transition matrices are given by

$$T^{(1)}_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad T^{(0)}_{xy} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.23}$$

and the initial state distribution is

$$\boldsymbol{\pi}_0 = (1, 0, 0, 0) \tag{3.24}$$

.

Given the quantity

$$\mu_{\min} = \min_{x \in [1,Q]} \mu_x, \tag{3.25}$$

if $\mu_{\min} > 0$, the following holds

$$\min_{x \in [1,Q]} s_x > 0. \tag{3.26}$$

Therefore, every state has a strictly positive probability that the current window is resolved within the retransmission budget. Furthermore, after m_s consecutive accepted windows, the state reaches $(0, \ldots, 0)$ independently of the initial state. This means that there are no absorbing states representing persistent budget exhaustion. It is important to remark that budget exhaustion, whose probability in state x is given by f_x, is the only detected mechanism through which an unresolved target block can enter the decoder history, while undetected CRC errors are not represented by the observable state since they yield to $U_\tau = 0$.

The state construction and transition model described above can be extended to an arbitrary retransmission budget. Given a maximum number of retransmissions γ, the conditional failure probability of attempt j can be expressed as

$$q_{x,j} = \Pr(S_{\tau,j} = 0 \mid X_\tau = x, S_{\tau,0} = \cdots = S_{\tau,j-1} = 0), \tag{3.27}$$

where $j \in [0, \gamma]$ and the conditioning on preceding attempts is omitted for $j = 0$, such that $q_{x,0} = 1 - \mu_x$. By the chain rule, the conditional probability of exhausting the retransmission budget is

$$f_x = \sum_{j=0}^{\gamma} q_{x,j}. \tag{3.28}$$

Considering that the attempt j is performed if and only if all preceding attempt have failed, the conditional mean number of attempts can be expressed as

$$m_x = 1 + \sum_{j=0}^{\gamma-1} \prod_{i=0}^{j} q_{x,i}. \tag{3.29}$$

If one assumes the attempts conditionally independent and stastistically identical given $X_\tau = x$, then $q_{x,j} = \mu_x$ for every j, and

$$f_x^{\text{ind}} = (1 - \mu_x)^{\gamma+1}, \qquad m_x^{\text{ind}} = \begin{cases} \dfrac{1 - (1 - \mu_x)^{\gamma+1}}{\mu_x}, & \mu_x > 0 \\ \gamma + 1, & \mu_x = 0. \end{cases} \tag{3.30}$$

In other terms, Equation (3.30) defines the independent-attempt reference for arbitrary γ. In the finite-state model, instead, the probabilities $q_{x,j}$ can be estimated individually and, in the same manner, if only state evolution, BLER, and average retransmission demand are required, f_x and m_x can be directly estimated as the state-conditioned budget exhaustion frequency and the sample mean number of attempts, respectively.

Note that the transition matrices $\mathbf{T}^{(1)}$ and $\mathbf{T}^{(0)}$, the state-transition equation (3.16), and the BLER expression derived later remain unchanged, since they depend only on whether the window is accepted or the transmission budget is exhausted. The specialization to $\gamma = 1$ used in the numerical validation has the advantage that

$$f_x = \kappa_x(1-\mu_x)^2, \qquad m_x = 2 - \mu_x,$$

where the expression for m_x is exact without any independence assumption. Conversely, for $\gamma > 1$ the intermediate conditional failure probability reported in (3.27), as well as the corresponding mean m_x, are additionally required to predict the retransmission overhead.

BLER estimation from the finite-state model

The finite-state model described above can also be used to predict the BLER at the output of the proposed type-I HARQ mechanism. In this scenario, it is necessary to distinguish retransmission budget exhaustion from an actual block error. In order to do so, note first that a correctly decoded CRC-protected block necessarily satisfies its CRC. Therefore, retransmission budget exhaustion, i.e., $\sigma_\tau = 0$, implies that the delivered target block is erroneous. For $s_x > 0$, define

$$\beta_x = \Pr(\text{block error} \mid \sigma_\tau = 1, X_\tau = x), \tag{3.31}$$

and set $\beta_x = 0$ when $s_x = 0$. Then, the conditional output BLER can be exactly expressed as

$$b_x = f_x + s_x\beta_x, \tag{3.32}$$

where the term f_x accounts for block delivered after retransmission budget exhaustion, while the second accounts for erroneous block accepted within the retransmission budget.

Note that detected failures followed by a successful retransmission do not contribute to the output BLER, even though they increase the overall number of decoding attempts. The average BLER over the regular portion of the coupled chain is therefore

$$\overline{\text{BLER}} = \frac{1}{L-W}\sum_{\tau=0}^{L-W-1}\sum_{x=1}^{Q}\pi_{\tau,x}b_x. \tag{3.33}$$

It is also common, in similar scenarios, to neglect undetected CRC errors. In this

work, for example, we employ a 16-bit CRC whose undetected error probability is known to be approximately 2^{-16} and to be independent of the lenght of the protected data [31]. Under this assumption, $\beta_x \approx 0$, and $b_x \approx f_x$, i.e., the conditional output BLER is dominated by the probability of retransmission budget exhaustion.

Parameter estimation and model validation

In the numerical validation presented below, we consider $\gamma = 1$. This validation is carried out with two separate Monte Carlo data sets: a training set, used only for parameter estimation, and a statistically independent validation set. Moreover, since the decoding success probabilities depend (also) on the operating channel conditions, parameter estimation is performed separately for each considered value of E_s/N_0[1].

At the final window position, the CRC is checked only for the first of the W remaining target blocks. Therefore, for the considered setting, i.e., $\gamma = 1$, its failure can trigger the retranmission of the final window, whereas the remaining $W-1$ blocks are delivered without individual CRC-based decisions. Since this terminal operation differs from what happens at the regular window positions, the corresponding blocks are excluded from the model-validation BLER and included only in the end-to-end results of Section 4.3. For this reason, the independently simulated BLER used to validate the model is computed only considering the window positions $\tau \in [0, L - W - 1]$.

For the parameter estimation, let n_x be the number of window visited in state x and r_x the number of those positions where the first decoding attempt fails. For the independent-attempt reference, we set

$$\widehat{\mu}_x = 1 - \frac{r_x}{n_x}, \qquad \widehat{f}_x^{\text{ind}} = (1 - \widehat{\mu}_x)^2. \tag{3.34}$$

For the finite-state model, the budget exhaustion probability is instead directly estimated from the training simulations. Denoting the number of positions at which the retransmission budget is exhausted as e_x, the corresponding estimate is

$$\widehat{f}_x = \frac{e_x}{n_x}. \tag{3.35}$$

For both the finite state model and the independent-attempt reference, $\widehat{m}_x = 2 - \widehat{\mu}_x$. For the finite-state model, $\widehat{s}_x = 1 - \widehat{f}_x$ and, when $\widehat{\mu}_x < 1$,

$$\widehat{\kappa}_x = \frac{\widehat{f}_x}{(1 - \widehat{\mu}_x)^2}. \tag{3.36}$$

Moreover, if $n_x = 0$, the state is never visited in the training simulation and is reported as unestimated, while we set $\widehat{\mu}_x = 1$ and $\widehat{f}_x = 0$, thus leading to $\widehat{m}_x = 1$. This convnetion does not appreciably effects the results since the predicted occupancy of

[1]For the sake of compactness, the dependence of the estimated quantities on E_s/N_0 is omitted.

such states is negligible. If $\widehat{\mu}_x = 1$, then $\widehat{f}_x = 0$ and $\widehat{\kappa}_x$ is not reported.

As already done above, define, for the finite-state model, the quantities

$$\widehat{\mathbf{D}}_f = \mathrm{diag}(\widehat{f}_1, \ldots, \widehat{f}_Q), \tag{3.37}$$

$$\widehat{\mathbf{D}}_s = \mathrm{diag}(1 - \widehat{f}_1, \ldots, 1 - \widehat{f}_Q), \tag{3.38}$$

such that

$$\widehat{\mathbf{P}} = \widehat{\mathbf{D}}_s \mathbf{T}^{(1)} + \widehat{\mathbf{D}}_f \mathbf{T}^{(0)}. \tag{3.39}$$

In the same way, define, for the independent-attempt reference, the quantities

$$\widehat{\mathbf{D}}_f^{\mathrm{ind}} = \mathrm{diag}(\widehat{f}_1^{\mathrm{ind}}, \ldots, \widehat{f}_Q^{\mathrm{ind}}), \tag{3.40}$$

$$\widehat{\mathbf{D}}_s^{\mathrm{ind}} = \mathrm{diag}(1 - \widehat{f}_1^{\mathrm{ind}}, \ldots, 1 - \widehat{f}_Q^{\mathrm{ind}}), \tag{3.41}$$

and

$$\widehat{\mathbf{P}}^{\mathrm{ind}} = \widehat{\mathbf{D}}_s^{\mathrm{ind}} \mathbf{T}^{(1)} + \widehat{\mathbf{D}}_f^{\mathrm{ind}} \mathbf{T}^{(0)}. \tag{3.42}$$

Therefore, starting from the same initial state distribution, the two state evolutions satisfy

$$\widehat{\boldsymbol{\pi}}_{\tau+1} = \widehat{\boldsymbol{\pi}}_\tau \widehat{\mathbf{P}}, \tag{3.43}$$

$$\widehat{\boldsymbol{\pi}}_{\tau+1}^{\mathrm{ind}} = \widehat{\boldsymbol{\pi}}\mathrm{ind}_\tau \widehat{\mathbf{P}}\mathrm{ind}, \tag{3.44}$$

where the independent-attempt reference uses $\widehat{\boldsymbol{\pi}}_\tau^{\mathrm{ind}}$ and $\widehat{f}_x^{\mathrm{ind}}$, whereas the finite state model uses $\widehat{\boldsymbol{\pi}}_\tau$ and $\widehat{f}_x$.

