UNIVERSITY OF SOUTHAMPTON
FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
ELECTRONICS AND COMPUTER SCIENCE

Elias Gamma Error Correction Code

by

Tao Wang
BEng, MSc

A doctoral thesis report submitted in partial fulfilment of
the requirements for the award of Doctor of Philosophy
at the University of Southampton

January 2016

Supervisor: *Prof. Lajos Hanzo*
FREng, FIEEE, FIEE, DSc, EIC IEEE Press
Chair in Communications, Signal Processing and Control Group

Supervisor: *Dr. Robert Maunder*
PhD, CEng, MIET, SMIEEE, FHEA
Academic staff in Telecommunications Group
Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ
United Kingdom

𝔇edicated to my family and friends

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

Faculty of Physical Sciences and Engineering
School of Electronics and Computer Science

A thesis submitted for the award of Doctor of Philosophy

Elias Gamma Error Correction Coding

by Tao Wang

Shannon's source-channel coding separation theorem states that near-capacity communication is theoretically possible, when employing Separate Source and Channel Codes (SSCCs), provided that an unlimited encoding/decoding delay and complexity can be afforded. However, it is typically impossible to remove all source redundancy with the aid of practical finite-delay and finite-complexity source encoding, which leads to capacity loss. As a potential remedy, Joint Source and Channel Codes (JSCCs) have been proposed for exploiting the residual redundancy and hence for avoiding any capacity loss. However, all previous JSCCs have been designed for representing symbols values that are selected from a set having a *low* cardinality and hence they suffer from an excessive decoding complexity, when the cardinality of the symbol value set is large, leading to an infinite complexity, when the cardinality is infinite.

Motivated by this, we propose the family of Unary Error Correction (UEC), Elias Gamma Error Correction (EGEC) and Reordered Elias Gamma Error Correction (REGEC) codes in this thesis. Our family of codes belong to the JSCC class designed to have only a modest complexity that is independent of the cardinality of the symbol value set. We exemplify the application of each of the codes in the context of a serially concatenated iterative decoding scheme. In each coding scheme, the encoder generates a bit sequence by encoding and concatenating codewords, while the decoder performs iterative decoding using the classic Logarithmic Bahl, Cocke, Jelinek and Raviv (Log-BCJR) algorithm. Owing to this, our proposed codes are capable of mitigating any potential capacity loss, hence facilitating near-capacity operation.

Our proposed UEC code is the first JSCC that maintains a low decoding complexity, when invoked for representing symbol values that are selected from a set having large or even infinite cardinality. The UEC trellis is designed to describe the unary codewords so that the transitions between its states are synchronous with the transitions between the consecutive codewords in the bit sequence. The unary code employed in the UEC code has a simple structure, which can be readily exploited

for error correction without requiring an excessive number of trellis transitions and states. However, the UEC scheme has found limited applications, since the unary code is not a *universal* code. This motivates the design of our EGEC code, which is the first *universal* code in our code family. The EGEC code relies on trellis representation of the EG code, which is generated by decomposing each symbol into two sub-symbols, for the sake of simplifying the structure of the EG code. However, the reliance on these two parts requires us to carefully tailor the Unequal Protection (UEP) of the two parts for the specific source probability distribution encountered, whilst the actual source distribution may be unknown or non-stationary. Additionally, the complex structure of the EGEC code may impose further disadvantages associated with an increased decoding delay, loss of synchronisation, capacity loss and increased complexity due to puncturing. This motivates us to propose a *universal* JSCC REGEC code, which has a significantly simpler structure than the EGEC code. The proposed codes were benchmarked against SSCC benchmarkers throughout this thesis and they were found to offer significant gains in all cases.

Finally, we demonstrate that our code family proposed in this thesis can be extended by several potential directions. The sophisticated techniques that have been subsequently proposed in the thesis for extending the UEC code, such as irregular trellis designs and the adaptive distribution-learning algorithm, can be readily applied to the REGEC codes which is an explicit benefit of its simple trellis structure. Furthermore, our proposed REGEC code can be extended using techniques that been subsequently proposed for extending the EGEC both to Rice Error Correction (RiceEC) codes and to Exponential Golomb Error Correction (ExpGEC) codes.

Declaration of Authorship

I, **Tao Wang**, declare that the thesis entitled

Elias Gamma Error Correction Code

and the work presented in it are my own and have been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published, as seen in the list of publications.

Signed:

Date:

Acknowledgements

I would like to express my heartfelt gratitude to my supervisors Professor Lajos Hanzo and Dr. Robert G. Maunder for their outstanding supervision and support throughout my four years study and research. Their patient guidance, continuous encouragement and inspiring advice have greatly benefited me not only in research but also in life.

I would also like to thank the co-authors of my papers: Robert G. Maunder, Lajos Hanzo, Yongkai Huo, Wenbo Zhang and Matthew F. Brejza for their willingness to help, discussions, knowledge.

Many thanks to my colleagues and the staff of the Southampton Wireless Group for the valuable discussions and comments throughout my research. Special thanks to my colleagues, Soon Xin Ng, Mohammed El-Hajjar, Chao Xu and all others not mentioned here for their technical support and collaborative work.

Finally, to my family, particularly my parents, for their unconditional love, support and care. Without you, I would not be where I am today.

Contents

[1]Some of the results in this chapter are reproduced from the collaborative work [1, 2]. The contribution to these papers by the author of this thesis was the video data collection and analysis, as well as the UEC trellis codebook error floor analysis in these papers.

List of Publications

Journal Paper

1. **T. Wang**, W. Zhang, M. F. Brejza, R. G. Maunder, and L. Hanzo, "Reordered Elias Gamma Error Correction Codes for the Near-Capacity Joint Source and Channel Coding of Multimedia Information," *to be submitted*, November 2015.
2. **T. Wang**, W. Zhang, R. G. Maunder, and L. Hanzo, "Near-capacity Joint Source and Channel coding of Symbol values from an Infinite Source Set using Elias Gamma Error Correction codes," IEEE Transactions on Communications, vol. 62, pp. 280–292, January 2014.
3. M. F. Brejza, **T. Wang**, W. Zhang, D. Al-Khalili, R. G. Maunder, B. M. Al-Hashimi and L. Hanzo, "Exponential Golomb and Rice Error Correction Codes for Near-Capacity Joint Source and Channel Coding", *to be submitted*, December 2015.
4. W. Zhang, Z. Song, M. F. Brejza, **T. Wang**, R. G. Maunder, and L. Hanzo, "Learning-aided Unary Error Correction Codes for Non-Stationary and Unknown Source", *to be submitted,* September 2015.
5. W. Zhang, M. F. Brejza, **T. Wang**, R. G. Maunder, and L. Hanzo, "An Irregular Trellis for the Near-Capacity Unary Error Correction Coding of Symbol Values for an Infinite Set," IEEE Transactions on Communications, in press.
6. Y. Huo, **T. Wang**, R. G. Maunder, and L. Hanzo, "Two-dimensional iterative source-channel decoding for distributed video coding," IEEE Communication Letters, vol. 18, pp. 90-93 January 2014.
7. Y. Huo, **T. Wang**, R. G. Maunder, and L. Hanzo, "Motion-aware mesh-structured trellis for correlation modelling aided distributed multi-view video coding," IEEE Transactions on Image Processing, vol. 23, pp.319-331,January 2014.
8. Y. Huo, **T. Wang**, R. G. Maunder, and L. Hanzo, "Iterative source and channel decoding relying on correlation modelling for wireless video transmission," IET Communications, vol. 7, pp. 1465–1475, September 2013.
9. R. G. Maunder, W. Zhang, **T. Wang**, and L. Hanzo, "A Unary Error Correction code for the Near-capacity Joint Source and Channel Coding of Symbol Values from an Infinite Set," IEEE Transactions on Communications, vol. 61, pp. 1977-1987, May 2013.

List of Symbols

Joint Source and Channel Error Correction Code(JSCECC)

a	number of sysbols.
b	number of bits.
$\mathbb{C}$	Trellis codewords set.
c	Trellis codeword.
$\mathbf{D}$	realization of vector $\mathbf{d}$ comprising with Independent and Identically Distributed (IID) Random Variables (RVs) .
$\mathbf{d}$	input symbol vector.
d_f	The free distance of a Error Correction (EC) code.
$\mathbf{F}$	Number of Unary bits have separate transition in the trellis.
f	Number of Unary States.
$\mathbf{H}$	The entropy.
h	Number of Fixed Length Code (FLC) States.
L	Symbol value limit for a finite-cardinality source set.
l	The average codeword length.
$\mathbf{m}$	The trellis path for encoding binary vector.
m_j	The trellis state.
n	The number of bit of trellis codeword.
$P(\cdot)$	The Probability function.
p_1	Probability of occurrence of the symbol value equal to 1 in a particular distribution.
R	Coding rate.
r	Number of trellis state.
$\mathbf{t}$	input sub-symbol vector for EGEC(UEC) encoder.
$\mathbf{T}$	realization of vector $\mathbf{t}$ comprising with IID RVs .
$\mathbf{u}$	binarization of sub-symbol vector $\mathbf{t}$.
$\mathbf{v}$	interlevered bit vector of binary vector $\mathbf{u}$.
$\mathbf{w}$	The encoded output bit vector for EGEC(FLC) encoder.
$\mathbf{X}$	realization of vector $\mathbf{x}$ comprising with IID RVs .
$\mathbf{x}$	input sub-symbol vector for EGEC(UEC) encoder.
$\mathbf{y}$	binarization of sub-symbol vector $\mathbf{x}$.
$\mathbf{z}$	The encoded output bit vector for EGEC(UEC) encoder.
$(\cdot)^a$	The *a prior* Logarithmic Likelihood Ratios (LLRs) pertaining to the symbol/bit vector .

$(\cdot)^e$	The *extrinsic* LLRs pertaining to the symbol/bit vector.
$(\cdot)^p$	The *a posteriori* LLRs pertaining to the symbol/bit vector.
$\hat{(\cdot)}$	Reconstrction of the symbol or bit vector having the symbol/bit vector.

Chapter 1

Introduction

Figure 1.1 illustrates the video transmission codec recommended in the first digital video coding standard published by the International Telecommunication Union (ITU), namely H.120 [3]. When the standard was proposed in 1984, this video transmission codec aimed for transmitting video conferencing signals. However, at the time these video schemes were required for wireline communication between fixed equipment. At the time of writing, typically mobile devices are used for videotelephony. Given the increasing popularity of video services on mobile devices, video codecs are applied in diverse scenarios. However, the structure of hybrid video codecs remained relatively unchanged over the decades, as exemplified by in state-of-the-art multimedia codecs, such as H.264 [4] and H.265 [5].

Uncompressed video sequences exhibit a high degree of intra-frame and inter-frame correlations [6], which represents redundancy. The predictive coding scheme of Figure 1.1 removes this redundancy in order to achieve compression and its compression ratio may be further improved with the aid of entropy coding. Hence in this thesis, we propose several novel Joint Source and Channel Code (JSCC) [7] schemes for exploiting the residual redundancy after source coding and hence to improve the

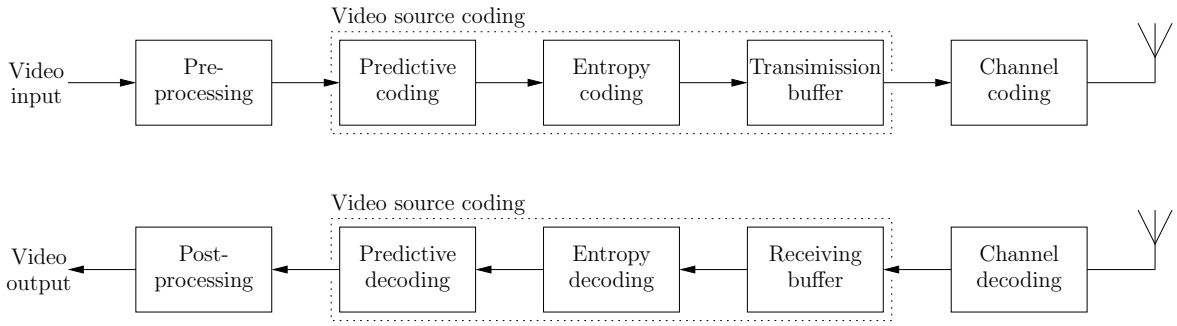


Figure 1.1: Video transmission using H.120 stardard [3].

error correction capability of the video transmission system.

In Section 1.1, we briefly review the history of video coding standards. Following this, we discuss the SSCC and JSCC philosophies in Section 1.2.1. Finally, we portray the outline and the novel contributions of the thesis in Section 1.3.

1.1 Brief history of video coding standards

The history of video coding standardization dates back to the 1980s. Over the past decades, several video coding standards have been proposed by the standardization bodies, namely by the ITU and by the International Standardization Organization/International Electrotechnical Commission (ISO/IEC). The ITU standards have mainly aimed for lip-synchronized interactive video communications, while ISO/IEC have focussed on delay-tolerant broadcast video distribution [8].

The ITU-T H.120 standard [3] was the first digital video coding standard proposed by the ITU in 1984. The codec was designed for transmitting the video at a similar resolution to that of the television standards of the time. Differential Pulse Code Modulation (DPCM) [6] was used as predictive coding, while scalar quantization of the pixels and a Variable Length Code (VLC) [9] was employed for achieving the compression. The standard also included error correction using Bose-Chaudhuri-Hocquenghem (BCH) [10, 11] codes for channel coding. The ITU-T H.261 standard [12] improved ITU-T H.120 and it was first published in 1988. This codec was designed for transmitting 352×288-pixel Common Intermediate Format [12] (Common Intermediate Format (CIF)) and for 176×144-pixel Quarter Common Intermediate Format [12] (Quarter Common Intermediate Format (QCIF)) video clips over Integrated Services Digital Networks (Integrated Services Digital Networks (ISDN)). This standard proposed the first so-called hybrid video coding scheme, which has then been adopted by all of its successors. The 'hybrid' terminology was used to indicated that motion-compensation was carried out in the spatio-temporal domain, followed by the residual encoding Discrete Cosine Transform (DCT) domain. We will discuss the structure of hybrid video coding in Section 1.2.1. In 1993, the Moving Picture Expert Group (MPEG), formed by ISO/IEO in 1988, released the MPEG-1 standard [13] for video distribution, which was also based on a hybrid coding scheme. However, MPEG-1 only supported progressively scanned images, preventing its employment for the interlaced video frames of the National Television System Committee's (National Television System Committee (NTSC)) standard or for the so-called Phase Alternating Line (Phase Alternating Line (PAL)) television formats [14]. Motivated by this, the MPEG-2 standard [15] was proposed in 1995, which was also adopted as H.262 [16]. This coding standard is the first joint video coding standard of two

standardization bodies. It supported both (720×576)–pixel and (720×480)-pixel resolutions, as well as high-definition (Hamming Distance (HD)) video with a resolution of 1920×1080 pixel. This standard was widely used in broadcasting for example in the European Digital Video Broadcasting standard − Terrestrial (DVB-T) [17]. After this joint work, the H.263 [18] and MPEG-4 [19] standards were individually released by ITU-T in 1996 and MPEG in 1999, respectively.

Although the target applications of the two main standardization bodies were different, most of the signal processing techniques invoked for real-time video communication and for broadcast of video are identical. Motivated by this, the Joint Video Team (JVT) was founded by the video experts of ITU and of ISO/IEC. Following this, JVT proposed the Advanced Video Coding (AVC) standard [4] in 2003, which is known as ITU-T H.264 and MPEG-4 part 10. Further extensions of the H.264/MPEG-4/AVC include Scalable Video Coding (SVC) [4, Annex G] [20] and Multiview Video Coding (MVC) [4, Annex H] [21], which were proposed in 2007 and 2009, respectively. As the collaboration between the ITU and ISO/IEC continued, they established the Joint Collaborative Team on Video Coding (JCT-VC) in 2010. The High Efficiency Video Coding (HEVC) standard [5] was published as a successor of H.264/AVC by JCT-VC in 2013, which is also known as ITU-T H.265/HEVC and ISO/IEC 23008-2 MPEG-H Part 2/HEVC. HEVC supports Full High Definition (FHD), Video Horizontal Resolution on the Order of 4,000 Pixels (4K) Ultra High Definition (UHD) and Video Horizontal Resolution on the Order of 8,000 Pixels (8K) UHD video formats. Compared to AVC, HEVC aims for halving the bitrate, while maintaining the same video quality. As in AVC, HEVC also includes SVC [5, Annex H] and MVC [5, Annex G] extensions.

1.2 Background on video transmission

The following discussions introduce the required background, which sets the scene for the introduction of the novel contributions of this thesis in Section 1.3.

1.2.1 Source coding

As we mentioned in Section 1.1, the H.261 standard proposed the first hybrid video coding scheme, which is the approach that has been adopted by all of its successors. The most important components in a hybrid video coding scheme are Motion Compensation (MC) [22] and the DCT [23] and quantization. Typically, a block-based MC [22] is employed as the basis of the predictive coding of Figure 1.1, in order to exploit temporal redundancy in the video source sequence. Additionally, the DCT [23] is employed for exploiting the spatial redundancy in the motion-compensated video

Table 1.1: The first twelve codewords of VLC table for motion vector data.

Symbol value	Motion vector	Codes
1	0	1
2	0.5	010
3	-0.5	011
4	1	0010
5	-1	0011
6	1.5	00010
7	-1.5	00011
8	2	0000110
9	-2	0000111
10	2.5	00001010
11	-2.5	00001011
12	3	00001000
⋮		⋮
61	15.5	0000000000110
62	-15.5	0000000000111
63	-16	0000000000101

data. Finally, carefully considered quantization is invoked for controlling the trade off between the reconstruction quality and the bit rate of the encoded video data.

Following this, entropy coding [24] may be used in order to reduce the number of bits in the encoded video stream. Until the H.263 standard was proposed, video codecs typically employed VLC tables for encoding the various parameters output by the predictive coding, such as DCT coefficients or motion vectors. Table 1.1[1] exmplifies a typical VLC table employed in video coding standards. Each symbol value in the VLC table indicates a mapping from a parameter also referred to in parlance as a syntax element to a binary codeword. In the example of Table 1.1, each symbol value represents a different motion vector value, where a step size of half a pixel is employed. In this way, a sequence of motion vectors may be converted into a sequence of codewords, which are then concatenated for generating the encoded bit sequence. For ensuring that a different codeword can be chosen for each physically realistic symbol value, the set of possible symbol values must have finite cardinality in order to design the VLC table. Similarly to entropy-coding or Huffman-coding, these codewords have to be generated by exploiting the knowledge of the probability of occurrence for each legitimate symbol value. In this way, the classic Huffman code [26] may be employed to design the codewords in the VLC table. However, encoding the syntax elements using a VLC table have a finite cardinality will restrict the set of possible syntax elements to also have a finite cardinality. For example, the

[1]This is a reordered version of [25, Table 14] for the sake of allowing convenient comparison with other codeword tables in this thesis.

Table 1.2: The first twelve codewords of the EG code

d_i	EG(d_i)
1	1
2	010
3	011
4	00100
5	00101
6	00110
7	00111
8	0001000
9	0001001
10	0001010
11	0001011
12	0001100
⋮	⋮

motion vector value is restricted to the range of [-16,15.5] in Table 1.1. By contrast, if this restriction can be removed, then the resultant unrestricted mode will facilitate a higher degree of flexibility for the predictive coding, hence improving its performance. In the example of Table 1.1, this would correspond to removing the limit imposed on the maximum number of symbol values representing the motion vectors, as in the so-called unrestricted motion vector mode of H.263. A symbol set without a pre-determined maximum symbol value may be considered to have an infinite cardinality. This prevents the employment of a Huffman code and of other near-entropy source codes, such as adaptive arithmetic codes [27] or a Lempel-Ziv code [28], since these require knowledge of the probability of all symbols corresponding to an infinite amount of knowledge.

Motivated by this, the Exponential Golomb (ExpG) code [29, 30] is employed in H.264 and H.265 codecs to encode both the syntax elements that reach large values and those having undetermined maximum values. The ExpG code [29] is a parametrized universal code, which subsumes the Elias Gamma (EG) code as a special case. Table 1.2 shows the codewords of the EG code. The set of ExpG codewords is infinite and structured, eliminating the requirement of explicit symbol probability knowledge during their design. Note that all syntax elements are encoded using the EG code in the H.264 Context-Based Adaptive Variable-Length Coding (CAVLC) mode [4, 31]. Additionally, EG code is also employed in the *binarization* process of the Context-Adaptive Binary Arithmetic Coding (CABAC) [32, 33] of H.264 and H.265 [4, 5]. A timeline of entropy coding milestones is shown in Figure 1.2.1.

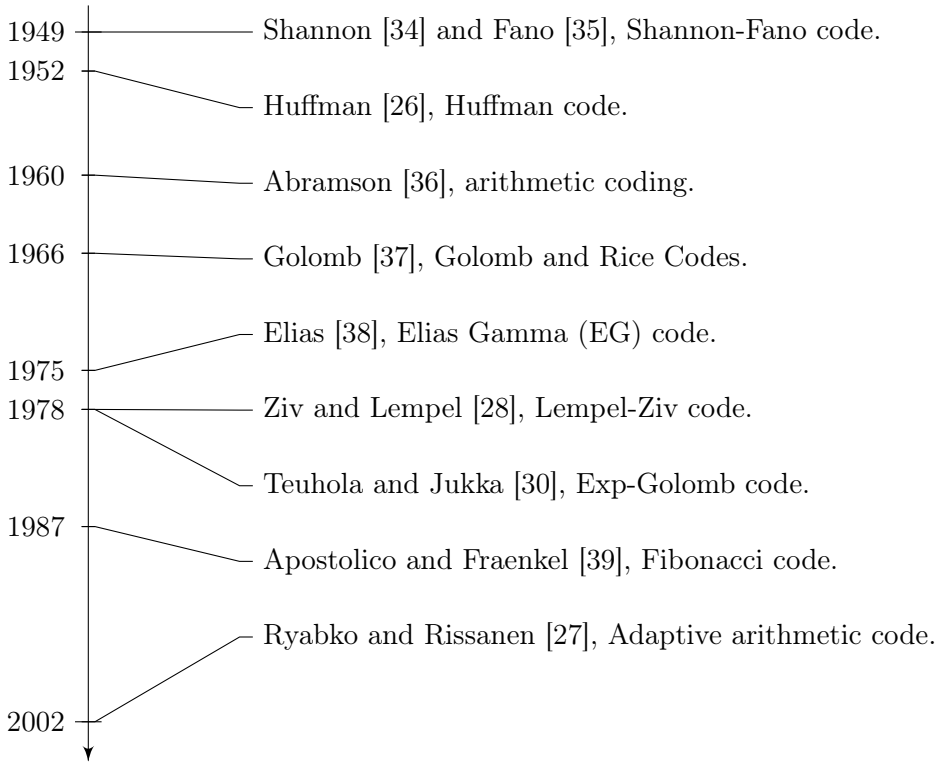


Figure 1.2: Timeline of entropy coding milestones.

1.2.2 Channel coding

As shown in Figure 1.1, channel coding is employed for protecting the video signal transmitted over realistic imperfect channel by providing an error correction capability. In contrast to source coding which removes redundancy, channel coding introduces specifically designed redundancy into the transmitted video signal. As a benefit, the receiver may correct the channel-induced errors by exploiting this redundancy. The Hamming code [40] was an early example of a block code [41], which encodes fixed-length blocks of bits at a time. Other well-known block codes include the so-called maximum-minimum-distance Reed-Solomon (RS) codes [42], BCH codes [11, 10] and the powerful Low Density Parity Check (LDPC) codes [43]. The other main type of channel coding is based on the classic Convolutional Code (CC) [44], which encodes streams of bits or symbols having an abitrary length. In particular, the state-of-the-art turbo codes [45] operate on the basis of CC codes.

Shannon's source-channel separation theorem [46] states that near-capacity communication is theoretically possible, when employing SSCC. For example, this may be achieved by combining a near-entropy source code, such as the adaptive arithmetic code [27] or the Lempel-Ziv code [28] of Figure 1.2.1, with a near-capacity channel code. State-of-the-art near-capacity channel codes, such as the LDPC [47]

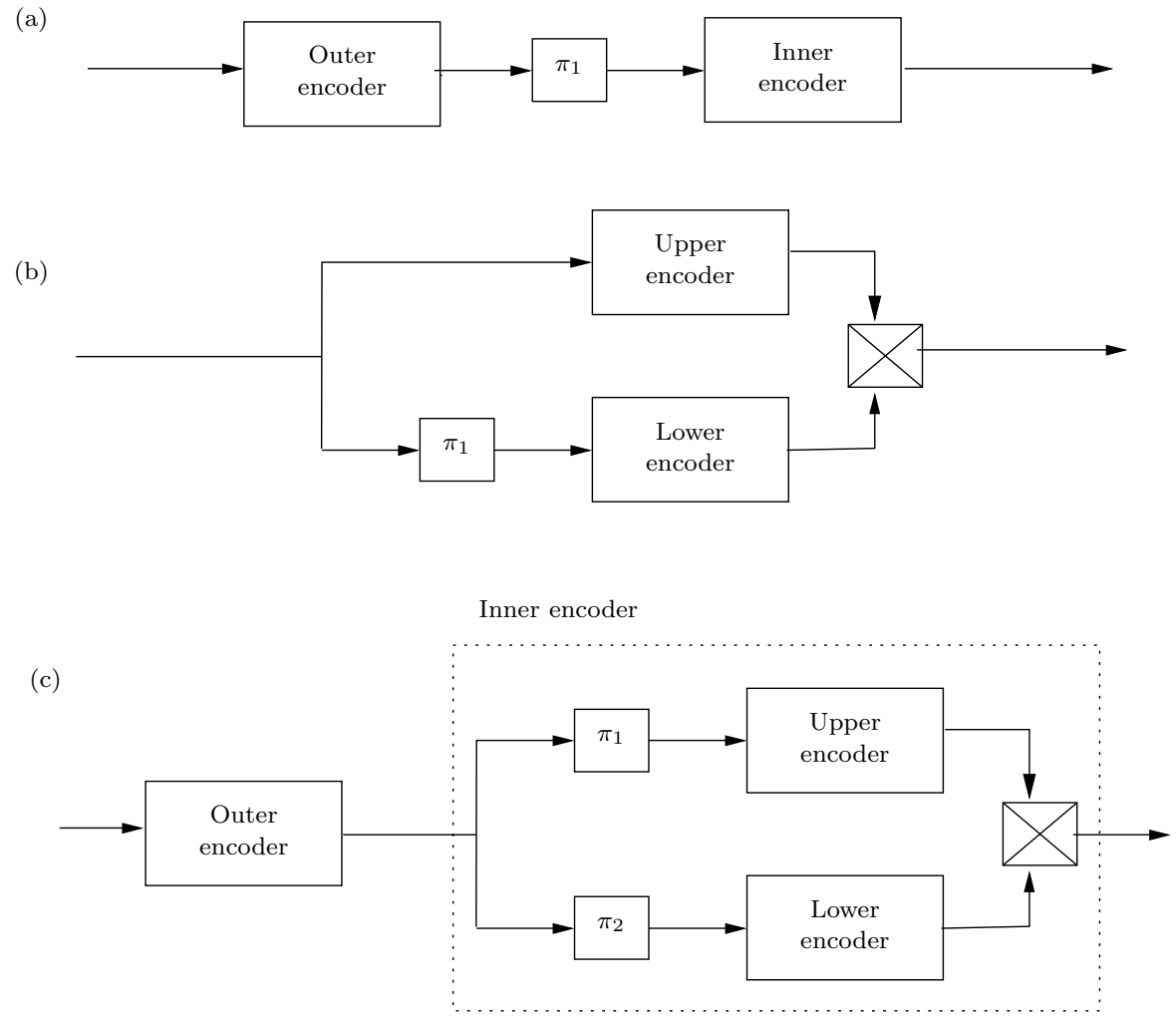


Figure 1.3: Encoder structure of (a) serial concatenation (b) parallel concatenation and (c) hybrid concatenation, where π_1 and π_2 represent interleaving operations.

and turbo codes [45] apply iterative decoding techniques for facilitating operation near the channel's capacity, which is the theoretical limit imposed upon the data rate, as will be discussed in Section 1.2.3. Inspired by this, this treatise is mainly focused on the family of concatenated schemes that have been conceived for iterative channel coding, including serial concatenation [48] like that of LDPC codes, parallel concatenation [45] like that of turbo codes and hybrid concatenation. Figure 1.3 shows the encoder structure of the serial concatenation, parallel concatenation and hybrid concatenation arrangements. In particular, we consider channel codes based on the CC, which operate on the basis of a trellis representation [49] and apply the Log-BCJR algorithm [50], which will be detailed in Sections 2.3 and 2.5, respectively. The error correction capability of a channel code may be characterized by its Free Distance (FD) [9], which is equal to the minimum number of differing bits in any pair of legitimate equal-length encoded sequence. This is because a channel code having a high FD is capable of tolerating more corrupted bits when decoding an error-infested

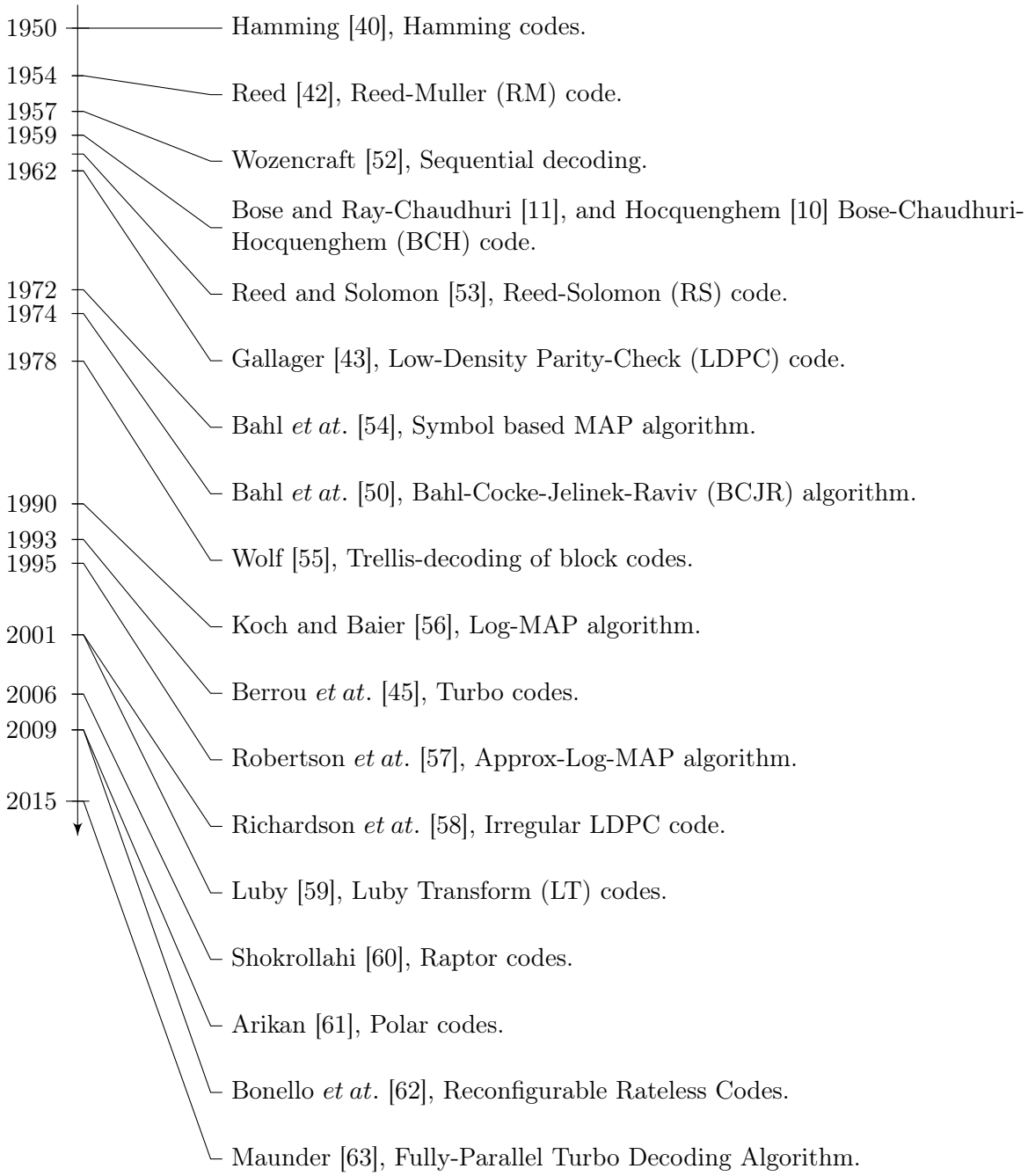


Figure 1.4: Timeline of channel coding milestones.

bit sequence. The FD of the channel codes considered in this thesis will be detailed in Sections 2.6.2 and 4.6.3. Additionally, we will employ EXIT chart analysis [51] for characterizing the iterative exchange of soft-information between the component decoders of our concatenated schemes, in order to design the component decoders, as it will be detailed in Section 2.5.2.2. A timeline of channel coding and decoding milestones is shown in Figure 1.2.2.

1.2.3 Channel capacity

The channel capacity of the noisy channel quantifies the upper bound on the rate (expressed in bits per second) [9] which can be reliably transmitted over the channel. The channel capacity is reduced upon reducing the channel's Signal to Noise Ratio (SNR), which depends on the transmit power, transmit distance the fading statistics, and the noise. In case of video transmission a certain maximum tolerable bit rate is associated with a minimum SNR threshold for the receiver to successfully decode the video sequence. Near-capacity operation is desirable, since it will reduce the SNR threshold, when transmitting the same video stream at a given quality. Capacity loss may be imposed both by source coding and channel coding. In Chapter 2, we will analyse the causes of capacity loss and we will consider several solutions for reducing this capacity loss in the following chapters.

1.2.4 Joint source-channel decoding

Shannon's source-channel separation theorem [46] states that near-capacity communication is theoretically possible, when employing SSCC, provided that an unlimited encoding/decoding delay and complexity can be afforded. However, it is typically impossible to remove all source redundancy with the aid of practical finite-delay and finite-complexity source encoding, which leads to capacity loss, as described in Section 1.2.3. As a potential remedy, JSCCs have been proposed for exploiting the residual redundancy and avoiding capacity loss [7].

JSCCs may be employed in two different scenarios. In the first scenario, the source encoder has extremely limited resources, hence the encoder will transmit the source information using only simple predictive coding or even uncompressed source streams. Thus the source correlation manifests itself in the transmitted sequence and may readily be exploited for error correction in the receiver. Inspired by this, the JSCCs scheme proposed in [64, 65] operated on the basis of iterative source-channel decoding by relying on the turbo principle [66, 67] and on the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [50], as well as on the Soft Bit Source Decoding (SBSD) algorithm [68]. Meanwhile, the schemes proposed in [69, 70, 71] exploited the spatial redundancy between a pixel in a video sequence and its eight neighbors using a Markov Random Field (MRF) model.

Another scenario for a JSCCs scheme is, where the source encoder employs complex predictive coding. In this case, although the source encoder will exploit most of the redundancy in the source for compression, some residual redundancy may

still persist due to the limited delay and complexity of this process, as we discussed above. Variable Length Error Correction (VLEC) codes [72] may be employed for exploiting this residual redundancy for error correction in the receiver, if the symbol values are selected from a set having a *low* cardinality and provided that accurate knowledge of the symbol value probabilities is available for designing the codebook of the VLEC code. However, as we discussed above, the symbol set of state-of-the-art multimedia codecs operating in the above-mentioned unrestricted mode have infinite or very large cardinalities. These symbol value sets are impractical for existing JSCCs, such as Self-Synchronizing Variable Length Codes (SSVLCs) [73], Reversible Variable Length Codes (RVLCs) [74], VLEC codes [75], Even Weight Variable Length Codes (EWVLCs) [76] and Irregular Variable Length Codes (IrVLCs) [72]. More specifically, these codes operate on the basis of trellis and graph structures [77, 78, 79, 80, 81, 82, 83] that become exponentially more complex, when the cardinality of the symbol value set grows. Motivated by this, this treatise will propose several JSCCs, where the decoder employs a trellis that has only a modest complexity, even when the cardinality of the symbol value set is infinite. The major milestones in the development of JSCCs are listed in Figure 1.2.4.

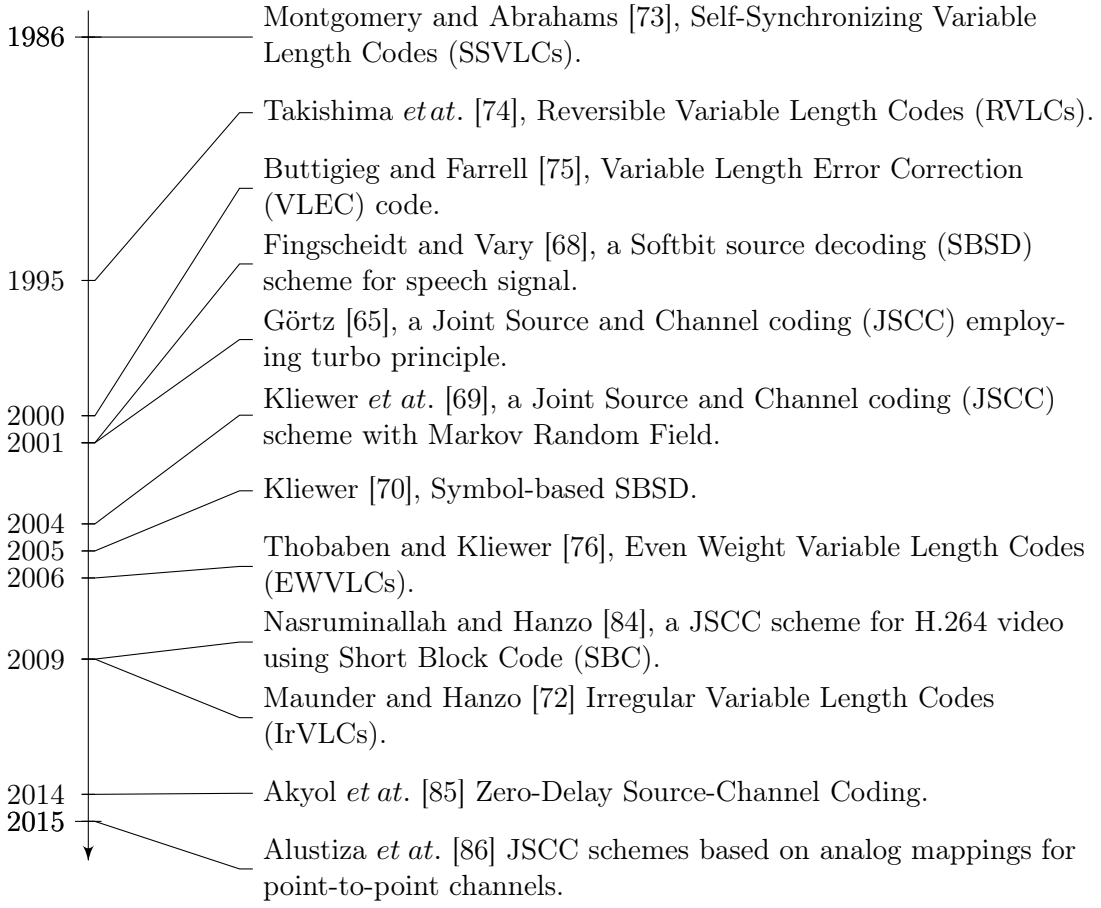


Figure 1.5: Timeline of Joint Source and Channel Coding milestones.

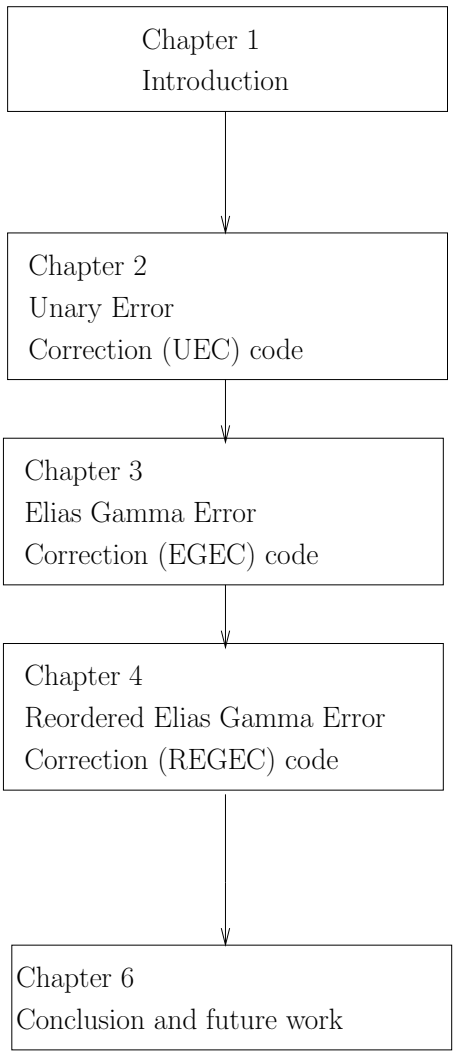


Figure 1.6: Outline of the thesis

1.3 Structure and novel contributions of the thesis

In this section we provide an overview of the remainder of this thesis and summarise the novel contributions of each chapter. Figure 1.6 shows the outline of this treatise. Through Chapters 2 to 4, we propose several novel JSCC schemes. Figure 1.7 illustrates the components that must be considered, when designing an iterative JSCC scheme. In each of the following chapters, we will focus our attention on one or more components listed in Figure 1.7.

In Chapter 2, we review the encoding and decoding operations of the UEC code and we exemplify the application of the UEC code for JSCC [7] in the context of a serially concatenated iterative decoding scheme. This chapter also serves as a background chapter, since it introduces the principles that are common to our family of JSCC codes conceived in Chapters 2 to 4. We detail the collection of video data and we model its symbol value probability distribution using the Zeta probability

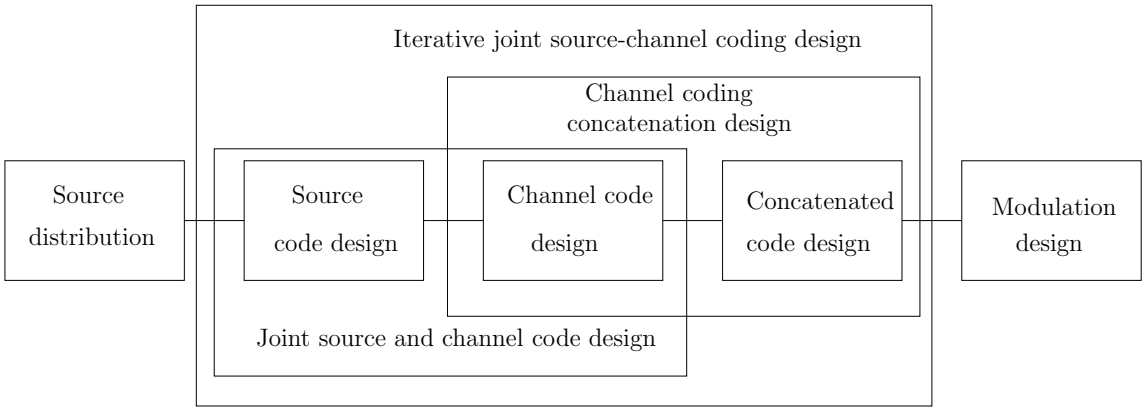


Figure 1.7: The components that must be considered when designing an iterative joint source and channel coding scheme.

distribution. Figure 2.2 illustrates the structure of the UEC code. We describe the operation of the unary encoder and trellis encoder structure of the UEC encoder in Section 2.3. Furthermore, we describe how to integrate the UEC encoder into a transmitter by describing the operation of the concatenated URC encoder [87], as well as of the interleaver and modulator. Additionally, we describe the operation of the UEC decoder and URC decoder, as well as the Log-BCJR algorithm [50], which forms the basis of the UEC trellis and URC decoding processes. Following this, the EXIT chart [51] concept is introduced and area properties of the UEC code are characterized. The novel contributions of Chapter 2 are as follows.

- We characterize and analyze the distribution of the symbol values produced, when test video sequences are encoded by the H.264 and H.265 video encoders.
- We characterize various candidate parametrizations of the UEC code in terms of the minimum channel SNR required for facilitating iterative decoding convergence to a low error probability (the open tunnel bound) and quantify how low that error probability is (the error floor).
- The error correction performance of our JSCC UEC scheme is comprehensively compared to that of an SSCC benchmarker based on the combination of an EG source code and a convolutional channel code.

In Chapter 3, we propose a *universal* JSCC EGEC code. In contrast to the UEC code of Chapter 2, the EGEC code achieves a finite average codeword length for all Zeta distributions. Firstly, we describe the decomposition of symbols into pairs of sub-symbols, which are then encoded separately by two distinct sub-encoders. Following this, we describe the operation of these sub-encoders, as well as of the corresponding sub-decoders. Furthermore, we detail the procedure of designing an UEP schemes

that capable of optimizing the contributions of the turbo sub-codes, facilitating near-capacity operation at a low decoder complexity. The novel contributions of Chapter 2 are as follows.

- A *universal* JSCC EGEC is designed, which is capable of achieving the near-capacity transmission of symbols that are randomly selected from infinite-cardinality symbol alphabets using *any* arbitrary monotonic probability distribution.
- We propose a UEP scheme for optimizing the relative contribution of the two EGEC sub-codes to the encoding process, facilitating near-capacity operation at a low decoder complexity.
- The performance of the proposed JSCC EGEC scheme is compared to that of the JSCC UEC and SSCC EG-CC benchmarkers in five scenarios, namely for four different Zeta distribution parametrizations and the H.265 distribution of Figure 2.3(b).

In Chapter 4, we propose a *universal* JSCC REGEC code, which has a significantly simpler structure than the EGEC code of Chapter 3. We describe the Zeta source probability distribution and we generalise the infinite cardinality source alphabet of our previous chapters to the case of a finite cardinality, where this cardinality represents an additional parameter to be considered. Furthermore, we introduce the novel Reordered Elias Gamma (REG) source code and describe the structure of the REG codewords. Following this, we introduce our novel REGEC encoder and decoder. Additionally, we analyze the parametrization of the proposed REGEC scheme and demonstrate that it facilitates near-capacity operation. The novel contributions of Chapter 2 are as follows.

- The REGEC *universal* JSCC is designed, which is capable of achieving the near-capacity transmission of symbols that are randomly selected from large or infinite cardinality symbol alphabets using *any* arbitrary monotonic probability distribution.
- We propose a finite Zeta-like distribution for modeling the distribution of the symbol values produced, when test video sequences are encoded by H.265 video encoders.
- A novel REG code is proposed by reordering the bits of the EG code in order to yield a simple codeword structure, which is suitable for trellis representation.
- We introduce a REGEC trellis, which is capable of describing the structure of our proposed REG code using a relatively small number of trellis states. Owing to this, our REGEC decoder has a low decoding complexity.

- We characterize various candidate parametrizations of the REGEC code, considering the corresponding inner code in terms of the SNR required for facilitating iterative decoding convergence to a low error probability (the open tunnel bound) and how low that error probability is (the error floor).
- A wide range of finite Zeta-like distributions and the H.265 distribution of Figure 2.3(b) are considered, when comparing the performance of the proposed JSCC REGEC scheme to that of the JSCC UEC and EGEC schemes, as well as the SSCC EG-CC benchmarker.

In Chapter 5, we summarise the thesis and the major findings of our work, as well as offer several potential directions for future work.

Chapter 2

Unary Error Correction Codes [1]

2.1 Introduction

In this chapter, we introduce the structure of a novel Joint Source and Channel Code (JSCC) [7] Unary Error Correction (UEC) codes [1, 2]. This was the first JSCC that has a low decoding complexity, when invoked for representing symbols values that are selected from an alphabet having a large or infinite cardinality. Near-capacity operation is facilitated, when our UEC code is serially concatenated with a Unity Rate Convolutional (URC) code [87], allowing an iterative exchange of increasingly reliable extrinsic information between the corresponding decoders [1]. Note that the JSCC schemes of Chapters 3 and 4 will also operate on the basis of serial concatenation and iterative decoding. Therefore, this chapter also serves as a background chapter and we will use the UEC code as an example to illustrate the structure of our family of JSCC codes, as well as to illustrate their encoding and decoding operations. Furthermore, this chapter will introduce the background knowledge required for designing and analyzing a JSCC scheme. More specifically, we will discuss the techniques listed in the boxes of the Figure 2.1, which we will refer back to in the following chapters.

2.1.1 Background and motivation

Shannon's source-channel separation theorem [46] states that near-capacity communication is theoretically possible, when employing Separate Source and Channel Code (SSCC). For example, this may be achieved by combining a near-entropy source code, such as an adaptive arithmetic code [27] or a Lempel-Ziv code [28], with a near-capacity channel code, such as a Low Density Parity Check (LDPC) code [47]

[1] Some of the results in this chapter are reproduced from the collaborative work [1, 2]. The contribution to these papers by the author of this thesis was the video data collection and analysis, as well as the UEC trellis codebook error floor analysis in these papers.

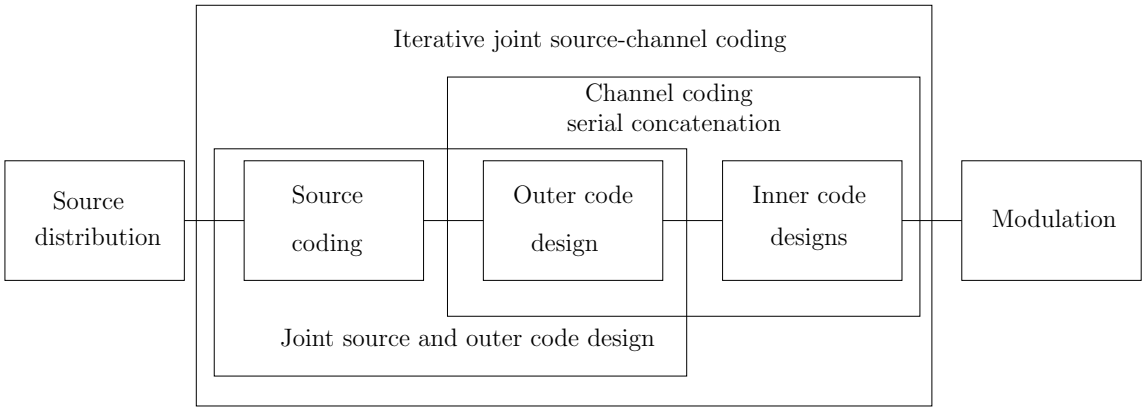


Figure 2.1: The components that must be considered when designing an iterative joint source and channel coding scheme based on serial concatenation.

or a turbo code [45]. However, the source-channel separation theorem relies upon a number of assumptions, which may not be valid in practice [9]. For example, near-entropy adaptive arithmetic coding or Lempel-Ziv coding requires both the transmitter and receiver to accurately estimate the occurrence probability of every value that is adopted by the symbols that the source produces. However, the occurrence probability of rare symbol values cannot be accurately estimated until a sufficiently high number of symbols have been generated, imposing an excessive latency which cannot be tolerated in many practical applications. This problem becomes particularly severe, when the symbol values are selected from a set having an infinite cardinality, such as the set of all positive integers. Furthermore, transmission errors may cause the estimated symbol probabilities to become desynchronized between the transmitter and receiver, potentially causing an avalanche-like propagation of decoding errors.

These issues motivate the design of universal source codes, such as the Elias Gamma (EG) code [38]. These codes facilitate the communication of symbols selected from infinite sets, without requiring any knowledge of the corresponding occurrence probabilities at either the transmitter or receiver. Other examples of universal codes include the Elias delta code [38], the Elias omega code [38], the Even-Rodeh code [88], the Stout code [89] and the Fibonacci code [90]. Furthermore, the Exponential Golomb (ExpG) code [30] is a parametrized universal code, which subsumes the EG code as a special case. Universal codes are typically employed in multimedia codecs, such as the H.264 video codec [4], where they are employed for encoding the values of various symbols, such as motion vectors. However, typically some residual redundancy remains in the source-coded bit-stream when EG codes are employed for representing symbols that are produced by multimedia codes, hence imposing a capacity loss and preventing near-capacity operation when SSCC is employed [1].

Furthermore, SSCC is sensitive to transmission errors, with a single bit error potentially causing the corruption of several video frames in H.264, for example.

Motivated by this, various JSCCs have been proposed for mitigating the impact of transmission errors, as well as for mitigating the capacity loss that is imposed by residual redundancy. However, all previous JSCCs have been designed for representing symbols values that are selected from a set having a *low* cardinality and they suffer from an excessive decoding complexity, when the cardinality of the symbol value set is large, leading to an infinite complexity, when the cardinality is infinite [1]. For example, the complexity of Variable Length Error Correction (VLEC) codes was characterized in [72], which was shown to increase rapidly with the cardinality of the symbol value set. This motivates our UEC code [1, 2] of Figure 2.2 , which is the

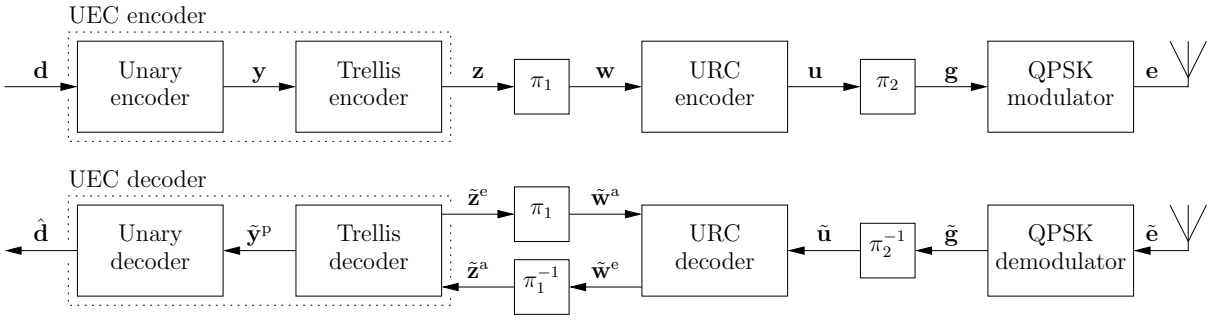


Figure 2.2: Schematic of the UEC code, when serially concatenated with URC and Gray-coded QPSK modulation schemes. Bold notation without a diacritic is used to denote a symbol vector or a bit vector. A diacritical hat represents a reconstruction of the symbol or bit vector having the corresponding notation. A diacritical tilde represents an LLR vector pertaining to the bit vector with the corresponding notation. A roman superscript 'a' is employed to denote an *a priori* LLR vector, while 'e' is employed for extrinsic LLR vectors. Furthermore, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. Puncturing may also be performed in π_2, while the corresponding depuncturing operations take place in π_2^{-1}.

first JSCC that maintains a low decoding complexity, when employed for representing symbols values that are selected from a set having large or infinite cardinalities. When the channel's Signal to Noise Ratio (SNR) is sufficiently high, our UEC code facilitates reliable communication, without requiring any knowledge of the symbol value occurrence probabilities at either the transmitter or receiver. However, once the UEC decoder has succeeded in recovering a sufficiently high number of source symbols, the receiver may be capable of improving the attainable performance by estimating the occurrence probabilities of the symbol values. This knowledge may then be invoked for exploiting the remaining residual redundancy for error correction, hence facilitating reliable communication at near-capacity SNRs.

