
Politecnico di Torino



Master's Degree in
Communications and Computer Networks Engineering

Master's Thesis

# Channel Coding for Massive IoT Satellite Systems

*Academic supervisors*:

***Roberto GARELLO***, Politecnico di Torino

***David GESBERT***, EURECOM

*Author*:

***Riccardo SCHIAVONE***

*Company supervisor*:

***Gianluigi LIVA***, German Aerospace Center (DLR)

Academic Year 2020/2021

# Abstract

Concatenation of linear block codes enhances the performance with respect to the use of single linear block codes, while increasing the decoding complexity. In particular, the serial concatenation of linear block codes has proven to be an important tool for the design of powerful binary linear short codes. This was decisive for the discovery of new codes capable of approaching the bounds of the best possible codes at finite length. The class of convolutional codes, upon termination, are known to provide very powerful codes for large memory values at the encoder. However, the decoding complexity of these codes grows exponentially with their encoding memory, preventing the practical use of very large memories. The serial concatenation of convolutional codes with outer (cyclic redundancy check) linear block codes can reduce the needed memory of the convolutional code to achieve very good performances, at the expense of using a list decoding scheme. In this Master thesis we study the performance of such concatenation via weight enumerator and the performances of the serial-list and the parallel-list Viterbi algorithm as decoding algorithms with bounded list sizes. Lastly, we propose a suitable sorting network capable of reducing the time complexity of the parallel-list Viterbi algorithm when implemented in hardware.

# Aknowledgments

Se queste pagine si possono leggere è grazie a un lungo percorso che ha coinvolto diverse persone che in qualche modo hanno contribuito a quest'opera.

Un grazie va senza dubbio a Roberto che mi ha ha dato la possibilità di conoscere Gianluigi ed il suo gruppo di lavoro a DLR a Monaco di Baviera. Un gruppo di lavoro con il quale ho scambiato e scambio ancora idee su molti argomenti e grazie a loro non smetto mai di imparare.

Gianluigi è stato una fortuna poterlo conoscere, sia dal punto di vista umano oltre che "lavorativo", e specie in questo periodo storico dove siamo tutti "vicini ma lontani", l'aspetto umano gioca un ruolo ancora più importante nelle nostre vite.

Oltre a Gianluigi ringrazio anche Mustafa, dottorando di Gianluigi, che ho potuto conoscere a TUM e che ha reso molte giornate leggere grazie alle sue battute sempre pronte e i suoi molti consigli. E poi c'è Schessi, che devo veramente ringraziare. Sono stati 6 mesi in Germania che potevano risultare molto pesanti per via di lockdown e restrizioni varie messe in atto per tutelare la salute di tutti e tutte in questa pandemia globale, eppure se non lo sono stati è sopratutto anche grazie a Schessi con la quale abbiamo scambiato molte birre, pranzi e cene, musica, risate, idee e tanti tanti sorrisi. Un'amica prima ancora che una collega.

Ringrazio anche Edo, Pietro, Niki e Teo per quei momenti del loro tempo libero che abbiamo condiviso insieme in quella grande città che è Monaco di Baviera e delle quali tengo ancora tutte le foto e tutti i video di molti momenti passati insieme.

Ma prima di arrivare in Germania, il mio percorso è passato da altri due Paesi, l'Italia e la Francia dove sono state molte altre le persone che ho potuto conoscere e con le quali ho potuto condividere una parte più o meno grande di quei lunghi periodi. Grazie a quelle persone che ho conosciuto ad EURECOM tra studenti e non, con cui abbiamo condiviso lo stress ed i tanti momenti felici. Da Pierina, a Viola, a Matte, s Giuse, Andre e molte altre bellissime persone. Spero ci rivedremo un giorno di nuovo sulla spiaggia di Juan o di Antibes a guardare il mare, ballare fino al calare del sole e bere quelle birre che anche se non fresche sapevano di semplice felicità.

Un grazie in Italia ai compagni e alle compagne del Poli e agli amici e alle amiche della Croce Rossa di Giaveno e di Arcigay Torino. Dagli amici di sempre come Mark, Meli, e Michi, a quelli nuovi come Ali, Toschina, Marcolino e Riccardino, a Matte, Oli,

Andre e Barbara e molte molte altre belle persone.

Un grazie a mio padre e mia madre, che seppur non sempre riusciamo a dircelo o dimostrarcelo, ci vogliamo bene.

Ed un grazie agli amici e alle amiche di Trento da Peterle alla Lina a Sara a Enrica e a tutte quelle anime che incontri al Bar Nuovo Paradiso.

Ed infine grazie a Shamy che ha colorato ancora di più questi quasi tre anni nonostante la distanza di questi viaggi. Un fiorellino tutto colorato, un raggio di sole caldo, una melodia che sa di festa, un bacio che sa di felicità, un abbraccio fin dentro l'anima.

"*Non so più dove metterla, la mia felicità, è talmente grande da diventare ingombrante. La sento debordare in un sorriso che non saprei spegnere nemmeno se volessi.*" (Samantha Cristoforetti, *Diario di un'apprendista astronauta*)

# Contents

# List of Tables

# List of Figures

# Acronyms

**AWE** average weight enumerator. 23, 24, 26

**bi-AWGN** binary input additive white Gaussian noise. 4, 11, 34

**BPSK** binary phase shift keying. 11

**CC** convolutional code. 2, 3, 5–13, 15, 21–25, 37–39

**CER** codeword error rate. 14–16, 23

**CRC** cyclic redundancy check. 2, 21–23, 25–27, 30, 34–36, 38, 39

**FSM** finite state machine. 5

**IoT** Internet-of-Things. 1

**LVA** list Viterbi algorithm. 2, 17, 30

**ML** maximum likelihood. 12, 13, 23, 25, 27, 30

**P-LVA** parallel-list Viterbi algorithm. 2, 17–20, 25, 27, 30, 31, 33, 34, 36

**RCU** random coding union bound. 15

**S-LVA** serial-list Viterbi algorithm. 17–20, 25, 27, 31, 34, 38, 39

**TBCC** tail-biting convolutional code. 8–10, 12, 13, 15–17, 20, 23, 25, 27, 30, 31, 34–39

**TSB** tangential sphere bound. 23, 25, 26

**UB** union bound. 4

**VA** Viterbi algorithm. 3, 9, 12, 13, 17–19, 27, 34

**WAVA** wrap-around Viterbi algorithm. 13, 15, 27, 34

**WE** weight enumerator. 4, 7, 9, 22, 23

**ZTCC** zero-tail terminated convolutional code. 7–10, 12, 18, 20, 23

# Notation

| | |
|---|---|
| $C$ | channel capacity |
| $R$ | code rate |
| $\nu$ | memory of a convolutional code |
| $m$ | redundancy of the CRC code |
| $A_d$ | weight enumerator |
| $P_B$ | block error probability |
| $k$ | number of information bits |
| $K$ | length of the information sequence of a terminated convolutional code |
| $n$ | number of transmitted bits |
| $\mathcal{C}$ | code book |
| $\mathbf{u}$ | information bits |
| $\mathbf{c}$ | codeword $\in \mathcal{C}$ |
| $\mathbf{G}$ | generator matrix of a code |
| $\mathbf{H}$ | parity-check matrix of a code |
| $\mathbb{F}_q$ | $q$-ary vector space |
| $d_{\min}$ | minimum distance of a code |
| $d_H$ | Hamming distance between two vectors |
| $d_E$ | Euclidean distance between two vectors |
| $w_H$ | Hamming weight of a vector |
| $L$ | list size |

*When you discover new things every minute and your mind is absorbing so many experiences, it feels like time expands.*
SAMANTHA CRISTOFORETTI

# Introduction

INTERNET-OF-THINGS (IoT) was "born" sometime between 2008 and 2009 as reported by CISCO in [Evans, 2011], under the idea of enabling internet access to electrical and electronic devices [Miraz et al., 2015], thus allowing them to collect and exchange data. Since its introduction, the number of connected devices has managed to surpass the number of humans connected to the internet [Evans, 2011] and it is expected to have 12.3 billions of IoT devices sold by 2022 [CISCO, 2019].

The increasing number of both mobile and embedded IoT devices has led to a sensors-rich world, capable of addressing a various number of real-time applications, such as security systems, healthcare monitoring, environmental meters, factory automation, autonomous vehicles and many others [Al-Fuqaha et al., 2015].

Satellite communications are networks capable of providing coverage in remote and arduous areas [De Sanctis et al., 2015], thus enabling to extend the use of IoT devices in those areas.

Most of the IoT devices, especially sensors and meters, are characterized by their low cost, their sporadic exchange of small data packets with essential information, their limited processing power, small storage capabilities. Furthermore, they are powered with batteries of limited capacity, which are required to last over 10 years without being replaced or recharged [GSMA, 2018]. All these important constraints have to be taken into account in the design of the communication modules for such devices.

The communication on the uplink channel of satellite IoT networks, the one where the IoT device is the transmitter which sends data to the satellite, is the most crucial part in this networks, because it affects the power consumption of the embedded device.

Claude Shannon in 1948 showed in his paper "A mathematical theory of communications" [Shannon, 1948] that it is possible to transmit information through a noisy channel with an arbitrarily small error probability by using a code with a rate $R$ smaller than the channel capacity $C$, the largest possible transmission rate of the given channel. However, his theorem did not tell us how to construct such capacity-achieving codes. Moreover, his theory assumes sufficiently large code word length.

Since then, a lot of work has been done, and nowadays, for large blocklength codes, well-established tools exist to approach channel capacity with a reasonable decoding complexity. On the other hand, for codes in the short blocklength regime, where the

amount of bits is “very small” (*e.g.,* $k$ = 64 information bits), such tools are not that effective and the design of good codes and decoding schemes is mainly driven by intuition [Liva, 2014].

The class of convolutional codes (CC), introduced by Elias in 1955 [Elias, 1955], are known to provide, upon termination, powerful short codes, for which an optimum and low complexity maximum-likelihood decoding algorithm was proposed by Viterbi in 1967 [Viterbi, 1967].

Recently [Lou et al., 2015; Yang et al., 2018a,b; Liang et al., 2019], a serial concatenation of CCs with outer cyclic redundancy check (CRC) codes was proposed, which can be decoded with a list Viterbi algorithm (LVA) to approach the decoding bounds on the block error probability of the best short code of a given rate.

This thesis is structured as follows.

In Chapter 1 we review necessary coding theory aspects about binary linear codes and convolutional codes, from their graphical representations to some decoding aspects.

Chapter 2 gives a review of an important family of convolutional code list decoders.

The design of serial concatenations of CCs with outer CRC codes is detailed in Chapter 3, with numerical results for various CCs and CRC codes.

A hardware friendly proposal for a fast implementation of a parallel-list Viterbi algorithm is presented in Chapter 4.

We summarize and conclude this contribution in the last chapter of this thesis, while in the appendix are shown some tables with the CRC codes found for various memories of the CCs and parameters of the CRC codes.

The algorithms used to design and to decode such codes are made publicly available at *github.com/rickyskv/Serially_concatenated_CC_with_CRC*.

# 1. Convolutional Codes

BINARY convolutional codes (CCs) are binary linear codes introduced by Elias in 1955 [Elias, 1955]. In this chapter we introduce binary linear codes and their properties in Section 1.1, we then introduce the properties and encoders of convolutional codes in Section 1.2. Finally, in Section 1.3 we present their trellis structure and the Viterbi algorithm (VA) for decoding.

## 1.1 Binary Linear Codes

**Definition 1.1.** An $(n, k)$ *binary linear code* is a $k-$dimensional subspace of the $n-$dimensional vector space $\mathbb{F}_2^n$.

An $(n, k)$ binary linear block code $\mathcal{C}$ is usually defined by its generator matrix $\mathbf{G}$, a $k \times n$ binary matrix. A $k-$bits long message $\mathbf{u}$ encoded by $\mathbf{G}$ produces a binary codeword $\mathbf{c}$ of length $n$,

$$\mathbf{c} = \mathbf{u}\mathbf{G}. \tag{1.1}$$

Alternatively, a binary linear block code can be defined by its $(n-k) \times n$ *parity check matrix* **H** as

$$\mathbf{c}\mathbf{H}^T = \mathbf{0} \tag{1.2}$$

where $\mathbf{0}$ is the all-zero vector.

The rows of $\mathbf{H}$ span the subspace orthogonal to $\mathcal{C}$ and we denote such a subspace by $\mathcal{C}^\perp$. The vectors in $\mathcal{C}^\perp$ form a $(n, n-k)$ binary linear code called the *dual code*.

For instance, consider the $(5,3)$ code and a 3-bit long message $\mathbf{u} = (0\,1\,1)$, then the corresponding codeword is defined in this way:

$$\underbrace{(0\,1\,1)}_{\mathbf{u}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}}_{\mathbf{G}} = \underbrace{(0\,1\,1\,0\,1)}_{\mathbf{c}}$$

**Definition 1.2.** Given a vector $\mathbf{v} \in \mathbb{F}_2^n$, its *syndrome* w.r.t. $\mathbf{H}$ is given by

$$\mathbf{s} = \mathbf{v}\mathbf{H}^T. \tag{1.3}$$

By definition $\mathbf{s}$ is $\mathbf{0}$ if and only if $\mathbf{v} \in \mathcal{C}$.

**Definition 1.3.** Given a binary vector $\mathbf{v} \in \mathbb{F}_2^n$, its *Hamming weight* $w_H(\mathbf{v})$ corresponds to the number of non-zero entries in the vector. The *Hamming distance* between two binary vectors is defined to be the Hamming weight of the sum over the binary field of the two vectors,

$$d_H(\mathbf{v_1}, \mathbf{v_2}) = w_H(\mathbf{v_1} + \mathbf{v_2}). \tag{1.4}$$

An important characterization of a code is its minimum distance $d_{\min}$, because it gives some information about the error detection and error capabilities of such code.

**Definition 1.4.** The *minimum distance* $d_{\min}$ of a binary linear code $\mathcal{C}$ is the minimum Hamming distance between any two distinct codewords in $\mathcal{C}$. Due to the linearity, it corresponds to the smallest Hamming weight among all codewords in $\mathcal{C}$, except the allzero codeword,

$$d_{\min}(\mathcal{C}) = \min_{\substack{\mathbf{c} \neq \mathbf{c}' \\ \mathbf{c}, \mathbf{c}' \in \mathcal{C}}} d_H(\mathbf{c}, \mathbf{c}') = \min_{\substack{c \neq \mathbf{0} \\ \mathbf{c} \in \mathcal{C}}} w_H(\mathbf{c}). \tag{1.5}$$

**Theorem 1.1.** *A linear code can detect all binary error vectors with Hamming weight up to $d_{min} - 1$, and all binary vector errors which do not correspond to codewords in $\mathcal{C}$. The same code can correct at least all errors with Hamming weight $\leq t$, with $t$ given by:*

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor. \tag{1.6}$$

A more complete characterization of a linear block code is its *weight enumerator* (WE) or *distance spectrum*.

**Definition 1.5.** The weight enumerator of an $(n, k)$ linear block code $A_d(\mathcal{C})$ corresponds to the multiplicity of all codewords in $\mathcal{C}$ with Hamming weight equal to $d$,

$$A_d(\mathcal{C}) = |\{\mathbf{c} \in \mathcal{C}, w_H(\mathbf{c}) = d\}|. \tag{1.7}$$

The WE can be also represented via the *weight enumerator function* (WEF) $A(X)$:

$$A(X) = \sum_{d=0}^{n} A_d X^d. \tag{1.8}$$

Given a code distance spectrum, we can estimate the decoding performance of such code under maximum likelihood decoding via the *union bound* (UB) on the block error probability, $P_B$. If we assume the channel is a *binary-input Additive White Gaussian Noise* channel (bi-AWGN), $P_B$ is given by

$$P_B \leq P_{UB} = \frac{1}{2} \sum_{d=d_{\min}}^{n} A_d \; erfc\left(\sqrt{d \; R \; \frac{E_b}{N_0}}\right) \tag{1.9}$$

with $R = k/n$ being the code rate, $E_b/N_0$ the energy per information bit to noise power spectral density ratio and $erfc$ the complementary error function. When the signal-to-noise ratio is large, only the $A_{d_{\min}}$ term in the UB is sufficient to approximate the performances of the code.