For the sake of clarity, Table 3.1 summarizes the quantities used in the finite-state model and how each of them is determined.

The state distribution predicted by the finite-state model $\widehat{\boldsymbol{\pi}}_\tau$ can also be used to predict the overall throughput, i.e., to predict the retransmission overhead introduced by the type-I HARQ mechansim. Note that, altough the conditional mean number of attempt depends solely on $\widehat{\mu}_x$, the average rertransmission count also depends on $\widehat{f}_x$ through the state occupancy. Nevertheless, for $\gamma = 1$, the expected number of retransmissions at a regular window position in state x can be expressed as

$$\widehat{m}_x - 1 = 1 - \widehat{\mu}_x. \tag{3.45}$$

Therefore, the expected number of retransmissions over the regular sliding portion of the coupled chain is

$$\widehat{N}_{retx}^{\mathrm{reg}} = \sum_{\tau=0}^{L-W-1} \sum_{x=1}^{Q} \widehat{\pi}_{\tau,x} (\widehat{m}_x - 1). \tag{3.46}$$

Then, let $\widehat{p}_{\mathrm{retx}}^{\mathrm{term}}$ be the probability, estimated separately from the training simulations, that the first decoding attempt for the final window fails. The estimated total

Table 3.1: Summary of the finite-state model quantities and their determination.

Quantity	Meaning	Determination
$S_{\tau,K}$, σ_τ, U_τ	Attempt-level CRC outcome, final window outcome, and budget exhaustion indicator	$S_{\tau,K} = 1$ when attempt K passes the CRC; $\sigma_\tau = 1$ if any performed attempt passes; and $U_\tau = 1 - \sigma_\tau$, so that $U_\tau = 1$ indicates budget exhaustion.
$\mathbf{Z}_\tau$, X_τ, Q	Observable state history, associated state, and number of states	$\mathbf{Z}_\tau = (U_{\tau-m_s}, \ldots, U_{\tau-1})$ and $Q = 2^{m_s}$; fixed by m_s and the state definition.
μ_x	First-attempt CRC success probability in state x	Estimated from the training simulations as $\widehat{\mu}_x = 1 - r_x/n_x$.
$q_{x,k}$	Failure probability of attempt k, conditioned on all preceding attempts having failed	For arbitrary γ, estimated from the outcomes of the positions reaching attempt k; under the independent-attempt reference, $q_{x,k} = 1 - \mu_x$.
f_x, s_x	Budget exhaustion and window acceptance probabilities	Independent-attempt reference: $f_x^{\text{ind}} = (1-\mu_x)^{\gamma+1}$. Finite-state model: $\widehat{f}_x = e_x/n_x$. In both cases, $s_x = 1 - f_x$.
κ_x	Measure of dependence between the two attempts for $\gamma = 1$	Derived as $\kappa_x = f_x/(1-\mu_x)^2$; it is not fitted or used independently.
m_x	Conditional mean number of attempts	For $\gamma = 1$, $m_x = 2 - \mu_x$ exactly. For arbitrary γ, $m_x = 1 + \sum_{k=0}^{\gamma-1} \prod_{j=0}^{k} q_{x,j}$.
$\mathbf{T}^{(1)}$, $\mathbf{T}^{(0)}$	State update matrices after acceptance and budget exhaustion	Determined analytically by shifting $\mathbf{Z}_\tau$ and appending 0 or 1; no estimation is required.
$\mathbf{D}_s$, $\mathbf{D}_f$, $\mathbf{P}$	Diagonal outcome matrices and overall transition matrix	$\mathbf{D}_s = \text{diag}(s_x)$, $\mathbf{D}_f = \text{diag}(f_x)$, and $\mathbf{P} = \mathbf{D}_s\mathbf{T}^{(1)} + \mathbf{D}_f\mathbf{T}^{(0)}$.
$\boldsymbol{\pi}_\tau$	State distribution at position τ	Initialized as $\boldsymbol{\pi}_0 = (1, 0, \ldots, 0)$ and propagated as $\boldsymbol{\pi}_{\tau+1} = \boldsymbol{\pi}_\tau\mathbf{P}$.
β_x, b_x	Error probability after acceptance and conditional output BLER	β_x is neglected when sufficiently small; $b_x = f_x + s_x\beta_x$.

number of retransmissions per frame can therefore be expressed as

$$\widehat{N}_{\text{retx}} = \widehat{N}_{\text{retx}}^{\text{reg}} + \widehat{p}_{\text{retx}}^{\text{term}}. \tag{3.47}$$

Substituting this expression into Equation (3.61) of the following section gives

$$\widehat{\mathcal{R}}_L^{\text{HARQ}} = \frac{L(a-c-r) - m_s c}{a(L + W\widehat{N}_{\text{retx}})}. \tag{3.48}$$

Consequently, the throughput estimate refers to the complete frame, whereas the BLER estimate refers only to the regular window positions.

The propagated state evolution provides an estimate of the average BLER, which is compared with the value measured on the validation set in Section 4.1.

3.3 Coding rate and throughput analysis

In this analysis, we assume that both target block decoding and feedback occur instantaneously and we consider the CRC-based detection technique for both adaptive resynchronization and type-I HARQ mechanism. Note that incorporating a fixed delay is feasible, but it would not offer any additional insights while complicating the expressions.

First of all, let us analyze the coding rate of an SC-LDPC code with a VNs and c CNs per temporal position, concatenated with an outer CRC encoder with r check symbols. Considering that, at each temporal position, the SC-LDPC code takes $a-c$ symbols as input, given a block of k_0 information bits, it must hold that $k_0+r=(a-c)$. This means that the r CRC bits are embedded within the SC-LDPC information block, thus reducing the number of useful information bits per position to $k_0=a-c-r$. Then, assuming a frame length of L blocks, the total number of transmitted bits is $n=aL$. Moreover, when $L\to\infty$, the number of parity-check constraints grows proportionally with L, and the rate can be expressed as

$$R_{\infty}^{\mathrm{CRC}}=\frac{a-c-r}{a}=R_{\infty}-\frac{r}{a}=\left(1-\frac{r}{k_0+r}\right)R_{\infty}. \tag{3.49}$$

For finite L, defining $k^{\mathrm{CRC}}\triangleq L(a-c-r)-m_s c$, the finite-length coding rate can be expressed as

$$R_{L}^{\mathrm{CRC}}=\frac{k^{\mathrm{CRC}}}{La}=R_{\infty}^{\mathrm{CRC}}-\frac{m_s c}{La}. \tag{3.50}$$

We define the *throughput* $\mathcal{R}$ as

$$\mathcal{R}=\frac{k}{\mathcal{N}} \tag{3.51}$$

where k and $\mathcal{N}$ are the total number of information symbols and of channel uses, respectively. Therefore, we will first discuss the effect of the mitigation techniques on the coding rate, and then we will focus on the overall throughput of the transmission. For later use, we also define the bit-level *goodput* as

$$\mathcal{G}=\frac{k-e_B^{(\mathrm{info})}}{\mathcal{N}}, \tag{3.52}$$

where $e_B^{(\mathrm{info})}$ is the number of incorrectly decoded information bits. Therefore, the payload of a block lost due to resynchronization or HARQ-budget exhaustion remains included in k for the throughput calculation, but contributes to $e_B^{(\mathrm{info})}$ in the goodput calculation.

According to the schedule described in Algorithm 3, after any resynchronization request triggered by an error detected in the V-th block, the transmitter must retransmit the $(W-1)a$ symbols corresponding to the temporal positions following the V-th block. This is because resynchronization fixes m_s blocks of information symbols, thereby altering the input to the SC-LDPC encoder from that point on-

ward. Therefore, all the encoded symbols beyond the V-th block are modified and must be retransmitted to maintain consistency with the resynchronized encoder. However, the transmitter only fixes to "0" symbols in blocks with indexes in $[\max\{0,(V-m_s+1)\},V]$ that were already decoded. This essentially implies that the resynchronization process does not alter the number of information symbols within each decoding window. In other terms, adaptive resyncrhonization does not affect the coding rate of the SC-LDPC code. For the sake of comparison, in adaptive VN doping, the symbols to be fixed to "0" were never part of previous target blocks. In fact, if an adaptive doping request is triggered by an error detected in the V-th block, the transmitter fixes to "0" symbols in the $(V+W)$-th block. This scheduling strategy leads to an improvement in the throughput as it avoids retransmissions, but it also introduces a rate loss (since the $(V+W)$-th block is doped and therefore cannot carry information symbols), and degrades the BER and BLER.

In accordance with the above discussion, the number of information symbols and the length of the transmitted frame remain unchanged, i.e, the following holds:

$$n^{\text{resync.}} = La, \quad k^{\text{resync.}} = La - (L+m_s)c. \tag{3.53}$$

and the coding rate is then given by

$$R_L^{\text{resync.}} = \frac{k^{\text{resync.}}}{n^{\text{resync.}}} = R_L. \tag{3.54}$$

However, if ρ resynchronization requests occur during a transmission, the total number of channel uses $\mathcal{N}^{\text{resync.}}$ increases as

$$\mathcal{N}^{\text{resync.}} = n^{\text{resync.}} + \rho(W-1)a = La + \rho(W-1)a, \tag{3.55}$$

and the throughput therefore becomes

$$\mathcal{R}_L^{\text{resync.}} = \frac{k^{\text{resync.}}}{\mathcal{N}^{\text{resync.}}} = \frac{La-(L+m_s)c}{a[L+\rho(W-1)]}. \tag{3.56}$$

Considering both outer CRC encoding and adaptive resynchronization as error detection and mitigation techniques, r CRC bits are embedded within each block of $(a-c)$ information symbols, leading to a total number of information bits of

$$k^{\text{CRC+resync.}} = L(a-c-r) - m_s c. \tag{3.57}$$

The total number of channel uses, given ρ resynchronization requests, remains as

$$\mathcal{N}^{\text{CRC+resync.}} = La + \rho(W-a)a. \tag{3.58}$$

Thus, the finite-length throughput, combining both CRC encoding and adaptive

resynchronization, is

$$\mathcal{R}_L^{\text{CRC+resync.}} = \frac{L(a-c-r)-m_s c}{a[L+\rho(W-1)]}. \tag{3.59}$$

Note that this expression takes into account the rate loss due to the CRC overhead and retransmissions due to resynchronization, while the coding rate remains $R_L^{\text{CRC+resync.}} = R_L^{\text{CRC}}$, and does not include the cost of retransmissions.