2.1.2 Novel contributions

In addition to outlining the background of UEC coding, this chapter details the author's novel contributions, listed as follows.

- We characterize and analyze the distribution of the symbol values produced, when test video sequences are encoded by the H.264 and H.265 video encoders.
- We characterize various candidate parametrizations of the UEC code in terms of the SNR required to facilitate iterative decoding convergence to a low error probability (the open tunnel bound) and how low that error probability becomes at high SNR (the error floor).
- The error correction performance of our JSCC UEC scheme is compared to that of an SSCC benchmark based on the EG source code and a convolutional channel code.

2.1.3 Chapter organisation

The rest of this chapter is organised as follows.

- In Section 2.2, we characterize the H.264 and H.265 source symbol value distributions and justify their modelling using the Zeta probability distribution.
- In Section 2.3, we introduce the proposed UEC encoder, detailing the operation of its unary encoder and trellis encoder. Furthermore, we describe the integration of the UEC encoder into a transmitter by introducing the operation of the interleaver, concatenated URC encoder and modulator.
- In Section 2.4, we introduce the wireless channel model used in our simulations, namely the uncorrelated narrow-band Rayleigh fading channel.
- In Section 2.5, we describe the integration of the UEC decoder into a receiver by detailing the operation of the demodulator, deinterleaver and URC decoder. Following this, we detail the operation of the UEC trellis decoder as well as of the iterative decoding process.
- In Section 2.6, we discuss the UEC's capability of facilitating near capacity operation, as well as the parameterization of the UEC.
- In Section 2.7, we introduce the SSCC EG-CC-URC benchmarker and compare its error correction performance to that of our UEC scheme.
- In Section 2.8, we conclude this chapter.

2.2 Source distribution

The Golomb code [37], Rice code [91] and EG code [38] facilitate the communication of symbols having values selected from infinite sets, without requiring any knowledge

of the corresponding occurrence probabilities at either the transmitter or receiver. Furthermore, the Golomb code is optimal if the source symbol probabilities obey the geometric distribution [92]. In this case, a separate Golomb code and channel code can achieve near capacity error correction performance without any capacity loss, provided that the receiver has the knowledge of how the geometric distribution is parametrized. The Golomb code itself can be parametrized and its simplest parametrization is identical to the special case constituted by the unary code. Likewise, the ExpG code [30] can be parametrized and the simplest parametrization yields the EG code as a special case. These codes have found application in multimedia codecs, such as the H.264 [4] and H.265 [5] video codec, where they are employed for encoding the values of various so-called syntax elements, which are symbols produced by motion vectors, for example [8]. In the following subsections, we will characterise the probability distributions of the symbols produced by the H.264 and H.265 codecs, as well as propose the employment of the Zeta probability distribution to model them.

2.2.1 Symbols value sets from video compression standards

The codes described above are designed to encode symbols that have values selected from the set of all positive integers. However, many of the syntax elements in the video codes have negative integer and zero values. Owing to this, a mapping rule may be used to map all the negative and zero-valued integers into the positive integer interval, as shown in Table 2.1. In the case of having syntax elements that do not have

syntax element	remapped symbol value
0	1
-1	2
1	3
-2	4
2	5
-3	6
3	7

Table 2.1: Mapping rule for mapping the negative integers to positive integers

any negative values, but include the value of zero, the above-mentioned remapping may be achieved by adding one to the value of each syntax element. Figure 2.3 demonstrates that both the H.264 and H.265 video encoders produce symbol values that may be represented using positive integers having values of up to about 1000, where higher values are observed with lower probabilities. In practice, the symbols may have values above 1000, albeit with low probabilities. Note that the H.264 and H.265 probability distributions have a near-constant gradient, when plotted on the log-log axes of Figure 2.3. This is a manifestation of Zipf's law [93], hence the

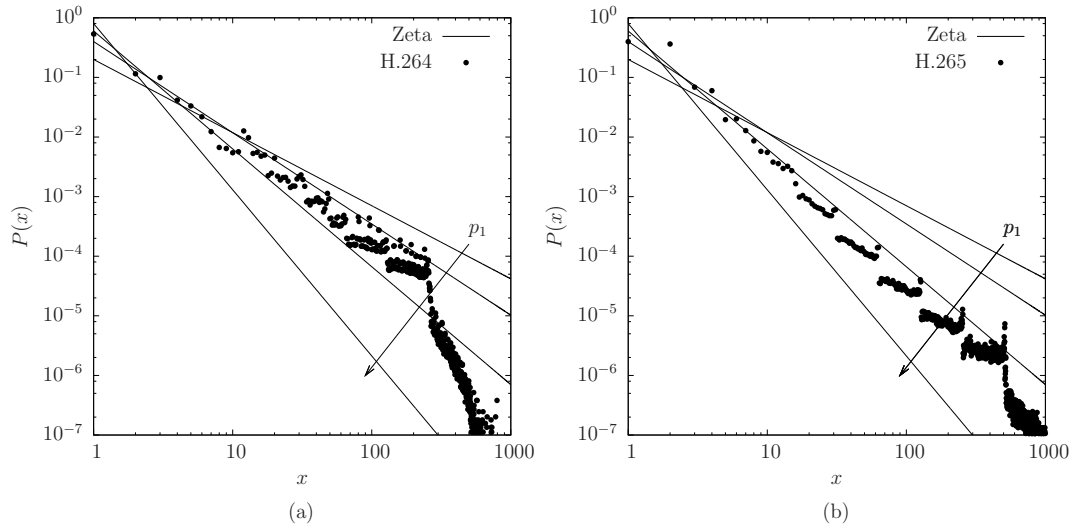


Figure 2.3: The Zeta probability distributions for $p_1 \in \{0.2, 0.4, 0.6, 0.8\}$, as well as the (a) H.264 distribution which was obtained by recording the values of the 44.6 million symbols that are EG encoded when the JM 18.2 H.264 video encoder employs the 'encoder_baseline.cfg' configuration to encode the 175 s of video that are comprised by 4:2:0 versions of the Video Quality Expert Group (VQEG) test sequences and (b) H.265 distribution which was obtained by recording the values of the 112.9 million symbols when the HM-9.0 H.265 video encoder employs the 'encoder_lowdelay_main.cfg' and 'encoder_random_access_main.cfg' configurations to encode the 220 s of video that are comprised by 4:2:0 versions of the 24 video test sequences that are commonly used for testing in H.265 [8, page 94].

symbol values may be modeled using a Zeta probability distribution [93], as it will be discussed in the following subsection.

2.2.2 Symbols value sets having an infinite cardinality

The UEC scheme considered in this this chapter is designed to convey a vector $\mathbf{d} = [d_i]_{i=1}^{a}$ comprising a number of symbols. This symbol vector is obtained as the realization of a corresponding vector $\mathbf{D} = [D_i]_{i=1}^{a}$ of Independent and Identically Distributed (IID) Random Variables (RVs). Each RV D_i adopts the symbol value $d \in \mathbb{N}_1$ with probability $\Pr(D_i = d) = P(d)$, where $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$ is the infinite-cardinality set comprising all positive integers. Here, the symbol entropy is given by

$$H_D = \sum_{d \in \mathbb{N}_1} H[P(d)], \tag{2.1}$$

where $H[p] = p \log_2(1/p)$.

Figure 2.3 depicts the Zeta distribution [94], which is defined as

$$P(d) = \frac{d^{-s}}{\zeta(s)}, \tag{2.2}$$

where $\zeta(s) = \sum_{d\in\mathbb{N}_1} d^{-s}$ is the Riemann Zeta function, $s > 1$ and $p_1 = 1/\zeta(s)$. Here, the Zeta distribution is parametrized by s, which gives the negative gradient of the distribution plotted versus the log-log axes of Figure 2.3. Equivalently and more conveniently, the Zeta distribution may be parametrized by p_1, which gives the probability of the most frequently occurring symbol value, namely $\Pr(D_i = 1)$. In the case, where the RVs of $\mathbf{D}$ obey the Zeta distribution, the symbol entropy is given by

$$H_D = \frac{\ln(\zeta(s))}{\ln(2)} - \frac{s\zeta'(s)}{\ln(2)\zeta(s)}, \tag{2.3}$$

where $\zeta'(s) = -\sum_{d\in\mathbb{N}_1} \ln(x)x^{-s}$ is the derivative of the Riemann Zeta function.

Figure 2.3 plots the Zeta distribution for $p_1 \in \{0.2, 0.4, 0.6, 0.8\}$, which crrespond to parameter values of $s \in \{1.2269, 1.5303, 1.9774, 2.7884\}$ and symbol entropies of $H_D \in \{9.1711, 4.4022, 2.4215, 1.1541\}$ bits per symbol, respectively. By comparison, the H.264 distribution of Figure 2.3(a) corresponds to a symbol entropy of $H_X = 2.980$ bits per symbol, while the symbol values of the H.265 distribution have an entropy of $H_D = 2.3922$ bits per symbol.

2.3 UEC encoder

As shown in Figure 2.2, the UEC encoder comprises a unary encoder and a UEC trellis encoder. The UEC encoder may be integrated into a transmitter by serially concatenating it with a URC and a Quaternary Phase Shift Keying (QPSK) modulator, where these serially concatenated components are separated by interleavers, as shown in Figure 2.2. In the following subsections, we will detail the operation of the above-mentioned components.

2.3.1 Unary encoder

As shown in Table 2.2, unary encoders represent each symbol d_i in the vector $\mathbf{d}$ using a corresponding binary codeword, namely Unary(d). Note that for the convenience of our discussions in the following chapters, the unary codewords shown in Table 2.2 are the complements of those that are conventionally employed, for example in [1, Table I]. Note that the d_i^{th} codeword Unary(d_i) comprises $l_{Unary}(d_i) = d_i$ number of bits, namely $(d-1)$ zeros followed by a single logical one-valued bit. Since different symbol values and hence different codewords occur with different probabilites, the average codeword length of a binary code like the unary code is given by

$$l = \sum_{d\in\mathbb{N}_1} P(d)l(d). \tag{2.4}$$

Table 2.2: The first twelve codewords of the unary and EG codes

d_i	Unary(d_i)	EG(d_i)
1	1	1
2	01	010
3	001	011
4	0001	00100
5	00001	00101
6	000001	00110
7	0000001	00111
8	00000001	0001000
9	000000001	0001001
10	0000000001	0001010
11	00000000001	0001011
12	000000000001	0001100
⋮	⋮	⋮

In the case where the source symbols obey the Zeta distribution of (2.2), the average unary codeword length is given by

$$l_{Unary} = \zeta(s-1)/\zeta(s). \tag{2.5}$$

Note that, the average unary codeword length is only finite for $s > 2$ and hence for $p_1 > 0.608$. Therefore, the unary code and hence the UEC code are not universal codes. More specifically, they have infinite average codeword lengths for many source distributions, including the Zeta distributions having $p_1 \leqslant 0.608$. This problem associated with the UEC motivates the Elias Gamma Error Correction (EGEC) and Reordered Elias Gamma Error Correction (REGEC) codes proposed in Chapter 3 and 4, which are by definition universal codes, since they have finite average codeword lengths for any monotonic source probability distribution.

The unary encoder of Figure 2.2 represents each symbol d_i in the vector $\mathbf{d}$ using the corresponding unary codeword Unary(d_i), as shown in Table 2.2. These codewords are then concatenated to obtain the b-bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$ shown in Figure 2.2, where we have $b = \sum_{i=1}^{a} d_i$. For example, the vector $\mathbf{d} = [2, 1, 3, 4, 1, 1, 2, 1]$ of $a = 8$ symbols yields the $b = 15$-bit vector $\mathbf{y} = 011001000111011$.

2.3.2 Trellis encoder

As shown in Figure 2.2, the bit vector of concatenated unary codewords $\mathbf{y}$ is forwarded to a trellis encoder. This employs a UEC trellis of the sort depicted in Figure 2.4(a) to encode the value of each bit y_j in the vector $\mathbf{y}$, in order of increasing bit-index j. Each bit forces the trellis encoder to traverse from its particular previous state

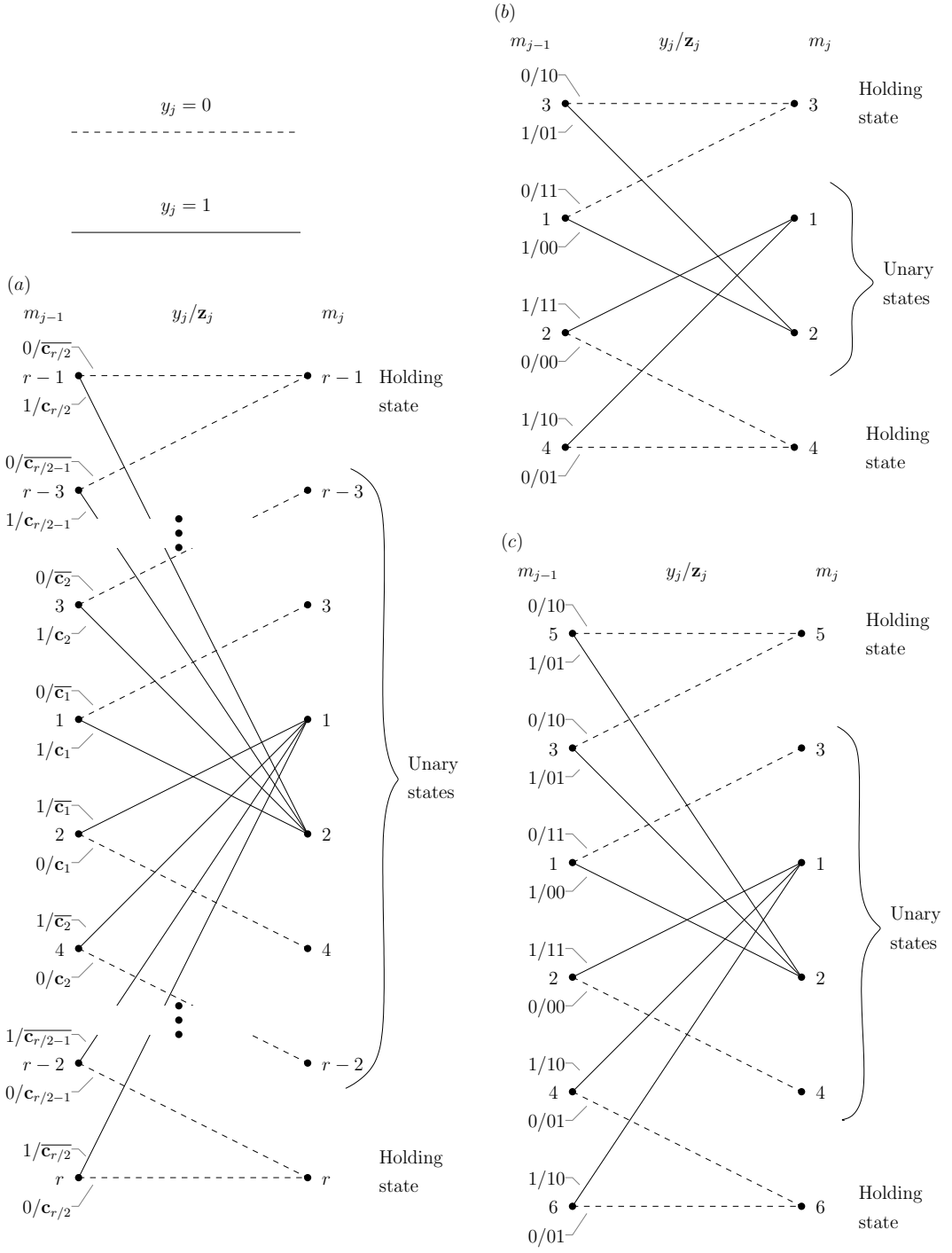


Figure 2.4: (a) The generalized UEC trellis, having r states and n-bit codewords, where $\mathbb{C} = [\mathbf{c}_1; \mathbf{c}_2; \ldots; \mathbf{c}_{r/2-1}; \mathbf{c}_{r/2}]$. (c) An $r = 4$-state $n = 2$-bit UEC trellis, where $\mathbb{C} = [00; 01]$. (d) An $r = 6$-state $n = 2$-bit UEC trellis, where $\mathbb{C} = [00; 01; 01]$.

$m_{j-1} \in \{1, 2, \ldots, r\}$ to a new state $m_j \in \{1, 2, \ldots, r\}$ that is selected from two legitimate alternatives, depending on the bit value y_j. More specifically, we have

$$m_j = \begin{cases} \min[m_{j-1} + 2, r - \text{odd}(m_{j-1})] & \text{if } y_j = 0 \\ 1 + \text{odd}(m_{j-1}) & \text{if } y_j = 1 \end{cases}, \tag{2.6}$$

where the number of possible states r is required to be even and the encoding process commences from the state $m_0 = 1$, with the function odd$(\cdot)$ yielding 1, if the operand is odd or 0 if it is even. In this way, the bit vector $\mathbf{y}$ identifies a path through the trellis, which may be represented by a vector $\mathbf{m} = [m_j]_{j=0}^{b}$ comprising $(b + 1)$ of state values. For example, the bit vector $\mathbf{y} = 011001000111011$ corresponds to the path $\mathbf{m} = [1, 3, 2, 1, 3, 3, 2, 4, 4, 4, 1, 2, 1, 3, 2, 1]$ through the $r = 4$-state trellis of Figure 2.4(b), as shown in Figure 2.5. The path created by the first six bits in this

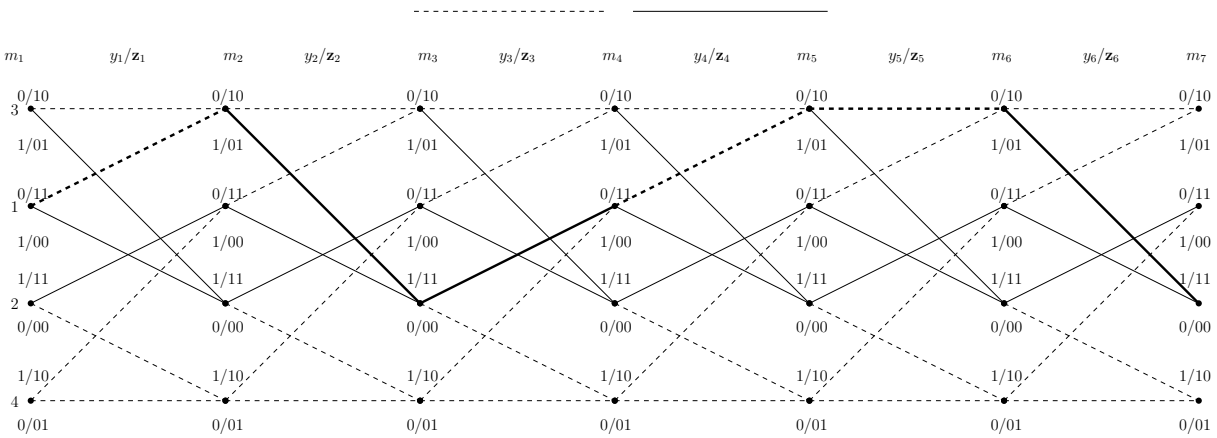


Figure 2.5: The trellis of the UEC code shown in Figure 2.4(b). Here the dashed transitions correspond to an input bit of $y_j = 0$ while the solid transitions correspond to an input bit of $y_j = 1$. In the label of each transition, the bit before the slash is the input bit y_j, while the bits after the slash are the output bits $\mathbf{Z}_j$. The transitions highlighted in bold indicate the path taken when encoding the input bit vector $\mathbf{y} = [011001]$.

example bit vector y is shown in Figure 2.5.

The trellis path $\mathbf{m}$ may be modeled as a particular realization of a vector $\mathbf{M} = [M_j]_{j=0}^{b}$ comprising (b+1) RVs, which are associated with the transition probabilities $\Pr(M_j = m, M_{j-1} = m') = P(m, m')$ of (2.7). These transition probabilities may be derived by observing that the UEC trellis of Figure 2.4(a) is designed so that the transitions between its states are synchronous with the transitions between the consecutive unary codewords in the bit vector $\mathbf{y}$. For example, the trellis path will merge into the state $m_j = 2$ if and only if y_j is constituted by the last bit of a unary codeword $\mathbf{y}_i$ having an odd symbol-index i. Furthermore, Figure 2.4(a) shows that when $r \geq 4$, the above-mentioned transition will emerge from the particular state $m_{j-1} = 1$, if and only if the corresponding symbol has the particular value of

$$P(m,m') = \begin{cases} \frac{1}{2l}\left[l - \frac{r}{2} + \sum_{d=1}^{\frac{r}{2}-1} P(d)\left(\frac{r}{2}-d\right)\right] & \text{if } \left\lceil \frac{m'}{2} \right\rceil = \frac{r}{2},\ m = m' \\ \frac{1}{2l}\left[1 - \sum_{d=1}^{\frac{r}{2}-1} P(d)\right] & \text{if } \left\lceil \frac{m'}{2} \right\rceil = \frac{r}{2},\ m = 1 + \text{odd}(m') \\ \frac{1}{2l}\left[1 - \sum_{d=1}^{\left\lceil \frac{m'}{2} \right\rceil} P(d)\right] & \text{if } \left\lceil \frac{m'}{2} \right\rceil < \frac{r}{2},\ m = m' + 2 \\ \frac{1}{2l} P(d)\Big|_{d=\left\lceil \frac{m'}{2} \right\rceil} & \text{if } \left\lceil \frac{m'}{2} \right\rceil < \frac{r}{2},\ m = 1 + \text{odd}(m') \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

$x_i = 1$. Hence, there will be a one-to-one correspondence between the transitions of this type and the symbols having an odd symbol-index i and simultaneously the particular value $x_i = 1$. Since we may expect $ap_1/2$ of the a symbols in the vector $\mathbf{x}$ to simultaneously have an odd symbol-index i and the particular value $x_i = 1$, we may expect $ap_1/2$ of the b transitions in the path $\mathbf{m}$ to be of the described type. Therefore, in the specific case, where $r \geq 4$, we obtain $P(M_j = 2, M_{j-1} = 1) = \frac{ap_1}{2b} = \frac{p_1}{2l}$ in agreement with (2.7), where we expect that $b = al$. The other transition probabilities may be derived by following similar logic. Note that owing to the synchronization between the UEC trellis and the unary codewords, the trellis path $\mathbf{m}$ is guaranteed to terminate in a particular state. More specifically, the final state is guaranteed to be $m_b = 1$, when the symbol vector $\mathbf{x}$ has an even length a, while $m_b = 2$ is guaranteed when a is odd. The joint probabilities of (2.7) may be converted into the conditional transition probabilities $\Pr(M_j = m|M_{j-1} = m') = P(m|m')$, according to

$$P(m|m') = \frac{P(m,m')}{\sum_{\check{m}=1}^{r} P(\check{m},m')}. \tag{2.8}$$

The trellis encoder represents each bit y_j in the vector $\mathbf{y}$ by an n-bit codeword $\mathbf{z}_j$. This codeword is selected from the set of $r/2$ codewords $\mathbb{C} = [\mathbf{c}_1; \mathbf{c}_2; \ldots; \mathbf{c}_{r/2-1}; \mathbf{c}_{r/2}]$ or from the complementary set $\overline{\mathbb{C}} = [\overline{\mathbf{c}_1}; \overline{\mathbf{c}_2}; \ldots; \overline{\mathbf{c}_{r/2-1}}; \overline{\mathbf{c}_{r/2}}]$. As shown in Figure 2.4(a), this is achieved according to

$$\mathbf{z}_j = \begin{cases} \overline{\mathbf{c}_{\lceil m_{j-1}/2 \rceil}} & \text{if } y_j \neq \text{odd}(m_{j-1}) \\ \mathbf{c}_{\lceil m_{j-1}/2 \rceil} & \text{if } y_j = \text{odd}(m_{j-1}) \end{cases}. \tag{2.9}$$

Following this, the selected codewords are concatenated to obtain the bn-bit vector $\mathbf{z} = [z_k]_{k=1}^{bn}$ of Figure 2.2. For example, the vector $\mathbf{y} = 011001000111011$ of $b = 15$ bits is represented by the vector $\mathbf{z} = 110111111001000101100011110111$ of $bn = 30$

bits, when employing the $r = 4$-state UEC trellis of Figure 2.2, with the $n = 2$-bit codebook $\mathbb{C} = [00; 01]$.

The bit vector $\mathbf{z}$ may be modeled as a specific realization of a vector $\mathbf{Z} = [Z_k]_{k=1}^{bn}$ comprising bn binary RVs. Furthermore, the UEC trellis of Figure 2.4(a) has been designed to obey symmetry and to rely on complementary codewords, so that it produces equiprobable bits[2]satisfying that $\Pr(Z_k = 0) = \Pr(Z_k = 1)$ and the bit entropy is $H_Z = 1$. The average length of the bit vector $\mathbf{z}$ is $a \cdot l \cdot n$ and the average coding rate of the UEC encoder is given by

$$R^o = \frac{H_D}{ln} = \frac{1}{ln} \sum_{d \in \mathbb{N}_1} H[P(d)]. \tag{2.10}$$

Here, we employ the roman superscript 'o' to indicate that this coding rate relates to the outer encoder of a serial concatenation, namely the UEC encoder shown in Figure 2.2. In Figure 2.6 we use (2.3) and (2.5) to plot $R^o n = \frac{H_D}{l_{Unary}}$ for the case of using the UEC encoder to represent the symbols obeying the Zeta distribution of (2.2), as a function of p_1. Here, the coding rate R is zero for $p_1 \leq 0.608$, since the average unary codeword length l becomes infinite in this case, as discussed in Section 2.3.1.

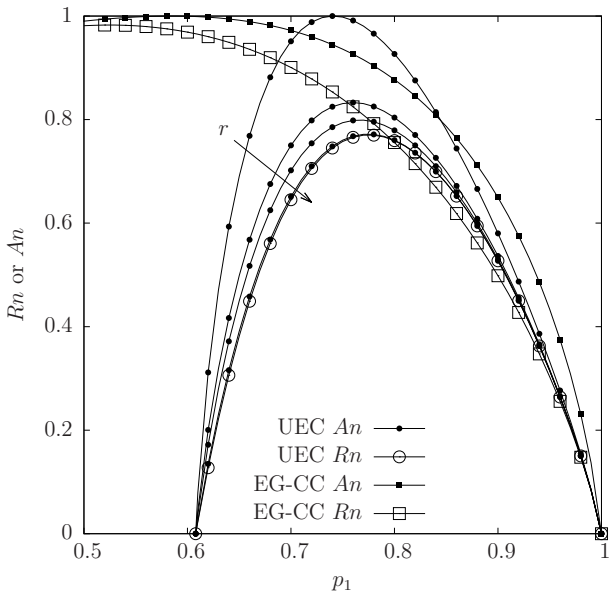


Figure 2.6: Plots of $R^o n$ and $A^o n$ that are obtained for the UEC and EG-CC schemes, in the case where the symbol values obey a Zeta distribution having the parameter p_1. The value of $A^o n$ is provided for UEC codes having various numbers of states $r \in \{2, 4, 6, 30\}$ [1].

[2]Note that owing to the edge effect, the binary RVs near either end of the vector $\mathbf{Z}$ do not adopt equiprobable values in general, like those in the middle of the vector. In practice however, this is only apparent for RVs that are within a few positions from the ends of the vector $\mathbf{Z}$. As a result, the edge effect is negligible for practical values of the bit vector length bn and will be disregarded throughout the remainder of this thesis.

2.3.3 Integration of the UEC encoder into a transmitter

As shown in Figure 2.2, the UEC encoder may be integrated into a transmitter by concatenating it with a URC encoder and a QPSK modulator, where these components are separated by interleavers. These transmitter components will be discussed in the following subsections.

2.3.3.1 Interleaver operation

Following UEC encoding, the bit vector $\mathbf{z}$ may be interleaved in the block π_1 shown in Figure 2.2. Since the UEC scheme employs iterative decoding, interleaving is necessary to mitigate the correlation within the soft information exchanged between the inner and outer decoder, as it will be detailed in Section 2.5. In the UEC scheme, we employ pseudo-random interleaver designs. Note that the receiver is required to employ the same pseudo-random interleaver designs as the transmitter. However, the entire set of interleavers can be generated independently by both the transmitter and receiver using only a single pseudo-random number generator seed. This seed may be hard-coded into both the transmitter and receiver, or may be reliably conveyed using only a very small amount of side information.

In our example, the UEC trellis encoded vector $\mathbf{z} = 11011111100100010110\ 0011110111$ of $bn = 30$ bits may be provided to the interleaver design of Figure 2.7, which is parametrized according to $\pi_1 = [6, 22, 3, 2, 26, 9, 13, 21, 18, 20, 12, 27, 29, 8, 5, 17, 7, 11, 19, 30, 14, 15, 1, 4, 24, 25, 28, 23, 16, 10]$. The interleaver π_1 outputs the bit vector

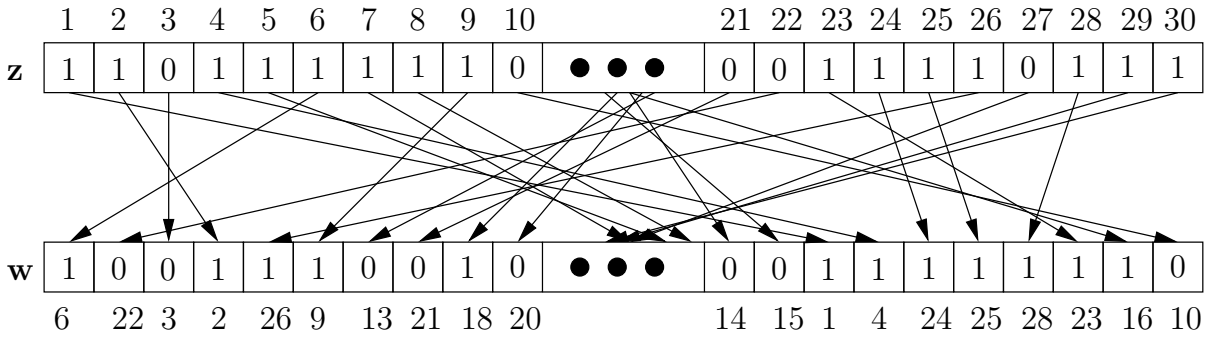


Figure 2.7: An example pseudo-random interleaver design having the parametrization $\pi_1 = [6, 22, 3, 2, 26, 9, 13, 21, 18, 20, 12, 27, 29, 8, 5, 17, 7, 11, 19, 30, 14, 15, 1, 4, 24, 25, 28, 23, 16, 10]$. Here the $\mathbf{z}$ and $\mathbf{w}$ are the corresponding bit vectors shown in Figure 2.2.

$\mathbf{W} = [W_k]_{k=1}^{bn}$, which is generated according to the interleaving rule $w_j = z_{\pi(j)}$, where $w_1 = z_6$ since $\pi(1) = 6$, for example. In our example, the interleaved output bit vector is $\mathbf{w} = 10011100101011101011001111111 0$.

2.3.3.2 URC code

As shown in Figure 2.2, the bit vector $\mathbf{w}$ is encoded by a URC encoder [87]. A URC code is a special case of a Convolutional Code (CC) that has a coding rate of 1. A URC encoder comprises a linear feedback shift register, having m register stages, as well as feed-forward and feed-back polynomials, as shown in Figure 2.8. Each URC parametrization has a corresponding trellis representation, comprising $r = 2^m$ states and $2r$ transitions, as shown in Figure 2.8.

Each bit w_k in the vector $\mathbf{w}$ is encoded by the URC encoder of Figure 2.2, in order of increasing bit-index k. Commencing from an initial state of $m_0 = 1$, each bit forces the trellis encoder to traverse from its particular previous state $m_{k-1} \in \{1, 2, \ldots, r\}$ to a new state $m_k \in \{1, 2, \ldots, r\}$ that is selected from two legitimate alternatives, depending on the bit value w_k, according to the trellis diagrams of Figure 2.8. The selected transition identifies a value for the corresponding encoded bit u_k, which contributes to the bit vector $\mathbf{U} = [U_k]_{k=1}^{bn}$, as shown in Figure 2.8. When employing the $r = 2$-state URC parametrization of Figure 2.8(a), the bit vector $\mathbf{w} = 10011100101011101011001111111 0$ of $bn = 30$ bits is URC encoded into $\mathbf{u} = 111010001100101100100010101011$. Figure 2.9 shows the path through the trellis of Figure 2.8(a) that is selected by the first six bits in the example bit sequence $\mathbf{w}$ above.

Note that in an Irregular Unity Rate Code (IrURC), different parametrizations are used for encoding different fractions of the bit vector $\mathbf{w}$. The careful selection of these parametrizations provide us with a substantial design freedom, which the URC may be exploit for improving the performance of the system [95]. However the parametrization of an IrURC must be tailored to a specific source distribution, hence preventing its application in cases, where the source distribution is unknown or non-stationary.

2.3.3.3 Interleaving and puncturing

The UEC scheme of Figure 2.2 employs another interleaver π_2 after the URC encoder, having a different pseudo-random design from that of π_1. However, for the sake of simplicity, we illustrate the operation of π_2 using the same interleaver design as in Figure 2.7. More specifically, when provided with the bit vector $\mathbf{u} = 111010001100101100100010101011$ of $bn = 30$ bits, the interleaver π_2 of Figure 2.2 outputs the interleaved bit vector $\mathbf{g} = 001101100001101000110110010111$ according to the interleaver rule of $g_k = u_{\pi_2(k)}$. Note that in addition to the interleaving operation, a puncturing operation may also be performed in π_2, in order to control the overall coding rate of the UEC scheme of Figure 2.2. This is achieved by discarding

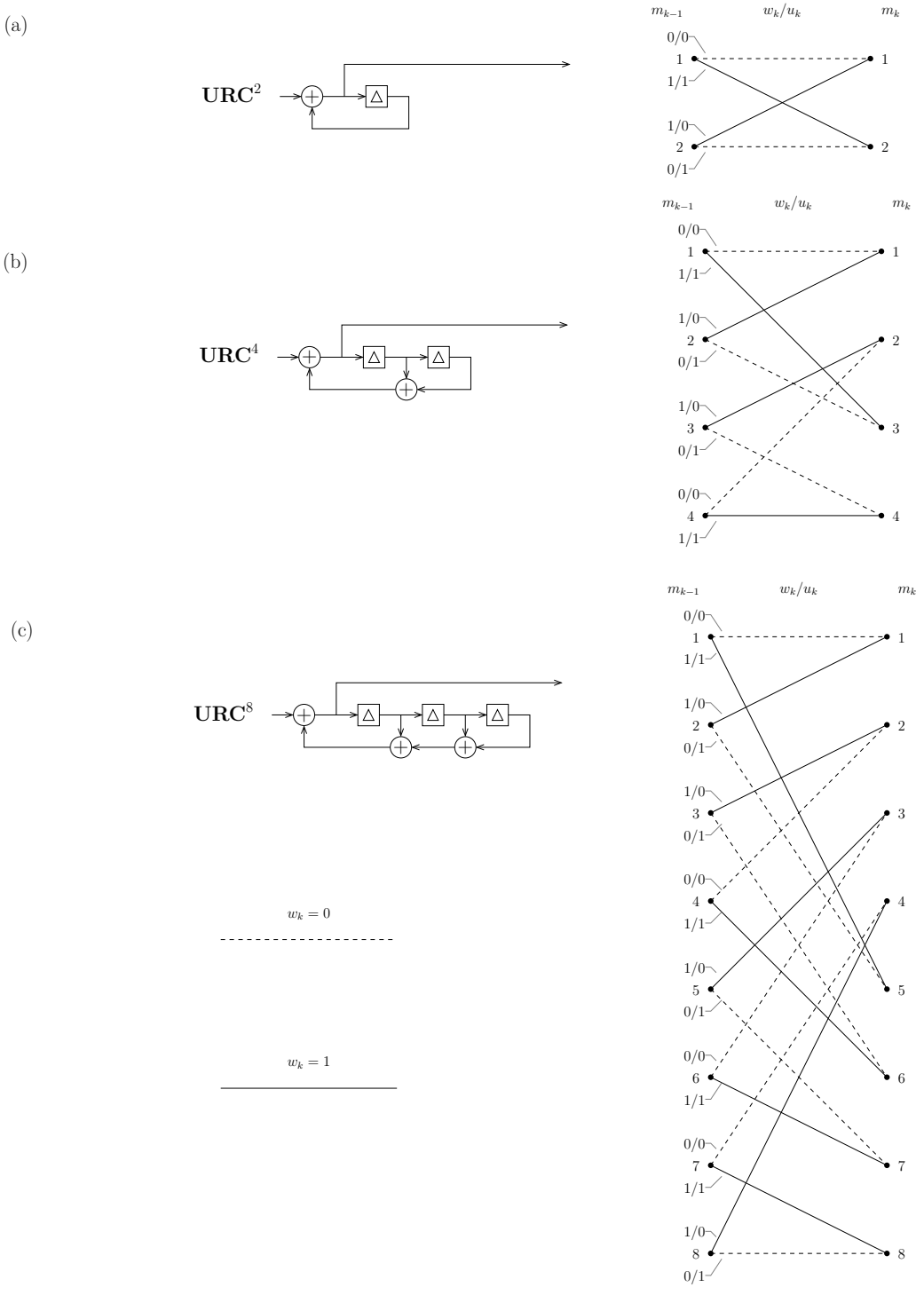


Figure 2.8: Linear feedback shift register encoder schematic for the recursive URC codec with the corresponding trellis representation, where (a) $m = 1$ and $r = 2$, (a) $m = 2$ and $r = 4$, (a) $m = 3$ and $r = 8$.

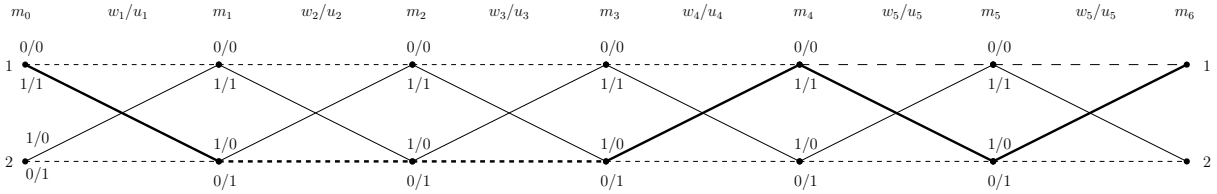


Figure 2.9: The trellis of the URC shown in Figure 2.4(a). Here the dashed transitions correspond to an input bit of $w_k = 0$, while the solid transitions correspond to an input bit of $w_k = 1$. In the label of each transition, the bit before the slash is the input bit w_k, while the bit after the slash is the output bit u_k. The transitions highlighted in bold indicate the path taken when encoding the input bit vector $\mathbf{w} = [100111]$.

a fraction of the bn bits from the end of the bit vector $\mathbf{g}$. This fraction $(1 - \frac{1}{R^i})$ may be chosen in order to give the desired inner coding rate R^i, which quantifies the ratio of the number of bits in $\mathbf{w}$ to the number of bits in $\mathbf{g}$.

2.3.3.4 QPSK modulation

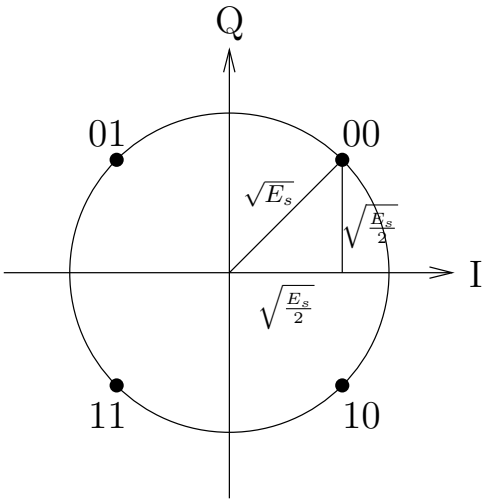
Following the URC encoder and the interleaver π_2 of Figure 2.2, $M = 4$-ary Gray-coded QPSK modulation may be employed for transmission, as shown in Figure 2.2. The classic Gray Mapping (GM) is employed here, since the performance of Gray-coded QPSK is superior to that of QPSK using Anti-Gray Mapping (AGM), when no iterations are performed between the demapper and URC decoder in the receiver [96]. The bits of $\mathbf{g}$ are processed $\log_2(\mathrm{M}) = 2$ bits at a time, as shown in Figure 2.10. According to the Gray-coded QPSK mapping rule of Figure 2.10, each pair of bits identifies one of the $M = 4$ constellation points, which modulate the amplitude of the In-phase (I) and Quadrature-phase (Q) carries. Here E_s is the energy per modulated constellation point. Without loss of generality, we assume that $E_s = 1$ throughout this treatise.

For example, the vector $\mathbf{g} = 001101100001101000110110010111$ of $bn = 30$ bits corresponds to the vector of $\mathbf{e} = [0.7071 + 0.7071i, -0.7071 - 0.7071i, -0.7071 + 0.7071i, 0.7071 - 0.7071i, 0.7071 + 0.7071i, -0.7071 + 0.7071i, 0.7071 - 0.7071i, 0.7071 - 0.7071i, 0.7071 + 0.7071i, -0.7071 - 0.7071i, -0.7071 + 0.7071i, 0.7071 - 0.7071i, -0.7071 + 0.7071i, -0.7071 + 0.7071i, -0.7071 - 0.7071i]$ of $\frac{bn}{\log_2(\mathrm{M})} = 15$ constellation points, which are represented using complex numbers where the real and imaginary parts correspond to I and Q, respectively.

The effective throughput of the transmitter shown in Figure 2.2 is given by

$$\eta = R^o \cdot R^i \cdot \log_2(M), \tag{2.11}$$

which quantifies the number of bits in the source symbol vector $\mathbf{d}$ that are transmitted



(a)

pair of bits from **g**	I	Q
00	$+\sqrt{\frac{E_s}{2}}$	$+\sqrt{\frac{E_s}{2}}$
10	$+\sqrt{\frac{E_s}{2}}$	$-\sqrt{\frac{E_s}{2}}$
11	$-\sqrt{\frac{E_s}{2}}$	$-\sqrt{\frac{E_s}{2}}$
01	$-\sqrt{\frac{E_s}{2}}$	$+\sqrt{\frac{E_s}{2}}$

(b)

Figure 2.10: (a) Constellation diagram for QPSK modulation with Gray mapping [97] (b) Mapping rule for QPSK with Gray mapping

by each constellation point. The transmitted energy per bit of information in the source symbol vector $\mathbf{d}$ is $E_b = E_s/\eta$.

Note that other mapping schemes or a modulation scheme having a higher M may be employed instead, although this may increase the complexity of the receiver. In [98], AGM QPSK is employed and the corresponding demodulator is serially concatenated for the sake of iteratively exchanging extrinsic information with a UEC-turbo scheme [99], since classic GM QPSK is unable to provide any iteration gain [96]. Reliable communication is theoretically possible, provided that the effective throughput η does not exceed the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity [100] of the modulation scheme and channel. However, the DCMC capacity depends only on the distance between the constellation points of the modulation scheme and it is independent of the specific bit-to-symbol mapping method [97]. Owning to this, both GM and AGM facilitate near-capacity operation. Furthermore, GM has the advantage that its demodulation complexity may be substantially reduced by exploiting its constellation symmetry [101]. More specifically, the first bit in each pair from $\mathbf{g}$ depends only on the Q component of a received constellation point, while the second bit depends only on the I component, as shown in Figure 2.10.

2.4 Uncorrelated narrow-band Rayleigh fading channel

After the modulator of Figure 2.2, the symbol vector $\mathbf{e}$ is upsampled, band limited by smooth pulse shaping and up converted to the carrier frequency, amplified and transmitted through an uncorrelated narrow-band Rayleigh fading channel. In the receiver, the received symbol is amplified, mixed down from the carrier frequency, matched-filtered and down sampled in order to obtain the vector of received constellation points $\tilde{\mathbf{e}}$. These operations of the transmitter, the uncorrelated narrow-band Rayleigh fading channel and the receiver can be modelled by

$$\tilde{\mathbf{e}} = \mathbf{h} \cdot \mathbf{e} + \mathbf{o}. \tag{2.12}$$

where $\mathbf{h}$ is a vector of complex channel gains and the $\mathbf{o}$ is a vector of Additive White Gaussian Noise (AWGN) sampler. Note that both $\mathbf{h}$ and $\mathbf{o}$ are modeled using complex-valued Gaussian distributions. More specifically, both the real and imaginary parts of $\mathbf{h}$ have a mean of 0 and a variance of $1/2$, while the real and imaginary parts of $\mathbf{o}$ both have a mean of 0 and a variance of $N_0/2$, where N_0 is the Power Spectral Density (PSD) of the AWGN. In a communication scheme having an effective throughput of η, the channel quality is typically characterized by the SNR per bit of information in the source, according to E_b/N_0 [dB] $= E_s/N_0$ [dB] $- 10 \cdot \log_{10}(\eta)$, where the effective throughput η can be calculated using (2.11). For example, when $E_b/N_0 = 5.5$[dB], we may have a channel gain vector of $\mathbf{h} = [-0.7478 + 0.7380i, 0.2859+0.7960i, -0.7396+0.4949i, -0.8272-0.1781i, -0.3165-0.7541i, -0.7457+0.0939i, 0.2121-0.6739i, -0.4047+0.3735i, 0.9777-0.2317i, -0.3057-0.0450i, 0.6493+0.1448i, 0.0325+0.4178i, -0.9116-0.0850i, -0.7511-0.7348i, -0.0166-0.0801i]$ and an AWGN vector $\mathbf{o} = [-0.3913+0.0150i, -0.6857+0.7339i, 0.6027+0.2811i, 0.4036-0.0047i, -0.3163+0.2076i, 0.3451-0.8253i, -0.5091+0.7529i, -0.9656-0.2282i, -0.2450-0.6508i, -0.1054+0.3645i, 0.7284-0.0507i, -0.4161+0.3724i, -0.1376-0.4943i, 0.3954+0.7921i, 0.3996 - 0.3983i]$. By using (2.12), we have $\tilde{\mathbf{e}} = [-0.1758 - 0.1831i, 1.0821 - 1.2587i, 0.8379-1.0520i, -0.8181+0.2400i, 0.6493-0.6443i, -0.3568-0.9577i, -0.4769-0.8638i, 0.0867+0.5710i, 0.2536+1.1124i, -0.4408+0.6165i, -0.2676+0.3788i, 0.5131-0.1331i, -0.4568 - 0.3829i, 1.0941 + 0.2831i, -0.5488 - 0.0143i]$

2.5 UEC decoder

In this section, we describe the operation of the UEC decoder of Figure 2.2. More specifically, Section 2.5.1 discusses the integration of the UEC decoder with the URC decoder and soft QPSK demodulator of Figure 2.2. Following this, we detail the operation of the UEC trellis decoder and the unary decoder in Section 2.5.2.

2.5.1 Integration into a receiver

In the receiver of Figure 2.2, soft QPSK demodulation [97], depuncturing and deinterleaving π_2^{-1}, Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR)-based URC decoding [9] and further deinterleaving π_1^{-1} may be performed, before invoking the proposed UEC decoder. These steps are discussed in the following subsections.

2.5.1.1 Soft QPSK demodulation

Soft QPSK demodulation is employed in the scheme of Figure 2.2 and the soft information is passed to the URC decoder in the form of Logarithmic Likelihood Ratios (LLRs). We will introduce the concept of LLRs in Section 2.5.1.1.1 and discuss the operation of soft demodulation in Section 2.5.1.1.2.

2.5.1.1.1 Log Likelihood Ratio

In communication systems employing iterative decoding, the LLR [102] is widely used for exchanging soft information between the concatenated decoders. As shown in Figure 2.2 the receiver of our UEC scheme operates on the basis of LLR vectors, rather than bit vectors. Each LLR conveys not only what the most likely value of the corresponding bit in the transmitter is, but also how likely this value is. More specifically, an LLR $\tilde{x}$ pertaining to a bit x in the transmitter can be defined as

$$\tilde{x} = \ln\left(\frac{P(x=1)}{P(x=0)}\right) = \ln\left(\frac{P(x=1)}{1-P(x=1)}\right). \tag{2.13}$$

Figure 2.11 shows how the value of the LLR $\tilde{x}$ varies as a function of the probability $P(x=1)$. The sign of the LLR $\tilde{x}$ expresses what the most likely value of the bit x is, where a positive value of $\tilde{x}$ indicates that $x=1$ is most likely. The magnitude $|\tilde{x}|$ quantifies how likely this value is, where a higher magnitude represents a higher confidence. When the LLR has the value of $x=0$, we have $P(x=0)=P(x=1)=0.5$, which means that the LLR expresses no confidence either way about the value of the bit x. For example, $\tilde{x}=-0.1758$ indicates that the probability of the bit x having the value 0 is 54%, while this probability is increased to 98% when the LLR has the value $\tilde{x}=-4.18$.

2.5.1.1.2 Soft Demodulator

In the soft demodulator of Figure 2.2, two LLRs can be derived for the vector $\tilde{\mathbf{g}}$ from each received constellation point in the vector $\tilde{\mathbf{e}}$ according to [97] :

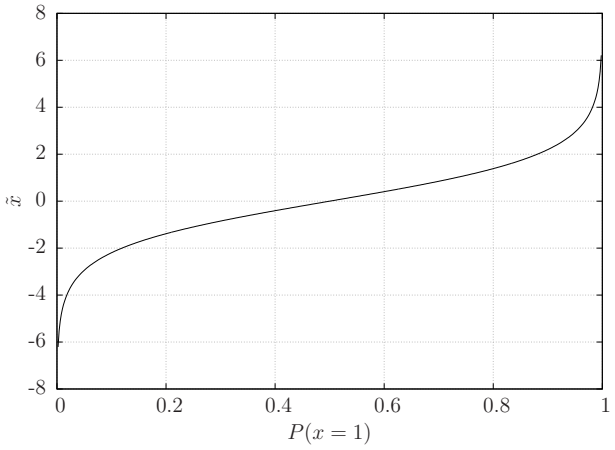


Figure 2.11: The LLR $\tilde{x}$ versus the probability $P(x=1)$

$$\begin{aligned} \tilde{g}_{2j-1} &= -\frac{4|h_j|^2}{N_0} \cdot \mathrm{Im}\left(\frac{\tilde{e}_j}{h_j}\right) \\ \tilde{g}_{2j} &= -\frac{4|h_j|^2}{N_0} \cdot \mathrm{Re}\left(\frac{\tilde{e}_j}{h_j}\right), \end{aligned} \tag{2.14}$$

where $\tilde{e}_j$ is the j^{th} constellation point in the vector $\tilde{\mathbf{e}}$, h_j is the corresponding channel gain in $\mathbf{h}$ and N_0 is the PSD of the AWGN. Here, it is assumed that the receiver has perfect knowledge of both the channel gain vector $\mathbf{h}$ and of the PSD N_0. In the case of the received vector $\tilde{\mathbf{e}}$ of Section 2.4, we have $\tilde{\mathbf{g}} = [-2.0302, 0.0278, 9.2971, 5.2724, -2.7662, 8.6817, 2.6201, -4.8266, -5.2804, -2.1344, -5.6918, -1.3403, 3.8418, -3.6613, 2.0059, -1.3563, -8.7267, 0.0749, 1.5856, -0.8144, -2.1675, 0.9050, 1.6650, 0.2962, -2.3619, -3.4178, -4.5016, 7.8396, 0.3327, -0.0780]$

2.5.1.2 Depuncturing and deinterleaving

Depuncturing may be achieved by replacing any punctured bits in $\mathbf{g}$ with zero-valued LLRs in $\tilde{\mathbf{g}}$. Deinterleaving may be achieved accoding to the rule of $\tilde{u}_{\pi_2(k)} = \tilde{g}_k$. Note that in order to achieve this, the receiver is required to employ the same pseudo-random interleaver design π_2 as the transmitter. However, the entire set of interleavers can be generated independently by both the transmitter and receiver using only a single pseudo-random number generator seed. This seed may be hard-coded into both the transmitter and receiver, or may be reliably conveyed using only a very small amount of side information. After the deinterleaver π_2^{-1} of Figure 2.2, we have the vector $\tilde{\mathbf{u}} = [1.6650, 5.2724, 9.2971, 0.2962, 2.0059, -2.0302, -8.7267, -3.6613, 8.6817, -0.0780, 0.0749, -5.691, 8, 2.6201, -2.1675, 0.9050, 0.3327, -1.3563, -5.2804, 1.5856, -2.1344, -4.8266, 0.0278, 7.8396, -2.3619, -3.4178, -2.7662, -1.3403, -4.5016, 3.8418, -0.81$ which comprises $bn = 30$ LLRs.

2.5.1.3 Log-BCJR-based URC decoding

The URC decoder may invoke the Log-BCJR algorithm for converting the vectors of $b{\cdot}n$ *a priori* LLRs $\tilde{\mathbf{w}}^a = [\tilde{w}_k^a]_{k=1}^{bn}$ and $\tilde{\mathbf{u}} = [u_k]_{k=1}^{bn}$ into the vector of $b{\cdot}n$ extrinsic LLRs $\tilde{\mathbf{w}}^e = [\tilde{w}_k^e]_{k=1}^{bn}$. A step by step demonstration of the Log-BCJR algorithm decoding process is provided in Section 2.5.1.3.1. Following this, Section 2.5.1.3.2 introduces the fundamental Add, Compare and Select (ACS) operations that form the basis of the Log-BCJR algorithms.

2.5.1.3.1 Log-BCJR algorithm

In order to present the Log-BCJR algorithm step by step, we consider the inner code URC decoder of Figure 2.2, when employing the $r = 2$-state trellis of Figure 2.8(a). As shown in Figure 2.2, the URC decoder converts the input *a priori* LLR vectors $\tilde{\mathbf{w}}^a = [\tilde{w}_k^a]_{k=1}^{bn}$ and $\tilde{\mathbf{u}} = [u_k]_{k=1}^{bn}$into the output *extrisic* LLR vectors $\tilde{\mathbf{w}}^e = [\tilde{w}_k^e]_{k=1}^{bn}$. The Log-BCJR algorithm calculates each *extrisic* LLR $\tilde{w}_k^e$ in the vector $\tilde{w}^e$ by calculating for intermediate terms, $\alpha_{k-1}(m')$ $\beta_k(m)$ and $\gamma_k(m', m)$, as shown in Figure 2.12, while the fourth intermediate term $\delta_k(m', m)$ is obtained by combining the results of the other three terms.

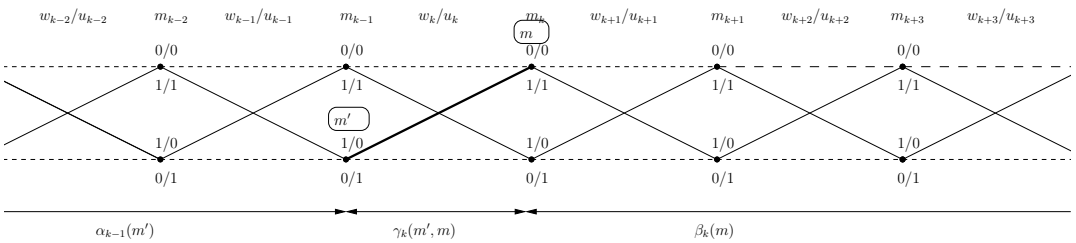


Figure 2.12: The decoder trellis of the URC with $r = 2$ [97]. In this section, we will use the transition highlighted in bold from m' to m as an example to demonstrate the calculation of each term in Log-BCJR algorithm.