## 1.2 Convolutional Encoders

CCs are a particular type of binary linear codes invented in 1955 by Elias [Elias, 1955] and which can be efficiently described through their encoder. Consider at time $t$ the encoder input to be given by the $k-$bit vector $\mathbf{u}_t$. The output at time $t$ is given by the $n-$bit vector $\mathbf{c}_t$

$$\mathbf{c}_t = \sum_{i=0}^{\nu} \mathbf{u}_{t-i}\mathbf{G}_i$$

where $\mathbf{G}_i$, $i = 0, \ldots, \nu$ are $k \times n$ binary matrices and where $\nu$ is referred to as the memory.

**Definition 1.6.** The *free distance* of an $(n, k, \nu)$ binary convolutional code $\mathcal{C}$ is the minimum Hamming distance between two distinct sequences produced by its encoder

$$d_{free}(\mathcal{C}) = \min_{\substack{\mathbf{c}\neq\mathbf{c}' \\ \mathbf{c},\mathbf{c}'\in\mathcal{C}}} d_H(\mathbf{c}, \mathbf{c}') \tag{1.10}$$

The largest possible *free distance* of a CC depends on its memory $\nu$, and well-known upper bounds there exist, as the Heller's upper bound [Heller, 1968] and the Griesmer's upper bound [Griesmer, 1960].

**Corollary 1.1** (**Heller's bound**). *The free distance $d_{free}$ for any rate $R = k/n$ convolutional code of memory $\nu$ satisfies*

$$d_{free} \leq \min_{i\geq 1} \left\lfloor \frac{(\nu + i) \cdot n}{2 \cdot (1 - 2^{-ki})} \right\rfloor \tag{1.11}$$

**Theorem 1.2** (**Griesmer's bound**). *The free distance $d_{free}$ for any binary rate $R = k/n$ convolutional code of memory $\nu$ satisfies*

$$\sum_{j=0}^{ki-1} \left\lceil \frac{d_{free}}{2j} \right\rceil \leq (\nu + i) \cdot c, \quad i = 1,2,\ldots \tag{1.12}$$

In this work we will focus only in $(n = 2, k = 1, \nu)$ CCs, but generalizations of the work are possible. An example of $(n = 2, k = 1, \nu = 2)$ CC encoder is shown in Figure 1, where $D$ represents a delay block.

We can view the CC encoder as a finite state machine (FSM) with $2^\nu$ states and $2^k$ possible transitions from each state and represent such FSM using a *state-transition diagram* as in Section 1.2 or its equivalent *state-transition table* as in Section 1.2.

The input-output relationships of a convolutional encoder can be represented using the $D$-transform which is defined as it follows:

$$\mathbf{x} = (..., \mathbf{x}_{-1}, \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ...) \xrightarrow{D} \mathbf{x}(D) = \sum_t \mathbf{x}_t D^t = ... + \mathbf{x}_{-1}D^{-1} + \mathbf{x}_0 + \mathbf{x}_1 D + \mathbf{x}_2 D^2 + ... \tag{1.13}$$

with $\mathbf{x}_t = (x_t^{(1)}, x_t^{(2)}, ...)$.

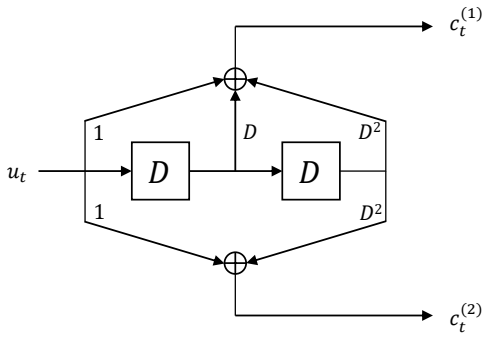
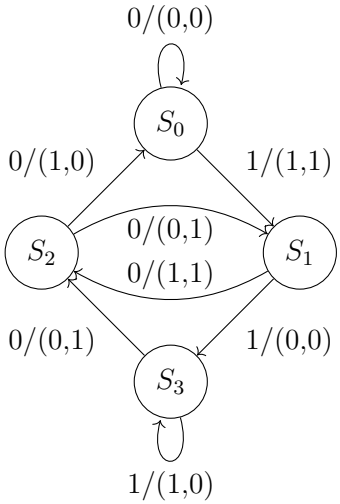



Figure 1: Convolutional encoder for an $(n = 2, k = 1, \nu = 2)$ code with $\mathbf{G}(D) = [1 + D + D^2, 1 + D^2]$.

| Current State $(u_{t-1}, u_{t-2})$ | Input $u_t$ | Next State $(u_t, u_{t-1})$ | Output $\mathbf{c}_t = (c_t^{(1)}, c_t^{(2)})$ |
|---|---|---|---|
| $S_0 = (0,0)$ | 0 | $S_0 = (0,0)$ | $(0,0)$ |
| | 1 | $S_1 = (1,0)$ | $(1,1)$ |
| $S_1 = (1,0)$ | 0 | $S_2 = (0,1)$ | $(1,1)$ |
| | 1 | $S_3 = (1,1)$ | $(0,0)$ |
| $S_2 = (0,1)$ | 0 | $S_0 = (0,0)$ | $(1,0)$ |
| | 1 | $S_1 = (1,0)$ | $(0,1)$ |
| $S_3 = (1,1)$ | 0 | $S_2 = (0,1)$ | $(0,1)$ |
| | 1 | $S_3 = (1,1)$ | $(1,0)$ |

(a) State transition table




(b) State transition diagram

Figure 2: State transition table (a) and state transition diagram (b) of the $(n = 2, k = 1, \nu = 2)$ convolutional code with $\mathbf{G}(D)$=$[1 + D + D^2, 1 + D^2]$.

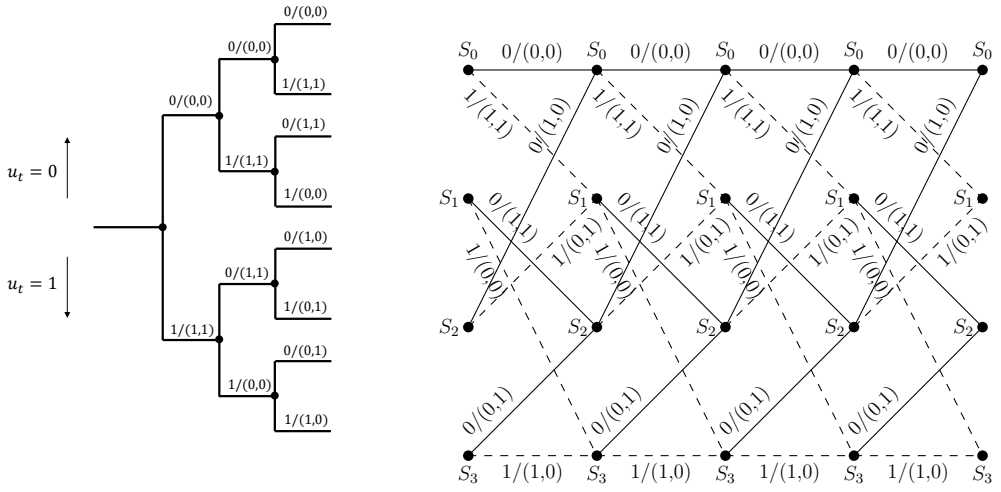
Doing so, the equations of Figure 1 can be expressed as

$$\mathbf{c}_t = [c_t^{(1)}, c_t^{(2)}] = [1 \cdot u_t + 1 \cdot u_{t-1} + 1 \cdot u_{t-2}, 1 \cdot u_t + 0 \cdot u_{t-1} + 1 \cdot u_{t-2}]$$
$$D \downarrow$$
$$\mathbf{c}(D) = [c^{(1)}(D), c^{(2)}(D)] = u(D)\,\mathbf{G}(D) = u(D)\,[(1 + 1 \cdot D + 1 \cdot D^2), (1 + 0 \cdot D + 1 \cdot D^2)]$$

where $\mathbf{G}(D)$ is a $k \times n$ matrix called the *transfer function* and each of its entries can be written as a polynomial rational function in the form $f(D)/q(D)$.

When we encode an information sequences $\mathbf{u}$ of length $K$, multiple of $k$, by a CC encoder, we can graphically represent the space of possibles transmitted sequences $\mathbf{c}_t$

through a *tree* structure (Figure 3a) or in a more compact way with a *trellis* representation (Figure 3b). Each *tree* node at depth $t$ is splitted in $2^k$ different branches, according to all possible information sequences $\mathbf{u}_t$. The *Trellis* diagram has $K/k$ sections, each having $2^\nu$ nodes, and each node at time $t$ is connected to the nodes in the following section according to the rules specified by the *state-transition diagram*.




(a) Tree representation

(b) Trellis graph

Figure 3: Tree representation (a) and trellis representation (b) of the $(n = 2, k = 1, \nu = 2)$ convolutional code with $\mathbf{G}(D)$=$[1 + D + D^2, 1 + D^2]$. (Dotted edges correspond to information bits equal to 1).

When a convolutional code is terminated it can be seen as a $(N, K)$ linear block code, where $N$ is the number of transmitted bits and we can compute its weight enumerator $A_d$ (and as a byproduct its minimum distance $d_{\min}$).

In the following section we present two popular termination schemes: the *zero-tail termination* and the *tail-biting* one.

### 1.2.1 Zero-Tail Termination Convolutional Codes

A very simple termination for a CC code, is the *zero-tail termination*. At time $t = 0$ the encoder is initialized to the state $S_0$, the allzero state, then the the $K$-bits sequence $u(D)$ is encoded through $\mathbf{G}(D)$. After that, $\nu$ zeroes are encoded through $\mathbf{G}(D)$ to force the encoder to come back to $S_0$. A code which respects the *zero-tail termination* condition is called a zero-tail terminated convolutional code (ZTCC).

The resulting binary linear block code is an $(N, K)$ code with $N = (n/k) \cdot (K + \nu)$. The code rate of such code is given by

$$R_{ZTCC} = \frac{K}{N} = \frac{k}{n}\frac{K}{K+\nu} \tag{1.14}$$

which is clearly smaller than $k/n$. However for a long information sequence $K$, $R_{ZTCC} \longrightarrow k/n$.

A way to obtain a rate $k/n$ is via puncturing $n \cdot \nu$ output bits of the ZTCC encoder, which means we do not transmit those bits. Puncturing affects the minimum distance of a CC code, and the $d_{\min}$ of a punctured code is upper bounded by the one of the original code.

An example of the *trellis* of a ZTCC is shown in Figure 4.

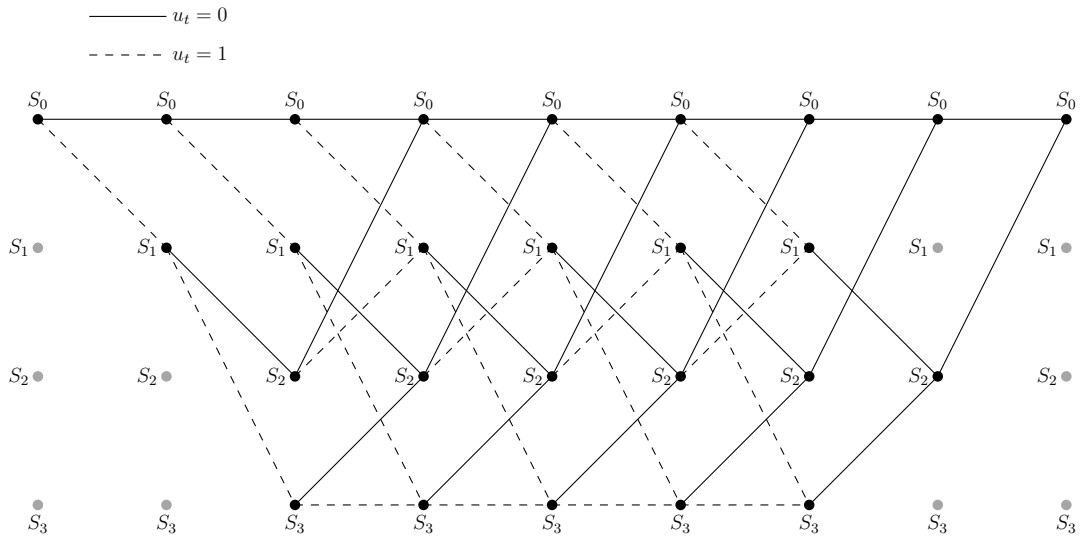



Figure 4: *Zero-terminated* trellis of the convolutional code with $\mathbf{G}(D)$=[1+$D$+$D^2$,1+$D^2$].

The generator matrix of the equivalent block code of a ZTCC is simply obtained as shown in Equation (1.15), where $\mathbf{G}_{ZTCC}$ has dimensions $N \times K$.

$$\mathbf{G}_{ZTCC} = \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \dots & & \mathbf{G}_\nu & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \dots & & \mathbf{G}_\nu & \\ & & \ddots & \ddots & & & \ddots \\ & & & \mathbf{G}_0 & \mathbf{G}_1 & \dots & \mathbf{G}_\nu \end{pmatrix}. \tag{1.15}$$

### 1.2.2 Tail-Biting Convolutional Codes

Another possible way to terminate CCs is by imposing the *tail-biting condition*. Differently from the *zero-tail termination* case where we start at $t = 0$ from $S_0$ state, in the tail-biting case we can start from whichever state $S_i$, but we have to terminate, after encoding the $K$ bits of $\mathbf{u}$, again in the same state $S_i$. A CC code of this kind is called tail-biting convolutional code (TBCC).

A *tail-biting path* in a trellis is depicted in red in Figure 5.

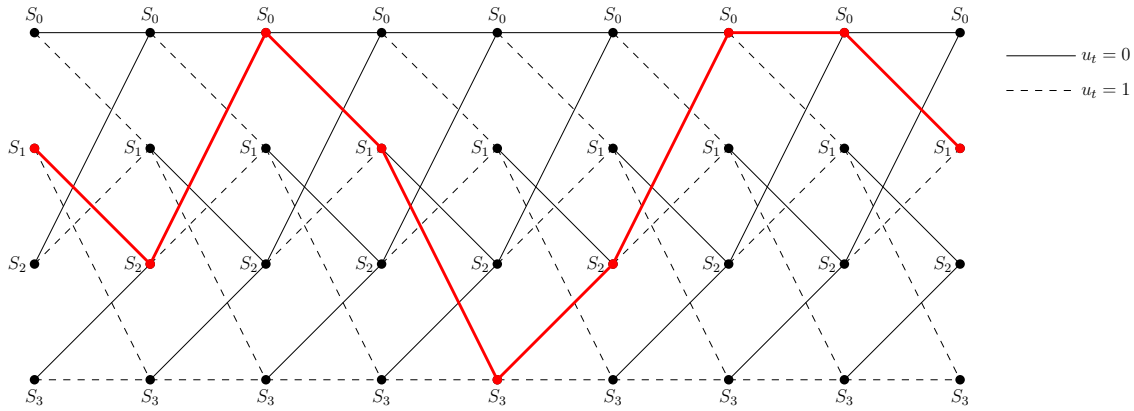



Figure 5: *Tail-biting terminated* trellis of the convolutional code with $\mathbf{G}(D)$=$[1 + D + D^2, 1 + D^2]$. An example of *tailbiting* path is highlighted in red.

The rate of TBCCs is $R_{TBCC} = K/N$ with $N = (n/k) \cdot K$,

$$R_{TBCC} = \frac{K}{N} = \frac{k}{n}\frac{K}{K} = \frac{k}{n}. \tag{1.16}$$

This means that TBCCs do not suffer of any rate loss, even for short block codes.

The generator matrix of the equivalent block code can be obtained using the *wrap-around* technique, which produces the result shown in Equation (1.17). $\mathbf{G}_{TBCC}$ is a $K \times N$ matrix.