Let us now analyze the effects of the type-I HARQ protocol described in Algorithm 4 on the coding rate and throughput of the overall system. As already discussed, the coding rate of a terminated SC-LDPC code concatenated with an outer CRC encoder is given by (3.50). The employment of HARQ does not alter the overall number of information symbols, i.e., it does not change the coding rate, but it penalizes the throughput, since certain blocks are retransmitted upon the detection of an error. In particular, denoting N_{retx} as the total number of requested retransmissions within a frame, the total number of channel uses increases as

$$\mathcal{N}^{\text{HARQ}} = La + N_{\text{retx}} W a, \tag{3.60}$$

and the throughput is then given by

$$\mathcal{R}_L^{\text{HARQ}} = \frac{k^{\text{CRC}}}{\mathcal{N}^{\text{HARQ}}} = \frac{L(a-c-r)-m_s c}{a(L+N_{\text{retx}}W)}. \tag{3.61}$$

Chapter 4

Numerical results

In this chapter, we first validate the finite-state model developed in Section 3.2.2, then we validate the effectiveness of both the proposed error detection and error mitigation strategies by presenting numerical results obtained through simulations.

In particular, in the first section, we analyze the retransmission-budget exhaustion probability $\widehat{f}_x$, and then we use the estimated transition matrix $\widehat{\mathbf{P}}$ to predict the output BLER by applying (3.33).

Then, in the second section, we focus on the effectiveness of the error detection mechanisms, comparing the proposed schemes to the LLR-averaging method.

In the third section, we study the impact of the proposed error mitigation strategies (paired with the aforementioned detection techniques) on BER and BLER, considering both AWGN and Rayleigh channels. Afterward, the trade-off between error rate performance and throughput (and goodput) is evaluated.

First, we considered a protograph-based TV code, $\mathcal{C}$, drawn uniformly at random from the $(3, 6)$ regular ensemble in Example 1, with $J' = 1$, $K' = 2$, $M = 200$ and $L = 500$. This choice is based on these codes' ability to offer performance close to the channel capacity [18]. The resulting code has $R_\infty = 1/2$, $a = 400$, $c = 200$, $m_s = 2$ and girth $g = 4$, from which we can note that $\mathcal{C}$ was not optimized to reduce the number of short cycles in its Tanner graph. The window size was set to $W = 5(m_s + 1) = 15$ blocks. Moreover, for the detection techniques that involve a CRC, we employed the CRC-CCITT code, a 16-bit code defined by the generator polynomial $G(x) = x^{16} + x^{12} + x^5 + 1$ that is widely used in many standards (e.g., Bluetooth, X.25, HDLC, and many others).

Moreover, it is important to remark that, even though the most common approach to compare codes is to plot their performance versus the SNR per information symbol E_b/N_0 (where E_b is the energy per information symbol and N_0 is the noise spectral power), when codes with different rates are being compared it is often preferred to plot the performance as functions of E_s/N_0, where $E_s = RE_b$ is the energy per transmitted symbol in order to remove the dependence on the code rate. Since the considered mitigation strategies result in different rates (due to the concatenation of a CRC code or the use of VN doping as a benchmark), we will express the numerical results in terms of E_s/N_0 [dB]. Note that the corresponding E_b/N_0 [dB] can be

obtained through the simple relation

$$\frac{E_b}{N_0}\ [\mathrm{dB}] = \frac{E_s}{N_0}\ [\mathrm{dB}] - 10\log_{10}(R) \tag{4.1}$$

4.1 Model validation

Taking into account the state labeling introduced in Section 3.2.2, Figure 4.1 shows the measured state dependent budget exhaustion probabilities $\widehat{f}_x$. As can be gathered from the Figure, State 1 dominates the occupancy and $\widehat{\kappa}_1$ ranges from about 2.1 to 3.2. This implies that the two attempts performed at the same window position fail together considerably more often then predicted by the independent-attempt reference, thus highlighting that X_τ does not fully describe the deocder conditions shared by the two attempts (e.g., the actual read-only information inherited by the current window).

It is worth noticing that the ordered state history also keeps information that would be lost retaining only the number of recent budget exhaustion events. States 2 and 3 have nearly identical average occupancies but, although both contain one exhaustion in the preceding two positions, the exhaustion occurred at $\tau - 1$ in State 2 and $\tau - 2$ in State 3. In the lower region of the SNR range, where both states are visited more frequently, State 2 exhibits a higher estimated first-attempt failure probability than State 3. Moreover, State 3 can also have a lower failure probability than State 1, since a retransmission at position $\tau - 2$ also refreshed observations that are used at position $\tau - 1$ and τ. Therefore, conditioning on acceptance at $\tau - 1$ may select cases where these observations are favorable, thus providing an explanation for the lower failure probability observed in State 3.

In all the considered simulations, the number of windows accepted despite delivering an erroneous target block is negligible: at $E_s/N_0 = -2.5$ dB they account for less than 1% of the erroneous blocks, and no such event is observed at $E_s/N_0 \geq -2.1$ dB. Therefore, using $\beta_x \approx 0$, and consequently $b_x \approx f_x$, is a justified approximation.

For each state x, the average state probability over the regular window positions can be defined as

$$\overline{\pi}_x = \frac{1}{L-W} \sum_{\tau=0}^{L-W-1} \pi_{\tau,x}. \tag{4.2}$$

In order to verify that the observable state provides an adeguate description of the decoding history, Table 4.1 compares the average state occupancy predicted by the finite-state model with the one measured in the simulations. At all the considered points, the predicted and measured occupancies agree to within $2 \cdot 10^{-4}$ in absolute terms and, for nonzero entries, to within about 1% in relative terms, over occupancies spanning more than five orders of magnitude. Moreover, the predicted average occupancies of States 2 and 3 are nearly identical. At $E_s/N_0 = -2.5$ dB, States 2 and 4 account for only about 9.3% of the window positions, while contributing to approximately 32% of the predicted block errors. Therefore, budget exhaustion

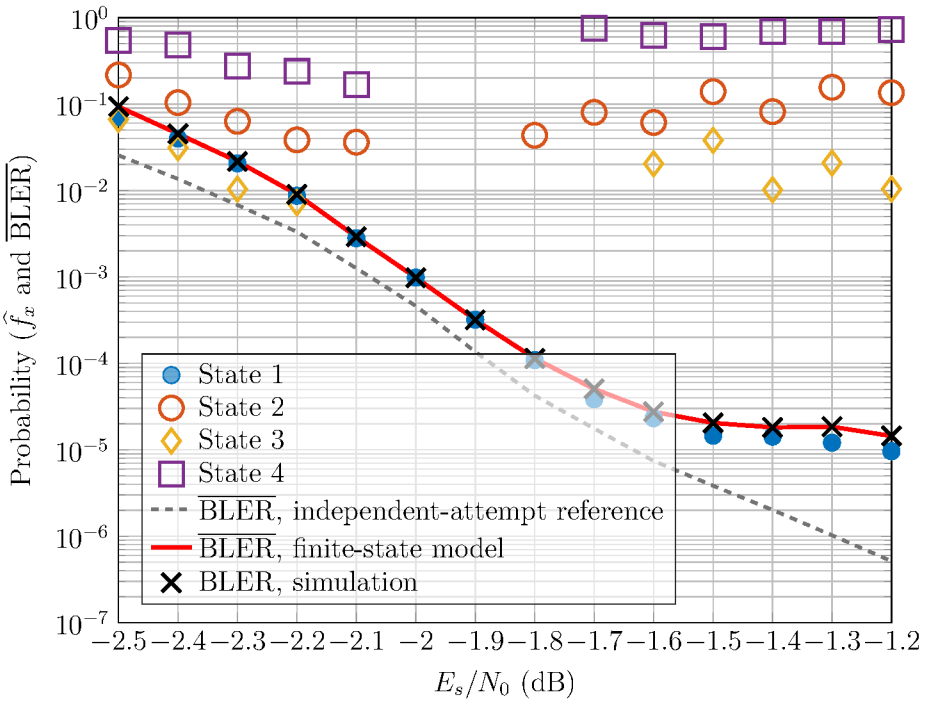


Figure 4.1: State dependent retransmission budget exhaustion probabilities and average BLER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with CRC-triggered HARQ, $F = 1$, $\gamma = 1$. The average BLER is computed over the regular window positions $\tau \in [0, L - W - 1]$.

is concentrated in the rare states that record an immediately preceding exhaustion rather than being uniformly distributed along the coupled chain.

