UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Unary Error Correction Coding

by

Wenbo Zhang

BSc., MSc.

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

March 2016

SUPERVISOR:

Dr. Robert G. Maunder

PhD, CEng, MIET, SMIEEE, FHEA

Department of Electronics and Computer Science

and

Professor Lajos Hanzo

FREng, FIEEE, FIEE, DSc, EIC IEEE Press

Chair of Communications, Signal Processing and Control Group

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

Doctor of Philosophy

Unary Error Correction Coding

by Wenbo Zhang

In this thesis, we introduce the novel concept of Unary Error Correction (UEC) coding. Our UEC code is a Joint Source and Channel Coding (JSCC) scheme conceived for performing both the compression and error correction of multimedia information during its transmission from an encoder to a decoder. The UEC encoder generates a bit sequence by concatenating and encoding unary codewords, while the decoder operates on the basis of a trellis that has only a modest complexity, even when the source symbol values are selected from a set having an infinite cardinality, such as the set of all positive integers. This trellis is designed so that the transitions between its states are synchronous with the transitions between the consecutive unary codewords in the concatenated bit sequence. This allows the UEC decoder to exploit any residual redundancy that remains following UEC encoding for the purpose of error correction by using the classic Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm. Owing to this, the UEC code is capable of mitigating any potential capacity loss, hence facilitating near-capacity operation, even when the cardinality of the symbol value set is infinite.

We investigate the applications, characteristics and performance of the UEC code in the context of digital telecommunications. Firstly, we propose an adaptive UEC design for expediting the decoding process. By concatenating the UEC code with a turbo code, we conceive a three-stage concatenated adaptive iterative decoding technique. A Three-Dimensional (3D) EXtrinsic Information Transfer (EXIT) chart technique is proposed for both controlling the dynamic adaptation of the UEC trellis decoder, as well as for controlling the activation order between the UEC decoder and the turbo decoder. Secondly, we develop an irregular UEC design for 'nearer-capacity' operation. The irregular scheme employs different UEC parametrizations for the coding of different subsets of each message frame, operating on the basis of a single irregular trellis having a novel structure. This allows the irregularity to be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis. Hence, nearer-to-capacity operation is facilitated by exploiting this fine-grained control of the irregularity. Thirdly, we propose a learning-aided UEC

design for transmitting symbol values selected from unknown and non-stationary probability distributions. The learning-aided UEC scheme is capable of heuristically inferring the source symbol distribution, hence eliminating the requirement of any prior knowledge of the symbol occurrence probabilities at either the transmitter or the receiver. This is achieved by inferring the source distribution based on the received symbols and by feeding this information back to the decoder. In this way, the quality of the recovered symbols and the estimate of the source distribution can be gradually improved in successive frames, hence allowing reliable near-capacity operation to be achieved, even if the source is unknown and non-stationary.

Finally, we demonstrate that the research illustrated in this thesis can be extended in several directions, by highlighting a number of opportunities for future work. The techniques proposed for enhancing the UEC code can be extended to the Rice Error Correction (RiceEC) code, to the Elias Gamma Error Correction (EGEC) code and to the Exponential Golomb Error Correction (ExpGEC) code. In this way, our UEC scheme may be extended to the family of universal error correction codes, which facilitate the near-capacity transmission of infinite-cardinality symbol alphabets having any arbitrary monotonic probability distribution, as well as providing a wider range of applications. With these benefits, this thesis may contribute to future standards for the reliable near-capacity transmission of multimedia information, having significant technical and economic impact.

Declaration of Authorship

I, **Wenbo Zhang**, declare that the thesis entitled **Unary Error Correction Coding** and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- 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;
- Where I have consulted the published work of others, this is always clearly attributed;
- 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;
- I have acknowledged all main sources of help;
- 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;
- Parts of this work have been published.

Signed: ..... Date: .....

Acknowledgements

I would like to express my heartfelt gratitude to my supervisors Dr. Robert G. Maunder and Professor Lajos Hanzo 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.

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, Dr. Soon Xin Ng, Dr. Mohammed El-Hajjar, Dr. Matthew F. Brejza, Dr. Tao Wang and Dr. Jie Hu for their technical support and collaborative work.

I would like to express my warmest gratitude to my father, Yuezhong, my mother, Xiaoling, for their lifelong support, for their eternal love and for their understanding and faith on me. Finally, thanks also go to my fiancée, Haibo, for her lover, support and care for me. Words cannot describe how lucky I am to have her in my life. I love you and look forward to our lifelong journey.

List of Publications

Journals:

1. Robert G. Maunder, **Wenbo Zhang**, Tao Wang and Lajos 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, no. 5, pp. 1977-1987, May 2013.
2. Tao Wang, **Wenbo Zhang**, Robert G. Maunder and Lajos 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, no. 1, pp. 280-292, January 2014.
3. **Wenbo Zhang**, Yanbo Jia, Xi Meng, Matthew F. Brejza, Robert G. Maunder and Lajos Hanzo, "Adaptive Iterative Decoding for Expediting the Convergence of Uanry Error Correction Codes", *IEEE Transactions on Vehicular Technology*, vol 64, no. 2, pp. 621-635, February 2015.
4. Jinhui Chen, **Wenbo Zhang**, Robert G. Maunder and Lajos Hanzo, "Bit-by-Bit Iterative Decoding Expedites the Convergence of Repeat Accumulate Decoders", *IEEE Transactions on Communications*, vol. 63, no. 6, pp. 1952-1962, June 2015.
5. **Wenbo Zhang**, Matthew F. Brejza, Tao Wang, Robert G. Maunder and Lajos Hanzo, "An Irregular 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. 63, no. 12, pp. 5073-5088, December 2015.
6. **Wenbo Zhang**, Zeyu Song, Matthew F. Brejza, Tao Wang, Robert G. Maunder and Lajos Hanzo, "Learning-aided Unary Error Correction Codes for Non-Stationary and Unknown Sources", *IEEE Access*, accepted, February 2016.
7. Tao Wang, **Wenbo Zhang**, Matthew F. Brejza, Robert G. Maunder and Lajos Hanzo, "Reordered Elias Gamma Error Correction Codes for the Near-Capacity Joint Source and Channel Coding of Multimedia Information", *IEEE Transactions on Vehicular Technology*, submitted, February 2016.
8. Jie Hu, Dimitrios Alanis, **Wenbo Zhang**, Lie-Liang Yang and Lajos Hanzo, "Energy-Efficient Cross-Layer Design of Wireless Mesh Networking for Content Sharing in Online Social Networks", *to be submitted*.
9. Matthew F. Brejza, Tao Wang, **Wenbo Zhang**, David Al-Khalili, Robert G. Maunder, Bashir M. Al-Hashimi and Lajos Hanzo, "Exponential Golomb and Rice Error Correction Codes for Near-Capacity Joint Source and Channel Coding", *to be submitted*.

Conferences:

1. **Wenbo Zhang**, Robert G. Maunder and Lajos Hanzo, "On the Complexity of Uanry Error Correction Codes for the Near-Capacity Transmission of Symbol Calues from An Infinite Set", *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2795-2800, Shanghai, CN, 7-10 April, 2013.
2. Matthew F. Brejza, **Wenbo Zhang**, Robert G. Maunder, Bashir M. Al-Hashimi and Lajos Hanzo, "Adaptive Iterative Decoding for Expediting the Convergence of Unary Error Correction Decoders Concatenated with Turbo Decoders and Iterative Demodulator", *IEEE International Conference on Communications (ICC)*, "Smart City and Smart World", pp. 2603-2608, London, GB, 8-12 June, 2015.

Contents

Glossary

2D	Two-Dimensional
3D	Three-Dimensional
ACS	Add, Compare and Select
APP	*A Posteriori* Probability
AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BCJR	Bahl, Cocke, Jelinek and Raviv
BEC	Binary Erasure Channel
BER	Bit Error Ratio
BICM	Bit-Interleaved Coded Modulation
CC	Convolutional Code
CRC	Cyclic Redundancy Check
DCMC	Discrete-input Continuous-output Memoryless Channel
EA	Evolutionary Algorithm
EG	Elias Gamma
EGEC	Elias Gamma Error Correction
EWVLC	Even Weight Variable Length Codes
EXIT	EXtrinsic Information Transfer
ExpGEC	Exponential Golomb Error Correction
FLC	Fixed length Code

HD Hamming Distance
HEVC High Efficiency Video Coding

IID Independent and Identically Distributed
IrCC Irregular Convolutional Code
IrTrellis Irregular Trellis
IrUEC *Irregular* Unary Error Correction
IrURC Irregular Unity-Rate Convolutional
IrVLC Irregular Variable Length Code

JSCC Joint Source and Channel Coding

LDPC Low-Density Parity-Check
LFSR Linear Feedback Shift Register
LLR Logarithmic Likelihood Ratio
LTE Long Term Evolution
LUT Look-Up-Table

MAP Maximum A-Posteriori
MI Mutual Information
ML Maximum Likelihood
MMIA Maximal Mutual Information Achieving

PCC Parallel Concatenated Codes
PSK Phase-Shift Keying

QPSK Quadrature Phase-Shift Keying

REG Reordered Elias Gamma
REGEC Reordered Elias Gamma Error Correction
RiceEC Rice Error Correction
RV Random Variable
RVLC Reversible Variable Length Code
RVLC Reversible Variable Length Codes

SCC Serially Concatenated Codes
SECC Self-Concatenated Codes

SER Symbol Error Ratio
SISO Soft-In Soft-Out
SNR Signal to Noise Ratio
SOVA Soft-Output Viterbi Algorithm
SSCC Separate Source and Channel Coding
SSVLC Self-Synchronizing Variable Length Codes

TCM Trellis-Coded Modulation

UEC Unary Error Correction
UEP Unequal Error Protection
URC Unity-Rate Convolutional

VLC Variable Length Code
VLEC Variable Length Error Correction
VQEG Video Quality Expert Group

XOR Boolean Exclusive-OR

List of Symbols

Special Symbols

Schematics

$\mathbf{x}$: The source symbol vector.

$\hat{\mathbf{x}}$: The reconstructed source symbol vector.

a: The length of source symbol vector.

$\mathbf{a}$: The source bit vector.

$\hat{\mathbf{a}}$: The reconstructed source bit vector.

J: The length of source bit vector.

$\mathbf{y}$: The encoded bit vector.

$\hat{\mathbf{y}}$: The reconstructed encoded bit vector.

b: The length of encoded bit vector.

π: The interleaving.

π^{-1}: The de-interleaving.

$\mathbf{z}$: The encoded bit vector.

$\mathbf{w}$: The interleaved bit vector.

$\mathbf{c}$: The multiplexed bit vector.

$\mathbf{g}$: The modulated vector.

$\tilde{(\cdot)}$: The Logarithmic Likelihood Ratio (LLR) vector pertaining to the specified bits/symbols vector.

$\tilde{(\cdot)}^{\mathrm{p}}$: The *a posteriori* LLR vector pertaining to the specified bits/symbols vector.

$\tilde{(\cdot)}^{\mathrm{a}}$: The *a priori* LLR vector pertaining to the specified bits/symbols vector.

$\tilde{(\cdot)}^{\mathrm{a}}_{\mathrm{u}}$: The *a priori* LLR vector of upper component code.

$\tilde{(\cdot)}^{\mathrm{a}}_{\mathrm{l}}$: The *a priori* LLR vector of lower component code.

$\tilde{(\cdot)}^{\mathrm{a}}_{\mathrm{o}}$: The *a priori* LLR vector of outer component code.

$\tilde{(\cdot)}^{\mathrm{e}}$: The extrinsic LLR vector pertaining to the specified bits/symbols vector.

$\tilde{(\cdot)}^{\mathrm{e}}_{\mathrm{u}}$: The extrinsic LLR vector of upper component code.

