UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

EDUARDO COSTA DE AQUINO

A Mathematical Tool for Constructing Parametrizable Spatially-Coupled LDPC Codes With Cyclic Structure and Large Girth

Porto Alegre

2019

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

EDUARDO COSTA DE AQUINO

A Mathematical Tool for Constructing Parametrizable Spatially-Coupled LDPC Codes With Cyclic Structure and Large Girth

Projeto realizado na Technische Universität Kaiserslautern

Supervisor: Dr.-Ing. Timo Lehnigk-Emden

Porto Alegre
2019

CIP - Catalogação na Publicação

Aquino, Eduardo Costa de
A Mathematical Tool for Constructing Parametrizable Spatially-Coupled LDPC Codes With Cyclic Structure and Large Girth / Eduardo Costa de Aquino. -- 2019.
55 f.
Orientador: Timo Lehnigk-Emden.

Trabalho de conclusão de curso (Graduação) -- Universidade Federal do Rio Grande do Sul, Escola de Engenharia, Curso de Engenharia Elétrica, Porto Alegre, BR-RS, 2019.

1. ldpc codes. I. Lehnigk-Emden, Timo, orient. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UFRGS com os dados fornecidos pelo(a) autor(a).

EDUARDO COSTA DE AQUINO

A Mathematical Tool for Constructing Parametrizable Spatially-Coupled LDPC Codes With Cyclic Structure and Large Girth

---

Prof. Dr. Ivan Müller
Supervisor - UFRGS

Prof. Dr. Luiz Fernando Ferreira
Chefe do Departamento de Engenharia Elétrica (DELET) - UFRGS

Aprovado em junho de 2019.

In memory of my grandmother Nely Haubert de Aquino who taught me all I really need to know.

Acknowledgements

I would like to express my special thanks to my advisor Dr.-Ing. Timo Lehnigk-Emden and my co-advisor Dipl.-Ing. Markus Fehrenz for conducting my work at the University of Kaiserslautern, for all the knowledge shared and guidance provided through the period of this work.

I am thankful to Dr.-Ing Christian Weis and to Prof. Dr.-Ing. Norbert Wehn for the opportunity and the amazing work structure provided at the Microelectronic Systems Design Research Group.

Resumo

Códigos *Spatially-coupled low-density parity-check* (SC-LDPC) têm apresentado melhor performance do que LDPC *block codes*, tanto em sistemas de comunicação quanto de armazenamento. Diversos métodos heurísticos de construção para estes códigos têm sido propostos na literatura, os quais possibilitam a obtenção de códigos SC-LDPC com específicos *node-degrees*, pequenos comprimentos de código e necessitam codificadores/decodificadores de arquitetura complexa não-parametrizável. Neste trabalho, construiu-se uma ferramenta matemática para a geração de códigos SC-LDPC com *node-degrees* arbitrários, *girth* de no mínimo seis e matriz de paridade com estrutura cíclica. Os códigos gerados satisfazem requisitos mínimos de performance de comunicação que podem ser previamente estabelecidos e podem ser codificados/decodificados por arquiteturas de hardware parametrizáveis de complexidade reduzida. Implementou-se em software um codificador de arquitetura parametrizável com tamanho de memória reduzido e baixa complexidade, conhecido como codificador baseado em *partial syndrome*, e verificou-se a codificação dos códigos construídos. As arquiteturas para codificadores do tipo *partial-syndrome* encontradas na literatura possuem taxas de codificação não arbitrárias e por isso, modificou-se os códigos SC-LDPC construídos, permitindo que os códigos gerados possam ser codificados com o mesmo codificador do tipo *partial-syndrome* para taxas de codificação arbitrárias. Implementou-se em software um decodificador de complexidade reduzida, conhecido como *window decoder*, e verificou-se a convergência dos códigos SC-LDPC construídos.

Palavras-chave: códigos spatially-coupled LDPC. códigos convolucionais. teoria de codificação.

Abstract

Spatially-coupled low-density parity-check codes (SC-LDPC) have been shown to be superior in performance than LDPC block codes for both communication and storage systems. Several heuristic construction methods for these codes have been proposed in the literature, but they allow the construction of SC-LDPC codes for only specific node-degrees, short code length and lead to encoders/decoders with non-parametrizable complex architectures. In this work we construct a mathematical tool for generating SC-LDPC codes with arbitrary node-degrees, girth of at least six and a parity-matrix with cyclic structure. The generated codes satisfy some minimum communication performance requirements which can be previously determined and can they can also be encoded/decoded with reduced-complexity parametrizable hardware architectures. An encoder architecture with reduced memory size and reduced-complexity, known as *partial-syndrome* based encoder, was implemented in software and the code encodability was verified. The partial-syndrome encoder structure proposed in the literature has constrained code rate and a modified SC-LDPC code was implemented, allowing the generated codes to be encoded with the partial-syndrome encoder architecture for arbitrary rates. A reduced-complexity decoder known as *window decoder* was implemented in software and the code decodability was also verified.

Keywords: spatially-coupled LDPC codes. convolutional codes. coding theory.

List of Figures

Figure 1 – A sketch of typical LDPC-BC decoded BER performance over the AWGNC. 13
Figure 2 – Communication system. 15
Figure 3 – Block code codeword. 17
Figure 4 – Convolutional code. 18
Figure 5 – (a) Parity-check matrix of an LPDC code and (b) its Tanner graph representation. 19
Figure 6 – (a) Protograph of the non-binary base matrix $B = [3\ 3]$. (b) Copying and transmission of the protograph over time for constructing a larger code. (c) Dispersion of the additional edges from each protograph to its neighbour protographs. (d) Graph representation of an infinite spatially coupled LDPC chain with memory $m_s = 2$. 22
Figure 7 – Spatial coupling between the code blocks at time instant t, $t-1$ and $t-2$ in an infinite SC-LDPC parity-check matrix. 23
Figure 8 – Parity-check matrix of an SC-LDPC code with the parameters given in Table 2. 24
Figure 9 – Information spreading along the codeword. The rectangles represent the systematic coupled code blocks. 25
Figure 10 – Encoded codeword for the information sequence $u_{[0,5]} = [1\ 0\ 1\ 1\ 1\ 0]$ using the parity-check matrix of the SC-LDPC code given in Figure 8. 27
Figure 11 – Summary of the SP-construction method for SC-LDPC codes. 28
Figure 12 – SC-LDPC code over $GF(7)$ based on the SP-method. 33
Figure 13 – Corresponding tanner graph representation of the SC-LDPC parity-check matrix of figure 12. 35
Figure 14 – Partial-syndrome based encoder for SC-LDPC Codes 36
Figure 15 – Finite State Machine for the partial-syndrome based encoder. 38
Figure 16 – (a) Graph representation of a $(3, 6, L)$ SC-LDPC code and (b) Graph representation of a $(4, 6, L)$ with modified structure. 39
Figure 17 – (a) Parity-check matrix of an SC-LDPC code over $GF(7)$ with modified structure. 40
Figure 18 – Window decoder of size W for SC-LDPC codes. 41
Figure 19 – Decoding process for the SC-LDPC parity-check matrix of figure 12. . . 43
Figure 20 – (a) Addition and multiplication tables for $GF(2)$. 52

List of Tables

Table 1 – SC-LDPC code parameters. 23
Table 2 – Parameters of the SC-LDPC code given in Figure 8. 24
Table 3 – Parameters of the SC-LDPC code given in Figure 12. 34

List of abbreviations and acronyms

AWGN	*Additive White Gaussian Noise*
BC	*Block Code*
BER	*Bit Error Rate*
BP	*Belief Propagation*
CN	*Check Node*
CPM	*Circulant Permutation Matrix*
FSM	*Finite State Machine*
GF	*Galois Field*
LDPC	*Low-Density Parity-Check*
LLR	*Log Likelihood Ratio*
ML	*Maximum Likelihood*
NB	*Non-Binary*
PTG	*Protograph*
SC	*Spatially-coupled*
SM	*Submatrix*
SNR	*Signal-to-Noise Ratio*
SP	*Superposition*
SW	*Section-Wise*
VN	*Variable Node*
ZM	*Zero Matrix*

Contents

1 INTRODUCTION 12

2 TECHNICAL BACKGROUND 15
2.1 Channel Coding 15
2.2 Channel Capacity 16
2.3 Linear Block Codes 16
2.4 Parity-Check Matrix 17
2.5 Convolutional Codes 17
2.6 Low-Density Parity-Check (LDPC) Codes 18
2.7 Spatially-Coupled LDPC Codes 20
2.7.1 Encoding Spatially-Coupled LDPC Codes: Example 23

3 SC-LDPC CODES CONSTRUCTION 28
3.1 Constructing SC-LDPC Codes With the Superposition Method . . . 28
3.1.1 Premises 28
3.1.2 Constructing Base Matrices With Doubly-Cyclic Structure Over Finite Fields 30
3.1.3 Constructing SC-LDPC Parity-Check Matrices 31
3.2 Partial-Syndrome Based Encoder for SC-LDPC Codes 34
3.2.1 Code Adaptation for Arbitrary Code Rates 38
3.2.2 Modified Partial-Syndrome Based Encoder for Encoding SC-LDPC Codes With Arbitrary Code Rates 39
3.3 Window Decoder for SC-LDPC Codes 40
3.3.1 Decoding Algorithm 42

4 RESULTS AND DISCUSSION 44

5 CONCLUSION 46

6 FUTURE WORK SUGGESTIONS 47

REFERENCES 48

APPENDIX 50

APPENDIX A – GALOIS FIELD ARITHMETIC 51

APPENDIX B – CIRCULANT PERMUTATION MATRIX DISPERSION 53

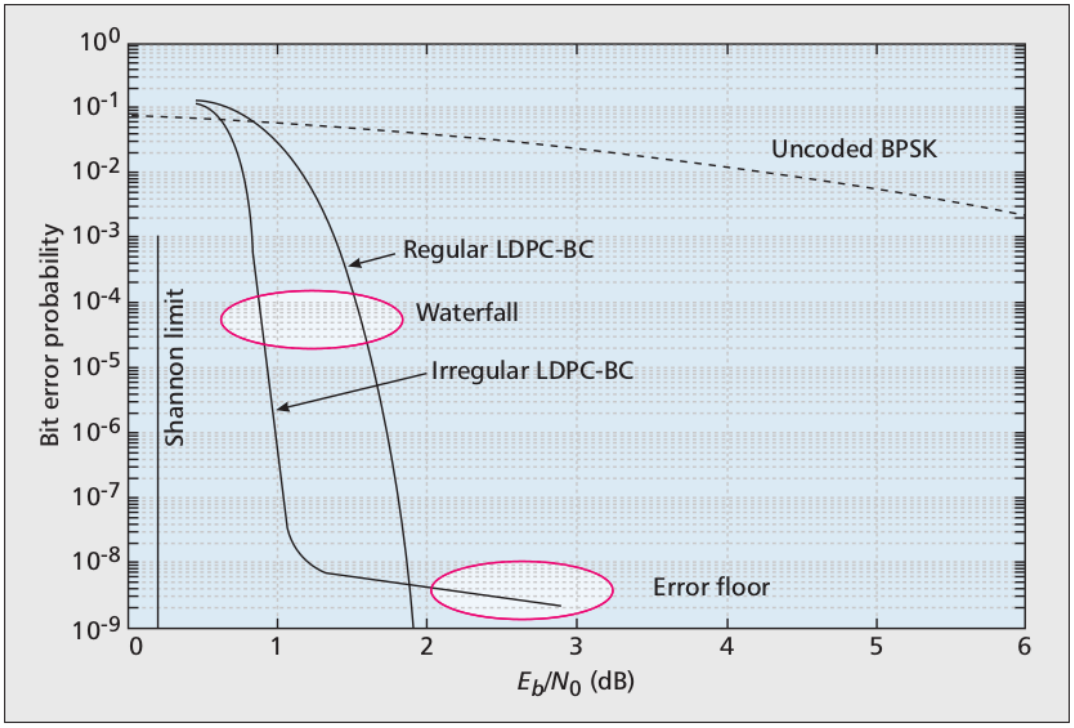
1 Introduction

Low-density parity-check (LDPC) codes were first proposed in 1962 by Robert Gallager (GALLAGER, 1962) and since their rediscovery in 1990, due to their channel capacity-approaching performance, they play an important role in the field of error correction for communication and storage systems. In the last decades LDPC codes have been used in many communication standards, such as WiMAX, WiFi, DVB-S2X, DVB-S2 and DVB-T2. The key feature that sets LDPC codes apart from other capacity approaching codes is that with suboptimal iterative *belief propagation*(BP) decoding, complexity grows only linearly with code block length, resulting in practically realizable decoder implementations for long block length codes (COSTELLO et al., 2014).

LDPC block codes (LDPC-BC) can be classified in two types: *regular* and *irregular*. Regular codes, as originally proposed by Gallager (GALLAGER, 1962), are asymptotically good in the sense that their *minimum distance* grows linearly with block length. This guarantees that these codes do not suffer from the *error floor* phenomenon, a flattening on the *bit error rate* (BER) curve that results in poor performance at high *signal-to-noise-ratios* (SNRs), as shown in Figure 1 for LDPC-BC decoded over the *additive white gaussian noise channel*(AWGN). However, the iterative decoding behavior of regular codes in the so-called *waterfall*, or moderate BER, region of the performance curve falls short of capacity, making them unsuitable for severely power-constrained applications, such as uplink cellular data transmission and digital satellite broadcasting systems, that must achieve the best possible performance at moderate BERs. On other hand, irregular codes, introduced by (LUBY et al., 2001), exhibit capacity approaching performance in the waterfall but are normally subject to a visible error floor, making them undesirable in certain applications, such as data storage and optical communication, that require very low decoded BERs (COSTELLO et al., 2014).