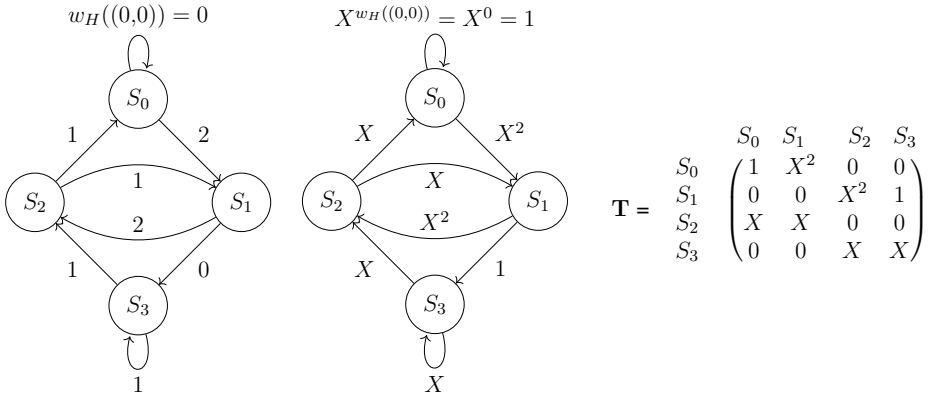
$$\mathbf{G}_{TBCC} = \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \dots & & \mathbf{G}_\nu & & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \dots & & \mathbf{G}_\nu & & \\ & & \ddots & \ddots & & & \ddots & \\ & & & \mathbf{G}_0 & \mathbf{G}_1 & \dots & & \mathbf{G}_\nu \\ \mathbf{G}_\nu & & & & \mathbf{G}_0 & \mathbf{G}_1 & \dots & \mathbf{G}_{\nu-1} \\ \mathbf{G}_{\nu-1} & \mathbf{G}_\nu & & & & \ddots & \ddots & \vdots \\ \vdots & & \ddots & & & & \ddots & \mathbf{G}_1 \\ \mathbf{G}_1 & \mathbf{G}_2 & \dots & \mathbf{G}_\nu & & & & \mathbf{G}_0 \end{pmatrix} \tag{1.17}$$

### 1.2.3 Weight Enumerator of Terminated Convolutional Codes

There are several ways to find the weight enumerator and the minimum distance of a terminated convolutional code. If the tailbiting code is long enough, the $d_{\min}$ of a TBCC is equivalent to that one of a ZTCC. An efficient and simple algorithm to find the $d_{\min}$ of a ZTCC is the Garello-Vila-Casado algorithm [Garello and Vila-Casado, 2004] and it is based on the VA [Viterbi, 1967] which we discuss in Section 1.3.2.

To find instead the weight enumerator of a CC we can use the *state-transition matrix*

$\mathbf{T}$ of its convolutional encoder, with $T_{i,j} = X^d$ where $d$ is the Hamming weight of the encoder output for the transition $S_i \rightarrow S_j$. An example of such matrix is shown in Figure 6.




(a) State transition diagram with weights

(b) State transition diagram with polynomials

(c) State transition matrix of Figure 6b with $Y = 1$

Figure 6: State transition diagram with the Hamming weight of the corresponding output vector (a), with corresponding output monomial (b) and the state transition matrix (c) of the convolutional code with $\mathbf{G}(D)$=$[1 + D + D^2, 1 + D^2]$.

To find $A(X)$ for a ZTCC we have

$$A(X) = \sum_{d=0}^{N} A_d \cdot X^d = (\mathbf{T}^{\frac{K+\nu}{k}})_{0,0} \tag{1.18}$$

while for a TBCC we have

$$A(X) = \sum_{d=0}^{N} A_d \cdot X^d = tr(\mathbf{T}^{\frac{K}{k}}). \tag{1.19}$$

Due to the sparsity of $\mathbf{T}$, the trellis can be exploited to reduce the computations through simple sums of polynomials and multiplications of polynomials with monomials. An example of the run of such algorithm for ZTCC is shown in Figure 7. For TBCC the algorithm is repeated $2^\nu$ times, each of them starting from a different state $S_i$.

Note that the number of different paths in the trellis representation of a $(n, k, \nu)$ CC encoder which start at state $S_i$ at section $0$ and reach the same state at section $t$ are $2^t/\nu$. So both the ZTCCs with $(K + \nu)/k$ trellis sections and the TBCCs with $K/k$ trellis sections and built from the same encoder have $2^K$ codewords.

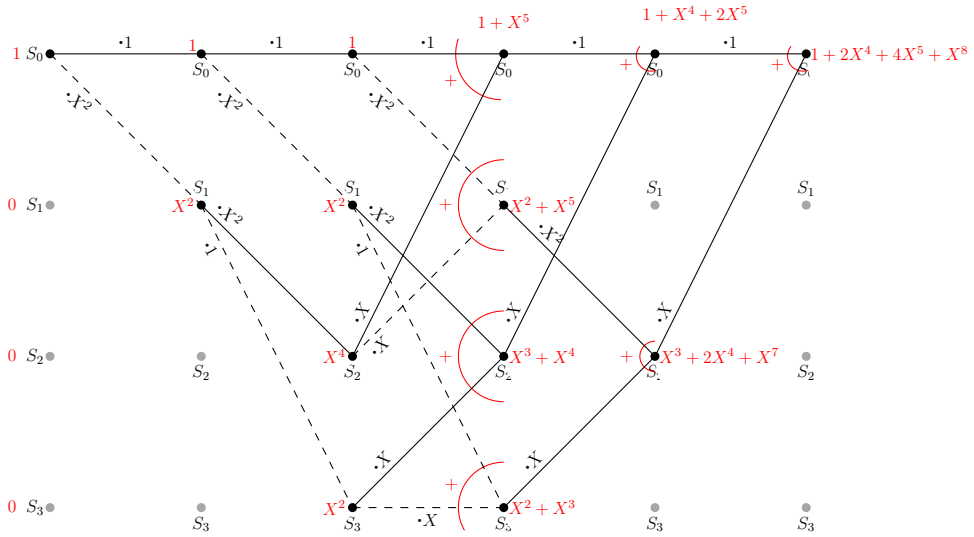



Figure 7: A trellis-based algorithm to compute the *weight enumerator* of the convolutional code with $\mathbf{G}(D)$=$[1 + D + D^2, 1 + D^2]$.

## 1.3 Decoding of Convolutional Codes

Previously we have seen how the convolutional encoder can be used to build block codes to protect the information bits, hereinafter we show how to recover such information sequence when the transmitted sequence is corrupted by a channel.

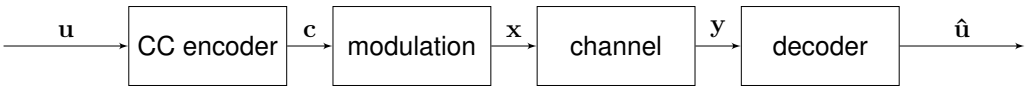



Figure 8: Encoding-Decoding block scheme

In particular we focus on transmission over an AWGN channel with binary phase shift keying (BPSK) modulation, with the bitwise mapping according to:

$$x_t : c_t \mapsto (-1)^{c_t} \implies \begin{matrix} 0 \mapsto +1 \\ 1 \mapsto -1 \end{matrix} \implies \mathbf{c} = (1\,0\,0\,1\,0) \overset{BPSK}{\longrightarrow} \mathbf{x} = (-1\,+1\,+1\,-1\,+1)$$

the binary input additive white Gaussian noise (bi-AWGN) channel is defined by

$$\mathbf{y} = \mathbf{x} + \mathbf{z}\,, \text{ with } z_t \text{ i.i.d. and } z_t \sim \mathcal{N}\left(0, \frac{N_0}{2}\right).$$

We have that the output distribution condition on the input is

$$p(y_t|c_t) = p(y_t|x_t(c_t)) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(y_t - x_t)^2}{2\sigma^2}}. \tag{1.20}$$

In the following section we discuss the ML decoding, then in Section 1.3.2 we present the ML decoder for CCs, based on the Viterbi algorithm. Finally, in Section 1.3.3 a near-ML decoder for TBCCs is detailed.

### 1.3.1 Maximum Likelihood Decoding

The aim of the maximum likelihood decoder is to estimate the transmitted codeword $\mathbf{c}$ and so the transmitted information sequence $\mathbf{u}$, choosing among all the possible transmitted codewords the one that maximizes the likelihood of observing $\mathbf{y}$ given that $\mathbf{c}$ has been transmitted, *i.e.*

$$\begin{aligned}\hat{\mathbf{c}} &= \arg\max_{\mathbf{c}\in\mathcal{C}} p(\mathbf{y}|\mathbf{c}) \\ &= \arg\max_{\mathbf{c}\in\mathcal{C}} \ln \prod_{t=1}^{N} p(y_t|x_t(c_t)) = \\ &= \arg\max_{\mathbf{c}\in\mathcal{C}} \sum_{t=1}^{N} \ln \left\{ \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(y_t - x_t(c_t))^2}{2\sigma^2}} \right\} \\ &= \arg\min_{\mathbf{c}\in\mathcal{C}} d_E^2(\mathbf{y}, \mathbf{x}(\mathbf{c})).\end{aligned}$$

Where we see that, over the AWGN channel, the ML decoding rule reduces to finding the (modulated) codeword at minimum Euclidean distance from the observation $\mathbf{y}$.

We should observe that a direct implementation of the rule above requires computing $2^K$ Euclidean distances, where $K$ is the number of information bits. This task has a complexity that is prohibitive already for $K$ in the order of a few tens of bits.

Note that the expression above can be further simplified in the case of constant envelope modulations and the minimization problem on the Euclidean distance square metric corresponds to a maximization problem of the correlation between $\mathbf{y}$ and all possible $\mathbf{x}(\mathbf{c})$.

### 1.3.2 The Viterbi Algorithm

For CCs there exist a ML decoder which can run over the code trellis, and it is based on the Viterbi algorithm (VA) [Viterbi, 1967].

It is based on the Bellman's theorem [Bellman, 1957] and Dijkstra's algorithm [Dijkstra et al., 1959] and on the additivity of the squared Euclidean distance Equation (1.21).

$$d_E^2(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{N} (y_i - x_i)^2 \tag{1.21}$$

To find the codeword at minimum Euclidean distance from $\mathbf{y}$, for a ZTCC, we have to:

1. draw $K+\nu$ consecutive trellis sections, starting at $t=0$ only with $S_0$ and ending at $t=K+\nu$ with only $S_0$

2. for all possible connections at each trellis section, we compute $\lambda_t^{S_i \to S_j}$, that is defined as the squared Euclidean distance at trellis section $t$ between the modulated codeword bits of the edge $S_i \to S_j$ and the values of the received vector $\mathbf{y}$ at time $t$

3. we define as $\Lambda_t^i$ as the squared Euclidean distance between state $S_0$ at section $0$ and state $S_i$ at section $t$ of the minimum path connecting those two states. We start with $\Lambda_0^0 = 0$ and thanks to the additivity property of the squared Euclidean distance, moving from from $t=1$ on, for all states $S_i$ at every section $t$ we can compute $\Lambda_t^i$ as the minimum $\Lambda_{t-1}^j + \lambda_t^{S_j \to S_i}$ for all states $S_j$ at section $t-1$ connected with state $S_i$ at section $t$. We also store the edge arriving at each state with the minimum cumulative metric (*i.e.*, minimum accumulated squared Euclidean distance)

4. at section $t=K+\nu$, after having found the $\Lambda_{K+\nu}^0$ we move from state $S_0$ backward along the stored edges. That path is the maximum likelihood path and its corresponding $\hat{\mathbf{c}}$ and $\hat{\mathbf{u}}$ are the maximum likelihood codeword and information sequence, respectively.

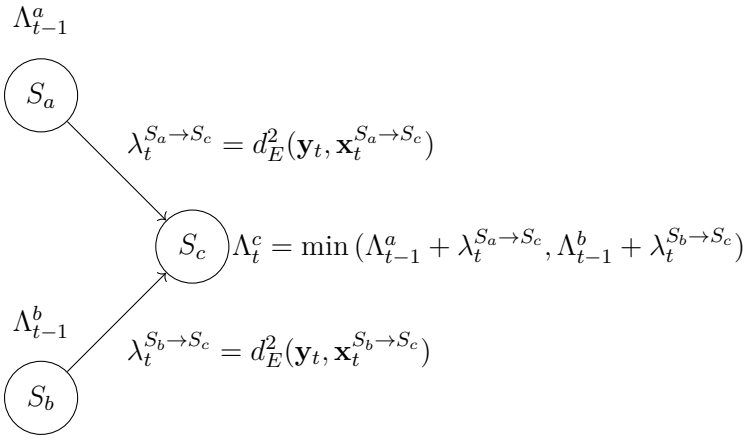
In Figure 9 is shown an example of a VA decoder.

In principle for TBCCs we can repeat the VA for each initial/final state $S_i$ finding the best paths $S_i^0 \to S_i^K$, and then choosing (among those) the one that minimizes the squared Euclidean distance from $\mathbf{y}$.

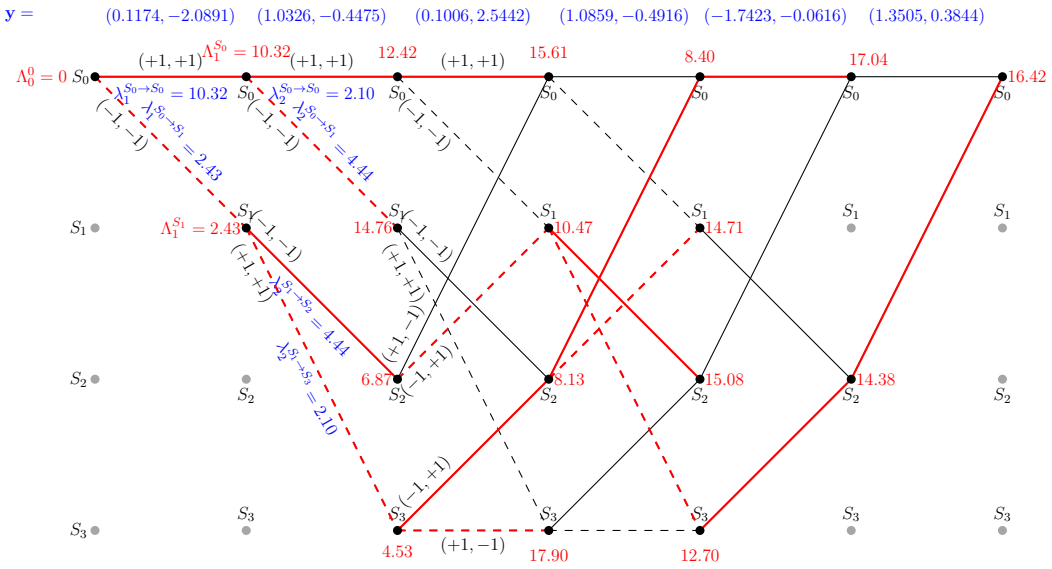
### 1.3.3 Wrap-Around Viterbi Algorithm

A drawback of the approach for decoding TBCCs discussed above is that $2^\nu$ instances of the VA have to be run, one of each initial/final state. Such algorithm becomes impractical for TBCCs with already small memory $\nu$. And since the minimum distance of a CC grows with $\nu$, a low complexity alternative is needed. We present next a near-ML decoder which is of practical utility: the wrap-around Viterbi algorithm (WAVA) decoder [Shao et al., 2003].

It consists in a round of the VA over the trellis, where we start at time $t=0$ from all states $S_i$ with $\Lambda_0^i = 0$. In fact, in probabilistic terms each state is equally probable to be the starting state. At section $t=K$ we check if the minimum path is also tail-biting ($S_i^0 \to S_i^K$). If it is tail-biting, we output the corresponding decisions $\hat{\mathbf{c}}$ and $\hat{\mathbf{u}}$. Otherwise we re-initialize the metrics $\Lambda_0^i$ of the states at $t=0$, but this time with the metrics obtained at $t=K$, $\Lambda_0^i = \Lambda_K^i$, and we re-run the VA. After the first round of Viterbi, we repeat this process until we find a tail-biting path or we reach the maximum number of iterations. If we do not find a TB path, we declare an error, while if we find more than one path we choose the one at minimum Euclidean distance from the received vector.