$\tilde{(\cdot)}_{\mathrm{l}}^{\mathrm{e}}$: The extrinsic LLR vector of lower component code.
$\tilde{(\cdot)}_{\mathrm{o}}^{\mathrm{e}}$: The extrinsic LLR vector of outer component code.
M: The storage memory employed in the learning-aided scheme.
m: The size of the storage memory M.
$\mathbf{D}$: The input data stream of demultiplexer.
$\tilde{\mathbf{D}}$: The output data stream of demultiplexer.

Probability distribution

$\Pr(\cdot)$: The probability of an arbitrary event occurring.
$\Pr(\cdot|\cdot)$: The conditional probability.
$\Pr(\cdot,\cdot)$: The joint probability.
H_X: The symbol entropy.
H_Z: The bit entropy.
$\mathbb{N}_1$: The infinite-cardinality set comprising all positive integers.
$\zeta(s)$: The Riemann zeta function.
p_1: The probability of symbol value equals 1.
$\bar{p_1}$: The mean value of p_1.
$\mathbf{p}$: The probabilities of the first $r/2-1$ symbol values.
$\hat{\mathbf{p}}$: The estimated value of $\mathbf{p}$.
l: The average codeword length for each symbol.
$\hat{l}$: The estimated value of l.
T: The number of successive symbol vectors produced per cycle.
σ: The standard deviation of the filtered Gaussian-distributed values.

Trellis

$\mathbb{C}$: The codeword of the trellis.
$\overline{\mathbb{C}}$: The complementary codeword of the trellis.
r: The number of states in the trellis.
n: The number of bits in the trellis.
j: The trellis state index.
m_j: The j-th trellis state.
M: The set of all possible trellis states.
d_f: The free-distance.
$g(\cdot)$: The hexadecimal generator polynomials.
$f(\cdot)$: The hexadecimal feedback polynomials.

Codebook

r: The number of states in the codebook.

n: The number of bits in the codebook.

t: The component URC code index.

T: The component URC codes count.

$\mathbf{URC}$: The component URC codebook.

$\mathbf{URC}^t$: The component URC codeword.

s: The component UEC code index.

S: The component UEC codes count.

$\mathbf{UEC}$: The component UEC codebook.

$\mathbf{UEC}^s$: The component UEC codeword.

n: The component CC code index.

N: The component CC codes count.

$\mathbf{CC}$: The component CC codebook.

$\mathbf{CC}^n$: The component CC codeword.

$\mathbf{CC}_{\mathrm{ns}}$: The non-systematic recursive CC codebook.

$\mathbf{CC}_{\mathrm{ns}}^n$: The non-systematic recursive CC codeword.

$\mathbf{CC}_{\mathrm{s}}$: The systematic recursive CC codebook.

$\mathbf{CC}_{\mathrm{s}}^n$: The systematic recursive CC codeword.

Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm

$\boldsymbol{\gamma}$: The *a priori* trellis transition probabilities.

$\gamma_j(m, m')$: The j-th *a priori* trellis transition probability from state m to state m'.

$\boldsymbol{\alpha}$: The forward trellis transition probabilities.

$\alpha_j(m)$: The j-th forward trellis transition probability obtained for the trellis state m.

$\boldsymbol{\beta}$: The backward trellis transition probabilities.

$\beta_j(m')$: The j-th backward trellis transition probability obtained for the trellis state m'.

$\boldsymbol{\delta}$: The *a posteriori* trellis transition probabilities.

$\delta_j(m, m')$: The j-th *a posteriori* trellis transition probability from state m to state m'.

I: The number of iterative decoding iterations.

EXtrinsic Information Transfer (EXIT) chart

$I(\cdot, \cdot)$: The Mutual Information (MI).

$f[I(\tilde{(\cdot)}^{\mathrm{a}}, (\cdot))]$: The EXIT function.

$f[I(\tilde{(\cdot)}^{\mathrm{e}}, (\cdot))]$: The inverted EXIT function.

$\mathcal{A}$: The area *above* the inverted EXIT curve.

A: The area beneath the EXIT curve.

A_{o}: The area beneath the EXIT curve of outer code.

A_{i}: The area beneath the EXIT curve of inner code.

Channel

$\mathbf{h}$: The channel gain vector.
$\mathbf{n}$: The Additive White Gaussian Noise (AWGN) vector.
$\mathbf{N_0}$: The power spectral density of the AWGN.
η: The effective throughput.
C: The channel capacity.
E_s: The energy per symbol.
T_s: The symbol duration time.
f_s: The frequency of the carrier.
$s_n(\cdot)$: The signal in the time domain.
E_s/N_0: The channel Signal to Noise Ratio (SNR).
E_b/N_0: The channel SNR per bit of source information.

Code parameters

R: The coding rate.
R_{o}: The coding rate of outer code.
R_{i}: The coding rate of inner code.
$w(\mathbf{y})$: The weight of bit vector $\mathbf{y}$.
$l(\mathbf{y})$: The length of bit vector $\mathbf{y}$.

Special Operations

$|\cdot|^2$: The Euclidean norm of a vector/matrix.
$\mathrm{Re}(\cdot)$: The real part of a complex value.
$\mathrm{Im}(\cdot)$: The imaginary part of a complex value.
$\angle B$: The angle of a complex value B.
$\sum$: The summation of all elements.
$\forall$: For all elements within a certain range.
$\lim$: The limitation operation.
$\exp(\cdot)$: The exponential operation.
$\log(\cdot)$: The logarithm operation.
$\ln(\cdot)$: The natural logarithm operation.
$E\{\cdot\}$: The expectation operation.
$\lceil\cdot\rceil$: Rounding a numerical value to its nearest higher integer.
$\lfloor\cdot\rfloor$: Rounding a numerical value to its nearest lower integer.
$\max(\cdot)$: The maximum value of a vector/matrix.

$\min(\cdot)$:	The minimum value of a vector/matrix.
$mod(x, y)$:	The remainder after the division of x by y.
$\overline{(\cdot)}$:	The complementary operation.
$\oplus$:	The Boolean Exclusive-OR (XOR) operation.
$odd(\cdot)$:	The function yields 1 if the operand is odd or 0 if it is even.

Chapter 1

Introduction

The novel concept of Unary Error Correction (UEC) coding is introduced. We investigate its applications, characteristics and performance in the context of digital telecommunications. Our UEC scheme is a Joint Source and Channel Coding (JSCC) [1] scheme conceived for performing both compression and error correction of multimedia information during its transmission from an encoder to a decoder. Moreover, our near-capacity UEC scheme is designed for facilitating the practical encoding and decoding of multimedia information at a moderate complexity.

1.1 Separate Source and Channel Coding

In Shannon's seminal contribution [2], his source and channel coding separation theorem stated that source coding designed for compression and channel coding invoked for error correction can be designed entirely independently, without any loss of performance. In general, a basic Separate Source and Channel Coding (SSCC) scheme in wireless digital communications may be represented by Figure 1.1. When communicating over a perfectly noiseless wireless communications channel, Shannon [2] showed that the minimum number of bits required after compression using source coding to reliably convey perfect knowledge of the source signal's information content to the receiver is given by the source entropy, because source coding eliminates the redundancy inherent in the source information. When communicating over noisy channels, Shannon [2] showed that if a source signal's information content is conveyed at a rate (bits per second) that does not exceed the channel's capacity, then it is theoretically possible to reconstruct it with an infinitesimally low probability of error. This motivates the employment of separate channel coding, which introduces carefully controlled redundancy that can be beneficially exploited for error correction. With this SSCC guidelines, the efforts of the academic and industrial research communities expended over the past 60 years are summarized at a glance in Table 1.1.

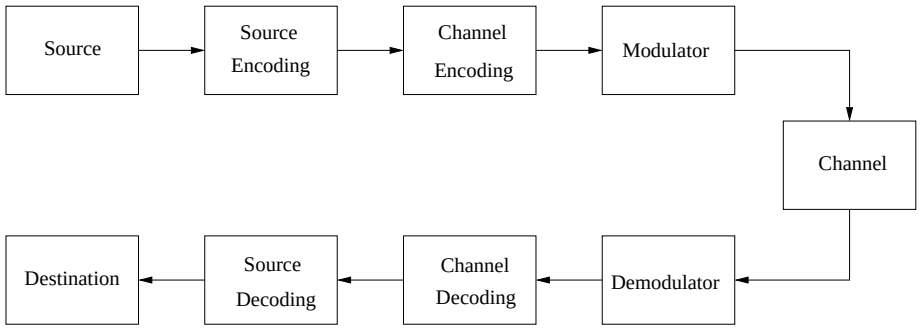

Figure 1.1: A digital communications system relying on Separate Source and Channel Coding (SSCC).

However, Shannon's findings are only valid under a number of idealistic assumptions [3], namely that the information is transmitted over an uncorrelated non-dispersive narrowband Additive White Gaussian Noise (AWGN) channel, while potentially imposing an infinite decoding complexity and buffering latency. These assumptions clearly have limited validity for practical finite-delay transmissions over realistic fading wireless channels [4]. Additionally, Shannon assumed that the source is stationary and that it is losslessly encoded. These assumptions have a limited validity in the case of multimedia transmission, since video, image, audio information is typically non-stationary, having characteristics that vary in time and/or space [5, 6]. Furthermore, 'lossy' compression [7] is often readily tolerated for multimedia information, since human observers can typically tolerate moderate signal degradation in exchange for requiring a reduced bandwidth. Owing to this, some residual redundancy is typically retained during source coding, hence preventing near-capacity operation. This observation motivated the conception of Joint Source and Channel Coding (JSCC) [1], as it will be introduced in Section 1.2.

1.2 Joint Source and Channel Coding

In order to exploit the residual redundancy and hence to achieve near-capacity operation, the classic SSCC schemes may be replaced by JSCC arrangements [1] in many applications. Generally, a basic JSCC scheme routinely used in wireless digital communications system is represented by Figure 1.2. The history and milestones of JSCC development are listed in Table 1.2. Diverse methods have been proposed for JSCC, which we will now briefly discuss.

Channel-optimised source coding [48, 49] may be employed for reducing the reconstruction error of the source, when the channel decoder is unable to correct all transmission errors. More particularly, the source encoder is designed with special consideration of the transmission errors that are most likely to occur, namely those causing a particular binary codeword to be confused with another similar codeword. In this way, channel-optimised

Year	Author(s)	Contribution
1948	Shannon [2]	Information theory and channel capacity
1949	Fano [8]	Discrete message transmission in noiseless systems
1950	Hamming [9]	Hamming code was proposed
1952	Huffman [10]	Huffman code was proposed
1954	Reed [11]	Reed-Muller (RM) code was introduced
1955	Elias [12]	Convolutional Code (CC) was conceived
1957	Wozencraft [13]	Sequential decoding
1962	Gallager [14]	Low-Density Parity-Check (LDPC) code
1966	Golomb [15]	Golomb and Rice Codes
1972	Bahl *et al.* [16]	Maximum A-Posteriori (MAP) algorithm
1973	Forney [17]	The Viterbi algorithm
1974	Bahl *et al.* [18]	Symbol based MAP algorithm
1975	Elias *et al.* [19]	Elias Gamma (EG) source code
1977	Imai and Hirawaki [20]	Bandwidth-efficient MultiLevel Coding (MLC)
1978	Wolf [21] Ziv and Lempel [22]	Trellis-decoding of block codes Lempel-Ziv coding
1979	Rissanen and Landgon [23]	Describes a broad class of arithmetic codes
1982	Ungerböck [24]	Trellis-Coded Modulation (TCM)
1987	Witten *et al.* [25]	Arithmetic coding for data compression
1988	Blahut [26]	Multiple trellis-coded modulation
1989	Calderbank [27] Hagenauer *et al.* [28]	Multilevel codes and multistage decoding Soft-Output Viterbi Algorithm (SOVA)
1990	Koch and Baier [29]	Classic Log-MAP algorithm
1991	Webb *et al.* [30]	Hard-decision Star QAM/Differential Amplitude Phase Shift Keying (DAPSK)
1992	Zehavi [31]	Bit-Interleaved Coded Modulation (BICM)
1993	Berrou [32]	Turbo codes
1994	Kofman *et al.* [33] Le Goff *et al.* [34]	Performance of a multilevel coded modulation BICM-based Turbo Coded Modulation (TuCM)
1995	Robertson *et al.* [35]	Approx-Log-MAP algorithm
1997	Li and Ritcey [36]	Bit-Interleaved Coded Modulation with Iterative Decoding (BICM-ID)
1998	Robertson and Wörz [37] Benedetto *et al.* [38]	Turbo trellis-coded modulation (TTCM) Self-concatenated coding
2001	Richardson *et al.* [39]	Irregular LDPC code
2002	Luby [40]	Rateless Luby Transform (LT) codes
2004	Hou and Lee [41]	Multilevel LDPC codes design for semi-BICM
2006	Shokrollahi [42]	Rateless Raptor codes
2007	Yue *et al.* [43] Grangetto *et al.* [44]	Finite-length LDPC code Iterative Decoding of Serially Concatenated Arithmetic and Channel Codes
2009	Arikan [45]	Polar codes
2012	Wang and Luo [46]	Generalized channel coding theory for random access communication
2015	Luo [47]	Generalized channel coding theory for distributed communication

Table 1.1: Milestones in Separate Source and Channel Coding (SSCC).