One type of LDPC code with a very particular structure is called *spatially-coupled* (SC) LDPC code, first invented by Jimenez Felström and Zigangirov in 1999 (FELSTRÖM; ZIGANGIROV, 1999). In fact, the SC-LDPC code is a convolutional LDPC code and its name comes from the *protograph*(PTG) based construction method, introduced by Thorpe in 2003 (THORPE, 2003). The SC-LDPC is constructed by spatially coupling classic LDPC block codes together, in such a way that one block of encoded information depends on the previously encoded blocks. The graph representation of each constituent LDPC block code is called a *protograph* and the graph representation of an SC-LDPC code is a chain of spatially-coupled protographs.

Figure 1 – A sketch of typical LDPC-BC decoded BER performance over the AWGNC.



Source: (COSTELLO et al., 2014)

SC-LDPC codes combine the best features of regular and irregular codes: they have capacity-approaching iterative decoding thresholds, characteristic of optimized irregular codes, thus promising excellent performance in the waterfall region, and their minimum code distance grows linearly with the block length, characteristic of regular codes, thus promising the elimination of an error floor (COSTELLO et al., 2014).

One specific algebraic method for constructing SC-LDPC codes, known as *superposition* (SP), guarantees that the size of the smallest cycle in the code graph representation is at least 6, an important property for achieving good communication performance, and if the parity-check matrix of the code has a cyclic structure the SC-LDPC code can be encoded/decoded with reduced-complexity encoder/decoder structures, based on the algebraic properties of the SC-LDPC parity-check matrix.

With the new requirements imposed by the current applications in communication and storage systems, such as transmission rate and power consumption, specific hardware architectures are necessary to achieve the most efficient error correction performance. In this work, we propose an efficient mathematical tool to construct parametrizable SC-LDPC codes based on the superposition method, where the code length, code memory and code rate can be arbitrarily chosen, making these codes suitable for a wide application range. The parity-check matrices of the SC-LDPC codes constructed with this tool have a regular

cyclic structure, allowing recursive encoding/decoding with reduced memory size, known as *partial-syndrome* SC-LDPC encoders/decoders, which achieve high throughputs, low latency and minimum occupation area. The constructed SC-LDPC codes also have a *girth* of at least 6, which guarantees that some minimum communication performance requirements are satisfied.

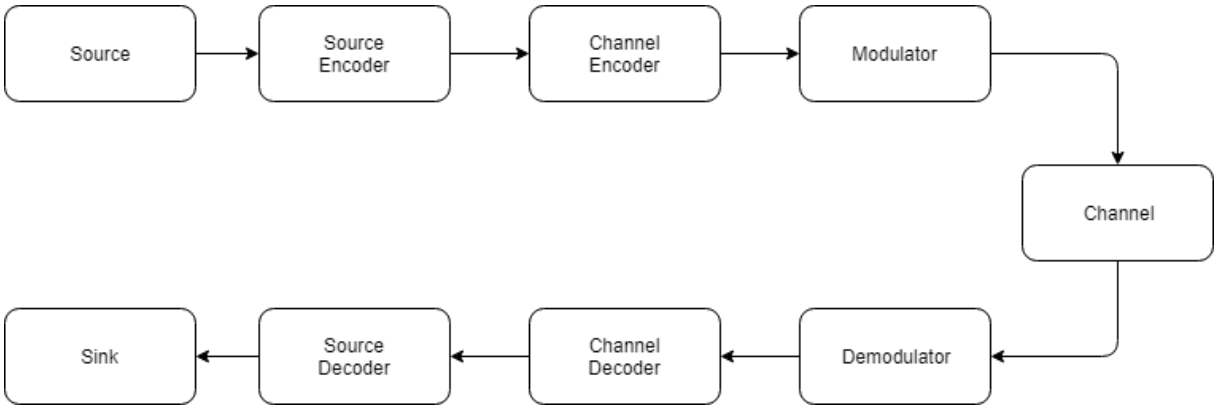
2 Technical Background

In this chapter it will be described the fundamentals of coding theory and it will be given an introduction to SC-LDPC codes.

2.1 Channel Coding

To understand what Channel Coding is, it is necessary to describe a communication system and its modules. The system shown on Figure 2 describes a basic communication system.

Figure 2 – Communication system.



Source: the author.

Source and Sink: information source provided from any system that has some data to be transmitted.

Source encoder and source decoder: the encoder converts information bits into a new bit sequence with more efficient representation. This process is also called compression. The source decoder recovers the source information based on the encoding process.

Channel encoder and channel decoder: the encoder adds some redundant information in order to protect the data to be transmitted over a channel that is subjected to interferences. Based on the added redundant bits, the decoder recovers the original data, despite the channel noise.

Modulator and demodulator: the modulator converts the bit stream provided by the encoder in a signal, in such a way that the signal to be transmitted matches the

channel characteristics. The demodulator makes the counterpart, recover the bit stream based on the used modulation.

Channel: the channel is the physical medium through which the data stream is transmitted. Channels can add interference to the data being transmitted. The channel output y is probabilistically modelled by the sum of the input x with the noise n, as given in Equation 1.

$$y = x + n \tag{1}$$

We usually define a communication channel with a triple, consistent of an input alphabet, an output alphabet, and for each input-output pair, a transition probability $p(o|i)$.

2.2 Channel Capacity

Based on the communication system model mentioned in Section 2.1, Shannon defined *channel capacity* (SHANNON, 1948), which measures the amount of information that can be transmitted through a channel. This notion of capacity is only a theoretical limit and it does not guarantee the existence of such a scheme that achieves this limit. The channel capacity C can be calculated as given by Equation 2.

$$C = Wlog(1 + SNR)\left[\frac{bits}{s}\right] \tag{2}$$

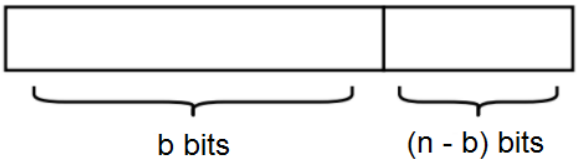
With SNR the signal-to-noise power ratio detected at the receiver and W the channel bandwidth. Setting a transmission rate R to a $R < C$ value, an error probability as small as desired can be achieved. In channel coding, the Sphere Packing Bound(SP59) limit is also used, which is based on the Shannon limit but also takes into account the code rate to calculate the communications limit.

2.3 Linear Block Codes

An alphabet is defined over a *finite field*, also known as *Galois field* (see Appendix A), of size 2, $GF(2)$, that means we have only two possible symbols, 0 and 1. Using block coding, we segment the information sequence into blocks of a fixed size b, having then $K = 2^b$ possible distinct messages. At the encoder, each of this input messages will be encoded into a longer binary sequence of size n, as shown in Figure 3, being $n > b$, we call that a *codeword*. With the set of all these messages, we have an (n, b) block code, with $m = n - b$ redundant bits, that will be used to correct the errors caused by the

communication noise. The ratio between the number of information bits and the total block size(information bits and redundant bits added by the encoder) we call *code rate*.

Figure 3 – Block code codeword.



Source: the author.

The *code minimum distance*, or *Hamming distance* (HAMMING, 1950) is the number of positions in which two different codewords c_1 and c_2 differ. That means, a receiver based on a code distance d, is able to detect up to $d-1$ transmission errors, since changing only $d-1$ positions of a codeword can never lead to an unwanted, but valid, codeword.

2.4 Parity-Check Matrix

A *parity-check matrix* is used in the decoding process to check if a codeword v is valid given the linear block code C, respect to the equation 3.

$$vH^T = 0 \tag{3}$$

As C is an (n, b) b-dimensional block code, the null space of this code, denoted C_d, is an $(n, n-b)$, $(n-b)$-dimensional code. Using $H = (h_0, h_1, \ldots, h_{n-b-1})$ as the set of $n-b$ linearly independent codewords in the basis of C_d, the codewords in C_d form the $(n-b)_x n$ parity-check matrix in Equation 4.

$$H = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{n-b-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{0,1} & \ldots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \ldots & h_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-b-1,0} & h_{n-b-1,1} & \ldots & h_{n-b-1,n-1} \end{bmatrix} \tag{4}$$

2.5 Convolutional Codes

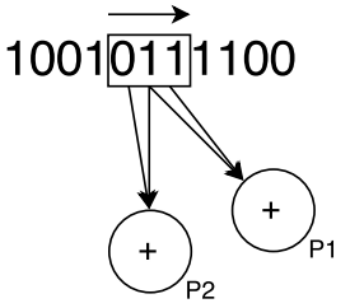
Convolutional codes were introduced in 1955 by Peter Elias (ELIAS, 1955). It was thought that convolutional codes could be decoded with arbitrary quality at the expense

of computation and delay. Unlike block codes, where the information bits are followed by the parity bits, convolutional coding spread the information bits along the bit sequence. This means that the convolutional codes map information to code bits not block wise, but sequentially convolve the sequence of information bits according to some rule. We also have the n and b code parameters, where the number of parity bits is $m = n - b$, but in addition, a new m_s parameter, that is the number of memory elements (registers) used to store data from past bits. That makes the code defined by a triple (n, m, m_s).

The encoder basically uses a sliding window, to calculate $m \geq 1$ parity bits by combining various subsets of bits in the window. Then in every time step, the window overlaps and slides, calculating new parity bits. This sliding process represents the convolution of the encoder over the data, which provides the term convolutional codes.

In the case of Figure 4, we are generating 2 parity-checks. One $P1$ based on the second and third bits in the window, and one $P2$ based on the first and second bits in the window. The size of the window, in symbols, is called constraint length. With a bigger window, the number of information bits that may influence each parity bit is potentially larger. With a bigger window we might have some resilience to errors. The trade-off is that it will also take more time to decode codes using a long constraint length.

Figure 4 – Convolutional code.



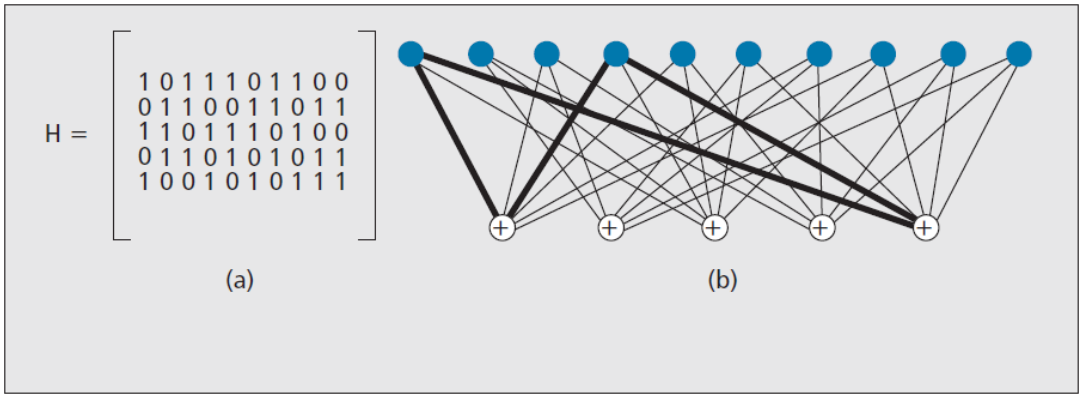
Source: the author.

2.6 Low-Density Parity-Check (LDPC) Codes

LDPC codes are a class of linear block codes. From their name, LDPC codes are codes defined by a very sparse, or low-density, parity-check matrix, that is, the H matrix has a very low density of non-zero entries. That sparseness guarantees a decoding complexity that increases only linearly with the code length.

LDPC codes are usually designed by constructing a parity-check matrix with the desired properties. As an example, an LDPC parity-check matrix H is shown in Figure 5.

Figure 5 – (a) Parity-check matrix of an LPDC code and (b) its Tanner graph representation.



Source: (COSTELLO et al., 2014)

The difference between LDPC codes and other block codes is in the way they are decoded. Instead of using the *maximum likelihood*(ML) decoding schemes, which use short block codes to make the ML task less complex, LDPC codes are decoded iteratively, using a graphical representation of the H matrix.

From the H matrix we observe:

- m rows, each one representing a parity-check equation, that corresponds to the function of a check node (CN).

- n columns, one for each code bit, that corresponds to the function of a variable node (VN).

- dc: the number of non-zero elements in a row, it is called *check node degree*.

- dv: the number of non-zero elements in a column, we name it textitvariable node degree.

The parity-check matrix H can also be represented using a *Tanner graph*, as shown in Figure 5(b). A check node j is connected to a variable node i if the position h_{ij} in the H matrix is a 1. In Figure 5(b), the darkened edges indicate a *cycle* of length 4. In general, codes with short cycles do not perform well in the waterfall due to the buildup of correlation in the iterative decoding process. Hence, it is desirable to choose codes with large *girth*, the length of the shortest cycle, for good waterfall performance.

2.7 Spatially-Coupled LDPC Codes

In order to describe the *spatial coupling* concept for an SC-LDPC code constructed over a finite field $GF(q)$, where q is the field size, let us consider a transmission of a sequence of LDPC code blocks v_t, $0 \leq t < L-1$, described by Equation 5.

$$v_{[0,L-1]} = [v_0, v_1, ..., v_{L-1}] \tag{5}$$

where each code block $v_t = [v_{t,0}, v_{t,1}, ..., v_{t,n-1}]$, $v_{t,j} = (v_{t,j}^{(0)}, v_{t,j}^{(1)}, ..., v_{t,j}^{(q-2)}), v_{t,j}^{(a)} \in GF(2)$, for $0 \leq j < n$, and $0 \leq a < (q-1)$. The number of bits in a code block v_t is $n(q-1)$. The code sequence described by Equation 5 is generated from an uncoded information sequence given by Equation 6.

$$u_{[0,L-1]} = [u_0, u_1, ..., u_{L-1}] \tag{6}$$

where $u_t = [u_{t,0}, u_{t,1}, ..., u_{t,b-1}]$, $u_{t,j} = (u_{t,j}^{(0)}, u_{t,j}^{(1)}, ..., u_{t,j}^{(q-2)}), u_{t,j}^{(a)} \in GF(2)$, for $0 \leq j < b$, and $0 \leq a < (q-1)$. The number of bits in an information block u_t is $b(q-1)$. Hence, each code block v_t has a block of parity bits $p_t = [p_{t,0}, p_{t,1}, ..., p_{t,m-1}]$, where $m = n - b$ and each $p_{t,j} = (p_{t,j}^{(0)}, p_{t,j}^{(1)}, ..., p_{t,j}^{(q-2)}), p_{t,j}^{(a)} \in GF(2)$, for $0 \leq j < m$, and $0 \leq a < (q-1)$. The number of bits in a parity block p_t is $m(q-1)$.