(a) Viterbi update rule




(b) Viterbi example

Figure 9: Example of the Viterbi algorithm on the trellis of the $(n = 2, k = 1, \nu = 2)$ convolutional code with $K = 4$, $\nu = 2$. The stored edges at each section are marked in red.

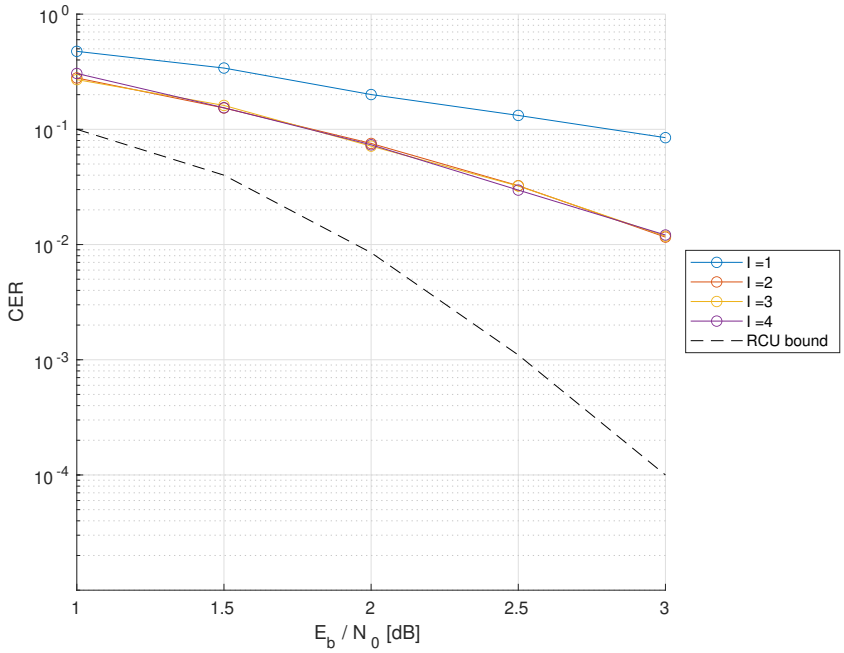
The re-initialization of the metrics $\Lambda_0^i$ at $t = 0$ can be seen as assigning a new probability to each initial state $S_i$.

In Figure 10 we show the error correction performances in terms of codeword error

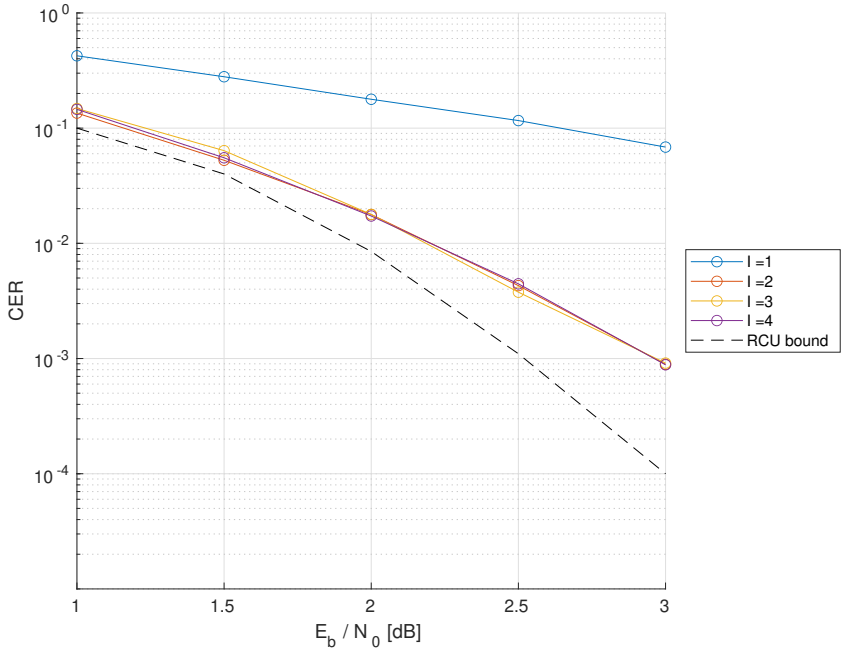
rate (CER) obtained by the WAVA algorithm as function of the number of rounds (iterations, $I$) around the trellis as function of the signal-to-noise ratio $E_b/N_0$ in dB. The codes used are the $(n = 2, k = 1, \nu = 5)$ and the $(n = 2, k = 1, \nu = 8)$ TBCCs with $\mathbf{G} = [53{,}75]$ and $\mathbf{G} = [561{,}753]$ respectively, where the polynomials are expressed in octave. The length of the information sequence is of $K = 64$ bits.

In the same figure the random coding union bound (RCU) bound [Polyanskiy et al., 2010] is plotted, an achievability bound predicting the existance of a code with such performance.

As we can see in Figure 10, increasing the memory $\nu$ of the CC, we increase the error correction performance of such code, approaching the bound.




(a) $(n = 2, k = 1, \nu = 5)$ TBCC with $\mathbf{G} = [53,75]$ and $K = 64$




(b) $(n = 2, k = 1, \nu = 8)$ TBCC with $\mathbf{G} = [561,753]$ and $K = 64$

Figure 10: Simulated codeword error rate on a bi-AWGN channel as a function of $E_b/N_0$ of a TBCC decoded via WAVA for different number of iterations $I$.

# 2. List Decoders for Convolutional Codes

We are now going to review some decoders which do not output only the minimum distance trellis path, but a sorted list $\mathcal{L}$ of size $L$ including the paths at low distance from the channel output. They are based on the VA, for this reason are called LVAs (LVAs).

They were firstly introduced in [Seshadri and Sundberg, 1994] and they are referred as: the parallel-list Viterbi algorithm (P-LVA), presented in Section 2.1, which outputs at once the full list, and as the serial-list Viterbi algorithm (S-LVA), presented in Section 2.2, which outputs at every iteration $i$ the $i-$th best path.

## 2.1 Parallel-List Viterbi Algorithm

P-LVA is a simple modification of VA, where at every state $S_i$ and at every trellis section $t$, we do not store only the metric $\Lambda_t^i$ of the minimum distance path reaching state $S_i$, but we store the metrics of the $L_i$ minimum distance paths. With $L_i$ equal to $L$, the algorithm guarantees to find the $L$ global paths at low distance from the channel output. The $l-$th shortest distance trellis path at section $t$ reaching the state $S_i$ is denoted by $\mathcal{L}_t^{i,(l)}$ and identified by three parameters: $\Lambda_t^{i,(l)}$, containing its squared Euclidean distance, $S_{t-1}^{i,(l)}$, identifying the previous trellis state, and $l_{t-1}^{i,(l)}$, which is the index of the path in the list $\mathcal{L}$ at the previous state.

At each state $S_i$ we merge the $L$ trajectories from each one of the $2^k$ incoming edges.

In Figure 11 is shown an example of such sorting algorithm when $k = 1$, while in Algorithm 1 the merging algorithm is shown.

For instance, the P-LVA can be used to decode TBCCs finding the best $L$ paths at each initial/final state $S_i$ and then check which of them matches the *tail-biting* condition, if any, and output the corresponding $\hat{\mathbf{c}}$ and $\hat{\mathbf{u}}$ of the shortest among them.

For TBCC, in order to save the time-complexity of checking the tail-biting condition of all the $L$ best paths, we can store in $\mathcal{L}_t^{i(l)}$ also the starting state of such path.

**Algorithm 1** Algorithm to merge the $2^k$ incoming lists in the P-LVA when $k = 1$.

$l_a = 1, l_b = 1$
$\Lambda_a = \Lambda_{t-1}^{a,(l_a)} + \lambda_t^{(S_a \to S_c)}$
$\Lambda_b = \Lambda_{t-1}^{b,(l_b)} + \lambda_t^{(S_b \to S_c)}$
**for** $l = 1$ , ... , $L$ **do**
  **if** $\Lambda_a < \Lambda_b$ **then**
    $\Lambda_t^{c,(l)} = \Lambda_a$
    $l_a = l_a + 1$
    $\Lambda_a = \Lambda_{t-1}^{a,(l_a)} + \lambda_t^{(S_a \to S_c)}$
  **else**
    $\Lambda_t^{c,(l)} = \Lambda_b$
    $l_b = l_b + 1$
    $\Lambda_b = \Lambda_{t-1}^{b,(l_b)} + \lambda_t^{(S_b \to S_c)}$
  **end if**
**end for**

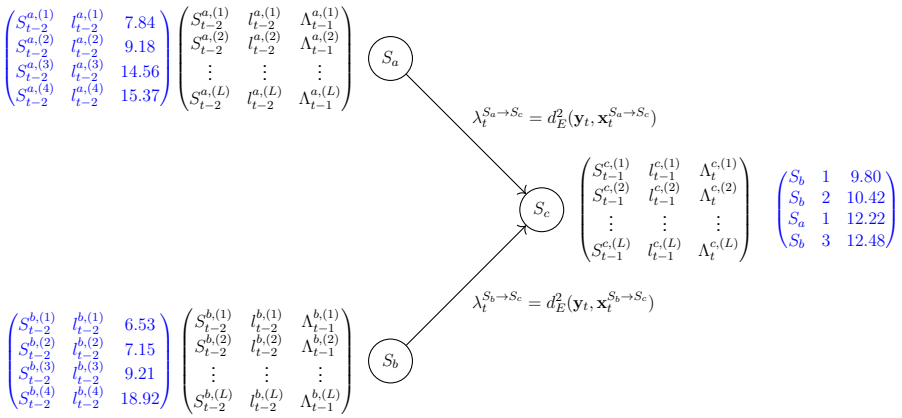



Figure 11: Example of the update rule of the P-LVA. Each entry of each list contains in position $l$ the parameters of the $l$-th minimum distance path according to the Euclidean distance metric. An example of such rule when $L = 4$ is depicted in blue.

## 2.2 Serial-List Viterbi Algorithm

In [Seshadri and Sundberg, 1994] was presented a naive approach of the S-LVA algorithm which is based on the following idea for a ZTCC: with the VA we can find the minimum distance path $\mathcal{L}_t^{i,(l)}$ at each state $S_i$ of the trellis. For instance, for a ZTCC, $\mathcal{L}_{K+\nu}^{0,(1)}$ is the minimum distance path starting/ending in $S_0$. The global second best path, always according to the Euclidean distance, $\mathcal{L}_{K+\nu}^{0,(2)}$ has to be a path which leaves the best path at a time $t_1$ and rejoin it at time $t_2$. In order to find it, at each section $t$ we store the best path merging $\mathcal{L}_{K+\nu}^{0,(1)}$ at that section and we select among the stored paths the one at minimum Euclidean distance from the channel output.

We repeat this procedure to find the third global best path $\mathcal{L}_{K+\nu}^{0,(3)}$, but this time we compare the minimum distance path merging $\mathcal{L}_{K+\nu}^{0,(2)}$ different from $\mathcal{L}_{K+\nu}^{0,(1)}$ with the best path merging $\mathcal{L}_{K+\nu}^{0,(1)}$ different from $\mathcal{L}_{K+\nu}^{0,(2)}$. We then select the path at minimum Euclidean distance from the channel output.

In order to find $\mathcal{L}_{K+\nu}^{0,(4)}$, we compare the minimum distance paths merging $\mathcal{L}_{K+\nu}^{0,(1)}$, $\mathcal{L}_{K+\nu}^{0,(2)}$ and $\mathcal{L}_{K+\nu}^{0,(3)}$, but different from those paths. The one at shortest Euclidean distance is our $4-$th global minimum distance path, and so on.

This procedure guarantees to find at each iteration the next global best path.

For TBCCs, we can extend such algorithm, but we do not only compare the paths reaching $S_0$ ($\mathcal{L}_K^{0,(l)}$), but all the paths $\mathcal{L}_K^{i,(l)}$ at final state $S_i$.

A faster version of such algorithm is known as tree-trellis list Viterbi algorithm and is detailed in [Roder and Hamzaoui, 2006].

This faster implementation is based on Viterbi and a data structure which can manage the list of best paths with fast insertion and search, *i.e.*, the heap data structure [Williams, 1964] or a red-black tree [Bayer, 1972] that have a logarithmic complexity in time.

In order to find the local best merging path at section $t$ to a given path, we have to look at the best of the merging paths at section $t$ incoming from a different edge with respect to the one selected by Viterbi. For $k = 1$, for instance, this corresponds to the path coming from the incoming edge which was not selected by Viterbi. We can store the information about that path in a node and sort it in our list based on its Euclidean distance from the received vector.

In order to find all the local best paths merging a given path, we need to repeat this operation for all the trellis sections. However, if the given path is the local best path of another path, and it merges its best path at time $t_1$, we can reduce the search space only looking at the sections with with $t < t_1$. This is due to the fact that for $t \geq t_1$ the two paths traverses the same nodes and edges, and so also the local best paths at those sections have to be the same.

We propose a visual example of the procedure in Figure 12.

## 2.3 On the complexities of P-LVA and S-LVA

The P-LVA with list size $L$ requires a single forward step of the VA, but its complexity w.r.t. such algorithm grows linearly in $L$ due to the merge operation at each node of the incoming lists. While the backward of the VA is repeated at most $L$ times, one for each of the $L$ global best paths.

The S-LVA complexity when the list size is $L$, with the fast tree-trellis implementation, is the same in forward of the VA, while in backward we have at most $L$ backwards. During the backward, for each trellis section we have to compute the metrics which

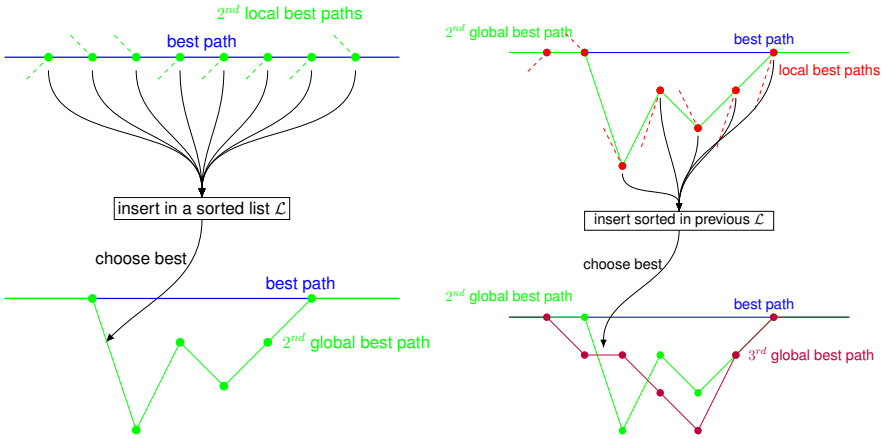



Figure 12: Visual example of the serial-list Viterbi algorithm.

identify the local best path merging at that section and then insert those metrics in the heap data structure or in the binary tree, which has a time complexity of at most $log_2(L)$.

In Table 1, we report the time complexities of P-LVA and the tree-trellis implementation of S-LVA for decoding a ZTCC, according to [Roder and Hamzaoui, 2006, Table I]. The TBCC case is similar, with the change that we have $K$ trellis sections instead of $K + \nu$.

Table 1: Time complexity for decoding the $L$ best paths of length $K + \nu$ sections for a binary rate$-1/n$ ZTCC with memory $\nu$. Source [Roder and Hamzaoui, 2006, Table I], with the notation adapted to the one used in this work. The complexities of the S-LVA are the ones of its tree-trellis implementation.

| | **Forward pass** | **Backward passes** | **Overall** |
|---|---|---|---|
| **P-LVA** | $O\left(L\,2^{\nu}\,(K+\nu)\right)$ | $O\left(L\,(K+\nu)\right)$ | $O\left(L\,2^{\nu}\,(K+\nu)\right)$ |
| **S-LVA** | $O\left(2^{\nu}\,(K+\nu)\right)$ | $O\left((K+\nu)\,L\,\log_2(L)\right)$ | $O\left((K+\nu)\,(2^{\nu}+L\,\log_2(L))\right)$ |

# 3. Serial Concatenation of Convolutional Codes with Outer (CRC) Linear Block Codes

## 3.1 Code construction

The concatenation of linear block codes can enhance the performance of such codes. The possible concatenations are the *parallel* one, where the same sequence of information bits is encoded by two or more different encoders; the *serial* concatenation, where a sequence of information bit is encoded firstly by an encoder and the output of such encoder is then encoded through another encoder. It is possible even to combine such concatenations in different schemes.