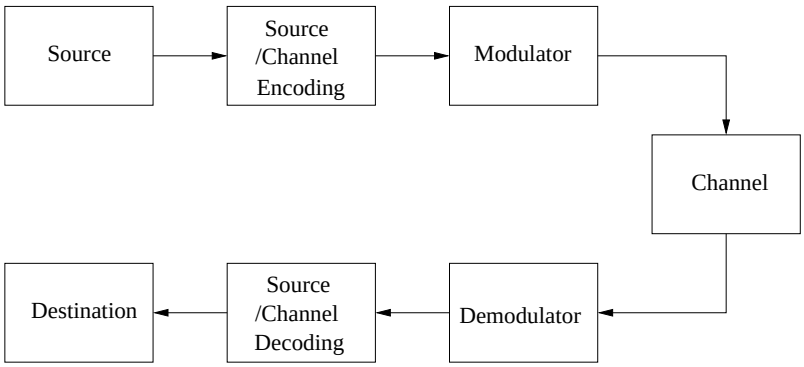

Figure 1.2: A digital communications system relying on Joint Source and Channel Coding.

source encoding allocates pairs of similar codewords to represent similar reconstructed source parameter values. This principle may be applied both to scalar quantisation [50, 51] and to vector quantisation [49, 52]. Furthermore, JSCC may be beneficially employed, if some of the source correlation is not removed during source encoding [53], or some redundancy is intentionally introduced during source coding [54]. Based on this approach, the receiver is capable of exploiting the residual redundancy in order to provide an error correction capability, which may be exploited for mitigating any transmission errors that could not be eliminated during channel decoding.

JSCC has been successfully applied to the encoding of symbols selected from finite sets, such as the 26 letters of the English alphabet $\{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots, \mathrm{z}\}$. However, when the source symbol values are selected from a set having an infinite cardinality, such as the positive integers in the range of 1 to infinity $\mathbb{N}_1 = \{1, 2, 3, \ldots, \infty\}$, the existing JSCCs, such as Self-Synchronizing Variable Length Codes (SSVLC) [55], Reversible Variable Length Codes (RVLC) [56], Variable Length Error Correction (VLEC) codes [57], Even Weight Variable Length Codes (EWVLC) [58] and Irregular Variable Length Code (IrVLC) [59] become unsuitable. More specifically, when the cardinality of the symbol value set is infinite, the trellis and graph structures [60–67] employed by these codes become infinitely large, hence the corresponding decoding algorithms become infinitely complex. This motivates the work in this treatise.

1.3 Motivation and Contribution

The motivation of our UEC code is summarized in Table 1.3. For those source symbols that are selected from a set having finite cardinality, classic SSCC based on Huffman [2] or Shannon-Fano [8] codes may indeed be capable of reconstructing the source information with an infinitesimally low probability of error, provided that the transmission rate does not exceed the channel's capacity [2]. However, SSCC schemes require both the transmitter

Year	Author(s)	Contribution
1959	Kotelnikov [68]	JSCC using Shannon-Kotelnikov mappings
1969	Kurtenbach and Wintz [69]	Quantizing for noisy channels
1978	Massey [1]	JSCC tutorial
1980	Linde *et al.* [48]	Channel-optimised indexing of vector quantisation
1984	Kumazawa *et al.* [49]	Channel-optimised vector quantisation
1986	Montgomery and Abrahams [55]	Self-Synchronizing Variable Length Codes
1987	Farvardin *et al.* [70]	Optimal quantizer design for noisy channels
1989	Wyrwas and Farrell [71]	JSCC for binary image transmission
1990	Farvardin [52]	Channel-optimised vector quantisation
1991	Sayood and Borkenhagen [53]	Use residual redundancy in JSCC design
1993	Ramchandran *et al.* [72]	Multiresolution JSCC for digital broadcast
1995	Takishima *et al.* [56]	Reversible Variable Length Codes
1998	Kozintsev and Ramchandran [73]	Multi-resolution JSCC over energy-constrained time-carrying channels
1999	Dyck and Miller [74]	JSCC in video transmission
2000	Cai and Chen [75] Buttigieg and Farrell [57] Jin *et al.* [76]	Robust JSCC in image transmission Variable Length Error Correction codes Irregular Repeat-Accumulate (IRA) codes
2001	Görtz [77] Balakirsky [60]	Iterative JSCC decoding framework JSCC using variable length codes
2002	Ramstad [78]	Analog JSCC using Shannon mapping
2003	Hagenauer and Görtz [79]	The turbo principle in JSCC
2005	Kliewer and Thobaben [80]	Iterative JSCC of variable length codes
2006	Thobaben and Kliewer [58]	Even Weight Variable Length Codes
2007	Jaspar *et al.* [81] Xu *et al.* [82]	JSCC turbo technique for discrete-sources Distributed JSCC using Raptor codes
2009	Maunder and Hanzo [59]	Irregular Variable Length Code
2011	Minero *et al.* [83]	JSCC via hybrid coding
2012	Persson *et al.* [84]	JSCC for the MIMO broadcast channel
2013	Kostina and Verdu [85]	Lossy JSCC having a finite block length
2014	Romero *et al.* [86]	Analog JSCC for wireless optical communications
2015	Tridenski *et al.* [87]	Ziv-Zakai-Rényi bound for JSCC

Table 1.2: Milestones in Joint Source and Channel Coding (JSCC).

and receiver to accurately estimate the occurrence probability of every symbol value that the source produces. For example, in the English alphabet, the letter 'e' occurs with a much higher probability than the letter 'x'. In practice, the occurrence probability of rare symbol values can only be accurately estimated, if a sufficiently large number of symbols has been observed, hence potentially imposing an excessive latency.

This motivates the design of so-called universal codes, such as the Elias Gamma (EG) codes [19], which facilitate the binary encoding of symbols selected from infinite sets, without requiring any knowledge of the corresponding occurrence probabilities at either

	Finite Symbol set e.g. $\{a, b, c, \ldots, z\}$	Infinite symbol set e.g. $\mathbb{N} = \{1, 2, 3, \ldots, \infty\}$
Separate Source and Channel Coding (SSCC)	• Shannon-Fano code [2] • Huffman code [10]	• Unary code [88] • Elias Gamma code [19]
Joint Source and Channel Coding (JSCC)	• Variable Length Error Correction (VLEC) code [57]	• **Unary Error Correction (UEC) code**

Table 1.3: The motivation for our UEC code.

the transmitter or receiver. In order to exploit the residual redundancy and hence to achieve near-capacity operation, the classic SSCC schemes may be replaced by JSCC arrangements [1], such as the VLEC code [57]. However, the decoding complexity of all previous JSCCs, such as Reversible Variable Length Code (RVLC) [56] and VLEC codes [57], increases rapidly with the cardinality of the symbol set becoming excessive for the cardinality of the symbols produced by practical multimedia encoders, such as H.264 [89] and H.265 [90], and asymptotically tending to infinity, when the cardinality is infinite. Against this background, we propose a novel JSCC scheme, which is referred to as the Unary Error Correction Code (UEC). As shown in Table 1.3, our UEC is designed to fill the gap for the combination of JSCC and infinite source symbol sets.

As it will be introduced in Chapter 3, our UEC encoder generates a bit sequence by concatenating unary codewords [88], while the decoder employs a trellis that has only a modest complexity, even when the cardinality of the symbol value set is infinite. This trellis is designed so that the transitions between its states are synchronous with the transitions between the consecutive unary codewords in the concatenated bit sequence. This allows the UEC decoder to exploit the residual redundancy using the classic Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm [91]. Furthermore, we prove that in the case of arbitrary symbol value distributions, the capacity loss asymptotically approaches zero, as the complexity of the UEC trellis is increased. In fact, we show that the capacity loss closely approaches zero, even if only a modest trellis complexity is employed.

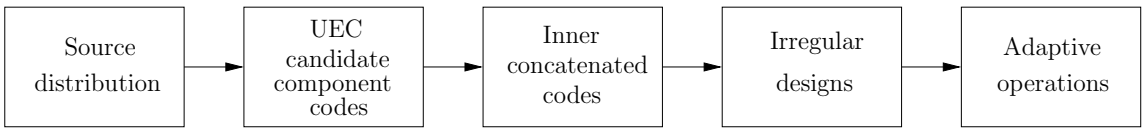

Figure 1.3: The design-flow of a UEC coded scheme.

In this treatise, we introduce the novel concept of UEC coding and investigate its applications, characteristics and performance in the context of digital telecommunications. As depicted in Figure 1.3, there are a range of different aspects that have to be considered, when designing UEC codes. In each of the following chapters, we will focus on and make

contributions to one or more of the aspects seen in Figure 1.3. The thesis is based on the four journal papers [92–95] and two conference papers [96, 97] shown in the **List of Publications**. The novel contributions of the thesis are as follows.

In Chapter 3, we will study the encoding and decoding operations of the UEC scheme and quantify the computational complexity of the UEC scheme.

- An iteratively-decoded serial concatenation of the UEC code and an Irregular Unity-Rate Convolutional (IrURC) code [98] is designed, which is capable of asymptotically eliminating any capacity loss, hence achieving near-capacity operation.
- It is formally shown that when the symbol values obey a geometric probability distribution, the UEC scheme eliminates all capacity loss, even when the UEC trellis has a very low complexity. It is also shown that for arbitrary source distributions, the capacity loss asymptotically approaches zero as the complexity of the UEC trellis is increased. Three scenarios associated with different source distributions are considered.
- We quantify the computational complexity of the Log-BCJR decoder in terms of the number of Add, Compare and Select (ACS) operations [96], for the sake of providing fair comparisons between our UEC scheme and the benchmarkers.

In Chapter 4, we will propose an *Adaptive* UEC design for expediting the attainable iterative decoding convergence.

- A novel three-stage concatenation of the UEC code and a half-rate turbo code is proposed, in which there are three decoders, hence facilitating a Three-Dimensional (3D) EXtrinsic Information Transfer (EXIT) chart [99] analysis and a novel adaptive iterative decoding algorithm.
- We employ the above-mentioned 3D EXIT chart to estimate the potential quantitative benefits associated with activating each decoder at each stage of the iterative decoding process. Then a more intuitive Two-Dimensional (2D) EXIT chart projection is employed to provide insights into whether or not any capacity loss is expected for the scheme.
- Based on the EXIT analysis, we dynamically adapt the activation order of the three decoders of the UEC-Turbo scheme, in order to expedite the iterative decoding convergence of the whole scheme. We also adaptively adjust the number of states that are employed in the UEC trellis, providing a reduced complexity and/or an improved performance.