Figure 4.1 also compares the BLER prediction obtained with the finite-state model and the independent-attempt reference with the actual value measured over the corresponding window positions in the intedependent validation simulations. As the Figure shows, the independent-attempt reference understimates the simulated BLER by a factor ranging from about 2.1 to 3.7, with the largest discrepancy at the lowest considered SNR. Then, the finite-state model propagates $\widehat{\mathbf{P}}$ using the state dependent budget exhaustion probabilities $\widehat{f}_x$ obtained from the training simulations, and it agrees with the independently simulated BLER to within about 0.2% with a discrepancy below 0.05% above all points except $E_s/N_0 = -1.7$ dB. Note that, since $\widehat{f}_x$ is estimated from the training set, this comparison verifies that the estimated parameters transfer to an independent simulation set and that the state occupancies predicted through $\widehat{\mathbf{P}}$ correctly combine them. In conclusion, the occupancy comparison in Table 4.1 and the throughput prediction provide separate validations of the state evolutions and retransmission demand.

Table 4.1: Predicted and measured average state probabilities.

E_s/N_0 [dB]		$\overline{\pi}_1$	$\overline{\pi}_2$	$\overline{\pi}_3$	$\overline{\pi}_4$
-2.5	Pred.	0.8437	0.0632	0.0631	0.0300
	Meas.	0.8439	0.0632	0.0630	0.0299
-2.0	Pred.	0.9980	$9.818{\cdot}10^{-4}$	$9.818{\cdot}10^{-4}$	0
	Meas.	0.9980	$9.798{\cdot}10^{-4}$	$9.778{\cdot}10^{-4}$	0
-1.2	Pred.	1.0000	$9.724{\cdot}10^{-6}$	$9.724{\cdot}10^{-6}$	$4.761{\cdot}10^{-6}$
	Meas.	1.0000	$9.703{\cdot}10^{-6}$	$9.673{\cdot}10^{-6}$	$4.715{\cdot}10^{-6}$

4.2 Error detection effectiveness

We now validate the effectiveness of the proposed iteration-based and CRC-based error detection methods. For comparison, we also present results for the resynchronization strategy paired with the LLR-averaging method reported in [25], [27]. For these evaluations, we performed Monte Carlo simulations of BPSK-modulated transmissions over the AWGN channel, considering the proposed sliding window decoders with $I_{\max} = 100$ and clipping value $\ell = 100$, therefore saturating all the LLRs to the interval $[-\ell, \ell]$ and assigning $\pm\ell$ to read-only symbols used for resynchronization, according to their known value.

In order to formalize the evaluation, the problem is modeled as a binary decision process, where the negative class corresponds to cases where mitigation, i.e., resynchronization or retransmission, is not required, whereas the positive class corresponds to cases where a mitigation request is necessary. Accordingly, four possible outcomes can be distinguished:

- True Negatives (TNs): decoding of the target block was correct and mitigation was correctly not triggered;
- False Positives (FPs): decoding was correct but mitigation was unnecessarily triggered;
- False Negatives (FNs): decoding was not correct but mitigation was not triggered, allowing error propagation;
- True Positives (TPs): decoding was not correct and mitigation was correctly triggered.

Considering that simulations are run until 100 wrongly decoded frames are observed for each value of E_s/N_0, the number of observations per outcome can vary considerably, leading therefore to an imbalance in the dataset, that may be dominated by a single outcome. For this reason, we adopted the *geometric mean* $\bar{G}$ as the performance metric, where

$$\bar{G} = \sqrt{\mathrm{TPR} \cdot \mathrm{TNR}}, \quad \mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \quad \mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}.$$

We also have

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \quad \text{and} \quad \mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}}.$$

Regarding the LLR-averaging method, three different values of the LLR threshold $\eta \in \{1, 10, 50\}$ were simulated while, for the iteration-based method, the threshold J was set to $I_{\max}$ as in [10].

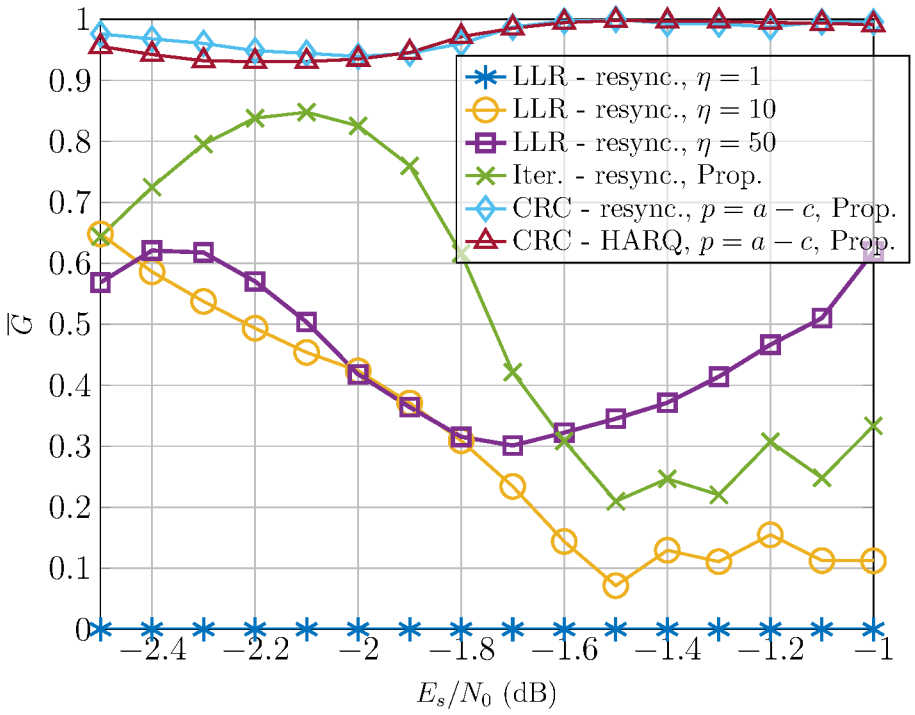


Figure 4.2: $\bar{G}$ vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods.

Figure 4.2 shows a comparison for code $\mathcal{C}$ between the LLR-based and the proposed iteration-based and CRC-based detection methods, labeled as "LLR -", "Iter. -" and "CRC -" in the figures, respectively. In addition to the detection method, the paired mitigation strategy ("resync." and "HARQ" for resynchronization and type-I HARQ, respectively), is also reported in the figures since it affects the residual error distribution at the decoder input, thus impacting the evaluation of the subsequent decoding stages.

From Figure 4.2, we observe that the average LLR-based method does not yield good results if the value of the threshold is not optimized. For instance, when $\eta = 1$, this method never detects errors, since the threshold is too low. While increasing η (e.g., setting $\eta = 10$ or $\eta = 50$) improves $\bar{G}$, it is clear that the optimal value of the threshold heavily depends on the channel conditions, i.e., on the value of E_s/N_0, with larger values of η performing better at higher SNRs. The iteration-based method typically yields better detection performance than the LLR-based method, even though the latter requires more computational effort. Moreover, the iteration-based

method does not require to tune any threshold since setting $J = I_{\max}$ already yields better detection performance in the low-to-medium SNR regime compared to LLR-based method. Finally, we note that the CRC-based schemes achieve the best error-detection performance, outperforming the empirical methods over the entire range of SNRs, as expected.

For the sake of completeness, in Figure 4.3, we plot the TPR (true detection rate) versus the FPR (false alarm rate), varying E_s/N_0.

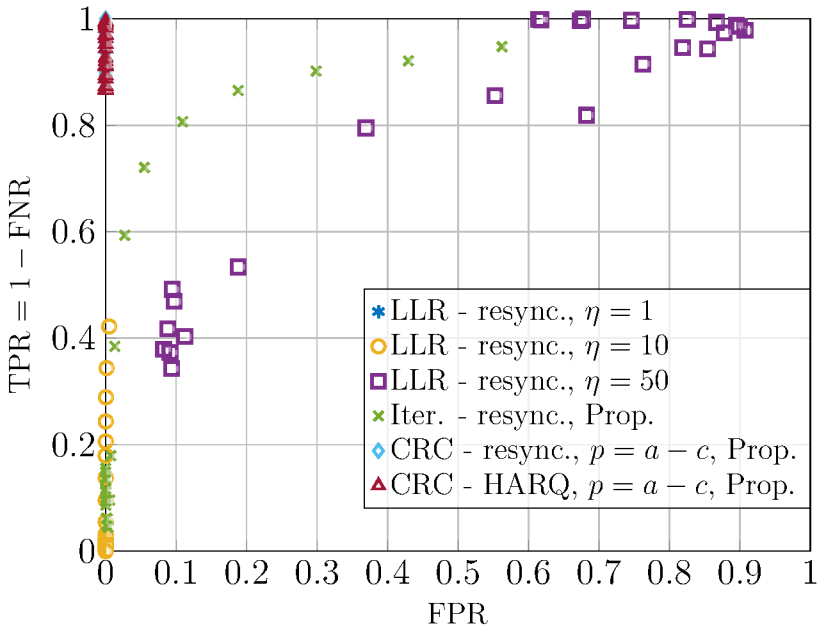


Figure 4.3: TPR (detection rate) vs FPR (false alarm rate) for $\mathcal{C}$, with $L = 500$, for different values of $\frac{E_s}{N_0}$, under SWD with different detection methods.

As the Figure shows, the CRC-based schemes almost behave as ideal error detectors since they cluster near the top-left region, essentially operating at zero FPR while maintaining TPR ≈ 1 over the entire SNR range. The dependence on the value of η for the average LLR-based method is also clearly illustrated in Figure 4.3. For instance, with small threshold ($\eta = 1$), the points remain close to the origin, denoting many undetected errors. Even with intermediate threshold ($\eta = 10$), a significant number of errors are still not detected in the SNR region of interest since the TPR remains limited. With large threshold ($\eta = 50$), a high TPR is reached only at the expense of a very large FPR (corresponding to the low SNR regime). In other words, in this scenario many true positives are obtained at the cost of frequent false rejections. Finally, the iteration-based detector normally operates in the upper region of the plot while maintaining a modest FPR; however, as the SNR increases, both the TPR and FPR decrease. This happens because, in that SNR region, the number of erroneous blocks diminishes and the dataset becomes increasingly imbalanced.