The Log-BCJR algorithm begins its operation by calculating the first term $\gamma_k(m', m)$ for each transition in the trellis, from the previous state m' to the new state m. The term $\gamma_k(m', m)$ is calculated according to

$$\gamma_k(m, m') = \tilde{w}_k^{\mathrm{a}} \cdot w(m, m') + \tilde{u}_k \cdot u(m, m') + \ln[P(m|m')], \tag{2.15}$$

where $w(m, m')$ and $u(m, m')$ are the corresponding bit values that are tentatively assumed for $\mathbf{w}$ and $\mathbf{u}$, when the trellis traverses from state m' to state m. Furthermore, $P(m|m')$ is the conditional transition probability of the trellis transition from state m' to state m. Note that for the URC decoder, all trellis transitions have the same conditional probability of $P(m|m') = 0.5$. So (2.15) can be further simplified to $\gamma_k(m, m') = \tilde{w}_k^{\mathrm{a}} \cdot w(m, m') + \tilde{u}_k \cdot u(m, m')$ for the case of the URC decoder. Note that

when either $w(m, m')$ or $u(m, m')$ adopt the value of o^k, the addition in the simplified calculation of $\gamma_k(m, m')$ is not required. Since only one of the four transitions seen in Figure 2.8(a) have both $w(m, m') = 1$ and $u(m, m') = 1$, only a single addition is required for computing $\gamma_k(m, m')$ for the set of four transitions corresponding to each bit in the vector $\mathbf{w}$.

The second step of the Log-BCJR algorithm is constituted by the calculation of the $\alpha_k(m)$ term for each state m in the trellis, according to

$$\alpha_k(m) = \max_{m' \to m}{}^*[(\alpha(m') + \gamma_k(m, m')], \tag{2.16}$$

where the notation of $m' \to m$ represents the set of all previous states m' that have a transition terminating in the state m. We calculate the additions in the logarithmic domain using the max* operator [103] as follows,

$$\ln(a + b) = \ln(e^A + e^B) = \max(A, B) + \ln(1 + e^{-|A-B|}) = \max^*(A, B), \tag{2.17}$$

where we have $A = \ln(a)$ and $B = \ln(b)$. Here, the max* operation is defined for a pair of operands in equation 2.17, but it may be readily extended to more than two operands by exploiting its associative property. Note that the calculation of the $\alpha_k(m)$ terms in equation 2.16 must be completed in a forward recursive manner, owing to the dependencies upon the $\alpha_{k-1}(m')$ terms of the previous states. This forward recursion starts from $k = 1$ using $\alpha_0(1) = 0$ and $\alpha_0(2) = -\infty$, since the encoding process always starts from state $m_0 = 1$, as described in Section 2.3.2. Noting that one of the four transitions in Figure 2.8(a) is guaranteed to have $\gamma_k(m, m') = 0$ and the corresponding addition in (2.16) is not required, a total of three additions and two max* operations are required per bit of w, in order to complete the second step of the Log-BCJR algorithm.

The third step of the Log-BCJR algorithm is to calculate the $\beta_{k-1}(m')$ term for each state m' in the trellis, according to

$$\beta_{k-1}(m') = \max_{m' \leftarrow m}{}^*[\beta_k(m) + \gamma_k(m', m)], \tag{2.18}$$

where $m' \leftarrow m$ is the set of all states m that have a transition connecting to the previous state $m'$of Figure 2.12. In contrast to the $\alpha_k(m)$ terms, the calculation of $\beta_{k-1}(m')$ depends upon the $\beta_k(m)$ terms of the next states, hence it must be completed in a backward recursive manner. This backward recursion commences from $k = b \cdot n$ using $\beta_{bn}(1) = 0$ and $\beta_{bn}(2) = 0$, since the final state is unknown during URC encoding. As in the calculation of the $\alpha_k(m)$ terms, a total of three

additions and two max* operations are required per bit of w, in order to complete the third step of the Log-BCJR algorithm on the URC trellis of Figure 2.8(a).

The fourth step of the Log-BCJR algorithm is to combine all the previous steps and calculate the $\delta_k(m', m)$ term for each transition in the trellis, according to

$$\delta_k(m, m') = \alpha_{k-1}(m') + \gamma_k(m, m') + \beta_k(m). \tag{2.19}$$

Noting that one of the four transitions in Figure 2.8(a) is guaranteed to have $\gamma_k(m, m') = 0$ and the corresponding addition in (2.19) is not required, a total of seven additions are required per bit of w, in order to complete the fourth step of the Log-BCJR algorithm.

Finally, the extrinsic LLR $\tilde{w}_k^{\mathrm{e}}$ can be obtained based on $\delta_j(m, m')$ according to

$$\tilde{w}_k^{\mathrm{e}} = \max^*_{m,m'|w(m,m')=1}[\delta_k(m, m')] - \max^*_{m,m'|w(m,m')=0}[\delta_k(m, m')] - \tilde{w}_k^{\mathrm{a}}, \tag{2.20}$$

where $m, m'|w(m, m') = 1$ and $m, m'|w(m, m') = 0$ are the sets of transitions having $w(m, m') = 1$ and $w(m, m') = 0$, respectively. A total of two subtractions and two max* operations are required per bit of w, in order to complete the final step of the Log-BCJR algorithm applied to the URC trellis of Figure 2.8(a). When using the example of the deinterleavered LLR vector $\tilde{\mathbf{u}}$ used in Section 2.5.1.2 and the *a priori* LLR vector $\tilde{\mathbf{w}}^a$ comprising only zero-valued LLRs, we have $\tilde{\mathbf{w}}^e = [1.6650, -1.6392, -5.2547, -0.2962, -0.2253, 1.3423, -2.0290, -3.6550, 3.6547, 0.0780, 0.0029, 0.0744, 2.5750, 1.6835, 0.7011, -0.1400, 0.1952, -1.3381, 1.5621, 1.1536, -2.0698, 0.0273, -0.0277, 2.3578, -2.0665, -2.3488, -1.1413, -1.3017, 3.4253, 0.7766]$.

In Section 2.3.3.2, we recommend the employment of URC Linear Feedback Shift Registers (LFSRs) having generator polynomials of the form $[1, 0, \ldots, 0]$ and feedback polynomials of the form $[1, 1 \ldots, 1]$. This is because this kind of URC outputs the non-zero *extrisic* LLR vectors $\tilde{\mathbf{w}}^e$ [9], when provided with an all-zero *a priori* LLR vector $\tilde{\mathbf{w}}^a$, as shown in Figure 2.13(b). By contrast, the *extrisic* LLR vectors $\tilde{\mathbf{w}}^e$ of the URC LFSRs having generator and feedback polynomials of other formats will have all-zero values, as shown in Figure 2.13(c). Furthermore, the URCs having LFSRs of the recommended format produce extrinsic LLRs $\tilde{\mathbf{w}}^{\mathrm{e}}$ having very high magnitudes, when provided the with *a priori* LLR vector $\tilde{\mathbf{w}}^a$ having very high magnitudes. Owing to these properties, the URCs having the recommended form of polynomials reduce the number of erroreous bit-decisions during iterative decoding. These features are important for the inner code of our UEC iterative decoding process, as it will be discussed in Section 2.5.2.2.

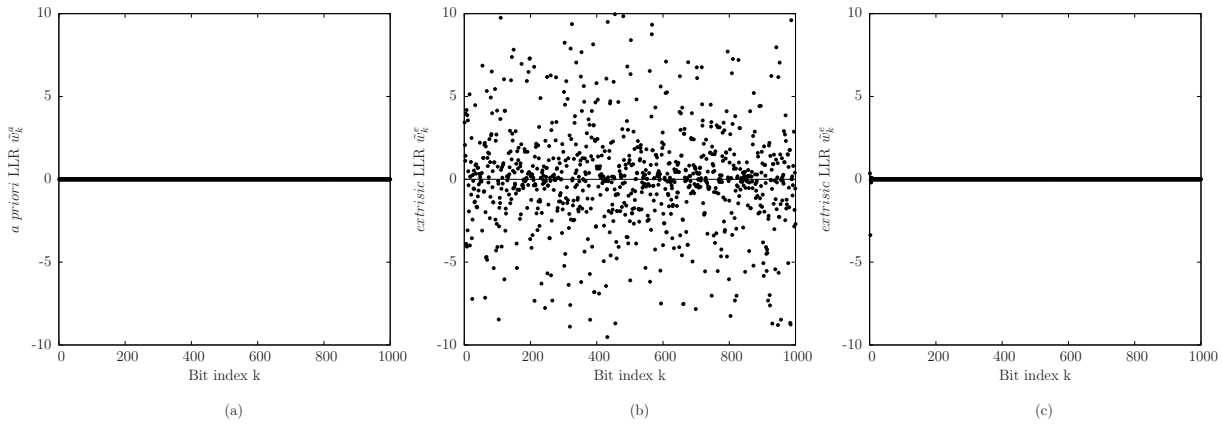


Figure 2.13: (a) All-zero *a prior* LLRs $\tilde{\mathbf{w}}^a$. (b) Corresponding non-zero extrinsic LLRs $\tilde{\mathbf{w}}^e$ produced by the URC of Figure 2.8(a) with input of (a) and mutual information $I(\tilde{\mathbf{w}}^e, \mathbf{w}) = 0.4081$. (c) Corresponding non-zero extrinsic LLRs $\tilde{\mathbf{w}}^e$ produced by the URC^3 in [9, page 354]. In this example, the URC scheme of Figure 2.2 transmits frames comprising $a = 2 \cdot 10^4$ symbols that obey the Zeta distibution with $p_1 = 0.7967$ using QPSK modulation over an uncorrelated narrowband Rayleigh fading channel having $E_\mathrm{b}/N_0 = 5.5$ dB.

Note that the UEC trellis decoder will also use the Log-BCJR algorithm, as it will be discussed in Section 2.5.2.1.

2.5.1.3.2 ACS operation

As described in Section 2.5.1.3.1, the Log-BCJR algorithm operates entirely on the basis of addition, subtraction and max* operations. In order to characterise the computational complexity of iterative decoding, each step of the Log-BCJR algorithm may be further decomposed into the fundamental ACS operations. As in [95], we assume that the addition and subtraction operations performed by the Log-BCJR each require a single ACS operation, while each max* operation may be approximated by a look up table operation, which can be completed using five ACS operations [103]. As shown in Table 2.3, the complexity of the URC decoder of Figure 2.8 increases, as the number of states r is increased. Note that we assume that all components in Figure 2.2 that do not operate on the basis of the Log-BCJR algorithm may be considered to have a relatively insignificant complexity [95, 68]. Table 2.3 also characterises the complexity of the LTE turbo code [104], which includes the complexity of both of its parallel-concatenated BCJR decoders. Note that when the UEC scheme employs a UEC trellis having $r = 10$ states and a URC trellis having $r = 8$ states, its complexity in each iteration is lower than that of the LTE turbo decoder. Furthermore, when the UEC scheme employs a UEC trellis having $r = 4$ states and and a URC trellis having $r = 2$ states, the complexity of the UEC scheme per iteration is roughly a quarter of that of the LTE turbo decoder.

Table 2.3: Number of max* and addition operations that are performed per bit of $\mathbf{z}$ for UEC, URC and turbo decoder employing trellis having r states.

Decoder	r	max*	add	ACS
UEC BCJR decoder	4	8	20	60
	6	12	29	89
	8	16	38	118
	10	20	47	147
URC BCJR decoder	2	6	16	46
	4	14	28	98
	8	30	52	202
LTE turbo decoder	8	60	106	406

2.5.2 UEC trellis decoder

The UEC trellis decoder is invoked after completing QPSK demodulation, depuncturing and deinterleaving π_2^{-1}, URC decoding and deinterleaving π_1^{-1}, as shown in Figure 2.2. Here, the deinterleaver π_1^{-1} operates according to the rule $\tilde{z}^a_{\pi_1(k)} = \tilde{w}^e_k$. During iterative decoding, the interleaver π_1 operates according to $\tilde{w}^a_k = \tilde{z}^e_{\pi_1(k)}$. In this section, we will discuss the UEC trellis decoder, its EXtrinsic Information Transfer (EXIT) chart properties and its iterative decoding operation.

2.5.2.1 Log-BCJR-based trellis decoding

As shown in Figure 2.2, the UEC trellis decoder is provided with a vector of *a priori* LLRs $\tilde{\mathbf{z}}^{\mathrm{a}} = [\tilde{z}^{\mathrm{a}}_k]^{bn}_{k=1}$ that pertain to the corresponding bits in the vector $\mathbf{z}$. The trellis decoder applies the Log-BCJR algorithm [41] to a UEC trellis of the sort shown in Figure 2.4(a) in order to consider every legitimate realization of $\mathbf{Z}$ having the particular length $b \cdot n$. Note that the values of a and b are assumed to be perfectly known to the receiver. In practice, the transmitter may employ a small amount of side information to reliably convey these values. Here, the Log-BCJR algorithm operates as described in Section 2.5.1.3.1, but uses (2.21) to (2.26) , rather than (2.15) to (2.20):

$$\gamma_j(m, m') = \sum_{\ddot{n}=1}^{n} \tilde{z}^{\mathrm{a}}_k \cdot z_{\ddot{n}}(m, m') + \ln[P(m|m')], \tag{2.21}$$

$$\alpha_j(m) = \max_{m' \to m}{}^{*}[\alpha_{j-1}(m') + \gamma_j(m, m')], \tag{2.22}$$

$$\beta_{j-1}(m') = \max_{m' \leftarrow m}{}^{*}[\beta_j(m) + \gamma_j(m', m)], \tag{2.23}$$

$$\delta_j(m, m') = \alpha_{j-1}(m') + \gamma_j(m, m') + \beta_j(m). \tag{2.24}$$

$$\tilde{z}^{\mathrm{e}}_k = \max_{m,m'|z_{\ddot{n}}(m,m')=1}{}^{*}[\delta_j(m, m')] - \max_{m,m'|z_{\ddot{n}}(m,m')=0}{}^{*}[\delta_j(m, m')] - \tilde{z}^{\mathrm{a}}_k, \tag{2.25}$$

$$\tilde{y}^{\mathrm{p}}_j = \max_{m,m'|z_{\ddot{n}}(m,m')=1}{}^{*}[\delta_j(m, m')] - \max_{m,m'|z_{\ddot{n}}(m,m')=0}{}^{*}[\delta_j(m, m')]. \tag{2.26}$$

Here, $k = n(j-1) + \ddot{n}$, where $z_{\ddot{n}}(m, m')$ is the value of the $\ddot{n}^{th}$ bit in the UEC codeword $\mathbf{z}$, that is implied by the transition from m' to m. The synchronization between the UEC trellis and the unary codewords is exploited during the Log-BCJR algorithm's $\gamma_j(m, m')$ calculation of (2.24). More specifically, the conditional transition probabilities $P(m|m')$ of (2.8) may be substituted into (2.21). Note that the UEC trellis should be terminated at $m_0 = 1$ and at $m_b = 1 + \text{odd}(a)$, which depends on whether the length a of the symbol vector $\mathbf{d}$ is even or odd, respectively. More specifically $\alpha_0(1) = 0$ and $\beta_b(1 + \text{odd}(a)) = 0$, while all other $\alpha_0(m')$ and $\beta_b(m)$ adopt the value of $-\infty$. As shown in Figure 2.2, the Bahl-Cocke-Jelinek-Raviv (BCJR) decoder generates the vector of extrinsic LLRs $\tilde{\mathbf{z}}^{\text{e}} = [\tilde{z}_k^{\text{e}}]_{k=1}^{bn}$, as well as the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}^p = [\tilde{y}_j^p]_{j=1}^b$. In our example, the *a priori* LLRs $\tilde{\mathbf{z}}^a = [-0.0277, -0.2962, -5.2547, 2.3578, 0.7011, 1.6650, 0.1952, 1.6835, 1.3423, 0.7766, -1.3381, 0.0029, -2.0290, -2.0698, 0.0273, 3.4253, -0.1400, 3.6547, 1.5621, 0.0780, -3.6550, -1.6392, -1.3017, -2.0665, -2.3488, -0.2253, 0.0744, -1.1413, 2.5750, 1.1536]$ generate the *extrisic* LLR vector $\tilde{\mathbf{z}}^e = [2.4682, 2.7366, 0.6174, 0.2073, 1.2578, 0.5970, 0.7132, -0.4847, 0.5066, -1.3697, -1.505, 4, 0.5621, -0.4169, -0.4811, -2.0287, 0.7633, -2.5809, 0.7344, 1.3093, -2.0033, -0.9102, -1.0814, -0.8777, -0.5282, 2.3261, -0.0925, -\infty, 0.8141, +\infty, -1.1763]$, as well the *a posteriori* LLR vector $\tilde{\mathbf{y}}^p = [-2.4405, 2.5651, 1.8948, -1.1564, -0.4963, 1.9868, -2.0558, -2.7558, -2.3210, 2.6359, 2.8822, -2.1556, 0.3272, -0.0227, +\infty]$.

As shown in Figure 2.14 the UEC trellis requires non-zero *a priori* LLRs $\tilde{\mathbf{z}}^a$ in order to generate non-zero extrinsic LLRs $\tilde{\mathbf{z}}^{\text{e}}$. Owing to this, the URC recommended in Section 2.3.3.2 must be used, since it can provide a non-zero *a priori* LLR vector $\tilde{\mathbf{z}}^a$.

2.5.2.2 EXIT chart analysis

The transformation of $\tilde{\mathbf{z}}^{\text{a}}$ into $\tilde{\mathbf{z}}^{\text{e}}$ may be characterized by plotting the inverted UEC EXIT curve in an EXIT chart [51], as exemplifies in Figure 2.15. Here, the x axis of the EXIT chart quantifies the quality of the *extrisic* LLR vector $\tilde{\mathbf{z}}^{\text{e}}$, using its Mutual Information (MI), $I(\tilde{\mathbf{z}}^{\text{e}}; z) \in [0, 1]$, where an MI of 0 indicates the absence of any information in $\tilde{\mathbf{z}}^{\text{e}}$ about $\mathbf{z}$ corresponding to all-zero LLRs, while an MI of 1 indicates the knowledge of perfect information, corresponding to LLRs having the correct sign and very high magnitudes. The EXIT curve characterizes the MI of the *extrinsic* LLR vector $\tilde{\mathbf{z}}^{\text{e}}$ as a function of the MI $I(\tilde{\mathbf{z}}^a; \mathbf{z})$ of the *a priori* LLR vector $\tilde{\mathbf{z}}^a$, which is plotted on the y axis. Note that if codewords comprising at least $n = 2$ bits are employed, then the free distance d_{free} of the UEC code will be at least two, and having $d_{\text{free}} = 2$ is a sufficient condition for its EXIT curve to reach the $(1, 1)$

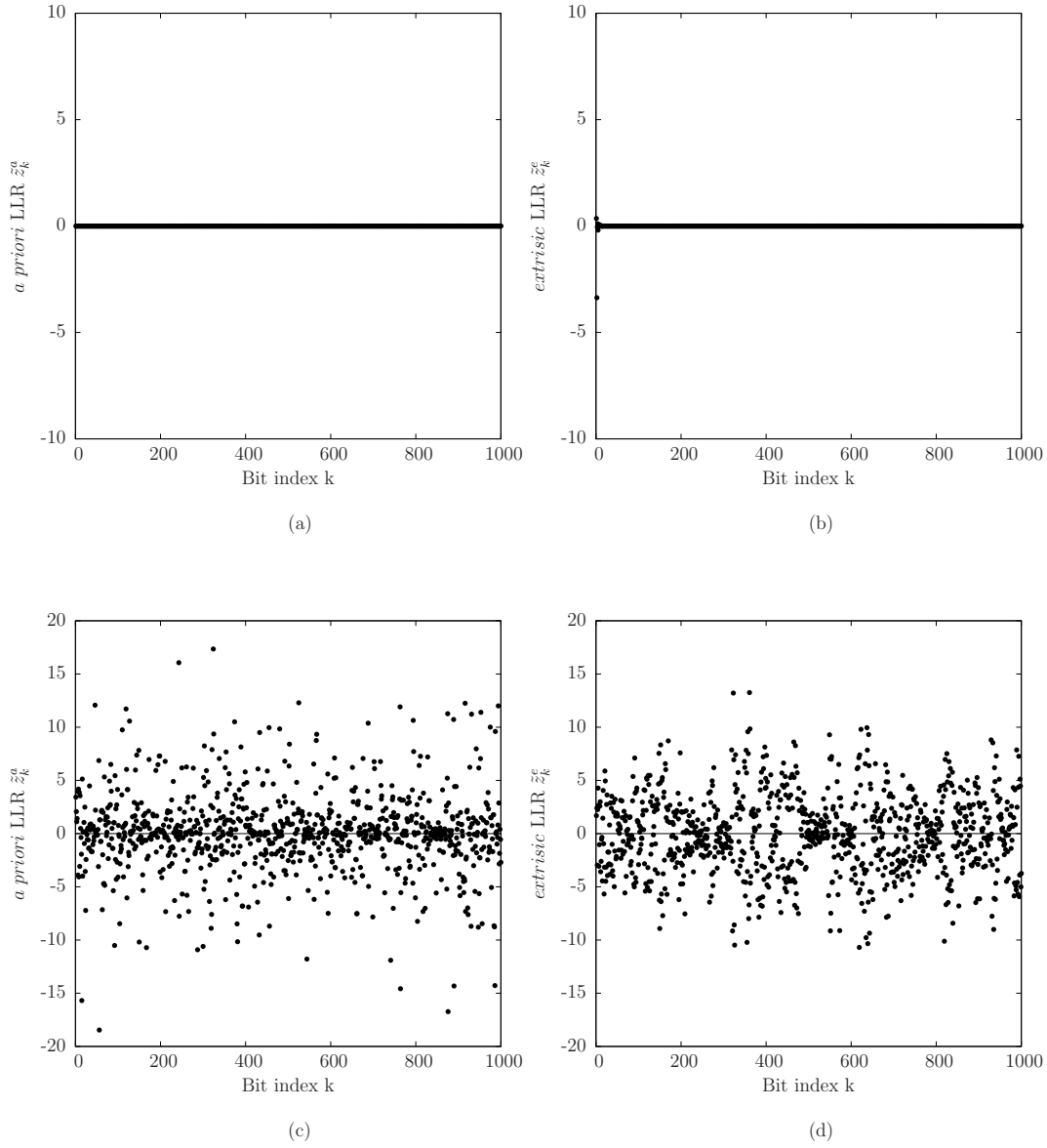


Figure 2.14: (a) ALL-zero *a priori* LLRs $\tilde{\mathbf{z}}^a$. (b) Corresponding all-zero *extrisic* LLRs $\tilde{\mathbf{z}}^e$. (c) Non-zero *a priori* LLRs $\tilde{\mathbf{z}}^a$. (d) Corresponding non-zero *extrisic* LLRs $\tilde{\mathbf{z}}^e$. In this example, the UEC scheme of Figure 2.2 transmits frames comprising $a = 2 \cdot 10^4$ symbols that obey the Zeta distibution with $p_1 = 0.7967$ using QPSK modulation over an uncorrelated narrowband Rayleigh fading channel having $E_\mathrm{b}/N_0 = 5.5$ dB.

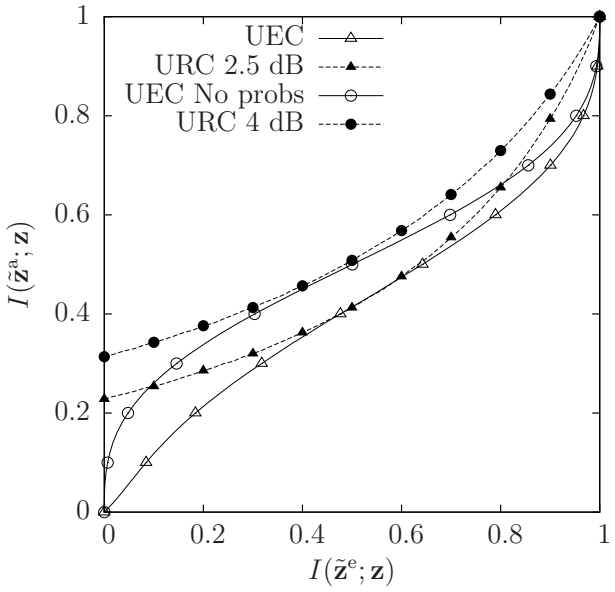


Figure 2.15: EXIT charts of the UEC scheme. Here the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$, while the UEC codewords $\mathbb{C} = [00; 01]$ comprise $n = 2$ bits and result in a UEC trellis having $r = 4$ states. Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_{b}/N_0 values.

point of perfect convergence to a vanishingly low Bit Error Ratio (BER) associated with the top right corner of the EXIT chart [71].

The EXIT chart area A that is situated below the inverted UEC EXIT curve is given by[3]

$$A = \frac{1}{n} \sum_{m'=1}^{r} \sum_{m=1}^{r} P(m, m') \log_2 \left(\frac{1}{P(m|m')} \right). \tag{2.27}$$

Substituting (2.7) and (2.8) into (2.27) and then rearranging the resultant expression yields [1]

$$\begin{aligned} A &= \tfrac{1}{ln} \textstyle\sum_{d=1}^{\frac{r}{2}-1} H[P(x)] + \tfrac{2}{ln} H\left[1 - \sum_{d=1}^{\frac{r}{2}-1} P(d)\right] \\ &+ \tfrac{1}{ln} H\left[l - \tfrac{r}{2} + \textstyle\sum_{d=1}^{r/2-1} P(d)\left(\tfrac{r}{2} - d\right)\right] \\ &- \tfrac{1}{ln} H\left[l + 1 - \tfrac{r}{2} + \textstyle\sum_{d=1}^{r/2-1} P(d)\left(\tfrac{r}{2} - 1 - d\right)\right]. \end{aligned} \tag{2.28}$$

Note that the UEC EXIT chart area A^o is independent of the UEC codebook design $\mathbb{C}$, but the design of the codebook $\mathbb{C}$ does affect the shape of the EXIT curve,

[3] This result may be obtained from [105, Equation (23)], where the notation may be converted according to $A = 1 - \mathcal{A}/I^2_{A,\max}$ and $\sum_{i=1}^{m} H(V_i) = alnH_Z$. Furthermore, we employ $H(\underline{V}|\underline{Y}) = H(\underline{V})$ as in [105, Equation (27)], since the UEC decoder is employed as an outer decoder, which has no access to channel information. Finally, we employ $H(\underline{V}) = H_{\mathbf{M}}$, where $H_{\mathbf{M}} = al \sum_{s'=1}^{r} \sum_{s=1}^{r} P(s, s') \log_2 \left(\frac{1}{P(s|s')} \right)$ is the entropy of the trellis path $\mathbf{M}$. This is justified since the proposed trellis decoder is an *A Posteriori* Probability (APP) decoder for a bit vector $\mathbf{z}$ that may be accurately modeled by the statistics of the trellis path $\mathbf{M}$.

as it will be discussed in Section 2.6.1. Figure 2.6 uses (2.28) to plot $A^o \cdot n$ for the case of using the UEC trellis decoder having various numbers of states $r \in \{2, 4, 6, 30\}$ to decode symbols obeying the Zeta distribution of (2.2), as a function of p_1.

2.5.2.3 Iterative decoding

The extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ of Figure 2.2 may be iteratively exchanged with the serially concatenated URC decoder. In order to facilitate iterative decoding convergence to an infinitesimally low BER, an open tunnel is required between the inverted UEC EXIT curve and the URC EXIT curve. Here, the URC EXIT curve quantifies the MI of the *a priori* LLR vector $\tilde{\mathbf{z}}^a$ abtained by deinterleaving the *extrinsic* LLR vector $\tilde{\mathbf{w}}^{\mathrm{e}}$ output by the URC decoder with the aid of π^{-1}, as a function of the MI of the *extrinsic* LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$, which are interleaved by π of Figure 2.2 to provide the *a priori* LLR vector $\tilde{\mathbf{w}}^a$ to be input to the URC decoder, of Figure 2.2. As we discussed in Section 2.5.2.1, if the UEC scheme employs URCs different from those recommended in Section 2.3.3.2, the URC decoder will provide an all-zero *a priori* LLR vector $\tilde{\mathbf{z}}^a$, when provided with an all-zero *extrinsic* LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ and hence the URC EXIT curve will emerge from the (0,0) point in the EXIT chart of Figure 2.15, hence preventing the creation of an open tunnel, regardless of the E_{b}/N_0 value. By contrast, the URC parametrizations recommended in Section 2.3.3.2 will facilitate the creation of an open tunnel, when the E_{b}/N_0 is sufficiently high, as shown in Figure 2.16, where the open tunnel E_b/N_0 bound is seen to be 2.5 dB. Since the URC decoder also has an EXIT curve that reaches the $(1, 1)$ point in the top right corner of the EXIT chart [106], iterative decoding convergence towards the Maximum Likelihood (ML) performance bound is facilitated [107].

Figure 2.17 shows how the extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ evolves, as $I = 1$, 3, and 5 decoding iterations are completed. As seen in Figure 2.17(a), the LLRs in the extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ have values close to 0 after the first decoding iteration, indicating a low decoding confidence. The MI of the LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ obtained during the first iteration is $I(\tilde{\mathbf{z}}^e, \mathbf{z}) = 0.4914$. After the third iteration, most of the LLR values have moved away from 0, as shown in Figure 2.17(b), indicating an increased decoding confidence associated with the MI $I(\tilde{\mathbf{z}}^e, \mathbf{z}) = 0.9588$. Finally, after the fifth iteration, all the LLR values are far away from 0, as shown in Figure 2.17(c) in conjunction with the MI $I(\tilde{\mathbf{z}}^e, \mathbf{z}) = 1$. Figure 2.18 shows the so-called iterative decoding trajectories, when the UEC scheme transmits frames comprising a number of various symbols. These trajectories are obtained by measuring and plotting the MI of $\tilde{\mathbf{z}}^{\mathrm{a}}$ and $\tilde{\mathbf{z}}^{\mathrm{e}}$ after each decoding iteration during the simulation of the transmitter and receiver of Figure 2.2. Observe that the iterative decoding trajectory corner-points

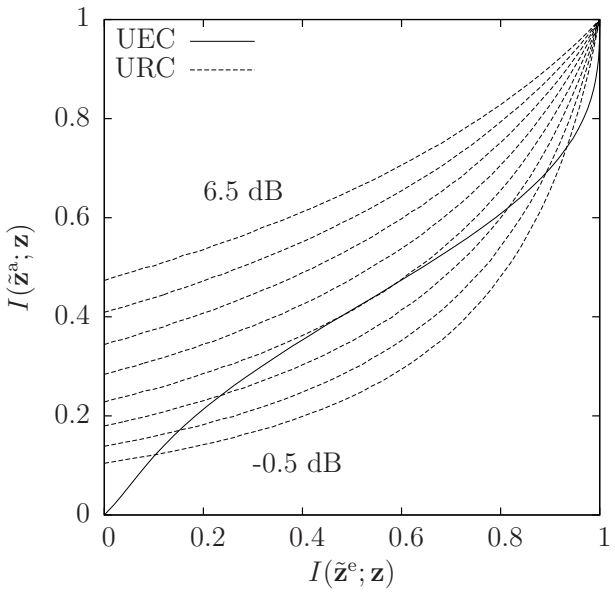


Figure 2.16: EXIT charts of the UEC scheme. Here the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$, while the UEC codewords $\mathbb{C} = [00; 10]$ comprise $n = 2$-bit and result in an UEC trellis having $r = 4$ states. Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having E_{b}/N_0 values from -0.5 dB to 6.5 dB with steps of 1 dB.

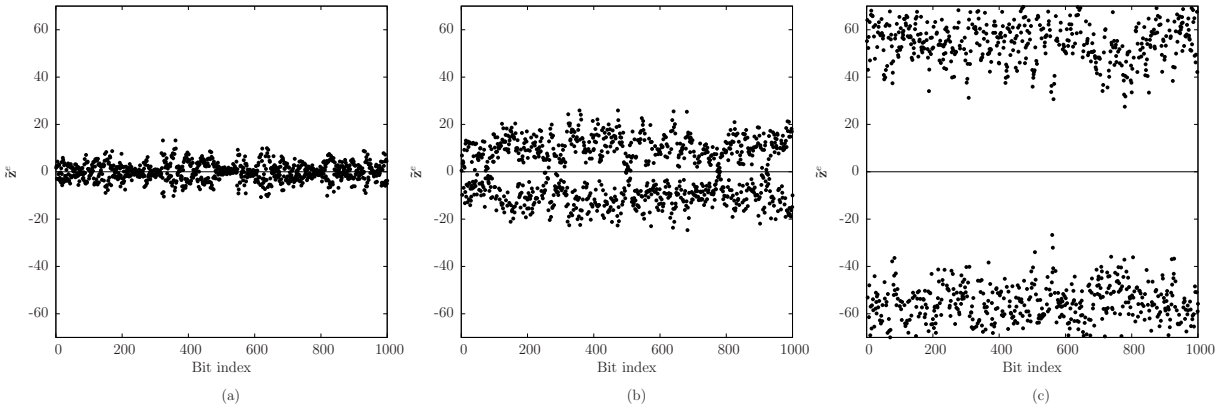


Figure 2.17: *Extrinsic* LLRs$\tilde{\mathbf{z}}^e$ of the UEC trellis decoder of Figure 2.2 when transmitting frames comprising $a = 2 \cdot 10^4$ symbols that obey the Zeta distibution with $p_1 = 0.7967$ using QPSK modulation over an uncorrelated narrowband Rayleigh fading channel having E_{b}/N_0 values equals to 5.5 dB. (a) *Extrinsic* LLRs$\tilde{\mathbf{z}}^e$ output after the first iteration with mutual information $I(\tilde{\mathbf{z}}^e, \mathbf{z}) = 0.4914$. (b) *Extrinsic* LLRs$\tilde{\mathbf{z}}^e$ output after the third iteration with mutual information $I(\tilde{\mathbf{z}}^e, \mathbf{z}) = 0.9588$. (c) *Extrinsic* LLRs$\tilde{\mathbf{z}}^e$ output after the fifth iteration with mutual information $I(\tilde{\mathbf{z}}^e, \mathbf{z}) = 1$

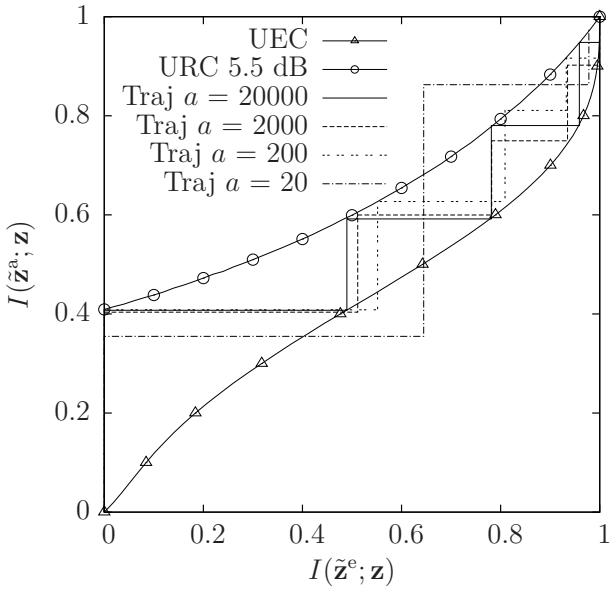


Figure 2.18: EXIT charts and iterative decoding trajectories for the UEC scheme, when transmitting frames comprising various numbers of symbols a . Here the symbols of $\mathbf{d}$ values obey a Zeta distribution having $p_1 = 0.7967$, while the UEC codewords $\mathbb{C} = [00; 01]$ comprise $n = 2$ bits and result in an UEC trellis having $r = 4$ states. Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having $E_b/N_0 = 5.5$ dB.

are on the EXIT curve of the inner and outer code in the case of frames comprising $a = 20\,000$ symbols. However, when the scheme employs shorter frames, the iterative decoding trajectory has a poorer match with the EXIT curve, which may prevent the trajectory from reaching the (1,1) point of the EXIT chart. This may be explained by the Log-BCJR algorithm's sensitivity to the correlation that is manifested in short vectors of *a priori* LLRs [9].

2.5.3 Unary decoder

Following the completion of the iterative decoding process, the UEC trellis decoder may invoke the Log-BCJR algorithm for generating the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}^{\mathrm{p}} = [\tilde{y}_j^{\mathrm{p}}]_{j=1}^{b}$ that pertain to the corresponding bits in the vector $\mathbf{y}$. Following this, the unary decoder of Figure 2.2 exploits the obseration that each unary codeword contains exactly one bit having the value 1 and hence there must be a number of logical one valued bits in the bit vector $\mathbf{y}$. More specifically, the unary decoder sorts the values in the *a-posteriori* LLR vector $\tilde{\mathbf{y}}^{\mathrm{p}}$ in order to identify the a number of bits in the vector $\mathbf{y}$ that are most likely to have values of logical one. A hard decision vector $\hat{\mathbf{y}}$ is then obtained by setting the value of these bits to logical zero and the value of all other bits to logical one. Finally, the bit vector $\hat{\mathbf{y}}$ can be unary decoded in order to obtain the symbol vector $\hat{\mathbf{x}}$ of Figure 2.2, which is guaranteed to comprise a

number of symbols. Figure 2.19 shows the SER performance of the scheme shown in

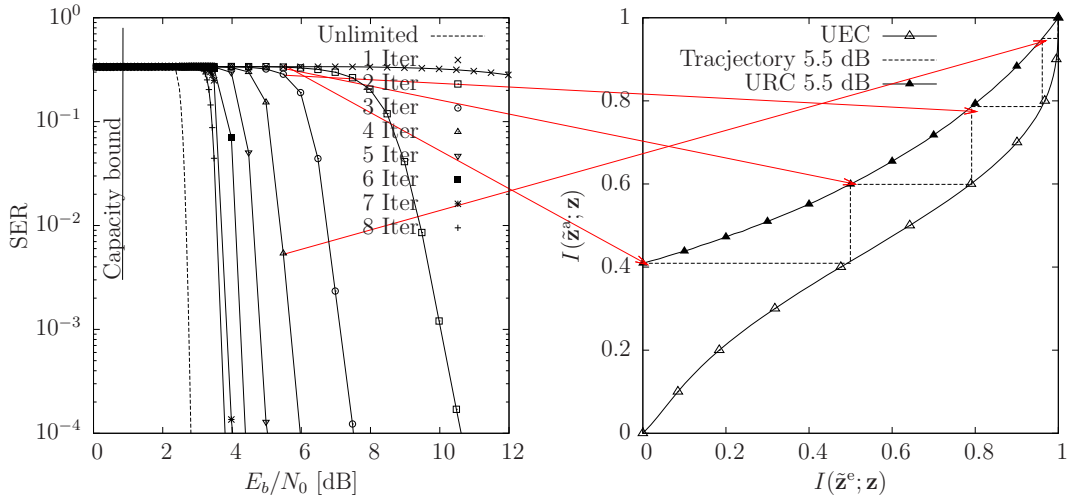


Figure 2.19: SER performance of the UEC scheme and the related EXIT chart when transmitting frames comprising $a = 2 \cdot 10^4$ symbols that obey the Zeta distibution with $p_1 = 0.7967$ using QPSK modulation over an uncorrelated narrowband Rayleigh fading channel.

Figure 2.2 after different number of iterations. Note that each iteration achieves more than 1 dB gain over the previous iteration, until five iterations have been completed. After the fifth iteration, the gain becomes less than 1 dB since the decoding trajectory converges to the (1,1) point of the EXIT chart. The capacity bound in Figure 2.2 will be discussed in Section 2.7.2.

Figure 2.20 illustrates the associated SER performance, when the UEC scheme transmits frames comprising a number of symbol.

In the example detailed in this chapter, the following reconstructed symbol vectors are obtained after each iteration.

$$1^{st} \text{ iteration}: \quad \hat{\mathbf{d}} = [2,1,3,4,1,2,1,1]$$

$$\vdots$$

$$5^{th} \text{ iteration}: \quad \hat{\mathbf{d}} = [2,1,3,4,1,2,1,1] \ .$$

Note that because the UEC scheme only employs the interleaver length of 30 bits, one symbol error remains, even when we have $E_\mathrm{b}/N_0 = 5.5\,\mathrm{dB}$ in conjunction with 5 iterations.

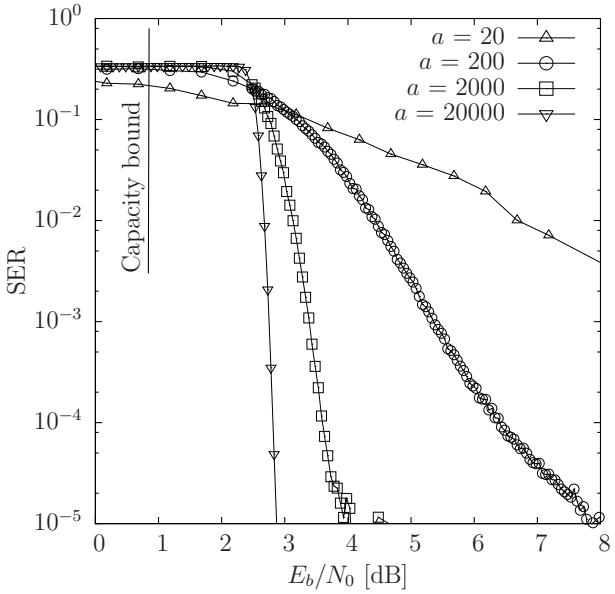


Figure 2.20: SER performance of the UEC scheme when transmitting frames comprising having various number of symbols a that obey the Zeta distibution with $p_1 = 0.7967$ and when using QPSK modulation for transmission over an uncorrelated narrowband Rayleigh fading channel.

2.6 The parameterization of the Unary Error Correction code

In the following subsections, we detail the considerations that dictate the selection of the number of UEC trellis states r to employ in order to achieve near-capacity operation, as well as the selection of the UEC codebook $\mathbb{C}$ in order to optimise the UEC SER performance.

2.6.1 Performance analysis

Near-capacity operation is achieved, when reliable communication can be maintained at effective throughputs η that approach the DCMC capacity C of the channel. When the UEC code of Figure 2.2 is serially concatenated with an URC code, near-capacity operation is facilitated, provided that the area A^o beneath the UEC code's inverted EXIT curve is equal to its coding rate R^o [105]. In this case, near-capacity operation will be achieved, if the CC decoder's EXIT curve has a shape closely matching that of the UEC decoder, hence creating a narrow but still open EXIT chart tunnel and facilitating iterative decoding convergence towards the ML performance.

In the case of the Zeta distribution of (2.2), Figures 2.6 and 2.21 suggest that the UEC EXIT area A^o asymptotically approaches the UEC coding rate R^o, as the number of states employed in the UEC trellis decoder r is increased. In fact, this is the case, regardless of which particular symbol value distribution is adopted by the

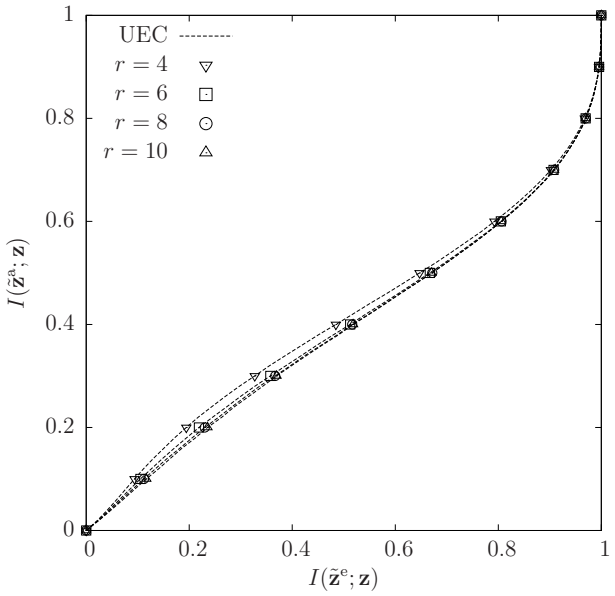


Figure 2.21: EXIT charts of the UEC scheme. Here the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$, while the UEC codewords $\mathbb{C} = [00; 01]$ comprise $n = 2$ bits and result in an UEC trellis having number of states $r \in \{4, 6, 8, 10\}$.

RVs in the vector $\mathbf{X}$. This may be proved by observing that the last three terms in (2.28) tend to zero as r is increased, leaving only the first term of

$$\lim_{r \to \infty} A^o = \frac{1}{ln} \sum_{d \in \mathbb{N}_1} H[P(d)], \tag{2.29}$$

which is equal to the expression provided for the coding rate R^o in (2.10). Figure 2.22 plots the discrepancy between $R^o n$ and $A^o n$ as a function of the number of UEC states r for Zeta distributions employing various values for the parameter p_1. Note that in all considered cases, the discrepancy becomes less than 10^{-2}, when at least $r = 30$ states are employed.

As described in Section 2.3.2, a UEC trellis having r number of states is parametrized by a set of $r/2$ codewords $\mathbb{C}$, each comprising n number of bits, where $\mathbb{C} = [00; 01]$ in the $r = 4$, $n = 2$ example of Figure 2.4(b) and $\mathbb{C} = [00; 01; 01]$ in the $r = 6$, $n = 2$ example of Figure 2.4(c). Note that when provided with the same unary-encoded bit vector $\mathbf{y}$, UEC trellis encoders employing the trellises of Figures 2.4(b) and 2.4(c) are guaranteed to generate identical UEC-encoded bit vectors $\mathbf{z}$, despite using different codebooks $\mathbb{C}$. This is because the $r = 6$ codebook of Figure 2.4(c) is an *extension* of the $r = 4$ codebook of Figure 2.4(b). In this way, the employment of extension allows a higher number of states r to be used in the UEC trellis decoder than in the UEC trellis encoder. This allows us to dynamically change the number of states employed

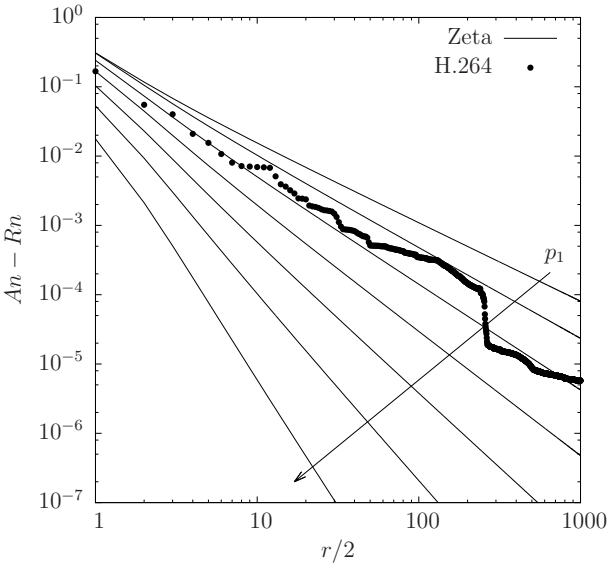


Figure 2.22: The discrepancy between $A^o n$ and $R^o n$ for the UEC scheme as a function of the number of states employed by the UEC trellis decoder r, for the case of various source distributions associated with $p_1 \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.

in the decoder in order to strike an attractive trade-off between its trellis complexity, which is proportional to r and near-capacity performance, which improves with r [99].

Figures 2.23 and 2.24 illustrate the SER performance of the UEC scheme having

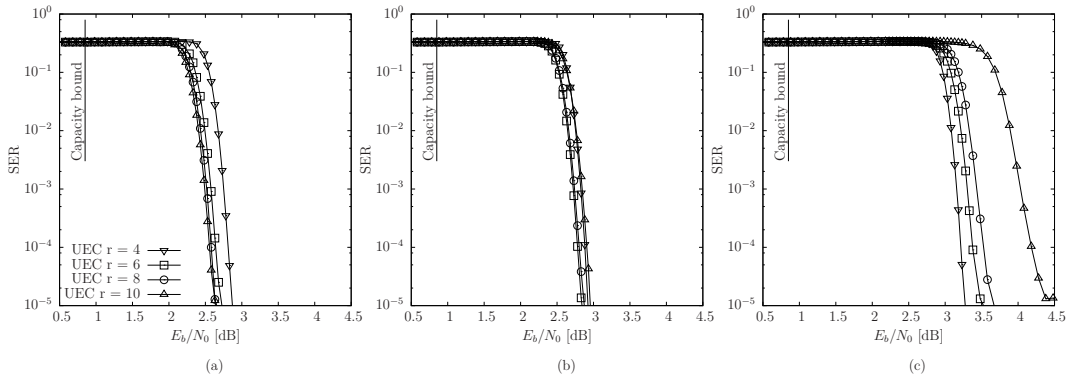


Figure 2.23: The SER performance of the UEC scheme of Figure 2.2 using as extended version of the codebook $\mathbb{C} = [00; 01]$ having $r \in \{4, 6, 8, 10\}$ number of states, when conveying symbols obey a Zeta distribution having the parameter $p_1 = 0.7967$ Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_b/N_0 values. A complexity limit of (a) unlimited, (b) 8000 and (c) 4000 ACS operations per decoding iteration is imposed for decoding each of the symbols in $\mathbf{d}$.

$r \in \{4, 6, 8, 10\}$ states when conveying symbols obeying a Zeta distribution having the parameter of $p_1 \in \{0.7967, 0.9\}$. In the case when $p_1 = 0.7967$ and there is no complexity limit, the UEC schemes associated with $r \in \{8, 10\}$ states have around

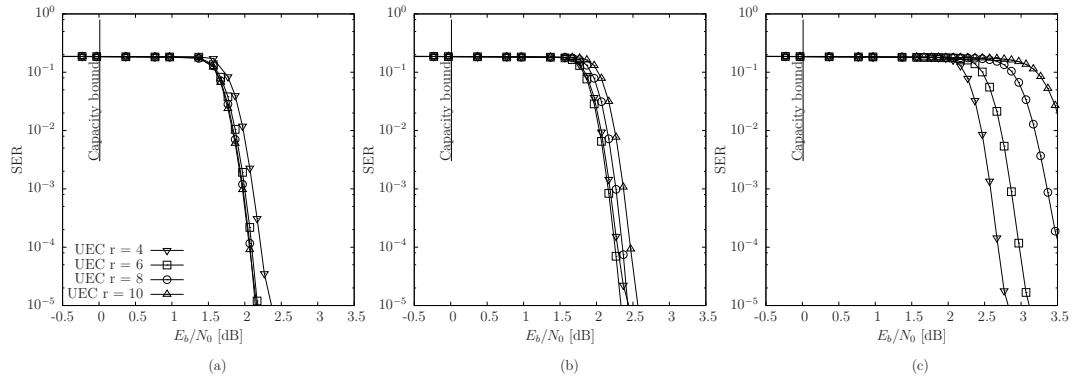


Figure 2.24: The SER performance of the UEC scheme of Figure 2.2 using as extended version of the codebook $\mathbb{C} = [00; 01]$ having $r \in \{4, 6, 8, 10\}$ number of states, when conveying symbols obeying a Zeta distribution having the parameter $p_1 = 0.9$ Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_b/N_0 values. A complexity limit of (a) unlimited, (b) 6000 and (c) 3000 ACS operations per decoding iteration is imposed for decoding each of the symbol in $\mathbf{d}$.

0.3 dB gain over the UEC scheme with the $r = 4$-state trellis, while the gain is 0.2 dB for the UEC scheme using the $r = 6$-state trellis over the UEC associated with the $r = 4$-state trellis, as shown in Figure 2.23(a). When the complexity limit of 8000 ACS operations is imposed for decoding each of the symbols in $\mathbf{d}$, this gain reduces to 0.1 dB for the schemes with $r \in \{6, 8\}$, while the $r = 10$-state scheme has the same SER performance, as the $r = 4$ scheme . The $r = 4$-state trelllis has the best performance for ACS=4000, offering the largest gain of 1 dB over the other schemes. The SER performance is similar, when $p_1 = 0.9$. Thus having $r = 4$ UEC trellis states represents an attractive trade-off between maintaining a low trellis complexity and facilitating near-capacity operation.

2.6.2 Unary Error Correction codebook selection

In this section, we will discuss the design of the codebook $\mathbb{C}$ for the case of the $n = 2$ $r = 4$ REGEC trellis. This selection is motivated since $n = 2$ is the minimum Free Distance (FD) required for achieving an infinitesimally low BER and for the EXIT curve to reach the (1,1) point in the EXIT chart. Furthermore, $r = 4$ was shown in Section 2.6.1 to offer an attractive trade off between the trellis complexity imposed and approaching near-capacity operation. An $n = 2$ $r = 4$ codebook $\mathbb{C}$ comprises $r/2 = 2$ codewords, each constituted by $n = 2$ bits. Therefore, there are 2^4 possible $n = 2$ $r = 4$ codebooks. However it can be readily shown that all of these are equivalent to one of the 3 codebooks shown in Table 2.4, which contains no pairs of equivalent codebooks. More specifically, recall that two codebooks are equivalent

Table 2.4: Candidate REGEC codebooks $\{\mathbb{C}_i\}_{i=1}^{3}$ for $n = 2$ bits and $r = 4$ states as well as their corresponding FD d_f. For Zeta probability distributions having $p_1 = 0.7967$, the number of states in the URC having the best matching EXIT curve is provided, together with the corresponding E_b/N_o tunnel bound in brackets.

candidate codebook		$\mathbb{C}_1$	$\mathbb{C}_2$	$\mathbb{C}_3$
$\mathbf{c}_1$		00	00	00
$\mathbf{c}_2$		00	01	11
d_f		2	4	4
Number of states in URC having best matching EXIT chart and resultant E_b/N_0 tunnel bound in dB	p_1 =0.7967	8 (1.8)	2 (2.5)	4 (2.3)

if each pairing of codewords within one of the codebooks has the same Hamming Distance (HD) as the corresponding pairing of codewords within the other codebook. Owing to this, two codebooks are equivalent if one of them can be transformed into the other by toggling all bits and/or changing the order of the bits in each codeword using the same reordering pattern.

The error correction capability of a codebook may be characterized by the FD of legitimate codewords at the output of the UEC trellis encoder [75]. The FD represents the minimum HD between any pair of encoded bit vectors produced by the different paths traversing through the UEC trellis. The total number of possible pairings of paths emerging from a particular state in a UEC trellis of length b is given by $2^{b-1}(2^b - 1)$, which grows exponentially with b. However, considering the symmetry of a UEC trellis, it is possible to use a step-by-step directed search for determining the FD, rather than using a brute force exhaustive search. Note that in the UEC trellis generalised in Figure 2.4(a), a bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$ identifies a unique path $\mathbf{m} = [m_j]_{j=0}^{b}$ that emerges from state 1 and terminates at either state 1 or 2, hence accordingly identifying a corresponding output bit sequence $\mathbf{z} = [z_j]_{j=1}^{bn}$. By exploiting this observation, the FD d_f can be obtained by computing the HD between each pair of paths and then selecting the specific pair having the minimum HD, whenever two paths merge at a particular state in the trellis [2]. Table 2.4 shows that the largest possible FD of the $n = 2$-bit $r = 4$-state UEC codes is 4.