The code sequence given by Equation 5 satisfies Equation 3, which can be rewritten as in Equation 7:

$$v_{[0,L-1]} H_{[0,L-1]}^T = 0 \tag{7}$$

where $H_{[0,L-1]}^T$ is the transposed parity-check matrix of a terminated spatially-coupled LDPC code C, as given by Equation 8.

$$H_{[0,L-1]} = \begin{bmatrix} H_0(0) & & \\ \vdots & \ddots & \\ H_{m_s}(0) & & H_0(L-1) \\ & \ddots & \vdots \\ & & H_{m_s}(L-1) \end{bmatrix} \tag{8}$$

All non-zero elements of $H_{[0,L-1]}$ are lying in a main diagonal and each element is a sub-matrix of dimension m x n defined in Equation 9.

$$H_i(t) = \begin{bmatrix} h_i^{(0,0)}(t) & \dots & h_i^{(0,n-1)}(t) \\ \vdots & \ddots & \vdots \\ h_i^{(m-1,0)}(t) & \dots & h_i^{(m-1,n-1)}(t) \end{bmatrix} \tag{9}$$

where $h_i^{(p,e)}(t)$ is a *circulant permutation matrix* (*CPM*) of size $(q-1)$ x $(q-1)$ (see Appendix B), $0 \leq t < L$, $0 \leq i \leq m_s$, for $0 \leq p < m$, $0 \leq e < n$.

In an SC-LDPC code the code blocks at different time instants are interconnected. Instead of encoding all codewords independently, the blocks v_t are coupled by the encoder to other time instants. The largest distance between a pair of coupled blocks defines the *syndrome-former memory* m_s of the SC-LDPC code. The corresponding sequence of coupled code blocks forms a codeword $v_{[0,t]} = [v_0, ..., v_t, ..., v_{L-1}]$ of a terminated SC-LDPC code, $t = 0, ..., L-1$. We assume $H_i(t) = H_i(t'), i = 0, ..., m_s, t' = t + m_s + 1$, $t' < L$, i.e, a time invariant-code where the sub-matrices $H_i(t)$ are periodically repeated with period $m_s + 1$.

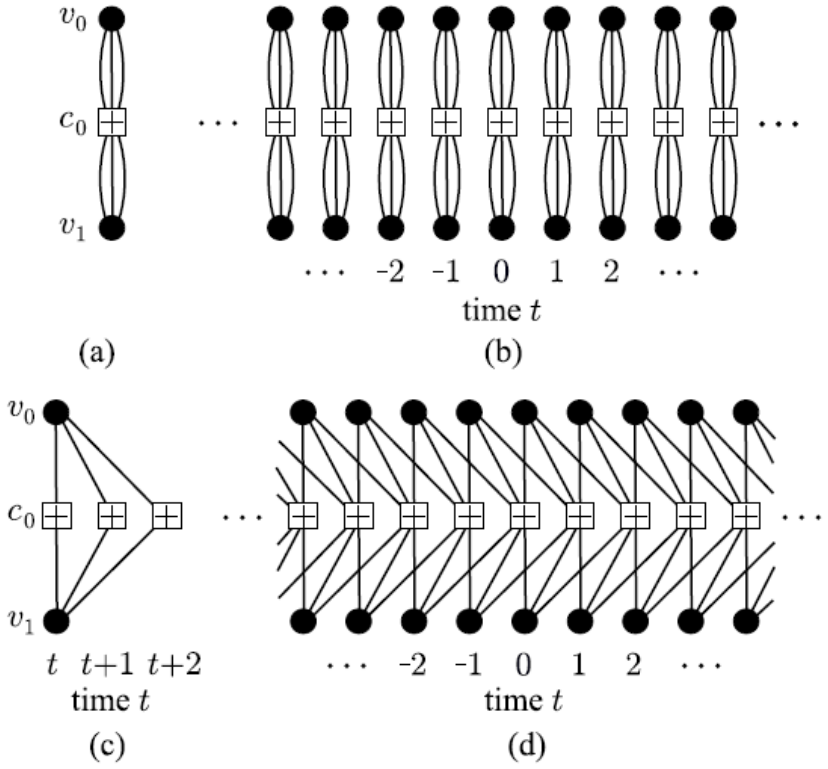
To better understand the graph representation of the SC-LDPC parity-check matrix and better explain the spatial coupling between the coupled blocks, let us construct a small SC-LDPC code using a graph-based construction method known as *protograph* (PTG). In this method, instead of constructing a large parity-check matrix that has the properties that we want, we start with a small protograph representing a smaller parity-check matrix, which has the desired properties. This smaller parity-check matrix is called a *base matrix*.

Figure 6(a) shows the protograph representation for a small base matrix B. This protograph has m check nodes and n variable nodes. Indeed, as this protograph has more than one edge between variable nodes and check nodes, the entries inside its corresponding base matrix are non-binary to represent the corresponding number of edges. In this example, the corresponding base matrix is $B = [3\ 3]$. Then, to construct a larger code, we copy the protograph as many times as desired, as shown in Figure 6(b), where each protograph can be viewed as a transmission of LDPC-BCs over time.

To construct the SC-LDPC code we change the connections of each two additional edges inside each protograph and connect them to its two neighbour protographs. This operation is called *edge dispersion* and it is shown in Figure 6(c). Applying the edge dispersion on all blocks of Figure 6(b), we get the infinite spatially-coupled LDPC chain of Figure 6(d). Now, to encode the block at time instant t, we also need the values of the previous encoded blocks at time instants $t-1$ and $t-2$, as shown in Figure 6(c). Basically, it means that now the values of the two previously encoded blocks must be stored for encoding the incoming information block, i.e., the code has *memory*. The graph representation of the interconnected blocks can be viewed as a spatially-coupling between those protographs. The code memory size depends on how many additional edges are inside each protograph to be connected to its neighbour protographs, i.e, the number of previously encoded blocks necessary to encode the next incoming block and it is given by $m_s = g.c.d.(dv, dc) - 1$, where *g.c.d.* stands for *greatest common divisor*, dv is the variable node degree and dc is the check node degree. In this example, $m_s = 2$, corresponding to the two additional edges connected from one protograph to its two neighbour protographs.

On practice, the encoding process will start and finish at some specific time and the parity-check matrix H must be terminated on the bounds. Its corresponding SC-LDPC

Figure 6 – (a) Protograph of the non-binary base matrix $B = [3\ 3]$. (b) Copying and transmission of the protograph over time for constructing a larger code. (c) Dispersion of the additional edges from each protograph to its neighbour protographs. (d) Graph representation of an infinite spatially coupled LDPC chain with memory $m_s = 2$.



Source: (MITCHELL; LENTMAIER; COSTELLO D. J., 2015)

chain shown on Figure 6(d) finishes at some some specific time instant and therefore, the check nodes on the bounds have less connections. For constructing an SC-LDPC code even larger, we can copy the SC-LDPC chain of Figure 6(d) after termination as many times as desired. Considering c the number of copied terminated chains, the length of the coupling chain L will be $(m_s + 1)c$ code blocks.

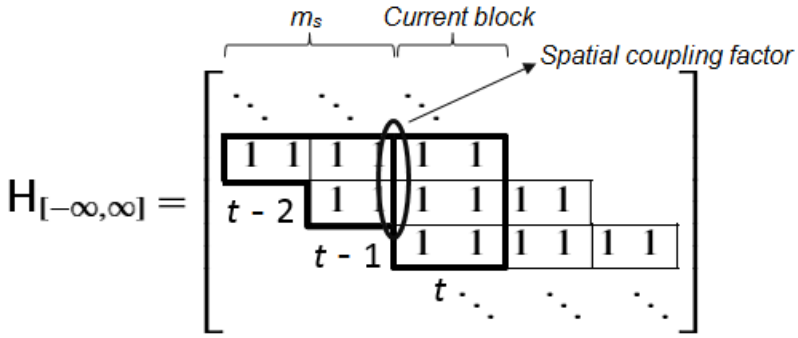
The parity-check matrix H represented by the SC-LDPC chain shown in Figure 6(d) has the structure given by Equation 8 and it is shown in Figure 7. All non-zero entries are lying in a main diagonal of width $v_s = (m_s + 1)n$ and outside this diagonal all entries are zeros. The entries are the sub-matrices of size mxn given in Equation 9, where each column of sub-matrices corresponds to a coupled code block of length $n(q - 1)$, with $m(q - 1)$ parity bits, as shown in Figure 6, assuming a systematic code. In this example, $m = dv/(m_s + 1) = 1$, $n = dc/(m_s + 1) = 2$ and the entries of each sub-matrix are just one element, considering $GF(2)$. v_s is called the *decoding constraint length* of the code

and gives the number of variable nodes involved in each parity-check equation. Table 1 summarizes all code parameters and their definitions.

Table 1 – SC-LDPC code parameters.

Parameter	Abbreviation	Definition
Finite field of size q	$GF(q)$	–
Variable node degree	dv	–
Check node degree	dc	–
Code memory size	m_s	$(g.c.d(dv, dc) - 1)$
Protograph's variable nodes	m	$\frac{dv}{m_s+1}(q-1)$
Protograph's check nodes	n	$\frac{dc}{m_s+1}(q-1)$
Decoding constraint length	v_s	$(m_s+1)(q-1)n$
Number of copied SC-LDPC blocks	c	–
Coupling chain length	L	$(m_s+1)c$
Code rate	R	$\frac{n-m}{n}$

Figure 7 – Spatial coupling between the code blocks at time instant t, $t-1$ and $t-2$ in an infinite SC-LDPC parity-check matrix.



Source: the author.

2.7.1 Encoding Spatially-Coupled LDPC Codes: Example

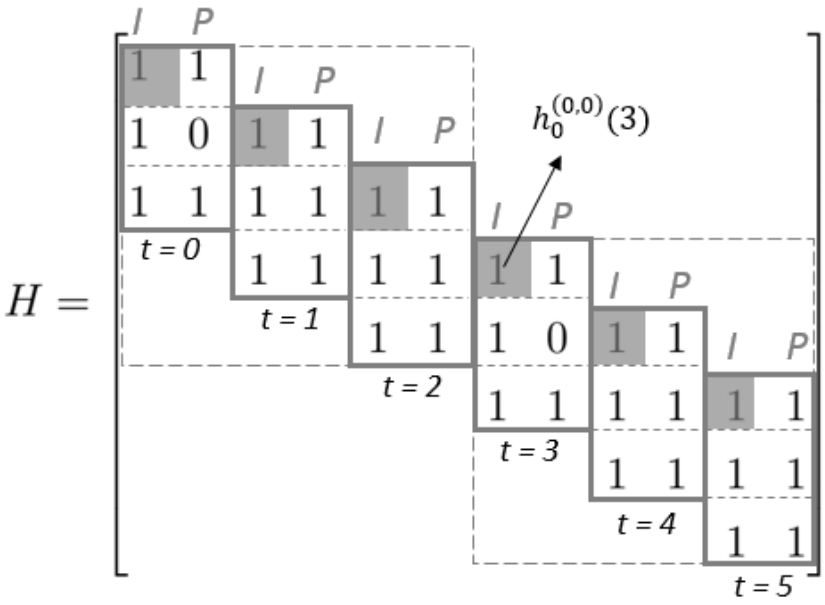
As an example, it will be shown now one type of encoding algorithm for encoding the information sequence $u_{[0,5]} = [1\ 0\ 1\ 1\ 1\ 0]$ using the terminated parity-check matrix

H of the SC-LDPC code over $GF(2)$ given in Figure 8. All code parameters are given in Table 2.

Table 2 – Parameters of the SC-LDPC code given in Figure 8.

Parameters	q	dv	dc	m_s	m	n	v_s	c	L	R
Values	2	3	6	2	1	2	6	2	6	1/2

Figure 8 – Parity-check matrix of an SC-LDPC code with the parameters given in Table 2.



Source: the author.

Assuming a systematic code, each column of sub-matrices corresponds to a code block, where the first $(n-m)$ bits are the information bits and the last m bits are the parity bits, as indicated above each column in Figure 8 by the symbols I (for information) and P (for parity). Therefore, the SC-LDPC code spreads the bits of information evenly along the coupled chain of code blocks, as shown in Figure 9.