4.3 Error rate performance

Before analyzing the BER and BLER performance of $\mathcal{C}$, it is important to emphasize that, since SC-LDPC codes are LDPC convolutional codes, the Frame Error Rate (FER) does not represent the actual decoding reliability. Indeed, when very long frames are considered, as in the present work, the FER essentially approaches 1 even if the BER and BLER are extremely low. Moreover, taking into account that the SWD retrieves information progressively rather than at the end of the entire frame, i.e., a target symbols are decided at each window position, it is clear that the local reliability metrics are more meaningful than the nominal FER. In the follwing, the reported BER and BLER are computed over the SC-LDPC systematic symbols, thus excluding parity symbols. Therefore, for the CRC-based schemes, they are evaluated over all the $a - c$ systematic symbols per spatial position, including the $a - c - r$ payload symbols and the r CRC symbols. Goodput, instead, counts only the $a - c - r$ payload information symbols, excluding both parity and CRC symbols.

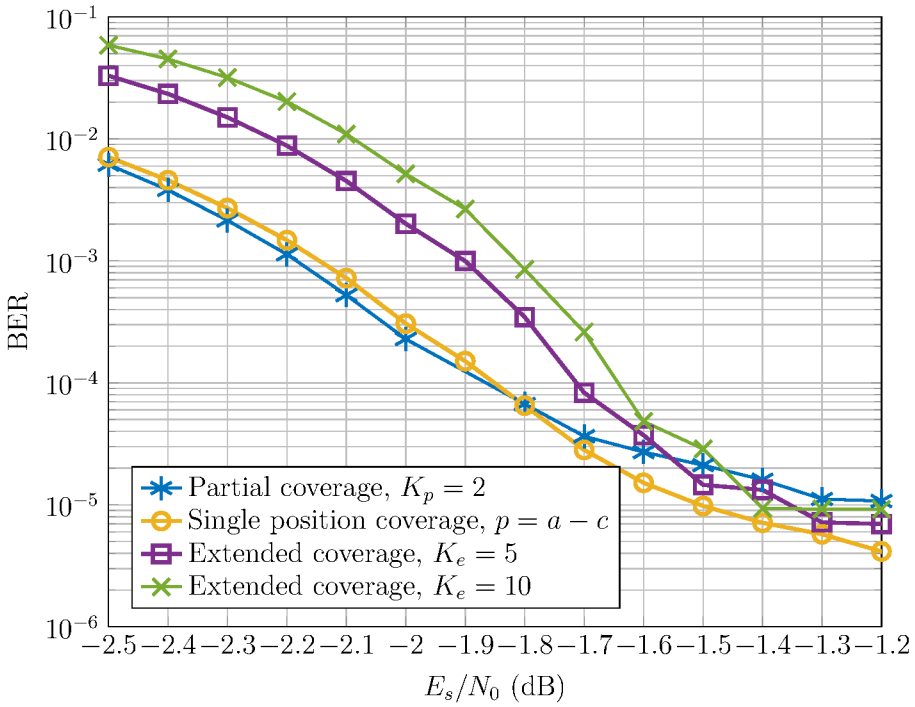


Figure 4.4: BER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with CRC-triggered resynchronization and different coverage strategies.

Figure 4.4 shows the performance of code $\mathcal{C}$ decoded with Algorithm 3, paired with CRC-based detection techniques implemented with different CRC coverage strategies. In particular, we considered partial coverage with $K_p = 2$, single-position coverage and extended coverage with $K_e = 5$ and $K_e = 10$ on the AWGN channel. In the low-to-medium SNR regime the single-position coverage turns out to be the most effective since it provides the best performance and introduces a lower rate

loss compared with the partial coverage strategy. However, for $E_s/N_0 > -1.7$ [dB], all the considered coverage strategies lead to comparable performance and therefore the extended coverage with $K_e = 10$ may be more suitable because it is the coverage strategy that introduces the lowest rate loss among those examined.

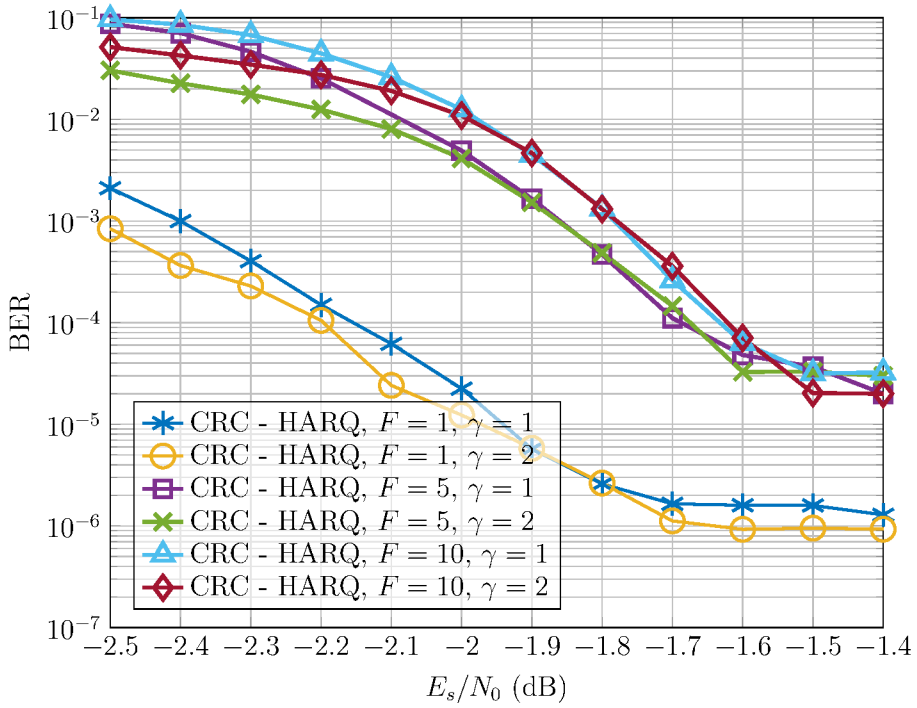


Figure 4.5: BER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with CRC-triggered HARQ and different values of F and γ.

Figure 4.5 illustrates the performance of code $\mathcal{C}$ decoded using Algorithm 4 by varying values of γ (the maximum number of allowed retransmissions per window) and F (the number of consecutive CRC failures required to trigger an HARQ request). As expected, bigger values of F degrade the performance since they reduce the number of overall retransmissions. Moreover, when $F = 1$ is employed, i.e., when retransmissions are requested after each CRC failure, a single retransmission per window (in other words, setting $\gamma = 1$) is often sufficient to achieve good results while maintaining a higher throughput than $\gamma = 2$.

In order to provide a detailed comparison between the proposed strategies and other approaches, Figures 4.6 and 4.7 present the BER and BLER achieved with $\mathcal{C}$ using the following detection and mitigation strategies:

- baseline SWD without retransmission or resynchronization (Algorithm 2) [22];
- periodic resynchronization with period $f = 10$, where the transmitter performs resynchronization every f blocks without requiring a feedback channel. This strategy is similar to periodic VN doping [28], but we periodically fix m_s blocks

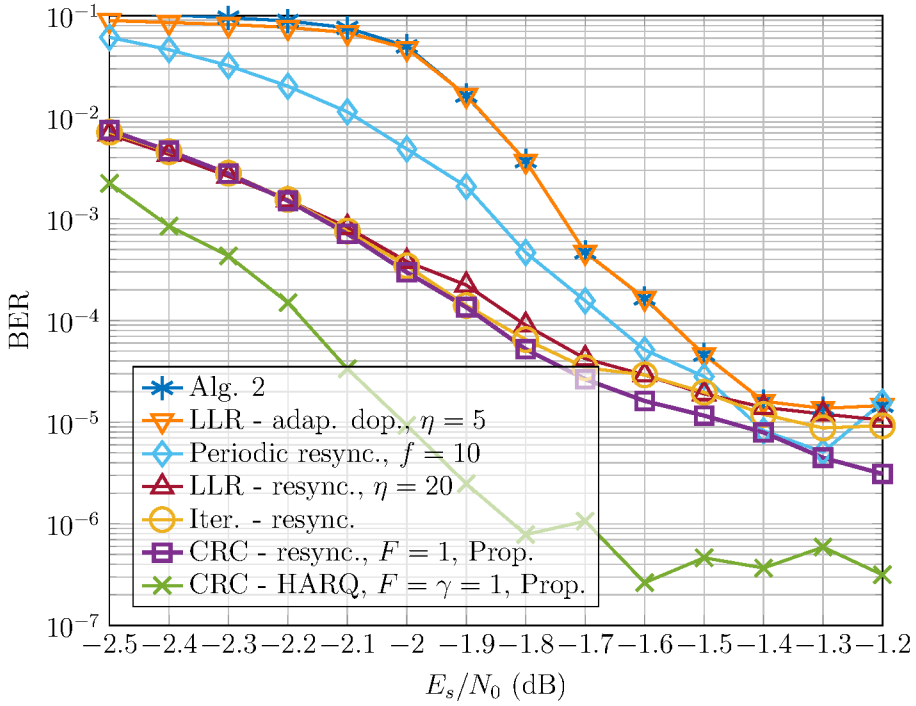


Figure 4.6: BER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies.

of symbols (equivalent to encoder termination), while [28] fixes a single block (corresponding to partial termination);

- LLR-based adaptive VN doping (labeled as "LLR - adap. dop.") with LLR threshold $\eta = 5$ [25];
- LLR-based adaptive resynchronization (labeled as "LLR - resync.") with $\eta = 20$[1]. In this case, the error detection method is based on an existing approach [25], [28], but the proposed error mitigation strategy is novel;
- iteration-triggered adaptive resynchronization (labeled as "Iter. - resync."), with the iteration threshold set to $I_{\max}$ (Algorithm 3) [10];
- CRC-based adaptive resynchronization (labeled as "CRC - resync.") with $F = 1$ (Algorithm 3);
- CRC-based adaptive type-I HARQ (labeled as "CRC - HARQ") with $F = 1$, $\gamma = 1$ (Algorithm 4).