As discussed in Section 2.6.1, the area A^o beneath the inverted UEC EXIT function and the UEC coding rate R^o are independent of the codebook design $\mathbb{C}$. However the shape of the UEC EXIT curve and therefore its match with the URC EXIT curve does depend on the codebook design $\mathbb{C}$. Since the candidate codebooks of Table 2.4 are unique with no pair of codebooks that are equivalent to each other, their inverted EXIT curves are all different from each other. Owing to this, different candidate codebooks have inverted EXIT curves that match best with the EXIT curve of URC

codes having different parametrizations. Figure 2.15, 2.25 and 2.26 show the resultant EXIT charts for the cases of using each of the candidate codebooks in Table 2.4 to encode symbols obeying the Zeta distribution having $p_1 = 0.7967$, as well as the resultant EXIT charts recorded for the cases where the receivers have no knowledge of the symbol probability distribution. In this case the $P(m|m')$ term is simply omitted from (2.21) during the UEC trellis decoder's Log-BCJR algorithm. Table 2.4 suggests that $\mathbb{C}_1$ should offer the best performance in the turbo cliff region of the Symbol Error Ratio (SER) plot, since it offers an open EXIT chart tunnel at the lowest E_b/N_0 value, implying that iterative decoding convergence to an approximation of the ML SER performance can be achieved [108].

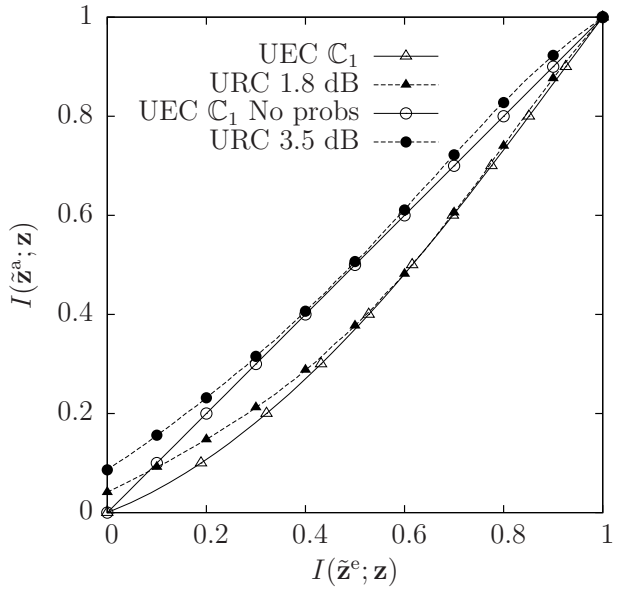


Figure 2.25: EXIT charts of the UEC scheme. Here the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$, while the UEC codewords $\mathbb{C}_1 = [00; 00]$ comprise $n = 2$ bits and result in a UEC trellis having $r = 4$-state. Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_{b}/N_0 values.

However, codebook $\mathbb{C}_1$ may not offer the best performance in the error floor region of the SER plot, as shown in Figure 2.27. Codebooks $\mathbb{C}_2$ and $\mathbb{C}_3$ offer steep turbo cliffs at E_b/N_0 values near the corresponding E_b/N_0 tunnel bounds, as predicted by the EXIT chart results of Table 2.4. The SER of these schemes drops to 10^{-4} within 0.3 dB of these E_b/N_0 tunnel bounds. By contrast, the SER of codebook $\mathbb{C}_1$ has an error floor above 10^{-3}, even when the E_b/N_0 value is 3 dB above its corresponding tunnel bound. In the case when the decoder has no knowledge of the symbol distribution, codebook $\mathbb{C}_2$ avoids having an error floor, while codebook $\mathbb{C}_1$ and $\mathbb{C}_3$ both suffer from pronounced error floors. Also, the candidate codebook $\mathbb{C}_2$ works best in conjunction with the URC inner code having the lowest complexity, namely that employing only

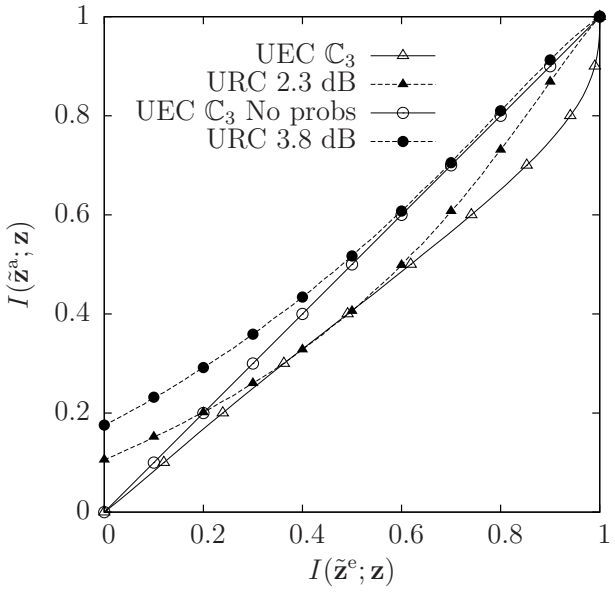


Figure 2.26: EXIT charts of the UEC scheme. Here the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$, while the UEC codewords $\mathbb{C}_3 = [00; 11]$ comprise $n = 2$ bits and result in a UEC trellis having $r = 4$ states. Furthermore, a URC decoder having $r = 2$- state is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_{b}/N_0 values.

$r = 2$ states. Therefore, we employ the candidate codebook $\mathbb{C}_2$ throughout the next section, when we compare the performance of the UEC scheme to a suitably designed SSCC benchmarker.

2.7 Performance comparison with the SSCC benchmarker

In this section, we introduce the EG-CC SSCC benchmarker and compare its performance to that of our UEC scheme.

2.7.1 EG-CC SSCC benchmarker

Figure 2.28 illustrates the architecture of an EG-CC SSCC benchmarker, which offers a fair comparison with the proposed UEC JSCC scheme. In the transmitter, the unary encoder of the UEC scheme seen in Figure 2.2 is replaced by an EG encoder. This employs the codewords shown in Table 2.2, yielding the $b = 18$-bit vector $\mathbf{y} = 010101100100110101$, when the vector of $a = 8$ symbols $\mathbf{d} = [2, 1, 3, 4, 1, 1, 2, 1]$ is encoded, for example. The average EG codeword length is given by

$$l_{EG} = \sum_{d \in \mathbb{N}_1} P(d) \left(2\lfloor \log_2(d) \rfloor + 1\right), \tag{2.30}$$

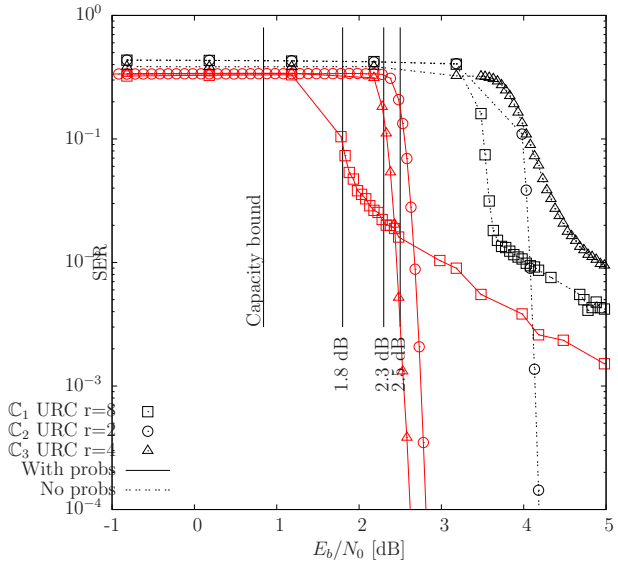


Figure 2.27: SER vs E_b/N_0 plot for the UEC codes employing the $n = 2$-bit $r = 4$-state codebooks $\mathbb{C} \in \{\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3\}$, when combined with URC codes having $r \in \{2, 4, 8\}$ states with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel.

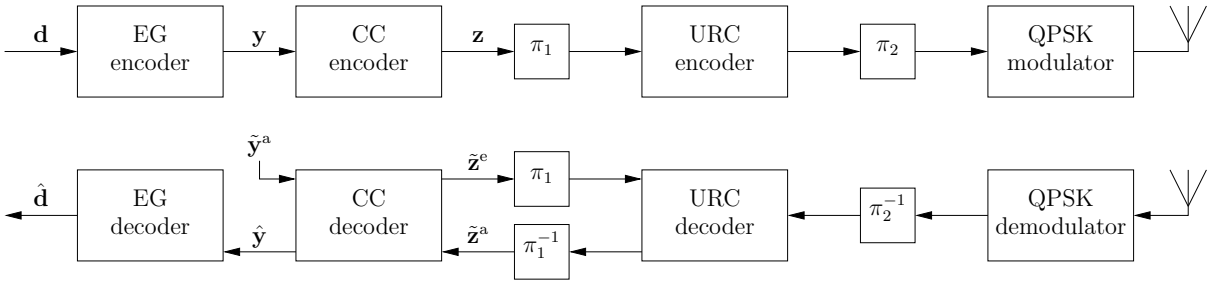


Figure 2.28: Schematic of the EG-CC code, when serially concatenated with URC and Gray-coded QPSK modulation schemes. Bold notation without a diacritic is used to denote a symbol vector or a bit vector. A diacritical hat represents a reconstruction of the symbol or bit vector having the corresponding notation. A diacritical tilde represents an LLR vector pertaining to the bit vector with the corresponding notation. A roman superscript 'a' is employed to denote an *a priori* LLR vector, while 'e' is employed for extrinsic LLR vectors. Furthermore, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. Puncturing may also be performed in π_2, while the corresponding depuncturing operations take place in π_2^{-1}.

which is guaranteed to be finite for any monotonic symbol value distribution having $P(d) \geq P(d+1)\ \forall\ d \in \mathbb{N}_1$, including the Zeta distribution of (2.2), when we have $p_1 \leq 0.608$. When $\mathbf{d}$ obeys the Zeta distribution, the average EG codeword length is given by

$$l_{\text{EG}} = 1 - \frac{2\zeta'(s)}{\ln(2)\zeta(s)} - \frac{2}{\zeta(s)} \sum_{x \in \mathbb{N}_1} x_i^{-s} \text{frac}[\log_2(x_i)], \tag{2.31}$$

where the frac$(\cdot)$ operator yields the fractional part of the operand, as in frac$(3.4) = 0.4$ and $x = \lfloor \log_2(d) \rfloor + 1$. Even though the H.264 distribution of Figure 2.3 is not monotonic, a finite average EG codeword length of $l = 3.029$ bits per symbol is obtained. As in the proposed UEC scheme, the b-bit vector $\mathbf{y}$ may be modeled as a realization of a vector $\mathbf{Y} = [Y_j]_{j=1}^{b}$ comprising b binary RVs. In the general case, these RVs do not adopt equiprobable values $\Pr(Y_j = 0) \neq \Pr(Y_j = 1)$, hence we have a less than unity bit entropy:

$$H_{Y_j} = H[\Pr(Y_j = 0)] + H[\Pr(Y_j = 1)]. \tag{2.32}$$

In order to generate equiprobable bits, the trellis encoder of the UEC scheme is replaced by a $1/n$-rate r-state CC encoder, as shown in Figure 2.28. We recommend the specific CCs obeying the generator and feedback polynomials provided in Table 2.5. More specifically, we found that these non-systematic recursive CCs offer the optimal distance properties [109] subject to the constraint of producing equiprobable bits $\Pr(Z_k = 0) = \Pr(Z_k = 1)$. This $H_Z = 1$ constraint has to be satisfied for avoiding any capacity loss, when the EG-CC scheme is serially concatenated with an URC [87].

The average coding rate R^o of the EG-CC encoder is given by (2.10). When the RVs in the vector $\mathbf{D}$ obey the Zeta distribution of (2.2), the product of the EG-CC coding rate R^o and the codeword length n is related to the distribution parameter p_1, as shown in Figure 2.6.

Table 2.5: The optimal generator and feedback polynomials that satisfy the $H_Z = 1$ constraint. Polynomials are provided in the format $(\mathbf{g}, f, d_{\text{free}})$, where $\mathbf{g}$ is an n-element vector of octal generator polynomials, f is the octal feedback polynomial and d_{free} is the decimal free distance [1].

r	n		
	2	3	4
2	([2,2],3,2)	([2,2,2],3,3)	([2,2,2,2],3,4)
4	([4,7],6,4)	([4,7,7],6,6)	([4,7,7,7],6,8)
8	([15,17],16,6)	([13,15,17],16,10)	([13,15,15,17],16,13)
16	([27,31],34,7)	([25,33,37],36,12)	([25,33,35,37],32,16)

In the receiver, the trellis decoder of the UEC scheme is replaced by a CC decoder, as shown in Figure 2.28. This employs the Log-BCJR algorithm during the iterative decoding process, in order to convert the *a priori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{a}}$ into the extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$. As shown in Figure 2.28, the BCJR algorithm is capable of exploiting some of the residual redundancy present within the bit vector $\mathbf{y}$ by employing the corresponding vector of *a priori* LLRs $\tilde{\mathbf{y}}^{\mathrm{a}} = [\tilde{y}_j^{\mathrm{a}}]_{j=1}^{b}$. Here, we have

$$\tilde{y}_j^{\mathrm{a}} = \ln\left(\frac{\Pr(Y_j = 0)}{\Pr(Y_j = 1)}\right), \tag{2.33}$$

where the bit value probabilities may be obtained heuristically.

As in the UEC scheme, the Log-BCJR algorithm may be characterized by the corresponding inverted EG-CC EXIT curve, as exemplified in Figure 2.29. It can be

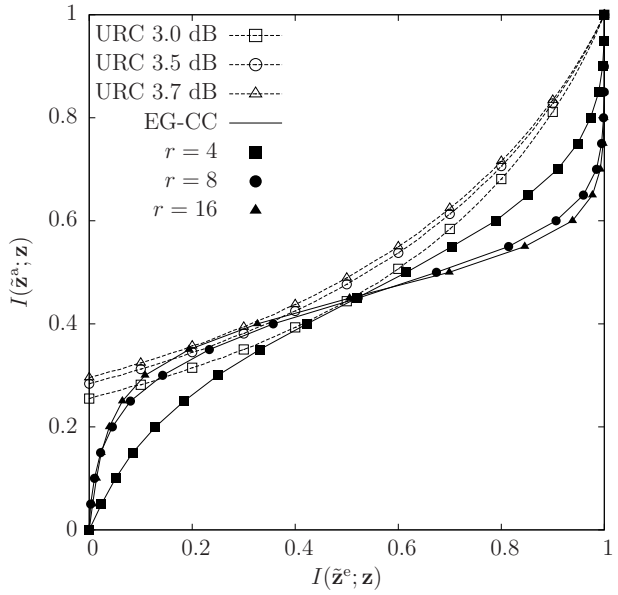


Figure 2.29: EXIT charts of the EG-CC scheme. Here the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$, while the CC trellis employs $r \in \{2, 8, 16\}$ states and $n = 2$ bits. Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_{b}/N_0 values.

shown that the EXIT chart area A that is situated below the inverted EG-CC EXIT curve is given by [1]

$$A^o = \frac{\sum_{j=1}^{b} H_{Y_j}}{bn}. \tag{2.34}$$

Note that unlike in the UEC scheme, the EG-CC EXIT chart area A^o is independent of the number of states r employed in the CC, as exemplified in Figure 2.29. Moreover, in contrast to the UEC scheme, increasing the number of states r employed in the CC will increase the open tunnel bound and hence degrades the performance of the

Table 2.6: Open tunnel bound of EG-CC scheme when employing CC trellises having $n = 2$ bits and different numbers of states r. Here, $\mathbf{d}$ obeys the Zeta probability distribution having $p_1 = 0.7967$. The number of states in the URC having the best matching EXIT curve is provided, together with the corresponding E_b/N_o tunnel bound in brackets.

Scheme	p_1	r	A^o	Number of states in URC having best matching EXIT chart and resultant E_b/N_0 tunnel bound in dB
EG-CC	0.7967	4	0.4410	2 (3.0)
		8		2 (3.5)
		16		2 (3.7)

EG-CC scheme, as shown both in Table 2.6 and Figure 2.29. This is because different number of states r employed in the CC will change the shape of the EXIT curve. Owing to this, we will use the CC code of Table 2.5 having $r = 4$ states and $n = 2$ bits as the benchmark throughout this thesis.

Note that in the EG-CC scheme, the values of $A^o n$ and $R^o n$ are separated by significant discrepancies of up to about 0.2, preventing near-capacity operation, as shown in Figure 2.5. This represents a significant disadvantage compared to the proposed UEC scheme, which is capable of eliminating the discrepancy, when the symbol values obey the Zeta distribution of (2.2). Meanwhile, the proposed UEC scheme can reduce the discrepancy to an infinitesimally small value by employing a sufficiently high number r of states, as discussed in Section 2.6.1.

Additionally, the EG-CC suffers from another significant disadvantage compared to the proposed UEC scheme, once iterative decoding convergence has been achieved. Explicitly, in this event, the CC decoder of Figure 2.2 may generate the decoded bit vector $\hat{\mathbf{y}}$ by employing the Viterbi algorithm, which can also exploit the *a priori* LLR vector $\tilde{\mathbf{y}}^{\mathrm{a}}$. Following this, the EG decoder of Figure 2.2 extracts the symbol vector $\hat{\mathbf{d}}$ by interpreting $\hat{\mathbf{y}}$ as a concatenation of the EG codewords shown in Table 2.2. However, if $\hat{\mathbf{y}}$ contains any bit errors, there is no guarantee that it will comprise the correct number a of codewords, or even that it will comprise an integer number of codewords. For example, the first 13 bits in the vector $\hat{\mathbf{y}} = 011001010111111$ correspond to a legitimate concatenation of eight EG codewords, but the last two bits do not form a complete legitimate codeword. In the case where $\hat{\mathbf{y}}$ does not correspond to a number of symbols, some symbols may be truncated from $\hat{\mathbf{d}}$ or a number of 1-valued symbols may be appended to $\hat{\mathbf{d}}$, as appropriate.

2.7.2 SER performance

Table 2.7 provides several parametrizations of the UEC scheme, which are designed for transmitting symbols that obey the Zeta distribution of (2.2) having different values of p_1. Table 2.7 also provides the corresponding parametrizations for the SSCC EG-CC benchmarker, which offer the same throughput η as our UEC scheme parametrizations. We parametrize the Zeta distribution using $p_1 \in \{0.9, 0.7967, 0.694\}$, which represents a wide selection of the p_1 values higher than 0.608, as shown in Figure 2.6. Recall that the specific value of $p_1 = 0.7967$ is chosen, since it results in the same coding rate for the unary code and the EG code, hence yielding the same outer coding rate R^o for both the UEC scheme and the EG-CC scheme. We selected codewords comprising $n = 2$ bits and employ $r = 4$-state trellises for both the UEC and for the CC, as we recommended in Sections 2.6.1 and 2.7.1, respectively.

Table 2.7: Outer coding rate R^{o}, inner coding rate R^{i} and throughput η for the UEC and EG-CC schemes designed for encoding Zeta distributed symbols having different p_1 values.

p_1	Scheme	n	r	R^{o}	A^{o}	R^{i}	η	E_{b}/N_0 [dB] for $C = \eta$	E_{b}/N_0 [dB] for $A^{\mathrm{i}} = A^{\mathrm{o}}$	E_{b}/N_0 [dB] for open tunnel	Complexity per iteration per symbol of **d**
0.9	UEC	2	4	0.2636	0.2682	1	0.5272	0.01	0.1	1.5	250
	EG-CC	2	4	0.2492	0.3247	1.0578			1.6	2.4	257
0.7967	UEC	2	4	0.3810	0.4041	1	0.7620	0.84	1.3	2.5	331
	EG-CC	2	4	0.3810	0.4410	1			2.0	3.0	322
0.6940	UEC	2	4	0.3112	0.3654	1.4565	0.9066	1.43	2.7	4.5	614
	EG-CC	2	4	0.4533	0.4877	1			2.0	3.0	410

The other parameters listed in Table 2.7 are described as follows.

1. Capacity bound (E_{b}/N_0 [dB] for $C = \eta$)
   The capacity bound is the lowest SNR, where reliable transmission is theoretically possible. It is the specific E_b/N_0 value, where the DCMC capacity C becomes equal to the effective throughput η of (2.11). Note that the capacity bound depends on the value of the parameter p_1 because of the different effective throughputs that result. However, the capacity bound of the UEC and EG-CC schemes designed for the same p_1 are the same, as shown in Figure 2.30. This is because we use puncturing to maintain the same effective throughput η for both schemes, in order to achieve a fair comparison.
2. Area bound (E_{b}/N_0 [dB] for $A^{\mathrm{i}} = A^{\mathrm{o}}$)
   As we discussed in Section 2.5.2.3, the area A^o beneath the URC EXIT function will increase with the E_b/N_0 value. The area bound identifies the specific E_b/N_0

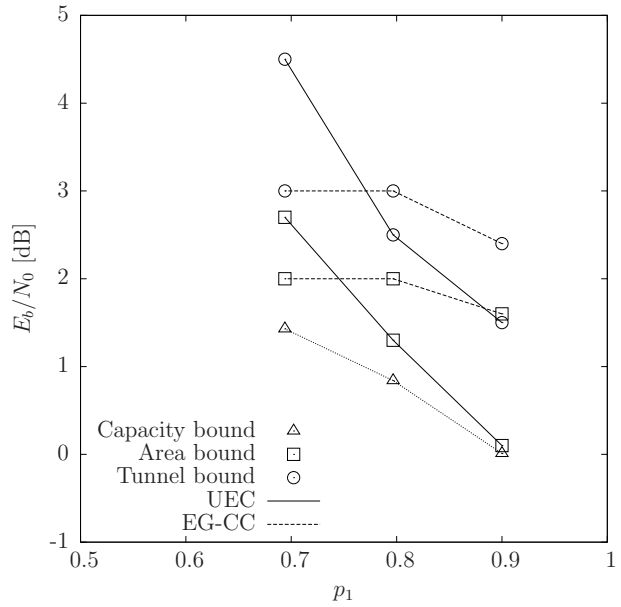


Figure 2.30: Capacity bound, area bound and tunnel bound for the UEC and EG-CC schemes designed for encoding Zeta distributed symbols having different p_1 values.

value, where the area A^o beneath the inverted outer decoder's EXIT curve becomes equal to the area A^i beneath the inner decoder's EXIT curve. This is the theoretical bound, where it is possible to create an open tunnel and to achieve a low SER during iterative decoding. Note that the area bounds of the UEC and EG-CC schemes are different, since these two schemes have different values for the areas A^o. The gap between the area bound and capacity bound quantifies the capacity loss of the scheme, as shown in as shown in Figure 2.30. This capacity loss can be reduced by increasing the number of states r employed in the trellis of the UEC code, as we discussed in Section 2.6.1. However, the capacity loss will remain the same for the SSCC EG-CC schemes regardless of the number of states r employed in the CC trellis. In particular, the EG-CC scheme can be seen to suffer from upto 1.59 dB of capacity loss in Table 2.7.

3. Tunnel bound

   The tunnel bound is the lowest E_b/N_0 value required for creating an actual open tunnel in the EXIT chart, , as shown in as shown in Figure 2.30. Note that the tunnel bound typically has a higher E_b/N_0 value than the area bound, since the shape of the inner and outer EXIT curves are typically not perfectly matched. The gap between the area bound and tunnel bound may be reduced by using irregular outer and inner codes for both schemes. However, as we discussed in Section 2.3.3.2, an irregular code must be tailored for a certain value of the parameter p_1, hence preventing its general applicability.

As shown in Figure 2.31, the proposed UEC scheme facilitates reliable communication

within 2 dB of the capacity bound and consistently offers the best SER performance for the cases of Zeta distributions having the parameters of $p_1 = 0.9$ and 0.7967, while having similar complexities to that of the EG-CC benchmarker. In the scenario where $p_1 = 0.7967$, the unary code and EG code have the same outer coding rate, allowing the same inner coding rate of 1 to be used for both scheme. In this case, our UEC scheme offers 0.6 dB gain over the EG-CC benchmarker. In the case where $p_1 = 0.9$, this gain is increased to 1.6 dB, where puncturing is employed by the EG-CC scheme. When $p_1 = 0.694$ however, the performance of our UEC scheme is 3 dB worse than that of the EG-CC benchmarker. This may be attributed to the severe puncturing that is required in our UEC scheme. Furthermore, the UEC scheme imposes a significantly higher decoding complexity than the EG-CC benchmarker, when $p_1 = 0.6940$, as shown in Table 2.7. This disappointing UEC performance in the case of low p_1 values will be addressed by the EGEC and REGEC schemes of the following chapters.

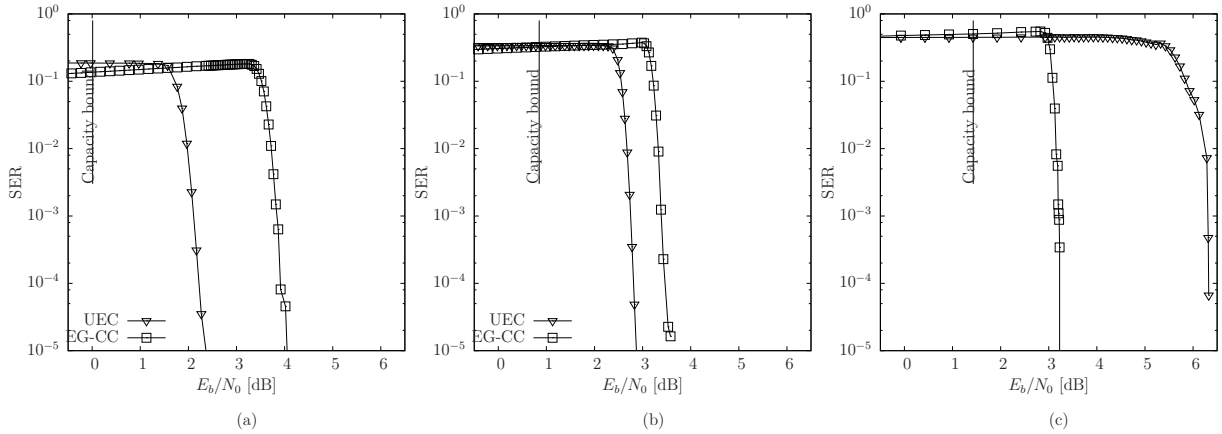


Figure 2.31: The SER performance of the UEC and EG-CC schemes when conveying symbols obeying a Zeta distribution having the parameter (a) $p_1 = 0.9$, (b) $p_1 = 0.7967$ and (c) $p_1 = 0.6940$. In all cases, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_b/N_0 values. No complexity limit is imposed during these iterative decoding processes.

2.8 Summary and conclusions

Multimedia codecs, such as the H.264 and H.265 video codecs, often employ source codes like the EG code, which are designed for encoding symbol values that are selected from a set having an infinite cardinality, such as the set of all positive integers. However, following the application of these source codes, typically some residual redundancy is retained, which can impose up to 1.59 dB of capacity loss and hence prevents near-capacity operation, as discussed in Section 2.6. Previously,

JSCCs have been proposed for exploiting all the residual redundancy in order to avoid any potential capacity loss. However, existing JSCCs are impractical for symbol values that are selected from a set having an infinite cardinality, since a distinct codeword is required for every legitimate symbol value. This motivates the novel UEC code proposed in this chapter.

In Section 2.2, we detailed how to collect the video data and model its symbol value probability distribution using the Zeta probability distribution.

In Section 2.3, we introduced the proposed UEC encoder, detailing the operation of both the unary encoder and of a trellis encoder. Furthermore, we described how to integrate the UEC encoder into a transmitter by introducing the operation of the concatenated URC encoder, interleaver and modulator of Figure 2.2.

In Section 2.4, we introduced the wireless channel model, namely the uncorrelated narrow-band Rayleigh fading channel. In Section 2.5, we described the integration of the UEC decoder into a receiver by discussing the operation of the demodulator, deinterleaver and URC decoder. In addition, we detailed the operation of the UEC trellis decoder, its iterative decoding process and the unary decoder.

In Section 2.6, we discussed the near-capacity operation of the UEC and its parameterization, including the appropriate selection of the number of trellis states and the codebook design.

In Section 2.7, we introduced an SSCC EG-CC-URC benchmarker. The performance of this benchmarker was compared to that of the proposed UEC scheme for the case of Zeta distributed source symbols having different p_1 parameters.

Our proposed UEC code [1] was the first JSCC that has a low decoding complexity, when invoked of representing symbols values that are selected from an alphabet having a large or infinite cardinality. However, the UEC code is based on the unary code [92], which is not a *universal* code, hence resulting in a rate R^o of zero for $p_1 \leq 0.608$,. Owing to this, the UEC code has limited applicability, since it only has a guaranteed finite average codeword length for particular source distributions, including only a limited subset of the Zeta probability distributions that does not include the specific Zeta distribution that most closely models the symbols produced by the H.264 or H.265 codec. Motivated by this, the next chapter will propose the EGEC code [110], which is based on the *universal* EG source code [38].

Chapter 3

Elias Gamma Error Correction Codes

3.1 Introduction

In this chapter, we propose a *universal* Joint Source and Channel Code (JSCC) for the near-capacity transmission of infinite-cardinality symbol alphabets that are randomly selected using *any* arbitrary monotonic probability distribution. As highlighted in Figure 3.1, this chapter addresses the source coding and outer code design aspects of the JSCC. We commence by introducing the background and motivation of this research, before discussing the novel contributions of the chapter, as well as its structure.

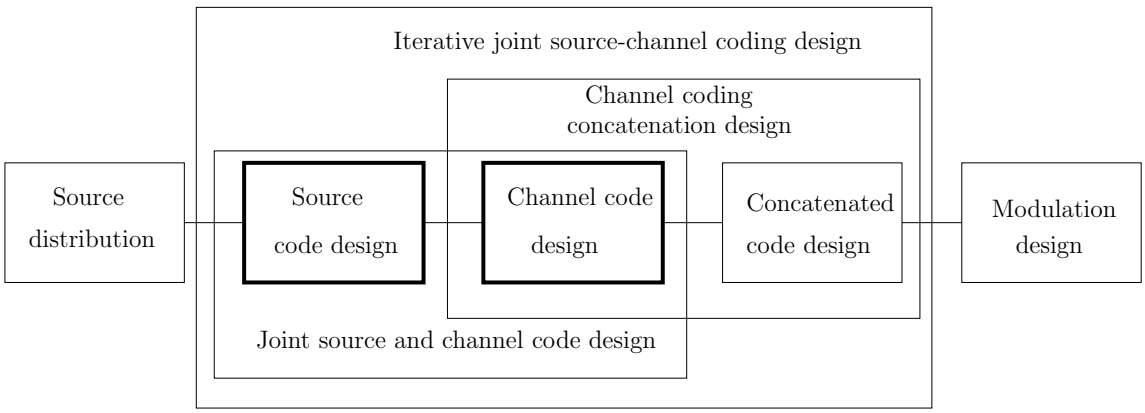


Figure 3.1: The component that must be considered when designing an iterative JSCC having serial concatenation. This chapter addresses the design aspects highlighted using the bold boxes.

3.1.1 Background and motivation

As we discussed in Section 2.1.1, following the encoding process of multimedia codecs, such as H.264 [111] and H.265 [5], typically some residual redundancy is retained in the encoded data, which leads to a capacity loss. The Unary Error Correction (UEC)

code introduced in Chapter 2 [1] was the first JSCC to exploit the residual redundancy in order to eliminate the capacity loss, while achieving a low decoding complexity, when invoked for representing symbols values that are selected from an alphabet having a large or infinite cardinality. However, the UEC code is based on the unary code [92], which is not a universal code[1]. This limits the employment of the UEC to specific situations, where the symbol values obey particular probability distributions, only including a limited subset of the Zeta probability distributions. Therefore, the employment of the UEC code is prevented in the case of arbitrary probability distributions, since its average codeword length may become infinite in these cases. Furthermore, some Zeta probability distributions that are supported by the UEC code may be outperformed by their Separate Source and Channel Code (SSCC) benchmarker, even though the latter imposes a capacity loss [95]. This is exemplified by the Zeta distribution having $p_1 = 0.6970$, when the UEC in outperformed by the EG-CC benchmarker, as described in Section 2.7.

Against this background, in this chapter we propose a *universal* JSCC for the near-capacity transmission of infinite-cardinality symbol alphabets that are randomly selected using *any* arbitrary monotonic probability distribution. As benefit of this, the proposed JSCC has a much wider applicability than the UEC of Chapter 2, facilitating its employment for the entire set of Zeta probability distributions. Like the UEC code of Chapter 2, the proposed JSCC does not require any knowledge of the symbol occurrence probabilities at either the transmitter or the receiver, when the channel's Signal to Noise Ratio (SNR) is sufficiently high. However, once the receiver has estimated the occurrence probabilities of the most frequently occurring symbol values, reliable communication at near-capacity SNRs is facilitated. Rather than employing a unary code as its basis, the proposed code is based upon the universal Elias Gamma (EG) code, hence we refer to it as the Elias Gamma Error Correction (EGEC) code.

3.1.2 Novel contribution

- A *universal* JSCC EGEC is designed, which is capable of achieving the near-capacity transmission of symbols that are randomly selected from infinite-cardinality symbol alphabets using *any* arbitrary monotonic probability distribution.

[1] A universal code is a countably infinite prefix code set. When encoding a symbol set following any monotonic probability distribution, the average codeword length is bounded by a function of the entropy of the distribution [38].

- We propose a Unequal Error Protection (UEP) scheme for optimizing the relative contribution of the two EGEC sub-codes to the encoding process, facilitating near-capacity operation at a low decoder complexity.
- The performance of the proposed JSCC EGEC scheme is compared to that of the JSCC UEC and SSCC EG-CC benchmarkers in five scenarios, namely for four different Zeta distribution parametrizations and the H.265 distribution of Figure 2.3(b).

3.1.3 Chapter organization

The rest of this chapter is organised as follows:

- As described in Section 3.2, the EGEC encoder decomposes each input symbol into two sub-symbols, which are encoded separately by two distinct sub-encoders. The first sub-encoder is referred to as the EGEC(UEC) encoder, which operates in the same manner as the UEC encoder of [1]. The second sub-encoder employs a serial concatenation of a Fixed Length Code (FLC) and a Convolutional Code (CC) encoders, which we refer to as the EGEC(FLC-CC) encoder.
- As described in Section 3.3, the EGEC decoder has corresponding sub-decoders, which operate on the basis of the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm [50] and the Soft Bit Source Decoding (SBSD) algorithm [68].
- In Section 3.4, we detail the procedure to design a UEP scheme for optimizing the relative contribution of the two sub-codes to the encoding process, facilitating near-capacity operation at a low decoder complexity. Furthermore, the parametrizations of our EGEC scheme as well as the benchmarkers are introduced.
- We will demonstrate in Section 3.5 that if the source symbols obey a particular Zeta probability distribution, our EGEC scheme offers a 3.4 dB gain over a UEC benchmarker, when Quaternary Phase Shift Keying (QPSK) is employed for transmission over an uncorrelated narrowband Rayleigh fading channel. For another Zeta probability distribution, our EGEC scheme will be shown to offer a 1.9 dB gain over a SSCC benchmarker, which we refer to as the Elias Gamma and Convolutional Code (EG-CC) scheme. Additionally, we will consider a wide range of other Zeta probability distributions and will show that our EGEC scheme is capable of offering gains over the relevant benchmarkers in each case.
- Finally, we offer our conclusions in Section 3.6.

3.2 EGEC encoder

In this section, we introduce the EGEC encoder, which is illustrated in Figure 3.2. In Section 3.2.1, we discuss the motivation for decomposing the input symbols into two sub-symbols and describe the operation of the corresponding symbol splitter in Figure 3.2, which is labeled S. The operation of the EGEC(UEC) and EGEC(FLC-CC) encoders is described in Sections 3.2.2 and 3.2.3. Finally, Section 3.2.4 describes the serial concatenation of the EGEC encoder with the Unity Rate Convolutional (URC) encoder and QPSK modulator of Figure 3.2.

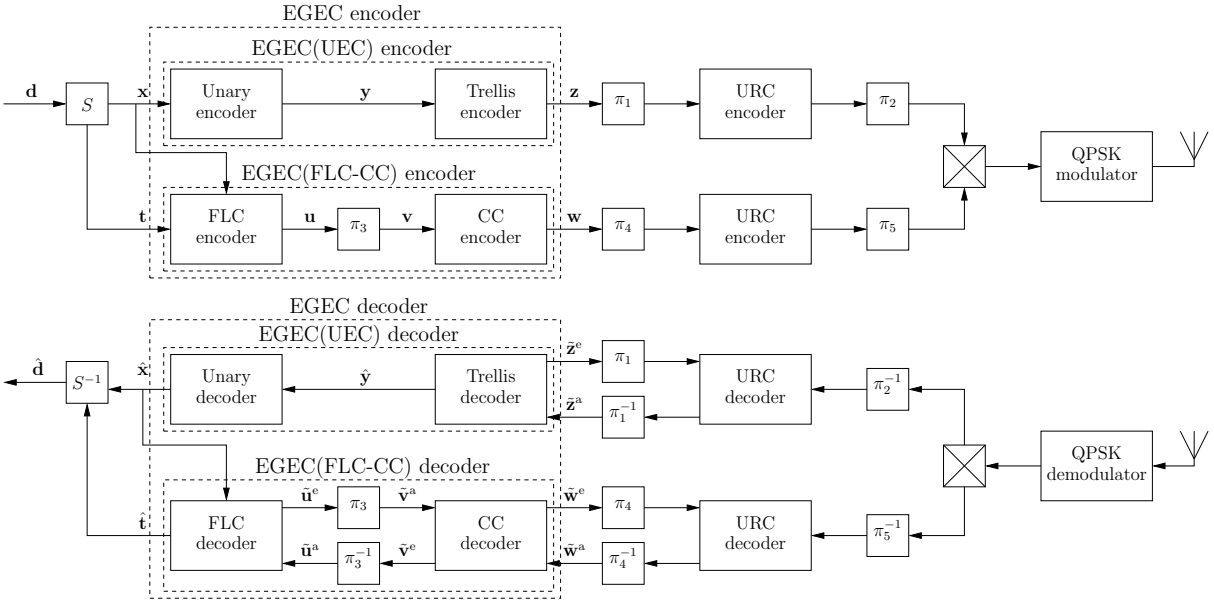


Figure 3.2: Schematic of the EGEC code, when serially concatenated with URC and Gray-coded QPSK modulation schemes. Bold notation without a diacritic is used to denote a symbol vector or a bit vector. A diacritical hat represents a reconstruction of the symbol or bit vector having the corresponding notation. A diacritical tilde represents an LLR vector pertaining to the bit vector with the corresponding notation. A roman superscript 'a' is employed to denote an *a priori* LLR vector, while 'e' is employed for extrinsic LLR vectors. Furthermore, $\{\pi_1, \ldots, \pi_5\}$ represent interleavers, while $\{\pi_1^{-1}, \ldots, \pi_5^{-1}\}$ represent the corresponding deinterleavers. Puncturing may also be performed in π_2 and π_5, while the corresponding depuncturing operations take place in π_2^{-1} and π_5^{-1}. Multiplexing and demultiplexing is performed in the crossed boxes.

3.2.1 Decomposition of symbols into pairs of sub-symbols

As shown in Figure 3.2, the EGEC encoder is designed for representing a vector $\mathbf{d} = [d_i]_{i=1}^{a}$ comprising a number of symbols, which can be obtained as a realization of a corresponding vector $\mathbf{D} = [D_i]_{i=1}^{a}$ comprising a number of Independent and Identically Distributed (IID) Random Variables (RVs). Each RV D_i adopts the symbol

Table 3.1: The first twelve codewords of various source codes.

d_i	Unary(d_i)	EG(d_i)	x_i	t_i	Unary(x_i)	FLC(t_i,x_i-1)
1	1	1	1	0	1	
2	01	010	2	0	01	0
3	001	011	2	1	01	1
4	0001	00100	3	0	001	00
5	00001	00101	3	1	001	01
6	000001	00110	3	2	001	10
7	0000001	00111	3	3	001	11
8	00000001	0001000	4	0	0001	000
9	000000001	0001001	4	1	0001	001
10	0000000001	0001010	4	2	0001	010
11	00000000001	0001011	4	3	0001	011
12	000000000001	0001100	4	4	0001	100
⋮	⋮	⋮	⋮	⋮	⋮	⋮

value $d \in \mathbb{N}_1$ with probability $\Pr(D_i = d) = P(d)$, where $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$ is the infinite-cardinality source set comprising all positive integers.

In this chapter, we focus our attention on symbol values that are randomly selected from a Zeta distribution [93], since the parameters of multimedia codecs typically obey Zipf's law, as discussed in Section 2.2.1 [1]. Here, the Zeta distribution defined in Section 2.2.2 is repeated for convenience, as

$$P(d) = \frac{d^{-s}}{\zeta(s)}, \tag{3.1}$$

where $\zeta(s)=\sum_{d\in\mathbb{N}_1} d^{-s}$ is the Riemann Zeta function and $s > 1$. In this case, $p_1 = \Pr(D_i = 1) = 1/\zeta(s)$ and the symbol entropy is given by

$$H_D = \sum_{d\in\mathbb{N}_1} H[P(d)] = \frac{\ln[\zeta(s)]}{\ln(2)} - \frac{s\zeta'(s)}{\ln(2)\zeta(s)}, \tag{3.2}$$

where we have $H[p] = p\log_2(1/p)$ and $\zeta'(s) = -\sum_{d\in\mathbb{N}_1} \ln(d)d^{-s}$ is the derivative of the Riemann Zeta function.

As shown in Table 3.1 , source encoders such as unary or EG encoders represent each symbol d_i in the vector $\mathbf{d}$ using a corresponding binary codeword, namely Unary(d) or EG(d), respectively. Note that for the convenience of our ensuing discussions, we will revisit several of the equations introduced in Chapter 2. The average codeword length is given by

$$l = \sum_{d\in\mathbb{N}_1} P(d)l(d), \tag{3.3}$$

where $l(d)$ is the length of the d^{th} codeword.

In the case of a unary code, the length of the codeword Unary(d) is given by $l(d) = d$, yielding an average codeword length of

$$l_{\text{Unary}} = \frac{\zeta(s-1)}{\zeta(s)}, \tag{3.4}$$

when the source symbols obey the Zeta distribution of (3.1). However, the average unary codeword length l is only finite for $s > 2$ and hence for $p_1 > 0.608$. Despite this, we proposed a JSCC scheme based on the unary code in Chapter 2, since its codewords have a relatively simple structure, which can be readily exploited for error correction. More specifically, the structure of the unary codewords can be described by the UEC trellis of Section 2.3.1, without requiring an infinite number of trellis transitions and states.

By contrast, an EG codeword EG(d) has a length of $l(d) = 2\lfloor \log_2(d) \rfloor + 1$. When the source symbols obey the Zeta distribution, the average codeword length becomes

$$l_{\text{EG}} = 1 - \frac{2\zeta'(s)}{\ln(2)\zeta(s)} - \frac{2}{\zeta(s)} \sum_{x \in \mathbb{N}_1} x^{-s} \text{frac}[\log_2(x)], \tag{3.5}$$

where the frac($\cdot$) operator yields the fractional part of the operand, as in frac(3.4) = 0.4. Note that the average EG codeword length l_{EG} is finite for all Zeta distributions, not only for those for which we have $p_1 > 0.608$.

In this chapter, we develop a trellis representation of the EG code by observing that the codeword EG(d) is prefixed by a unary codeword Unary(x), where we have [38]

$$x_i = \lfloor \log_2(d_i) \rfloor + 1, \tag{3.6}$$

as may be observed in Table 3.1. Furthermore, the length of the EG codeword's remaining suffix FLC(t) depends on the selected unary codeword Unary(x). More specifically, the suffix FLC(t) comprises $(x-1)$ bits, which form the binary representation of the decimal value t_i, where

$$t_i = d_i - 2^{\lfloor \log_2(d_i) \rfloor}. \tag{3.7}$$

This approach is motivated by the difference in the structures of the unary and EG codewords shown in Table 3.1. In Chapter 2, a UEC code was designed for the joint source and channel coding of unary-encoded symbols, in order to facilitate near-capacity communication. This is achieved by designing the UEC trellis of Section

2.3.2 for ensuring that the path through the trellis remains synchronised with the unary codewords. More specifically, the UEC trellis uses the logical 1-valued bit at the end of each codeword to detect the boundary between consecutive codewords and to trigger a return to state 1 or 2. By contrast, maintaining trellis synchronisation during the JSCC of EG-coded symbols is more complicated. This is because the length of the EG codeword depends on the length of its unary prefix, which may be detected using the logical 1-valued bit at the end. However, an EGEC trellis designed for maintaining synchronisation with the EG codewords would require states, i.e. memory for storing the length of the unary prefix until the end of the FLC suffix is reached, whereupon a return to state 1 or 2 could be triggered. Since the unary prefix can have any arbitrary length selected from an infinite set, an infinite number of states would be required for storing this information, hence preventing the construction of a practical trellis. Instead, we can maintain synchronisation by decomposing the sequence of EG codeword into separate sequences of unary prefixes and FLC suffixes, supporting the separate UEC encoding and FLC-CC encoding of the two sequences, hence facilitating near-capacity joint source and channel coding.

Inspired by this, the splitter S of Figure 3.2 decomposes each symbol d_i in the vector $\mathbf{d}$ into two sub-symbols, namely into x_i and t_i according to (3.6) and (3.7), where

$$d_i = 2^{x_i - 1} + t_i. \tag{3.8}$$

Each set of sub-symbols is concatenated to form the vectors $\mathbf{x} = \{x_i\}_{i=1}^{a}$ and $\mathbf{t} = \{t_i\}_{i=1}^{a}$. For example, the vector $\mathbf{d} = [6, 15, 1, 17, 2, 1, 1, 2]$ of $a = 8$ symbols yields the vector $\mathbf{x} = [3, 4, 1, 5, 2, 1, 1, 2]$ of $a = 8$ sub-symbols and the vector $\mathbf{t} = [2, 7, 0, 1, 0, 0, 0, 0]$ comprising $a = 8$ sub-symbols.

shown in Figure 3.2, each sub-symbol x_i in the vector $\mathbf{x}$ is encoded by the EGEC(UEC) encoder of Section 3.2.2, while each sub-symbol t_i in the vector $\mathbf{t}$ is encoded by the EGEC(FLC-CC) encoder of Section 3.2.3, in order to produce the codewords Unary(x) and FLC(t), as exemplified in Table 3.1, respectively. Note that since the codewords Unary(x) and FLC(t) collectively comprise the same number of bits as the codeword EG(d), the proposed EGEC code produces the same number of unary- and FLC-encoded bits as are produced by an EG encoder. Therefore, since an EG code is a universal code, so too is the proposed EGEC code, granting it a finite average codeword length when the symbol values are selected according to any monotonic probability distribution.

3.2.2 EGEC(UEC) encoder

As shown in Figure 3.2, the vector $\mathbf{x} = [x_i]_{i=1}^{a}$ is encoded by the EGEC(UEC) encoder, which operates in the same way as the UEC encoder of Section 2.3. The vector of sub-symbols $\mathbf{x}$ can be modeled as a realization of a vector of RVs $\mathbf{X} = [X_i]_{i=1}^{a}$, where each RV X_i adopts a symbol value from the set $x \in \mathbb{N}_1$ with a probability of $\Pr(X_i = x) = P(x)$. In the scenario where the RV D_i obeys the Zeta distribution of (3.1), the RV X_i will obey the distribution

$$P(x) = \frac{1}{\zeta(s)} \sum_{d=2^{x-1}}^{2^x-1} d^{-s}, \tag{3.9}$$

where the entropy of the RV X_i is given by

$$H_X = \log_2[\zeta(s)] - \frac{1}{\zeta(s)} \sum_{x \in \mathbb{N}_1} \left(\sum_{d=2^{x-1}}^{2^x-1} d^{-s} \right) \log_2 \left(\sum_{d=2^{x-1}}^{2^x-1} d^{-s} \right), \tag{3.10}$$

The sub-symbol vector $\mathbf{x}$ is forwarded to the unary encoder of Figure 3.2, which represents each symbol x_i in the vector using the corresponding x_i-bit unary codeword Unary(x) of Table 3.1. When the sub-symbols in the vector $\mathbf{x}$ obey the Zeta distribution of (3.9), the average unary codeword length is given by $l_1 = l_{\text{Unary}}$, where we have

$$\begin{aligned} l_1 &= \sum_{x \in \mathbb{N}} P(x) \cdot x \\ &= 1 - \frac{\zeta'(s)}{\ln(2)\zeta(s)} - \frac{1}{\zeta(s)} \sum_{x \in \mathbb{N}_1} x^{-s} \text{frac}[\log_2(x)], \end{aligned} \tag{3.11}$$

which is guaranteed to be finite, regardless of the value of $s > 1$. Note that (3.11) may be derived from (3.5) by observing that $l_{\text{Unary}(x_i)} = (l_{\text{EG}(d_i)} + 1)/2$, as shown in Table 3.1. Following this, the unary codewords are concatenated for generating the b-bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$ of Figure 3.2. For example, the vector $\mathbf{x} = [3, 4, 1, 5, 2, 1, 1, 2]$ of $a = 8$ sub-symbols is represented by the vector $\mathbf{y} = 0010001100001011101$ of $b = 19$ bits.

As shown in Figure 3.2, the vector of concatenated unary codewords $\mathbf{y}$ is input to the trellis encoder of Section 2.3.2. This operates on the basis of a UEC trellis, such as the $r_1 = 4$-state trellis that is exemplified in Figure 3.3. This represents a special case of the generalized r_1-state UEC trellis in Figure 2.4. Each bit y_j of the input

bit sequence $\mathbf{y} = [y_j]_{j=1}^{b}$ forces the trellis encoder to traverse from its previous state $m_{j-1} \in \{1, 2, \ldots, r_1\}$ to its next state $m_j \in \{1, 2, \ldots, r_1\}$, in order of increasing bit-index j. Each next state m_j is selected from two legitimate alternatives, depending on the bit value y_j, according to

$$m_j = \begin{cases} 1 + \text{odd}(m_{j-1}) & \text{if } y_j = 1 \\ \min[m_{j-1} + 2, r_1 - \text{odd}(m_{j-1})] & \text{if } y_j = 0 \end{cases}, \tag{3.12}$$

where the number of possible states r_1 has to be even and the encoding process always begins from the state $m_0 = 1$. The function odd$(\cdot)$ yields 1, if the operand is odd or 0, if it is even. In this way, the bit vector $\mathbf{y}$ identifies a path through the trellis, which may be represented by a vector $\mathbf{m} = [m_j]_{j=0}^{b}$ comprising $(b + 1)$ state values. For example, the bit vector $\mathbf{y} = [0010001100001011101]$ yields the path $\mathbf{m} = [1, 3, 3, 2, 4, 4, 4, 1, 2, 4, 4, 4, 4, 1, 3, 2, 1, 2, 4, 1]$ through the $r_1 = 4$-state trellis of Figure 3.3. Following this, the trellis encoder represents each bit y_j in the vector $\mathbf{y}$ by an n_1-bit codeword $\mathbf{z}_j$. This is selected from the set of $r_1/2$ codewords $\mathbb{C} = [\mathbf{c}_1; \mathbf{c}_2; \ldots; \mathbf{c}_{r_1/2}]$ or from the complementary set $\overline{\mathbb{C}} = \{\overline{\mathbf{c}_1}; \overline{\mathbf{c}_2}; \ldots; \overline{\mathbf{c}_{r_1/2}}\}$, which is achieved according to

$$\mathbf{z}_j = \begin{cases} \overline{\mathbf{c}_{\lceil m_{j-1}/2 \rceil}} & \text{if } y_j \neq \text{odd}(m_{j-1}) \\ \mathbf{c}_{\lceil m_{j-1}/2 \rceil} & \text{if } y_j = \text{odd}(m_{j-1}) \end{cases}. \tag{3.13}$$

Following this, the selected codewords are concatenated to obtain the bn_1-bit vector $\mathbf{z} = [z_k]_{k=1}^{bn_1}$ of Figure 3.2. For example, the vector $\mathbf{y} = [0010001100001011101]$ of $b = 19$ bits is represented by the vector $\mathbf{z} = [1110010001011000000101011011011000010]$ of $bn_1 = 38$ bits, when employing the UEC trellis of Figure 3.3, which has $r_1 = 4$ states and the $n_1 = 2$-bit codewords $\mathbb{C} = [00; 01]$. Note that the selection of the parameter r_1 is discussed in Section 3.4.

Note that UEC trellis encoder operates in a similar manner to a CC encoder, but with some important differences, as follows.

1. The UEC trellis encoder is specifically designed for maintaining synchronization with the unary codewords that are concatenated to form the bit vector $\mathbf{y}$. More specifically, the last bit y_j in each unary codeword Unary(x_i) is guaranteed to induce a transition in the state $m_j = 1$ or state $m_j = 2$, depending on whether the corresponding symbol x_i has an odd or even index i. This is exploited by the UEC trellis decoder in order to mitigate capacity loss, as described in Section 3.3.1. By contrast, in a generalized CC encoder, the last bit in each unary codeword can potentially cause a transition into any state, preventing

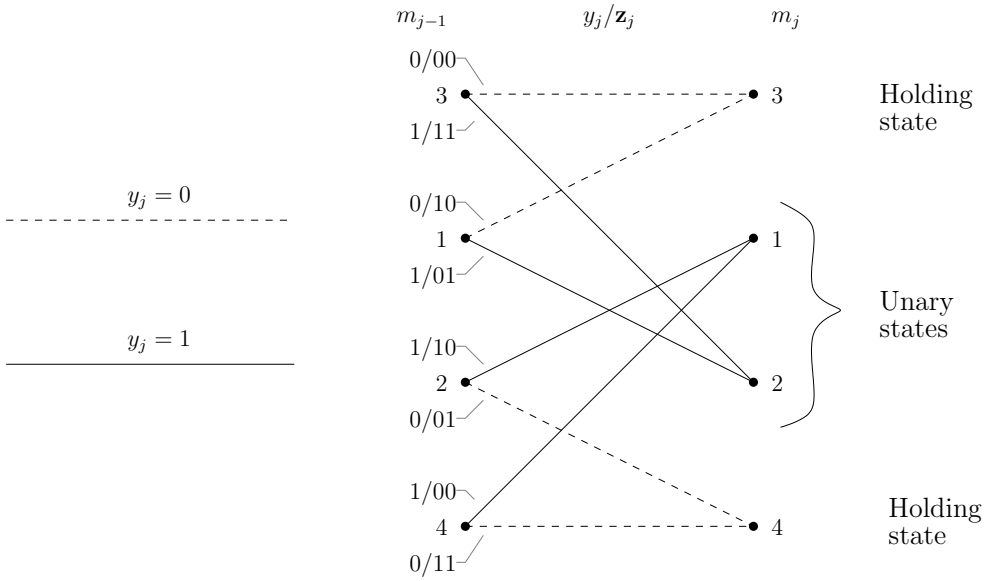


Figure 3.3: An $r_1 = 4$-state $n_1 = 2$-bit UEC trellis, where $\mathbb{C} = [00; 01]$.

synchronization.

2. The unary encoded bit vector $\mathbf{y}$ is guaranteed to terminate the UEC trellis into state $m_b = 1$ or state $m_b = 2$, depending on whether the length a of the symbol vector $\mathbf{x}$ is odd or even. This may be exploited by the UEC trellis decoder in order to assist its operation, as described in Section 3.3.1. By contrast, a generalized CC encoder is not terminated by the unary encoded bit vector $\mathbf{y}$.
3. The UEC trellis is designed to obey symmetry and to rely on complementary codewords, so that the binary values in the vector $\mathbf{z}$ are equiprobable. As described in [1], this is a necessary condition for avoiding capacity loss. By contrast, CC encoders produce binary values that are not guaranteed to be equiprobable, unless they are specifically parametrized for this purpose.