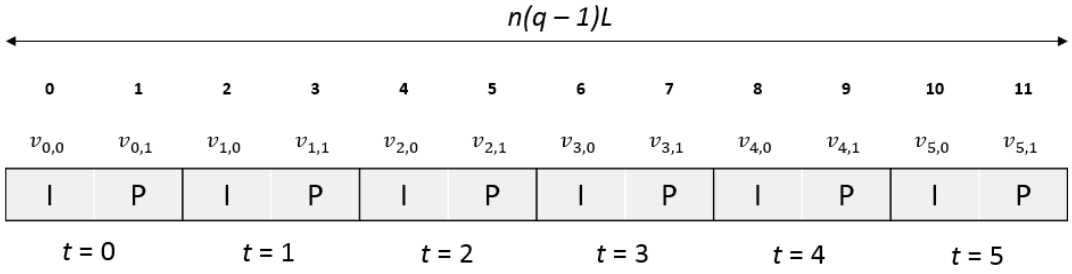
Step 1: as the first $(n-m)(q-1) = 1$ bits of each code block are the information bits, we start sending the incoming bit of information to the encoder's output:

$$v_{0,0} = u_{0,0} = 1$$

Step 2: the last $m(q-1)$ bits of each code block are the parity bits and they are calculated based on the received message as:

$$v_{0,1} = u_{0,0} h_0^{0,0}(0)^T = 1$$

Figure 9 – Information spreading along the codeword. The rectangles represent the systematic coupled code blocks.



Source: the author.

where $h_0^{0,0}(0)$ is the part of the first sub-matrix in the column associated with each code block corresponding to information, which is shown by the shaded squares in Figure 8. At this point, we have already calculated the bits corresponding to the first code block of the codeword in Figure 9 and solved all variables involved in the first parity-check equation (first row) of the SC-LDPC parity-check matrix given in Figure 8.

$$v_0 = [v_{0,0}\ v_{0,1}] = [1\ 1]$$

Step 3: as the already calculated code block at $t = 0$ is coupled by the sub-matrix $H_1(0)$ in the second parity-check equation (second row) to the next code block to be calculated at $t = 1$ and also coupled by the sub-matrix $H_2(0)$ in the third parity-check equation (third row) to the code block to be calculated at $t = 2$, these contributions of the calculated code block at $t = 0$ necessary to encode the next code blocks will be calculated now and stored in the memory to be used later. The sub-matrices $H_i(t)$ necessary for calculating the next code blocks are called *partial-syndrome former matrices*.

$$S_{0,0} = v_0 H_1(0)^T = 1$$

$$S_{0,1} = v_0 H_2(0)^T = 0$$

$$S_0 = [S_{0,0}\ S_{0,1}] = [1\ 0]$$

The product $v_t H_i(t)^T$ is called *partial syndrome* $S_{t,r}$, $0 \leq r < m_s$.

To calculate the next code block at time instant $t = 1$ the steps 1 to 3 are repeated again for the second column of sub-matrices in the parity-check matrix.

Step 1:

$$v_{1,0} = u_{1,0} = 0$$

Step 2: for calculating the parity bits of the code block at $t = 1$ the values of the previous encoded block at $t = 0$ stored in $S_{0,0}$ must also be included:

$$v_{1,1} = u_{1,0} h_0^{0,0}(1)^T + S_{0,0} = 1$$

$$v_1 = [v_{1,0}\ v_{1,1}] = [0\ 1]$$

Step 3: the calculated code block at $t = 1$ is also coupled to the next two code blocks to be encoded and the syndromes corresponding to the partial-syndrome former matrices $H_1(1)$ and $H_2(1)$ will be now calculated and stored in the memory for further use.

$$S_{1,0} = v_1 H_1(1)^T + S_{0,1} = 1$$

$$S_{1,1} = v_1 H_2(1)^T = 1$$

$$S_1 = [S_{1,0}\ S_{1,1}] = [1\ 1]$$

To calculate the next code block at time instant $t = 2$ the steps 1 to 3 are repeated again for the third column of sub-matrices in the parity-check matrix.

Step 1:

$$v_{2,0} = u_{2,0} = 1$$

Step 2: for calculating the parity bits of the code block at $t = 2$ the values of the previous encoded block at $t = 1$ stored in $S_{1,0}$ must also be included:

$$v_{2,1} = u_{2,0} h_0^{0,0}(2)^T + S_{1,0} = 0$$

It is worth mentioning that the syndrome corresponding to the partial-syndrome former matrix $H_2(0)$ of the first encoded block at time instant $t = 0$ does not need to be considered here to calculate $v_{2,1}$, as it was already included in $S_{1,0}$.

$$v_2 = [v_{2,0}\ v_{2,1}] = [1\ 0]$$

Step 3: again the calculated code block at $t = 2$ is also coupled to the next two code blocks to be encoded and the syndromes corresponding to the partial-syndrome former matrices $H_1(2)$ and $H_2(2)$ are calculated and stored in the memory.

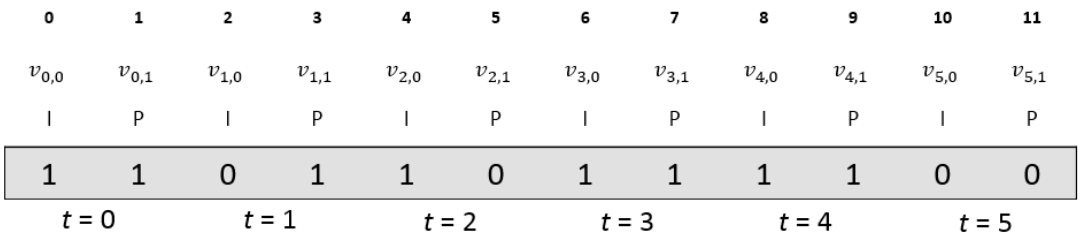
$$S_{2,0} = v_2 H_1^T(2) + S_{1,1} = 0$$

$$S_{2,1} = v_2 H_2^T(2) = 1$$

$$S_2 = [S_{2,0}\ S_{2,1}] = [1\ 1]$$

The steps 1 to 3 are repeated until all coupled code blocks are encoded. The steps 2 and 3 do not need to be calculated for the last time instant $t = 5$, as there is no next code block to be encoded. Figure 10 shows the complete codeword composed by all calculated coupled code blocks.

Figure 10 – Encoded codeword for the information sequence $u_{[0,5]} = [1\ 0\ 1\ 1\ 1\ 0]$ using the parity-check matrix of the SC-LDPC code given in Figure 8.



Source: the author.

3 SC-LDPC Codes Construction

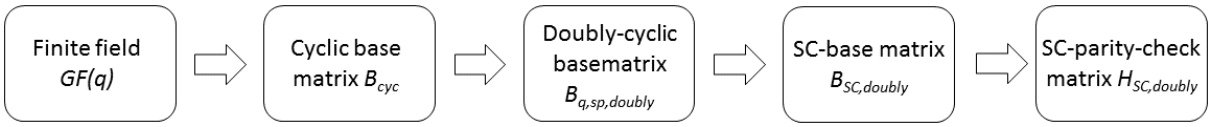
In this section it will be shown how to construct SC-LDPC codes based on an algebraic method called *superposition* (SP). This method makes use of the cyclic properties of *finite fields* to construct an SC-LDPC parity-check matrix with cyclic structure. The field elements determine the edges distribution pattern of the tanner graph associated to the parity-check matrix of the code in such a way that the performance requirements explained on the subsection 3.1.1 are met.

3.1 Constructing SC-LDPC Codes With the Superposition Method

The first step of the *superposition* method is the choice of a finite field $GF(q)$ where q is the field size. Then, instead of constructing the desired large parity-check matrix, which would be a complex task, the field elements are used to construct a small non-binary base matrix which has cyclic structure and some other important algebraic properties to ensure a minimum communication performance. Next, some elements of the base matrix are grouped together to construct another base matrix which has a *doubly* cyclic structure. This array with doubly structure is repeated over an infinite chain of base matrices which are *spatially coupled* together. Finally, each non-binary entry is dispersed by a CPM to get the binary SC-LDPC parity-check matrix.

All these steps are summarized on figure 11 and they will be described on the following subsections.

Figure 11 – Summary of the SP-construction method for SC-LDPC codes.



Source: the author.

3.1.1 Premises

The SP method for SC-LDPC codes has mainly two premises:

(1) The tanner graph associated to the SC-LDPC parity-check matrix must have a girth of at least six.

(2) The SC-LDPC parity-check matrix must have a doubly-cyclic structure.

The *girth* of the code is the size of the smallest cycle on the bipartite graph associated to the SC-LDPC parity-check matrix. The number of edges on a path is called the *length* of the path. A closed cycle that begins and ends at the same node is called a *cycle*. For good communication performance cycles smaller than six must be avoided (JOHNSON, 2010).

To satisfy the first premise the base matrix of the SC-LDPC code is constructed over a non-binary(NB) *finite field* (see Appendix A) satisfying certain constraints. The following theorem gives the necessary and sufficient conditions on the base matrix B for the tanner graph of a SC-LDPC code to satisfy these constraints. This theorem was proved in (DIAO et al., 2012).

Theorem 1 *Let B_{sp} be the base matrix of a binary SC-SP-LDPC code C_{sp} whose parity-check matrix H is the binary CPM-dispersion (see Appendix B) of B_{sp}, where the subindex sp stands for superposition. A necessary and sufficient condition for the tanner graph of C_{sp} to have girth at least six is that every 2 x 2 submatrix inside the base matrix B_{sp} contains at least one zero entry or is non-singular*

For convenience, the conditions given in the Theorem 1 are called the *2 x 2 submatrix (SM) constraint*.

To satisfy the second premise the parity-check matrix H of the SC-LDPC code must have a *doubly* cyclic structure, where the term *doubly* stands for a base matrix B_{sp} that has both *block-cyclic*(BC) and *section-wise*(SW) cyclic structures.

The BC-structure means that each row-block is the cyclic-shift of the row-block above it one constituent matrix to the right, and the top row-block is the cyclic-shift of the last row-block one constituent matrix to the right. Each column-block is the cyclic shift of the column block on its left one constituent matrix downward, and the first column-block is the cyclic-shift of the last column-block one constituent matrix downward.

The SW-cyclic structure means that if the base matrix B_{sp} is grouped into n sections, each containing k columns, if we shift all n sections simultaneously one position to the right within each section, the base matrix B_{sp} will still be an array with block-cyclic structure.

The doubly-cyclic structure allows the SC-LDPC code to be decoded with the reduced-complexity iterative decoding scheme proposed by (LIU; LIN; ABDEL-GHAFFAR, 2013) based on its block-cyclic structure or the reduced-complexity iterative decoding scheme presented by (LI et al., 2014b) based on its SW-cyclic structure, using the same parity-check matrix.

3.1.2 Constructing Base Matrices With Doubly-Cyclic Structure Over Finite Fields

Let α be a primitive element of the finite field $GF(q)$, where q is the field size. For $1 \leqslant m, n \leqslant q$, let $S_0 = \alpha^{i_0}, \alpha^{i_1}, ..., \alpha^{i_{m-1}}$ and $S_1 = \alpha^{j_0}, \alpha^{j_1}, ..., \alpha^{j_{n-1}}$ be two arbitrary subsets of elements in $GF(q)$ with i_k and j_l in $L = -\infty, 0, 1, 2, ..., q-2$, where $i_0 < i_1 < ... < i_{m-1}$ and $j_0 < j_1 < ... < j_{n-1}$. Let η be any non-zero element in $GF(q)$. Then the following m x n matrix can be constructed over the finite field $GF(q)$:

$$B = \begin{Bmatrix} \alpha^{i_0}\alpha^{j_0} - \eta & \alpha^{i_0}\alpha^{j_1} - \eta & \dots & \alpha^{i_0}\alpha^{j_{n-1}} - \eta \\ \alpha^{i_1}\alpha^{j_0} - \eta & \alpha^{i_1}\alpha^{j_1} - \eta & \dots & \alpha^{i_1}\alpha^{j_{n-1}} - \eta \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{i_{m-1}}\alpha^{j_0} - \eta & \alpha^{i_{m-1}}\alpha^{j_1} - \eta & \dots & \alpha^{i_{m-1}}\alpha^{j_{n-1}} - \eta \end{Bmatrix} \tag{10}$$

The matrix B in equation 10 has the following structural properties:

(1) All the entries in a row (or in a column) of B are distinct elements of $GF(q)$;

(2) Each row (or column) of B contains at most one zero element;

(3) No two rows (or two columns) in B have identical entries at any position. It was proved in (LI et al., 2014a) that the base matrix B satisfies the $2\,x\,2\,SM - constraint$.

As a special case of this construction, if we set $S_0 = \alpha^0, \alpha^{-1}, ..., \alpha^{-(q-2)}$ and $S_1 = \alpha^0, \alpha^1, ..., \alpha^{q-2}$ and $\eta = 1$, the following $2\,x\,2\,SM - constrained$ $(q-1)$ x $(q-1)$ base matrix over $GF(q)$ can be obtained:

$$B_{cyc} = \begin{bmatrix} \alpha^0 - 1 & \alpha^1 - 1 & \alpha^2 - 1 & \dots & \alpha^{q-3} - 1 & \alpha^{q-2} - 1 \\ \alpha^{q-2} - 1 & \alpha^0 - 1 & \alpha^1 - 1 & \dots & \alpha^{q-4} - 1 & \alpha^{q-3} - 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha^1 - 1 & \alpha^2 - 1 & \alpha^3 - 1 & \dots & \alpha^{q-2} - 1 & \alpha^0 - 1 \end{bmatrix} \tag{11}$$

The base matrix B_{cyc} in equation 11 has the following structural properties:

(1) All the entries in a row (or in a column) of B_{cyc} are distinct elements of $GF(q)$;

(2) Each row (or column) of B_{cyc} contains at most one zero element;

(3) No two rows (or two columns) in B_{cyc} have identical entries at any position;

(4) The $q-1$ entries lying on the main diagonal are the zero elements of $GF(q)$.