In Chapter 5, we will propose an *Irregular* UEC design for achieving 'nearer-capacity' operation.

- A novel single irregular UEC trellis is designed, which is referred to as the Irregular Trellis (IrTrellis). This allows the irregularity to be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis, hence facilitating nearer-to-capacity operation than the state-of-the-art approach.
- In order to select UEC candidate codes having a desirable performance, the free-distance properties of the UEC codebooks are characterised for the first time, using a heuristic method that is capable of obtaining an approximate measurement of the free-distance.
- We also propose a novel extension of the double-sided EXIT chart matching algorithm of [100] that can be employed for jointly designing the EXIT function matching between the IrUEC and IrURC constituent codes.

In Chapter 6, we will propose a *Learning-aided* UEC design for operating on non-stationary and unknown sources.

- We analyse and characterise the symbol values that are entropy-encoded in the H.265 video codec. Inspired by the distribution of these symbol values, we propose a non-stationary probability distribution for modelling the source distribution, which can be readily parametrized to represent the H.265 distribution.
- A novel learning-aided UEC scheme is proposed. In contrast to the conventional UEC schemes, the learning-aided scheme does not require any prior knowledge of the source distribution at the receiver in order to achieve near-capacity operation. The algorithm operates by learning the source distribution based on the received symbols and then by feeding this information back to the decoder in order to assist the decoding process.
- In order to implement the learning algorithm, we employ a memory storage at the receiver, which is used to store the source distribution statistics that have been observed from successively recovered symbol vectors. We quantify the size of this memory storage in terms of the number of the most-recently recovered symbol vectors, as well as considering how to strike a desirable trade-off between the size of the memory and the error correction capability.

1.4 Thesis Organisation

The organization of the thesis is listed below, with reference to Figure 1.4:

Chapter 2: In this chapter, we introduce all background knowledge relied upon by our discussions of the UEC in the following chapters. We commence with the two basic structures of iterative decoding schemes, namely serially concatenation and

parallel concatenation. Following this, we focus our attention on the serially concatenated scheme and successively describe how each module of the transmitter operates, including the Unity-Rate Convolutional (URC) encoder, multiplexer, interleaver, Quadrature Phase-Shift Keying (QPSK) modulator, as well as how each corresponding module operates in the receiver, including the URC decoder, demultiplexer, deinterleaver and QPSK demodulator. We also highlight how the uncorrelated narrow-band Rayleigh fading channel affects the transmitted symbols.

Chapter 3: In this chapter, we commence by characterizing the symbol value sets that have an infinite cardinality by considering the geometric distribution, the zeta distribution and the H.264 distribution. We study a serially concatenated UEC-IrURC scheme and detail the operations of both the UEC encoder and decoder, as well as the iterative decoding operations exchanging extrinsic information between the UEC decoder and the IrURC decoder. We introduce the EXIT chart concept and its area properties in the context of the UEC code, demonstrating semi-analytically that near-capacity operation can be achieved depending on the UEC trellis decoder's complexity, regardless of the specific symbol value distribution. An SSCC EG-CC-IrURC benchmarker is introduced, and three different scenarios are considered in order to offer a deeper insight into the parametrizations of both the proposed scheme and the benchmarker. Moreover, we quantify the computational complexity of the UEC and EG-CC schemes in terms of the number of ACS operations, in order to strike a desirable trade-off between the contradictory requirements of low complexity and near-capacity operation.

Chapter 4: In this chapter, we propose an Adaptive UEC-Turbo scheme, which is a three-stage concatenated arrangement that applies an adaptive iterative decoding technique for expediting the iterative decoding convergence. With the assistance of a 3D EXIT chart analysis, we extend our adaptive iterative decoding approach, allowing the dynamic adjustment of the UEC decoder's operation, as well as of its activation order with the aid of two turbo decoder components. We also provide its decoding complexity and storage requirement analysis. Finally, the proposed Adaptive UEC scheme is compared to four benchmarkers.

Chapter 5: In this chapter, we propose a novel Irregular UEC scheme that operates on the basis of a single irregular trellis, which we refer to as the IrUEC code. Like the regular UEC code, it employs a unary code, but replaces the UEC's trellis code with a novel Irregular Trellis (IrTrellis) code, which operates on the basis of a single amalgamated irregular trellis, rather than a number of separate trellises. This allows the irregularity of the proposed IrUEC code to be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis, hence facilitating nearer-to-capacity

operation than the state-of-the-art. The IrUEC code is constructed from several UEC codes employing different UEC codebooks. In order to select UEC codebooks having an attractive performance, the free-distance properties of the UEC codebooks are characterised for the first time, using a heuristic method that is capable of estimating the free-distance. A novel extension of the double-sided EXIT chart matching algorithm is proposed for jointly designing the EXIT function matching between the IrUEC and IrURC codes. We also construct a Parallel UEC benchmarker and two versions of the SSCC EG-IrCC-IrURC benchmarkers, namely the EG-IrCC(sys)-IrURC benchmarker and the EG-IrCC(nonsys)-IrURC benchmarker, respectively.

Chapter 6: In this chapter, we commence by analysing the nature of the source symbol distribution in multimedia application by introducing a non-stationary zeta distribution inspired by the non-stationary nature of the symbols produced by the H.265 video encoder. We propose a learning-aided UEC scheme, which facilitates near-capacity operation without requiring any knowledge of the symbol occurrence probabilities at either the transmitter or at the receiver. The learning algorithm is capable of heuristically and iteratively estimating the source symbol statistics based on the recovered symbol vectors, which are then fed back to the trellis decoder as *a priori* information, in order to improve the receiver's error correction capability. We also propose a pair of SSCC benchmarkers that employ a similar learning technique, namely a learning-aided EG-CC scheme and a learning-aided Arithmetic-CC scheme, as well as their corresponding idealized but impractical versions, in order to characterize the upper bounds on their performance.

Chapter 7: In this chapter, we summarise the thesis and the main findings of our investigations, before offering design guidelines based on the previous chapters. Finally, we present a number of avenues for future research.

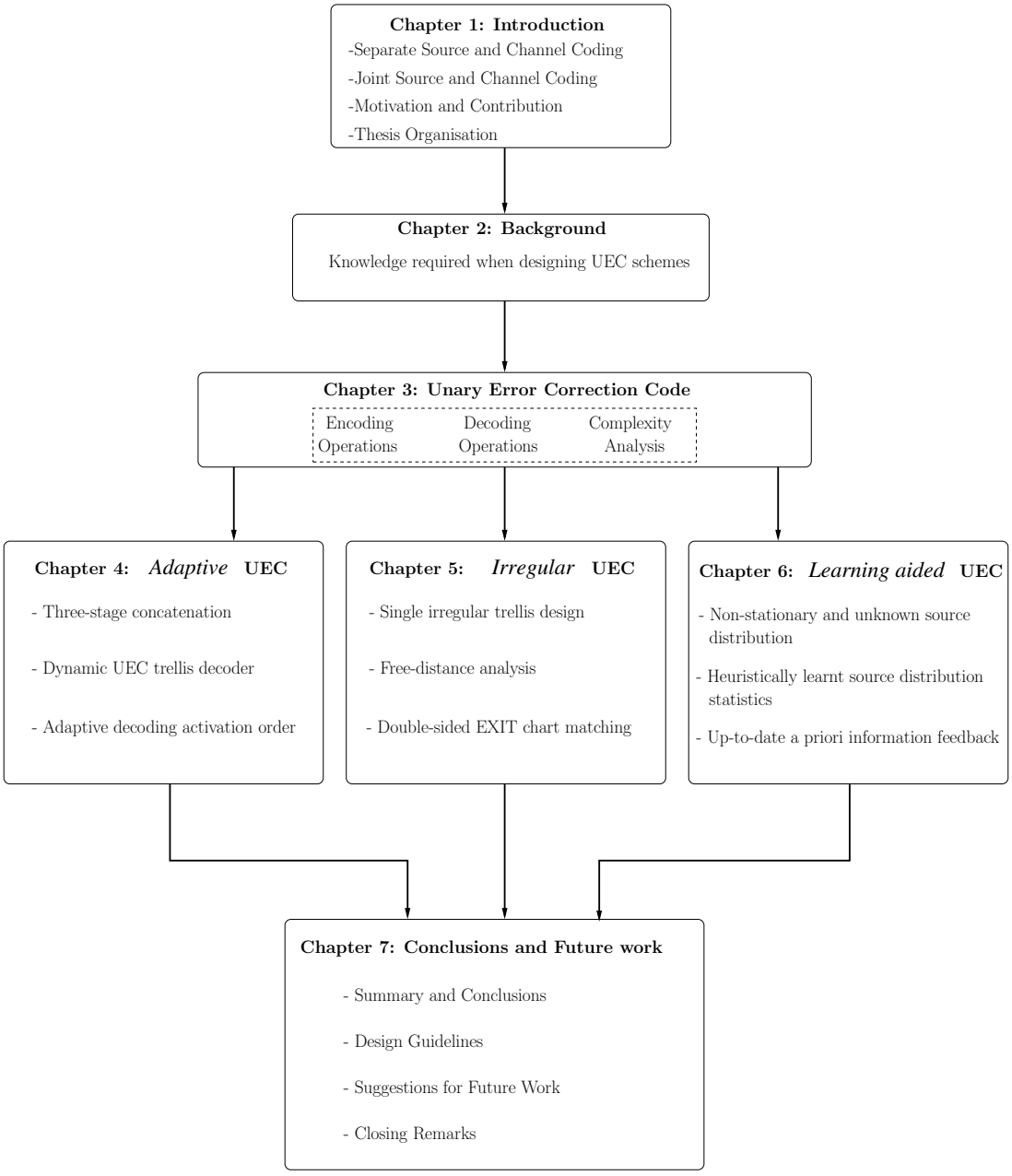

Figure 1.4: The structure of the thesis.

Chapter 2

Background

2.1 Introduction

In this chapter, we present the background knowledge required for designing a Unary Error Correction (UEC) scheme. More specifically, this chapter portrays the state-of-the-art proceeding the introduction of the UEC code, whilst the discussion of the UEC code itself is postponed to Chapter 3. As highlighted in Figure 2.1, this chapter refers to, but it is not constrained to, the topics related to the inner code design and irregular code design. The techniques that are introduced in this chapter are widely used in wireless communication systems. In the following chapters, we will frequently refer back to the knowledge that has been introduced in this chapter.

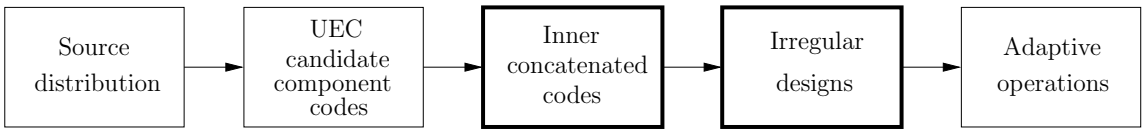

Figure 2.1: The design-flow of a UEC coded scheme. This chapter deals with the design aspects in the order indicated using the bold boxes.