The use of concatenation usually increases also the decoder complexity, but can lead to very powerful codes such as the *turbo codes* which are a parallel concatenation of two CCs.

A powerful concatenation for short information sequences is the serial concatenation of CC with an outer linear block code such as a cyclic redundancy check (CRC) code. The use of a CRC code as outer code is not new and can lead to very powerful codes such as in the case of the concatenation with polar codes [Arikan, 2009; Tal and Vardy, 2015].

## 3.2 Cyclic Redundancy Check Codes

CRC codes [Peterson and Brown, 1961] are $(n, k)$ binary linear codes built from a polynomial $g(X)$ of degree $m = n - k$ with $g_0 = g_m = 1$ as

$$g(X) = g_0 + g_1 \cdot X + g_2 \cdot X^2 + \ldots + g_m \cdot X^m = 1 + \sum_{i=1}^{m-1} g_i \cdot X^i + X^m. \tag{3.1}$$

Their encoder follows Equation (3.2), where $c(X)$ and $u(X)$ are the codeword and the information sequence respectively expressed in polynomial form

$$c(X) = u(X) \cdot g(X). \tag{3.2}$$

**Definition 3.1.** A *cyclic code* is a binary linear code with the added property that the circular shift of a codeword is also a codeword. An $(n, k)$ binary linear code built from a polynomial $g(X)$ according to Equation (3.2) of degree $m = n - k$ with $g(X)$ being a

factor of $X^n - 1$ is a cyclic code.

**Definition 3.2.** Cyclic codes can be shortened by setting the $j$ most significant bits of the message $u(X)$ of Equation (3.2) to zero, which forces to select the subset of $c(X)$ of the cyclic code with the $j$ most significant bits always equal zero. This procedure goes under the name of code shortening.

**Theorem 3.1.** *The minimum distance of a shortened cyclic code is at least as large as the original cyclic code, since we are using a subset of the original codewords, although the error detection performance over the binary symmetric channel (BSC) can be deteriorated by shortening [Wolf and Blakeney, 1988].*

The generator matrix of the CRC codes for an information sequence of length $k$ is a $k \times n$ matrix

$$\mathbf{G}_{CRC} = \begin{pmatrix} g_0 & g_1 & \cdots & & g_m & & \\ & g_0 & g_1 & \cdots & & g_m & \\ & & \ddots & \ddots & & & \ddots \\ & & & g_0 & g_1 & \cdots & & g_m \end{pmatrix} \tag{3.3}$$

Their *parity check matrix* $\mathbf{H}$ can be easily obtained in different ways, for instance with a *Gaussian-Jordan elimination* over $\mathbb{F}_2$.

From the $m \times n$ $\mathbf{H}$ of the CRC code we can study the WE of the code itself, firstly computing the WE Function $B(X)$ (see Equation (1.7) and Equation (1.8)) of its *dual code* which has only $2^m$ codewords and then applying the MacWilliams identity [MacWilliams, 1963] shown in Equation (3.4).

$$A^{CRC}(X) = \frac{(1+X)^n}{2^m} B\left(\frac{1-X}{1+X}\right) \text{, with } m = n - k \tag{3.4}$$

## 3.3 Design of CRC codes

The design of an outer code, *e.g.*, of a CRC code to be used in concatenation with a CC requires to find the best code in terms of WE: larger minimum distance $d_{\min}$ and smaller $A_{d_{\min}}$ number of codewords with Hamming weight of $d_{\min}$ for the overall concatenated code.

We can say that only the codewords of the inner code whose messages respect the outer code conditions are also codewords of the concatenated code. For this reason, we can measure the performance of the outer code based on its capability of discarding (e.g., via *syndrome* decoding it corresponds to a non allzero *syndrome* vector) the input words of the inner code with low nonzero Hamming weight. The same problem can be viewed as the capability of the outer code of detecting the undetectable error events of the inner code with low nonzero Hamming weight.

For an inner CC, the input words associated to codewords with Hamming weight smaller than a design parameter $\tilde{d}$ correspond to paths over the trellis of weight smaller

than $\tilde{d}$. In [Lou et al., 2015; Yang et al., 2020] are presented some efficient methods to find the best degree-$m$ CRC code to be used in concatenation with an inner ZTCC or a TBCC.

To evaluate the distance spectrum of the concatenation up to the $A_{\tilde{d}}$ coefficient, the inner code input extracted words according to [Lou et al., 2015; Yang et al., 2020], are tested via polynomial check, but the same results can be obtained via *syndrome* check using the parity check matrix of the outer (CRC) linear block code. Input words which satisfy the outer code constraints correspond to valid codewords of the concatenated code and so they are counted in the weight enumerator of the concatenated code.

When the outer code is a CRC code, we can compute the distance spectra of the concatenation of the CC code with all the possible $2^{m-1}$ $(K+m, K)$ CRC codes and choose the CRC code with the best distance spectrum.

By using some symmetries of the ZTCC and TBCC codewords and trellis paths, and by using the detection properties of the CRC code, we can deeply reduce the number of checks to be performed for all the possible $2^{m-1}$ CRC codes, in order to compute the weight enumerator of the concatenated code and find the best CRC code in an efficient way [Lou et al., 2015; Yang et al., 2020].

## 3.4 Study CRC codes

We are going now to use the *Poltyrev's tangential sphere bound* (TSB) [Poltyrev, 1994] to evaluate the performance under a ML decoding of such concatenation scheme in terms of codeword error rate (CER), using the distance spectrum of such concatenation.

We compare such performance with the performance of a random code of the same rate $R$ which is in our case :

$$R = \frac{k}{n}\frac{K}{(K+m)}$$

with $k/n$ the rate of the CC, $K$ the length of the information sequence and $m$ the degree of the polynomial of the CRC code.

Since the combination of the CRC with the TBCC may results in a large number of codewords at minimum distance, affecting the performance of the codes, for this reason, applying a random interleaver between the outer and the inner encoder may results beneficial to enhance the general performances of these codes.

In order to evaluate the possible gain we are going to use the Poltyrev's TSB on the average weight enumerator (AWE) $\overline{A}_d$ of the ensemble of the combination of the TBCC with the CRC code. To do so, firstly we computed the input-output WE $A_{i,d}^{TBCC}$ of the TBCC and the WE $A_i^{CRC}$ of the CRC. $A_{i,d}^{TBCC}$ is always obtained using the algorithm in Figure 7, but using as edges of the trellis the ones in Figure 13. $Y$ is the input variable, while $X$ is always the output variable. The exponent of $Y$ is given by the Hamming weight of the $k$ input bits, while $X$ the Hamming weight of the $n$ output bits.

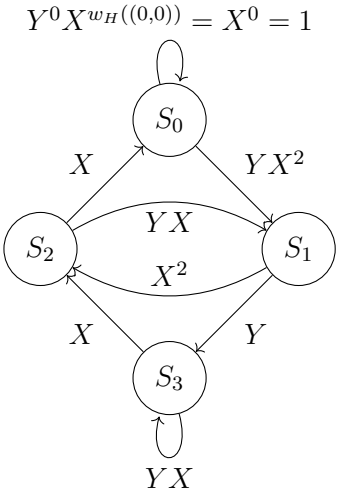



Figure 13: Input-output state transition diagram of the $(n = 2, k = 1, \nu = 2)$ convolutional code with $\mathbf{G}(D) = [1 + D + D^2, 1 + D^2]$.

The AWE $\overline{A}_d$ is then obtained with the formula in Equation (3.5).

$$\overline{A}_d = \sum_{i=0}^{K+m} \frac{A_i^{CRC} \cdot A_{i,d}^{CC}}{\binom{K+m}{i}} \tag{3.5}$$

However the ensemble can contain also bad codes (codes with lower minimum distance than the average minimum distance of the ensemble). For this reason, in order to understand which is the average performance of the random interleavers applied between our outer code and the convolutional encoder, we need to derive an upper bound to the ensemble expurgated from the bad codes. In [Gallager, 1963], a simple derivation is shown. Firstly we know that the cumulative distribution of the minimum distance for the codes in the ensemble satisfies

$$P\{d_{min}(\mathcal{C}) \leq d\} \leq \sum_{w=0}^{d} \overline{A}_d - 1 =: f(d)$$

For an arbitrary $\theta \in (0,1)$, we define:

$$d^* := \max_{d \in \mathbb{N}} \{d | f(d) < \theta\}$$

Then,

$$P\{d_{min}(\mathcal{C}) \leq d^*\} \leq \sum_{d=0}^{d} \overline{A}_d - 1 \leq \theta$$

It follows that a fraction of at least $(1 - \theta)$ codes from the ensembles has $d_{min} > d^*$, these codes are the expurgated ensemble.

In Section 2.2 of [Gallager, 1963] it is shown that the average weight enumerator of the expurgated ensemble can be upper bounded by

$$\begin{cases} \overline{A}_d^{exp} = 1 \ , & d = 0 \\ \overline{A}_d^{exp} = 0 \ , & 0 < d \leq d^* \\ \overline{A}_d^{exp} \leq \frac{1}{1-\theta} \overline{A}_d \ , & d > d^* \end{cases}$$

We studied the case with $k = 1$, $n = 2$ and $K = 64$ for various cases of $\nu$ and $m$ and it appears that the concatenation of TBCC with CRC codes is indeed very powerful and match the performance of the expurated ensemble as shown in Figure 14.

In the same figure is shown the Poltyrev's tangential sphere bound of a random code with the same code rate $R$. Such bound is achievable, which means that there exist codes which can achieve such bound.

## 3.5 List size

In this section we shows some simulated results of the decoding of the codes obtained with the serial concatenation presented before, comparing the list size when a S-LVA and a P-LVA are used.

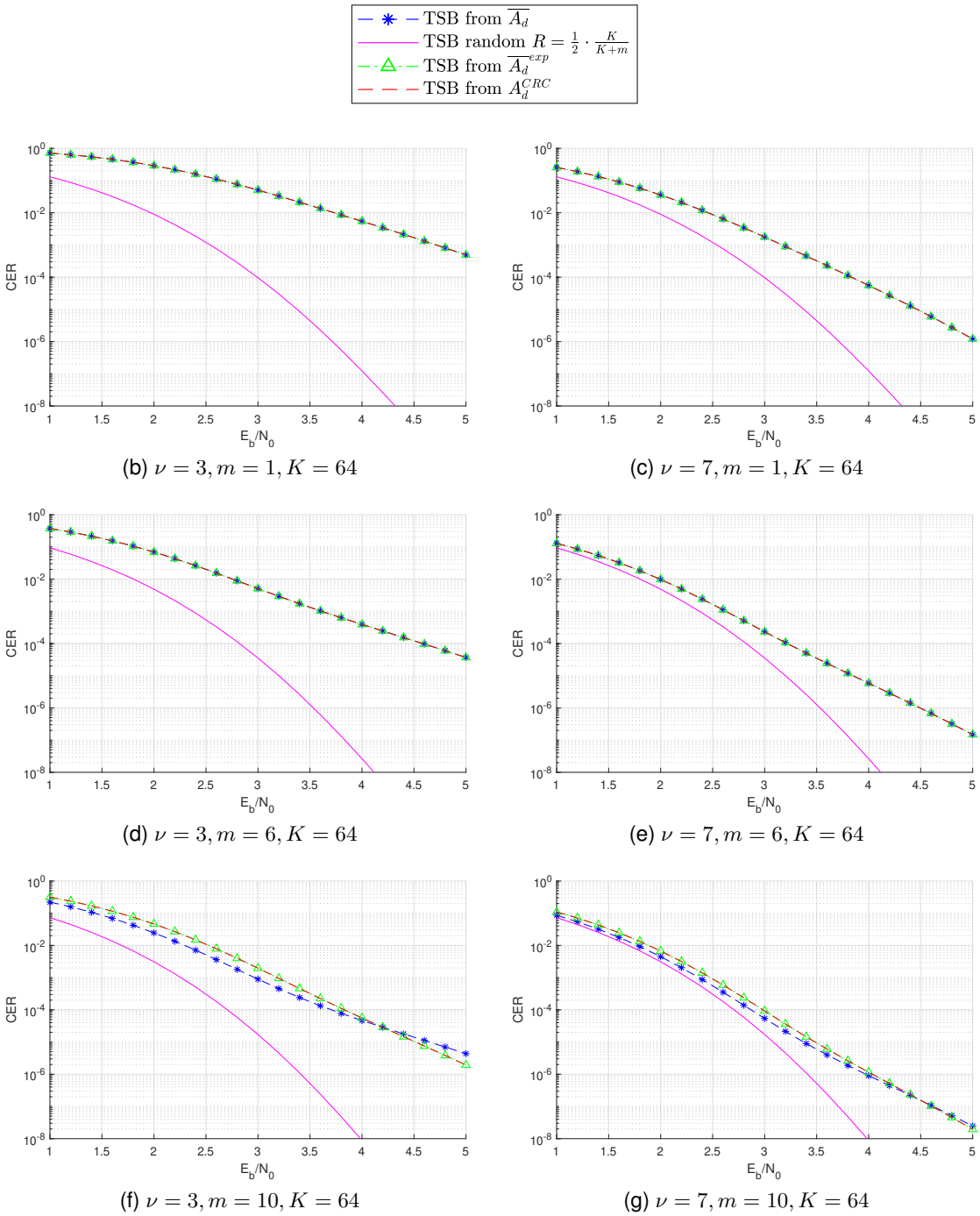
We use a $(n = 2, k = 1, \nu)$ TBCC as convolutional code, and a $(K + m, K)$ CRC. $K = 64$ and we use as rate of the concatenation $R = 1/2$. Such rate is achieved via puncturing one bit of $\mathbf{c}$ every $T$ sections with $T = \lfloor (K + m)/(2 \cdot m) \rfloor$. If in one section the punctured bit is the one of the first generator of the CC, the punctured bit at the next punctured section is the one of the second generator of the CC.

The CRC is found using the messages extracted from the already punctured Trellis.

In the P-LVA decoder each entry of the list stores also the starting state, and in order to reduce the backward operations we do as shown in Figure 15: firstly we check the tail-biting condition simply comparing that the starting state of the entries in each state $S_i$ is exactly $S_i$, then since we can compute the minimum distance of the concatenated code [Lou et al., 2015; Yang et al., 2020], in order to avoid the selection of a tail-biting path not too close to the received vector with the risk of increasing the *undetected error probability*, we check its distance from the received vector. Note that the *undetected error probability* is a really important parameter in satellite communications. After the BPSK mapper our code can be represented in the Euclidean space and each codeword is mapped to a signal code vector. We can compute the minimum Euclidean distance square $d^2_{E,\min}$ among its signal code vectors which is given by

$$d^2_{E,\min} = \min_{\substack{\mathbf{c} \neq \mathbf{c}' \\ \mathbf{c}, \mathbf{c}' \in \mathcal{C}}} d^2_E(\mathbf{x}(\mathbf{c}), \mathbf{x}(\mathbf{c}')) = 4\, d_{\min}, \tag{3.6}$$

where $d_{\min}$ is the minimum distance of the code in the Hamming space. Similar to the Hamming space, also in this Euclidean space, under ML decoding, we can define the correcting capability of the code. In the BPSK case, under ML decoding, the code is capable to correct at least all errors at Euclidean distance square smaller than $\frac{d^2_{E,\min}}{2}$. For this reason, we check that any tail-biting path at the last trellis section is at most at




(b) $\nu = 3, m = 1, K = 64$

(c) $\nu = 7, m = 1, K = 64$

(d) $\nu = 3, m = 6, K = 64$

(e) $\nu = 7, m = 6, K = 64$

(f) $\nu = 3, m = 10, K = 64$

(g) $\nu = 7, m = 10, K = 64$

Figure 14: Poltyrev's tangential sphere bound of : the average weight enumerator $\overline{A}_d$, of the ensemble, of the expurgated ensemble $\overline{A}_d^{exp}$, of the CRC $A_d^{CRC}$ and of a random code of the same rate.

that distance.