For $1 \leqslant m, n \leqslant q$, any m x n submatrix of B_{cyc} can be used as the base matrix for constructing a SC-LDPC code with doubly cyclic structure. Suppose $q-1$ can be factored

as the product of two positive integers l and r. Then, we partition B_{cyc} into an array of size r x r composed by submatrices of size l x l as follows:

$$B_{cyc} = \begin{bmatrix} B_{0,0} & B_{0,1} & \dots & B_{0,r-1} \\ B_{0,r-1} & B_{0,0} & \dots & B_{0,r-2} \\ \vdots & \vdots & \ddots & \vdots \\ B_{0,1} & B_{0,2} & \dots & B_{0,0} \end{bmatrix} \tag{12}$$

For $0 \leqslant j < r$ and $1 \leqslant m, n < l$, we now take an m x n submatrix $R_{0,j}$ from $B0, j$. The submatrices $R_{0,0}, R_{0,1}, ..., R_{0,r-1}$ are taken from the constituent matrices $B_{0,0}, B_{0,1}, ..., B_{0,r-1}$ under the following location constraint: for $j' \neq j$, the locations of the entries of $R_{0,j}$ in $B_{0,j}$ are identical to the locations of the entries of $R_{0,j'}$ in $B_{0,j'}$. Then, we form the following r x r array $B_{q,sp,doubly_{(m,n)}}$ of m x n sub-matrices over $GF(q)$ with doubly-cyclic structure:

$$B_{q,sp,doubly_{(m,n)}} = \begin{bmatrix} R_{0,0} & R_{0,1} & \dots & R_{0,r-2} & R_{0,r-1} \\ R_{0,1} & R_{0,2} & \dots & R_{0,r-1} & R_{0,0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ R_{0,r-1} & R_{0,0} & \dots & R_{0,r-3} & R_{0,r-2} \end{bmatrix} \tag{13}$$

The array $B_{q,sp,doubly_{(m,n)}}$ in equation 13 is a subarray of B_cyc in equation 12 and therefore, also satisfies the 2 x 2 SM-constraint. As the array has a doubly-cyclic structure, shifting the field elements inside all submatrices $R_{0,j}$ simultaneously does not affect the block-cyclic structure of the entire array $B_{q,sp,doubly_{(m,n)}}$.

3.1.3 Constructing SC-LDPC Parity-Check Matrices

A parity-check matrix for a SC-LDPC code which satisfies the premises given in 3.1.1 can be constructed from the 2 x 2 SM-constrained r x r array $B_{q,sp,doubly_{(m,n)}}$ over $GF(q)$ with doubly-cyclic structure.

The first construction step is to cut the matrix $B_{q,sp,doubly_{(m,n)}}$ along its main diagonal to construct two triangular sub-arrays, $T_{lower(r,r,m,n)}$ with lower triangular form

and $T_{upper(r,r,m,n)}$ with upper triangular form, as given by equations 14 and 15, respectively. The main diagonal of $B_{q,sp,doubly_{(m,n)}}$ is kept by $T_{lower(r,r,m,n)}$.

$$T_{lower(r,r,m,n)} = \begin{bmatrix} R_{0,0} & 0 & 0 & \dots & 0 & 0 \\ R_{0,1} & R_{0,2} & 0 & \dots & 0 & 0 \\ R_{0,2} & R_{0,3} & R_{0,4} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ R_{0,r-2} & R_{0,r-1} & R_{0,0} & \dots & R_{0,r-4} & 0 \\ R_{0,r-1} & R_{0,0} & R_{0,1} & \dots & R_{0,r-3} & R_{0,r-2} \end{bmatrix} \tag{14}$$

$$T_{upper(r,r,m,n)} = \begin{bmatrix} 0 & R_{0,1} & R_{0,2} & \dots & R_{0,r-2} & R_{0,r-1} \\ 0 & 0 & R_{0,3} & \dots & R_{0,r-1} & R_{0,0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & R_{0,r-3} \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix} \tag{15}$$

where the 0 elements in equations 14 and 15 correpond to zero-matrices with size m x n.

Each constituent matrix in either $T_{lower(r,r,m,n)}$ or $T_{upper(r,r,m,n)}$ is a member matrix in the set $R_{0,0}, R_{0,1}, ..., R_{0,r-1}$ and each member matrix in R appears r times in the union of $T_{lower(r,r,m,n)}$ and $T_{upper(r,r,m,n)}$. Since $B_{q,sp,doubly_{(m,n)}}$ satisfies the 2 x 2 SM-constraint and $T_{upper(r,r,m,n)}$ and $T_{lower(r,r,m,n)}$ are two disjoint sub-arrays of $B_{q,sp,doubly_{(m,n)}}$, $T_{lower(r,r,m,n)}$ and $T_{upper(r,r,m,n)}$ also satisfy the 2 x 2 SM-constraint.

Using $T_{lower(r,r,m,n)}$ and $T_{upper(r,r,m,n)}$ as building blocks the semi-infinite base matrix for a SC-LDPC code given in equation 16 can be constructed, where $T_{lower(r,r,m,n)}$ is placed above $T_{upper(r,r,m,n)}$ to keep the doubly cyclic structure for the entire array.

$$B_{SC,doubly(r,r,m,n)} = \begin{bmatrix} T_{lower} & & & \\ T_{uppper} & T_{lower} & & \\ & T_{upper} & T_{lower} & \\ & & T_{upper} & T_{lower} \\ & & & \ddots \end{bmatrix} \tag{16}$$

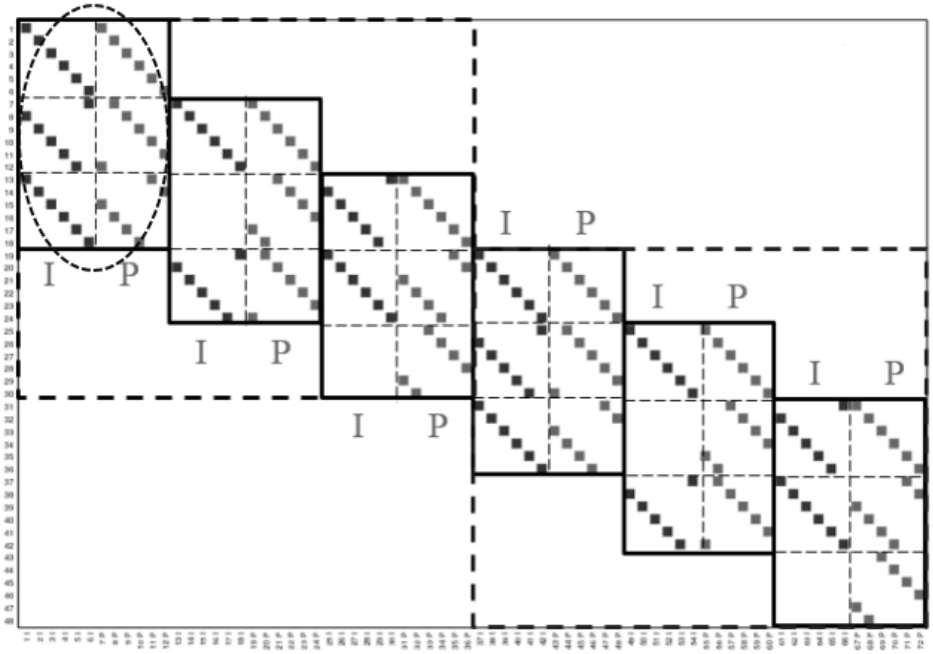
Since $T_{lower(r,r,m,n)}$ and $T_{upper(r,r,m,n)}$ satisfy the 2 x 2 SM-constraint, the semi-infinite array $B_{SC,doubly(r,r,m,n)}$ also satisfies the 2 x 2 SM-constraint.

Dispersing each non-zero entry, i.e, the field elements of $B_{SC,doubly(r,r,m,n)}$ into a binary CPM of size $(q-1)$ x $(q-1)$ and each 0-entry into a zero matrix of size $(q-1)$ x $(q-1)$ according to appendix B, we obtain a semi-infinite array $H_{SC,doubly(r,r,m,n)}$ of CPMs and zero matrices of size $(q-1)$ x $(q-1)$. The null space of $H_{SC,doubly(r,r,m,n)}$ gives

a periodically time-varying SC-LDPC code with period r and rate close to $(n-m)/n$ (assuming m < n) with doubly-cyclic structure whose its associated tanner graph has girth of at least six.

As the SC-LDPC codes have shown better performance for large code lengths, it suggests that the parity-check matrices of SC-LDPC codes used in real applications are in general too large. To give a small example of a parity-check matrix of the SC-LDPC code, the figure 12 shows the parity-check matrix of an SC-LDPC code over $GF(7)$, which is exactly the array $B_{SC,doubly(r,r,m,n)}$ given by equation 16 where each element is dispersed with a CPM of size 6 x 6. Figure 13 shows its corresponding tanner graph where I and P denote the variable nodes assigned to information and parity bits, respectively.

Figure 12 – SC-LDPC code over $GF(7)$ based on the SP-method.



Source: the author.

From the structural point of view, the main difference between this specific SC-LDPC code and the SC-LDPC code used for the encoding example of subsection 2.7.1 is that the entries inside the sub-matrices are CPMs of size $(q-1)$ x $(q-1)$, as shown in Equation 9. It can be viewed as a replication of each protograph by a factor $(q-1)$ in Figure 6, where the edge distributions are given by CPMs. The parameters of the SC-LDPC code in Figure 12 are given in Table 3.

Table 3 – Parameters of the SC-LDPC code given in Figure 12.

Parameters	q	dv	dc	m_s	m	n	v_s	c	L	R
Values	7	3	6	2	1	2	6	2	6	1/2

3.2 Partial-Syndrome Based Encoder for SC-LDPC Codes

A linear partial-syndrome based encoder for SC-LDPC codes can be derived from Equation 7. First, it will be assumed $(dc/dv) \in \mathbb{Z}$. Considering each element inside the submatrices $H_i(t)$ as a CPM of size $(q-1)$ x $(q-1)$, as shown in Figure 12, we divide Equation 7 into several sub-equations, and the encoder can be defined by Equation 17.

$$v_{[0,t]} H^T_{[0,t]}(t) = [0_{[0,t]}, S_{t+1}] \tag{17}$$

where $0_{[0,t]}$ is a length-$(q-1)(t+1)$ zero vector. The length-$m_s(q-1)$ vector $S_{t,r}$, $0 \leq r < m_s$ is the partial syndrome from the t-th sub-equation and it is described by Equation 18.

$$S_t = [S_{t,0}, S_{t,1}, ..., S_{t,m_s-1}] \tag{18}$$

where $S_{t,r} = (S^{(0)}_{t,r}, S^{(1)}_{t,r}, ..., S^{(q-2)}_{t,r})$, $S^{(j)}_{t,r} \in GF(2)$, $0 \leq j < q-1$, $0 \leq r < m_s$.

The partial syndrome $S_{t+1,r}$ can be updated as given by Equation 19, which describes the operations done in step 3 in the encoding example of subsection 2.7.1.

$$S_{t+1,r} = \begin{cases} S_{t,r+1} + v_t H^T_{r+1}(t), & r = 0, 1, ..., m_s - 2 \\ v_t H^T_{r+1}(T), & r = m_s - 1 \end{cases} \tag{19}$$

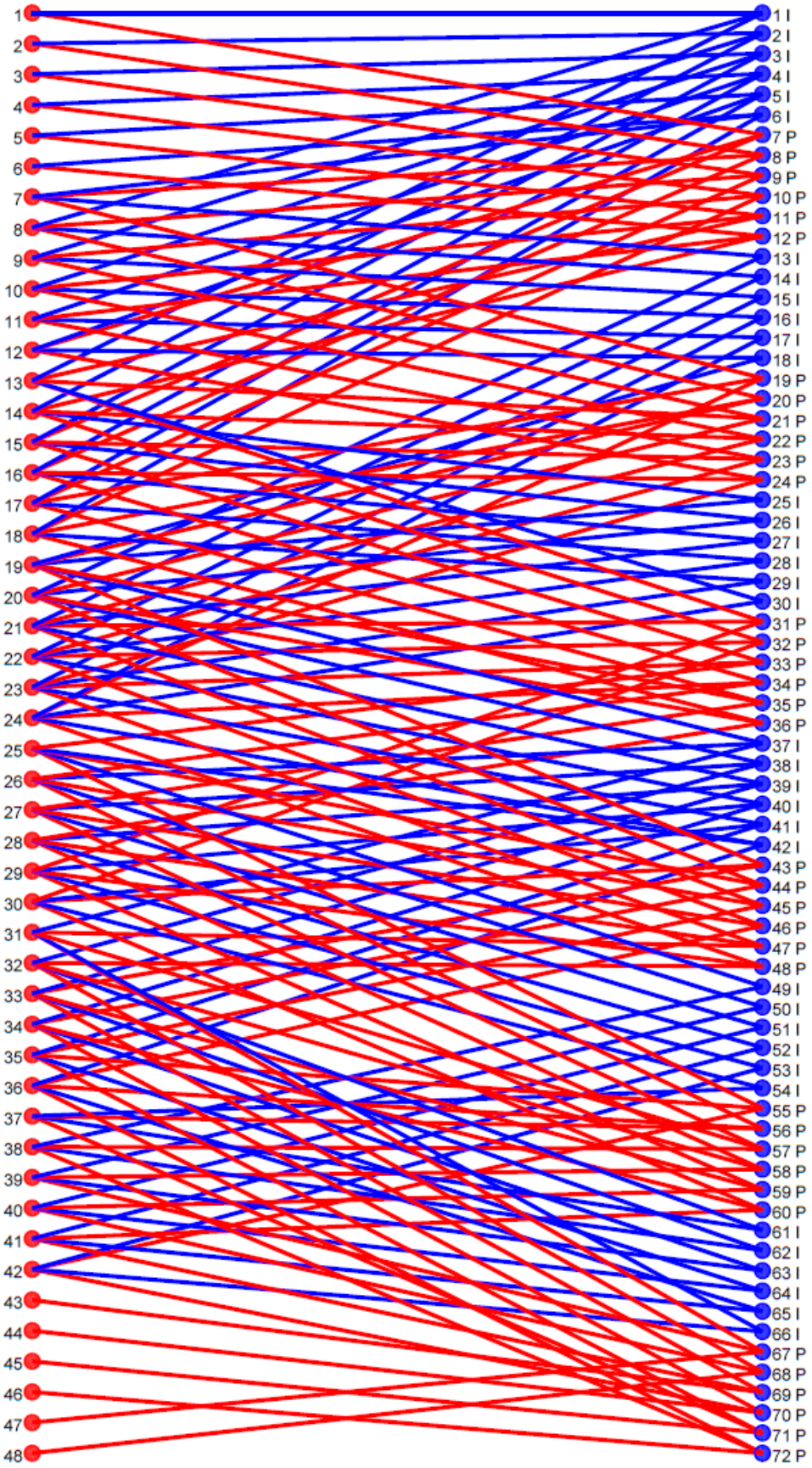
A systematic partial syndrome encoder can be obtained if any m rows of the sub-matrices $H^T_0(t)$ are linearly independent (TAVARES, 2010). The encoding equations are given by Equation 20, which describes the operations done in steps 1 and 2 for the encoding example of subsection 2.7.1.

$$v_{t,j} = \begin{cases} u_{t,j}, & j = 0, 1, ..., n-m-1 \\ S_{t,0} + \sum_{k=0}^{n-m-1} u_{t,k} h^{(0,k)}_0(t)^T, & j = n-1 \end{cases} \tag{20}$$

where for $t = 0$, $S_{t,0} = 0$.