The rest of this chapter is organised as follows. In Section 2.2, we commence by introducing two basic concatenated iterative source-channel coding schemes, namely serial and parallel concatenated codes, respectively. Then in the following sections, we focus our attention on serial concatenated scheme and describe how each module operates. In Sections 2.3 and 2.11, we detail how the encoder and decoder of Unity-Rate Convolutional (URC) code operate, respectively, while in Sections 2.4 and 2.10, we introduce the multiplexer and demultiplexer concepts. Then in Sections 2.5 and 2.9, the random interleaver and its corresponding deinterleaver are described, respectively, followed by the Quadrature Phase-Shift Keying (QPSK) modulator and soft demodulator

in Sections 2.6 and 2.8, respectively. In Section 2.7, we introduce the wireless channel model used throughout this thesis, namely the uncorrelated narrow-band Rayleigh fading channel. In Section 2.12, we describe the iterative decoding process exchanging extrinsic information between the two URC decoders of the serially concatenated scheme and characterize its Bit Error Ratio (BER) performance. We then introduce then powerful analysis tool of EXtrinsic Information Transfer (EXIT) charts in Section 2.13, which may be used to analyse the convergence behaviour of iteratively decoded schemes. In Section 2.14, we discuss the associated irregular code design, which may be employed to facilitate the near-capacity operation at low channel Signal to Noise Ratio (SNR), and conclude this chapter in Section 2.15.

2.2 Concatenated Schemes

Several commonly used concatenated schemes have been conceived for iterative source-channel coding, namely serial concatenation [101], parallel concatenation [32] and hybrid concatenation of various constituent codes. In our work, we mainly focus on the family of serial and parallel concatenated schemes, which will be introduced in Section 2.2.1 and Section 2.2.2, respectively.

2.2.1 Serially Concatenated Schemes

The serial concatenation constitutes a general structure and many decoding and detection schemes can be described as serially concatenated structures, such as those used in turbo equalization [102], Joint Source and Channel Coding (JSCC) [103, 104], Low-Density Parity-Check (LDPC) coding [105] and so on. The basic structure of Serially Concatenated Codes (SCC) [101] is composed of at least two codes, which are referred to as the inner and the outer codes that are interconnected by an interleaver. An example of an SCC having two constituent components is shown in Figure 2.2, where both the inner and the outer codes are URC codes [106].

At the encoder of Figure 2.2(a), the bit vector $\mathbf{a}$ is firstly encoded by the outer URC encoder, resulting in the bit vector $\mathbf{b}$, which has the same length as $\mathbf{a}$. Then a multiplexer is employed for concatenating the bit vectors $\mathbf{a}$ and $\mathbf{b}$, in order to create a bit vector $\mathbf{c}$ having a length twice that of $\mathbf{a}$ and resulting in an outer coding rate of $1/2$. The introduction of the interleaver π_1 scrambles the bits of $\mathbf{c}$ before they are passed to the inner URC encoder, providing a practical manifestation of time diversity. To elaborate a little further, this ensures that even if a specific bit has been gravely contaminated by the channel, the chances are that the other constituent decoder is still capable of providing reliable information concerning this bit [107]. The interleaved bit vector $\mathbf{d}$ is then encoded by the inner URC encoder to provide the bit vector $\mathbf{e}$ and interleaved by π_2 to obtain the bit vector $\mathbf{f}$, which

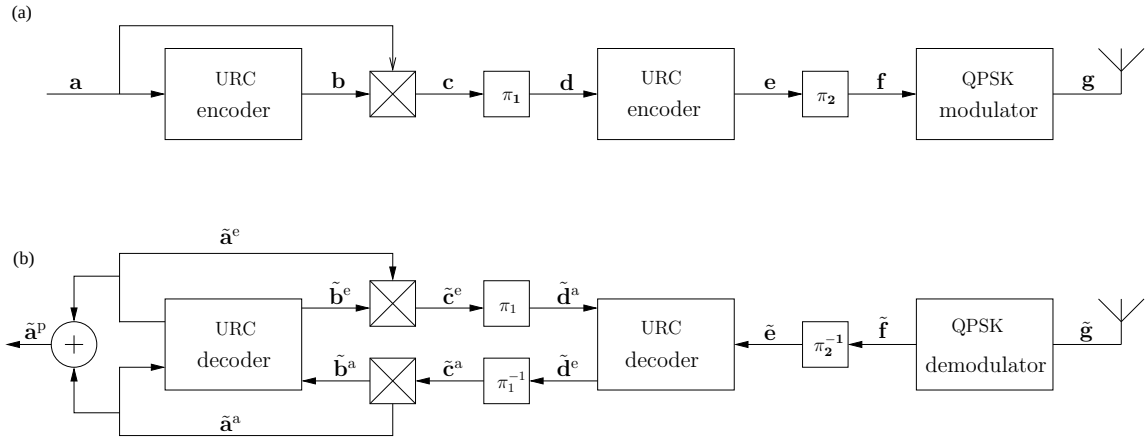

Figure 2.2: Schematic of a serially concatenated code, including (a) encoder and (b) decoder, in which both the component codes are URC codes. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers, respectively. A multiplexer is employed in order to give a half-rate outer code. Bold notation without a diacritic is used to denote a bit vector. A diacritical tilde represents an LLR frame pertaining to the bit vector with the corresponding notation. The superscripts 'a', 'e' and 'p' denote *a priori*, extrinsic and *a posteriori* LLRs, respectively.

is modulated by the QPSK modulator in order to obtain the symbol vector $\mathbf{g}$. Without loss of generality, here we consider transmission over an uncorrelated narrow-band Rayleigh channel.

In the receiver of Figure 2.2(b), a QPSK demodulator converts the vector $\tilde{\mathbf{g}}$ of received symbols into the vector $\tilde{\mathbf{f}}$ of Logarithmic Likelihood Ratio (LLR). These soft-bits express not only what the most likely value of each bit in $\mathbf{f}$ is, but also how likely or reliable that value is. A deinterleaver π_2^{-1} is employed to convert the Logarithmic Likelihood Ratio (LLR) vector $\tilde{\mathbf{f}}$ into the reordered vector $\tilde{\mathbf{e}}$, before invoking iterative decoding for exchanging extrinsic information between the outer URC decoder and the inner URC decoder. More particularly, the outer URC decoder and the inner URC decoder sequentially exchange the *a priori* LLR vectors, $\tilde{\mathbf{c}}^{\mathrm{a}}$ and $\tilde{\mathbf{d}}^{\mathrm{a}}$, and the extrinsic LLR vectors, $\tilde{\mathbf{c}}^{\mathrm{e}}$ and $\tilde{\mathbf{d}}^{\mathrm{e}}$, while deinterleaving is performed to convert $\tilde{\mathbf{d}}^{\mathrm{e}}$ to $\tilde{\mathbf{c}}^{\mathrm{a}}$, as seen in Figure 2.2. Demultiplexing is employed for separating the vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ into sub-vectors $\tilde{\mathbf{a}}^{\mathrm{a}}$ and $\tilde{\mathbf{b}}^{\mathrm{a}}$. Meanwhile, observe in Figure 2.2 that multiplexing combines $\tilde{\mathbf{a}}^{\mathrm{e}}$ and $\tilde{\mathbf{b}}^{\mathrm{e}}$ to obtain $\tilde{\mathbf{c}}^{\mathrm{e}}$. With this iterative exchange of soft information, the two decoders become capable of providing each other with increasingly reliable LLRs. After a certain number of iterations, the *a posteriori* LLR vector $\tilde{\mathbf{a}}^{\mathrm{p}} = \tilde{\mathbf{a}}^{\mathrm{a}} + \tilde{\mathbf{a}}^{\mathrm{e}}$ of Figure 2.2 is finally obtained and used to make a hard-decision about the value of the recovered bits.

In Sections 2.3 to 2.14, we will further detail the serially concatenated scheme of Figure 2.2, by sequentially introducing each module by following the bit flow in the encoding and decoding process.

2.2.2 Parallel Concatenated Schemes

The parallel concatenated structure is employed in the classic turbo codes [32]. The component encoders are often CCs, but binary Bose-Chaudhuri-Hocquenghem (BCH) codes [108] have also been used. In this section, we provide an example of Parallel Concatenated Codes (PCC), in which URC codes are used as the component codes, as shown in Figure 2.3.

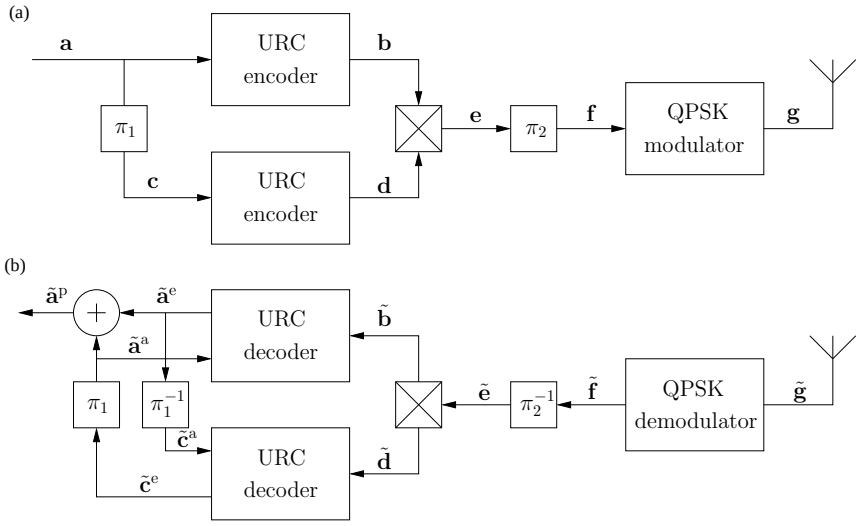

Figure 2.3: Schematic of a parallel concatenated code, including (a) encoder and (b) decoder, in which both the component codes are URC codes. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers, respectively. A multiplexer is employed in order to acquire a half-rate parallel concatenated code. Bold notation without a diacritic is used to denote a bit vector. A diacritical tilde represents an LLR vector pertaining to the bit vector with the corresponding notation. The superscripts 'a', 'e' and 'p' denote *a priori*, extrinsic and *a posteriori* LLRs, respectively.

In the transmitter of Figure 2.3(a), the bit vector $\mathbf{a}$ is encoded by the upper URC encoder, and the bit vector $\mathbf{c}$ that is a π_1-interleaved version of $\mathbf{a}$ is encoded by the lower constituent URC encoder. The pair of bit vectors, $\mathbf{b}$ and $\mathbf{d}$, that are output by the two URC encoders may be punctured in order to obtain arbitrary coding rates. In our example shown in Figure 2.3, the URC-encoded bit vectors $\mathbf{b}$ and $\mathbf{d}$ are multiplexed, hence a half-rate parallel code is obtained. The resultant bit vector $\mathbf{e}$ of Figure 2.3 is interleaved by π_2 to obtain $\mathbf{f}$ and then modulated by the QPSK modulator. The transmitter outputs the resultant symbol vector $\mathbf{g}$, which is conveyed over an uncorrelated narrow-band Rayleigh

fading channel. Note that this structure can be extended to a parallel concatenation of more than two component codes, leading to multiple-stage codes [109, 110].

In the receiver of Figure 2.3(b), the received symbol vector $\tilde{\mathbf{g}}$ is demodulated to give the LLR vector $\tilde{\mathbf{f}}$, which is then deinterleaved to give $\tilde{\mathbf{e}}$ and demultiplexed to give $\tilde{\mathbf{b}}$ and $\tilde{\mathbf{d}}$. Iterative processing is employed for exchanging extrinsic information between the two component URC decoders of the PCC scheme. Similar to that of the SCC scheme of Figure 2.2, the iterative decoder exchanges the *a priori* LLR vectors, $\tilde{\mathbf{a}}^{\mathrm{a}}$ and $\tilde{\mathbf{c}}^{\mathrm{a}}$, and the extrinsic LLR vectors, $\tilde{\mathbf{a}}^{\mathrm{e}}$ and $\tilde{\mathbf{c}}^{\mathrm{e}}$, between the two URC decoders, hence providing increasingly reliable information for each other. Finally, the *a posteriori* LLR vector $\tilde{\mathbf{a}}^{\mathrm{p}} = \tilde{\mathbf{a}}^{\mathrm{a}} + \tilde{\mathbf{a}}^{\mathrm{e}}$ of Figure 2.3 is obtained to make the final decision for the decoded bits.