Since the binary values in the vector $\mathbf{z}$ are equiprobable, the average coding rate of the EGEC(UEC) encoder is given by

$$R_1^{\mathrm{o}} = \frac{H_X}{l_1 n_1}. \tag{3.14}$$

Here, we employ the roman superscript 'o' to indicate that this coding rate relates to the outer code of a serial concatenation, namely the EGEC(UEC) code shown in Figure 3.2.

3.2.3 EGEC(FLC-CC) encoder

As shown in Figure 3.2, the EGEC(FLC-CC) encoder requires both $\mathbf{t}$ and $\mathbf{x}$ vectors, in order to perform FLC encoding. As described in Section 3.2.1, this is because each

sub-symbol t_i in the vector $\mathbf{t}$ is mapped to an FLC codeword FLC(t_i), having the length $(x_i - 1)$, where x_i is the corresponding sub-symbol in the vector $\mathbf{x}$. When the sub-symbols in the vector $\mathbf{x}$ obey the distribution of (3.9), the average FLC codeword length is given by $l_2 = l_{\mathrm{FLC}(t_i)}$, where we have

$$\begin{aligned} l_2 &= \sum_{x\in\mathbb{N}} P(x)\cdot(x-1) \\ &= -\frac{\zeta'(s)}{\ln(2)\zeta(s)} - \frac{1}{\zeta(s)}\sum_{x\in\mathbb{N}_1} x^{-s}\mathrm{frac}[\log_2(x)], \end{aligned} \tag{3.15}$$

which is guaranteed to be finite, regardless of the value of $s > 1$.

to the dependencies between $\mathbf{t}$ and $\mathbf{x}$, we model the sub-symbol vector $\mathbf{t}$ as a realization of a RV vector $\mathbf{T} = [T_i]_{i=1}^{a}$, where each RV T_i is dependent on the corresponding RV X_i. By considering (3.1) and (3.8), the joint probability $\Pr(T_i = t, X_i = x) = P(t, x)$ is given by

$$P(t,x) = \frac{1}{\zeta(s)}(2^{x-1}+t)^{-s}, \tag{3.16}$$

where $0 \le t < 2^{x-1}$. Furthermore, the conditional probability $\Pr(T_i = t|X_i = x) = P(t|x)$ is given by

$$P(t|x) = \frac{P(t,x)}{P(x)} = \frac{(2^{x-1}+t)^{-s}}{\sum\limits_{d=2^{x-1}}^{2^x-1} d^{-s}}, \tag{3.17}$$

where $0 \le t < 2^{x-1}$. Finally, the conditional entropy of the RV T_i is given by

$$H_{T|X} = \sum_{x\in\mathbb{N}_1}\sum_{t=0}^{2^{x-1}-1} P(t,x)\log_2\left(\frac{1}{P(t|x)}\right). \tag{3.18}$$

The codewords FLC(t_i) are concatenated to obtain the $(b-a)$-bit vector $\mathbf{u} = [u_e]_{e=1}^{b-a}$ of Figure 3.2. the example of $\mathbf{x} = [3,4,1,5,2,1,1,2]$ and $\mathbf{t} = [2,7,0,1,0,0,0,0]$, the combination $x_1 = 3$ and $t_1 = 2$ yields the codeword FLC(t_1) = 10, as shown in Table 3.1. Similarly, $x_2 = 4$ and $t_2 = 7$ yields FLC(t_2) = 111, while $x_3 = 1$ and $t_3 = 0$ yields an empty bit vector for FLC(t_3). Completing this encoding process and concatenating the resultant codewords yields the vector $\mathbf{u}$ = [10111000100] of $(b-a) = 11$ bits. The bit vector $\mathbf{u}$ is interleaved in the block π_3 of Figure 3.2, in order to obtain the vector $\mathbf{v} = [v_e]_{e=1}^{b-a}$. Note that the binary values in the vectors $\mathbf{u}$ and $\mathbf{v}$ will not be equiprobable in general.

This motivates the employment of the r_2-state n_2-bit recursive CC encoders of Table 2.5, since these produce equiprobable binary values for the encoded bit vector

$\mathbf{w} = [w_f]_{f=1}^{n_2(b-a)}$ of Figure 3.2. This is necessary because producing equiprobable binary values is a necessary condition for avoiding capacity loss [1]. For example, if the $r_2 = 4$-state $n_2 = 2$-bit recursive CC encoder of Table 2.5 is employed for encoding the $(b-a) = 11$-bit vector $\mathbf{v} = [00101100110]$, the $n_2(b-a) = 22$-bit vector of $\mathbf{w} = [0000111000101011001010]$ is generated. The average coding rate of the EGEC(FLC-CC) encoder is given by

$$R_2^{\text{o}} = \frac{H_{T|X}}{l_2 n_2}. \tag{3.19}$$

3.2.4 Integration of the EGEC encoder into a transmitter

Following EGEC encoding, the bit vectors $\mathbf{z}$ and $\mathbf{w}$ are interleaved by π_1 and π_4, URC encoded using accumulators [87] and then interleaved again by π_2 and π_5, as shown in Figure 3.2. These URC encoders are recursive and have a coding rate of unity, satisfying the corresponding conditions that are sufficient for facilitating near-capacity operation [112, 113]. Alternatively, Low Density Parity Check (LDPC) or turbo codes may be employed for this purpose, although this implies a significantly increased complexity. This is because turbo decoders employ the iterative operation of two component decoders, while LDPC decoders employ the iterative operation of variable nodes and check nodes. By contrast, URC codes comprise only a single component, requiring no internal iterations. Note that the lengths required for these interleavers and URC codes depend on the particular sub-symbol values in the vector $\mathbf{x}$. However, these components can glean the required lengths from the lengths of their respective input bit vectors. Puncturing may also be performed within π_2 and π_5, in order to achieve the desired throughput for the transmitter, as well as for UEP, as discussed in Section 3.4. While interleaving and URC encoding are associated with one input bit per output bit, puncturing within π_2 and π_5 are respectively associated with coding rates of $R_1^{\text{i}} \geq 1$ and $R_2^{\text{i}} \geq 1$ input bits per output bit. Here, we employ the roman superscript 'i' to indicate that these coding rates relate to the inner codes of serial concatenations, namely the two URC codes shown in Figure 3.2. Following URC encoding, the multiplexer of Figure 3.2 appends the encoded bit sequence derived from the EGEC(FLC-CC) encoder onto the end of that derived from the EGEC(UEC) encoder. Following this, $M = 4$-ary Gray-coded QPSK modulation may be employed for transmission, as shown in Figure 3.2. Note that other mapping schemes or a modulation scheme having a higher order M can be employed instead, although this may increase the complexity of the receiver, as

we will discuss in Section 3.4. The throughput of the transmitter is given by

$$\eta = \frac{H_D \log_2(M)}{l_1 n_1 / R_1^{\mathrm{i}} + l_2 n_2 / R_2^{\mathrm{i}}}. \tag{3.20}$$

3.3 EGEC decoder

In this section, we describe the operation of the EGEC decoder of Figure 3.2. The EGEC(UEC) decoder and EGEC(FLC-CC) decoder are described in Sections 3.3.1 and 3.3.2, respectively. Following this, Section 3.3.3 discusses the serial concatenation of the EGEC decoder with the URC decoder and QPSK demodulator of Figure 3.2.

3.3.1 EGEC(UEC) decoder

As shown in Figure 3.2, the EGEC(UEC) decoder's trellis decoder is provided with a vector of *a priori* Logarithmic Likelihood Ratios (LLRs) $\tilde{\mathbf{z}}^{\mathrm{a}} = [\tilde{z}_k^{\mathrm{a}}]_{k=1}^{bn_1}$ that pertain to the corresponding bits in the vector $\mathbf{z}$. The trellis decoder operates on the basis of the Log-BCJR algorithm of Section 2.5.2. This generates the vector of extrinsic LLRs $\tilde{\mathbf{z}}^{\mathrm{e}} = [\tilde{z}_k^{\mathrm{e}}]_{k=1}^{bn_1}$, which is provided for the next iteration of the concatenated URC decoder's operation. Following the completion of iterative decoding, the trellis decoder may also be employed to generate the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}^{\mathrm{p}} = [\tilde{y}_j^{\mathrm{p}}]_{j=1}^{b}$ that pertain to the corresponding bits in the vector $\mathbf{y}$. The unary decoder of Figure 3.2 sorts the values in this LLR vector in order to identify the a number of bits in the vector $\mathbf{y}$ that are most likely to have values of one. A hard decision vector $\hat{\mathbf{y}} = [\hat{y}_j]_{j=1}^{b}$ is then obtained by setting the value of these a bits to one and the value of all other bits to zero. Here, the value of a is assumed to be perfectly known to the receiver and may be reliably conveyed by the transmitter using a small amount of side information in practice. Finally, the bit vector $\hat{\mathbf{y}}$ can be unary decoded in order to generate the sub-symbol vector $\hat{\mathbf{x}} = [\hat{x}_i]_{i=1}^{a}$ of Figure 3.2, which is guaranteed to comprise a number of sub-symbols. This has the benefit of mitigating, although not totally eliminating, the error propagation that may occur owing to the variable lengths of the unary codewords.

Note that the trellis decoder's Log-BCJR algorithm has only a modest complexity, since it may employ a low number r_1 of states. Furthermore, it facilitates error correction even if the sub-symbol probability distribution $P(x)$ is unknown, provided that the channel SNR is sufficiently high, as we shall demonstrate in Section 3.5. However, once a sufficient number of sub-symbol vectors $\hat{\mathbf{x}}$ has been recovered, the average unary codeword length l_1 and the sub-symbol probabilities $P(x)$ for $x \leq r_1/2-1$ may

be heuristically estimated. When decoding subsequent sub-symbol vectors, this information may be exploited by the Log-BCJR algorithm as illustrated Section 2.5.2.1 in order to facilitate error correction at near-capacity SNRs as discussed in Section 2.6.1. Note that this is made possible by the termination of the UEC trellis and its synchronization with the unary codewords, as described in Section 3.2.2. Also note that the availability of l_1 and $P(x)$ for $x \leq r_1/2 - 1$ is assumed throughout the remainder of this chapter, unless explicitly stated otherwise.

The transformation of $\tilde{\mathbf{z}}^{\mathrm{a}}$ into $\tilde{\mathbf{z}}^{\mathrm{e}}$ may be characterized by plotting the inverted EGEC(UEC) EXtrinsic Information Transfer (EXIT) curve in an EXIT chart [51], as exemplified in Figure 3.4. Note that if UEC codewords comprising at least $n_1 = 2$ bits are employed, then the free distance d_{free} of the UEC code will be at least two, and its EXIT curve will reach the $(1, 1)$ point at the top right corner of the EXIT chart [71]. Reaching this point is important because in this case a vanishingly low Bit Error Ratio (BER) may be attained.

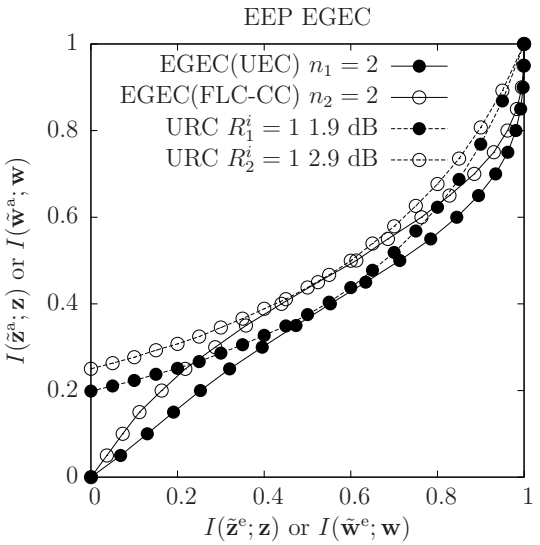


Figure 3.4: EXIT charts of the EEP EGEC scheme. Here, the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$ and the codewords comprise the numbers of bits n_1 and n_2. Furthermore, the punctured URC decoders adopt the coding rates R_1^{i} and R_1^{i}, for Gray-coded QPSK modulation onto an uncorrelated narrowband Rayleigh fading channel having various E_{b}/N_0. The EXIT curves are provided for an EGEC(UEC) code having $r_1 = 4$ states, as well as for an EGEC(FLC-CC) code having $x_{\max} = 3$.

The EXIT chart area $A_1^{\rm o}$ that is situated below the inverted EGEC(UEC) EXIT curve of the demodulator is given by

$$\begin{aligned} A_1^{\rm o} &= \tfrac{1}{l_1 n_1} \textstyle\sum_{x=1}^{\frac{r_1}{2}-1} H[P(x)] + \tfrac{2}{l_1 n_1} H\left[1 - \sum_{x=1}^{\frac{r_1}{2}-1} P(x)\right] \\ &+ \tfrac{1}{l_1 n_1} H\left[l_1 - \tfrac{r_1}{2} + \textstyle\sum_{x=1}^{r_1/2-1} P(x)\left(\tfrac{r_1}{2} - x\right)\right] \\ &- \tfrac{1}{l_1 n_1} H\left[l_1 + 1 - \tfrac{r_1}{2} + \textstyle\sum_{x=1}^{r_1/2-1} P(x)\left(\tfrac{r_1}{2} - 1 - x\right)\right]. \end{aligned} \tag{3.21}$$

as described in Section 2.5.2.2.

3.3.2 EGEC(FLC-CC) decoder

As shown in Figure 3.2, the EGEC(FLC-CC) decoder iteratively exchanges extrinsic information between the CC decoder and the FLC decoder. More specifically, the n_2-bit r_2-state CC decoder employs the Log-BCJR algorithm for transforming the *a priori* LLR vectors $\tilde{\mathbf{v}}^{\rm a} = [\tilde{v}_e^{\rm a}]_{e=1}^{b-a}$ and $\tilde{\mathbf{w}}^{\rm a} = [\tilde{w}_f^{\rm a}]_{f=1}^{n_2(b-a)}$ into the extrinsic LLR vectors $\tilde{\mathbf{v}}^{\rm e} = [\tilde{v}_e^{\rm e}]_{e=1}^{b-a}$ and $\tilde{\mathbf{w}}^{\rm e} = [\tilde{w}_f^{\rm e}]_{f=1}^{n_2(b-a)}$. Here, the extrinsic LLR vector $\tilde{\mathbf{w}}^{\rm e}$ is provided for the next iteration of the concatenated URC decoder's operation. Meanwhile, the extrinsic LLRs of $\tilde{\mathbf{v}}^{\rm e}$ are deinterleaved π_3^{-1}, in order to obtain the *a priori* LLR vector $\tilde{\mathbf{u}}^{\rm a} = [\tilde{u}_e^{\rm a}]_{e=1}^{b-a}$. As shown in Figure 3.2, the *a priori* LLR vector $\tilde{\mathbf{u}}^{\rm a}$ is then forwarded to the FLC decoder, together with the sub-symbol vector $\hat{\mathbf{x}}$, which is provided by the EGEC(UEC) decoder. The sub-symbols of $\hat{\mathbf{x}}$ are employed for partitioning the *a priori* LLR vector $\tilde{\mathbf{u}}^{\rm a}$ into sub-vectors, where the $i^{\rm th}$ sub-vector comprises $(\hat{x}_i - 1)$ bits. Note that since $\hat{\mathbf{x}}$ is guaranteed to contain a number of sub-symbols, the sum of the sub-vector lengths is given by $\sum_{i=1}^{a}(\hat{x}_i - 1)$, which is guaranteed to be equal to the length $(b-a)$ of the LLR vector $\tilde{\mathbf{u}}^{\rm a}$. The FLC decoder employs the SBSD algorithm of [68] to generate the vector of extrinsic LLRs $\tilde{\mathbf{u}}^{\rm e} = [\tilde{u}_e^{\rm e}]_{e=1}^{b-a}$. This is then interleaved in the block π_3 of Figure 3.2, in order to obtain the *a priori* LLR vector $\tilde{\mathbf{v}}^{\rm a}$ for the next iteration of the CC decoder's operation. Following the completion of iterative decoding, the FLC decoder may also be employed to generate the vector of a sub-symbols $\hat{\mathbf{t}} = [\hat{t}_i]_{i=1}^{a}$, as shown in Figure 3.2.

Note that the FLC decoder can recover the sub-symbol vector $\hat{\mathbf{t}}$ even if the conditional probabilities of (3.17) are unknown, provided that the channel SNR is sufficiently high, as we shall demonstrate in Section 3.5. However, once a sufficient number of sub-symbol vectors $\hat{\mathbf{t}}$ has been recovered, the conditional probabilities $P(t|x)$ can be heuristically estimated for all pairs of t and x where $x \leq x_{\max}$. When decoding subsequent sub-symbol vectors, this information may be exploited by the SBSD algorithm in order to facilitate error correction at near-capacity SNRs as discussed in Section 2.6.1. More specifically, the SBSD algorithm can apply the conditional

probabilities of (3.17) to each of the a sub-vectors of $\tilde{\mathbf{u}}^{\mathrm{a}}$ for which the corresponding sub-symbols in $\hat{\mathbf{x}}$ do not exceed $x_{\max}$. This improves the reconstruction of the corresponding sub-symbols in $\hat{\mathbf{t}}$, as well as providing extrinsic information for the corresponding LLRs in $\tilde{\mathbf{u}}^{\mathrm{e}}$. Note that zero values are adopted by the LLRs in $\tilde{\mathbf{u}}^{\mathrm{e}}$ which correspond to sub-symbols in $\hat{\mathbf{x}}$ that do exceed $x_{\max}$. Also note that the availability of the conditional probabilities $P(t|x)$ for $x \leq x_{\max}$ is assumed throughout the remainder of this paper, unless explicitly stated otherwise.

The transformation of $\tilde{\mathbf{w}}^{\mathrm{a}}$ into $\tilde{\mathbf{w}}^{\mathrm{e}}$ using the iterative operation of the CC and FLC decoders may be characterized by plotting the inverted EGEC(FLC-CC) EXIT curve in an EXIT chart [51]. This is exemplified in Figure 3.4 for the scenario where the CC and FLC decoders are operated, until iterative decoding convergence is achieved. Note that if CC codewords comprising at least $n_2 = 2$ bits are employed, then the free distance d_{free} of the CC code will be at least two, and the EGEC(FLC-CC) EXIT curve will reach the $(1, 1)$ point at the top right corner of the EXIT chart [71]. The EXIT chart area that is situated below the inverted EGEC(FLC-CC) EXIT curve is given by

$$\begin{aligned} A_2^{\mathrm{o}} &= \frac{1}{n_2} \sum_{x=2}^{x_{\max}} \sum_{t=0}^{2^{x-1}-1} \frac{H[P(t|x)]}{x-1} \frac{(x-1)P(x)}{l_2} \\ &+ \frac{1}{n_2} \left(1 - \sum_{x=2}^{x_{\max}} \sum_{t=0}^{2^{x-1}-1} \frac{(x-1)P(x)}{l_2}\right). \end{aligned} \tag{3.22}$$

3.3.3 Integration of EGEC decoder into a receiver

At the receiver of Figure 3.2, QPSK demodulation, demultiplexing, depuncturing as well as deinterleaving, URC decoding as well as deinterleaving and is performed before invoking the EGEC(UEC) and EGEC(FLC-CC) decoders. Note that the receiver is required to employ the same pseudo-random interleaver designs as the transmitter. However, the entire set of interleavers can be generated independently by both the transmitter and receiver using only a single pseudo-random number generator seed. This seed may be hard-coded into both the transmitter or receiver, or may be reliably conveyed using only a very small amount of side information. As shown in Figure 3.2, it is necessary to operate the EGEC(UEC) decoder before the EGEC(FLC-CC) decoder, since the former outputs the sub-symbol vector $\hat{\mathbf{x}}$, which is required as an input to the latter. The extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ of Figure 3.2 may be iteratively exchanged with the serially concatenated URC decoder. In-turn, the URC decoder may also iteratively exchange extrinsic LLRs with the demodulator [114], in order to avoid capacity loss, when a mapping scheme other than Gray coding or a higher-order modulation scheme is employed. Since the combination of the URC decoder and the demodulator will also have an EXIT curve that reaches the $(1, 1)$ point at the top right corner of the EXIT chart [106], iterative decoder convergence towards

an approximation of the Maximum Likelihood (ML) performance is facilitated [107]. After completing the EGEC(UEC) decoding process, a similar iterative operation is performed for the EGEC(FLC-CC) decoder.

Following the completion of both the EGEC(UEC) and the EGEC(FLC-CC) iterative decoding processes, the sub-symbol vectors $\hat{\mathbf{x}}$ and $\hat{\mathbf{t}}$ are input to the sub-symbol recombination block S^{-1} of Figure 3.2, which recomposes the symbol vector $\hat{\mathbf{d}}$, according to (3.8). Observe that since the sub-symbol vectors $\hat{\mathbf{x}}$ and $\hat{\mathbf{t}}$ are guaranteed to comprise a number of symbols, the symbol vector $\hat{\mathbf{d}}$ will also comprise a number of symbols.

3.4 Near-capacity performance of EGEC codes and UEP design

Near-capacity operation is achieved, when reliable communication can be maintained at transmission throughputs η that approach the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity C that is associated with $M = 4$ QPSK modulation and uncorrelated narrowband Rayleigh fading. This is facilitated, if the following conditions are satisfied [112]:

1. The area A_1^{o} beneath the inverted EGEC(UEC) EXIT curve is required to approach the corresponding coding rate R_1^{o};
2. Likewise, the area A_2^{o} beneath the inverted EGEC(FLC-CC) EXIT curve is required to approach the corresponding coding rate R_2^{o};
3. The URC EXIT curves are required to satisfy $A_1^{\text{i}} = C/[R_1^{\text{i}} \log_2(M)]$ and $A_2^{\text{i}} = C/[R_2^{\text{i}} \log_2(M)]$.

If these three conditions are satisfied, then near-capacity operation will be achieved when the shape of URC decoders' EXIT curves are matched to those of the EGEC(UEC) and EGEC(FLC-CC) decoders. This creates narrow, but marginally open EXIT chart tunnels, which facilitate iterative decoding convergence towards an approximation of the ML performance.

When the RVs in the vector $\mathbf{X}$ obey the Zeta distribution of (3.9), Figure 3.5 suggests that the first two of the above-mentioned conditions are satisfied. This figure plots the EGEC(UEC) coding rate R_1^{o} of (3.14) and the EGEC(FLC-CC) coding rate R_2^{o} of (3.19), when multiplied with the codeword lengths n_1 and n_2, respectively. Note that since the proposed EGEC code is a universal code, its coding rates are non-zero, regardless of the parameter value $0 < p_1 < 1$ employed for the Zeta distribution of (3.1). Furthermore, Figure 3.5 plots the product of n_1 and the EGEC(UEC) EXIT area A_1^{o} of (3.21), for the case where the trellis decoder employs

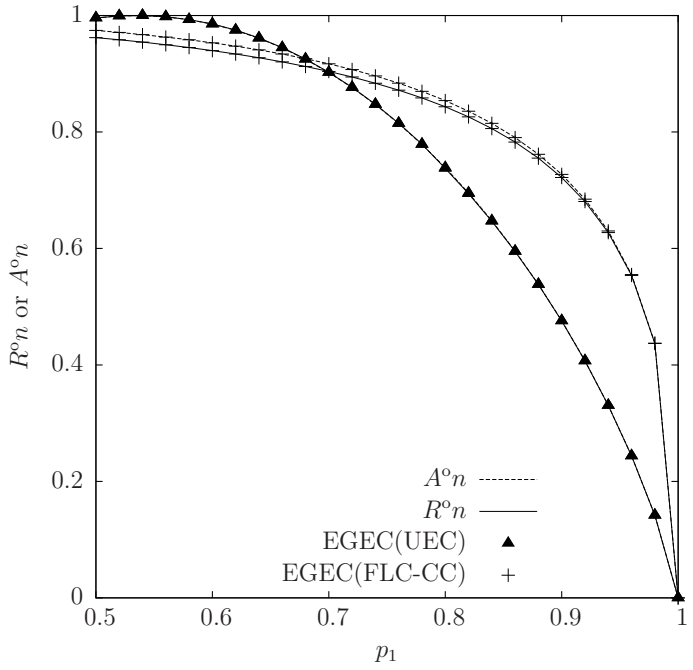


Figure 3.5: Plots of $R^{o}n$ and $A^{o}n$ that are obtained for the EGEC scheme, in the case where the system input symbol values of $\mathbf{d}$ obey a Zeta distribution having the parameter p_1. Here, R^{o} is the coding rate, A^{o} is the area beneath the inverted EXIT curve and n is the codeword length of the corresponding scheme. The value of $A^{o}n$ is provided for an EGEC(UEC) code having $r_1 = 4$ states, as well as for an EGEC(FLC-CC) code having $x_{\max} = 3$.

$r_1 = 4$ states. Likewise, the product of n_2 and the EGEC(FLC-CC) EXIT area A_2^{o} of (3.22) is plotted in Figure 3.5, for the case where $x_{\max} = 3$ is employed. Figure 3.5 shows that in all cases, the EXIT chart areas A_1^{o} and A_2^{o} approach the corresponding coding rates R_1^{o} and R_2^{o}. Figure 3.6 plots the discrepancy between $A_1^{o}n_1$ and $R_1^{o}n_1$ as a function of the number of EGEC(UEC) states r_1, where the source symbols of $\mathbf{d}$ obey the Zeta distribution of (3.1) for various values for the parameter p_1. Note that in all the scenarios considered, the discrepancy becomes less than 10^{-3} for $r_1 = 4$, demonstrating that the EGEC(UEC) code imposes only an insignificant amount of capacity loss. Therefore, $r_1 = 4$ represents an attractive trade-off between facilitating near-capacity operation and maintaining a low trellis complexity. Similarly, the discrepancy between $A_2^{o}n_2$ and $R_2^{o}n_2$ is plotted for the EGEC(FLC-CC) code as a function of $x_{\max}$ in Figure 3.7, for Zeta distributions having various values for the parameter p_1. Note that in all the scenarios considered, the discrepancy is around 10^{-2} for $x_{\max} = 3$, demonstrating that the EGEC(FLC-CC) code imposes only an insignificant amount of capacity loss. Therefore, $x_{\max} = 3$ represents an attractive trade-off between facilitating near-capacity operation and maintaining a low computational complexity. Note that unlike the EGEC(UEC), the EGEC(FLC-CC) EXIT chart area A_2^{o} is independent of the number of states r_2

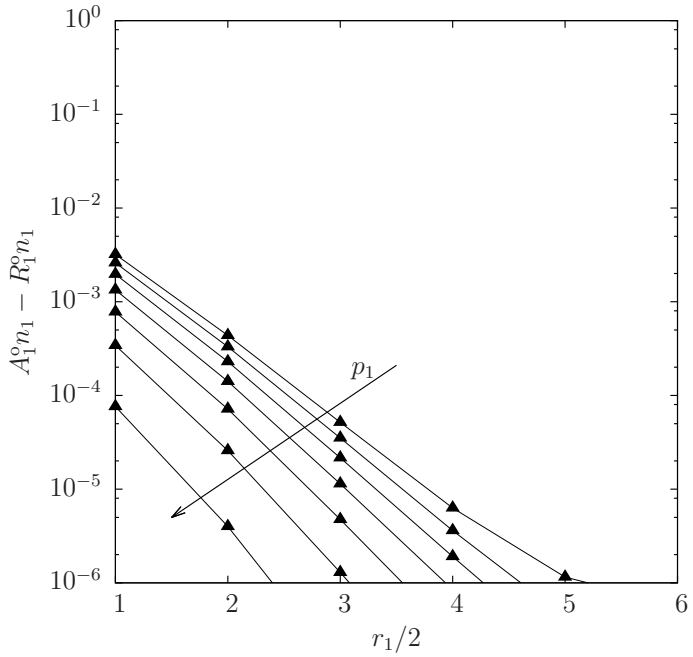


Figure 3.6: The discrepancy between $A_1^o n_1$ and $R_1^o n_1$ that results when EGEC(UEC) codes having various numbers of states r_1 are employed to encode symbol values having Zeta distributions with the parameters $p_1 \in \{0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$.

employed in the CC. Finally, the third condition is satisfied by a punctured URC code, as discussed in [112].

Figure 3.5 shows that the areas beneath the EGEC(UEC) and EGEC(FLC-CC) EXIT curves A_1^o and A_2^o are different from each other in general. Owing to this, different areas are required beneath the URC EXIT curves A_1^i and A_2^i, so that narrow but still open EXIT chart tunnels can be created simultaneously for both the EGEC(UEC) and EGEC(FLC-CC) codes. This can be achieved by employing different values for the coding rates R_1^i and R_2^i, hence exhibiting UEP. More specifically, when decreasing one of these coding rates, the other should be increased, in order to maintain the same throughput η and facilitate a fair comparison with the Equal Error Protection (EEP) scheme in which $R_1^i = R_2^i$.

The advantages of UEP may be illustrated by comparing Figures 3.4 and 3.8, which consider the scenario where the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$. More specifically, Figure 3.4 considers an EEP scheme having an $n_1 = 2$-bit EGEC(UEC) code and an $n_2 = 2$-bit EGEC(FLC-CC) code, as well as inner coding rates of $R_1^i = 1$ and $R_2^i = 1$, which gives a throughput of $\eta = 0.7620$ bit/s/Hz. Observe in Figure 3.4 that an open EXIT chart tunnel is created by the EGEC(UEC) code at an E_b/N_0 of 1.9 dB, but by contrast this is not facilitated until reaching an E_b/N_0 of 2.9 dB for the EGEC(FLC-CC) code.

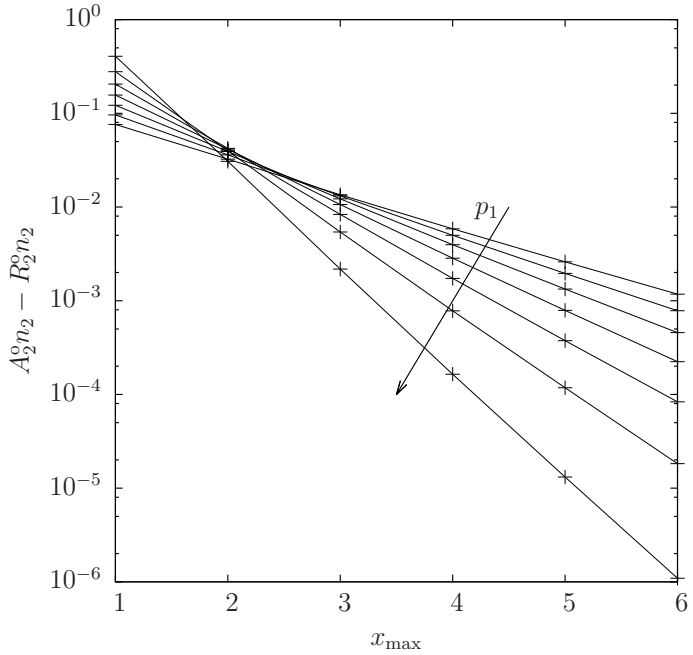


Figure 3.7: The discrepancy between $A_2^o n_2$ and $R_2^o n_2$ that results when EGEC(FLC-CC) codes having various values for the parameter $x_{\max}$ are employed to encode symbol values having Zeta distributions with the parameters $p_1 \in \{0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$.

Let us now consider an UEP scheme, which employs an $n_1 = 2$-bit EGEC(UEC) code and an $n_2 = 3$-bit EGEC(FLC-CC) code, as well as inner coding rates of $R_1^i = 1.0385$ and $R_2^i = 1.2767$, in order to achieve the same throughput of $\eta = 0.7620$ bit/s/Hz. Observe in Figure 3.8 that this scheme can simultaneously create open EXIT chart tunnels for both the EGEC(UEC) code and the EGEC(FLC-CC) code at an E_b/N_0 of 2.4 dB, offering 0.5 dB of gain over the EEP scheme.

In general, an UEP scheme can be designed by appropriately choosing n_1, n_2, R_1 and R_2 for ensuring that the desired throughput η is achieved and the two EXIT chart tunnels become marginally open at the same E_b/N_0 value. Here, we recommend the value of $n_1 = n_2 = 2$ for the codeword lengths when possible, while the value of $n_2 = 3$ whenever necessary to achieve the desired throughput η.

Table 3.2 provides parametrizations for both EEP and UEP EGEC schemes, designed for transmitting symbols that obey the Zeta distribution of (3.1). We parametrize the Zeta distribution using $p_1 \in \{0.9000, 0.7967, 0.6940, 0.6000\}$, which represents a wide selection of the p_1 values shown in Figure 3.5. Here, $p_1 = 0.7967$ is chosen for consistency with the results of Chapter 2, while $p_1 = 0.6940$ is chosen because it is the specific value, where R_1^o and R_2^o are equal to each other. Table 3.2 also considers the case of source symbols obeying the H.265 distribution of Figure 2.3(b). The EGEC(UEC) scheme adopts the $n_1 = 2$-bit $r_1 = 4$-state UEC trellis

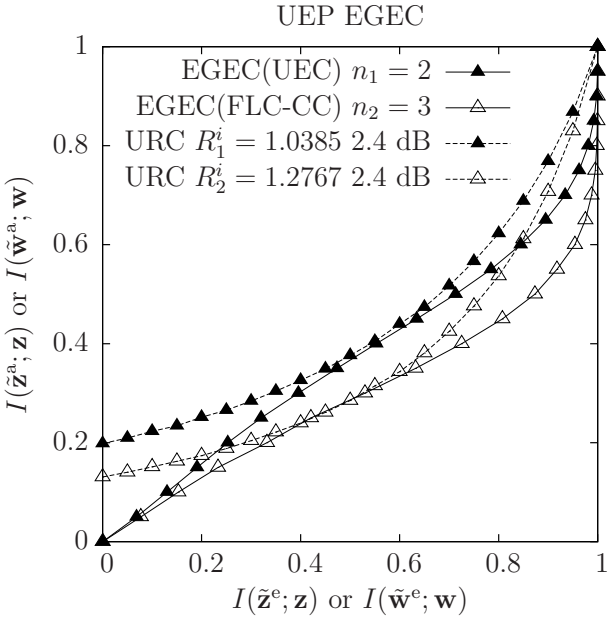


Figure 3.8: EXIT charts of the UEP EGEC scheme. Here, the symbols of $\mathbf{d}$ obey a Zeta distribution having $p_1 = 0.7967$ and the codewords comprise the numbers of bits n_1 and n_2. Furthermore, the punctured URC decoders adopt the coding rates R_1^{i} and R_1^{i}, for Gray-coded QPSK modulation onto an uncorrelated narrowband Rayleigh fading channel having various E_{b}/N_0. The EXIT curves are provided for an EGEC(UEC) code having $r_1 = 4$ states, as well as for an EGEC(FLC-CC) code having $x_{\max} = 3$.

of Figure 3.3, while the EGEC(FLC-CC) scheme adopts $x_{\max} = 3$, as well as the $r_2 = 4$-state recursive CC of Table 2.5 having either $n_2 = 2$-bit codewords or $n_2 = 3$-bit codewords, as appropriate. As discussed above, we select $r_1 = 4$ UEC states, since this is sufficiently high for imposing only an insignificant amount of capacity loss, while $r_2 = 4$ CC states were selected, because having a higher number of states was found to be detrimental in Section 2.7.1.

Table 3.2 also characterizes the complexity of the EEP and UEP EGEC schemes. Here, the complexity is quantified by the average number of Add, Compare and Select (ACS) operations performed per decoding iteration and per symbol in the vector $\mathbf{d}$. This is justified, since the EGEC(UEC) trellis decoder, the EGEC(FLC-CC) decoder and the URC decoder operate entirely on the basis of addition, subtraction and max* operations, while all other components in Figure 3.2 may be considered to have a relatively insignificant complexity [95, 68]. As we discussed in Section 2.5.1.3.2, we assume that each max* operation may be completed using five ACS operations, while the addition and subtraction operations each require a single ACS operation. As shown in Table 3.2, the complexity tends to increase as the Zeta distribution parameter p_1 is reduced, which is due to the resultant increases in the average codeword lengths l_1 and l_2. Furthermore, since the UEC EGEC schemes employ $n_2 = 3$-bit

Table 3.2: Outer coding rate R^{o}, inner coding rate R^{i} and throughput η for three schemes with different p_1 values.

p_1	Scheme			n	r	R^{o}	A^{o}	R^{i}	η	E_b/N_0 [dB] for $C=\eta$	E_b/N_0 [dB] for $A^{i}=A^{o}$	E_b/N_0 [dB] for open tunnel	Complexity
0.9	EGEC	EEP	UEC	2	4	0.2378	0.2378	1.0578	0.5272	0.01	2.4	3.9	267
			FLC-CC	2	4	0.3609	0.3636						
		UEP	UEC	2	4	0.2378	0.2378	1.1251			0.1	1.0	286
			FLC-CC	3	4	0.2406	0.2424	1					
	UEC			2	4	0.2636	0.2682	1			0.1	1.5	250
	EG-CC			2	4	0.2492	0.3247	1.0578			1.6	2.4	257
0.7967	EGEC	EEP	UEC	2	4	0.3721	0.3721	1	0.7620	0.84	1.6	2.9	338
			FLC-CC	2	4	0.4229	0.4283						
		UEP	UEC	2	4	0.3721	0.3721	1.0385			0.9	2.4	379
			FLC-CC	3	4	0.2820	0.2855	1.2767					
	UEC			2	4	0.3810	0.4041	1			1.3	2.5	331
	EG-CC			2	4	0.3810	0.4410	1			2.0	3.0	322
0.6940	EGEC	EEP	UEC	2	4	0.4533	0.4535	1	0.9066	1.43	1.5	2.5	431
			FLC-CC	2	4	0.4533	0.4599						
	UEC			2	4	0.3112	0.3654	1.4565			2.7	4.5	614
	EG-CC			2	4	0.4533	0.4877	1			2.0	3.0	410
0.6	EGEC	EEP	UEC	2	4	0.4906	0.4910	1	0.9690	1.69	1.8	2.8	547
			FLC-CC	2	4	0.4699	0.4766						
	EG-CC			2	4	0.4845	0.4998	1			2.0	3.0	522
H.265	EGEC	EEP	UEC	2	4	0.4639	0.4652	1	0.8786	1.3	1.8	2.9	588
			FLC-CC	2	4	0.3862	0.3955	1					
	UEC			2	4	0.3480	0.4249	1.2624			3.1	4.7	715
	EG-CC			2	4	0.4393	0.4961	1			2.4	3.3	558

EGEC(FLC-CC) codes, they are associated with a higher complexity than the corresponding EEP schemes, which employ shorter $n_2 = 2$-bit codes.

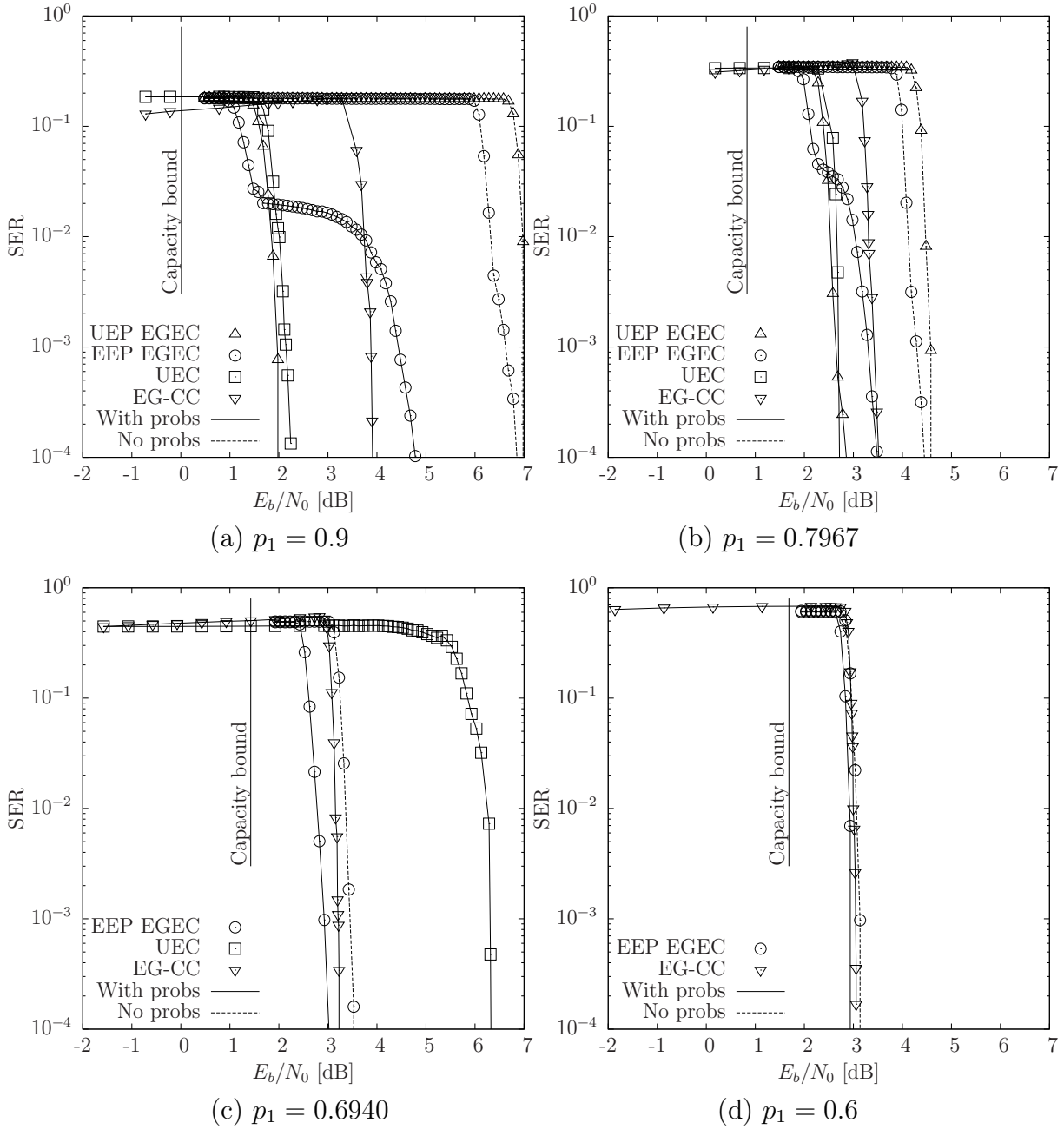
Table 3.2 provides the E_b/N_0 values, where the DCMC capacity C becomes equal to the throughput η of each scheme considered. These E_b/N_0 values represent *capacity bounds*, above which it becomes practically possible to achieve reliable communication, provided that the scheme facilitates near-capacity operation. Furthermore, the specific E_b/N_0 values, where we have $A^{i} = A^{o}$ are provided for each of the schemes considered in Table 3.2. These *area bounds* represent the lowest E_b/N_0 values, where it is theoretically possible to create an open EXIT chart tunnel, provided that the EGEC and URC EXIT functions have shapes that closely match each other. Note that the discrepancy between the capacity bound and the area bound of each scheme represents *capacity loss*. For each of the UEP schemes considered, the capacity loss is less than 0.1 dB, demonstrating that the proposed EGEC scheme facilitates near-capacity operation. Finally, Table 3.2 provides the *tunnel bound* of each scheme, which quantifies the lowest E_b/N_0 value, where an open EXIT chart tunnel can be created upon employing the two-state accumulator of Figure 2.8 for the URC code. Note that in all cases, our experiments revealed that two-state URC codes facilitate the creation of open tunnels at lower E_b/N_0 values than four- or eight-state URCs,

as well as having a lower decoding complexity. The proposed EGEC schemes facilitate reliable communication at E_b/N_0 values that exceed the corresponding tunnel bound, provided that the symbol vector $\mathbf{d}$ comprises a sufficiently high number a of symbols.

3.5 Performance comparison with the benchmarkers

In this section, we compare the proposed EEP and UEP EGEC schemes to the UEC and EG-CC benchmarkers of Chapter 2. Table 3.2 provides parametrizations for these benchmarkers, which offer the same throughput η as the proposed EGEC schemes. Here, the UEC benchmarker adopts the $n = 2$-bit $r = 4$-state UEC trellis of Figure 2.4(b), since this is recommended in Section 2.6. Furthermore, the UEC is serially-concatenated with a URC, for the sake of facilitating iterative decoding. While the proposed EGEC schemes and the UEC benchmarker constitute examples of JSCCs, the EG-CC benchmarker represents SSCC. More specifically, the EG-CC benchmarker employs an EG code for source coding, while an iteratively-decoded serial-concatenation of a CC and a URC is employed for separate channel coding. Here, we select the $n = 2$-bit $r = 4$-state CC of Table 2.5, since higher numbers of states were found to be detrimental, as discussed in Section 2.7.1. As in the proposed EGEC schemes, the UEC and EG-CC benchmarkers employ the two-state accumulator of Figure 2.8(a) for their URCs, since these were found to yield open EXIT chart tunnels at the lowest E_b/N_0 values. Note that the UEC and EG-CC benchmarkers offer fair and natural comparisons with the proposed EGEC schemes, since they all employ simple unary, FLC or EG codewords, as well as UEC or CC trellises having four states. Furthermore, EG codes and CCs are employed in numerous multimedia transmission standards, such as H.264 [4], DVB-T [115] and H.265 [5].

Figure 3.9 characterizes the Symbol Error Ratio (SER) performance of the schemes parametrized in Table 3.2. We consider the transmission of source symbol vectors $\mathbf{d}$ comprising $a = 2 \cdot 10^4$ symbols, which we found to be typical of the number of EG-encoded symbols that appear in the H.265-encoded bit stream of a slice [5]. Therefore, the SER performance of Figure 3.9 may be considered to be achievable, without imposing any additional latency in multimedia applications. We employed QPSK modulation for transmission over an uncorrelated narrowband Rayleigh fading channel, since this is representative of transmission over realistic wireless channels and because this facilitates direct comparison with the results of Chapter 2. In the receivers, iterative decoding was continued until convergence was achieved, without imposing a complexity limit.



(a) $p_1 = 0.9$ (b) $p_1 = 0.7967$

(c) $p_1 = 0.6940$ (d) $p_1 = 0.6$

Figure 3.9: The SER performance that is obtained following the achievement of iterative decoding convergence in the EGEC scheme of Figure 3.2, as well as in the UEC and EG-CC benchmarkers of Chapter 2, when transmitting frames comprising $a = 2 \cdot 10^4$ symbols using QPSK modulation over an uncorrelated narrowband Rayleigh fading channel. The plots labeled 'With probs' and 'No probs' indicate the SER performance that is achievable when the source distribution $P(d)$ is known and unknown to the receiver, respectively.

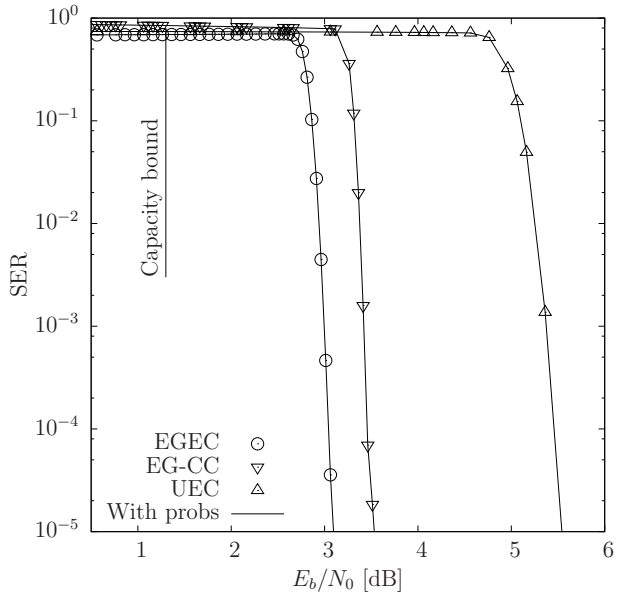


Figure 3.10: The SER performance of the EGEC scheme, as well as of the UEC and EG-CC benchmarkers of Chapter 2, when transmitting frames comprising $a = 2 \cdot 10^4$ symbols that obey the H.265 distribution, using QPSK modulation for transmission over an uncorrelated narrowband Rayleigh fading channel.

As shown in Figure 3.9, the proposed EGEC schemes facilitate reliable communication within 1.2 dB of the capacity bound and consistently offer the best SER performance for each of the p_1 values considered. This consistency is a key benefit of the proposed EGEC scheme, because while it offers only a modest gain over the best of the two benchmarkers in each case, the performance of these benchmarkers is particularly inconsistent. More explicitly, while the proposed EGEC scheme offers only a marginal gain over the UEC benchmarker for $p_1 \in \{0.9, 0.7967\}$, this gain becomes 3.4 dB for $p_1 = 0.6940$, owing to the severe puncturing that the UEC scheme requires in this case, as discussed in Section 2.7 [95]. Furthermore, the UEC benchmarker cannot be invoked for $p_1 = 0.6$, since the average unary codeword length l_{Unary} becomes infinite in this case. Similarly, while the proposed EGEC scheme offers only a marginal gain over the EG-CC benchmarker $p_1 \in \{0.6940, 0.6\}$, this gain becomes 1.9 dB for $p_1 = 0.9$ and 0.8 dB for $p_1 = 0.7967$, as shown in Figure 3.9.

In the case where the source symbols obey the H.265 distribution of Figure 2.3(b), our EGEC scheme offers a gain of 0.4 dB over the SSCC EG-CC benchmarker, as shown in Figure 3.10. Note that the EGEC scheme only employs EEP, since we found that UEP does not improve the performance of the EGEC scheme in this scenario. The UEC benchmarker has the worst performance of all the schemes considered in this scenario, owing to the severe puncturing that it requires for achieving the same effective throughput as the other schemes.

The complexity of the proposed EGEC schemes is compared to that of the benchmarkers in Table 3.2. For each source distribution considered, it can be seen that the complexity of the various schemes is similar, demonstrating that the gains offered by the proposed EGEC scheme are not accrued at the cost of a significantly increased complexity. Instead, these gains may be attributed to the EGEC scheme's avoidance of the capacity loss suffered by the EG-CC benchmarker, as well as due to avoiding the UEC scheme's requirement for excessive puncturing, in the cases of the $p_1 = 0.6940$ Zeta distribution and the H.265 distribution.

Note that the UEP EGEC schemes offer superior SER performance over the EEP schemes, as shown in Figure 3.9. The stair-case shaped SER performance of the EEP schemes may be attributed to the opening of the EGEC(UEC) EXIT chart tunnel at a lower E_b/N_0 value than the EGEC(FLC-CC) EXIT chart tunnel, as shown in Table 3.2. At E_b/N_0 values where the EGEC(UEC) EXIT chart tunnel is open, but the EGEC(FLC-CC) EXIT chart tunnel is closed, symbols having a value of $d_i = 1$ are typically correctly decoded, since these symbols are conveyed without using any EGEC(FLC-CC)-encoded bits. By contrast, symbols having values of $d_i > 1$ are typically incorrectly decoded in this case, owing to transmission errors affecting their corresponding EGEC(FLC-CC)-encoded bits.

A similar phenomenon may be observed for the EG-CC benchmarker, increasing the SER as the E_b/N_0 value is increased towards the threshold value, where the EXIT chart tunnel becomes open. More specifically, at very low E_b/N_0 values, the information received over the channel is associated with a very low confidence and hence the iterative decoding process is dominated by the *a priori* knowledge that the symbol value $d_i = 1$ is most likely. This causes a value of 1 to be selected for all symbols in the vector $\hat{\mathbf{d}}$, resulting in an SER of $(1 - p_1)$. As the E_b/N_0 value is increased, the information received over the channel gradually achieves a higher influence on the iterative decoding process, allowing values other than unity to be increasingly selected for some symbols in $\hat{\mathbf{d}}$. However, these non-unity values are typically allocated to the wrong symbols, owing to the loss of synchronization that is caused by the frequent occurrence of decoding errors at E_b/N_0 values below the open tunnel threshold. This effect occurs more frequently as the E_b/N_0 value is further increased towards the threshold value, causing the SER to increase, as shown in Figure 3.9. By contrast, for E_b/N_0 values above the threshold, the decoding errors are mitigated and the SER reduces rapidly.

Note that throughout the discussions above, it is assumed that the receiver of the proposed EGEC scheme has knowledge of the average unary codeword length l_1. Furthermore, we assume knowledge of the sub-symbol probabilities $P(x)$ for

$x \leq r_1/2 - 1$, as well as the conditional sub-symbol probabilities $P(t|x)$ for all pairs of t and x, where we have $x \leq x_{\max}$. These probabilities can be calculated with the knowledge of $P(d)$ for $d \leq 2^{\max(x_{\max}, r_1/2-1)} - 1$. Since we employ $r_1 = 4$ and $x_{\max} = 3$, the SER results presented above may be obtained with knowledge of only the first seven symbol probabilities $P(d)$, as well as of l_1. However, Figure 3.9 shows that when the channel SNR is sufficiently high, the proposed EGEC receiver facilitates a low SER, even if it does not have access to this information. This may be exploited to recover a sufficiently high number of symbol vectors $\hat{\mathbf{d}}$, in order to heuristically estimate the small amount of required information, hence facilitating the near-capacity transmission of the subsequent symbol vectors.

3.6 Summary and Conclusions

In this chapter we have proposed novel EGEC codes for the near-capacity transmission of symbol values that are randomly selected from a source symbol set having an infinite cardinality. In contrast to the UEC code previously proposed for the same purpose, our EGEC code is a universal code, facilitating the transmission of symbol values that are randomly selected using any mototonic probability distribution.

As described in Section 3.2, the EGEC encoder decomposes each input symbol into two sub-symbols, which are encoded separately by two distinct sub-encoders, as shown in Figure 3.2. The first sub-encoder is referred to as the EGEC(UEC) encoder, which operates in the same manner as the UEC encoder of Chapter 2. The second sub-encoder employs a serial concatenation of a FLC and a CC encoders, which we refer to as the EGEC(FLC-CC) encoder.

As seen in Figure 3.2 of Section 3.3, the EGEC decoder has corresponding sub-decoders, which operate on the basis of the Log-BCJR algorithm [50] and the SBSD algorithm [68].

In Section 3.4, we detailed the procedure of designing a UEP scheme that optimizes the relative contribution of the two sub-codes to the encoding process, facilitating near-capacity operation at a low decoder complexity. Furthermore, the parametrizations of both our EGEC scheme as well as of the benchmarkers were introduced.

We demonstrated in Section 3.5 that when the source symbols obey a particular Zeta probability distribution, our EGEC scheme was shown to offer a 3.4 dB gain over a UEC benchmarker, when QPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel. In the case of another Zeta probability distribution, our EGEC scheme was shown to offer a 1.9 dB gain over

the SSCC EG-CC benchmarker. Furthermore, we considered a wide range of Zeta probability distributions and our EGEC scheme was found to offer beneficial gains over the relevant benchmarkers in each case. Additionally, we considered the case, when the source symbols obey the H.265 distribution of Figure 2.3(b), demonstrating that our EGEC scheme offers a 0.4 dB gain over the EG-CC benchmarker.