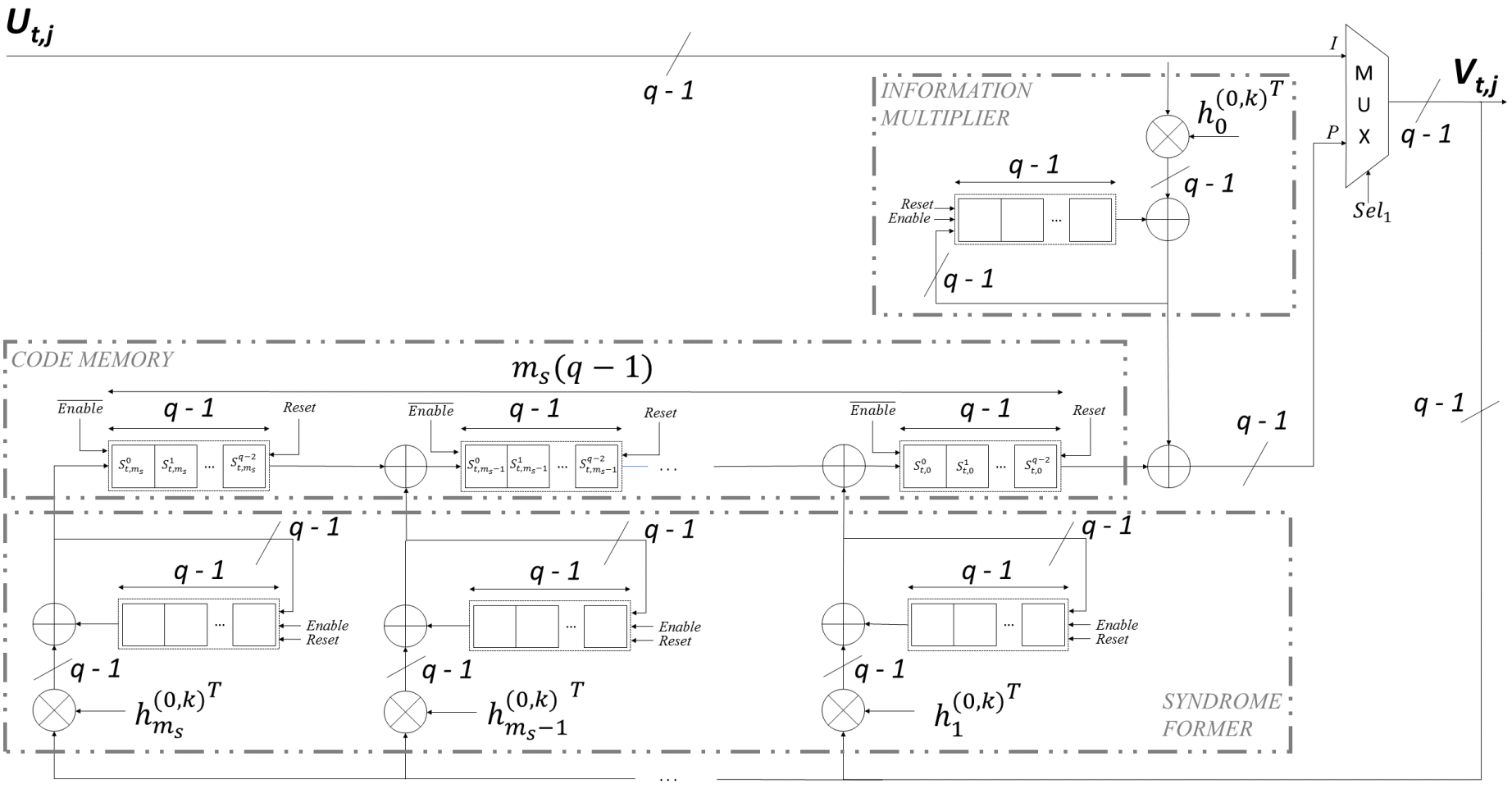
Considering the Equations 17, 19 and 20, the encoding circuit of Figure 14 can be obtained. The number of memory elements is $m_s m(q-1)$, which is less than the amount of memory used by a normal syndrome former encoder $(m_s n(q-1))$. The number of multiply-accumulate operations is exactly the same for both encoder types (TAVARES, 2010). For $(dc/dv) \in \mathbb{Z}$, there is only one sub-block of parity bits in each code block and m is always 1. Therefore, the size of all registers in Figure 14 is $(q-1)$, corresponding to the size of a CPM.

Figure 13 – Corresponding tanner graph representation of the SC-LDPC parity-check matrix of figure 12.



Source: the author.

Figure 14 – Partial-syndrome based encoder for SC-LDPC Codes



Source: the author.

The encoding process starts with mux input I selected and all registers are initialized with zeros. At time instant t, the incoming information sequence $u_{t,j}$ is sent to the output to compose the first $(n-m)(q-1)$ bits of the code block v_t, corresponding to the operation given by Equation 20 for $0 \leq j < n-m$. At the same time, $u_{t,j}$ is multiplied by the sub-matrices $h_0^{(0,j)}(t)^T$, represented in module *information multiplier* in Figure 14, and also by the sub-matrices $h_r^{(0,j)}(t)^T$, $1 \leq r \leq m_s$, represented in module *syndrome former* in Figure 14.

The module *information multiplier* in Figure 14 represents the multiplication of the incoming information blocks $u_{t,j}$, $0 \leq j < n-m$, by the CPMs corresponding to information inside the sub-matrix $H_0(t)$ in Equation 8. This operation corresponds to the sum in Equation 20 for $j = n-m$. The result of each multiplication $u_{t,j}h_0^{(0,k)}(t)^T$ is stored in a register of size $(q-1)$ to be summed with the result of the next multiplication $u_{t,j+1}h_0^{(0,k+1)}(t)^T$. This multiplication process occurs simultaneously in module *syndrome former* of Figure 14 to calculate the syndromes $S_{t+1,r} = u_{t,j}h_p^{(0,k)}(t)^T$, $1 \leq p \leq m_s$, $0 \leq r < m_s$ for encoding the next code block at time instant $t+1$, which corresponds to Equation 18.

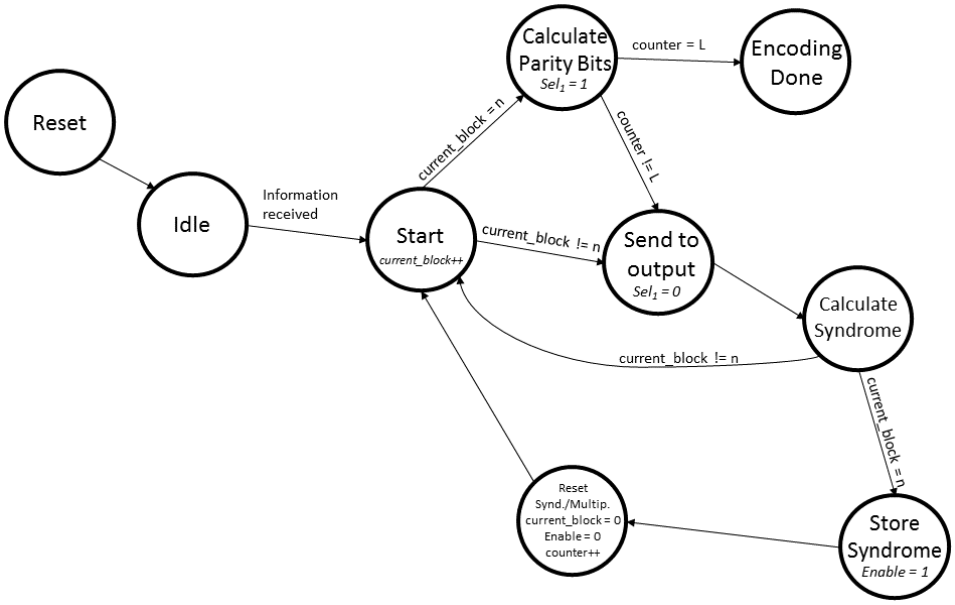
When the last multiplications in modules *information multiplier* and *syndrome former* are done, the mux input P is selected. Then, the rightmost $(q-1)$ memory elements in module *code memory*, which store the calculated syndrome at time instant $t-1$, are added to the product $u_t h_0^{(0,k)}(t)^T$ in order to obtain the last $(q-1)$ symbols of v_t. At this time, the encoder has already calculated Equation 20 for $j = n-m$. The newly obtained $v_{t,n-1}$ corresponding to parity bits is then multiplied by $h_1^{(0,n-1)}(t)^T, ..., h_{m_s}^{(0,n-1)}(t)^T$ in module *syndrome former* and the result of each multiplication is summed to the values stored in each register. This operation corresponds to the multiplication of the current encoded block v_t by the current syndrome former sub-matrices $H_1^T(t), ..., H_{m_s}^T(t)$ in Equation 8. Then, the registers in module *code memory* are enabled and all other registers outside this module are disabled. The results of each multiplication $v_t H_r^T(t)$ are used to update the contents of the registers in module *code memory*. The mux input I is selected again and all registers outside module *code memory* are reset to zero state. The same process is repeated at each time instant t to encode each code block v_t.

It is important to observe that all calculated syndromes at time instants before $t-1$ related to the current code block are also stored in the $(q-1)$ rightmost memory elements in Figure 14, since the calculation of the syndrome $S_{t,0}$ given in Equation 19 also includes the values of all previous syndrome former sub-matrices related to the current code block.

In the described encoding process the data flow and the control signals *Enable*, Sel_1 and *Reset* are controlled by the simplified Finite State Machine (FSM) shown in Figure 15. It has the generic *Idle* and *Reset* states and a state *Start* to start encoding

the received message. All control signals are equal to zero on the states where their values are not specified.

Figure 15 – Finite State Machine for the partial-syndrome based encoder.



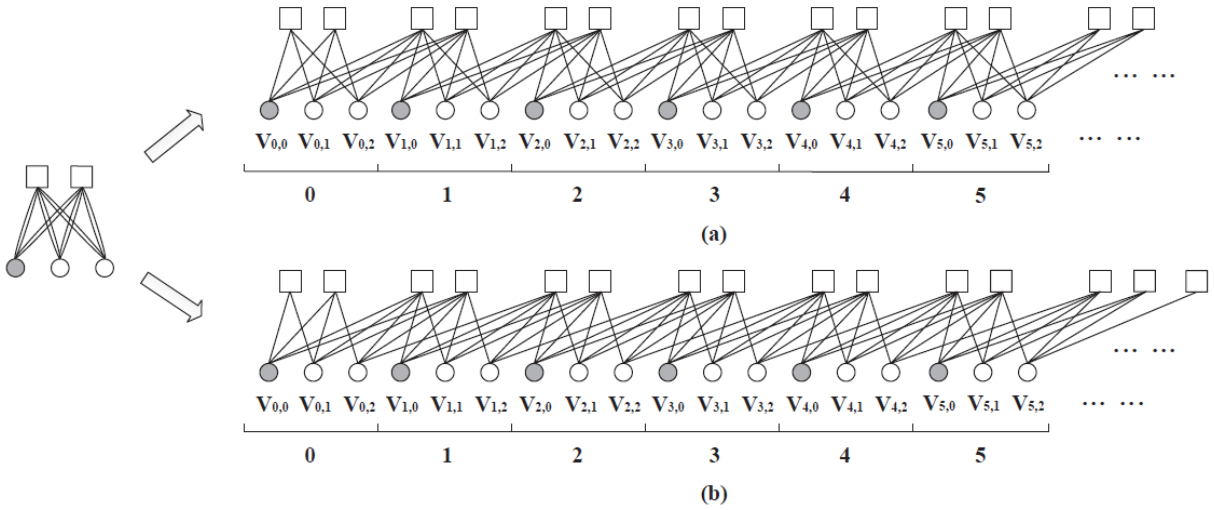
Source: the author.

3.2.1 Code Adaptation for Arbitrary Code Rates

To make use of the recursive encoder structure shown in Figure 14 the ratio (dc/dv) must be an integer. If this constraint is not satisfied, there will be more than one check nodes inside each protograph on the Tanner graph representation of the SC-LDPC parity-check matrix, as shown in Figure 16(a). For this example, $dc = 6$ and $dv = 4$ and each protograph contains two check nodes. The proposed solution for encoding non-integer ratio (dc/dv) SC-LDPC codes is to change the edge connections between the last $(m-1)(q-1)$ variable nodes and the check nodes inside each protograph, connecting them to the check nodes of its neighbour protographs (MA; SI; NIU, 2015), as shown on Figure 16(b).