The baseline curve (blue star) corresponds to Algorithm 2 without any retransmission nor resynchronization. The adaptive VN doping curve (orange down triangle)

[1]To maintain clarity and avoid clutter in the figures, only one representative curve is shown, corresponding to a value of η that offers a good balance between performance and throughput.

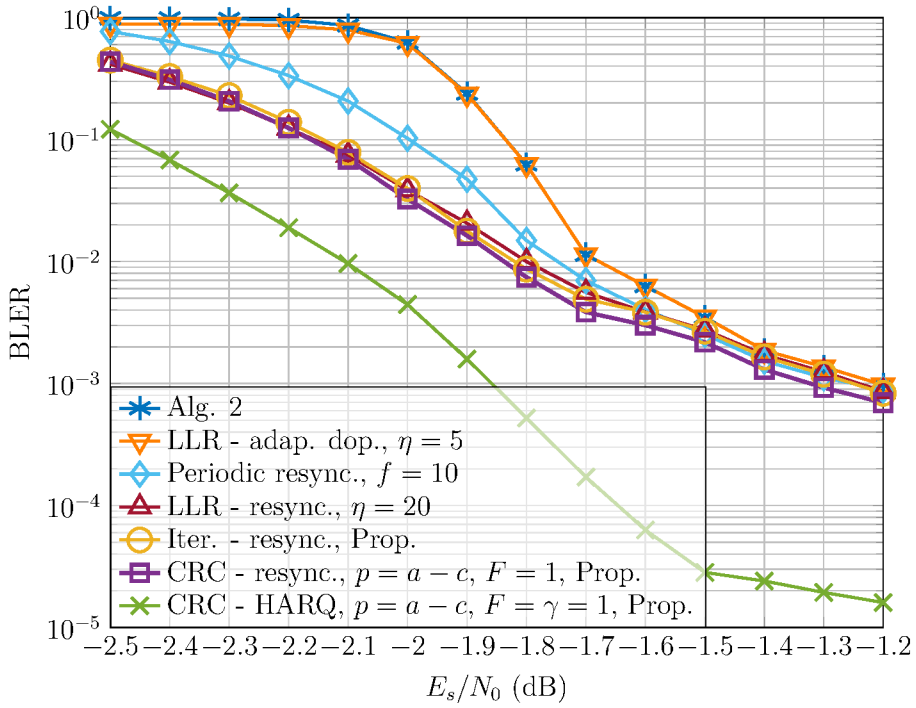


Figure 4.7: BLER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies.

achieves a slight gain over the baseline at the cost of a small rate penalty, thus exhibiting a behaviour consistent with the discussion in Section 3.2.1. A further improvement in performance is obtained with periodic resynchronization (cyan diamond curve), that does not require a feedback channel but introduces a larger rate penalty since f is relatively small in this case. The performance associated with LLR-based detection (red up triangle curve) and iteration-based detection (yellow circle curve) paired with encoder resynchronization shows additional improvement in the whole SNR range with respect to the baseline and the periodic resynchronization strategy. However, it is important to recall that, in order to obtain the results represented by the red curve, it was necessary to run several simulations to optimize the value of η, while the results shown in the yellow curve were obtained directly by setting $J = I_{\max}$. The performance of the purple curve (Algorithm 3 paired with CRC-based detection), which introduces CRC checks on individual blocks, denotes additional improvement in the mid-to-high SNR region thanks to the better detection performance given by the CRC, that allows the decoder to send more precise mitigation requests. Finally, the green curve (Algorithm 4), which incorporates a feedback-based HARQ strategy that uses CRC failures to trigger selective retransmissions, achieves the best overall performance. Indeed, Figure 4.6 clearly shows that Algorithm 4 achieves a BER of approximately 10^{-5}, i.e., the BER of the SWD baseline's floor region, at $E_s/N_0 \approx -2$ dB, resulting in an improvement of ≈ 0.5 dB

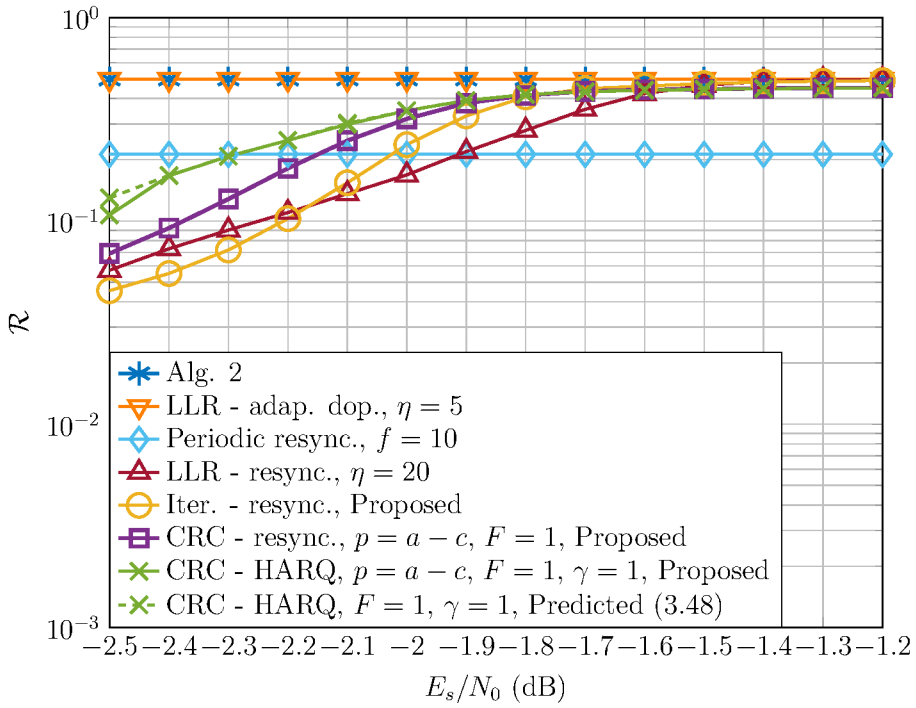


Figure 4.8: $\mathcal{R}$ vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies.

with respect to both blue and orange curves. Moreover, in the high SNR region, i.e., in the region where $\mathcal{C}$ exhibits error floor, Algorithm 4 yields a BER of $\approx 6 \cdot 10^{-7}$, while the standard SWD stops at $\approx 3 \cdot 10^{-5}$ and the other approaches do not reach any BER lower than $\approx 3 \cdot 10^{-6}$. In other words, when channel conditions are sufficiently favorable, the proposed type-I HARQ mechanism yields BER gains of several orders of magnitude.

The aforementioned gains achieved with Algorithm 4 are even more pronounced in Figure 4.7. These results can be justified by the fact that, by retransmitting the window that includes the (suspected) erroneous target block, the type-I HARQ based mechanism is the only one, among those considered, that can correct the *current* target block, therefore directly impacting the BLER.

Figure 4.8 shows the throughput $\mathcal{R}$ achieved by code $\mathcal{C}$ under the discussed mitigation strategies. Moreover, we also report in Figure 4.9 the *goodput* $\mathcal{G}$, defined as the number of successfully delivered bits divided by the total number of channel uses $\mathcal{N}$. As expected, each of the proposed adaptive schemes yields significant BER and BLER improvements at the cost of some throughput and goodput loss, especially at low SNRs. We also note that for $E_s/N_0 > -1.5$ dB all mitigation strategies converge to comparable throughput and goodput. However, we also observe that the CRC-based HARQ strategy has the best error performance and the smallest loss in terms of throughput and goodput among the proposed scheme and, although

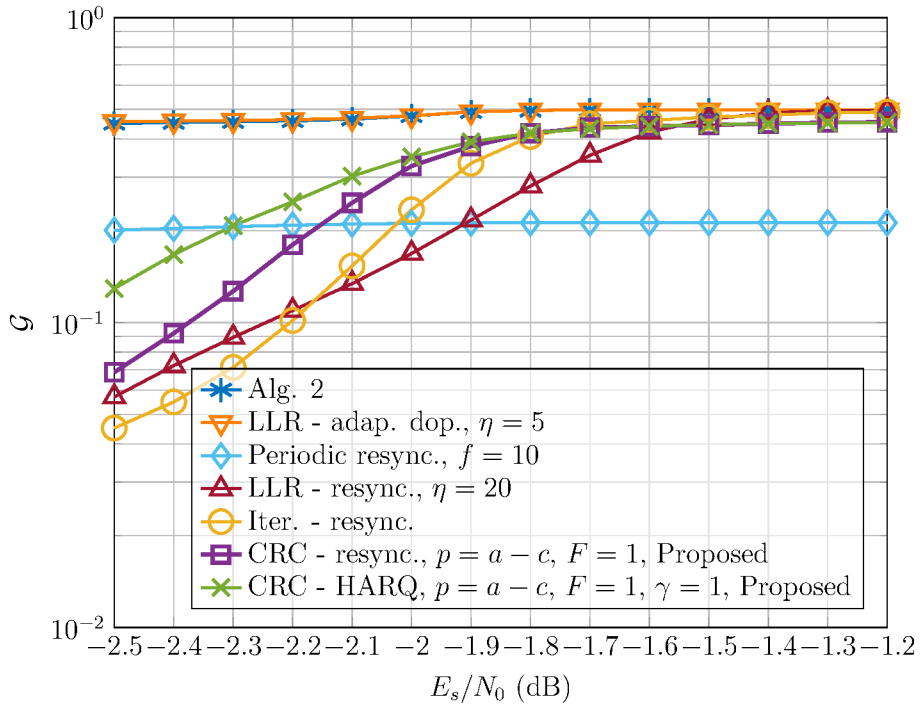


Figure 4.9: $\mathcal{G}$ (bit-level goodput) vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies.