However the EGEC scheme has a complex structure and the UEP must be specifically designed for a particular source probability distribution, which prevents the application of the EGEC scheme to sources having an unknown or non-stationary probability distribution. Motivated by this, in next chapter we will introduce a *universal* JSCC having a simpler structure.

Chapter 4

Reordered Elias Gamma Error Correction Codes

4.1 Introduction

In this chapter, we propose a *universal* Joint Source and Channel Code (JSCC) having a significantly simpler structure than the EGEC code of Chapter 3. We refer to this novel code as the Reordered Elias Gamma Error Correction (REGEC) code, which is designed for the near-capacity transmission of symbols that are randomly selected either from a large or even infinite alphabet using *any* arbitrary monotonic probability distribution. As highlighted in Figure 4.1, this chapter addresses the source coding, channel coding and concatenated code design aspects of the entire JSCC design. We commence by introducing the background and motivation of this research, before discussing the novel contributions of the chapter, as well as its structure.

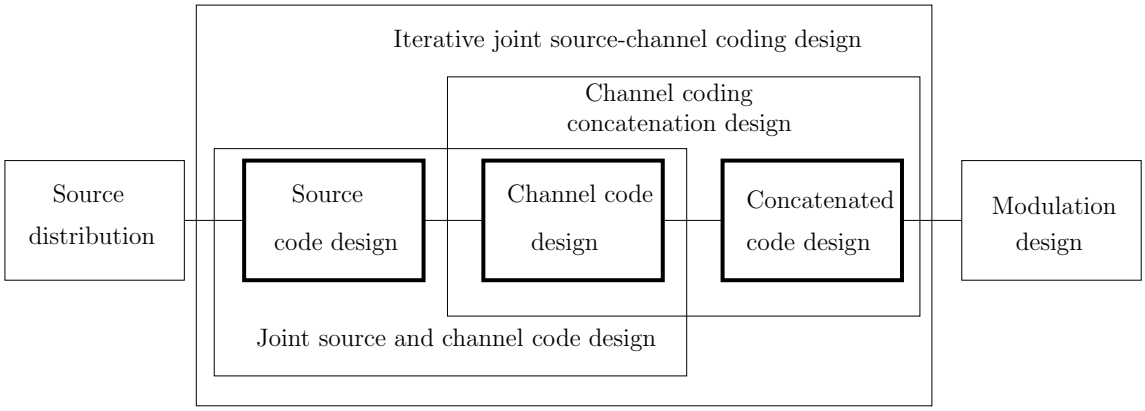


Figure 4.1: The components that must be considered, when designing an iterative JSCC relying on serial concatenation. This chapter addresses the design aspects highlighted using the bold boxes.

4.1.1 Background and motivation

As we discussed in the previous chapters, JSCCs [7] have been proposed for exploiting the residual redundancy that remains following source coding in order to enhance the attainable channel coding performance, whilst avoiding capacity loss. We previously proposed two JSCCs schemes, which were designed for the near-capacity transmission of source symbols that are randomly selected either from a large or even infinite alphabet, namely the Unary Error Correction (UEC) code [1] and the Elias Gamma Error Correction (EGEC) code [110], which are detailed in Chapters 2 and 3, respectively. More specifically, our previously proposed UEC code of Chapter 2 was the first JSCC scheme that exhibited a low decoding complexity, when employed for representing symbol values that are selected from an alphabet having a large or infinite cardinality. However, the UEC code is based on the unary code [92], which is not a *universal* code. Owing to this, the UEC code has a limited applicability, since it only has a finite average codeword length for particular source distributions, including only a limited subset of the Zeta probability distributions that does not include the particular Zeta distribution that is capable of most closely modeling the symbols produced by the H.264 and H.265 video encoders. Motivated by this, we proposed the EGEC code of Chapter 3, which is based on the *universal* Elias Gamma (EG) source code [38] and hence it was the first *universal* JSCC, allowing the low complexity and near-capacity transmission of symbol values that are randomly selected from a large or infinite alphabet using *any* arbitrary monotonic probability distribution. Since the average codeword length of the EGEC code is always finite, it has a much wider applicability than the UEC code. However the EGEC scheme has a complex structure, comprising two parts, namely the EGEC(UEC) part and the EGEC(FLC-CC) part, as shown in Figure 3.2. Meanwhile the EGEC(UEC) part operates on the basis of the UEC code of Chapter 2. The EGEC(FLC-CC) part employs a serial concatenation of a Fixed Length Code (FLC) with a Convolutional Code (CC) and relies on side information provided by the EGEC(UEC) part. Owing to this, the EGEC(FLC-CC) part cannot be operated until after the operation of the EGEC(UEC) part has been completed, which imposes additional processing delay. Furthermore, if the side information provided by the EGEC(UEC) part contains any decoding errors, it will cause the EGEC(FLC-CC) part becoming desynchronised, hence introducing a high number of decoding errors. Depending on the particular source probability distribution, the two parts of the EGEC code typically have different error correction performances, with one or other of the parts becoming the bottle-neck on the overall error connection performance. This can be solved by using

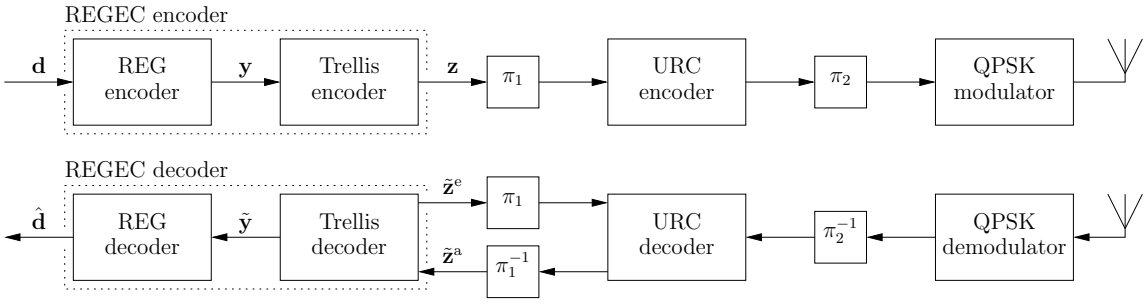


Figure 4.2: Schematic of the REGEC scheme, when serially concatenated with URC and Gray-coded QPSK modulation schemes. Bold notation without a diacritic is used to denote a symbol vector or a bit vector. A diacritical hat represents a reconstruction of the symbol or bit vector having the corresponding notation. A diacritical tilde represents an LLR vector pertaining to the bit vector with the corresponding notation. A roman superscript 'a' is employed to denote an *a priori* LLR vector, while 'e' is employed for extrinsic LLR vectors. Furthermore, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. Puncturing may also be performed in π_2, while the corresponding depuncturing operations take place in π_2^{-1}.

puncturing to introduce Unequal Error Protection (UEP) for the two parts, as discussed in Section 3.4, giving them identical error correction performances. However, the puncturing will impose some capacity loss and it will also increase the complexity of the system, since the punctured bits still have to be decoded. Furthermore, the UEP must be specially parametrized for a particular source probability distribution. If the actual source distribution is unknown or is non-stationary, then it will typically not match the distribution chosen to parametrize the UEP, hence imposing a further capacity loss.

Against this background, this chapter proposes a *universal* JSCC scheme, which we refer to as the REGEC code. This has a simple structure, which facilitates the low-complexity near-capacity transmission of symbol values that are randomly selected from large or infinite alphabets using *any* arbitrary monotonic probability distribution. Since it is a universal code, the applicability of the REGEC code is not limited to any particular source symbol distribution, like the UEC. Furthermore, since the REGEC code has a simple structure comprising only a single part, as shown in Figure 4.2, it does not suffer from the delay, loss of synchronisation, loss of capacity and increased complexity of puncturing, that are unavoidable by the EGEC scheme. Furthermore, the REGEC code is a "one size fits all" solution, since in contrast to the EGEC code it does not require UEP that is tailored for a specific source distribution. Our REGEC code is based on a novel source code, which we termed as the Reordered Elias Gamma (REG) code. This reorders the bits in each of the EG codewords in order to give them a relatively simple structure. Since this is achieved without changing the length of the codewords, the REG code is a universal

code, like the EG code. The proposed REGEC code combines the REG source code with a novel trellis-based channel code. Reordering the bits in the EG codewords allows the REGEC trellis to be designed so that the transitions between its states are synchronous with the transitions between the consecutive codewords in the REG encoded bit sequence. This allows the residual redundancy in the REG encoded-bit sequence to be exploited for error correction by the REGEC trellis decoder, hence facilitating near-capacity operation.

4.1.2 Novel contributions

The author's novel contributions that are detailed in this chapter are as follows

- The REGEC *universal* JSCC is designed, which is capable of achieving the near-capacity transmission of symbols that are randomly selected from large or infinite cardinality symbol alphabets using *any* arbitrary monotonic probability distribution.
- We propose a finite Zeta-like distribution for modeling the distribution of the symbol values produced, when test video sequences are encoded by H.265 video encoders.
- A novel REG code is proposed by reordering the bits of the EG code in order to yield a simple codeword structure, which is suitable for trellis representation.
- We introduce a REGEC trellis, which can describe the structure of our proposed REG code using a relatively small number of trellis states. Owing to this, our REGEC decoder has a low decoding complexity.
- We characterize various candidate parametrizations of the REGEC code, considering the corresponding inner code in terms of the Signal to Noise Ratio (SNR) required for facilitating iterative decoding converge to a low error probability (the open tunnel bound) and quantify how low that error probability is (the error floor).
- A wide range of finite Zeta-like distributions and the H.265 distribution of Figure 2.3(b) are considered when comparing the performance of the proposed JSCC REGEC scheme to that of the JSCC UEC and EGEC schemes as well as to the conventional Separate Source and Channel Code (SSCC) based EG-CC benchmarker.

4.1.3 Chapter organization

The rest of this chapter is organised as follows:

- In Section 4.2, we describe the Zeta source probability distribution and we generalise the infinite cardinality source alphabet of our previous work to the case

of a finite cardinality, where this cardinality represents an additional parameter to be considered.

- In Section 4.3, we introduce the novel REG code and describe the structure of the REG codewords.
- Section 4.4 and 4.5 introduce our novel REGEC encoder and decoder, repectively.
- In Section 4.6, we analyze the parametrization of the proposed REGEC scheme and demonstrate that it facilitates near-capacity operation.
- In Section 4.7, we will consider a wide range of finite Zeta-like probability distributions as well as the H.265 distribution and we will show that our REGEC scheme is capable of offering gains over the best of the UEC, EGEC and SSCC benchmarkers in each case, when employing Quaternary Phase Shift Keying (QPSK) for communication over an uncorrelated narrowband Rayleigh fading channel.
- Finally, we offer our conclusions in Section 4.8.

4.2 Symbol value sets having a large cardinality

The schemes considered in this chapter are designed for conveying a vector $\mathbf{d} = [d_i]_{i=1}^{a}$ comprising a symbols. This symbol-vector is obtained as the realization of a corresponding vector $\mathbf{D} = [D_i]_{i=1}^{a}$ of Independent and Identically Distributed (IID) Random Variables (RVs). Each RV D_i adopts the symbol value $d \in \mathbb{N}_L$ with probability $\Pr(D_i = d) = P(d)$, where $\mathbb{N}_L = \{1, 2, 3, \ldots, L\}$ is the finite-cardinality alphabet comprising positive integers with the cardinality L. Our previous contributions in Chapters 2 and 3 characterized the performance of the UEC, EGEC and EG-CC schemes invoked for representing symbols values that are selected from a set having an infinite cardinality. Instead, in this chapter we will use the symbol set $\mathbb{N}_L$ having the finite cardinality of $L = 1000$, since the symbol values produced by H.264 and H.265 are selected from alphabets having finite cardinalities of approximately 1000 as shown in Figure 2.3. Here, the symbol entropy is given by $H_D = \sum_{d \in \mathbb{N}_L} H[P(d)]$, where $H[p] = p \log_2(1/p)$. Figure 4.3 illustrates the proposed finite Zeta-like distribution having different parametrization as well as the H.265 distribution of Figure 2.3(b). Here, we define the finite Zeta-like distribution as

$$P(d) = \frac{d^{-s}}{H_L^{(s)}}, \tag{4.1}$$

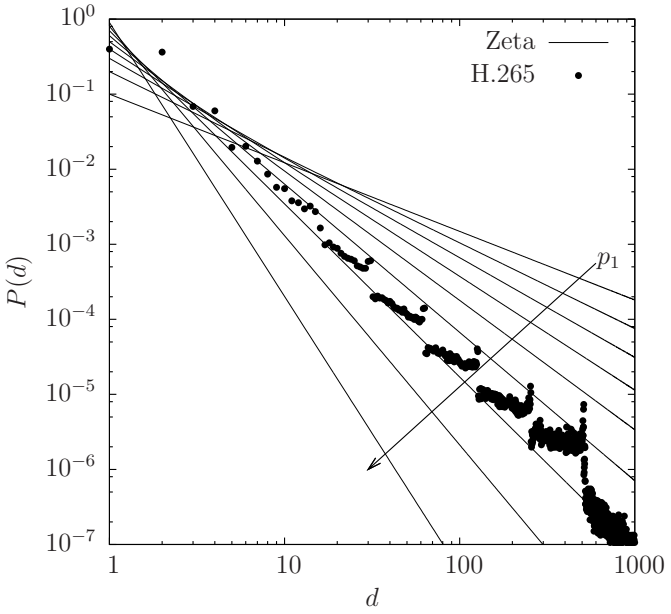


Figure 4.3: The finite Zeta-like probability distribution for $L = 1000$ and $p_1 \in \{0.1, 0.2, \ldots, 0.7, 0.8, 0.9\}$ which have the corresponding entropies of $H_D \in \{8.0297, 6.5140, 5.2299, 4.1314, 3.1974, 2.4085, 1.7388, 1.1541, 0.6118\}$, as well as the H.265 distribution, which has the entropy $H_D = 2.3922$. This was obtained by recording the values of the 112.9 million symbols produced when the HM-9.0 H.265 video encoder employs the 'encoder_lowdelay_main.cfg' and 'encoder_random_access_main.cfg' configurations to encode the 220 s of video that are comprised by 4:2:0 versions of the 24 video test sequences that are commonly used for testing in H.265 [8, page 94].

where $H_L^{(s)}=\sum_{d\in\mathbb{N}_L} d^{-s}$ is the generalized harmonic number[1] of order L of s, where $s \in \mathbb{R}$ for a finite L. The limit of $L \to \infty$ exists when $s > 1$ and the generalized harmonic number converges to the Riemann Zeta funtion[2]. The finite Zeta-like distribution may be more conveniently parameterized by the probability of the symbols adapting the most likely value of 1, which is given by $p_1 = \Pr(D_i = 1) = 1/H_L^{(s)}$. In the case of the finite Zeta-like distribution, the symbol entropy is given by

$$H_D = \sum_{d\in\mathbb{N}_L} H[P(d)] = \frac{\ln(H_L^{(s)})}{\ln(2)} - \frac{s(\partial H_L^{(s)}/\partial s)}{\ln(2) H_L^{(s)}}, \tag{4.2}$$

where $\partial H_L^{(s)}/\partial s = -\sum_{d\in\mathbb{N}_L} \ln(d) d^{-s}$ is the derivative of the harmonic number with respect to s.

[1]Note the difference between the notation $H_\bullet$ of the entropy and the notation $H_\bullet^{(\bullet)}$ of generalized harmonic number.

[2]When $L \to \infty$, we have $\lim_{L\to\infty} H_L^{(s)} = \zeta(s)$, where $\zeta(s)=\sum_{d\in\mathbb{N}_1} d^{-s}$ is the Riemann Zeta function and $\mathbb{N}_1 = \{1, 2, 3, \ldots \infty\}$ is the infinite-cardinality set comprising positive integers.

Table 4.1: The first twelve codewords of various source codes

d_i	Unary(d_i)	EG(d_i)	x_i	t_i	Unary(x_i)	FLC(t_i,x_i-1)	REG(d_i)
1	1	1	1	0	1		1
2	01	010	2	0	01	0	001
3	001	011	2	1	01	1	011
4	0001	00100	3	0	001	00	00001
5	00001	00101	3	1	001	01	00011
6	000001	00110	3	2	001	10	01001
7	0000001	00111	3	3	001	11	01011
8	00000001	0001000	4	0	0001	000	0000001
9	000000001	0001001	4	1	0001	001	0000011
10	0000000001	0001010	4	2	0001	010	0001001
11	00000000001	0001011	4	3	0001	011	0001011
12	000000000001	0001100	4	4	0001	100	0100001
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

4.3 Reordered Elias Gamma code

As shown in Table 4.1, source encoders such as the unary or EG encoders represent each symbol d_i in the vector $\mathbf{d}$ using a corresponding binary codeword, namely Unary(d_i) or EG(d_i), respectively. Note that for the convenience of our ensuing discussions, the unary codewords shown in Table 4.1 are the complements of those that are conventionally employed, for example in [1, Table I]. The average codeword length is given by

$$l = \sum_{d \in \mathbb{N}_L} P(d) l(d), \tag{4.3}$$

where $l(d)$ is the length of the d^{th} codeword.

In the case of a unary code, the length of the codeword Unary(d_i) is given by $l_{Unary}(d_i) = d_i$, giving an average codeword length of

$$l_{\text{Unary}} = \frac{H_L^{(s-1)}}{H_L^{(s)}}, \tag{4.4}$$

when the source symbols obey the finite Zeta-like distribution of (3.1). However, the average unary codeword length l is only finite for $s > 2$ and hence for $p_1 > 0.608$ when L tends to infinity. For the case of the finite Zeta-like distribution having the cardinality $L = 1000$, the average codeword length of the unary code is almost double that of the EG code when $p_1 = 0.608$, as we will characterise below. Despite this, the unary code was used as the basis of the JSCC UEC scheme of Figure 4.2(c) [1], since its codewords have a relatively simple structure, which can be readily exploited for error correction. More specifically, the structure of the unary codewords can be described by the UEC trellis of [1], without requiring an excessive number of trellis transitions and states.

By contrast, an EG codeword EG(d_i) has a length of $l_{EG}(d_i) = 2\lfloor \log_2(d_i) \rfloor + 1$. When the source symbols obey the finite Zeta-like distribution, the average EG

codeword length becomes [110]

$$l_{\mathrm{EG}} = 1 - \frac{2(\partial H_L^{(s)}/\partial s)}{\ln(2) H_L^{(s)}} - \frac{2}{H_L^{(s)}} \sum_{d \in \mathbb{N}_L} d^{-s} \mathrm{frac}(\log_2(d)), \tag{4.5}$$

where the frac$(\cdot)$ operator yields the fractional part of the operand, where frac$(3.4) = 0.4$ for example [110]. Note that the average EG codeword length l is finite for all Zeta distributions as $L \to \infty$, not just for those for which we have $p_1 > 0.608$. For the case of $L = 1000$, the average EG codeword length is lower than that of the unary code for all cases where $p_1 < 0.794$. However, the conventional EG codewords have a relatively complicated structure, which cannot be readily described by a single trellis and hence cannot be readily exploited for low-complexity error correction using a simple JSCC structure. Owing to this, our previous work [110] was only able to develop a trellis representation of the EG code by decomposing each symbol d_i into two sub-symbols x_i and t_i as shown in Figure 4.2(b). This was motivated by the observation that each EG codeword $EG(d_i)$ may be considered to be a concatenation of a unary codeword Unary(x_i) and a FLC suffix FLC$(t_i, x_i - 1)$, where $x_i = \lfloor \log_2(d_i) \rfloor + 1$ and $t_i = d_i - 2^{\lfloor \log_2(d_i) \rfloor}$ as shown in Table 4.1. Here FLC$(t_i, x_i - 1)$ is the binary representation of the integer t_i using $(x_i - 1)$ bits. As shown in Figure 4.2(b), each sub-symbol x_i is encoded by the EGEC(UEC) part of the EGEC code, while each sub-symbol t_i is encoded by the EGEC(FLC-CC) part. However, the reliance on these two parts leads to the requirement to tailor the UEP of the two parts for the specific source probability distribution, which may not match with the actual source distribution if it is unknown or non-stationary, as well as imposing for the disadvantages associated with an increased delay, loss of synchronisation, capacity loss and increased complexity due to puncturing, as described in Section 4.1.

In order to eliminate the requirement for a complicated code structure comprising two parts with the design-objective of neating a simple trellis structure, we propose a novel reordering of the bits in each EG codeword. We refer to the reordered code as the REG code, where the generalized structure of each REG codeword is shown in Figure 4.4. The reordering is conceived as follows. As described above, the first

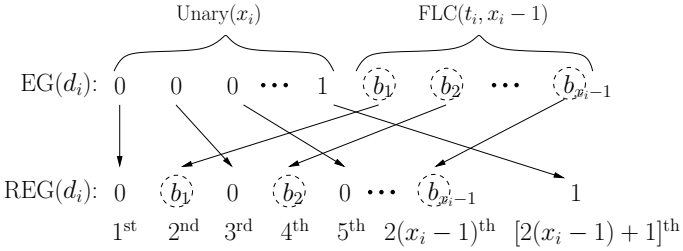


Figure 4.4: The reordering of an EG codeword to obtain the corresponding REG codeword

x_i bits of the conventional EG codeword, EG(d_i) are given by a unary codeword Unary(x_i). These bits become the odd-indexed bits of the our corresponding REG codeword. Notice that the final 1-valued bit in Unary(x_i) becomes the final bit in REG(d_i), since all REG codewords comprise an odd number of bits, in common with all EG codewords. The last $x_i - 1$ bits of the conventional codeword EG(d_i) comprise the FLC codeword FLC($t_i, x_i - 1$), which become the even-indexed bits of the corresponding REG codeword REG(d_i). Since each REG codeword has the same length of $l_{REG(d_i)} = 2\lfloor \log_2(d_i) \rfloor + 1$ as the corresponding EG codeword, the REG code will have the same average codeword length l_{REG} as the EG code, which is given by (4.5). Therefore, since the EG is a universal code, so too is the REG. This approach is motivated by the difference in the structures of the unary and EG codewords shown in Table 4.1. In Chapter 2, a UEC code was designed for the JSCC of unary-encoded symbols, in order to facilitate near-capacity communication. This is achieved by designing the UEC trellis of Section 2.3.2 for ensuring that the path through the trellis remains synchronised with the unary codewords. More specifically, the UEC trellis uses the logical 1-valued bit at the end of each codeword to detect the boundary between consecutive codewords and to trigger a return to state 1 or 2. By contrast, maintaining trellis synchronisation during the joint source and channel coding of EG-coded symbols is more complicated. This is because the length of the EG codeword depends on the length of its unary prefix, which may be detected using the 1-valued bit at the end. However, an EGEC trellis designed for maintaining synchronisation with the EG codewords would require states, i.e. memory for storing the length of the unary prefix all the way until the end of the FLC suffix is reached, whereupon a return to state 1 or 2 could be triggered. Since the unary prefix can have any length selected from an infinite set, an infinite number of states would be required to store this information, hence preventing the construction of a practical trellis. Instead, we can maintain synchronisation by reordering the bits in the EG codeword, so that the logtical 1-valued bit at the end of the unary prefix appears instead at the end of the REG codeword. In this way, this logical 1-valued bit may be used for detecting the boundary between consecutive codewords and to trigger a return to state 1 or 2 in the proposed REGEC trellis. In this way, synchronisation can be maintained and near-capacity joint source and channel coding can be achieved, as it will be introduced in Section 4.4.

4.4 Reordered Elias Gamma Error Correction encoder

In this section, we introduce the REGEC encoder, which is illustrated in Figure 4.2(a). In Section 4.4.1, we discuss the operation of the REG source encoder. The operation

of the REGEC trellis is described in Section 4.4.2. Finally, Section 4.4.3 describes the serial concatenation of the REGEC encoder with the Unity Rate Convolutional (URC) encoder and QPSK modulator of Figure 4.2(a).

4.4.1 Reordered Elias Gamma encoder

The REG encoder of Figure 4.2(a) represents each symbol d_i in the vector $\mathbf{d}$ using the corresponding REG codeword REG(d_i), as shown in Table 4.1. These codewords are then concatenated to obtain the b-bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$ shown in Figure 4.2(a). For example, the vector $\mathbf{x} = [6, 3, 1, 9, 2, 1, 1, 2]$ of $a = 8$ symbols yields the $b = 24$-bit vector $\mathbf{y} = 010010111000001100111001$.

4.4.2 Reordered Elias Gamma Error Correction trellis encoder

As shown in Figure 4.2(a), the bit vector of concatenated REG codewords $\mathbf{y}$ is forwarded to a trellis encoder, which employs a novel REGEC trellis for encoding each bit y_j in the vector $\mathbf{y}$, in order of increasing bit-index j. The trellis comprises b number of concatenated trellis stages of the type depicted in Figure 4.5. Each trellis stage comprises $2r$ number of transitions between r number of states, where r is required to satisfy $r = 2f + 2$, where f must be even. For example, an $r = 6$-state trellis is shown in Figure 4.5(a), an $r = 14$-state trellis is shown in Figure 4.5(b) and the general case is given in Figure 4.5(c). Each successive bit of y forces the trellis encoder to transition from its particular previous state $m_{j-1} \in \{1, 2, \ldots, r\}$ into a new state $m_j \in \{1, 2, \ldots, r\}$ that is selected from two legitimate alternatives, depending on the bit value y_j. In the trellis stages of Figure 4.5, $y_j = 0$ forces the trellis to make the dashed transition, while $y_j = 1$ forces the trellis to obey the solid transition. The encoding process always commences from the state $m_0 = 1$. The bit vector $\mathbf{y}$ identifies a path through the trellis, which may be represented by a vector $\mathbf{m} = [m_j]_{j=0}^{b}$ comprising $(b + 1)$ state values. For example, the bit vector $\mathbf{y} = 010010111000001100111001$ yields the path $\mathbf{m} = [1, 3, 6, 4, 6, 1, 3, 6, 1, 2, 4, 6, 4, 6, 4, 5, 2, 4, 6, 1, 2, 1, 3, 5, 2, 1]$ through the $r = 6$-state trellis of Figure 4.5(a).

As discussed in Section 4.3, the odd-indexed bits in the REG codewords derive from a unary codeword, while the even-indexed bits come from an FLC codeword. These unary and FLC bits force the trellis path into different sub-sets of the r trellis states. More specifically, we decompose the set of r states in to three sub-sets, namely the unary states, the FLC states and the holding states. The trellis is designed for ensuring that each input bit y_j that is provided by a unary bit causes a transition from one of the first f number of states $m_{j-1} \in \{1, 2, \ldots, f\}$, which we refer to as the

unary states, where f must be even. The transition enters a next state m_j, according to

$$m_j = \begin{cases} 1 + \text{odd}(m_{j-1}) \\ \qquad \text{if } \mathrm{y_j} = 1 \text{ and } \mathrm{m_{j-1}} \in \{1, 2, \ldots, \mathrm{f}\} \\ m_{j-1} + f \\ \qquad \text{if } \mathrm{y_j} = 0 \text{ and } \mathrm{m_{j-1}} \in \{1, 2, \ldots, \mathrm{f}\} \end{cases}, \tag{4.6}$$

where odd$(\cdot)$ yields 1 if its operand is odd or 0 if it is even. Note that since each REG codeword ends with a unary bit having the value $y_j = 1$, the trellis path $\mathbf{m}$ is guaranteed to enter either state $m_j = 1$ or $m_j = 2$ after each codeword. In this way, the transitions between the states of the REGEC trellis are synchronised with the transitions between the REG codewords in the bit vector $\mathbf{y}$. For the same reason, the trellis path $\mathbf{m}$ is guaranteed to terminate in the state $m_b = 1$ or $m_b = 2$ of the end of the encoding process. By contrast, the other unary bits in each REG codeword have the value $y_j = 0$, which cause transitions to one of the next $(f - 2)$ states $m_{j-1} \in \{f + 1, f + 2, \ldots, 2f - 2\}$, which we refer to as the FLC states, since the next bit will be an FLC bit. This FLC bit is guaranteed to cause a transition from the FLC state to a unary state, since an FLC bit is always followed by a unary bit in the REG codewords. The next state m_j, is selected according to

$$m_j = \begin{cases} m_{j-1} - f + 2 \cdot \text{odd}(m_{j-1}) + 1 \\ \qquad \text{if } \mathrm{y_j} = 1 \text{ and } \mathrm{m_{j-1}} \in \{\mathrm{f} + 1, \ldots, 2\mathrm{f} - 2\} \\ m_{j-1} - f + 2 \\ \qquad \text{if } \mathrm{y_j} = 0 \text{ and } \mathrm{m_{j-1}} \in \{\mathrm{f} + 1, \ldots, 2\mathrm{f} - 2\} \end{cases}. \tag{4.7}$$

Observe that when $f = 2$, there are no FLC states in the trellis, as shown in Figure 4.5(a). Note that REG codewords having a length $l(d_i) \leqslant (2f - 2)$ cause the path $\mathbf{m}$ to enter only the unary and FLC states described above. However, REG codewords having a length $l(d_i) > (2f - 2)$ require four additional states, which we refer to as the holding states, since they act as a 'holding pattern' for the bits in the REG codeword from the $(2f - 1)^{\text{st}}$ bit onward. More specifically, the FLC holding states $m_j \in \{2f - 1, 2f\}$ are entered into, if the unary bit $y_j = 0$ is encountered, while being in one of the unary states of the set $m_{j-1} \in \{f - 1, f\}$, as shown in (4.6). Upon emerging from the FLC holding states $m_{j-1} \in \{2f - 1, 2f\}$, the next state will be chosen from the unary holding states of the set $m_j \in \{2f + 1, 2f + 2\}$ according to

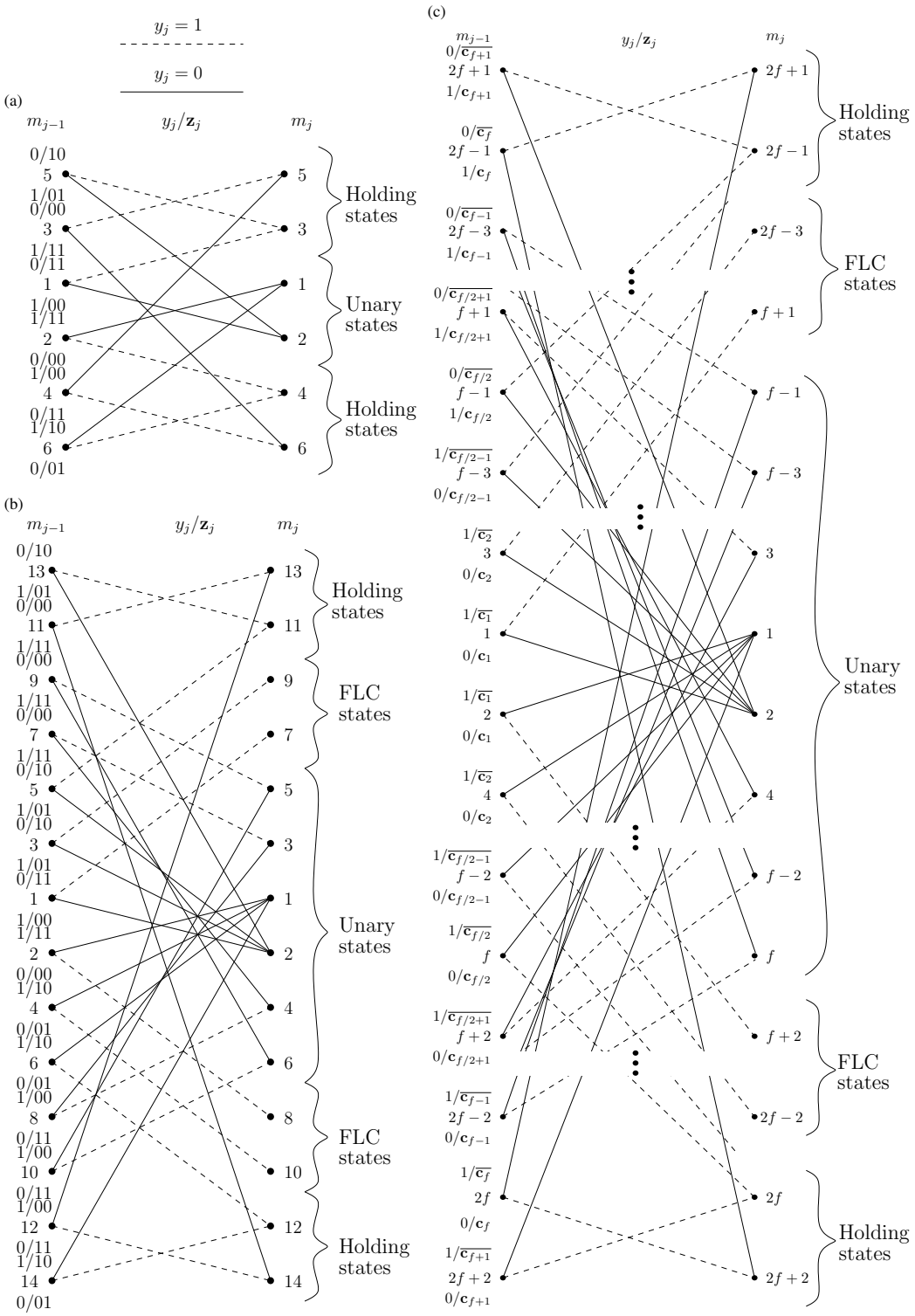


Figure 4.5: (a) An $f = 2$, $r = 6$-state $n = 2$-bit REGEC trellis using the codebook $\mathbb{C} = [00; 11; 01]$. (b) An $f = 6$, $r = 14$-state $n = 2$-bit REGEC trellis where using the codebook $\mathbb{C} = [00; 01; 01; 11; 11; 11; 01]$ which is constructed by extending the codebook $\mathbb{C} = [00; 11; 01]$. (c) The generalized REGEC trellis, having r states and n-bit codewords, where $\mathbb{C} = [\mathbf{c}_1; \mathbf{c}_2; \ldots, \mathbf{c}_f, \mathbf{c}_{f+1}]$.

$$m_j = \begin{cases} m_{j-1} + 2 \cdot \text{odd}(m_{j-1}) + 1 \\ \qquad \text{if } \mathrm{y_j} = 1 \text{ and } \mathrm{m_{j-1}} \in \{2\mathrm{f}-1, 2\mathrm{f}\} \\ m_{j-1} + 2 \\ \qquad \text{if } \mathrm{y_j} = 0 \text{ and } \mathrm{m_{j-1}} \in \{2\mathrm{f}-1, 2\mathrm{f}\} \end{cases} . \tag{4.8}$$

Likewise, upon traversing from the unary holding states $m_{j-1} \in \{2f+1, 2f+2\}$, the next state will be chosen according to

$$m_j = \begin{cases} 1 + \text{odd}(m_{j-1}) \\ \qquad \text{if } \mathrm{y_j} = 1 \text{ and } \mathrm{m_{j-1}} \in \{2\mathrm{f}+1, 2\mathrm{f}+2\} \\ m_{j-1} - 2 \\ \qquad \text{if } \mathrm{y_j} = 0 \text{ and } \mathrm{m_{j-1}} \in \{2\mathrm{f}+1, 2\mathrm{f}+2\} \end{cases} . \tag{4.9}$$

Note that the trellis path $\mathbf{m}$ will remain in the holding states, as long as unary bits having the value of $y_j = 0$ are encountered. When the final $y_j = 1$-valued unary bit of the REG codeword is encountered, the trellis path returns to state $m_j = 1$ or $m_j = 2$, ready for the start of the next REG codeword. Finally, combining Equations (4.6) to (4.9) yields (4.10). Note that the total number of states is given by $r = (2f + 2)$.

The path $\mathbf{m}$ may be modeled as a particular realization of a vector $\mathbf{M} = [M_j]_{j=0}^{b}$ comprising $(b+1)$ RVs, which are associated with the transition probabilities $\Pr(M_j = m, M_{j-1} = m') = P(m, m')$ of (4.11), which depends on the source symbol probabilities $P(d)$, as derived in the Appendix. In (4.11), l_1 is the average length of Unary(x_i), as described in Section 4.3. In the case of the finite Zeta-like distribution of (3.1), l_1 is given by [110]

$$l_1 = 1 - \frac{H_L^{(s)}}{\ln(2) H_L^{(s)}} - \frac{1}{H_L^{(s)}} \sum_{d \in \mathbb{N}_\mathbb{L}} d^{-s} \text{frac}[\log_2(d)]. \tag{4.12}$$

The conditional transition probabilities $Pr(M_j = m | M_{j-1} = m')$ are given by [1]

$$P(m|m') = \frac{P(m, m')}{\sum_{\check{m}=1}^{r} P(\check{m}, m')}. \tag{4.13}$$

Once the path $\mathbf{m}$ has been determined, the trellis encoder uses it to represent each bit y_j in the vector $\mathbf{y}$ by an n-bit codeword z_j. This is selected from the matrix of $r/2$ codewords $\mathbb{C} = [\mathbf{c}_1; \mathbf{c}_2; \ldots; \mathbf{c}_{f+1}]$ or from the complementary matrix $\overline{\mathbb{C}} = [\overline{\mathbf{c}_1}; \overline{\mathbf{c}_2}; \ldots; \overline{\mathbf{c}_{f+1}}]$. As shown in Figure 4.5(c), this is achieved according to

$$
m_j = \begin{cases} 1 + \mathrm{odd}(m_{j-1}) & \text{if } y_j = 1 \text{ and } m_{j-1} \in \{1, 2, \ldots, f, 2f+1, 2f+2\} \\ m_{j-1} + f & \text{if } y_j = 0 \text{ and } m_{j-1} \in \{1, 2, \ldots, f\} \\ m_{j-1} - f + 2 - y_j + 2 \cdot y_j \cdot \mathrm{odd}(m_{j-1}) & \text{if } y_j \in \{0,1\} \text{ and } m_{j-1} \in \{f+1, f+2, \ldots, 2f-2\} \\ m_{j-1} + y_j \cdot (2 \cdot \mathrm{odd}(m_{j-1}) + 1) + 2 \cdot \mathrm{odd}(y_j + 1) & \text{if } y_j \in \{0,1\} \text{ and } m_{j-1} \in \{2f-1, 2f\} \\ m_{j-1} - 2 & \text{if } y_j = 0 \text{ and } m_{j-1} \in \{2f+1, 2f+2\} \end{cases} \tag{4.10}
$$

$$P(m,m') = \begin{cases} \frac{1}{2l}\left[1-\sum_{d=1}^{2^{\dot{x}}-1} P(d)\right] & \text{if } m' \in \{1,\dots,f\} \text{ and } m = m'+f \\ \frac{1}{2l}\sum_{d=2^{\dot{x}-1}}^{2^{\dot{x}}-1} P(d) & \text{if } m' \in \{1,\dots,f\} \text{ and } m = \text{odd}(m')+1 \\ \frac{1}{2l}\sum_{x=\ddot{x}+1}^{\lfloor \log_2(L)\rfloor}\sum_{\ddot{t}=0}^{2^{\ddot{x}}-1} \text{odd}(\ddot{t}+1+y_j)\sum_{d=2^x+\ddot{t}\frac{2^x}{2^{\ddot{x}}}}^{2^x+(\ddot{t}+1)\frac{2^x}{2^{\ddot{x}}}-1} P(d) & \text{if } m' \in \{f+1,\dots,2f-2\} \text{ and } m = \ddot{d}-y_j+2y_j\cdot\text{odd}(m') \\ \frac{1}{2l}\sum_{x=f}^{\lfloor \log_2(L)\rfloor}\sum_{\dot{x}=f}^{x}\sum_{\mathring{t}=0}^{2^{\dot{x}}-1} \text{odd}(\mathring{t}+1+y_j)\sum_{d=2^x+\mathring{t}\frac{2^x}{2^{\dot{x}}}}^{2^x+(\mathring{t}+1)\frac{2^x}{2^{\dot{x}}}-1} P(d) & \text{if } m' \in \{2f-1,2f\} \text{ and } m = m'+y_j(2\cdot\text{odd}(m')+1)+2\cdot\text{odd}(y_j+1) \\ \frac{1}{2l}\left[l_1-\frac{f}{2}+\sum_{d=1}^{2^{(f/2+1)}-1} p(d)(\lfloor \log_2(d)\rfloor+1-\frac{f}{2})\right] & \text{if } m' \in \{2f+1,2f+2\} \text{ and } m = m'-2 \\ \frac{1}{2l}\left[1-\sum_{d=1}^{2^{f/2}-1} p(d)\right] & \text{if } m' \in \{2f+1,2f+2\} \text{ and } m = \text{odd}(m')+1 \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

$$d \in \{1,2...,L\};\quad y_j \in \{0,1\};\ \dot{x} = \lceil m'/2\rceil;\ \ddot{d} = m'-f+2;\quad \ddot{x} = \lceil \ddot{d}/2\rceil - 1$$

$$l_1 = P(d_i)\cdot[\lfloor \log_2(d_i)\rfloor + 1]$$

$$\mathbf{z}_j = \begin{cases} \overline{\mathbf{c}_{\lceil m_{j-1}/2 \rceil}} & \text{if } y_j \neq \text{odd}(m_{j-1}) \\ \mathbf{c}_{\lceil m_{j-1}/2 \rceil} & \text{if } y_j = \text{odd}(m_{j-1}) \end{cases} . \tag{4.14}$$

Following this, the selected codewords are concatenated to obtain the bn-bit vector $\mathbf{z} = [z_k]_{k=1}^{bn}$ of Figure 4.2. For example, the vector $\mathbf{y} = 010010111000001100111001$ of $b = 24$ bits is represented by the vector $\mathbf{z} = 111101111011111000001101110100010011100011110001$ of $bn = 48$ bits, when employing the $r = 6$-state REGEC trellis of Figure 4.5(a), with the $n = 2$-bit codebook $\mathbb{C} = [00; 11; 01]$.

Note that the selection of the number of trellis states r is discussed in Section 4.6.4, while the selection of the codebook $\mathbb{C}$ is discussed in Section 4.6.5. We emphasize that REGEC trellis encoder operates in a similar manner to a UEC trellis encoder and a CC encoder, but subject to the following important differences, as follows.

1. As in the UEC trellis encoder, a bit having the value of $y_j = 1$ will force a transition from the odd-indexed states at the top half of the REGEC trellis to the even-indexed states in the bottom half and vice-versa. Owing to this symmetry and due to using complementary codewords, the REGEC trellis encoder produces equiprobable bit values for the bit vector $\mathbf{z}$. This results in a bit entropy of $H_z = 1$, which is a necessary condition for avoiding capacity loss, as described in [1]. However, in contract to the unary codewords of the UEC encoder, $y_j = 1$ does not only occur at the end of a REG codeword, resulting in transitions between the top and bottom halves of the REGEC trellis more frequently than only at the end of each codeword. By contrast, CC encoders produce binary values that are not guaranteed to be equiprobable, unless they are specifically parametrized for this purpose, as characterized in [1, Table II].
2. As we described above, the final unary-bit y_j in each REG codeword is guaranteed to induce a transition to either state $m_j = 1$ or state $m_j = 2$ of the REGEC trellis, in analogy with the UEC trellis. However, unlike in the UEC encoder, the particular one from the pair of states $m_b = 1$ or state $m_b = 2$ that is selected at the end of the REGEC trellis path $\mathbf{m}$ depends on more than factors just deciding whether the length a of the symbol vector $\mathbf{d}$ is odd or even. This is due to the transitions between the top and bottom halves of the REGEC trellis that are caused by bits having the value $y_j = 1$ in the middle of REG codewords, as described above. By contrast, in a generalized CC encoder, the trellis path can potentially end in any state, since the transitions between states are not synchronized with the codewords of the source encoder.

Since the binary values in the vector $\mathbf{z}$ are equiprobable, the average coding rate of the REGEC encoder is given by

$$R^{\mathrm{o}} = \frac{H_D}{l_{REG} n}. \tag{4.15}$$

Here, we employ the roman superscript 'o' to indicate that this coding rate relates to the outer encoder of a serial concatenation, namely the REGEC encoder shown in Figure 4.2(a).

4.4.3 Integration of the REGEC encoder into a transmitter

Following REGEC encoding, the bit vector $\mathbf{z}$ is interleaved by the block π_1, URC encoded [87] and then interleaved again by the block π_2, as shown in Figure 4.2(a). Puncturing may also be performed within π_2 in order to achieve a particular desired effective throughput η for the transmitter. This is achieved by discarding an appropriate number of bits following interleaving. The inner coding rate R^i is defined by the ratio of bits input into the URC encoder to the number of bits output by π_2, where $R^i > 1$ will be obtained if puncturing is used. Here we employ the roman superscript 'i' to indicate that this coding rate relates to the inner code of a serial concatenation, namely the punctured URC code shown in Figure 4.2(a). Following this, M = 4-ary Gray-coded QPSK modulation may be employed for transmission, as shown in Figure 4.2(a). Note that other mapping schemes or a modulation scheme having a higher order M can be employed instead, although this may increase the complexity of the receiver, as we will discuss in Section 4.6. The effective throughput of the transmitter is given by

$$\eta = R^o \cdot R^i \cdot \log_2(M). \tag{4.16}$$

Note that no knowledge of the source probability distribution $P(x)$ is required anywhere in the transmitter.

4.5 Reordered Elias Gamma Error Correction decoder

In this section, we describe the operation of the REGEC decoder of Figure 4.2(a). In Section 4.5.1, we discuss the integration of the REGEC decoder with the URC decoder and QPSK demodulator of Figure 4.2(a). Following this, we detail the operation of the REGEC trellis decoder in Section 4.5.2, while the REG decoder is described in Section 4.5.3.

4.5.1 Integration of Reordered Elias Gamma Error Correction decoder into a receiver

In the receiver, soft QPSK demodulation [97], depuncturing and deinterleaving π_2^{-1}, Bahl-Cocke-Jelinek-Raviv (BCJR)-based URC decoding [9] and further deinterleaving π_1^{-1} may be performed, before invoking the proposed REGEC decoder of Figure 4.2(a). Note that the receiver is required to employ the same pseudo-random interleaver designs as the transmitter. However, the entire set of interleavers can be generated independently by both the transmitter and receiver using only a single pseudo-random number generator seed. This seed may be hard-coded into both the transmitter and receiver, or may be reliably conveyed using only a very small amount of side information. The REGEC decoder is provided with the *a priori* Logarithmic Likelihood Ratio (LLR) vector $\tilde{\mathbf{z}}^a$ and in response it generates the extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ of Figure 4.2(a), which may be iteratively exchanged with the serially concatenated URC decoder, until iterative decoding convergence to an infinitesimally low SER is achieved. In turn, the URC decoder may also iteratively exchange extrinsic LLRs with the demodulator [114], in order to avoid capacity loss when a mapping scheme other than Gray coding or when a higher-order modulation scheme is employed. Note that the combination of the URC decoder and the demodulator will have an EXtrinsic Information Transfer (EXIT) curve that reaches the $(1, 1)$ point at the top right corner of the EXIT chart [106].

4.5.2 Reordered Elias Gamma Error Correction trellis decoder

As shown in Figure 4.2(a), the REGEC trellis decoder is provided with a vector of *a priori* LLRs $\tilde{\mathbf{z}}^{\mathrm{a}} = [\tilde{z}_k^{\mathrm{a}}]_{k=1}^{bn}$ that pertain to the corresponding bits in the vector $\mathbf{z}$. The trellis decoder applies the BCJR algorithm [50] to a REGEC trellis of the sort shown in Figure 4.5(c) to consider every legitimate bit vector that could be represented by $\tilde{\mathbf{z}}^a$, having the particular length bn. Here the value of bn is assumed to be perfectly known to the receiver, where the transmitter may employ a small amount of side information to reliably convey this value in practice. Here, the synchronization between the REGEC trellis and the REG codewords is exploited during the BCJR algorithm's γ_t calculation of [50, Equation (9)], by employing the conditional transition probabilities $P(m|m')$ of (4.11). Note that the REGEC trellis should be terminated at $m_0 = 1$ and at both possibilities for the final state, namely $m_b = 1$ and $m_b = 2$, as described in Section 4.5.1. As shown in Figure 4.2(a), the BCJR decoder generates the vector of extrinsic LLRs $\tilde{\mathbf{z}}^{\mathrm{e}} = [\tilde{z}_k^{\mathrm{e}}]_{k=1}^{bn}$ which is provided for the next iteration of the concatenated URC decoder's operation. Note that the REGEC trellis decoder's BCJR algorithm has only modest complexity, since it may employ a low number r

of states. Furthermore, it facilitates error correction even if the symbol probability distribution $P(d)$ is unknown, provided that the channel SNR is sufficiently high, as we shall demonstrate in Section 4.6.5. In this case, the conditional transition probabilities $P(m|m')$ of (4.11) will also be unknown and so they are simply omitted from the BCJR algorithm's γ_t calculation.

The transformation of $\tilde{\mathbf{z}}^{\mathrm{a}}$ into $\tilde{\mathbf{z}}^{\mathrm{e}}$ by the trellis decoder of Figure 4.2(a) may be characterized by plotting the inverted REGEC EXIT curve in an EXIT chart [51], as exemplified in Figure 4.6. Note that if a suitably designed codebook $\mathbb{C}$ comprising

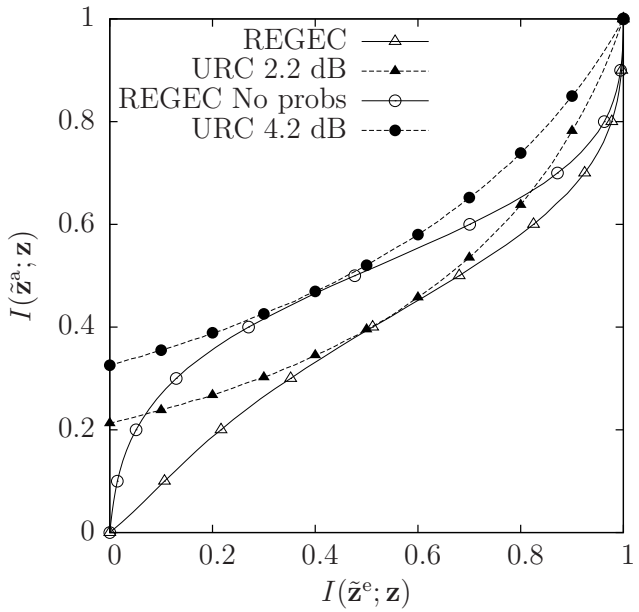


Figure 4.6: EXIT charts of the REGEC scheme. Here, the symbols of $\mathbf{d}$ obey a finite Zeta-like distribution having $L = 1000$ and $p_1 = 0.7942$, while the REGEC codewords $\mathbb{C} = [00; 11; 01]$ comprise $n = 2$ bits and result in an REGEC trellis having $r = 6$ states. Furthermore, a URC decoder having $r = 2$ states is concatenated with Gray-coded QPSK modulation, for communication over an uncorrelated narrowband Rayleigh fading channel having various E_{b}/N_0 values.

codewords having at least $n = 2$ bits is employed, then the free distance d_{free} of the REGEC code will be at least two, as it will be quantified in Section 4.6. In this case the inverted REGEC EXIT curve will reach the $(1, 1)$ point in the top right corner of the EXIT chart [71]. Since the URC decoder and demodulator also have an EXIT curve that reaches the $(1, 1)$ point in the top right corner of the EXIT chart [106] as shown in Figure 4.6, iterative decoder convergence towards the Maximum Likelihood (ML) performance is facilitated [107].

The EXIT chart area A^o that is situated below the inverted REGEC EXIT curve is given by [110, 112]

$$A^o = \frac{1}{n} \sum_{m'=1}^{r} \sum_{m=1}^{r} P(m, m') \log_2 \left(\frac{1}{P(m|m')} \right). \tag{4.17}$$

Note that, the REGEC EXIT chart area A^o is independent of the codebook design, but using different codebooks can affect the shape of the EXIT curve, as will be discussed in Section 4.6.2. Following the completion of iterative decoding, the REGEC trellis decoder may employ the Viterbi algorithm to generate the vector $\hat{\mathbf{y}} = [\hat{\mathbf{y}}_j]_{j=1}^{b}$ of recovered bits, which pertain to the corresponding bits in the vector $\mathbf{y}$, as shown in Figure 4.2(a).

4.5.3 Reordered Elias Gamma decoder

The decoded bit vector $\hat{\mathbf{y}}$ can be REG decoded in order to obtain the recovered symbol vector $\hat{\mathbf{d}}$ of Figure 4.2(a). If there are any bit errors in the vector $\hat{\mathbf{y}}$, then we might arrive either at the wrong legitimate REG codeword or fail to find a legitimate codeword. In this case, these bits are discarded. If the decoded symbol vector $\hat{\mathbf{d}}$ does not contain the correct number a of symbols, then an appropriate number of symbols is removed from the end of $\hat{\mathbf{d}}$ or an appropriate number of 1-valued symbols is appended to the end of $\hat{\mathbf{d}}$, accordingly. Here, it is assumed that the REG decoder has perfect knowledge of a. In practice, this value may be fixed in both the transmitter and receiver, or it may be reliably conveyed from transmitter to receiver using a small amount of side information.

4.6 Parametrization of the Reordered Elias Gamma Error Correction code

In this section, we discuss the parametrization of the REGEC code. In Section 4.6.1, we introduce the extension rule of the REGEC codebook extension. In Section 4.6.2, we analyse the the near-capacity operation of the REGEC decoder. In Section 4.6.3, we discuss the codebook design of the REGEC trellis encoder, considering the free distance properties of various candidate codebooks. The EXIT curves of the candidate codebooks and their EXIT chart matching are discussed in Section 4.6.4. Finally we analyse the error floor of the candidate codebooks in Section 4.6.5 and selected a recommended codebook.