Note that while Sections 2.3 to 2.14 focus on the serially concatenated scheme of Figure 2.2, their discussions are also analogously relevant to the parallel concatenated scheme of Figure 2.3.

2.2.3 Self-Concatenated Schemes

In this section, we provide an example of Self-Concatenated Codes (SECC) [111], in which a URC code is used as the component code, as shown in Figure 2.4.

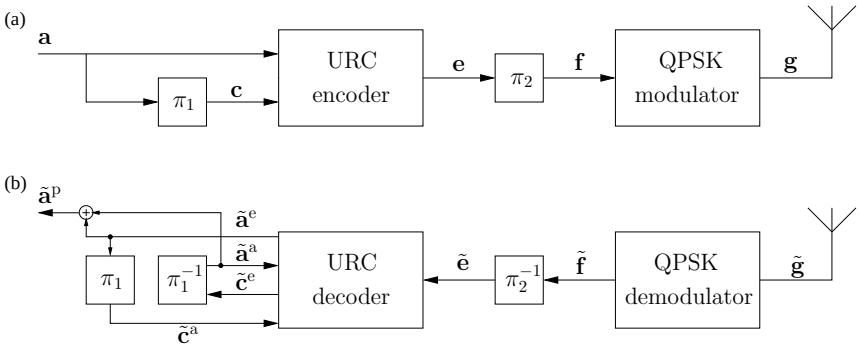

Figure 2.4: Schematic of a self-concatenated code, including (a) encoder and (b) decoder, in which the component code is provided by a URC code. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers, respectively. Bold notation without a diacritic is used to denote a bit vector. A diacritical tilde represents an LLR vector pertaining to the bit vector with the corresponding notation. The superscripts 'a', 'e' and 'p' denote *a priori*, extrinsic and *a posteriori* LLRs, respectively.

A SECC is similar to a PCC when its two component codes are replaced by a single component code employing an odd-even separated turbo interleaver, as discovered in [112]. SECC exhibits a low complexity, since it invokes only a single encoder as depicted in Figure 2.4(a) and only a single decoder as shown in Figure 2.4(b).

2.3 Unity-Rate Convolutional code

As shown in Figure 2.2, the source bit vector $\mathbf{a}$ is forwarded to the URC [113] encoder. The URC code may be employed either as an intermediate code [114] or as a precoder [115], in order to improve the attainable decoding convergence and to achieve an infinitesimally low BER. In Section 2.3.1, we will highlight how to construct a URC code using a Linear Feedback Shift Register (LFSR). Section 2.3.2 details how to invoke polynomials for mathematically describing the URC code. Finally, Section 2.3.3 uses a truth table, a state transition diagram and a trellis diagram for demonstrating how the URC code works.

2.3.1 Linear Feedback Shift Register

A URC code constitutes a special case of a Convolutional Code (CC) [116] having a coding rate of 1, which implies that the number of bits output by the encoder is equal to the number input. The URC encoder can be constructed using a LFSR structure, which is composed of binary memory elements and modulo-2 adders. The general LFSR structure of a URC encoder having m binary memory elements is illustrated in Figure 2.5. Each memory element can hold one bit, having either a 0-state or 1-state. As a result, the total number of states for memory m registers is $2^m = r$, where r is defined as the number of states that can be adopted by the URC encoder. The modulo-2 adder shown in Figure 2.5 can be simply implemented using a single Boolean Exclusive-OR (XOR) gate.

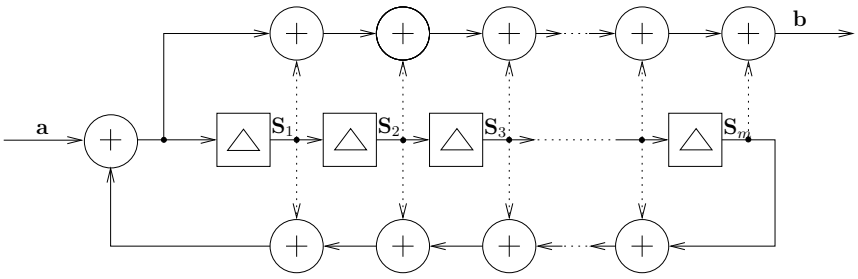

Figure 2.5: A general linear feedback shift register structure of a URC encoder, where $\mathbf{a}$ is the input bit vector and $\mathbf{b}$ is the corresponding output bit vector.

Note that, unless otherwise specified, all registers initially store a value of 0, which means that all the m registers of Figure 2.5 are initially at their 0-states. More particularly, the state of each register is sequentially driven by each successive bit in the vector $\mathbf{a}$ that is fed into the left-most register. Then the bit stored in each register is sequentially passed to its next register, resulting in one output bit in $\mathbf{b}$. Since the overall coding rate of URC codes is 1, the bit vectors $\mathbf{a}$ and $\mathbf{b}$ have the same length.

The selection of dotted arrows seen in Figure 2.5 can be invoked for creating a particular combination of connections in the forward and backward paths, according to the

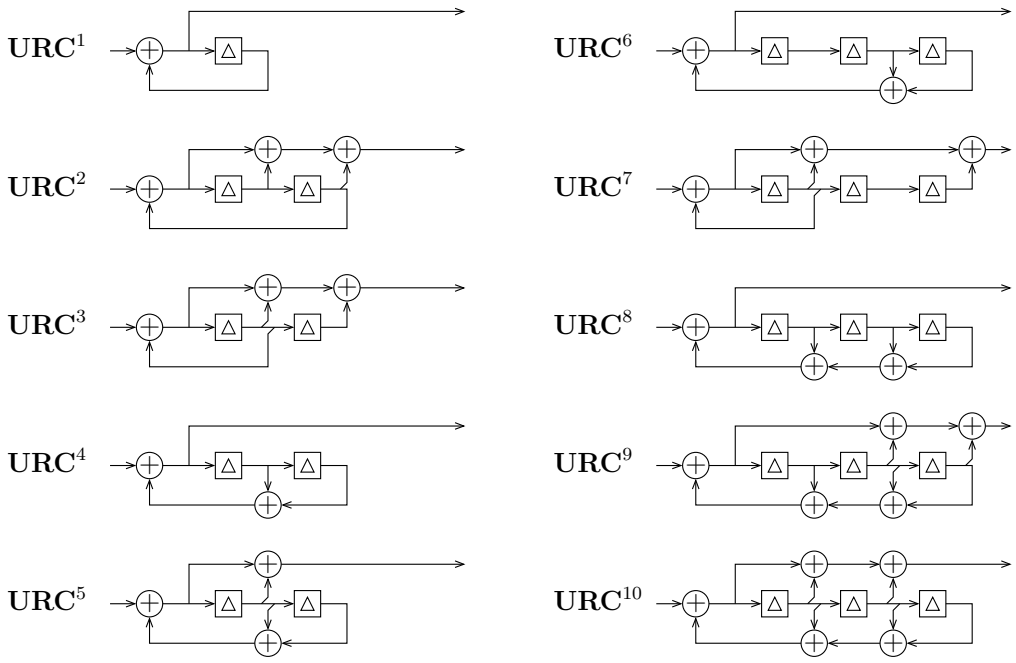

Figure 2.6: The Linear Feedback Shift Register (LFSR) encoder structures for the $T = 10$ URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ [98].

specific parametrization of the URC encoder. For example, the LFSR designs [98] of Figure 2.6 can be employed to construct $T = 10$ URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ from the set of all possible designs, containing a number of registers in the set $m \in \{1, 2, 3\}$. These ten URC codes are selected as they can provide sufficiently diverse EXIT characteristics [99], as it will be demonstrated in Section 2.13.

2.3.2 Generator and Feedback Polynomials

Generator and feedback polynomials can be employed in pairs to uniquely identify each URC encoder parametrization mathematically. Conventionally, a code generator matrix is a k-by-n-element matrix, if the code gives n output bits for every k input bits. The specific element in the ith row and the jth column indicates how the ith input contributes to the jth output. For a URC code, the generator matrix can be simplified to a scalar, since we have $k = n = 1$. Likewise, the feedback polynomial of a URC code may be simplified to a scalar. More specifically, both may be denoted by a hexadecimal number, as exemplified in Table 2.1.

r	$r = 2$	$r = 4$				$r = 8$				
URC	URC^1	URC^2	URC^3	URC^4	URC^5	URC^6	URC^7	URC^8	URC^9	URC^{10}
(g, f)	(2,3)	(7,5)	(7,6)	(4,7)	(6,7)	(8,B)	(D,C)	(8,F)	(B,F)	(E,F)

Table 2.1: The $T = 10$ URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ of Figure 2.6 are denoted by using the format $(g(\mathrm{URC}), f(\mathrm{URC}))$, where $g(\mathrm{URC})$ and $f(\mathrm{URC})$ are the hexadecimal generator and feedback polynomials of the URC code [98], respectively.

As illustrated in Figure 2.6, the left-most bit in the binary representation of the hexadecimal number specifies the current input, while the rightmost bit represents the oldest input that still remains in the shift register. In order to obtain the generator and feedback polynomials of the URC code, we build a binary number representation by placing a 1 at each spot, where a connection line from the shift register feeds into the adder, and a 0 elsewhere. Here, the upward facing dotted arrows in Figure 2.5 correspond to the generator polynomial, while the downward facing arrows contribute to the feedback polynomial. Let us consider the URC^7 code of Figure 2.6 as an example, where the generator polynomial is $g(\mathrm{URC}^7) = 1101$, while the feedback polynomial is $f(\mathrm{URC}) = 1100$. By converting the binary representation into hexadecimal representation, we have $g(\mathrm{URC}^7) = \mathrm{D}$ and $f(\mathrm{URC}) = C$, respectively. Similarly, the generator and feedback polynomials of the $T = 10$ URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ of Figure 2.6 can be found in Table 2.1.

2.3.3 Encoding Operation

In this section, we will demonstrate how the URC encoder operates by using the LFSR structure, the truth table, the state transition diagram and the trellis diagram. Without loss of generality, we consider the URC^1 code of Table 2.1 as an example.

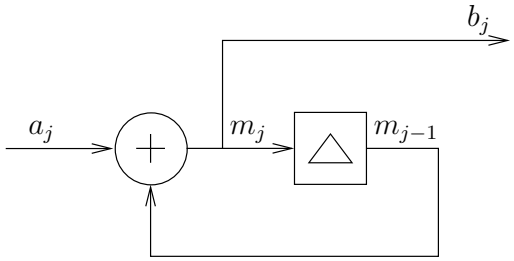

Figure 2.7: The Linear Feedback Shift Register (LFSR) encoder of the URC^1 code, including one shift register and one modulo-2 adder. Here, we denote the input bit by a_j and the corresponding output bit by b_j. Meanwhile, m_{j-1} and m_j are the previous state and the current state of the shift register, respectively.

As shown in Figure 2.7, there is only a single shift register element and one modulo-2 adder in the LFSR schematic of the URC^1 code. For each input bit a_j of the vector $\mathbf{a} = [a_j]_{j=1}^{J}$, the encoder will update the current state m_j of its shift register according to the feedback polynomial, and output bit b_j of the vector $\mathbf{b} = [b_j]_{j=1}^{J}$ according to the generator polynomial. Therefore, we have

$$\begin{aligned} m_j &= a_j \oplus m_{j-1}, \\ b_j &= m_j, \end{aligned} \tag{2.1}$$

where m_{j-1} is the previous state of the shift register and m_j is the current state. Note that the initial state of the register $m_0 = 0$, since all shift register memory elements are initialized to store a logic-0, as discussed in Section 2.3.1. A J-bit input vector $\mathbf{a} = [a_j]_{j=1}^{J}$ yields a $(J+1)$-element state vector $\mathbf{m} = [m_j]_{j=0}^{J}$, comprising each successive state held by the shift register. For example, the input bit vector $\mathbf{a} = [10010]$ yields a state vector $\mathbf{m} = [0, 1, 1, 1, 0, 0]$, as well as the output bit vector $\mathbf{b} = [11100]$.

m_{j-1}	a_j	m_j	b_j
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Table 2.2: The truth table of the URC^1 code shown in Figure 2.7. Here, we denote the input bit by a_j and the corresponding output bit by b_j. And m_{j-1} and m_j are the previous state and the current state of the shift register, respectively.