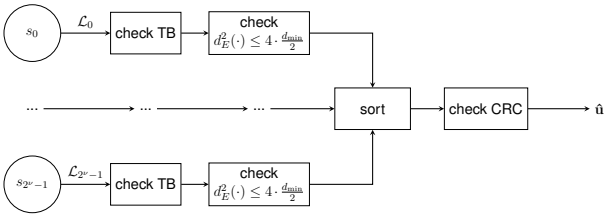



Figure 15: Last stage operations at each State $S_i$ of the P-LVA decoder.

The simulations highlight the important role of the number of iterations in WAVA, for both the P-LVA and S-LVA decoder to reduce the list size $L$ for decoding TBCCs. The list decoder is used only in the last round of WAVA, while the pure VA ($L = 1$) is used in all the other rounds.

In Figure 16 with the $(n = 2, k = 1, \nu = 3)$ TBCC with $\mathbf{G} = [13,17]$ and its CRC we see that with 2 rounds of Viterbi, $I = 2$, we can nearly reduce by 2 times the list size and with simply $L = 32$ we can nearly reach the ML correcting performance that we approximate by decoding the same code with S-LVA with $L = 1e6$. In the Figure 17 with the same code decoded by the S-LVA, since all the paths are inserted in a unique list with one controller, the list size needs to be larger and we need $L \approx 1000$ to reach the ML performance with $I = 1$ and also $I = 2$.

Despite the list size of the S-LVA decoder seems large, in practice the algorithm do not need to reach the maximum value of $L$, but it stops itself before it if the *tailbiting condition* and the CRC *parity check conditions* are satisfied. This deeply reduce its latency and in practice in software on a single core machine the needed time is $\approx \times 1.4$ the pure VA. It is important to note that if we compute the average list size used by the S-LVA decoder, as done in [Yang et al., 2018b], and we compare it with the required maximum list size value to approach the ML performance, we can see that the needed maximum list size is much larger than the average one. It means also that the decoding latency and the decoding complexity of some codewords is much larger than the average case.

The performance of the analyzed code with only 8 states per section is nearly the same of the $\nu = 6$ TBCC with $\mathbf{G} = [133,171]$, which has instead 64 states.

As shown in Figure 18 and Figure 19, another interesting result is the possibility of reaching the correcting performances of the memory $\nu = 8$ TBCC (Figure 10b) using a memory $\nu = 6$ code and the P-LVA with $L \approx 16 - 32$.

Differently from polar codes [Arikan, 2009; Tal and Vardy, 2015] where the bits of the outer code replace frozen bits, the use of the CRC in our concatenation force us to puncture the code for achieving the desired code rate $R = k/n$. This effect deeply affects the list size, because many non *tailbiting paths* and *tailbiting paths* come closer to our transmitted codeword. This phenomenon affects the performances at the decoder,

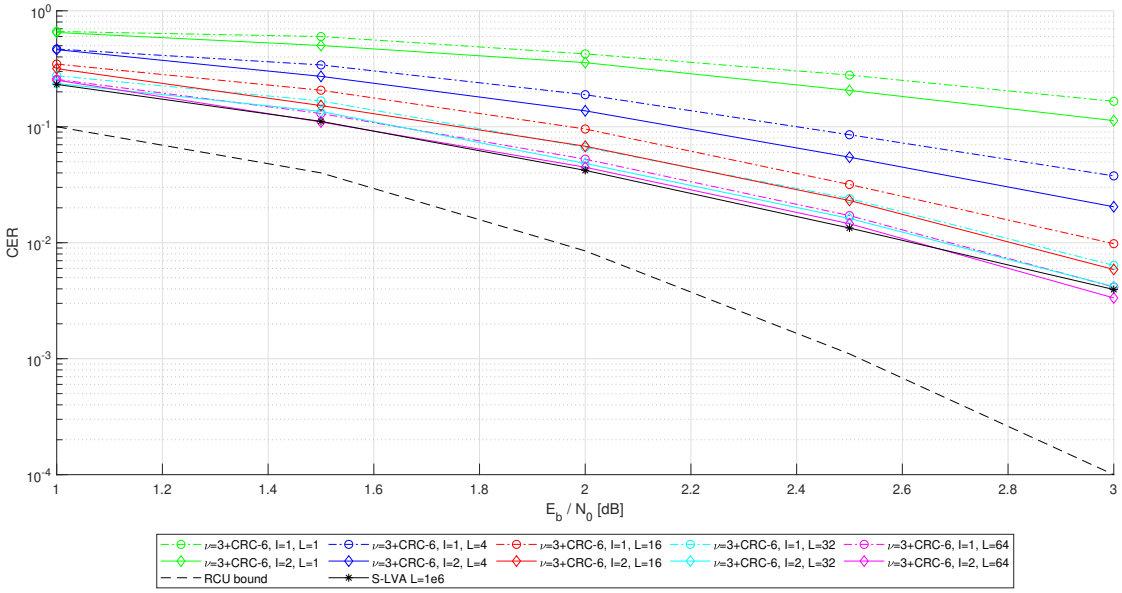



Figure 16: Codeword Error Rate of the serial concatenation of the $(n = 2, k = 1, \nu = 3)$ TBCC with $\mathbf{G} = [13,17]$ and its CRC of $m = 6$ over the bi-AWGN channel decoded via P-LVA. $K = 64$ and puncturing is applied to obtain $R = 1/2$.

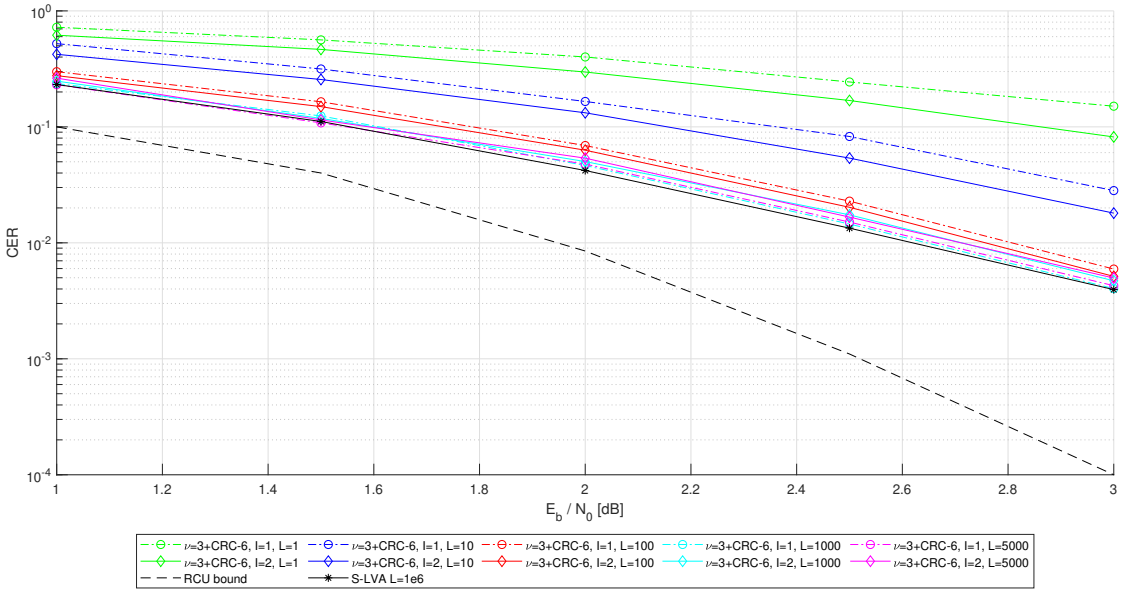



Figure 17: Codeword Error Rate of the serial concatenation of the $(n = 2, k = 1, \nu = 3)$ TBCC with $\mathbf{G} = [13,17]$ and its CRC of $m = 6$ over the bi-AWGN channel decoded via S-LVA. $K = 64$ and puncturing is applied to obtain $R = 1/2$.

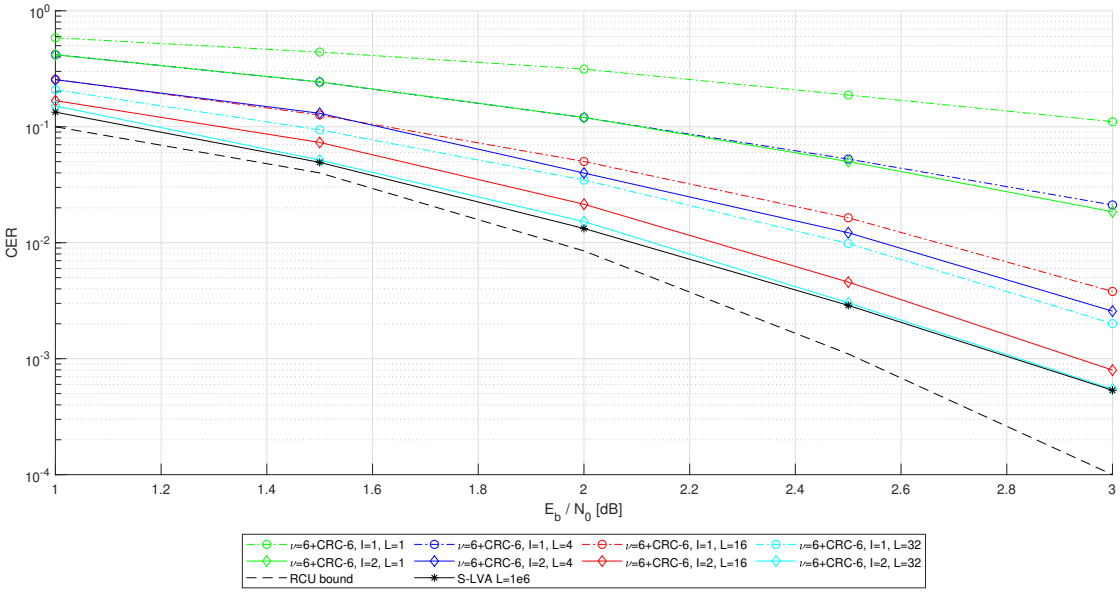



Figure 18: Codeword Error Rate of the serial concatenation of the $(n = 2, k = 1, \nu = 6)$ TBCC with $\mathbf{G} = [133,171]$ and its CRC of $m = 6$ over the bi-AWGN channel decoded via P-LVA. $K = 64$ and puncturing is applied to obtain $R = 1/2$.

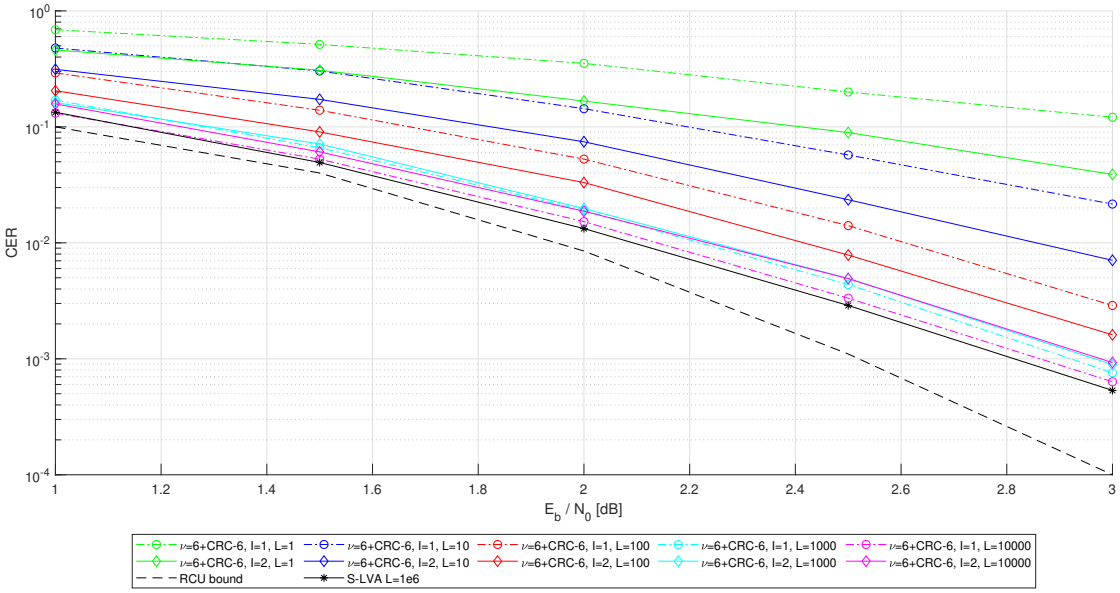



Figure 19: Codeword Error Rate of the serial concatenation of the $(n = 2, k = 1, \nu = 6)$ TBCC with $\mathbf{G} = [133,171]$ and its CRC of $m = 6$ over the bi-AWGN channel decoded via S-LVA. $K = 64$ and puncturing is applied to obtain $R = 1/2$.

forcing the list to grow every time we increase $m$ by 1 to obtain at least the same performances we had at previous $m$. In table 2 is shown the minimum list size to reach the maximum likelihood performances of such concatenation using the $(n = 2, k = 1, \nu = 5)$ TBCC and puncturing.

In the *zero-termination* case without puncturing decoded via list Viterbi algorithm (LVA), the effect of the CRC affects only the paths corresponding to the codewords of the TBCC and so the list size should grows similarly to the case of polar codes concatenated with a CRC.

Table 2: Simulated required list size as a function of the number $m$ of redundancy bits of the outer CRC code to approach the maximum likelihood performance of the serial concatenation of that outer code with the $(2,1,5)$ TBCC with $\mathbf{G} = [53,75]$. Puncturing is applied and the P-LVA is used at the decoder.

| $E_b/N_0$ [dB] | $m =$**1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **1.0** | 2 | 4 | 8 | 16 | 32 | 32 | 128 | 256 | 512 | 512 |
| **1.5** | 2 | 4 | 8 | 8 | 32 | 32 | 64 | 256 | 256 | 512 |
| **2.0** | 2 | 4 | 8 | 8 | 32 | 32 | 64 | 64 | 256 | 512 |
| **2.5** | 2 | 4 | 4 | 8 | 32 | 32 | 64 | 64 | 256 | 512 |
| **3.0** | 2 | 4 | 4 | 8 | 16 | 32 | 64 | 64 | 256 | 512 |

# 4. Fixed-latency Parallel-List Viterbi Algorithm

FROM the simulations of the P-LVA and S-LVA it is clear that the two algorithms require different list sizes for decoding TBCC and that the list size of the P-LVA seems smaller.

P-LVA can be of interest in hardware because it has a fixed latency, while the S-LVA seems to have an average latency which is really different with respect to the worst case latency.

In this section we propose an efficient implementation in hardware of the P-LVA decoder based on.

## 4.1 Simplified Bitonic Circuit

Sorting networks [Knuth, 1998] are networks made of comparators and wires and are used to sort a fixed number of elements. Each wire carries one single element and each wire move its element from left to right. When a pair of values, traveling through a pair of wires, encounter a comparator, the comparator swaps the values if and only if the top wire's value is greater or equal to the bottom wire's value. An exaple is depicted in

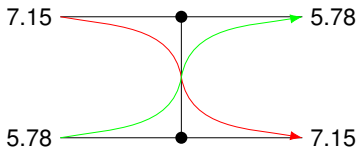



Figure 20: Example of a simple sorting network made of two wires and a comparator.

Such networks can be very useful in hardware implementation, because when some comparisons are made in parallel, we can deeply reduce the sorting time of elements, decreasing the total time complexity of an algorithm.

A very well-known sorting network is the bitonic sorter [Batcher, 1964] which can sort a list $L$ of elements with a time complexity of $\log_2^2(L)$, using $L \cdot \log_2^2(L)$ comparators and $L$ wires. Its network is shown in Figure 21.