LLR-based adaptive doping has better throughput and goodput, it has much worse error rate performance as depicted in Figures 4.6, 4.7.

Figure 4.8 also compares the simulated throughput of the proposed CRC-based HARQ strategy with the model estimate obtained from (3.48). As depicted in the Figure, the prediction closely matches the simulation over the entire considered range, with an absolute discrepancy below $2.3 \cdot 10^{-3}$.

In order to further validate the effectiveness of the proposed mitigation techniques, we also analyzed the error rate performance achieved on a Rayleigh channel, since it allows for the modelling of propagation scenarios that differ from those covered by the AWGN model, such as tropospheric and ionospheric signal propagation and urban environments. Recalling that the signal received from a Rayleigh channel can be represented by Equation (2.6), we have assumed that the fading coherence time of the channel, i.e., the time interval during which the coefficients α do not vary, is constant and an integer multiple of the symbol time. Therefore, denoting T_C as the fading coherence time, and T_S as the symbol time, $T_C = KT_S$ and, in terms of coefficients, $\alpha_j = \alpha_{\lfloor \frac{j}{K} \rfloor}$.

Figures 4.10, 4.11 report the BER and BLER achieved by code $\mathcal{C}$ under different decoding schedules, in a *fast fading* scenario, i.e., one in which the fading coherence time is relatively low ($T_C = 5T_S$). Comparing these results with those shown in Figures 4.6, 4.7, it is clear that the Rayleigh channel degrades the signal much

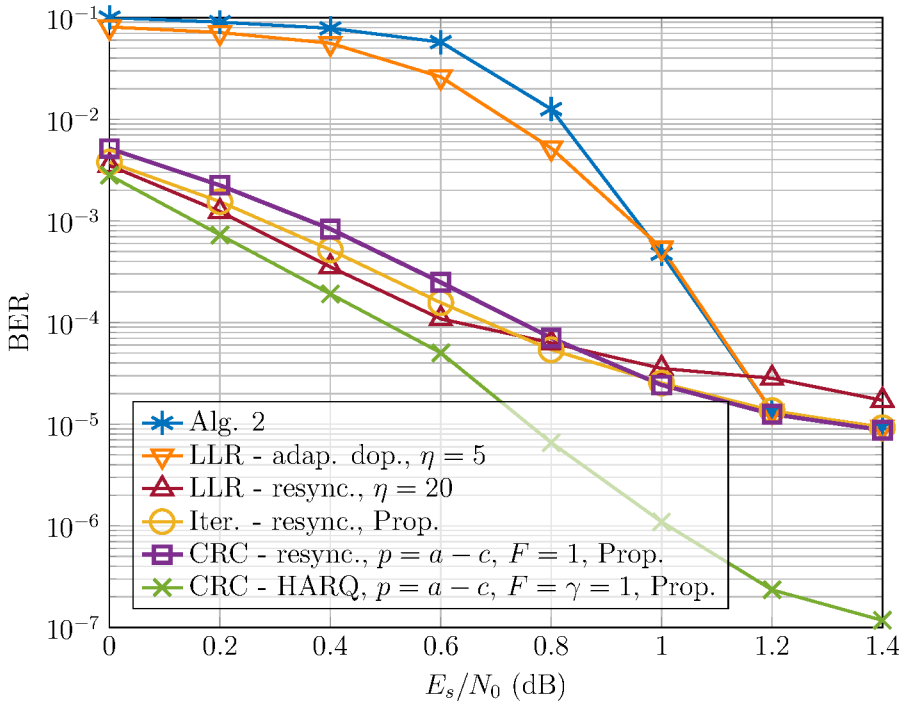


Figure 4.10: BER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies, on Rayleigh channel with $K = 5$.

more, causing the curves to shift to the right. In any case, however, the trend in the results obtained follows those illustrated earlier, since all the proposed techniques introduce improvements with respect to the baseline in the low-to-mid SNR range, while the CRC-based HARQ mechanism yields BER and BLER gains of several orders of magnitude even at higher SNRs.

We also analyzed the impact on the performance of a Rayleigh channel with a higher fading coherence time. In fact, Figures 4.12, 4.13 show the BER and BLER achieved by code $\mathcal{C}$ in a *slow fading* scenario, in which we assumed $T_C = 50T_S$. In this case, resynchronization and retransmission yield similar BER and BLER gains over the baseline SWD and VN doping. This scenario is more detrimental than the previous one because, under *deep fading*, i.e., when the signal is heavily degraded and the coefficients α_j have low magnitude, more consecutive symbols are affected (since K is bigger), thus introducing large clusters of unreliable LLRs that are more difficult to mitigate. Still, the proposed mitigation techniques are able to break the error chain, leading to a gain of ≈ 0.8 dB and ≈ 0.4 dB in the BER and BLER curves, respectively.

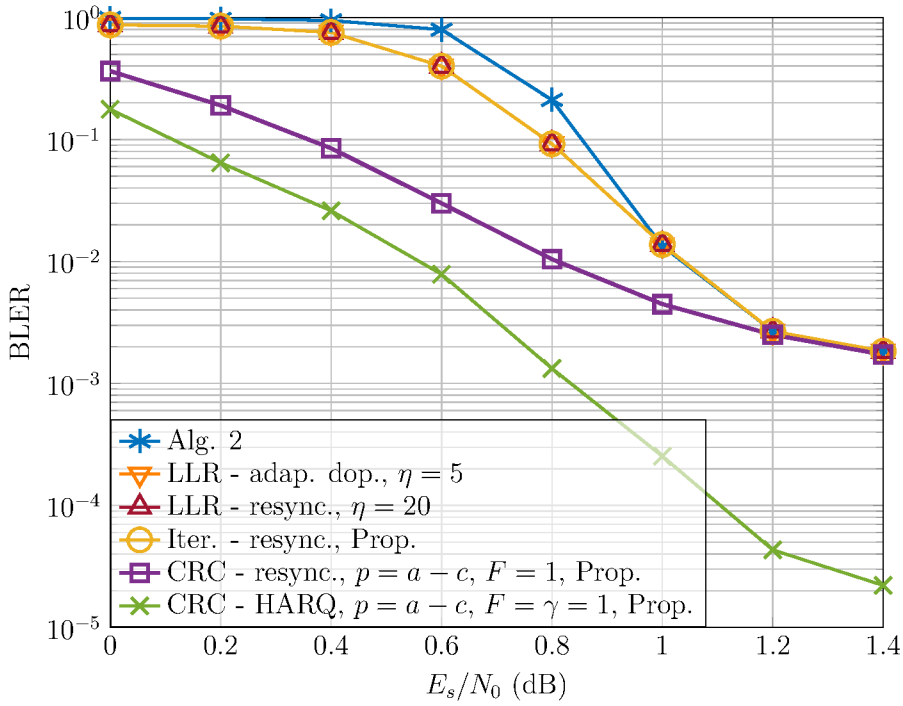


Figure 4.11: BLER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies on Rayleigh channel with $K = 5$.

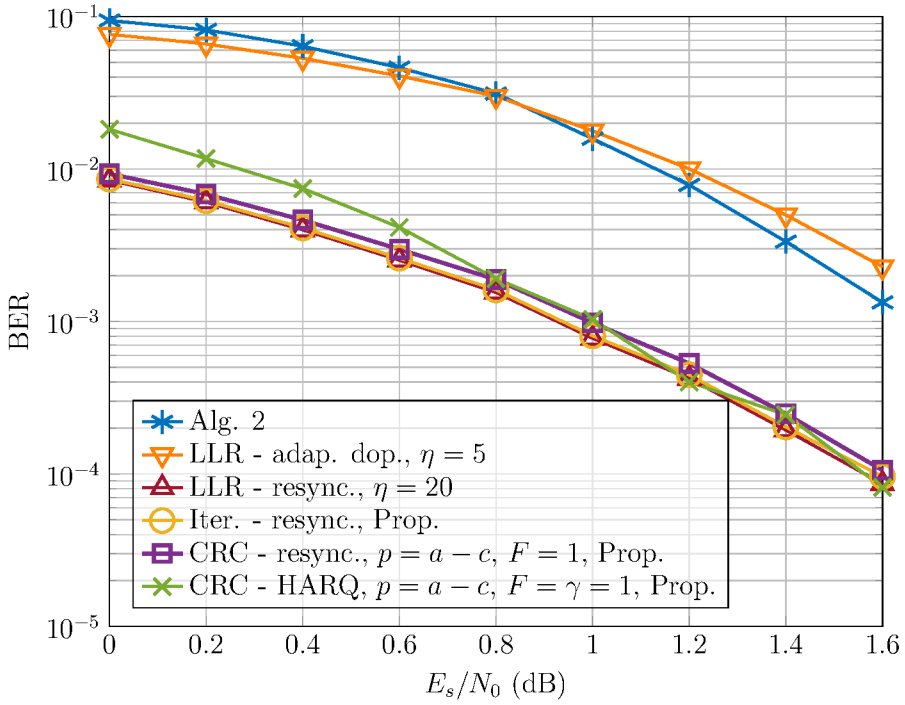


Figure 4.12: BER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies, on Rayleigh channel with $K = 50$.