According to the logic connection in Eq. (2.1), we can derive the truth table of the URC^1 code, as shown in Table 2.2. More particularly, given an initial state m_{j-1} and an input bit a_j, we can directly read the next state m_j and the output bit b_j from the table.

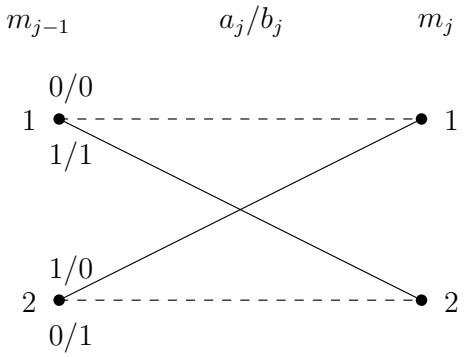

Figure 2.8: The state transition diagram of the URC^1 code shown in Figure 2.7. Here, we denote the input bit by a_j and the corresponding output bit by b_j. And m_{j-1} and m_j are the previous state and the current state of the shift register, respectively.

Furthermore, we may characterise the operations of the truth table of Table 2.2 in a state transition diagram, as illustrated in Figure 2.8. It is clear that each input bit a_j forces the trellis encoder to traverse from its particular previous state m_{j-1} into a new state m_j that is selected from two legitimate alternatives, whilst outputting the bit b_j at the same time. In contrast to the truth table, we denote the register state by its state index, rather than the physical register state, for the sake of avoiding confusion between the states and the input or output bits. For example, the set of all possible states of the shift register shown in Figure 2.7 may be denoted by $\mathbf{M} = \{1, 2\}$, rather than using the corresponding binary

contents of the memory elements 0 and 1, respectively. Hence, we have $m_{j-1} \in \{1, 2\}$ and $m_j \in \{1, 2\}$, as illustrated in Figure 2.8. This is particularly useful, when the URC code has more than two states, hence avoiding the use of binary strings to identify each state. In our following chapters, we will employ the state transition diagram to describe the proposed UEC codes.

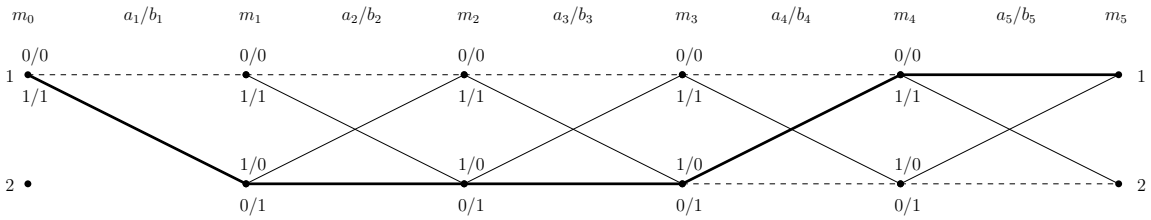

Figure 2.9: The trellis diagram of the URC^1 code shown in Figure 2.7. Here, each line connecting two states denotes the transformation between the previous and the new state of the register. At each state, the first bit before slash is the input bit and the second is the output bit. The bold line demonstrates the coding process of the input bit vector $\mathbf{a} = [10010]$.

As depicted in Figure 2.9, we can portray the trellis diagram of the URC^1 code by concatenating the state transition diagrams. This allows us to see how the state transitions flow from one to another during the encoding of successive bits in the vector $\mathbf{a}$. For example, the encoding process of the input bit vector $\mathbf{a} = [10010]$ is demonstrated by the bold line of Figure 2.9. The trellis diagram will be shown to be especially useful, when designing the *Irregular* Unary Error Correction (IrUEC) code of Chapter 5. Note that the state transition diagram for the first bit a_1 is pruned, since having an initial state of $m_0 = 1$ is guaranteed.

2.4 Multiplexer Operation

In the serially concatenated code of Figure 2.2, a multiplexer (or MUX) is employed to combine the bit vector $\mathbf{a}$ and the URC-encoded bit vector $\mathbf{b}$ in order to obtain a coding rate of $1/2$. Generally, the function of a multiplexer is to combine multiple input data streams into a single data stream. In electronics, the multiplexer is a device that may select one of several analog or digital input signals and forwards the selected input to the single output [117].

A basic multiplexer structure is depicted in Figure 2.10, where there are three input data streams, namely A, B, C, and only one selected for creating the output data stream D. Note that the signal selection may proceed in a sequential order, for example, [A, B, C, A, B, C, ...]. On the other hand, the signal selection may also be pre-defined in a specified order, e.g., [A, C, A, C, B, A, C, A, C, B, ...]. In the case of the schematic of Figure 2.2, the multiplexer combines the pair of bit vectors $\mathbf{a} = [10010]$ and $\mathbf{b} = [11100]$ having the

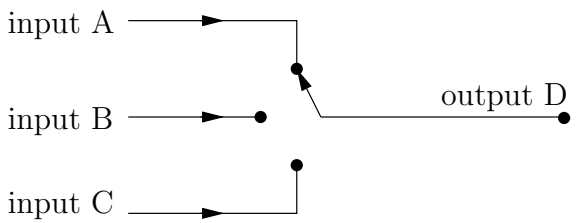

Figure 2.10: An example of multiplexer schematic, where there are three input data streams A, B and C, and one output data stream D.

length of $J = 5$ into a single bit vector $\mathbf{c} = [1001011100]$ having the length $2J = 10$, by concatenating the bit vector $\mathbf{b}$ to the end of the bit vector $\mathbf{a}$.

2.5 Interleaver Operation

As shown in Figure 2.2, the bit vector $\mathbf{c}$ is fed into an interleaver π_1, which is widely used in communication systems to disperse the bursts of errors imposed by fading [107]. The error correction code of the communication system may fail to recover the original codeword, if the errors within a received bit vector are too concentrated, too close together. An interleaver ameliorates this problem by shuffling the bit errors across the received bit vector, thereby creating a more uniform-distribution of errors [118].

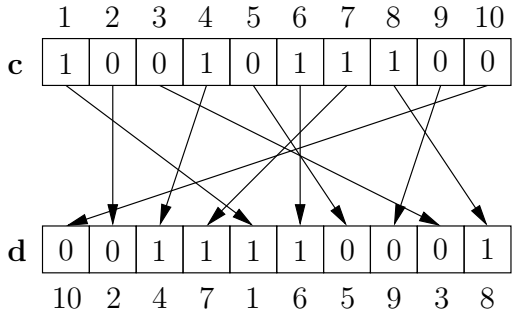

Figure 2.11: An example of random interleaver π_1 in Figure 2.2, having the random interleaver vector $\pi_1 = [10, 2, 4, 7, 1, 6, 5, 9, 3, 8]$. Here, $\mathbf{c}$ and $\mathbf{d}$ are the corresponding bit vectors shown in Figure 2.2.

A random design of the interleaver π_1 of Figure 2.2 is illustrated in Figure 2.11, where the interleaver imposes a random permutation that is known to both the transmitter and receiver. More particularly, the interleaver vector is denoted by $\pi_1 = [10, 2, 4, 7, 1, 6, 5, 9, 3, 8]$. As shown in Figure 2.11, each bit c_j of the vector $\mathbf{c} = [1001011100]$ swaps its position according to the interleaver vector π_1, resulting in the interleaved bit vector $\mathbf{d} = [0011110001]$, according to $d_j = c_{\pi_1(j)}$. For example, we have $d_1 = c_{10}$, since $\pi_1(1) = 10$. Similarly, the interleaver π_2 of Figure 2.2 also has a random interleaving design.

Apart from the random interleaver, there are other interleaver designs, such as the rectangular (or uniform) interleaver, convolutional interleaver and contention-free quadratic permutation polynomial (QPP) interleaver [119], which is used in the 3GPP Long Term Evolution (LTE) mobile telecommunication standard [120]. For the sake of adequately shuffling the bits, the S-random interleaver [121] has been proposed to ensure that no input bits within a distance of S appear within a distance of S from each other in the output bit vector. Moreover, a novel Evolutionary Algorithm (EA) capable of designing improved interleavers for SCCs was proposed in [122], without artificially limiting the achievable error correction capability, even if only a modest complexity can be afforded. In our following discussions, we will employ the random interleaver design, unless specified otherwise.

2.6 Quadrature Phase-Shift Keying

Following the inner URC encoder and the interleaver π_2 of Figure 2.2, QPSK modulation is invoked for the transmission of the bit vector $\mathbf{f}$ through the channel. Phase-Shift Keying (PSK) is a digital modulation scheme that maps the bits onto the phase of a carrier wave, in order to convey the data. Since QPSK modulation is a particularly common PSK scheme, we will mainly use QPSK modulation in our following investigations.

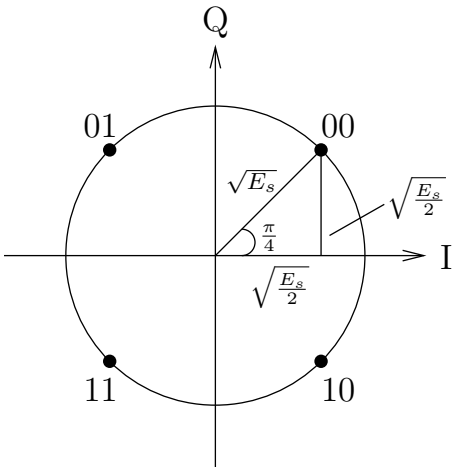

Figure 2.12: Constellation diagram for Quadrature Phase-Shift Keying (QPSK) modulation with Gray coding, where the label of each adjacent symbol only differs by one bit. Here, the real and imaginary axes are termed as the in-phase and quadrature axes, respectively.

A constellation diagram is a convenient way of characterising a PSK scheme, as shown in Figure 2.12 for QPSK relying on Gray coding [123]. The Gray coding is invoked for minimizing the BER, since this results in each constellation point having only a single bit difference from its neighboring constellation points. As shown in Figure 2.12, the four QPSK constellation points are plotted in a complex plane, where the real and imaginary

axes are termed as the in-phase and quadrature axes, respectively. The amplitude of each point along the in-phase axis is used for modulating a cosine carrier wave, while that of the quadrature axis is used to modulate a sine carrier wave. In order to create the maximum phase-separation between the adjacent points and hence the best immunity to corruption, the four points on the constellation diagram have uniform angular spacing around a circle. Owing to this, all of the codewords can be transmitted with the same energy.

The amplitudes applied to the cosine and sine carriers by each constellation point can be combined to produce a single cosine carrier having a particular phase, according to

$$s_n(t) = \sqrt{\frac{2E_s}{T_s}} \cos[2\pi f_c t + (2n-1)\frac{\pi}{4}], \qquad n = 1, 2, 3, 4, \tag{2.2}$$

where E_s is the energy per symbol, T_s is the symbol duration time, f_c is the frequency of the carrier and $s_n(\cdot)$ is the signal in the time domain. This yields the four phases of $\pi/4$, $3\pi/4$, $5\pi/4$ and $7\pi/4$, corresponding to the four constellation points having $n = 1, 2, 3, 4$, which are Gray-mapped to the bits 00, 01, 11, 10, respectively, resulting in the four points of $\pm\sqrt{E_s/2} \pm \sqrt{E_s/2}i$. Without loss of generality, we assume that $E_s = 1$. Therefore, we have four points $(I, Q) = \pm 0.7071 \pm 0.7071i$ in the complex plane of Figure 2.12, each of which is mapped to one QPSK codeword.