4.6.1 Reordered Elias Gamma Error Correction codebook extension

As described in Section 4.4.2, an REGEC trellis having r number of states is parametrized by a set of $r/2$ codewords $\mathbb{C}$, each comprising n number of bits, where $\mathbb{C} = [00; 11; 01]$ in the $r = 6$, $n = 2$ example of Figure 4.5(a) and $\mathbb{C} = [00; 01; 01; 11; 11; 11; 01]$ in the $r = 14$, $n = 2$ example of Figure 4.5(b). Any codebook $\mathbb{C}'$ corresponding to a trellis having $r' = 2f' + 2$ number of states can be extended to a codebook $\mathbb{C}''$ corresponding to $r'' > r'$ number of states including several new unary and FLC states. Note that

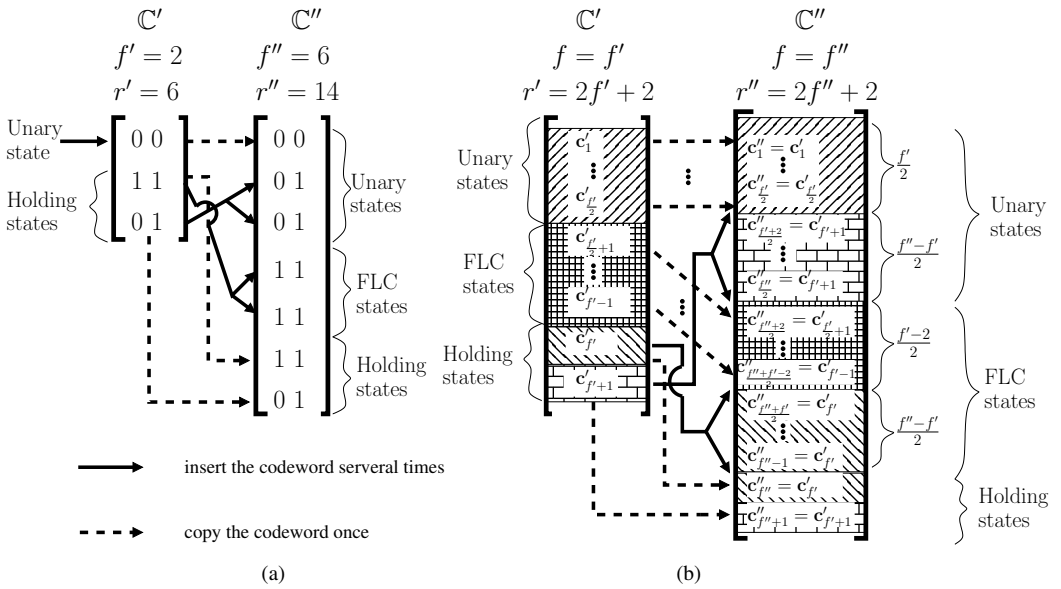


Figure 4.7: (a) Codebook extension from the codebook $\mathbb{C}'$ of the $r' = 6$-state trellis of Figure 4.5(a) to the codebook $\mathbb{C}''$ of the $r'' = 14$-state trellis of Figure 4.5(b). (b) The generalized extension of a codebook $\mathbb{C}'$ for an $r' = (2f' + 2)$-state trellis to a codebook $\mathbb{C}''$ for an $r'' = (2f'' + 2)$-state trellis, where $f'' > f'$.

when provided with the same REG-encoded bit vector $\mathbf{y}$, REGEC trellis encoders employing the trellises of Figure 4.5(a) and Figure 4.5(b) are guaranteed to generate identical REGEC-encoded bit vectors $\mathbf{z}$, despite using different codebooks $\mathbb{C}$. This is because the $r = 14$ codebook of Figure 4.5(b) is an *extension* of the $r = 6$ codebook of Figure 4.5(a). In this way, the use of extension allows a higher number of states r to be used in the REGEC trellis decoder than in the REGEC trellis encoder. This allows us to dynamically change the number of states employed in the decoder in order to strike an attractive trade-off between its performance versus trellis complexity, as characterized in Section 4.6.2 [99].

This process is illustrated in Figure 4.7(a) for the specific case of extending the $f' = 2$ $r' = 6$-state codebook $\mathbb{C}' = [00; 11; 01]$ of Figure 4.5(a) to the $f'' = 6$ $r'' = 14$-state codebook $\mathbb{C}'' = [\mathbf{c}_1; \mathbf{c}_2; \mathbf{c}_3; \mathbf{c}_4; \mathbf{c}_5; \mathbf{c}_6; \mathbf{c}_7]$ of Figure 4.5(b). Meanwhile, Figure 4.7(b) illustrates the process for the generalised case. The process comprises 5 steps listed as follows:

1. The first $f'/2$ codewords in the codebook C' relate to the unary states, as described in Section 4.4. These codewords should be copied to the first $f'/2$ codewords of the new codebook C''. In the example of Figure 4.7(a), this gives $\mathbb{C}'' = [\mathbf{00}; \mathbf{c}_2; \mathbf{c}_3; \mathbf{c}_4; \mathbf{c}_5; \mathbf{c}_6; \mathbf{c}_7]$.
2. The next $f'/2 - 1$ codewords in the codebook C' relate to the FLC states, as described in Section 4.4. These codewords should be copied to the codewords in codebook C'' having the indices from $f''/2 + 1$ to $\frac{f''+f'-2}{2}$. Note that as described in Section 4.4.2, there are no FLC states when we have $f' = 2$ unary

states. Therefore in the example of Figure 4.7(a), nothing is copied to the codebook C'' in this step.

3. The $(f'+1)^{\text{th}}$ codeword of the codebook C' relates to the FLC holding state. This codeword should be copied $\frac{f"-f'}{2}$ times to form the codewords $[\mathbf{c}_{\frac{f'}{2}+1}; \ldots; \mathbf{c}_{\frac{f''}{2}}]$ of the codebook C". In the example of Figure 4.7(a), this gives $\mathbb{C}'' = [00, \mathbf{01}, \mathbf{01}, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7]$.
4. Similarly, the f'^{th} codeword of the codebook C' relates to the unary holding state. This codeword should be copied $\frac{f''-f'}{2}$ times to form the codewords $[\mathbf{c}_{\frac{f''+f'}{2}}; \ldots; \mathbf{c}_{f''-1}]$ of the codebook C'', which relate to the new unary states. In the example of Figure 4.7(a), this gives $\mathbb{C}'' = [00; 01; 01; \mathbf{11}; \mathbf{11}; \mathbf{c}_6; \mathbf{c}_7]$.
5. Finally, the f'^{th} and $(f'+1)^{\text{th}}$ codewords of codebook $\mathbb{C}'$ should be copied to form the f''^{th} and $(f''+1)^{\text{th}}$ codewords of the codebook $\mathbb{C}''$. In the example of Figure 8(a), this gives $\mathbb{C}'' = [00; 01; 01; 11; 11; \mathbf{11}; \mathbf{01}]$;

4.6.2 Performance analysis

Near-capacity operation is achieved, when reliable communication can be maintained at transmission throughputs η that approach the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity C [100] that is associated with $M = 4$ QPSK modulation and uncorrelated narrowband Rayleigh fading. This is facilitated, if the following conditions are satisfied [112]:

1. The URC decoder of Figure 4.2(a) is required to have an EXIT curve having an area beneath it of $A^{\text{i}} = C/[R^{\text{i}} \log_2(M)]$;
2. The area A^{o} beneath the inverted EXIT curve of the REGEC trellis decoder is required to approach the REGEC coding rate R^{o}.

If these two conditions are satisfied, then near-capacity operation will be achieved, when the shape of URC decoder's EXIT curve is closely matched to that of the inverted REGEC EXIT curve. This creates a narrow, but marginally open EXIT chart tunnel, which facilitates iterative decoding convergence towards the ML performance [107].

The first condition listed above is satisfied by a punctured URC code, as discussed in [112]. Figure 4.8 shows that the second of the above-mentioned conditions is satisfied when the RVs in the vector $\mathbf{D}$ obey the finite Zeta-like distribution of (3.1) having the cardinality $L = 1000$ and various values for the parameter p_1. This figure plots the REGEC coding rate R^{o} of (4.15) when multiplied with the REGEC trellis codeword lengths n. Furthermore, Figure 4.8 plots the product of n and the area A^{o} of (4.17) beneath the inverted REGEC EXIT curve for the case where the trellis decoder employs $f/2 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, giving $r \in \{6, 10, 14, 18, 22, 26, 30, 34, 38\}$

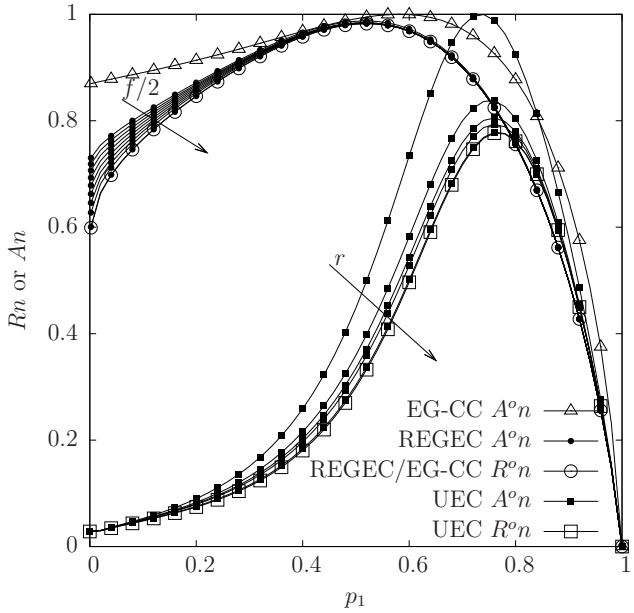


Figure 4.8: Plots of $R^o n$ and $A^o n$ that are obtained for the REGEC scheme, EG-CC scheme and UEC scheme, in the case where the symbol values of $\mathbf{d}$ obey a finite Zeta-like distribution having the parameter p_1 and cardinality $L = 1000$. Here, R^o is the coding rate, A^o is the area beneath the inverted EXIT curve and n is the codeword length of the corresponding scheme. The value of $A^o n$ is provided for an REGEC code having $f/2 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, while the value of $A^o n$ is provided for a UEC code having $r \in \{2, 4, 6, 8, 30\}$.

states. Note that according to (4.15) and (4.17), the area A^o and coding rate R^o are dependent on the symbol entropy H_D, average REG codeword length l_{REG} and trellis codeword length n, but are independent of the codebook design $\mathbb{C}$. Furthermore, the product of the REGEC EXIT chart area A^o and the codeword length n is related to the number of unary states f, as shown in Figure 4.8. In the case of the H.265 symbol value distribution of Figure 4.3, we obtain $R^o n = 0.8787$.

Figure 4.9 plots the discrepancy between $A^o n$ and $R^o n$ for the REGEC code as a function of $f/2$, where the source symbols of $\mathbf{d}$ obey the finite Zeta-like distribution of (3.1) for a cardinality of $L = 1000$ and for various values for the parameter p_1. Note that in all the scenarios considered, the discrepancy is less than 10^{-1} and becomes less than 10^{-2} when $f/2 \geq 4$, including the case of the H.265 symbol value distribution of Figure 4.3.

However, the trellis complexity and hence the complexity of REGEC decoding is proportional to the number of states. Our experiments revealed that $f = 2$ and $r = 6$ represents an attractive trade-off between maintaining a low trellis complexity and facilitating near-capacity operation.

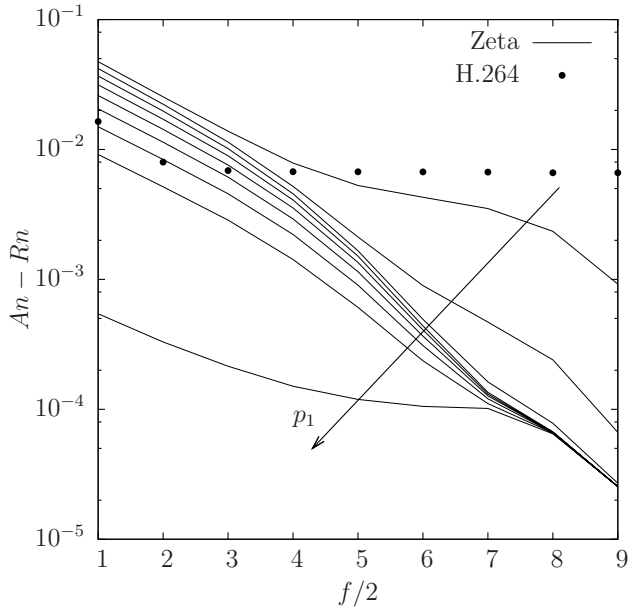


Figure 4.9: The discrepancy between $A^o n$ and $R^o n$ that results when REGEC codes having various values of $f/2$ are employed to encode symbol values having finite Zeta-like distributions with the parameters $L = 1000$ and $p_1 \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, as well as for symbol values obeying the H.265 distribution of Figure 4.3.

Table 4.2: Candidate REGEC codebooks $\{\mathbb{C}_i\}_{i=1}^{10}$ for $n = 2$ bits and $r = 6$ states and their corresponding FD d_f. For finite Zeta-like probability distributions having $L = 1000$ and various values of p_1, the number of states in the URC having the best matching EXIT curve is provided, together with the corresponding E_b/N_o tunnel bound in brackets.

candidate codebook		$\mathbb{C}_1$	$\mathbb{C}_2$	$\mathbb{C}_3$	$\mathbb{C}_4$	$\mathbb{C}_5$	$\mathbb{C}_6$	$\mathbb{C}_7$	$\mathbb{C}_8$	$\mathbb{C}_9$	$\mathbb{C}_{10}$
$\mathbf{c}_1$		00	00	00	00	00	00	00	00	00	00
$\mathbf{c}_2$		00	00	00	01	01	01	01	11	11	11
$\mathbf{c}_3$		00	01	11	00	01	10	11	00	01	11
d_f		2	3	2	3	4	4	3	4	3	4
Number of states in URC having best matching EXIT curve and resultant E_b/N_o tunnel bound in dB	p_1 =0.7942	8 (1.6)	4 (2.2)	4 (1.9)	4 (2.1)	2 (2.2)	2 (2.2)	2 (2.2)	4 (1.9)	2 (2.2)	4 (1.8)
	p_1 =0.6	8 (2.2)	2 (2.7)	8 (2.3)	2 (2.7)	2 (2.7)	8 (2.7)	2 (2.8)	8 (2.3)	2 (2.8)	8 (2.3)
	p_1 =0.4	8 (2.2)	2 (2.7)	8 (2.3)	2 (2.7)	2 (2.9)	2 (2.9)	2 (2.7)	8 (2.4)	2 (2.7)	8 (2.3)
	p_1 =0.2	8 (1.8)	2 (2.4)	8 (2)	2 (2.4)	2 (2.7)	2 (2.7)	2 (2.4)	4 (2.4)	2 (2.4)	4 (2.2)

4.6.3 REGEC codebook candidate selection

In this section, we will discuss the codebook design for an $n = 2$ $r = 6$ REGEC trellis. An $n = 2$ $r = 6$ codebook comprises $r/2 = 3$ codewords, each comprising $n = 2$ bits. Therefore, there are 2^6 possible $n = 2$ $r = 6$ codebooks. However it can be shown that all of these are equivalent to one of the 10 codebooks shown in Table 4.2, which contains no pairs of equivalent codebooks. More specifically, two codebooks are equivalent, if each pairing of codewords within one of the codebooks has the same Hamming Distance (HD) as the corresponding pairing of codewords within the other codebook. Owing to this, two codebooks are equivalent, if one can

be transformed into the other by toggling all bits and/or changing the order of the bits in each codeword using the same reordering pattern.

The 10 $n = 2$ $r = 6$ candidate REGEC codebooks are shown in Table 4.2, where the bits of the codewords have been toggled and reordered in order to minimise the decimal values that are represented by successive codewords. The error correction capability of a codebook may be characterized by the Free Distance (FD) that results at the output of the REGEC trellis encoder [75]. Table 4.2 quantifies the FD of each candidate codebook, which was obtained using a brute-force search. As described in Section 4.4, the REGEC trellis path always starts at the state $m_0 = 1$ and will always end at either state $m_b = 1$ or state $m_b = 2$. Therefore, our brute-force search only needs to consider the free distance between paths that start and end at these states. Owing to this, our experiments revealed that a trellis comprising five stages like that of Figure 4.5(a) is sufficient for finding the free distance, resulting in only a moderate searching complexity.

Table 4.2 suggest that the candidate codebooks $\mathbb{C}_5$, $\mathbb{C}_6$, $\mathbb{C}_8$ and $\mathbb{C}_{10}$ will produce the best error correction capability, since they have the highest FD of 4. However, in iterative decoding schemes it is necessary to separately consider the error correction capability in the turbo cliff and error-floor regions of the Symbol Error Ratio (SER) plot, before the best candidate parametrization can be identified with certainty, as we shall discuss in the following sections.

Note that the FD of on REGEC code remains unaltered if its codebook is extended using the process of Section 4.6.1, since extension does not change the REGEC-encoding bit vector $\mathbf{z}$ produced for a given REG-encoded bit vector $\mathbf{y}$. Furthermore, depending on the length n of each codeword, the FD of an REGEC code cannot be increased by increasing the number of states r above a particular limit. For example, the largest possible FDs of $n = 2$-bit REGEC codes is 4, regardless of whether $r = 6$ or $r > 6$ number of states are employed. This is because the legitimate transition path set of an r state trellis is a subset of the legitimate transition path set of a trellis having a higher number of possible states $r' > r$. Therefore, we will focus our attention on codebooks corresponding to trellises having $r = 6$ states throughout the remainder of this chapter.

4.6.4 EXIT charts of the REGEC candidate codebooks and the best matching URCs

As discussed in Section 4.6.2, the area A^o beneath the inverted REGEC EXIT function and the REGEC coding rate R^o are independent of the codebook design $\mathbb{C}$. However, the shape of the REGEC EXIT curve and therefore its match with the

URC EXIT curve does depend on the specific codebook design $\mathbb{C}$. Since the candidate codebooks of Table 4.2 are unique with no pair of codebooks that are equivalent to each other, their inverted EXIT curves are all different from each other. Owing to this, different candidate codebooks have inverted EXIT curves that match best with the EXIT curve of URC codes having different parametrizations. In order to investigate this, we plotted the inverted EXIT curves of each candidate REGEC codebook, when used to encode source symbols obeying finite Zeta-like distributions having the cardinality $L = 1000$ and various values for the parameter $p_1 \in \{0.7942, 0.6, 0.4, 0.2\}$. In each case, the resultant EXIT curve was plotted together with the EXIT curves of URC codes having 2, 4 and 8 states. Here generator polynomials of the form $[1, 0, \ldots, 0]$ and feedback polynomials of the form $[1, 1 \ldots, 1]$ were employed, since they are capable of creating open EXIT chart tunnels [9]. The channel E_b/N_0 value was adjusted in each case, until marginally open EXIT chart tunnels were obtained. For each of the cases considered, Table 4.2 quantifies the number of states employed by the URC code that creates a marginally open EXIT chart tunnel at the lowest E_b/N_0 value, as well as providing this E_b/N_0 tunnel bound.

Figures 4.6, 4.10(a) and 4.10(b) show the resultant EXIT charts for the cases of using the candidate codebooks $\mathbb{C}_1, \mathbb{C}_8$ and $\mathbb{C}_9$ to encode symbols obeying the finite Zeta-like distribution for $L = 1000$ and $p_1 = 0.7942$. Figures 4.6, 4.10(a) and 4.10(b) also show the corresponding EXIT charts that result in the case, where the symbol probability distribution is unknown in the receiver, as described in Section 4.5.2. The results of Table 4.2 show that $\mathbb{C}_1$ is the codebook that facilitates an open EXIT chart tunnel at the lowest E_b/N_0 value. This suggests that $\mathbb{C}_1$ should offer the best performance in the turbo cliff region of the SER plot, since an open EXIT chart tunnel implies that iterative decoding convergence to an ML SER performance can be achieved [108]. However, $\mathbb{C}_1$ may not offer the best performance in the error floor region of the SER plot, as we will investigate in the next section.

4.6.5 Error floor analysis

The error correction capability of the candidate REGEC codebooks in the error floor region may be evaluated by considering the SER plots of Figure 4.10(c). Note that when knowledge of the source probability distribution $P(d)$ is available at the receiver, the candidate codebooks $\mathbb{C}_8$ and $\mathbb{C}_9$ offer steep turbo cliffs at E_b/N_0 values near the corresponding E_b/N_0 tunnel bounds, as predicted by the EXIT charts analysis of Section 4.6.4. However, the candidate codebook $\mathbb{C}_1$ can be seen to suffer from an error floor, which prevents us from achieving a low SER at E_b/N_0 values near the corresponding E_b/N_0 tunnel bound of 1.6 dB. This may explained by the observation

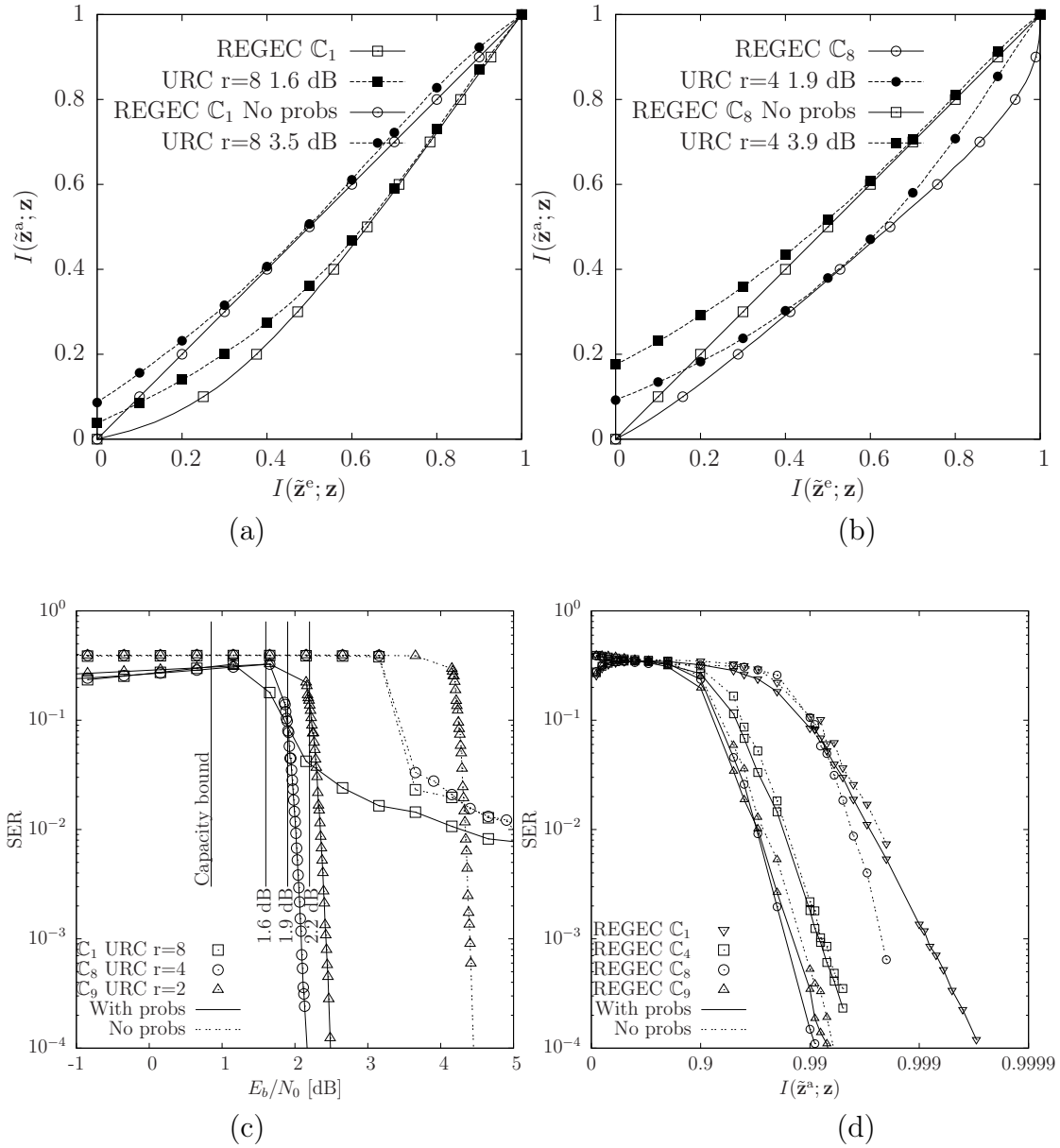


Figure 4.10: (a) and (b) EXIT charts of the proposed REGEC scheme. The EXIT curves are provided for REGEC codes employing the $n = 2$-bit $r = 6$-state codebooks $\mathbb{C} \in \{\mathbb{C}_1, \mathbb{C}_8\}$, as well as for a URC having $r \in \{4, 8\}$ states. (c) SER vs E_b/N_0 plot for the REGEC codes employing the $n = 2$-bit $r = 6$-state codebooks $\mathbb{C} \in \{\mathbb{C}_1, \mathbb{C}_8, \mathbb{C}_9\}$, when combined with URC codes having $r \in \{2, 4, 8\}$ states. (d) SER vs $I(\tilde{z}^a; z)$ plot for the REGEC codes employing the $n = 2$-bit $r = 6$-state codebooks $\mathbb{C} \in \{\mathbb{C}_1, \mathbb{C}_4, \mathbb{C}_8, \mathbb{C}_9\}$ when *a priori* LLR vectors $\tilde{\mathbf{z}}^a$ having different MI $I(\tilde{\mathbf{z}}^a; \mathbf{z})$ are provided to the REGEC trellis decoder. In all plots, the symbols of $\mathbf{d}$ obey the finite Zeta-like distribution having $p_1 = 0.7942$ and $L = 1000$. The plots labeled 'No Probs' indicate the case where the source distribution $P(d)$ is unknown to the receiver.

that the candidate codebook $\mathbb{C}_1$ requires the *a priori* LLR vector $\tilde{\mathbf{z}}^a$ of Figure 4.2(a) to have a higher Mutual Information (MI) $I(\tilde{\mathbf{z}}^a; \mathbf{z})$ than $\mathbb{C}_8$ and $\mathbb{C}_9$ require, in order to achieve a low SER, as shown in Figure 4.10(d). Owing to this, the candidate codebook $\mathbb{C}_1$ requires the iterative decoding process to converge closer towards the (1,1) point of the EXIT chart, which becomes difficult when the interleaver π_1 of Figure 4.2(a) has only a moderate length [116]. As shown in Figure 4.10(d), the candidate codebooks $\mathbb{C}_8$ and $\mathbb{C}_9$ require the lowest MIs $I(\tilde{\mathbf{z}}^a; \mathbf{z})$ in order to achieve low SERs.

Meanwhile, the codebooks $\mathbb{C}_5$, $\mathbb{C}_6$, and $\mathbb{C}_7$ have similar SER vs MI curves as $\mathbb{C}_4$ while $\mathbb{C}_2$, $\mathbb{C}_3$ and $\mathbb{C}_{10}$ have similar performance with $\mathbb{C}_1$. Note that the FD-3 codebook $\mathbb{C}_9$ offers better SER performance than serveral of the other codebooks having FDs of 4. We may speculate that this is because the the error correction capability of a candidate codebook is not only decided by the overall FD but also by the Hamming distances between the codewords that are associated with the transitions in the REGEC trellis having the highest transition probabilities of (4.11). In the case where the receiver has no knowledge of the source probability distribution $P(d)$, the SER curve of each candidate codebook is degraded, as shown in Figure 4.10(c). However, this degradation is particularly apparent in the case of $\mathbb{C}_8$, since this causes it to develop an error floor. By contrast, the candidate codebooks $\mathbb{C}_4$, $\mathbb{C}_5$, $\mathbb{C}_6$, $\mathbb{C}_7$ and $\mathbb{C}_9$ do not suffer from an error floor, regardless of whether knowledge of the source probability distribution is available in the receiver while $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{C}_3$ and $\mathbb{C}_{10}$ suffer from error floors for both cases. Overall, we recommend the candidate codebook $\mathbb{C}_9$, since it offers the best performance among the candidate codebooks that never suffer from an error floor. Also, the candidate codebook $\mathbb{C}_9$ works best with the URC inner code having the lowest complexity, namely that employing only $r = 2$ states. Therefore, we employ the candidate codebook $\mathbb{C}_9$ throughout the next section, when we compare the performance of the proposed REGEC scheme with suitably designed benchmarkers.

4.7 PERFORMANCE COMPARISON WITH THE BENCHMARKERS

In this section, we compare the proposed REGEC scheme to the EGEC, UEC and EG-CC benchmarkers of Figures 4.2(b), 4.2(c) and 4.2(d), respectively. Like the proposed REGEC schemes, both the EGEC and the UEC benchmarkers constitute examples of JSCCs, while the EG-CC benchmarker represents SSCC. More specifically, the EG-CC benchmarker employs an EG code for source coding, while an iteratively-decoded serial-concatenation of a CC and a URC is employed for separate

channel coding. We used QPSK modulation for transmission over an uncorrelated narrowband Rayleigh fading channel for all schemes, since this is representative of transmission over realistic wireless channels and because this facilitates direct comparison with the results of [1, 95]. In Section 4.7.1, we will discuss the parameterization of the REGEC scheme as well as of the three benchmarkers, in order to facilitate fair comparisons. Then we will analyse the SER performance of the proposed REGEC scheme and the three benchmarkers in Section 4.7.2.

4.7.1 Parametrization

Table 4.3 provides several parametrizations of the REGEC scheme, which are designed for transmitting symbols that obey the finite Zeta-like distribution of (3.1). Table 4.3 also provides corresponding parametrizations for the three benchmarkers, which offer the same throughput η as our REGEC scheme parametrizations. We parametrize the finite Zeta-like distribution using a cardinality of $L = 1000$ and the parameter of $p_1 \in \{0.7942, 0.6, 0.4, 0.2\}$, which represents a wide selection of the p_1 values shown in Figure 4.8. Note that the specific value of $p_1 = 0.7942$ is chosen, since it results in the same coding rate for the unary code and the EG code, and hence the same outer coding rate R^o for all schemes considered in this section. Note that, when we have $L \to \infty$, the UEC code becomes impractical for $p_1 = 0.2, 0.4$ and 0.6, since the average unary codeword length becomes infinite in these cases [1]. For finite case of $L = 1000$, the average unary codeword length is more than twice that of the EG code when $p_1 = 0.2$ and 0.4, hence severely degrading the performance of the UEC benchmarker. For this reason, the UEC benchmarker is not considered for these values of p_1. Table 4.3 also considers the case of source symbols obeying the H.265 distribution of Figure 4.3. Note that as described in Section 4.1, the EGEC benchmarker has two parts that must be jointly optimized for each particular source symbol distribution using UEP. More specifically, the puncturing rates R^i for the UEC part and the FLC-CC part must be carefully selected so that they have the same E_b/N_0 tunnel bound [110], as shown in Table 4.3.

For all the schemes considered, we selected codewords comprising $n = 2$ bits when possible, while $n = 3$-bit codewords were selected for the FLC-CC part of the EGEC benchmarker, whenever necessary to achieve the desired effective throughput η for designing the UEP. We selected $r = 6$ states for the proposed REGEC scheme, since this is sufficiently high for imposing only an insignificant amount of capacity loss, as discussed in Section 4.6.1. Furthermore, we employ the REGEC codebook $\mathbb{C}_9 = [00; 11; 01]$ in order to avoid the error floors that are charactorized in Section 4.5.2. Futhermore, we adopt the $r = 4$-state UEC trellis of [1] for both the UEC

Table 4.3: The parameters and characteristic of each scheme considered, for the case of source symbols obeying finite Zeta-like distributions having $L = 1000$ and different $p_1$values, as well as for the H.265 distribution of Figure 4.3.

P_1	Scheme		n	r	R^o	A^o	R^i	η	E_b/N_0 [dB] for $C = \eta$	E_b/N_0 [dB] for $A^i = A^o$	E_b/N_0 [dB] for open tunnel	Complexity
0.7942	REGEC		2	6	0.3834	0.3903	1	0.7669	0.85	1.0	2.2	412
	EGEC	UEC	2	4	0.3746	0.3815	1.0422			1.0	2.4	344
		FLC-CC	3	4	0.2862	0.2953	1.2623					
	UEC		2	4	0.3834	0.4021	1			1.0	2.5	322
	EG-CC		2	4	0.3834	0.4444	1			2.1	3.2	317
0.6	REGEC		2	6	0.4842	0.4877	1	0.9684	1.69	1.8	2.8	662
	EGEC	UEC	2	4	0.4904	0.4910	1			1.8	2.8	530
		FLC-CC	2	4	0.4696	0.4775	1					
	UEC		2	4	0.2482	0.2910	1.9505			2.9	5.7	1009
	EG-CC		2	4	0.4842	0.4995	1			2.0	2.9	510
0.4	REGEC		2	6	0.4789	0.4845	1	0.9578	1.65	1.8	2.7	1147
	EGEC	UEC	2	4	0.4735	0.4783	1			2.0	2.9	907
		FLC-CC	2	4	0.4876	0.4930	1					
	EG-CC		2	4	0.4789	0.4845	1			1.8	2.7	884
0.2	REGEC		2	6	0.4231	0.4401	1	0.8462	1.18	1.4	2.4	2048
	EGEC	UEC	2	4	0.3678	0.3956	1.1512			1.7	3.0	1596
		FLC-CC	3	4	0.3301	0.3390	1.2814					
	EG-CC		2	4	0.4231	0.4584	1			1.8	2.9	1578
H.265	REGEC		2	6	0.4393	0.4486	1	0.8786	1.3	1.5	2.6	724
	EGEC	UEC	2	4	0.4639	0.4652	1			1.8	2.9	588
		FLC-CC	2	4	0.3862	0.3955	1					
	UEC		2	4	0.3480	0.4249	1.2624			3.1	4.7	715
	EG-CC		2	4	0.4393	0.4961	1			2.4	3.3	558

benchmarker and for the UEC part of the EGEC benchmarker. Meanwhile, we employ an $r = 4$-state CC trellis in both the FLC-CC part of the EGEC benchmarker and in the EG-CC benchmarker, as recommended in [95, 110] and because using higher numbers of states was found to be detrimental in [1]. All of the schemes considered in this section employ URC inner codes, for the sake of facilitating iterative decoding. As discussed in Section 4.6.4, the selected REGEC codebook $\mathbb{C}_9$ has an EXIT curve that matches best with that of a URC code having 2 states, shown in Table 4.2. The EGEC, UEC and EG-CC benchmarkers also have EXIT curves that match best with a 2-state URC, since these were found to yield open EXIT chart tunnels at the lowest E_b/N_0 values in [110]. Therefore, we employ 2-state URCs for the inner codes of all schemes considered in this section. Note that the EGEC, UEC and EG-CC benchmarkers offer fair and natural comparisons with the proposed REGEC scheme, since they all employ simple unary, FLC or EG codewords, as well as trellis-based iterative decoding.

Table 4.3 provides the E_b/N_0 values where the DCMC capacity C becomes equal to the throughput η of each scheme considered. These E_b/N_0 values represent *capacity bounds*, above which it is theoretically possible to achieve reliable communication, provided that the scheme facilitates near-capacity operation. Furthermore, the specific E_b/N_0 values, where we have $A^\mathrm{i} = A^\mathrm{o}$ are provided for each scheme considered

in Table 4.3. These *area bounds* represent the lowest E_b/N_0 values, where it is theoretically possible to create an open EXIT chart tunnel, provided that the outer and inner EXIT curves have shapes that closely match each other. Note that the discrepancy between the capacity bound and the area bound of each scheme represents an E_b/N_0 *capacity loss*, as exemplified by Figure 4.8 for the REGEC, UEC and EG-CC schemes. As in the proposed REGEC code, the EXIT chart area A^o below the inverted UEC curve approaches the UEC coding rate R^o, when the number of states r is increased. By contrast, the EXIT chart area A^o below the inverted EG-CC EXIT curve is not affected by the number of states in the CC trellis, hence resulting in large discrepancies between A^o and R^o, therefore imposing significant amounts of *capacity loss*.

As shown in Table 4.3, the E_b/N_0 the *capacity loss* of all JSCC schemes is more significant for smaller p_1 values, indicating that trellises having higher numbers of states are required to mitigate *capacity loss* in these cases. However, these capacity losses are smaller than those of the SSCC EG-CC benchmarker, as shown in Table 4.3. For each of the source symbol distributions considered the capacity loss of the REGEC scheme is less than 0.3 dB, which is the smaller than the capacity loss of all the benchmarkers in each case, demonstrating that the proposed REGEC scheme facilitates near-capacity operation. Finally, Table 4.3 provides the *tunnel bound* of each scheme, which quantifies the lowest E_b/N_0 value, where an open EXIT chart tunnel can be created upon employing a two-state accumulator for the URC code, as it was discussed in Section 4.6.4.

The proposed REGEC schemes facilitate reliable communication at E_b/N_0 values that exceed the corresponding tunnel bound, provided that the symbol vector $\mathbf{d}$ comprises a sufficiently high number a of symbols. Note that higher E_b/N_0 values will be required to achieve low SERs, when employing short frames [117]. For all considered values of p_1 as well as for the H.265 distribution, our proposed REGEC scheme offers an open tunnel at the lowest E_b/N_0 values, facilitating low SERs at low E_b/N_0 values. At high E_b/N_0 values, the REGEC scheme will offer the widest open EXIT chart tunnel, requiring fewer decoding iterations to achieve a low SER than the benchmarkers.

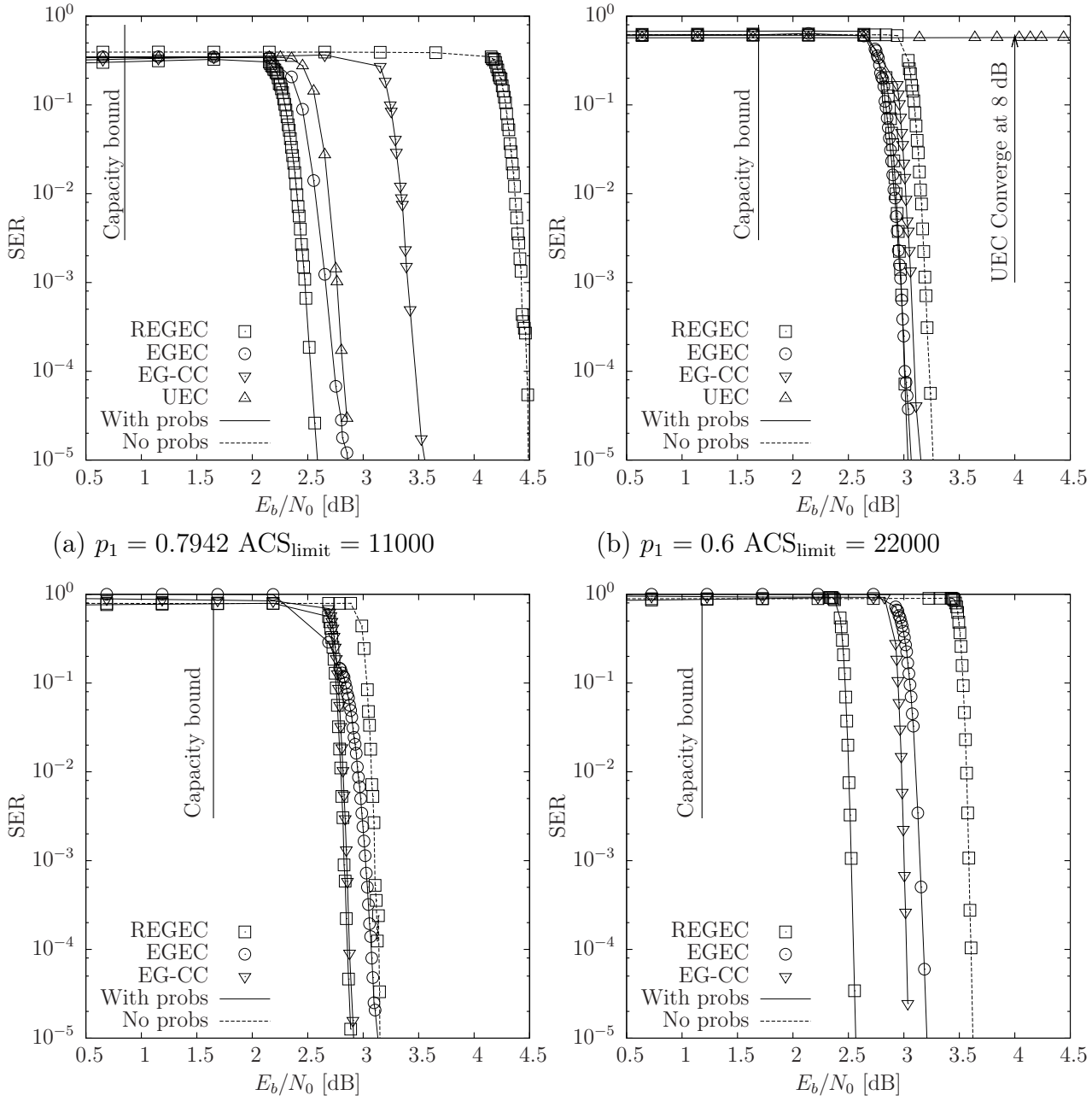
Table 4.3 also characterizes the complexity of all the schemes considered in this section. Here, the complexity is quantified by the average number of Add, Compare and Select (ACS) operations performed per decoding iteration and per symbol in the vector $\mathbf{d}$. This is justified, since the REGEC trellis decoder UEC trellis decoder, FLC decoder, CC decoder and the URC decoder operate entirely on the basis of addition,

subtraction and max* operations, which can be further decomposed into ACS operations. All other components in Figure 4.2 may be considered to have a relatively insignificant complexity [95,68]. As in [95], we assume that the addition and subtraction operations each require a single ACS operation, while each max* operation may be approximated by a look up table operation, which can be completed using five ACS operations [103]. As shown in Table 4.3, the complexity tends to increase as the Zeta distribution parameter p_1 is reduced, which may be explained by the resultant increases in the average codeword lengths l_{REG}, l_{EG} and l_{Unary}. Note that the complexity of the proposed REGEC scheme is higher than those of the benchmarkers, because the REGEC scheme employs an $r = 6$-state trellis, while all benchmarkers employ $r = 4$-state trellises. In order to make fair comparisons in Section 4.7.2, we will limit the number of decoding iterations performed by the proposed REGEC scheme, so that all schemes operate within the same overall complexity limits. These complexity limits will be chosen to be sufficient for the benchmarker having the lowest complexity to achieve an SER performance that is within 0.1 dB of the performance it can achieve with unlimited complexity. This facilitates a fair comparison by ensuring that the selected complexity limit is not sufficiently high to favour the schemes having the highest complexity, such as the proposed REGEC scheme.

4.7.2 SER comparison with the benchmarkers

Figures 4.11 and 4.12 characterize the SER performance of the schemes parametrized in Table 4.3. We consider the transmission of source symbol vectors $\mathbf{d}$ comprising $a = 2 \cdot 10^4$ symbols, which we found to be typical of the number of symbols in a H.265 [5] slice. Therefore, the SER performance of Figures 4.11 and 4.12 may be considered to be achievable without imposing any additional latency in multimedia applications.

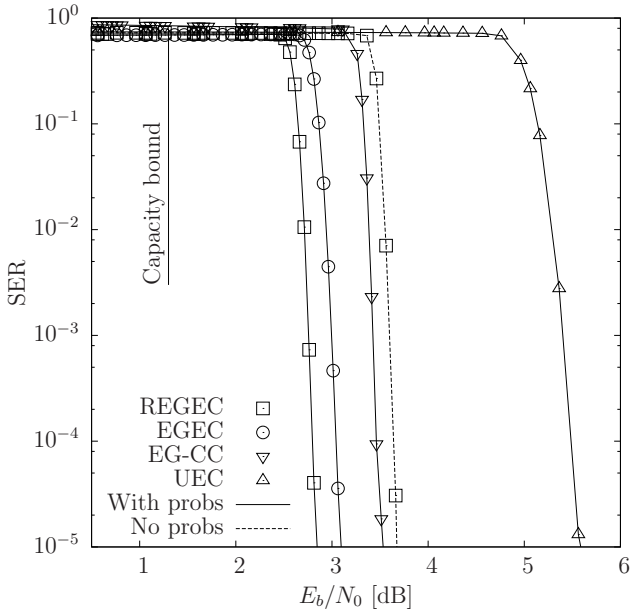
As shown in Figure 4.11, the proposed REGEC scheme facilitates reliable communication within 1.5 dB of the capacity bound and consistently offers the best SER performance for each of the finite Zeta-like distribution p_1 values considered. This consistency is a key benefit of the proposed REGEC scheme, because while it offers only a small gain over the best of the three benchmarkers in each case, the performance of these benchmarkers is particularly inconsistent. More explicitly, while the proposed REGEC scheme offers a gain of 0.4 dB over the UEC benchmarker for $p_1 = 0.7942$, this gain becomes 5 dB for $p_1 = 0.6$, owing to the severe puncturing that the UEC scheme requires in this case [95]. Similarly, while the proposed REGEC scheme offers only a marginal gain over the EGEC benchmarker for $p_1 = 0.6$, this gain becomes 0.8 dB for $p_1 = 0.2$, owing to the severe puncturing of the two parts of the



(a) $p_1 = 0.7942$ $\text{ACS}_{\text{limit}} = 11000$ (b) $p_1 = 0.6$ $\text{ACS}_{\text{limit}} = 22000$

(c) $p_1 = 0.4$ $\text{ACS}_{\text{limit}} = 44000$ (d) $p_1 = 0.2$ $\text{ACS}_{\text{limit}} = 66000$

Figure 4.11: The SER performance of the REGEC scheme and the UEC, EGEC and EG-CC benchmarkers of Figure 4.2, when transmitting frames comprising $a = 2 \cdot 10^4$ symbols that obey the finite Zeta-like distribution having the cardinality of $L = 1000$ and various $p_1$values, using QPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel. Iterative decoding continues until $\text{ACS}_{\text{limit}}$ number of ACS operation have been performed per symbol in the vector $\mathbf{d}$.



$\mathrm{ACS}_{\mathrm{limit}} = 25000$

Figure 4.12: The SER performance of the REGEC scheme and the UEC, EGEC and EG-CC benchmarkers of Figure 4.2, when transmitting frames comprising $a = 2 \cdot 10^4$ symbols that obey the H.265 distribution, using QPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel. Iterative decoding continues until $\mathrm{ACS}_{\mathrm{limit}}$ number of ACS operation have been performed per symbol in the vector $\mathbf{d}$.

EGEC benchmarker in order to achieve UEP [95], as described in Section 4.1. Note that the EGEC scheme has worse performance than the SSCC EG-CC benchmarker for $p_1 \in \{0.2, 0.4\}$. In the case of $p_1 = 0.2$, this may also be attributed to the severe puncturing invoked for UEP. In the case of $p_1 \in \{0.4, 0.6\}$, UEP does not improve the performance of the EGEC benchmarker, beyond that of the Equal Error Protection (EEP). Since our proposed REGEC scheme does not have two parts that must be carefully balanced, it does not suffer from these problems. Similarly, while the proposed REGEC scheme offers only a marginal gain over the EG-CC benchmarker for $p_1 = 0.4$, this gain becomes 0.6 dB for $p_1 = 0.2$ and 0.9 dB for $p_1 = 0.7942$, as shown in Figure 4.11.

In the case where the source symbols obey the H.265 distribution of Figure 4.3, our REGEC scheme offers a gain of 0.7 dB over the SSCC EG-CC benchmarker, as shown in Figure 4.12 . Furthermore, our REGEC scheme offers 0.3 dB gain over the EGEC benchmarker, where UEP does not improve the performance of the EGEC benchmarker in this scenario. The UEC benchmarker has the worst performance of all the schemes considered in this scenario, owing to the severe puncturing that it requires to achieve the same effective throughput as the other schemes.

Note that since the SER results of Figures 4.11 and 4.12 offer fair comparisons in terms of complexity and effective throughput, the gains offered by our proposed

REGEC scheme are obtained for free, with no cost in terms of transmit-duration, transmit-bandwidth, transmit-energy or decoding complexity. Therefore, these gains of up to 0.9 dB can be considered to be significant, particularly since they are achieved within about 1.5 dB of the E_b/N_0 capacity bound.

Note that throughout our discussions above, it was assumed that the receiver of the proposed REGEC scheme has knowledge of the average REG codeword length l. Furthermore, it was assumed that the decoder has knowledge of the probabilities of occurrence $P(d)$. However, Figure 4.11 and 4.12 show that when the channel SNR is sufficiently high, the proposed REGEC receiver facilitates a low SER, even if it does not have any knowledge of the symbol probabilities $P(d)$. The symbol probabilities may be estimated by storing a sufficient number of symbol vectors $\hat{\mathbf{d}}$, in order to heuristically estimate the required information, hence facilitating near-capacity communication for the subsequent symbol vectors.

4.8 Conclusions

In this chapter, we have proposed a novel REGEC code for the near-capacity transmission of symbol values that are randomly selected from a source set having a large or infinite cardinality. In contrast to the UEC code previously proposed for the same purpose, our REGEC code is a universal code, facilitating the transmission of symbol values that are randomly selected using any mototonic probability distribution. On the other hand, in contrast to the EGEC code previously proposed for the same purpose, our REGEC code has a simple structure, which solves the delay, synchronization and computational complexity problems associated with the two parts of the EGEC code. In particular, the EGEC code must be specifically parametrized for operation in conjunction with the particular source distribution, preventing its application for unknown or non-stationary sources. By contrast, the proposed REGEC code can be applied for any distribution, without requiring specific parametrization.

In Section 4.2, we described the Zeta source probability distribution and we adapted the infinite cardinality source alphabet of our previous work to the case of a finite cardinality, where this cardinality represents an additional parameter to be considered. In Section 4.3, we introduced the novel REG code and described the structure of the REG codewords. Section 4.4 and 4.5 introduced our novel REGEC encoder and decoder, repectively. In Section 4.6, we analyzed the parametrization of the proposed REGEC scheme and demonstrated that it facilitates near-capacity operation. In Section 4.7, we considered a wide range of finite Zeta-like probability distributions as well as the H.265 distribution and we showed that our REGEC scheme is capable of offering gains over the best of the UEC, EGEC and SSCC

benchmarkers in each case, when employing QPSK for communication over an uncorrelated narrowband Rayleigh fading channel. In some practical scenarios, where the source symbols obey particular finite Zeta-like probability distributions, our REGEC scheme was shown to offer gains of up to 0.9 dB over the best benchmarkers. In the scenario where the source symbols obey the H.265 distribution, our REGEC scheme was shown to offer a gain of 0.7 dB over the SSCC benchmarker. These gains are achieved for free, without increasing the required transmit-duration, transmit-bandwidth, transmit-energy or decoding complexity.

Chapter 5

Conclusions and Future Work

In this thesis, we have introduced several novel Joint Source and Channel Code (JSCC) schemes in order to exploit the residual redundancy that typically remains following the source coding process, which would otherwise cause capacity loss. As we described in Section 2.2, the symbols generated by multimedia codecs, such as H.264 and H.265 typically have values that are selected from an alphabet having a large cardinality, which may be modeled using a Zeta-like probability distribution. In order to exploit the knowledge of the Zeta-like probability distribution in the receiver to aid channel decoding, we developed a low-complexity *universal* Reordered Elias Gamma Error Correction (REGEC) code through Chapters 2 to 4. In Section 5.1, we will proceed by presenting a summary of each chapter and the main findings of our investigations. In Section 5.2, we will discuss a range of tangible design guidelines, while in Section 5.3 we will provide a range of future research ideas.

5.1 Summary and conclusions

Chapter 1 portrayed the background of the research presented in this thesis. In Section 1.1, we briefly reviewed the history of video compression techniques and discussed the intra-frame and inter-frame correlations that exist in an uncompressed video sequence. Multimedia codecs are capable of exploiting most of this correlation in order to achieve compression, although typically some residual redundancy remains following this process. Separate Source and Channel Code (SSCC) and JSCC schemes are discussed in Section 1.2.1. According to Shannon's source and channel separation theorem, a SSCC is capable of near-capacity transmission, provided that a potentially unlimited encoding/decoding delay and complexity can be afforded. However, it is typically impossible to remove all source redundancy with the aid of practical finite-delay and finite-complexity source encoding. As a remedy, various JSCC schemes

have been proposed for mitigating the capacity loss imposed by the residual source redundancy. Despite their benefits, JSCCs have only found limited applications in practice, since their decoding complexity increases rapidly with the cardinality of the symbol set.

Against this background, in Chapters 2, 3 and 4 we considered the family of Unary Error Correction (UEC), Elias Gamma Error Correction (EGEC) and the REGEC codes, respectively. Their applications, characteristics and performance was investigated in these chapters. Figure 5.1 lists the components to be considered

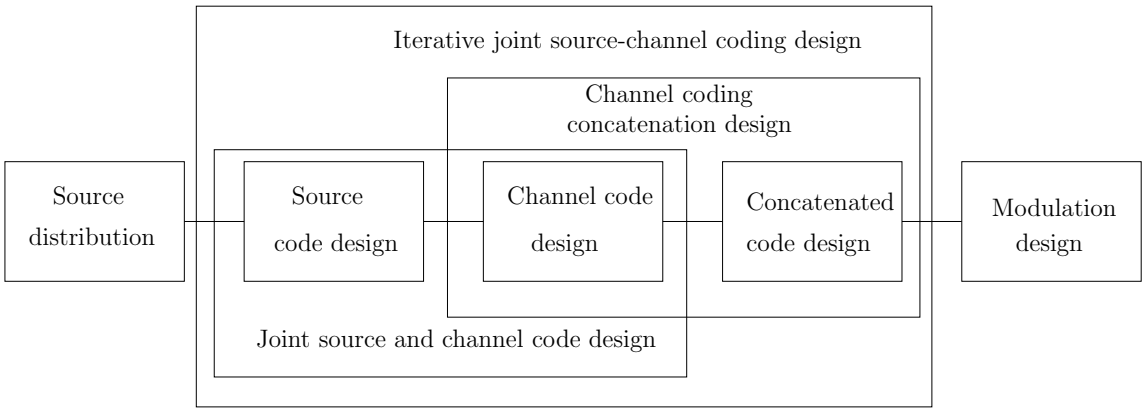


Figure 5.1: The components that must be considered when designing an iterative joint source and channel coding scheme.

when designing an iterative JSCC scheme. Note that the operation of our proposed JSCC schemes follows similar principles to each other. Hence Chapters 2, 3 and 4 considered similar subsets of the components portrayed in Figure 5.1. The important conclusions of these chapters are as follows.

In Chapter 2, we reviewed the encoding and decoding operations of the UEC code and exemplified the application of the UEC code in the context of a serially concatenated iterative decoding scheme. This chapter also served as a background chapter, since it introduced the principles that are common to our family of JSCC codes. In Section 2.2, we detailed the collection of video data used and modeled its symbol value probability distribution using the Zeta probability distribution. Figure 2.2 illustrates the structure of the UEC code. We described the operation of the unary encoder and the trellis of the UEC encoder in Section 2.3. Furthermore, we described how to integrate the UEC encoder into a transmitter by describing the operation of a concatenated Unity Rate Convolutional (URC) encoder, as well as of the interleaver and modulator. Section 2.5 described the operation of the UEC decoder and URC decoder, providing the equations of the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm, which forms the basis of the UEC trellis and of the URC decoding processes. The EXIT chart and its area properties exemplified in

the context of the UEC code were characterized in Section 2.6. Figure 2.6 shows that in the case of Zeta distributions having a parameter value $p_1 > 0.608$, the capacity loss asymptotically approaches zero, as the affordable complexity of the UEC trellis is increased. Additionally, we discussed the near-capacity operation of the UEC and its parameterization, including the selection of the number of trellis states and the codebook design. Figures 2.23 and 2.24 indicate that having $r = 4$ UEC trellis states represents an attractive trade-off between maintaining a low trellis complexity and facilitating near-capacity operation, while Figure 2.27 shows that the codebook $\mathbb{C} = [00; 01]$ has the best performance for a $r = 4$-state UEC trellis. In Section 2.7, the performance of the SSCC EG-CC-URC benchmarker was compared to that of the proposed UEC scheme for the case of Zeta distributed source symbols having different values of the parameter p_1. Figure 2.31 demonstrated that our proposed UEC scheme outperforms the EC-CC benchmarker of Figure 2.28, offering a 1.6 dB gain.