In the SC-LDPC parity-check matrix this procedure corresponds to shifting one position down the last $(m-1)$ CPMs assigned to parity bits, as shown in Figure 17. The modified SC-LDPC code can be encoded with the same reduced-complexity encoder structure of Figure 14, but now there are $(m-1)(q-1)$ more elements involved in the calculation of the syndrome $S_{t+1,0}$ for encoding the next code block, due to the shifting on the columns of the last $(m-1)(q-1)$ parity elements as show in Figure 17.

Figure 16 – (a) Graph representation of a $(3, 6, L)$ SC-LDPC code and (b) Graph representation of a $(4, 6, L)$ with modified structure.



Source: (MA; SI; NIU, 2015)

3.2.2 Modified Partial-Syndrome Based Encoder for Encoding SC-LDPC Codes With Arbitrary Code Rates

A linear modified partial-syndrome based encoder for modified SC-LDPC codes can be derived from Equation 7. Considering each element inside the submatrices $H_i(t)$ as CPMs of size $(q-1)$ x $(q-1)$, we divide Equation 7 into several sub-equations, and the encoder can be defined by Equation 21.

$$v_t H_t^T(t) = [S_t, q_t] \tag{21}$$

The sub-matrix $H_t^T(t)$ is the structure inside the dashed ellipse in Figure 17. The length-$m(q-1)$ vector S_t is the partial syndrome from the t-th sub-equation, which describes the operations done in step 3 in the encoding example of subsection 2.7.1. q_t and S_t are given by Equations 22 and 23, respectively.

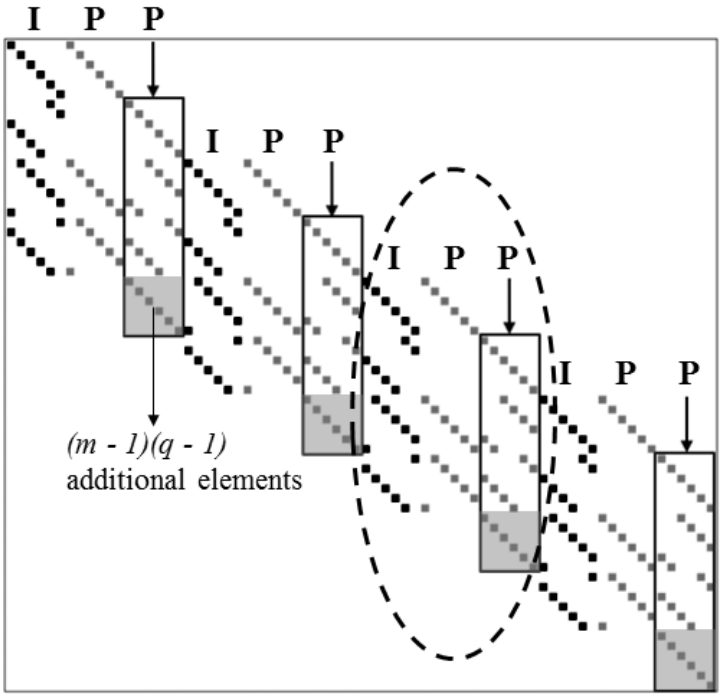
$$q_t = [q_{t,0}, q_{t,1}, ..., q_{t,m_s m-1}] \tag{22}$$

where $q_{t,j} = (q_{t,j}^{(0)}, q_{t,j}^{(1)}, ..., q_{t,j}^{(q-2)})$, $q_{t,j}^{(r)} \in GF(2)$, $0 \leq r < q-1$, $0 \leq j < m$ for integer ratio (dc/dv) or $0 \leq j < m_s m$ for non-integer ratio (dc/dv).

$$S_t = [q_{t-1,0}, q_{t-1,1}, ..., q_{t-1,m-1}] \tag{23}$$

A systematic partial syndrome encoder can be obtained if any m rows of the sub-matrices $H_0^T(t)$ are linearly independent (TAVARES, 2010). The encoding equations are given by Equation 24, which describes the operations done in steps 1 and 2 for the encoding

Figure 17 – (a) Parity-check matrix of an SC-LDPC code over $GF(7)$ with modified structure.



Source: the author.

example of subsection 2.7.1.

$$v_{t,j} = \begin{cases} u_{t,j}, & j = 0, 1, ..., n-m-1 \\ S_{t,0} + \sum_{k=0}^{n-m-1} u_{t,k} h_0^{(0,k)}(t)^T, & j = n-m \\ S_{t,j-n+m} + \sum_{k=0}^{n-m-1} v_{t,k} h_0^{(j-n+m,k)}(t)^T + \sum_{k=n-m}^{j-1} v_{t,k} h_0^{(j-n+m,k)}(t)^T, & n-m < j < n \end{cases} \tag{24}$$

where for $t = 0$, $S_{t,j-n+m} = 0$ for $j = n-m, ..., n-1$.

Considering the Equations 21, 23 and 24, the same encoding circuit of Figure 14 can be obtained for encoding SC-LDPC codes where the ratio $(dc/dv) \notin \mathbb{Z}$. The number of memory elements is $m_s m(q-1)$, which again is less than the amount of memory used by a normal syndrome former encoder ($m_s n(q-1)$). The number of multiply-accumulate operations is exactly the same for both encoder types (TAVARES, 2010).

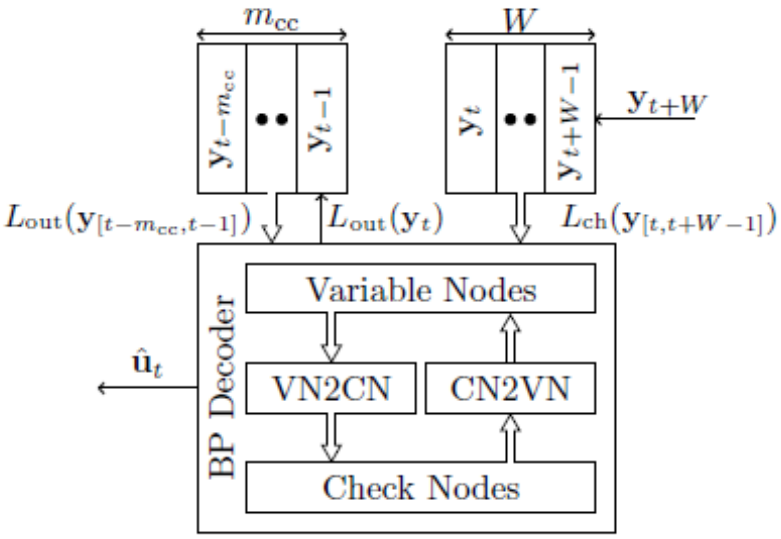
3.3 Window Decoder for SC-LDPC Codes

Due to the convolutional structure on the parity-check matrix of the SC-LDPC code, a latency constrained window decoder of size W can be derived. Considering the

blocks v_t and vt' in the chain of the coupled code, if $t' \geq (t+m_s+1)$ v_t and $v_{t'}$ do not share any parity-check equation (or check node). The window decoder exploits this property of the SC-LDPC parity-check matrix to define a decoder consisting of W received blocks, where W must include at least (m_s+1) blocks, as at least one block must be decoded on the window, the window must include all the parity-check equations related to the current block to be decoded.

In order to decode symbols in y_t, the decoder consists of W received blocks and m_s+1 previously decoded blocks stored in the decoder memory, where y_t is a sequence containing the incoming received channel values associated to the code block at time instant t. The decoder structure is shown on figure 18 as a shift register of size W that holds the channel values of $y[t, t+W-1]$, and a shift register of size m_s corresponding to the decoder memory, which holds the LLR values of the m_s previously decoded blocks.

Figure 18 – Window decoder of size W for SC-LDPC codes.



Source: the author.

On the SC-LDPC parity-check matrix the encoding process is equivalent to running a *belief-propagation* (BP) algorithm on a section of $W*m$ rows and $W*n$ columns of the parity-check matrix $H_{SC,doubly(r,r,m,n)}$, which includes W coupled blocks. Hence, the BP decoder has $Mw = Wm$ check nodes and $Nw = Wn$ variable nodes. The modules VN2CN and CN2VN defines the connections from Nw variable to Mw check nodes and vice versa.

3.3.1 Decoding Algorithm

Let v_k, $k = 1, ..., N_w$ and c_j, $j = 1, ..., M_w$ denote Nw variable and Mw check nodes within a window. Then $L_{ch}(v_k)$ denotes the input channel LLR for variable node v_k and $N(c_j)$ represents the set of variable nodes connected to the check node c_j. The message from variable node v_k to the check node c_j in $n'th$ iteration is denoted as $L^{(n)}_{v_j,c_j}$.

1. *Initialization*: in the initialization, all variable nodes k sends $L_{ch}(v_k)$ to all its connected check nodes, as given by equation 25.

$$L^{(0)}_{v_k,c_j} = L_{ch}(v_k) \tag{25}$$

2. *Iterative process*: on the following iterations, the extrinsic $LLRs$ from the variable node v_k to the check node c_j are calculated as given by equation 26:

$$L^{(n)}_{v_k,c_j} = L_{ch}(v_k) + \sum_{l \in N(v_k) \setminus c_j} L^{(n-1)}_{c_l,v_k} \tag{26}$$

$L^{(n)}_{c_j,v_k}$ denotes the message from check node c_j to the variable node v_k. $N(v_k) \setminus c_j$ represents the set of check nodes connecting to v_k excluding c_j. The exclusion of c_j precludes the information received by v_k from c_j to be reused to calculate the message L_{v_k,c_j}. Similarly, the extrinsic $LLRs$ from the check node c_j to the variable node v_k are calculated as given by equation 27.

$$L^{(n)}_{v_k,c_j} = 2tanh^{-1}\left(\prod_{l \in N(c_j) \setminus v_k} tanh\left(\frac{L^{(n-1)}_{v_l c_j}}{2}\right)\right) \tag{27}$$

Similarly to equation 26, the calculation of $L^{(n)}_{c_j,v_k}$ excludes the variable node v_k. The window at time instant t decodes the symbols in the received block y_t only, and hence they are called target symbols. The BP algorithm is applied to the nodes within the window only, however, due to the memory of the coupled code, a read access to the m_s previously decoded block is also required.

3. *After I_{max} Iterations*: the iterative process continues until the maximum number of iterations I_{max} is reached. The output LLR $L_{out}(v_k)$ for variable node v_k is then computed as given by equation 28.

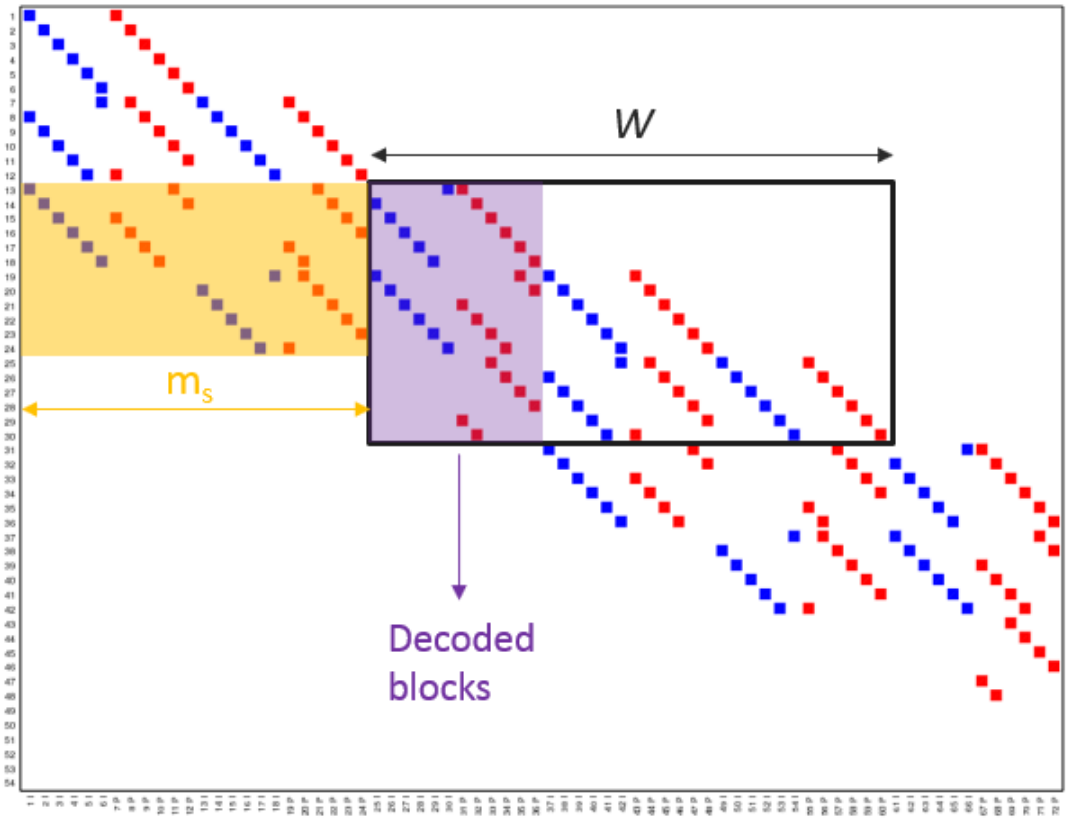
$$L_{out}(v_k) = L_{ch}(v_k) + \sum_{l \in N(v_k)} L^{I_{max}}_{c_l,v_k} \tag{28}$$

The output $LLRs$ corresponding to the target symbols $L_{out}(y_t)$ are then fed back to the memory shift register, as shown in figure 18. Once the target symbols are decoded, c new channel values corresponding to $y_t + w$, enter the window decoder from right and

the estimates of the information bits u_t leaves the decoder from left. The decoding process continues until all code blocks are decoded.