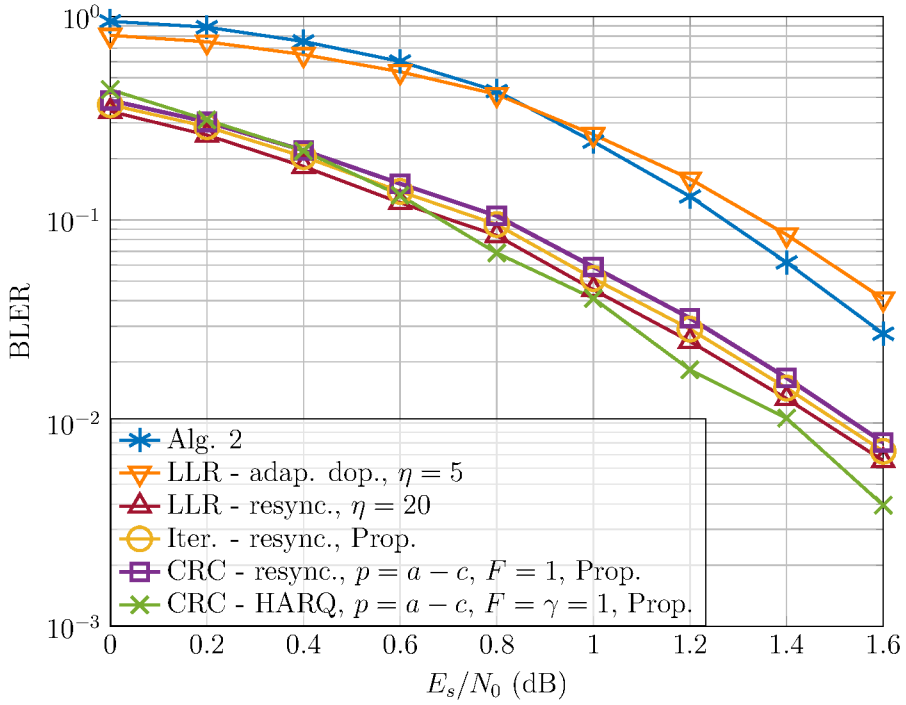


Figure 4.13: BLER vs SNR per coded bit for $\mathcal{C}$, with $L = 500$, under SWD with different detection methods and mitigation strategies on Rayleigh channel with $K = 50$.

Chapter 5

Conclusions

We introduced two novel adaptive strategies to improve the SWD performance of SC-LDPC codes: a resynchronization strategy and a type-I HARQ strategy that selectively requests retransmissions. Moreover, we proposed two error detection strategies to complement the aforementioned mitigation techniques. We proposed an iteration-based threshold that triggers a mitigation request based on the convergence behavior of the decoding windows, and explored the effects of the concatenation of an outer CRC code with SC-LDPC codes. All the proposed techniques, both for detection and mitigation, maintain minimal deviation from conventional SWD architectures, require slight changes in the hardware (e.g., the HARQ mechanism can be implemented by adding two small buffers to the baseline decoder) and do not change the decoding process. Even though resynchronization and retransmission cause some throughput and goodput loss, simulations show significant BER and BLER gains compared to standard SWD and comparable techniques such as adaptive doping. Moreover, especially for CRC-aided HARQ mechanism, the trade-off between throughput/goodput and BER/BLER performance is favorable under typical operating conditions. In addition, we also demonstrated the effectiveness of the proposed techniques under harsher channel conditions, considering both fast and slow fading in a Rayleigh channel. Even in these scenarios, the proposed strategies introduced significant gains in error rate performance compared with the baseline SWD, thus proving their ability to mitigate error propagation and to break the error chain.

To characterize the proposed type-I HARQ strategy, we also developed a simulation-calibrated finite-state model based on the observable history of retransmission budget exhaustion events, with analytically derived state transitions and state-dependent success probabilities estimated from finite-length simulations. Then, we demonstrated that the resulting model-predicted state occupancies, BLERs, and throughputs compare favorably with simulated data.

Numerical simulations showed that the proposed CRC-based type-I HARQ mechanism is very promising, and therefore SC-LDPC codes paired with the schedule illustrated in Algorithm 4 may be employed in Ultra Reliable Low Latency Communications (URLLC) applications, since, for instance, 5G already includes several mechanisms to reduce the latency incurred by NACKs and retransmissions (e.g., flex-

ible numerology). Furthermore, this technique can be easily converted to implement blind HARQ retransmissions [32] to further reduce the overall latency. Moreover, in future work, more advanced HARQ protocols, e.g., including chase-combining or incremental redundancy, could also be considered in order to enhance reliability of SWD even further.

Bibliography

[1] C. Deng et al., "IEEE 802.11be Wi-Fi 7: New challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2136–2166, 2020.

[2] 3GPP, *3rd Generation Partnership Project; 5G; NR; Multiplexing and channel coding*, TS 38.212 V15.2.0, 2018.

[3] Consultative Committee for Space Data Systems (CCSDS), "Low Density Parity Check codes for use in near-earth and deep space applications," *Exp. Specification CCSDS 131.1-O-2*, 2007.

[4] Consultative Committee for Space Data Systems (CCSDS), *Short Blocklength LDPC codes for TC synchronization and channel coding*, Orange Book, 2012.

[5] A. J. Feltstrom and K. S. Zigangirov, "Time-varying periodic convolutional codes with Low-Density Parity-Check matrix," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2181–2191, 1999.

[6] M. Lentmaier, G. P. Fettweis, K. S. Zigangirov, and D. J. Costello, "Approaching capacity with asymptotically regular LDPC codes," in *2009 Information Theory and Applications Workshop*, IEEE, 2009, pp. 173–177.

[7] S. Kudekar, T. Richardson, and R. L. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 7761–7813, 2013.

[8] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza, "Windowed decoding of protograph-based LDPC convolutional codes over erasure channels," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2303–2320, 2011.

[9] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[10] M. M. M. Costantino, M. Battaglioni, and D. G. Mitchell, "A Strategy to Detect Error Propagation in Sliding Window Decoding of SC-LDPC Codes," in *2025 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2025, pp. 1–6.

[11] W. E. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[12] R. Gallager, "Low-Density Parity-Check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[13] D. J. MacKay and R. Neal, "Good codes based on very sparse matrices," in *IMA International Conference on Cryptography and Coding*, Springer, 1995, pp. 100–111.

[14] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," *IPN progress report*, vol. 42, no. 154, pp. 42–154, 2003.

[15] J. G. Proakis and M. Salehi, *Digital Communications, 5th Edition*. McGraw-Hill Higher Education, 2008.

[16] D. J. Costello, L. Dolecek, T. E. Fuja, J. Kliewer, D. G. M. Mitchell, and R. Smarandache, "Spatially coupled sparse codes on graphs: Theory and practice," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 168–176, 2014.

[17] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 803–834, 2011.

[18] D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, "Spatially coupled LDPC codes constructed from protographs," *IEEE Transactions on Information Theory*, vol. 61, no. 9, pp. 4866–4889, 2015.

[19] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, IEEE, vol. 2, 1993, pp. 1064–1070.

[20] N. Wiberg, "Codes and decoding on general graphs," 1996.

[21] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding Low-Density Parity-Check (LDPC) codes," *IEEE transactions on communications*, vol. 53, no. 4, pp. 549–554, 2005.

[22] I. Ali, J.-H. Kim, S.-H. Kim, H. Kwak, and J.-S. No, "Improving windowed decoding of SC LDPC codes by effective decoding termination, message reuse, and amplification," *IEEE access*, vol. 6, pp. 9336–9346, 2017.

[23] N. U. Hassan, M. Schlüter, and G. P. Fettweis, "Fully parallel window decoder architecture for Spatially-Coupled LDPC codes," in *2016 IEEE International Conference on Communications (ICC)*, IEEE, 2016, pp. 1–6.

[24] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM) 2001*, vol. 2, 2001, 1036–1036E. DOI: 10.1109/GLOCOM.2001.965575

[25] M. Zhu, D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, "Adaptive doping of spatially coupled LDPC codes," in *2020 IEEE Information Theory Workshop (ITW)*, IEEE, 2021, pp. 1–5.

[26] K. Klaiber, S. Cammerer, L. Schmalen, and S. ten Brink, "Avoiding burst-like error patterns in windowed decoding of spatially coupled LDPC codes," in *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, IEEE, 2018, pp. 1–5.

[27] M. Zhu, D. G. Mitchell, M. Lentmaier, and D. J. Costello, "Error Propagation Mitigation in Sliding Window Decoding of Spatially Coupled LDPC Codes," *IEEE Journal on Selected Areas in Information Theory*, 2023.

[28] M. Zhu, D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, "Decoder error propagation mitigation for spatially coupled LDPC codes," in *2020 International Symposium on Information Theory and Its Applications (ISITA)*, IEEE, 2020, pp. 175–179.

[29] O. Griebel, B. Hammoud, and N. Wehn, "Adaptive sliding window decoding of spatially coupled low-density parity-check codes: Algorithms and energy efficient implementations," *IEEE Access*, vol. 12, pp. 191 140–191 161, 2024.

[30] M. Zhu, D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, "A novel design of spatially coupled LDPC codes for sliding window decoding," in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 473–478.

[31] J. Cantillo, J. Lacan, and I. Buret, "A CRC Usefulness Assessment for Adaptation Layers in Satellite Systems," in *24th AIAA International Communications Satellite Systems Conference*, 2006, p. 5358.

[32] M. T. Kawser, I. Kabir, S. Sultana, and T. Ahmad, "Blind HARQ Retransmissions with Different Multiplexing between Time and Frequency for URLLC in 5G," in *2025 IEEE International Conference on Electro Information Technology (eIT)*, IEEE, 2025, pp. 120–125.