In the case of our P-LVA, for a code with $k = 1$ at each step, for each node, there are two lists each of size $L$ which have to be merged and only the best $L$ values have to be stored at each node. If we sort at each state the list of elements at section $t$, when we

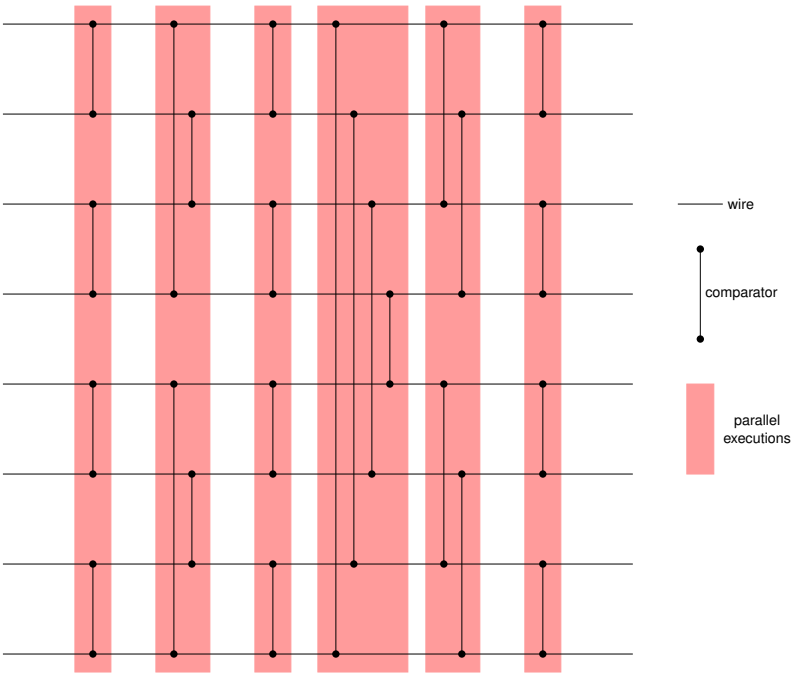



Figure 21: Example of the bitonic sorter to sort $8$ elements. The red boxes underline the operations which can run in parallel.

transmit such list to the states at section $t+1$, there is no need to sort again such list. It means that at each section $t$ the incoming lists are already sorted.

Since the values are already sorted, and since we have $2 \cdot L$ incoming elements, but only the best $L$ are of our interest, we can simplify the bitonic sorter to obtain the sorting network of our interest, which is the one shown in Figure 22 and whose algorithm works as expressed in Algorithm 2. If we compute the time complexity of such network is $\log_2(2 \cdot L) = 1 + \log_2(L)$, while the number of comparators is $\frac{L}{2}(\log_2(L) + 2)$.

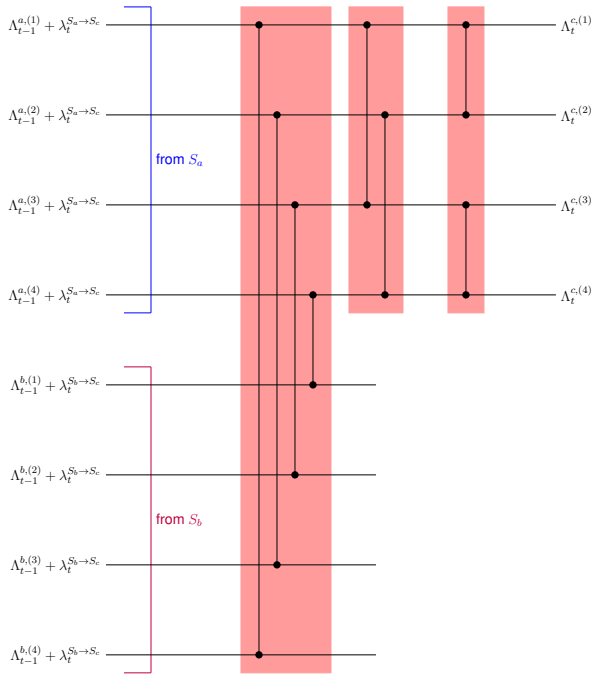



Figure 22: Example of circuit of the new sorting network for the P-LVA to merge the two incoming edges at state $S_c$ from state $S_a$ and $S_b$, respectively. $L = 4$. The red boxes underline the operations which can run in parallel.

**Algorithm 2** Simplified bitonic sorter for the parallel-list Viterbi algorithm

**for** $i = 1$ , ... , $L$ **do**
  $\Lambda^{c,(i)} = \min\{\Lambda^{a,(i)}, \Lambda^{b,(L-i+1)}\}$
**end for**
$N = L$
**for** $t = 1$ , ... , $log_2(L)$ **do**
  **for** $g = 1$ , ... , $2^{t-1}$ **do**
    **for** $i = 1$ , ... , $N/2$ **do**
      $t_1 = (g-1) \cdot N + i$
      $t_2 = t_1 + N/2$
      $(\Lambda^{c,(t_1)}, \Lambda^{c,(t_2)}) = (\min\{\Lambda^{c,(t_1)}, \Lambda^{c,(t_2)}\}, \max\{\Lambda^{c,(t_1)}, \Lambda^{c,(t_2)}\})$
    **end for**
  **end for**
  $N = N/2$
**end for**

# 5. Conclusion and Outlook

To conclude this thesis, in this section we are going to summarize the main results and discuss some interesting topics for further studies.

In this thesis we have analyzed the decoding performances of the parallel-list Viterbi algorithm and the serial-list Viterbi algorithm to decode the serial concatenation of the tail-biting convolutional code of rate $R = 1/2$ and outer cyclic redundancy check codes with $K = 64$ over the binary input additive white Gaussian noise channel.

As shown in the simulations, the analyzed concatenation scheme is very effective to enhance the minimum distance of these codes and so their correcting power, thus enabling to reduce the required memory at the encoder to achieve the same error correction performance. However, this gain comes at the price of an exponential increase in the required list size with respect to $m$, the amount of redundancy of the outer code. The wrap-around technique of the wrap-around Viterbi algorithm is effective to help reducing such list size, especially in combination with the parallel-list Viterbi algorithm decoder, but it is not sufficient when $m$ is greater than 6.

The implementation of the P-LVA is possible in hardware with a sorting network, allowing the possibility of using such decoder with a fixed-latency hardware implementation, while in software, the implementation of the serial-list Viterbi algorithm at the decoder remains the “preferred” solution, especially at very high list sizes $L$. In fact, such decoder is only $\approx \times 1.4$ times slower than the pure Viterbi algorithm for practical target values of error rate or signal-to-noise ratio.

For further improving this concatenation scheme while containing the list size, in order to construct lower rate codes (*e.g.*, 1/3) with a higher rate inner encoder (*e.g.,* 1/2), investigating the design of random outer codes or parity bit schemes that can be used to filter the list size at earlier decoding sections might be worth.

Another option would be the use of this concatenation design as outer constituent code of a serial concatenation with an inner polar code, thus improving the error correction performance of the actual polarization-adjusted convolutional codes or reducing their undetected error probability.

# Appendix

## A.1 CRC Codes

In this Appendix we present the results obtained of the searching for the best, in terms of distance spectrum, $(K, K+m)$ CRC code with redundancy $m$ for the serial concatenation of a $(n=2, k=1, \nu)$ TBCCs with the outer CRC linear block code. We provide those results for $\nu$ between $3$ and $8$, $m$ between $1$ and $10$ and $K=64$. For each case we report the minimum distance $d_{\min}$ of the serial concatenation, the number of codewords at that distance $A_{d_{\min}}$ and the generator $\mathrm{g}$ of such concatenation expressed in octave. The results for the unpunctured case are shown in Table 3, while Table 4 reports the results for the punctured case.

Table 3: Minimum distance $d_{\min}$ and number of codewords at that minimum distance $A_{d_{\min}}$ of the serial concatenation between the $(n=2, k=1, \nu)$ TBCC with the outer CRC code of generator $\mathrm{g}_{CRC}$, expressed in octave, for the unpunctured case.

| | **CRC** | $m$=1 | | | 2 | | | 3 | | | 4 | | | 5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\nu$ | $\mathbf{G}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ |
| 3 | [13,17] | 6 | 65 | 3 | 8 | 198 | 5 | 8 | 130 | 17 | 9 | 71 | 37 | 10 | 20 | 55 |
| 4 | [27,31] | 8 | 195 | 3 | 9 | 66 | 7 | 8 | 4 | 17 | 10 | 68 | 21 | 10 | 7 | 63 |
| 5 | [53,75] | 8 | 65 | 3 | 9 | 132 | 7 | 10 | 12 | 11 | 12 | 340 | 21 | 12 | 218 | 77 |
| 6 | [133,171] | 10 | 325 | 3 | 10 | 66 | 5 | 10 | 8 | 17 | 12 | 86 | 33 | 12 | 8 | 75 |
| 7 | [247,371] | 10 | 65 | 3 | 12 | 528 | 5 | 12 | 266 | 17 | 14 | 612 | 21 | 14 | 203 | 63 |
| 8 | [561,753] | 12 | 260 | 3 | 12 | 66 | 5 | 12 | 4 | 17 | 14 | 68 | 21 | 14 | 11 | 63 |

| | **CRC** | $m$=6 | | | 7 | | | 8 | | | 9 | | | 10 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\nu$ | $\mathbf{G}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathrm{g}_{CRC}$ |
| 3 | [13,17] | 12 | 735 | 143 | 12 | 154 | 355 | 12 | 11 | 407 | 14 | 219 | 1511 | 14 | 75 | 2235 |
| 4 | [27,31] | 12 | 64 | 117 | 12 | 12 | 265 | 14 | 432 | 555 | 15 | 73 | 1145 | 14 | 1 | 2321 |
| 5 | [53,75] | 14 | 700 | 143 | 12 | 2 | 275 | 16 | 1188 | 555 | 15 | 73 | 1511 | 16 | 1 | 2033 |
| 6 | [133,171] | 14 | 210 | 177 | 14 | 15 | 377 | 16 | 432 | 505 | 16 | 25 | 1275 | 16 | 10 | 2561 |
| 7 | [247,371] | 14 | 140 | 143 | 14 | 1 | 357 | 16 | 72 | 505 | 17 | 73 | 1641 | 18 | 247 | 2727 |
| 8 | [561,753] | 16 | 210 | 177 | 16 | 86 | 377 | 18 | 360 | 653 | 18 | 146 | 1401 | 18 | 17 | 2365 |

Table 4: Minimum distance $d_{\min}$ and number of codewords at that minimum distance $A_{d_{\min}}$ of the serial concatenation between the $(n = 2, k = 1, \nu)$ TBCC with the outer CRC code of generator $\mathbf{g}_{CRC}$, expressed in octave, for the punctured case. The code is periodically punctured according to the description provided in Section 3.5.

| | **CRC** | $m$ =1 | | | 2 | | | 3 | | | 4 | | | 5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\nu$ | $\mathbf{G}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ |
| 3 | [13,17] | 5 | 6 | 3 | 7 | 48 | 5 | 7 | 26 | 15 | 8 | 30 | 35 | 8 | 1 | 55 |
| 4 | [27,31] | 7 | 24 | 3 | 8 | 20 | 7 | 7 | 1 | 11 | 8 | 7 | 25 | 9 | 6 | 63 |
| 5 | [53,75] | 7 | 8 | 3 | 7 | 4 | 7 | 9 | 1 | 11 | 9 | 6 | 21 | 10 | 17 | 45 |
| 6 | [133,171] | 9 | 50 | 3 | 9 | 20 | 5 | 10 | 41 | 17 | 10 | 12 | 33 | 10 | 1 | 55 |
| 7 | [247,371] | 9 | 10 | 3 | 10 | 17 | 5 | 10 | 13 | 11 | 11 | 10 | 23 | 11 | 8 | 41 |
| 8 | [561,753] | 11 | 48 | 3 | 11 | 24 | 5 | 11 | 1 | 11 | 12 | 11 | 21 | 11 | 1 | 63 |

| | **CRC** | $m$ =6 | | | 7 | | | 8 | | | 9 | | | 10 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\nu$ | $\mathbf{G}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ |
| 3 | [13,17] | 9 | 16 | 107 | 10 | 69 | 325 | 10 | 22 | 651 | 10 | 1 | 1041 | 11 | 15 | 2353 |
| 4 | [27,31] | 10 | 28 | 117 | 10 | 4 | 221 | 11 | 48 | 523 | 11 | 11 | 1501 | 12 | 51 | 2763 |
| 5 | [53,75] | 10 | 9 | 153 | 11 | 28 | 373 | 11 | 8 | 451 | 11 | 2 | 1747 | 12 | 8 | 2301 |
| 6 | [133,171] | 11 | 5 | 123 | 11 | 1 | 255 | 12 | 7 | 761 | 13 | 79 | 1323 | 13 | 30 | 3565 |
| 7 | [247,371] | 12 | 115 | 143 | 12 | 1 | 351 | 13 | 75 | 421 | 13 | 20 | 1103 | 13 | 2 | 2727 |
| 8 | [561,753] | 12 | 2 | 171 | 13 | 40 | 205 | 14 | 70 | 653 | 14 | 17 | 1467 | 14 | 10 | 3637 |

## A.2 Polar Codes Comparison

With the 5G standardization process, a new family of codes has risen interests in the coding community: the family of the so-called polar codes [Arikan, 2009]. Such codes have shown powerful performances when designed for very short block lengths. Successive cancellation list decoding algorithm [Tal and Vardy, 2015] can reach the maximum likelihood performance of such codes by using a list. When these codes are serially concatenated with outer CRC codes, they do not requires any puncturing due to their encoding structure and such concatenation is beneficial to improve their performances.

We have compared their performances with respect to the use of TBCCs for various memories and decoded via P-LVA. The results are shown in Figure 23 and we can see that TBCCs can approach the performances of polar codes with similar list size and even beat those performances for memory value of $8$. The polar codes results are obtained using the tools in [Liva and Steiner, 2021].

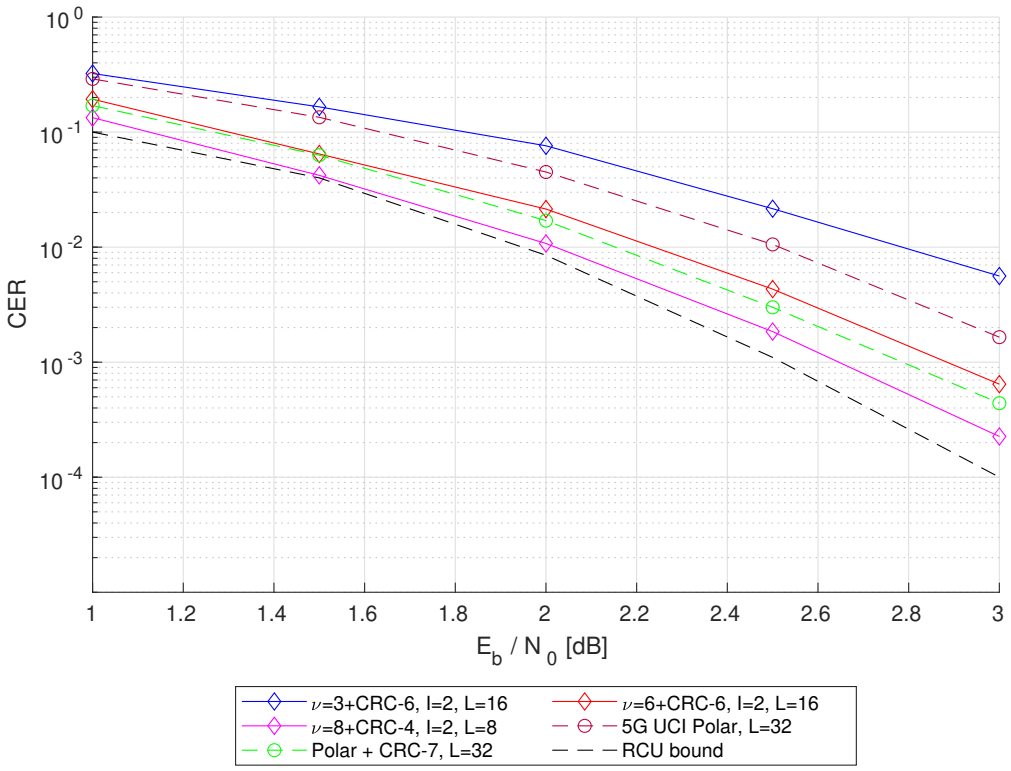



Figure 23: Performance comparison between polar codes and tail-biting convolutional codes for various memory of the TBCCs for ($N = 128$,$K = 64$).