In Chapter 3, we proposed a *universal* JSCC EGEC code. In contrast to the UEC code of Chapter 2, the EGEC code achieves a finite average codeword length for all Zeta distributions, not just those having $p_1 > 0.608$. As described in Section 3.2 and motivated by the observation that each EG codeword is comprised of a unary codeword prefix and a Fixed Length Code (FLC) post-fix, the EGEC encoder decomposes each input symbol into two sub-symbols, which are encoded separately by two distinct sub-encoders, as shown in Figure 3.2. The first sub-encoder is referred to as the EGEC(UEC) encoder, which operates in the same manner as the UEC encoder of Chapter 2. The second sub-encoder employs a serial concatenation of a FLC and of a Convolutional Code (CC) encoders, which we refer to as the EGEC(FLC-CC) encoder. As described in Section 3.3, the EGEC decoder has corresponding sub-decoders, which operate on the basis of the Log-BCJR algorithm and the Soft Bit Source Decoding (SBSD) algorithm. As shown in Figure 3.8, an open EXtrinsic Information Transfer (EXIT) chart tunnel is created by the EGEC(UEC) code at an E_b/N_0 of 1.9 dB in a particular practical scenario, but by contrast, this is not facilitated until reaching an E_b/N_0 of 2.9 dB for the EGEC(FLC-CC) code. Thus the parametrization of our proposed EGEC code has to optimize the relative contribution of the two EGEC sub-codes to the encoding process. In Section 3.4, we detailed the procedure of designing an Unequal Error Protection (UEP) scheme that optimizes these contributions, facilitating near-capacity operation at a low decoder complexity. Furthermore, the parametrizations of our EGEC scheme as well as of the benchmarkers were introduced in Section 3.4. We demonstrated in Section 3.5 that when the source symbols obey a particular Zeta probability distribution, our

EGEC scheme offers a 3.4 dB gain over a UEC benchmarker, when Quaternary Phase Shift Keying (QPSK) modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel. In the case of another Zeta probability distribution, our EGEC scheme was shown to offer a 1.9 dB gain over the SSCC EG-CC benchmarker, as shown in Figure 3.9. Furthermore, we considered a wide range of Zeta probability distributions and our EGEC scheme was found to offer gains over the relevant benchmarkers in each case. Additionally, Figure 3.10 shows that when the source symbols obey the H.265 distribution of Figure 2.3(b), our EGEC scheme offers a 0.4 dB gain over the EG-CC benchmarker.

In Chapter 4, we proposed a *universal* JSCC REGEC code, which has a significantly simpler structure than the EGEC code of Chapter 3. This avoids the requirement of tailoring UEP for the particular scenario considered and reduces capacity loss and the complexity by avoiding puncturing. In Section 4.2, we described the Zeta source probability distribution and we generalised the infinite cardinality source alphabet of our previous chapters to the case of a finite cardinality, where this cardinality represents an additional parameter to be considered. In Section 4.3, we introduced the novel Reordered Elias Gamma (REG) code and described the structure of the REG codewords. Section 4.4 and 4.5 introduced our novel REGEC encoder and decoder, respectively. In Section 4.6, we analyzed the parametrization of the proposed REGEC scheme and demonstrated that it facilitates near-capacity operation. In Section 4.7, we considered a wide range of finite Zeta-like probability distributions as well as the H.265 distribution and we showed that our REGEC scheme is capable of offering gains over the best of the UEC, EGEC and EG-CC benchmarkers in each case, when employing QPSK for communication over an uncorrelated narrowband Rayleigh fading channel. In some practical scenarios where the source symbols obey particular finite Zeta-like probability distributions, our REGEC scheme was shown to offer gains of up to 0.9 dB over the best benchmarkers. In the scenario where the source symbols obey the H.265 distribution, our REGEC scheme was shown to offer a gain of 0.7 dB over the EG-CC benchmarker. These gains are achieved for free, without increasing the required transmit-duration, transmit-bandwidth, transmit-energy or decoding complexity.

5.2 Design Guidelines

Based on the above-mentioned investigations, this section summarizes the general design guidelines of each component that must be considered when designing an iterative joint source and channel coding scheme, as depicted in Figure 5.1,

5.2.1 Source distribution

The design of a JSCC scheme should start by analyzing the source distribution. Figure 2.3 demonstrates that both the H.264 and H.265 video encoders produce symbol values that may be represented using positive integers having values of up to around 1000, where higher values are observed with lower probabilities. Note that the H.264 and H.265 probability distributions have a roughly constant gradient, when plotted on the log-log axes of Figure 2.3. Therefore, these symbol values obey Zipf's law and their distribution may be approximated by the Zeta distribution. The UEC, EGEC and REGEC codes of Chapters 2 to 4 are designed for conveying a vector $\mathbf{d} = [d_i]_{i=1}^{a}$ comprising a number of Zeta-distributed symbols. This symbol vector is obtained as the realization of a corresponding vector $\mathbf{D} = [D_i]_{i=1}^{a}$ of Independent and Identically Distributed (IID) Random Variables (RVs). In Chapter 4, the source distribution was generalized to the case where each RV D_i adopts the symbol value $d \in \mathbb{N}_L$ with probability $\Pr(D_i = d) = P(d)$, where $\mathbb{N}_L = \{1, 2, 3, \ldots L\}$ is the finite-cardinality alphabet comprising positive integers with the cardinality L. In Chapter 4, we characterized our proposed REGEC code using symbol values obeying both the finite Zeta-like distribution and the H.265 distribution. We considered the operation of the decoder both with and without knowledge of the source distribution. As discussed in Section 4.6, some of the REGEC codebooks result in an error floor, when the source distribution is unknown. *Therefore, the design of a JSCC scheme must be based on the source distribution analysis, considering whether the source distribution is known or unknown.*

Source distribution	• Zeta distribution of (2.2) • Zeta-like distribution of (4.1)	Distribution state	• Stationary • Non-stationary
Distribution parameter	• p_1 • L	Distribution parameter availability	• Known • Unknown
Distribution parameter estimation	• Online • Offline		

Table 5.1: List of considerations for source distribution analysis

5.2.2 Source code design

- Entropy coding

  After analyzing the source distribution, a suitable entropy code must be selected for the joint source and channel code design. The unary code was chosen as the

basis of the JSCC UEC scheme, since its codewords have a relatively simple structure, which can be readily exploited for error correction. However, the unary code is not a *universal* code, which limits the applicability of the UEC code to only a subset of the source ditribution, which does not include the Zeta distribution that most closely models the source symbols produced by H.264 and H.265. By contrast, the conventional Elias Gamma (EG) codewords have a relatively complex structure, which cannot be readily described by a single trellis and hence cannot be readily exploited for low-complexity error correction in conjunction with a simple JSCC structure. By contrast, our proposed REG code is a *universal* code associated with a simple structure. *Likewise, the design of future JSCC schemes should be based on universal source codes that have a simple structure, which can be readily described by a single trellis.*

Source code	• unary of Table 2.2 • EG of Table 3.1 • REG of Table 4.1	• ExpG • ExpG		• Universal • Not Universal
Source code Parameter	• average codword length of (2.5), (3.5) and (4.5)			

Table 5.2: List of considerations for entropy coding design

5.2.3 Channel code design

The relatively simple structure of the REG codewords proposed in Chapter 4 may be described by a simple trellis structure employed by the REGEG trellis encoder and decoder. As discussed in Section 2.1.1, the complexity of Variable Length Error Correction (VLEC) codes and of other previously proposed JSCCs increases rapidly with the cardinality of the symbol value set. By contrast, the decoding complexity of the REGEC trellis decoder is independent of the cardinality of the symbol value set. Furthermore, our REGEC code employs a single trellis that avoids the delay, synchronization and computational complexity problems, which may be imposed by the EGEC code. *A practical JSCC design must employ a channel code having a relatively low-complexity with a simple structure.*

	• Number of states • Codeword design • Free distance	• Codeword length • EXIT function • Complexity

Table 5.3: List of consideration for channel code design

5.2.4 Concatenated code design

Near-capacity operation is facilitated when our REGEC code is serially concatenated and iteratively decoded by exchanging extrinsic information with a URC code, as described in Chapter 4. EXIT chart analysis may be used for beneficially selecting the inner concatenated code. We recommend the employment of URC Linear Feedback Shift Registers (LFSRs) having generator polynomials of the form $[1, 0, \ldots, 0]$ and feedback polynomials of the form $[1, 1 \ldots, 1]$. This is because this kind of URC outputs non-zero *extrisic* Logarithmic Likelihood Ratios (LLRs), when provided with an all-zero *a priori* LLR vector.

	• Number of states	• Puncturing
	• Interleaver design	• EXIT function
	• Interleaver length	• Complexity

Table 5.4: List of to consideration for concatenated code design

5.2.5 Modulation design

M = 4-ary Gray-coded QPSK modulation was employed in Chapters 2, 3 and 4. *The classic Gray Mapping (GM) is employed here, since the performance of Gray-coded QPSK is superior to that of QPSK with Anti-Gray Mapping (AGM), when no iterations are performed between the demapper and URC decoder in the receiver [96].*

	• Number of constellation points	• Iteration
	• Position of constellation points	• DCMC capacity
	• Mapping	

Table 5.5: List of considerations for modulation design

5.2.6 Summary

In this section, we have summarized the design guidelines of an iterative JSCC scheme. The five salient considerations are: the source distribution, source code design, channel code design, concatenated code design and the modulation design. The design of a JSCC scheme should commence by characterizing the source distribution. Equipped with the knowledge of the source distribution, a source code having a relatively simple structure may be selected for the JSCC design. A *universal* code may be selected for ensuring the wide applicability of the JSCC scheme. Following this, a trellis code having a low decoding complexity may be designed for describing

the source code. This trellis code may then be combined with a concatenated code, in order to achieve near-capacity performance with the aid of iterative decoding. In the case of reconstructing uncompressed video, the attainable performance can be improved by exploiting both the inter-frame and intra-frame correlation in the video frames.

The above-mentioned design guidelines have been demonstrated in the context of the novel UEC, EGEC, REGEC and DVC schemes of Chapters 2 to 4, respectively. These facilitate the practical near-capacity joint source and channel coding of multimedia information.

5.3 Future work

The following section discuss several promising avenues for future work in order to extend the schemes proposed in Chapters 2 to 4.

5.3.1 Learning-aided REGEC code

Figure 4.11 shows that the proposed REGEC scheme does not require any knowledge of the symbol occurrence probabilities at either the transmitter or receiver, when the channel Signal to Noise Ratio (SNR) is sufficiently high. However, having no knowledge about the symbol occurrence probabilities at the receiver causes nonetheless some capacity loss. If the receiver were capable of estimate the occurrence probabilities of the most frequently occurring symbol values, reliable communication at near-capacity SNRs would be facilitated for both unknown and non-stationary source probability distributions. A novel learning-aided UEC scheme was proposed in [118][1], which was designed for transmitting symbol values selected from unknown and non-stationary probability distributions. In [118], the learning algorithm was implemented using a memory storage at the receiver, which uses symbols recovered from previous frames for estimating the symbol probabilities of each received frame. In our future work, this technique may be used by the REGEC code for improving its performance when transmitting symbols selected from unknown and non-stationary source probability distributions.

5.3.2 Reordered Exponential Golomb Error Correction code

As we mentioned in Section 2.1.1, the Exponential Golomb (ExpG) code of [29] is parametrized by the non-negative integer parameter k, where $k = 0$ represents the special case of the EG code. The codewords of the ExpG code are shown in Table

[1]The author of this treatise contributed the source distribution statistics and to the simulations of this paper.

Table 5.6: The first twelve codewords of various source codes

d_i	Unary(d_i)	EG(d_i)	REG(d_i)	ExpG(d_i) k=1	RExpG(d_i) k=1	RExpG(d_i) k=2	RExpG(d_i) k=2
1	1	1	1	10	01	100	001
2	01	010	001	11	11	101	011
3	001	011	011	0100	0001	110	101
4	0001	00100	00001	0101	0011	111	111
5	00001	00101	00011	0110	1001	01000	00001
6	000001	00110	01001	0111	1011	01001	00011
7	0000001	00111	01011	001000	000001	01010	01001
8	00000001	0001000	0000001	001001	000011	01011	01011
9	000000001	0001001	0000011	001010	001001	01101	10011
10	0000000001	0001010	0001001	001100	100001	01110	11001
11	00000000001	0001011	0001011	001101	100011	01111	11011
12	000000000001	0001100	0100001	001111	101011	0010000	0000001
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

5.6 for $k = 1$ and $k = 2$. An ExpG codeword ExpG(d_i) has a length of $l_{ExpG}(d_i) = 2\lfloor \log_2(d_i + 2^k - 1) \rfloor + 1 - k$. ExpG source coding is beneficial for the coding of Zeta distributed source symbols having low p_1 values. More specifically, when p_1 is low,

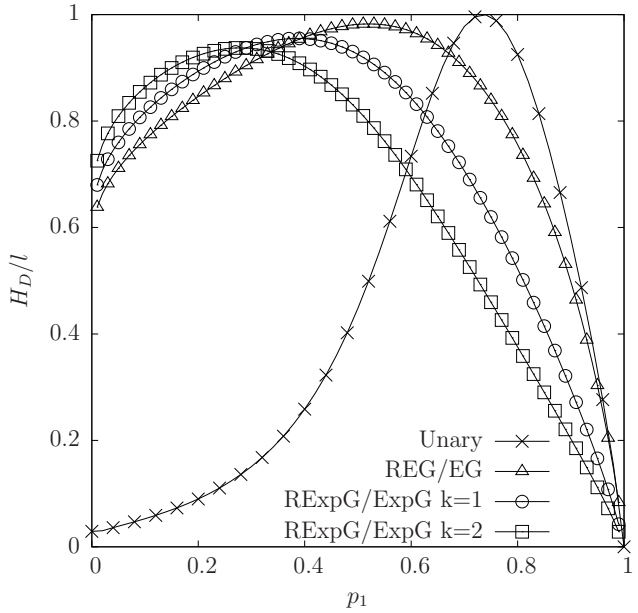


Figure 5.2: Plots of H_D/l that are obtained for various source codes, in the case where the symbol values of $\mathbf{d}$ obey a finite Zeta-like distribution having the parameter p_1 and cardinality of $L = 1000$.

lower ExpG average codeword lengths l_{ExpG} are achieved by using higher values of the parameter k, as shown in Figure 5.2. However, like the EG code, the rest of the codes in the ExpG family have a complex structure, which cannot be readily represented by a single trellis structure. Motivated by this, we split the ExpG codewords into two parts in [119][2], by extending the technique of Section 3.2.1. In this way, we were able to develop the Exponential Golomb Error Correction (ExpGEC) code of [119], by extending the EGEC code of Chapter 3. Furthermore, [119] demonstrated that ExpGEC associated with $k = 1$ offers a superior error correction performance over the EGEC in the case of Zeta distributions having low $p_1$values, as well as in the case

[2]The author of this treatise contributed the source distribution statistics presented in this paper.

of the H.265 distribution. However the ExpGEC code of [119] suffers from a complex structure similar to that of the EGEC code of Chapter 3, which may impose delay, as well as synchronization and computational complexity problems. Motivated by this, future research could extend the REG code of Chapter 4 to propose a RExpG code, which has a structure that can be more readily represented by a single trellis than that of the ExpG code. The generalized structures of each Reordered Exponential Golomb (RExpG) codeword for $k = 1$ and $k = 2$ are shown in Figure 5.3. The RExpG

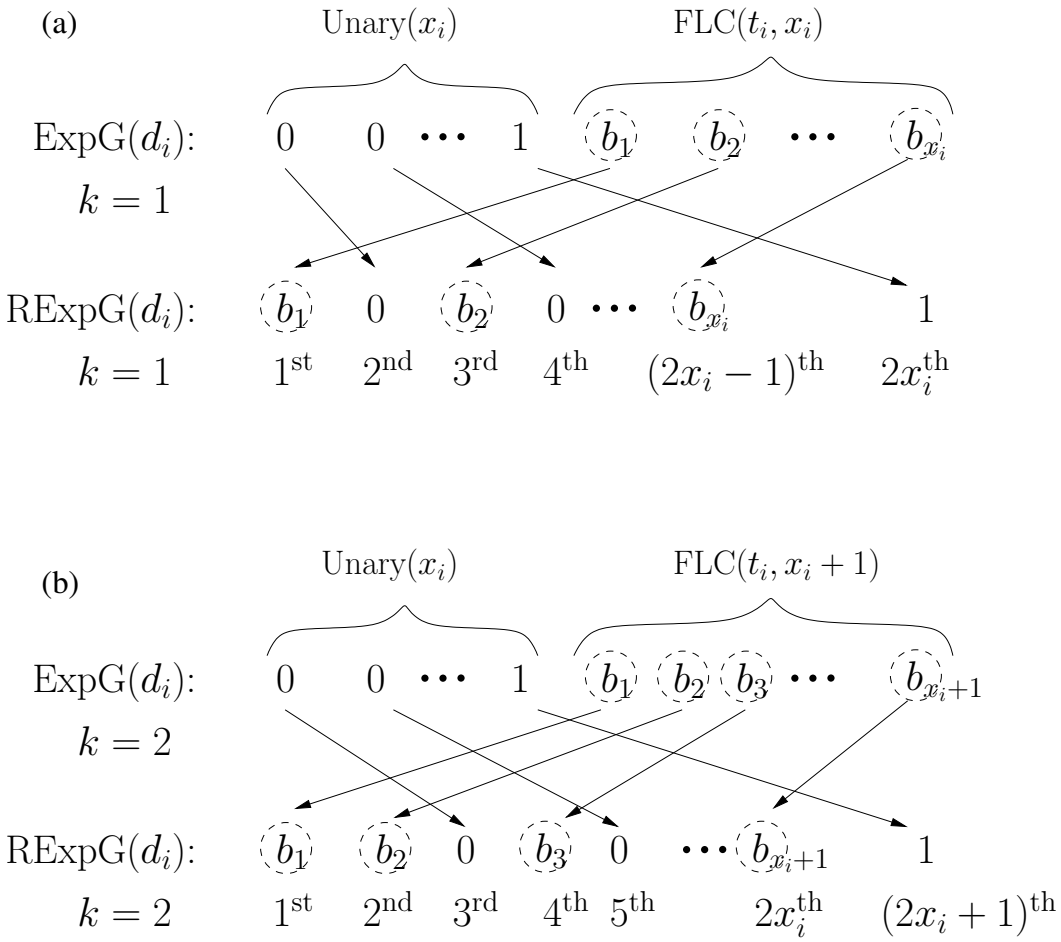


Figure 5.3: The reordering of (a) an ExpG $k = 1$ codeword and (b)(a) an ExpG $k = 2$ codeword to obtain the corresponding RExpG codeword in conjunction with $k = 1$

reordering rules in Figure 5.3 and 4.4 can be generalized as follows. Each codeword of RExpG starts with the first k bits from the FLC postfix $FLC(t_i, x_i + k - 1)$, followed by the first bit from the unary prefix $Unary(x_i)$ of the corresponding ExpG codeword. The rest of the RExpG codeword is formed by alternately selecting the remaining bits from the FLC post-fix $FLC(t_i, x_i + k - 1)$ and the unary prefix $Unary(x_i)$. Notice that the final 1-valued bit from the unary prefix $Unary(x_i)$ will always become the final bit in RExpG(d_i).

The RExpG code proposed above has a simple structure that may be readily described by the RExpGEC trellis of Figure 5.4 . This may be employed as the basis

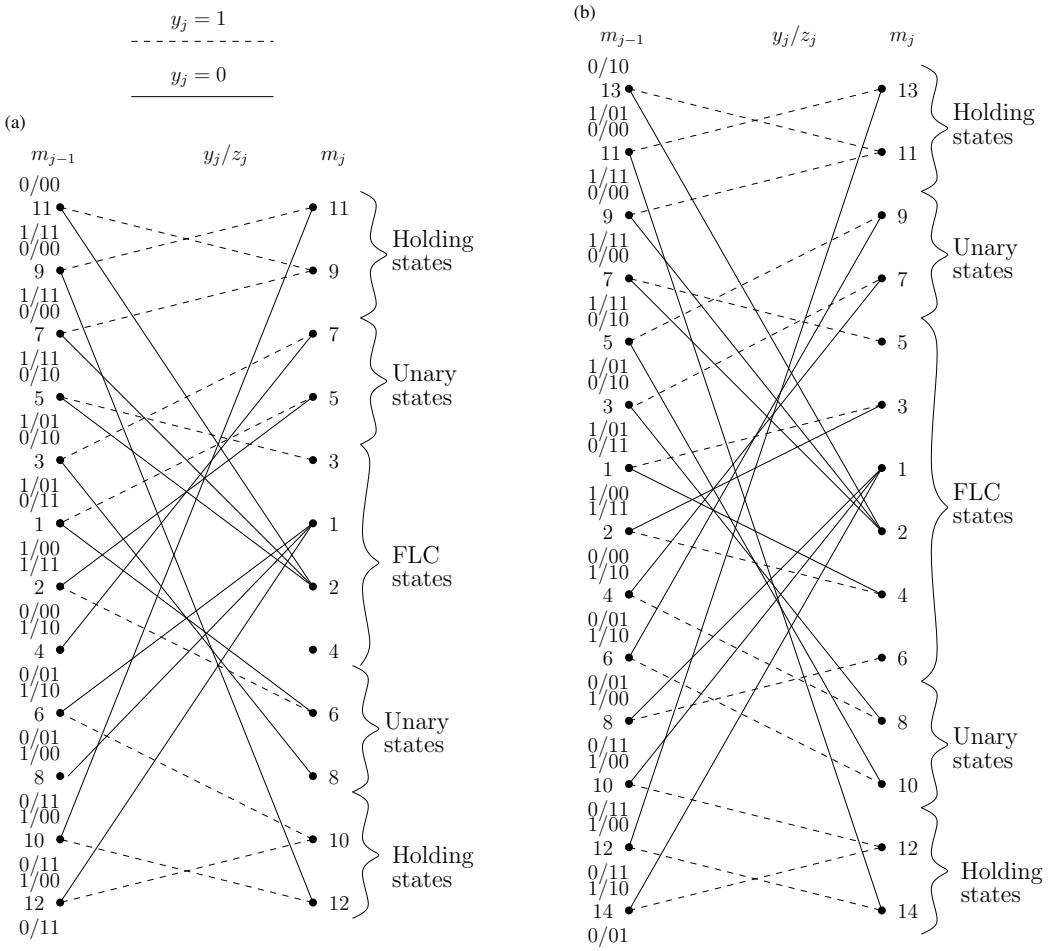


Figure 5.4: (a) An $r = 12$-state $n = 2$-bit Reordered Exponential Golomb Error Correction (RExpGEC) trellis in conjunction with $k = 1$ using the codebook $\mathbb{C} = [00; 01; 01; 11; 11; 11]$. (b) An $r = 14$-state $n = 2$-bit RExpGEC trellis in conjunction with $k = 2$ using the codebook $\mathbb{C} = [00; 01; 01; 11; 11; 11; 01]$.

of an RExpGEC encoder and decoder in analogy to the REGEC code of Chapter 4. Note that the initial state of the RExpGEC trellis is an FLC state, which is in contrast to the REGEC trellis of Chapter 4, because that starts from a unary state. This is because in contrast to the REG codewords, each RExpGEC codeword starts with an FLC bit, as illustrated in Figure 5.3. Furthermore, the transitions from states 1 and 2 of the RExpGEC trellis associated with $k = 2$ are from a FLC state, to another FLC state as shown in Figure (a), since the codewords of RExpG having $k = 2$ start with two consecutive FLC bits.

While the discussions above provide the basis of designing the JSCC RExpGEC code, several future research issues have to be solved to complete the RExpGEC code design. In a first step, it is necessary to calculate the conditional transition

probabilities $P(m|m')$, in analogy to Equations (2.7) and (4.11). Following this, we have to design the concatenated code with the aid of EXIT chart analysis using the guidelines of Section 5.2.4. Finally, we will have to perform the candidate codebook selection, in analogy to that of Section 4.6.3.

5.3.3 REGEC-turbo scheme

An *adaptive* UEC-turbo scheme was proposed in [99], comprising a three-stage concatenated architecture that employs an adaptive iterative decoding technique for expediting iterative decoding convergence. Owing to the similarity between the UEC code and the REGEC code, the adaptive technique of [99] may be readily extended to the corresponding Adaptive-REGEC-turbo scheme of Figure 5.5. Note that since the

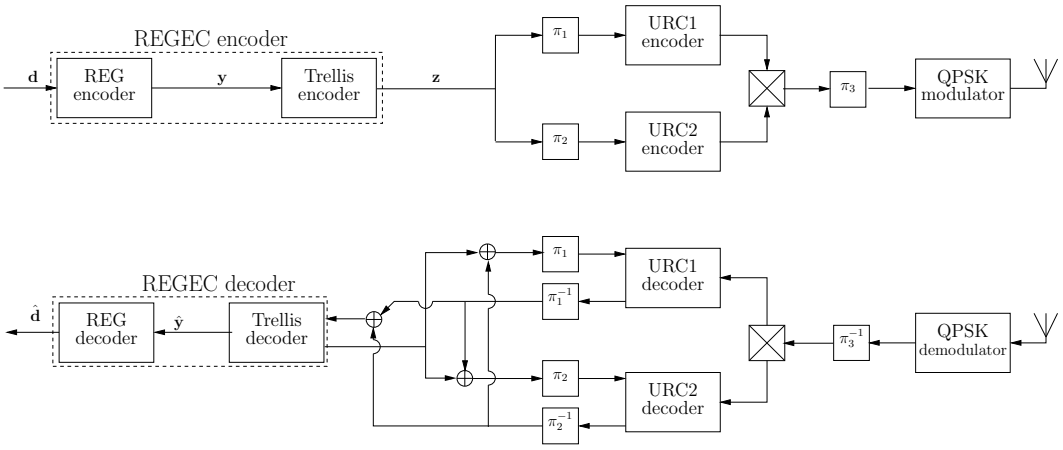


Figure 5.5: Schematic of the REGEC-turbo scheme, which facilitates adaptive iterative decoding.

turbo code of Figure 5.5 provides a strong error-correction capability, the REGEC can afford a higher coding rate, which may be achieved using $n = 1$-bit codewords in the REGEC trellis encoder. Future work may consider the parametrization of these $n = 1$-bit codewords, in order to maximise the resultant error correction performance. Furthermore, the REGEC codebook may be adaptively extended in the receiver in order to allow a larger number of REGEC trellis states to be used, as discussed in Section 4.6.1. This codebook extension technique allows the number of states employed in the REGEC trellis decoder to be dynamically selected, in order to strike an attractive trade-off between the decoding complexity and the error correction capability. Furthermore, the three-stage concatenation of the REGEC code with a turbo code may be controlled using the Three-Dimensional (3D) EXIT chart analysis concept proposed in [99]. More specifically, this may be used for controlling the dynamic adaptation of the number of states employed by the REGEC trellis decoder, as well as for controlling the decoder activation order between the REGEC decoder

and the turbo decoder. This will guarantee a strong error correction capability at a significantly reduced complexity. Furthermore, the REGEC-turbo scheme of Figure 5.5 may be used as the basis of interfacing the REGEC code with standardized communication systems based on turbo codes.

5.3.4 Iterative demodulation

$M = 4$-ary Gray-coded QPSK modulation was employed in Chapters 2, 3 and 4. The classic GM is employed here, since the performance of Gray-coded QPSK is superior to that of QPSK combined with AGM, when no iterations are performed between the demapper and URC decoder in the receiver [96]. Note however that if the higher complexity of iterative demodulation can be afforded, then other mapping schemes or a modulation scheme having a higher M may be employed instead. Motivated by this, AGM-based QPSK was employed in [98] and the corresponding demodulator was serially concatenated and iteratively exchanged its extrinsic information with a UEC-turbo scheme [99]. Using this approach, the REGEC scheme of Chapter 4 and of Section 5.3.3 may be enhanced with the aid of other mapping schemes or a modulation scheme having a higher complexity, in order to achieve an improved error correction capability.

Appendix A

Derivation of the REGEC transition probability

Derivation of (4.11)

The method of [1, Appendix] may be used to derive the transition probabilities $P(m, m')$ of (4.11) by observing the expected number of transitions of each type when encoding symbols in the vector $\mathbf{d}$. More specifically, a transition from a unary state $m' \in \{1, 2, \ldots, f\}$ to a FLC state $m = m' + f$ will occur for each symbol in the vector $\mathbf{d}$ satisfying $d_i \geq 2^{\lceil m'/2 \rceil}$. The number of symbols in the vector $\mathbf{d}$ that satisfy this conditions has an expected value of

$$\begin{aligned} & \tfrac{a}{2} \sum_{d=2^{\lceil \frac{m'}{2} \rceil}}^{L} P(d) \\ = \; & \tfrac{a}{2} \left[\sum_{d=1}^{L} P(d) - \sum_{d=1}^{2^{\lceil \frac{m'}{2} \rceil}-1} P(d) \right] \\ = \; & \tfrac{a}{2} \left[1 - \sum_{d=1}^{2^{\lceil \frac{m'}{2} \rceil}-1} P(d) \right]. \end{aligned}$$

Therefore we may expect half of this number of the transitions in the path $\mathbf{m}$ to be of each of the above-mentioned types on average, since the trellis is symmetric and the transitions where $\mathrm{odd}(m') = 0$ and $\mathrm{odd}(m') = 1$ are equiprobable.

Similarly, a transition from a unary state $m' \in \{1, 2, \ldots, f\}$ to a unary state $m = 1 + \mathrm{odd}(m')$ will occur for each symbol in the vector $\mathbf{d}$ satisfying $\lfloor \log_2(d_i) \rfloor = \lceil m'/2 \rceil$. We can expect $\left[\frac{a}{2} \cdot \sum_{d=2^{\lceil m'/2 \rceil - 1}}^{2^{\lceil m'/2 \rceil}-1} P(d) \right]$ of the symbols in the vector $\mathbf{d}$ to satisfy these

conditions and therefore we can expect half of this many of the transitions in the path $\mathbf{m}$ to be of each of the above-mentioned types.

Furthermore, a transition from an FLC state $m_{j-1} \in \{f+1, f+2, \ldots, 2f-2\}$ to a unary state $m = \ddot{d} - y_j + 2y_j \cdot \mathrm{odd}(m')$ will occur for each symbol in the vector $\mathbf{d}$ satisfying $d_i \geq 2^{\lceil (m'-f)/2 \rceil}$. We can expect $\left\lceil \frac{a}{2} \cdot \sum_{x=\ddot{x}+1}^{\lfloor \log_2(L) \rfloor} \sum_{\ddot{t}=0}^{2^{\ddot{x}}-1} \mathrm{odd}(\ddot{t}+1+y_i) \sum_{d=2^x+\ddot{t}\frac{2^x}{2^{\ddot{x}}}}^{2^x+(\ddot{t}+1)\frac{2^x}{2^{\ddot{x}}}-1} P(d) \right\rceil$ of the symbols in the vector $\mathbf{d}$ to satisfy these conditions and therefore we can expect half of this many of the transitions in the path $\mathbf{m}$ to be of each of the above-mentioned types.

Moreover, a transition from a holding state $m' \in \{2f-1, 2f\}$ to a holding state $m = m' + y_j \cdot (2 \cdot \mathrm{odd}(m') + 1) + 2 \cdot \mathrm{odd}(y_j + 1)$ will occur for each symbol in the vector $\mathbf{d}$ satisfying $d_i \geq 2^{\lceil f/2 \rceil}$. We can expect $\left\lceil \frac{a}{2} \cdot \sum_{x=f}^{\lfloor \log_2(L) \rfloor} \sum_{\mathring{x}=f}^{x} \sum_{\mathring{t}=0}^{2^{\mathring{x}}-1} \mathrm{odd}(\mathring{t}+1+y_i) \sum_{d=2^x+\mathring{t}\frac{2^x}{2^{\mathring{x}}}}^{2^x+(\mathring{t}+1)\frac{2^x}{2^{\mathring{x}}}-1} P(d) \right\rceil$ of the symbols in the vector $\mathbf{d}$ to satisfy these conditions and therefore we can expect half of this many of the transitions in the path $\mathbf{m}$ to be of each of the above-mentioned types. In addition, a transition from a holding state $m' \in \{2f+1, 2f+2\}$ to a unary state $m = 1 + \mathrm{odd}(m')$ will occur for each symbol in the vector $\mathbf{d}$ satisfying $d_i \geq 2^{\lceil f/2 \rceil}$. We can expect $\left\lceil \frac{a}{2} \cdot \left\lceil 1 - \sum_{d=1}^{2^{f/2}-1} p(d) \right\rceil \right\rceil$ of the symbols in the vector $\mathbf{d}$ to satisfy these conditions and therefore we can expect half of this many of the transitions in the path $\mathbf{m}$ to be of each of the above-mentioned types.

Finally, each symbol in the vector $\mathbf{d}$ satisfying $d_i \geq 2^{\lceil f/2 \rceil}$ will yield $\log_2(d_i) - f/2$ transitions from a holding state $m' \in \{2f+1, 2f+2\}$ to a holding state $m = m' - 2$. Therefore, the number of transitions in the path $\mathbf{m}$ that can be expected to be of each of the above-motioned types is given by

$$\begin{aligned} & \tfrac{a}{2} \sum_{d=2^{f/2}}^{L} P(d)(\lfloor \log_2(d) \rfloor - \tfrac{f}{2}) \\ = \; & \tfrac{a}{2} \left[\sum_{d=1}^{L} P(d)(\lfloor \log_2(d) \rfloor - \tfrac{f}{2}) - \sum_{d=1}^{2^{f/2}-1} P(d)(\lfloor \log_2(d) \rfloor - \tfrac{f}{2}) \right] \\ = \; & \tfrac{a}{2} \left[l_1 - \tfrac{f}{2} - \sum_{d=1}^{2^{f/2}-1} P(d)(\lfloor \log_2(d) \rfloor - \tfrac{f}{2}) \right], \end{aligned}$$

where l_1 is the average length of the unary codeword $\mathrm{Unary}(x_i)$, as described in Section 4.3.

Dividing the result for all cases by the expected number of transition in the path $\mathbf{m}$, namely al, yields the trasition probability given in (4.11).

Glossary

Symbols

3D Three-Dimensional.

4K Video Horizontal Resolution on the Order of 4,000 Pixels.

8K Video Horizontal Resolution on the Order of 8,000 Pixels.

A

ACS Add, Compare and Select.

AGM Anti-Gray Mapping.

APP *A Posteriori* Probability.

AVC Advanced Video Coding.

AWGN Additive White Gaussian Noise.

B

BCH Bose-Chaudhuri-Hocquenghem.

BCJR Bahl-Cocke-Jelinek-Raviv.

BER Bit Error Ratio.

C

CABAC Context-Adaptive Binary Arithmetic Coding.

CAVLC Context-Based Adaptive Variable-Length Coding.

CC Convolutional Code.

CIF Common Intermediate Format.

D

DCMC Discrete-input Continuous-output Memoryless Channel.

DCT Discrete Cosine Transform.

DPCM Differential Pulse Code Modulation.

DVB-T Digital Video Broadcasting standard − Terrestrial.

E

EEP Equal Error Protection.

EG Elias Gamma.

EG-CC Elias Gamma and Convolutional Code.

EGEC Elias Gamma Error Correction.

EWVLC Even Weight Variable Length Code.

EXIT EXtrinsic Information Transfer.

ExpG Exponential Golomb.

ExpGEC Exponential Golomb Error Correction.

F

FD Free Distance.

FHD Full High Definition.

FLC Fixed Length Code.

G

GM Gray Mapping.

H

HD Hamming Distance.

HEVC High Efficiency Video Coding.

I

IID Independent and Identically Distributed.

IrURC Irregular Unity Rate Code.

IrVLC Irregular Variable Length Code.

ISDN Integrated Services Digital Networks.

ISO/IEC International Standardization Organization/International Electrotechnical Commission.

ITU International Telecommunication Union.

J

JSCC Joint Source and Channel Code.

JVT Joint Video Team.

L

LDPC Low Density Parity Check.

LFSR Linear Feedback Shift Register.

LLR Logarithmic Likelihood Ratio.

Log-BCJR Logarithmic Bahl-Cocke-Jelinek-Raviv.

M

MC Motion Compensation.

MI Mutual Information.

ML Maximum Likelihood.

MPEG Moving Picture Expert Group.

MRF Markov Random Field.

MVC Multiview Video Coding.

N

NTSC National Television System Committee.

P

PAL Phase Alternating Line.

PSD Power Spectral Density.

Q

QCIF Quarter Common Intermediate Format.

QPSK Quaternary Phase Shift Keying.

R

REG Reordered Elias Gamma.

REGEC Reordered Elias Gamma Error Correction.

RExpG Reordered Exponential Golomb.

RExpGEC Reordered Exponential Golomb Error Correction.

RS Reed-Solomon.

RV Random Variable.

RVLC Reversible Variable Length Code.

S

SBSD Soft Bit Source Decoding.

SER Symbol Error Ratio.

SNR Signal to Noise Ratio.

SSCC Separate Source and Channel Code.

SSVLC Self-Synchronizing Variable Length Code.

SVC Scalable Video Coding.

U

UEC Unary Error Correction.

UEP Unequal Error Protection.

UHD Ultra High Definition.

URC Unity Rate Convolutional.

V

VLC Variable Length Code.

VLEC Variable Length Error Correction.

Bibliography

[1] R. Maunder, W. Zhang, T. Wang, and L. Hanzo, "A unary error correction code for the near-capacity joint source and channel coding of symbol values from an infinite set," *IEEE Transactions on Communications*, vol. 61, pp. 1977–1987, May 2013.

[2] W. Zhang, M. Brejza, T. Wang, R. Maunder, and L. Hanzo, "Irregular trellis for the near-capacity unary error correction coding of symbol values from an infinite set," *IEEE Transactions on Communications*, vol. 63, pp. 5073–5088, Dec 2015.

[3] ITU-T, *Recommendation H.120: Codecs for videoconferencing using primary digital group transmission*, March 1993.

[4] ITU-T, *Recommendation H.264: Advanced Video Coding for Generic Audiovisual Services*, March 2010.

[5] ITU-T, *Recommendation H.265: High efficiency video coding*, June 2015.

[6] L. Hanzo, P. Cherriman, and J. Streit, *Video Compression and Communications: From Basics to H.261, H.263, H.264, MPEG2, MPEG4 for DVB and HSDPA-Style Adaptive Turbo-Transceivers*. New York: John Wiley, 2007.

[7] J. Massey, "Joint source and channel coding," *Communication Systems and Random Process Theory*, pp. 279–293, December 1978.

[8] M. Wien, *High Efficiency Video Coding: coding Tools and Specification*. Springer, 2015.

[9] L. Hanzo, R. Maunder, J. Wang, and L.-L. Yang, *Near-Capacity Variable Length Coding*. Chichester, UK: Wiley, 2010.

[10] A. Hocquenghem, "Codes correcteurs d'Erreurs," *Chiffres (Paris)*, vol. 2, pp. 147–156, September 1959.

[11] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68–79, March 1960.

[12] ITU-T, *Recommendation H.261: Video codec for audiovisual services at px64 Kbit/s*, March 1993.

[13] ISO/IEC 11172, *Inforation technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s - Part 2: Video*, 1993.

[14] ITU-R, *BT.470 : Conventional analogue television systems*, Febrary 2005.

[15] ISO/IEC 13818, *Inforation technology - Generi coding of moving pictures and associated audio information - Part 2: Video*, 1993.

[16] ITU-T, *H.262 : Information technology - Generic coding of moving pictures and associated audio information: Video*, Febrary 2012.

[17] ETSI, *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz Satellite Services*, August 1997. EN 300 421 V1.1.2.

[18] ITU-T, *Recommendation H.263: Video Coding for Low Bitrate Communication*, March 1996.

[19] ISO/IEC 14496-2, *Inforation technology - Coding of audio-visual objects - Part 2: Visual*, 1993.

[20] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, pp. 1103–1120, September 2007.

[21] Y. Chen, Y.-K. Wang, K. Ugur, M. M. Hannuksela, J. Lainema, and M. Gabbouj, "The emerging MVC standard for 3D video services," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, pp. 1–13, January 2009.

[22] A. Netravali and J. Robbins, "Motion-compensated television coding: Part i," *The Bell System Technical Journal*, vol. 58, pp. 631–670, March 1979.

[23] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, pp. 90–93, Jan 1974.

[24] D. Lelewer and D. Hirschberg, "Data compression," *ACM Computing Surveys (CSUR)*, vol. 19, no. 3, pp. 261–296, 1987.

[25] ITU-T, *Recommendation H.263: Video Coding for Low Bitrate Communication*, Jan 2005.

[26] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, pp. 1098–1101, Sept 1952.

[27] B. Ryabko and J. Rissanen, "Fast adaptive arithmetic code for large alphabet sources with asymmetrical distributions," in *Proc. IEEE International Symposium on Information Theory*, (Lausanne, Switzerland), p. 319, June 2002.

[28] J. Ziv and A. Lempel, "Compression of indivdual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, pp. 530–536, Sept. 1978.

[29] S. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, pp. 399–401, Jul 1966.

[30] J. Teuhola, "A compression method for clustered bit-vectors," *Information processing letters*, vol. 7, no. 6, pp. 308–311, 1978.

[31] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, July 2003.

[32] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 620–636, July 2003.

[33] V. Sze and M. Budagavi, "High throughput cabac entropy coding in hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1778–1791, Dec 2012.

[34] C. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[35] R. Fano, *The transmission of information.* Massachusetts Institute of Technology, Research Laboratory of Electronics, 1949.

[36] N. Abramson, *Information Theory and Coding.* New York, USA: McGraw-Hill, 1966.

[37] S. Golomb, "Run-length encodings (corresp.)," *IEEE Transactions on Information Theory*, vol. 12, pp. 399–401, Jul 1966.

[38] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Transactions on Information Theory*, vol. 21, pp. 194–203, March 1975.

[39] A. Apostolico and A. Fraenkel, "Robust transmission of unbounded strings using fibonacci representations," *IEEE Transactions on Information Theory*, vol. 33, pp. 238–245, Mar 1987.

[40] R. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, pp. 147–160, 1950.

[41] L. Hanzo, T.H.Liew, B.L.Yeap, R. Tee, and S. Ng, *Turbo Coding, Turbo Equalisation and Space-Time Coding.* New York: John Wiley, 2011.

[42] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, pp. 38–49, September 1954.

[43] R. Gallager, "Low-Density Parity-Check Codes," *IEEE Transactions on Information Theory*, pp. 21–28, 1962.

[44] P. Elias, "Coding for noisy channels," *Proceeding of IRE*, vol. 43, no. 3, pp. 37–47, 1955.

[45] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes (1)," in *Proceedings of the International Conference on Communications*, vol. 2, (Geneva, Switzerland), pp. 1064–1070, May 1993.

[46] C. E. Shannon, *Mathematical Theory of Communication*. University of Illinois Press, 1963.

[47] D. Mackay and R. Neal, "Near Shannon limit performance of low density parity check codes," *Electronic Letter*, vol. 32, pp. 457–458, Aug. 1996.

[48] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Transactions on Information Theory*, vol. 44, pp. 909–926, May 1998.

[49] G. Forney Jr, "Review of random tree codes," *NASA Ames Research Center, Moffett Field, CA, USA, Tech. Rep. NASA CR73176*, 1967.

[50] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimising symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, pp. 284–287, March 1974.

[51] S. Ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Transactions on Communications*, vol. 49, pp. 1727–1737, October 2001.

[52] J. Wozencraft, "Sequential decoding for reliable communication," *IRE National Convention Record*, vol. 5, pt.2, pp. 11–25, 1957.

[53] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, pp. 300–304, June 1960.

[54] L. Bahl, C. Cullum, W. Frazer, and F. Jelinek, "An efficient algorithm for computing free distance (corresp.)," *IEEE Transactions on Information Theory*, vol. 18, pp. 437–439, May 1972.

[55] J. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Transactions on Information Theory*, pp. 76–80, 1978.

[56] W. Koch and A. Baier, "Optimum and Sub-Optimum Detection of Coded Data Disturbed by Time-Varying Intersymbol Interference," in *IEEE Global Telecommunications Conference, 1990*, pp. 1679–1684, 1990.

[57] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *IEEE International Conference on Communications,*, vol. 2, (Seattle, USA), pp. 1009–1013 vol.2, Jun 1995.

[58] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 619–637, Feb 2001.

[59] M. Luby, "Lt codes,," in *Proceedings of 43rd Annunal IEEE Symposium Foundations of Computer Science,*, (Vancouver, BC, Canada), pp. 1009–1013 vol.2, November 2002.

[60] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, pp. 2551–2567, June 2006.

[61] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory,*, vol. 55, pp. 3051–3073, July 2009.

[62] N. Bonello, R. Zhang, S. Chen, and L. Hanzo, "Reconfigurable rateless codes," *IEEE Transactions on Wireless Communications*, vol. 8, pp. 5592–5600, November 2009.

[63] R. Maunder, "A fully-parallel turbo decoding algorithm," *IEEE Transactions on Communications*, vol. 63, pp. 2762–2775, Aug 2015.

[64] N. Görtz, "Joint source channel decoding using bit-reliability information and source statistics," in *International Symposium on Information Theory*, (Cambridge, MA, USA), p. 9, August 1998.

[65] N. Görtz, "On the iterative approximation of optimal joint source-channel decoding," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 1662–1670, September 2001.

[66] J. Hagenauer, "The Turbo Principle: Tutorial Introduction and State of the Art," in *Proceedings of International Symposium on Turbo Codes and related topics*, (Brest, France), pp. 1–11, September 1997.

[67] L. Hanzo, J. Woodard, and P. Robertson, "Turbo decoding and detection for wireless applications," *Proceedings of the IEEE*, vol. 95, pp. 1178–1200, June 2007.

[68] M. Adrat, R. Vary, and J. Spittka, "Iterative Source-Channel Decoder Using Extrinsic Information from Softbit-Source Decoding," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, (Salt Lake City, UT, USA), pp. 2653–2656, May 2001.

[69] J. Kliewer, N. Görtz, and A. Mertins, "On iterative source-channel image decoding with Markov random field source models," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, (Montreal, Canada), pp. iv–661–iv–664, May 2004.

[70] J. Kliewer and R. Thobaben, "Iterative joint source-channel decoding of variable-length codes using residual source redundancy," *IEEE Transactions on Wireless Communications*, vol. 4, pp. 919–929, May 2005.

[71] J. Kliewer, N. Görtz, and A. Mertins, "Iterative source-channel decoding with Markov random field source models," *IEEE Transactions on Signal Processing*, vol. 54, pp. 3688–3701, October 2006.

[72] R. Maunder and L. Hanzo, "Genetic algorithm aided design of component codes for irregular variable length coding," *IEEE Transactions on Communication*, vol. 57, pp. 1290–1297, May 2009.

[73] B. Montgomery and J. Abrahams, "Synchronization of binary source codes (corresp.)," *IEEE Transactions on Information Theory*, vol. 32, pp. 849–854, Nov 1986.

[74] Y. Takishima, M. Wada, and H. Murakami, "Reversible variable length codes," *IEEE Transactions on Communications*, vol. 43, pp. 158–162, Feb 1995.

[75] V. Buttigieg and P. Farrell, "Variable-length error-correcting codes," *IEE Proceedings Communications*, vol. 147, pp. 211–215, Aug 2000.

[76] R. Thobaben and J. Kliewer, "Design considerations for iterativelydecoded source-channel coding schemes," in *Allerton Conference on Communications, Control, and Computing, 2006. Proceedings*, 2006.

[77] V. Balakirskii, "Joint source-channel coding using variable-length codes," *Problemy Peredachi Informatsii*, vol. 37, no. 1, pp. 12–27, 2001.

[78] M. Park and D. Miller, "Joint source-channel decoding for variable-length encoded data by exact and approximate map sequence estimation," *IEEE Transactions on Communications,*, vol. 48, pp. 1–6, Jan 2000.

[79] K. Sayood, H. Otu, and N. Demir, "Joint source/channel coding for variable length codes," *IEEE Transactions on Communications,*, vol. 48, pp. 787–794, May 2000.

[80] R. Bauer and J. Hagenauer, "Symbol-by-symbol map decoding of variable length codes," *ITG FACHBERICHT*, pp. 111–116, 2000.

[81] R. Thobaben and J. Kliewer, "Robust decoding of variable-length encoded markov sources using a three-dimensional trellis," *IEEE Communications Letters,*, vol. 7, pp. 320–322, July 2003.

[82] C. Weidmann, "Reduced-complexity soft-ln-soft-out decoding of variable-length codes," in *IEEE International Symposium on Information Theory, 2003. Proceedings,*, pp. 201–201, June 2003.

[83] R. Maunder, J. Kliewer, S. Ng, J. Wang, L.-L. Yang, and L. Hanzo, "Joint iterative decoding of trellis-based vq and tcm," *IEEE Transactions on Wireless Communications*, vol. 6, pp. 1327–1336, April 2007.

[84] Nasruminallah and L. Hanzo, "Exit-chart optimized short block codes for iterative joint source and channel decoding in h.264 video telephony," *IEEE Transactions on Vehicular Technology,*, vol. 58, pp. 4306–4315, Oct 2009.

[85] E. Akyol, K. Viswanatha, K. Rose, and T. Ramstad, "On zero-delay source-channel coding," *IEEE Transactions on Information Theory,*, vol. 60, pp. 7473–7489, Dec 2014.

[86] I. Alustiza, P. Crespo, and B. Beferull-Lozano, "Analog multiple description joint source-channel coding based on lattice scaling," *IEEE Transactions on Signal Processing,*, vol. 63, pp. 3046–3061, June 2015.

[87] D. Divsalar, S. Dolinar, and F. Pollara, "Serial concatenated trellis coded modulation with rate-1 inner code," in *Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE*, vol. 2, pp. 777–782 vol.2, 2000.

[88] S. Even and M. Rodeh, "Economical Encoding of Commas Between Strings.," *ommunications of the ACM*, vol. 21, no. 4, pp. 315–317, 1978.

[89] Q. Stout, "Improved prefix encodings of the natural numbers (Corresp.).," *IEEE Transactions on Information Theory*, vol. 26, no. 5, pp. 607–609, 1980.

[90] A. S. Fraenkel and S. T. Klein, "Robust Universal Complete Codes for Transmission and Compression.," *Discrete Applied Mathematics*, vol. 64, no. 1, pp. 31–55, 1996.

[91] R. Rice and J. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data," *IEEE Transactions on Communication Technology*, vol. 19, pp. 889–897, December 1971.

[92] R. Gallager and D. van Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Transactions on Information Theory*, vol. 21, pp. 228–230, March 1975.

[93] N. L. Johnson, A. W. Kemp, and S. Kotz, *Univariate Discrete Distributions*. New York, NY, USA: John Wiley & Sons, 2005.

[94] N. Johnson, A. Kemp, and K. Samuel, *Univariate Discrete Distributions*. John Wiley & Sons, 2005.

[95] W. Zhang, R. Maunder, and L. Hanzo, "On the complexity of unary error correction codes for the near-capacity transmission of symbol values from an infinite set," in *Proc. IEEE Wireless Communications and Networking Conference*, (Shanghai, China), 2013 http://eprints.soton.ac.uk/344059/.

[96] M. El-Hajjar and L. Hanzo, "Exit charts for system design and analysis," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 127–153, 2014.

[97] L. Hanzo, S. Ng, T. Keller, and W. T. Webb, *Quadrature amplitude modulation: From basics to adaptive trellis-coded, turbo-equalised and space-time coded OFDM, CDMA and MC-CDMA systems*. IEEE Press-John Wiley, 2004.

[98] M. Brejza, W. Zhang, R. Maunder, B. Al-Hashimi, and L. Hanzo, "Adaptive iterative detection for expediting the convergence of a serially concatenated unary error correction decoder, turbo decoder and an iterative demodulator," in *2015 IEEE International Conference on Communications (ICC)*, pp. 2603–2608, June 2015.

[99] W. Zhang, Y. Jia, X. Meng, M. Brejza, R. Maunder, and L. Hanzo, "Adaptive iterative decoding for expediting the convergence of unary error correction codes," *IEEE Transactions on Vehicular Technology*, vol. 64, pp. 621–635, Feb 2015.

[100] J. Proakis, *Digital Communications*. New York, USA: McGraw-Hill, 3rd ed., 1995.

[101] C. Xu, D. Liang, S. Sugiura, S. Ng, and L. Hanzo, "Reduced-complexity approx-log-map and max-log-map soft psk/qam detection algorithms," *IEEE Transactions on Communications*, vol. 61, pp. 1415–1425, April 2013.

[102] P. Robertson, "Illuminating the structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) Codes," in *IEEE Globecom*, (San Francisco, USA), pp. 1298–1303, November 1994.

[103] L. Li, R. Maunder, B. Al-Hashimi, and L. Hanzo, "A low-complexity turbo decoder architecture for energy-efficient wireless sensor networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, pp. 14–22, Jan 2013.

[104] "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding (Release 9) 3GPP Organizational Partners TS 36.212, Rev. 8.3.0," tech. rep., May 2008.

[105] A. Ashikhmin, G. Kramer, and S. Ten Brink, "Extrinsic information transfer functions: Model and erasure channel properties," *IEEE Transactions on Information Theory*, vol. 50, pp. 2657–2673, Nov. 2004.

[106] R. Maunder and L. Hanzo, "Iterative decoding convergence and termination of serially concatenated codes," *IEEE Transactions on Vehicular Technology*, vol. 59, pp. 216–224, January 2010.

[107] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for 'turbo-like' codes," in *Proc. Allerton Conf. on Communications, Control and Computing*, pp. 201–210, September 1998.

[108] L. Hanzo, R. Maunder, J. Wang, and L.-L. Yang, *Near-Capacity Variable-Length Coding: Regular and Exit-Chart Aided Irregular Designs*. John Wiley & Sons Ltd, 2010.

[109] P. Frenger, P. Orten, and T. Ottosson, "Convolutional codes with optimum distance spectrum," *Communications Letters, IEEE*, vol. 3, pp. 317–319, Nov 1999.

[110] T. Wang, W. Zhang, R. Maunder, and L. Hanzo, "Near-capacity joint source and channel coding of symbol values from an infinite source set using elias gamma error correction codes," *IEEE Transactions on Communications*, vol. 62, pp. 280–292, January 2014.

[111] "Advanced video coding for generic audiovisual services," tech. rep., ITU-T Std. H.264, March 2005.

[112] A. Ashikhmin, G. Kramer, and S. Ten Brink, "Extrinsic information transfer functions: model and erasure channel properties," *IEEE Transactions on Information Theory*, vol. 50, pp. 2657–2673, November 2004.

[113] J. Kliewer, A. Huebner, and D. J. Costello, "On the achievable extrinsic information of inner decoders in serial concatenation," in *IEEE International Symposium on Information Theory 2006* , (Seattle, WA, USA), pp. 2680–2684, July 2006.

[114] M. Tuchler, "Convergence prediction for iterative decoding of threefold concatenated systems," in *IEEE Global Telecommunications Conference*, vol. 2, (Taipei, Taiwan), pp. 1358–1362, November 2002.

[115] ETSI, *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, August 1997. EN 300 744 V1.1.2.

[116] J. Hokfelt, O. Edfors, and T. Maseng, "A turbo code interleaver design criterion based on the performance of iterative decoding," *IEEE Communications Letters*, vol. 5, pp. 52–54, Feb 2001.

[117] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 429–445, March 1996.

[118] W. Zhang, Z. Song, M. Brejza, T. Wang, R. Maunder, and L. Hanzo, "Learning-aided unary error correction codes for non-stationary and unknow sources," *to be submitted*, Dec 2015.

[119] M. Brejza, T. Wang, W. Zhang, D. Khalili, R. Maunder, B. Al-Hashimi, and L. Hanzo, "Exponential golomb and rice error correction codes for near-capacity joint source and channel coding," *to be submitted*, Dec 2015.

Author Index

Subject Index