Figure 19 illustrates the decoding process done from steps 1 to 3 on the SC-LDPC parity-check matrix of figure 12 for a window at time instant t and its corresponding code block to be decoded.

Figure 19 – Decoding process for the SC-LDPC parity-check matrix of figure 12.



Source: the author.

4 Results and Discussion

In this section, the code characteristics of the SC-LDPC code constructed by the algebraic SP-method are analysed and a comparison of its parameters to constraints of the proposed encoder/decoder structure is done.

As one result of this work, a very flexible tool for constructing parametrizable SC-LDPC codes was implemented in *Octave*[6] allowing the construction of SC-LDPC codes with arbitrary parameters, such as code length, code rate and memory size. In this code construction method, each non-binary entry of the SC-LDPC base matrix is dispersed by a binary CPM of size $(q-1)$ x $(q-1)$, where q is the size of the finite field. Hence, this matrix dispersion corresponds to a protograph expansion on the corresponding tanner graph of the SC-LDPC code and the size of the field plays an important role on the code length of the SC-LDPC code. As the code distance of the SC-LDPC code shows a linear growth with the code length, large codes are desirable. The code length can also be increased by copying the coupled LDPC chain as many times as desirable, but in this case, the memory size of the code will still have the same size of the smaller code before the copying process, as the memory is defined to be the $gcd(d_c, d_v)$ and these two parameters are constant on this copying operation.

For the implemented encoder/decoder structures, changing the ratio d_c/d_v will increase or decrease the encoder memory size and both the decoder memory and window sizes. It also changes the amount of connections of the decoder modules VN2CN/CN2VN. The number of check and variable nodes of each block on the SC-LDPC parity-check matrix (or of each protograph on its corresponding tanner graph representation) is $n = d_c/m_s$ and $m = d_v/m_s$, respectively. The code rate of the code is $R = (n \breve{} m)/n$, $(m < n)$. Therefore, all parameters depends on the check node degree d_c and the variable node degree d_v and only by choosing these two parameters and the code length L, an SC-LDPC code satisfying any specifications can be constructed.

It is also important to mention that for constructing regular SC-LDPC codes using the SP-method, the zero elements of $GF(q)$ lying on the main diagonal of the doubly base matrix $B_{q,sp,doubly(m,n)}$ must be avoided . This can be easily done by changing the value of the shifting parameter η in equation 10. The code regularity makes the structure of the decoder modules CN2VN/VN2CN to be regular.

The SC-LDPC code convergence was verified through the proposed partial-syndrome encoder and the window decoder, both implemented in Python. As the SC-LDPC code performance for specific channel characteristics is very dependent on the decoder structure and decoding algorithm, the code performance for specific communication scenarios was

not evaluated in this project. In fact, the proposed window decoder requires approximately 50 iterations to achieve convergence. Therefore, the sliding window operation as well the decoding algorithm must be first better exploited to evaluate the code performance for specific scenarios.

5 Conclusion

In this project, a very flexible tool for constructing parameterizable SC-LDPC codes was constructed by making use of a new algebraic construction method, called superposition. This tool gives the possibility of constructing SC-LDPC codes matching any communication channel specifications.

Based on the parity-check matrix cyclic structure of the developed SC-LDPC code, reduced-complexity encoder and decoder were derived. The implemented partial syndrome encoder has reduced memory size in comparison to the classic syndrome former encoders. The partial-syndrome encoder found on the literature has constrained rate and to get around this requirement a modified SC-LDPC code was implemented, making the SC-LDPC code encodable with the same partial syndrome encoder structure for arbitrary rates. The thresholds of the modified SC-LDPC code for rate adaptation seems to outperform the original SC-LDPC code.

The implemented window decoder runs the decoding algorithm only in a smaller portion of the SC-LDPC code, providing a way to decode arbitrarily large SC-LDPC codes and it is also suitable for continuously decode the input blocks. Since the decoding algorithm is applied only to a smaller number of received blocks, the memory and hardware requirements for the window decoder are much less compared to classic block-wise decoding schemes of SC-LDPC codes. In general, the storage requirements for the window decoder reduces by a factor of L/W compared to block-wise decoders.

6 Future Work Suggestions

Having in mind the ever-growing necessity for better communication performance, the implemented algebraic method for constructing SC-LDPC codes guarantees an organised way for constructing SC-LDPC codes that satisfy some requirements for achieving good communication performance, such as a girth of at least 6. Other techniques for increasing the girth, of the code such as masking techniques applied to the SC-LDPC base matrix, are worth of further investigation. It's also worth of investigation the evaluation about how the coupling factor between two coupled code blocks affects the code performance. A comparison between the properties of the SC-LDPC codes constructed by the SP-method and the properties of SC-LDPC codes constructed based on heuristic methods is also worth of investigation.

References

COSTELLO, D. J. et al. Spatially coupled sparse codes on graphs: Theory and practice. *IEEE, Commun. Mag.,*, v. 52, n. 7, p. 168–176, July 2014. Cited 3 times on pages 12, 13, and 19.

DIAO, Q. et al. A matrix-theoretic approach for analysing quasi-cyclic low-density parity-check codes. *IEEE Transactions Information Theory*, n. 58, p. 4030 – 4048, 2012. Cited on page 29.

ELIAS, P. Coding for noisy channels. *IRE Convention Record*, n. 4, p. 37 – 46, 1955. Cited on page 17.

FELSTRÖM, A. J.; ZIGANGIROV, K. Time-varying periodic convolutional codes with low-density parity-check matrix. *Information Theory, IEEE Transactions on*, v. 45, n. 6, p. 2187 – 2191, set. 1999. Cited on page 12.

GALLAGER, R. G. Low density parity check codes. *The Bell System Technical Journal*, 1962. Cited on page 12.

HAMMING, R. W. Error detecting and error correcting codes. *Bell System Technical Journal*, v. 29, n. 2, p. 147 – 160, Apr 1950. Cited on page 17.

JOHNSON, S. Introducing low-density parity-check codes. *Cambridge University*, 2010. Cited on page 29.

LI, J. et al. *LDPC Code Designs, Constructions, and Unification*. [S.l.]: Cambridge University Press, 2016. Cited on page 53.

LI, J. et al. Algebraic quasi-cyclic ldpc codes: Construction, low error-floor, large girth, and reduced-complexity decoding scheme. *IEEE Trans. Commun.*, v. 62(8), p. 2626–2637, 2014. Cited on page 30.

LI, J. et al. Decoding of quasi-cyclic ldpc codes with section-wise cyclic structure. *Proc. IEEE Inf. Theory Applic. Workshop. San Diego, CA, USA*, p. 1–10, 2014. Cited on page 29.

LIU, K.; LIN, S.; ABDEL-GHAFFAR, K. A revolving iterative algorithm for decoding algebraic cyclic and quasi-cyclic ldpc codes. *IEEE Transactions on Communications*, v. 61, p. 4816–4827, 2013. Cited on page 29.

LUBY, M. et al. Improved low-density parity-check codes using irregular graphs. *Information Theory, IEEE Transactions on*, v. 47, n. 2, p. 585–598, fev. 2001. Cited on page 12.

MA, J.; SI, Z.; NIU, K. Recursive encoding of spatially coupled ldpc codes with arbitrary rates. *IEEE 26th International Symposium on Personal, Indoor and Mobile Radio Communications*, p. 127 – 131, 2015. Cited 2 times on pages 38 and 39.

MITCHELL, D. G. M.; LENTMAIER, M.; COSTELLO D. J., J. Spatially coupled LDPC codes constructed from protographs. *Information Theory, IEEE Transactions on*, v. 61, n. 9, p. 4866 – 4889, set. 2015. Cited on page 22.

MOON, T. K. *Error Correction Coding Mathematical Methods and Algorithms*. [S.l.]: Wiley, 2005. Cited on page 51.

SHANNON, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, 1948. Cited on page 16.

TAVARES, M. B. S. On low-density parity-check convolutional codes: Constructions, analysis and vlsi implementations. *Jörg Vogt Verlag*, 2010. Cited 3 times on pages 34, 39, and 40.

THORPE, J. Low density parity check (LDPC) codes constructed from protographs. *The Interplanetary Network Progress Report*, v. 42, n. 154, p. 1–7, August 2003. Cited on page 12.

Appendix

APPENDIX A – Galois Field Arithmetic

Finite field arithmetic is widely used in a variety of applications such as codification and cryptography algorithms. It differs from standard arithmetic for the fact that there is a limited number of elements in that field, and every operation performed over it, results in an element within. A finite field with q elements is also called Galois Field – $GF(q)$, in honor to the founder of finite field theory, Variste Galois. We may see that for the case of a $GF(2)$, the addition will result in an exclusive OR (XOR) operation and the multiplication in an AND operation. The content of this section is based on the work of (MOON, 2005).

A finite field $GF(q)$ is a set of q objects on which the operations of addition and multiplication, subtraction (or additive inverse), and division (or multiplicative inverse) apply in a manner analogous to the way these operations work for real numbers.

In particular, the addition operation + and the multiplication operation . (or juxtaposition) satisfy the following:

1. Closure under addition: for every a and $b \in GF(q)$, $(a + b) \in GF(q)$.
2. Additive identity: there is an element in $GF(q)$, which we denote as 0, such that $a + 0 = 0 + a = a$ for every $a \in GF(q)$.
3. Additive inverse(subtraction): for every $a \in GF(q)$, there exists an element $b \in GF(q)$ such that $a + b = b + a = 0$. The element b is frequently called the additive inverse of a and is denoted as $-a$.
4. Associativity: $(a + b) + c = a + (b + c)$ for every $a, b, c \in GF(q)$.
5. Commutativity: $a + b = b + a$ for every $a, b \in GF(q)$.

The first four requirements mean that the elements of $GF(q)$ form a group under addition; with the fifth requirement, a commutative group is obtained.

6. Closure under multiplication: for every a and $b \in GF(q)$, $a.b\ in GF(q)$.
7. Multiplicative identity: there is an element in $GF(q)$, which we denote as 1, such that $a.1 = 1.a = a$ for every $a \in GF(q)$, for $a \neq 0$.
8. Multiplicative inverse: for every $a \in GF(q)$ with $a \neq 0$, there is an element $b \in GF(q)$ such that $a.b = b.a = 1$. The element b is called the multiplicative inverse, or reciprocal, of a and is denoted as a^{-1}.

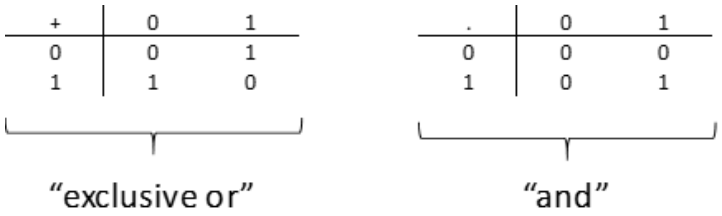
9. Associativity: $(a.b).c = a.(b.c)$ for every $a, b, c \in GF(q)$.

10. Commutativity: $a.b = b.a$ for every $a, b \in GF(q)$.

11. Multiplication distributes over addition: $a.(b + c) = a.b + a.c$.

Thus, the non-zero elements of $GF(q)$ form a commutative group under multiplication. As an example, the finite field with two elements in it $GF(2)$ has the following addition and multiplication tables:

Figure 20 – (a) Addition and multiplication tables for $GF(2)$.



Source: the author.

The field $GF(2)$ is very important to our work, since it is the field where the operations involved in binary codes work.

The non-zero elements of a finite field form a cyclic group, so all non-zero elements can be expressed as powers of a single element, called a primitive element of the field. In general, there will be several primitive elements for a given field. Let α be a primitive element of $GF(q)$. Then, the powers of $\alpha(\alpha^0 = 1, \alpha, \alpha^2, \ldots, \alpha^{q-2})$ give all the nonzero elements of $GF(q)$. Commonly, the 0-element of $GF(q)$ is represented by $\alpha^{-\infty}$, i.e., $0 = \alpha^{-\infty}$.

APPENDIX B – Circulant Permutation Matrix Dispersion

If the row weight (or column weight) of a regular matrix A is w, we say that A has weight w and call it a weight-w regular matrix. For $w \neq 0$, a weight-w regular matrix must be a square matrix. A binary weight-1 regular matrix is called a *permutation matrix* (PM). It is clear that a weight-w regular matrix is the sum of w disjoint PMs.

A square matrix over $GF(q)$ is called a *circulant* if every row of the matrix is the cyclic-shift of the row above it one place to the right, and the top row of the matrix is the cyclic shift of the last row one place to the right. It is clear that a circulant is a regular matrix whose weight equals the weight of its top row. The top row of a circulant is called the generator of the circulant. A binary circulant of weight 1 is a binary circulant permutation matrix (CPM). It is clear that a weight-w circulant is the sum of w disjoint CPMs.

In many constructions of binary QC LDPC codes, a nonzero element in $GF(q)$ is represented by a binary CPM. Let α be a primitive element of $GF(q)$. For $0 \leq j \leq q-1$, we represent the element α^j by a binary CPM, denoted by $A(\alpha^j)$, of size $(q-1)x(q-1)$, whose generator has the unit-element "1" of $GF(q)$ as its single nonzero component at position j. It is clear that the representation of the element α^j by the binary $CPM(\alpha^j)$ of size $(q-1)x(q-1)$ is unique and the mapping between α^j and $A(\alpha^j)$ is one-to-one. This matrix representation of α^j is referred to as the binary CPM-dispersion of α^j. Note that the size of the binary CPM $A(\alpha^j)$ is the order of α. The 0-element of $GF(q)$ is represented by a $(q-1)x(q-1)$ ZM (LI et al., 2016).