## A.3 Time-Variant Convolutional Codes

It is possible to make the relationship between the input and the output of a convolutional encoder vary over time. In this way its input-output relationship can be written as:

$$\mathbf{c}_t = \sum_{i=0}^{\nu} \mathbf{u}_{t-i}\mathbf{G}_i(t)\,,\ \forall t = 0,1,2,\ldots$$

This operation can be beneficial in terms of distance spectrum and the resulting convolutional code is referred as a *time-variant convolutional code*. Among the classes of time-variant CCs, there are the *periodic* time-variant CCs, where $G_i$ changes periodically according to a period $T$ and, so, it can be expressed as $G_i(t) = G_i(t \mod T)$. Note that if $T = 1$, then the encoder remains time-invariant.

A possible way to obtain a time-variant CC is via periodically puncturing a lower rate CC.

A powerful periodic time-variant CC which can be graphically represented using only 16 states is the Golay CC [Calderbank et al., 1999] which has period $T = 4$ and upon termination, when $K$ is multiple of its period, it has a free distance equal to $8$.

So, its free distance is larger then the best, from the distance spectrum point of view, time-invariant memory $4$ CC which instead has $d_{free} = 7$.

This code can be constructed in various way, for instance by periodically puncturing the $(5,1,4)$ CC with $\mathbf{G}(D) = [25,35,27,33,37]$ expressed in octal format [Riedel and Weiss, 1999] and obtaining so the following generator transfer matrix $G(D) = \{t = 0 : [25,33], t = 1 : [27,37], t = 2 : [35,37], t = 3 : [25,33]\}$.

However, due to the high multiplicity of codewords at minimum distance, when $E_b/N_0$ is low, the Golay CC performs similarly with respect to the memory $4$ TBCC with $\mathbf{G}(D) = [27,31]$. And even if concatenated with an outer CRC code, when $E_b/N_0$ is low, the two codes behaves similarly, despite the Golay CC has larger minimum distance. We show some simulated results where we compare the two codes when no outer code is present (Figure 24) and when the CRC code as its generator polynomial of degree $m = 8$ (Figure 25).

Table 5: Comparison of the minimum distances of the serial concatenation of the outer CRC code with $m$ redundancy bits and the $(2,1,\nu = 4)$ time-invariant TBCC code with $\mathbf{G} = [27,31]$ and with the time-variant Golay convolutional code. The information sequence has $K = 64$ and the codes are unpunctured.

| | no CRC | | $m = 4$ | | | $m = 8$ | | |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{G}$ | $d_{\min}$ | $A_{d_{\min}}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ | $d_{\min}$ | $A_{d_{\min}}$ | $\mathbf{g}_{CRC}$ |
| **[27,31]** | 7 | 128 | 10 | 68 | 21 | 14 | 432 | 555 |
| **Golay** | 8 | 784 | 12 | 1700 | 21 | 16 | 982233 | 421 |

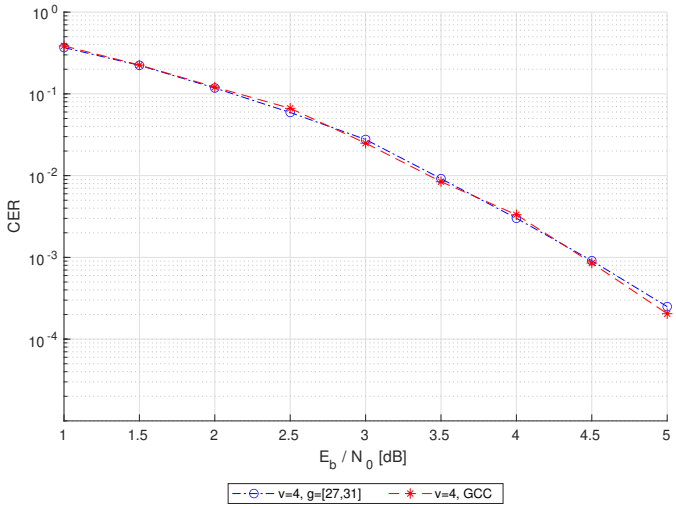
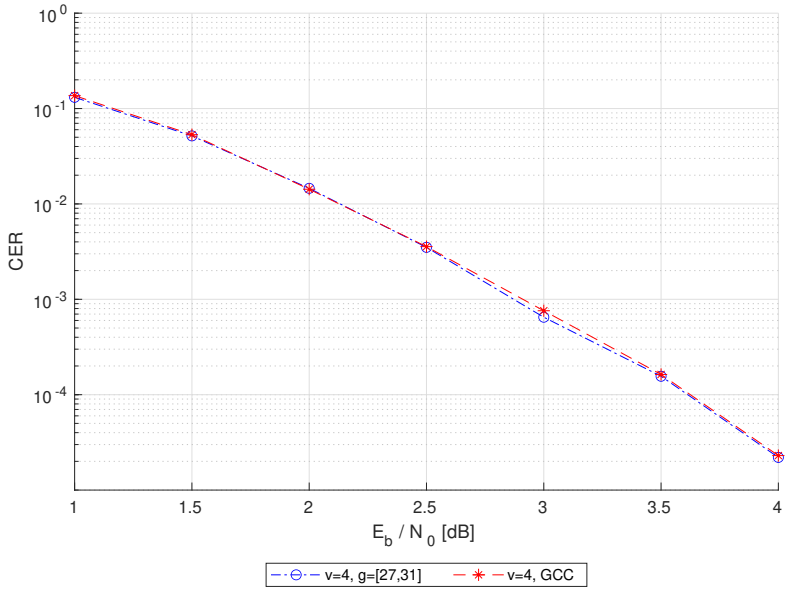
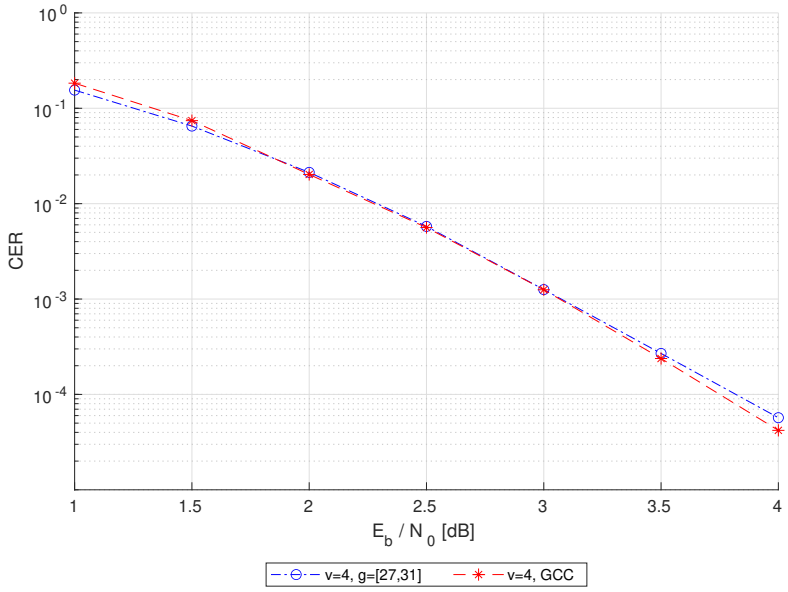



Figure 24: Performance comparison between the $(128,64)$ Golay CC with 16 states and the $(128,64)$ $(2,1,4)$ time-invariant TBCC without outer code and decoded via S-LVA with $L = 100000$.




(a) $(144,64), m = 8$




(b) $(128,64), m = 8$

Figure 25: Performance comparison between the tail-biting Golay CC with 16 states and the $(2,1,4)$ time-invariant TBCC when serially concatenated with outer CRC code with $m = 8$ when decoded via S-LVA with $L = 100000$. The unpunctured and punctured cases are shown repectively in (a) and (b).

# Bibliography


[Al-Fuqaha et al. 2015] AL-FUQAHA, Ala ; GUIZANI, Mohsen ; MOHAMMADI, Mehdi ; ALEDHARI, Mohammed ; AYYASH, Moussa: Internet of things: A survey on enabling technologies, protocols, and applications. In: *IEEE communications surveys & tutorials* 17 (2015), Nr. 4, P. 2347–2376

[Arikan 2009] ARIKAN, Erdal: Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. In: *IEEE Transactions on information Theory* 55 (2009), Nr. 7, P. 3051–3073

[Batcher 1964] BATCHER, Kenneth E.: Bitonic sorting. In: *Goodyear Aerospace Corp., Rep. GER-11869* (1964)

[Bayer 1972] BAYER, Rudolf: Symmetric binary B-trees: Data structure and maintenance algorithms. In: *Acta informatica* 1 (1972), Nr. 4, P. 290–306

[Bellman 1957] BELLMAN, Richard: A Markovian decision process. In: *Journal of mathematics and mechanics* 6 (1957), Nr. 5, P. 679–684

[Calderbank et al. 1999] CALDERBANK, A R. ; FORNEY, G D. ; VARDY, Alexander: Minimal tail-biting trellises: The Golay code and more. In: *IEEE Transactions on Information Theory* 45 (1999), Nr. 5, P. 1435–1455

[CISCO 2019] CISCO, Global Mobile Data T.: Cisco visual networking index: global mobile data traffic forecast update, 2017–2022. In: *Update* 2017 (2019), P. 2022

[De Sanctis et al. 2015] DE SANCTIS, Mauro ; CIANCA, Ernestina ; ARANITI, Giuseppe ; BISIO, Igor ; PRASAD, Ramjee: Satellite communications supporting internet of remote things. In: *IEEE Internet of Things Journal* 3 (2015), Nr. 1, P. 113–123

[Dijkstra et al. 1959] DIJKSTRA, Edsger W. et al.: A note on two problems in connexion with graphs. In: *Numerische mathematik* 1 (1959), Nr. 1, P. 269–271

[Elias 1955] ELIAS, Peter: Coding for noisy channels. In: *IRE Conv. Rec.* 3 (1955), P. 37–46

[Evans 2011] EVANS, Dave: The internet of things: How the next evolution of the internet is changing everything. In: *CISCO white paper* 1 (2011), Nr. 2011, P. 1–11

[Gallager 1963] GALLAGER, R: Low density parity check codes (Ph. D. dissertation). In: *Massachusetts Institute of Technology, Cambridge, Mass, USA* (1963)

[Garello and Vila-Casado 2004] GARELLO, Roberto ; VILA-CASADO, Andres: The all-zero iterative decoding algorithm for turbo code minimum distance computation. In: *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)* Bd. 1 IEEE , 2004, P. 361–364

[Griesmer 1960] GRIESMER, James H.: A bound for error-correcting codes. In: *IBM Journal of Research and Development* 4 (1960), Nr. 5, P. 532–542

[GSMA 2018] GSMA: 3GPP Low Power Wide Area Technologies (white paper) / GSMA (Global System for Mobile Communications Association). 2018. – Research Report

[Heller 1968] HELLER, JA: Short constraint length convolutional codes. In: *Space Program Summary 37-54, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Dec. 1968* 3 (1968), P. 171–174

[Knuth 1998] KNUTH, Donald E.: *The art of computer programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998

[Liang et al. 2019] LIANG, Ethan ; YANG, Hengjie ; DIVSALAR, Dariush ; WESEL, Richard D.: List-Decoded Tail-Biting Convolutional Codes with Distance-Spectrum Optimal CRCs for 5G. In: *2019 IEEE Global Communications Conference (GLOBECOM)* IEEE , 2019, P. 1–6

[Liva 2014] LIVA, Gianluigi: *Code Design in the Short Block Length Regime*. University Lecture at University of Bremen. 2014. – URL `https://www.ant.uni-bremen.de/ait/programm/Liva_TutorialSlides.pdf`

[Liva and Steiner 2021] LIVA, Gianluigi ; STEINER, Fabian: *pretty-good-codes.org: Online library of good channel codes*. `http://pretty-good-codes.org`. January 2021

[Lou et al. 2015] LOU, Chung-Yu ; DANESHRAD, Babak ; WESEL, Richard D.: Convolutional-code-specific CRC code design. In: *IEEE Transactions on Communications* 63 (2015), Nr. 10, P. 3459–3470

[MacWilliams 1963] MACWILLIAMS, Jessie: A theorem on the distribution of weights in a systematic code. In: *Bell System Technical Journal* 42 (1963), Nr. 1, P. 79–94

[Miraz et al. 2015] MIRAZ, Mahdi H. ; ALI, Maaruf ; EXCELL, Peter S. ; PICKING, Rich: A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT). In: *2015 Internet Technologies and Applications (ITA)*, 2015, P. 219–224

[Peterson and Brown 1961] PETERSON, William W. ; BROWN, Daniel T.: Cyclic codes for error detection. In: *Proceedings of the IRE* 49 (1961), Nr. 1, P. 228–235

[Poltyrev 1994] POLTYREV, Gregory: Bounds on the decoding error probability of binary linear codes via their spectra. In: *IEEE Transactions on Information Theory* 40 (1994), Nr. 4, P. 1284–1292

[Polyanskiy et al. 2010] POLYANSKIY, Yury ; POOR, H V. ; VERDÚ, Sergio: Channel coding rate in the finite blocklength regime. In: *IEEE Transactions on Information Theory* 56 (2010), Nr. 5, P. 2307–2359

[Riedel and Weiss 1999] RIEDEL, Sven ; WEISS, Christian: The Golay convolutional code-some application aspects. In: *IEEE Transactions on Information Theory* 45 (1999), Nr. 6, P. 2191–2199

[Roder and Hamzaoui 2006] RODER, Martin ; HAMZAOUI, Raouf: Fast tree-trellis list Viterbi decoding. In: *IEEE transactions on communications* 54 (2006), Nr. 3, P. 453–461

[Seshadri and Sundberg 1994] SESHADRI, Nambirajan ; SUNDBERG, C-EW: List Viterbi decoding algorithms with applications. In: *IEEE transactions on communications* 42 (1994), Nr. 234, P. 313–323

[Shannon 1948] SHANNON, Claude E.: A mathematical theory of communication. In: *The Bell system technical journal* 27 (1948), Nr. 3, P. 379–423

[Shao et al. 2003] SHAO, Rose Y. ; LIN, Shu ; FOSSORIER, Marc P.: Two decoding algorithms for tailbiting codes. In: *IEEE transactions on communications* 51 (2003), Nr. 10, P. 1658–1665

[Tal and Vardy 2015] TAL, Ido ; VARDY, Alexander: List decoding of polar codes. In: *IEEE Transactions on Information Theory* 61 (2015), Nr. 5, P. 2213–2226

[Viterbi 1967] VITERBI, Andrew: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE transactions on Information Theory* 13 (1967), Nr. 2, P. 260–269

[Williams 1964] WILLIAMS, John William J.: Algorithm 232: heapsort. In: *Commun. ACM* 7 (1964), P. 347–348

[Wolf and Blakeney 1988] WOLF, Jack K. ; BLAKENEY, Robert D.: An exact evaluation of the probability of undetected error for certain shortened binary CRC codes. In: *MILCOM 88, 21st Century Military Communications-What's Possible?'. Conference record. Military Communications Conference* IEEE , 1988, P. 287–292

[Yang et al. 2018a] YANG, Hengjie ; LIANG, Ethan ; WESEL, Richard D.: Joint design of convolutional code and crc under serial list viterbi decoding. In: *arXiv preprint arXiv:1811.11932* (2018)

[Yang et al. 2018b] YANG, Hengjie ; RANGANATHAN, Sudarsan V. ; WESEL, Richard D.: Serial list Viterbi decoding with CRC: Managing errors, erasures, and complexity. In: *2018 IEEE Global Communications Conference (GLOBECOM)* IEEE , 2018, P. 1–6

[Yang et al. 2020] YANG, Hengjie ; WANG, Linfang ; LAU, Vincent ; WESEL, Richard D.: An Efficient Algorithm for Designing Optimal CRCs for Tail-Biting Convolutional Codes. In: *2020 IEEE International Symposium on Information Theory (ISIT)* IEEE , 2020, P. 292–297