For example, when the bit vector $\mathbf{f} = [1000110111]$ is fed into the QPSK modulator of Figure 2.2, the bits of $\mathbf{f}$ are processed two at a time to obtain the output symbol vector $\mathbf{g} = [0.7071{-}0.7071\text{i}, 0.7071{+}0.7071\text{i}, {-}0.7071{-}0.7071\text{i}, {-}0.7071{+}0.7071\text{i}, {-}0.7071{-}0.7071\text{i}]$. Now the symbol vector $\mathbf{g}$ is ready to be transmitted through the channel. Hence, we have introduced all operations of the transmitter shown in Figure 2.2(a).

The effective throughput of a QPSK-based communication scheme is given by $\eta = R_\text{o} \cdot R_\text{i} \cdot \log_2(M)$ bits per symbol, where $R_\text{o} = 1/2$ is the coding rate of the outer code of Figure 2.2, $R_\text{i} = 1$ is that of the inner code and QPSK has $M = 4$ constellation points.

2.7 Uncorrelated Narrow-band Rayleigh Channel

The symbol vector $\mathbf{g}$ of Figure 2.2 is transmitted through an uncorrelated narrow-band Rayleigh fading channel. Rayleigh fading is a reasonable model, when there are many objects in the environment that scatter the radio signal before it arrives at the receiver [124]. It is assumed that the magnitude of a signal that has passed through such an uncorrelated narrow-band Rayleigh channel will fade randomly according to a Rayleigh distribution, which is the radial component of the sum of two uncorrelated Gaussian random variables.

In general, a Rayleigh channel can be modeled by

$$\tilde{\mathbf{g}} = \mathbf{h} \cdot \mathbf{g} + \mathbf{n}, \tag{2.3}$$

where $\mathbf{g}$ is the input symbol vector of the channel, $\tilde{\mathbf{g}}$ is the output signal vector, $\mathbf{h}$ is the channel gain vector and $\mathbf{n}$ is the Additive White Gaussian Noise (AWGN) vector. The real part and imaginary part of a channel gain h may be denoted by $\mathrm{Re}(h)$ and $\mathrm{Im}(h)$, respectively. Likewise, the real part and imaginary part of a noise element n may be denoted by $\mathrm{Re}(n)$ and $\mathrm{Im}(n)$, respectively. Note that both the channel gain h and the noise n obey complex-valued Gaussian distributions. More specifically, both $\mathrm{Re}(h)$ and $\mathrm{Im}(h)$ have a mean of 0 and a variance of $1/2$, while both $\mathrm{Re}(n)$ and $\mathrm{Im}(n)$ have a mean of 0 and a variance of $N_0/2$, where N_0 is the power spectral density of the AWGN.

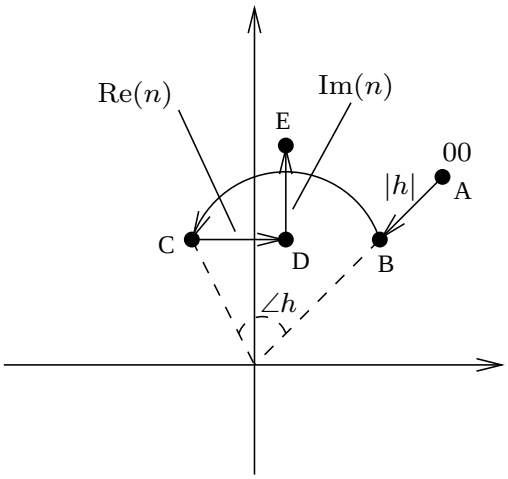

Figure 2.13: Effects of uncorrelated narrow-band Rayleigh channel on the 00 point in the constellation diagram of QPSK modulation, including amplitude fading, phase rotation and additive noise.

Generally, there are three different categories of effects that a Rayleigh channel imposes on the transmitted signals, including amplitude fading, phase rotation and additive noise. Considering the 00 constellation point in the first quadrant of Figure 2.12 for example, Rayleigh fading affects the magnitude of the signal according to $|h| = \sqrt{\mathrm{Re}(h)^2 + \mathrm{Im}(h)^2}$, which forces the constellation point to traverse from the point A to the point $\mathrm{B} = |h| \cdot \mathrm{A}$, as demonstrated in Figure 2.13. Meanwhile, the Rayleigh channel also rotates the signal from the point B to the point C by an angle of $\angle h = \arctan \dfrac{\mathrm{Im}(h)}{\mathrm{Re}(h)}$, where $\angle \mathrm{B} = \angle \mathrm{C} + \angle h$. Moreover, the AWGN corrupts the constellation point from the point C to the point $\mathrm{D} = \mathrm{C} + \mathrm{Re}(n)$, then from the point D to the point $\mathrm{E} = \mathrm{D} + \mathrm{Im}(n) \cdot \mathrm{i}$. Finally, the point E represents the corresponding received symbol in the vector $\tilde{\mathbf{g}}$, which may then be QPSK demodulated, as described in later Section 2.8. For example, assuming that we have $\mathbf{h} = [0.1818 - 0.0388\mathrm{i}, -0.6678 + 0.6443\mathrm{i}, -0.9346 + 0.4204\mathrm{i}, 0.6540 + 0.2476\mathrm{i}, 0.0000 +$

$0.8841\text{i}]$, $\mathbf{n} = [0.6575{-}1.1397\text{i}, 0.1695{-}0.0173\text{i}, -0.4882{-}1.3780\text{i}, -0.4607{+}0.7216\text{i}, 0.8429{+}0.6093\text{i}]$, upon transmitting the symbol vector $\mathbf{g} = [0.7071{-}0.7071\text{i}, 0.7071{+}0.7071\text{i}, -0.7071{-}0.7071\text{i}, -0.7071 + 0.7071\text{i}, -0.7071 - 0.7071\text{i}]$, the received symbol vector becomes $\tilde{\mathbf{g}} = \mathbf{h} \cdot \mathbf{g} + \mathbf{n} = [0.7585 - 1.2957\text{i}, -0.7582 - 0.0339\text{i}, 0.4700 - 1.0144\text{i}, -1.0982 + 1.0089\text{i}, 1.4680 - 0.0158\text{i}]$.

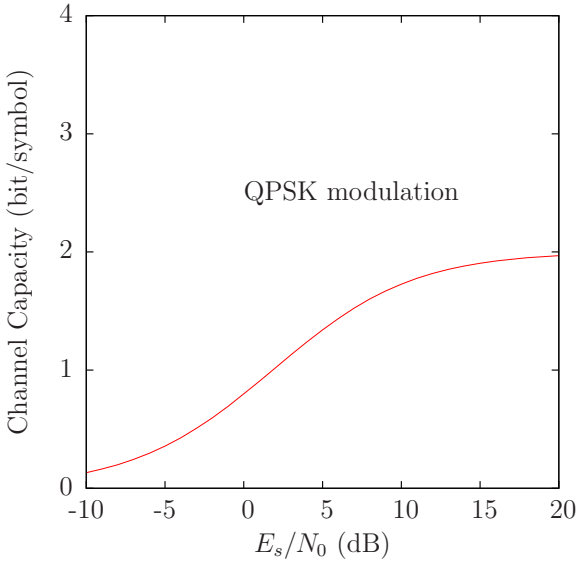

Figure 2.14: The Discrete-input Continuous-output Memoryless Channel (DCMC) capacity of uncorrelated Rayleigh fading channels for QPSK modulation.

The Discrete-input Continuous-output Memoryless Channel (DCMC) capacity [125] of uncorrelated Rayleigh fading channels for QPSK is shown in Figure 2.14. The DCMC capacity provides an upper bound on the effective throughput η, for which reliable communication is possible. Near-capacity communication is facilitated, if a communication scheme can be designed that achieves reliable communication using an effective throughput η that closely approaches the DCMC capacity. Near-capacity operation is of particular interest in the design of state-of-the-art wireless communication systems. This is because the channel's capacity reduces with the channel's Signal to Noise Ratio (SNR) E_s/N_0, which depends on the transmit power, on the distance to the remote receiver and on the amount of noise that is imposed upon the signal. Hence, wireless communication systems having a particular effective throughput η are associated with a minimum threshold SNR at which the channel's achievable throughput drops below that particular effective throughput η. If a wireless communication system can be designed to support near-capacity operation, then the SNR required to achieve high quality reception is reduced, facilitating lower transmit powers, longer transmission ranges or tolerance to more noise. Note that for a communication scheme having an effective throughput of η, the channel SNR is typically expressed as $E_b/N_0\ [dB] = E_s/N_0\ [dB] - 10 \cdot \log_{10}(\eta)$. In the following chapters, we

will propose some novel schemes employing sophisticated techniques in order to achieve near-capacity operation.

2.8 Soft QPSK Demodulation

From this section onwards, we will introduce the operations of the receiver shown in Figure 2.2(b). After receiving the symbol vector $\tilde{\mathbf{g}}$ through an uncorrelated narrow-band Rayleigh channel, the QPSK demodulator invokes the classic soft demodulation algorithm for processing the soft information about the transmitted bits of $\mathbf{f}$. In Section 2.8.1, we will introduce the LLR used by the soft decoding algorithms. Following this, Section 2.8.2 describes how the soft QPSK demodulator processes the LLRs.

2.8.1 Logarithmic Likelihood Ratio

The LLR [108, 126] is the most common representation of soft information used by the soft demodulator of Section 2.8.2 and by the Log-BCJR algorithm of Section 2.11. An LLR expresses not only what the most likely value of a transmitted bit is, but also how likely that value is. The LLR of a bit is defined as the natural logarithm of the quotient between the probabilities that the bit is equal to '1' or '0', which is formulated as

$$\tilde{g}_j = \ln\left(\frac{P(g_j = 1)}{P(g_j = 0)}\right). \tag{2.4}$$

Here, g_j denotes one bit in the bit vector $\mathbf{g}$ of Figure 2.2 and $\tilde{g}_j$ is the corresponding LLR of this bit, while $P(g_j = 0)$ denotes the probability of $g_j = 0$ and $P(g_j = 1)$ denotes the probability of $g_j = 1$. Naturally, we have $P(g_j = 0) + P(g_j = 1) = 1$. Given the LLR $\tilde{g}_j$, it is possible to calculate $P(g_j = 0)$ or $P(g_j = 1)$ by taking the exponent of both sides in Eq. (2.4), hence we have

$$P(\tilde{g}_j = 0) = \frac{1}{1 + \exp(\tilde{g}_j)}, \tag{2.5}$$

$$P(\tilde{g}_j = 1) = \frac{\exp(\tilde{g}_j)}{1 + \exp(\tilde{g}_j)}. \tag{2.6}$$

Figure 2.15 depicts how $\tilde{g}_j$ varies as the probability $P(g_j = 1)$ varies. It is clear that the sign of the LLR $\tilde{g}_j$ indicates whether the bit g_j is more likely to have the value 0 or 1, where a positive LLR $\tilde{g}_j$ indicates that $g_j = 1$ is more likely. Meanwhile, the magnitude of the LLR gives an indication of how likely it is that the sign of the LLR gives the correct value of g_j. For example, when the LLR is $\tilde{g}_j = 0$, we have $P(g_j = 0) = P(g_j = 1) = 0.5$. This means that the LLR $\tilde{g}_j$ indicates no confidence in the value of the bit g_j and the selection of a bit value g_j will only give the correct result with a probability of 50%. On the other hand, when $\tilde{g}_j \gg 0$, we have $P(g_j = 0) \ll P(g_j = 1)$, which implies that the LLR

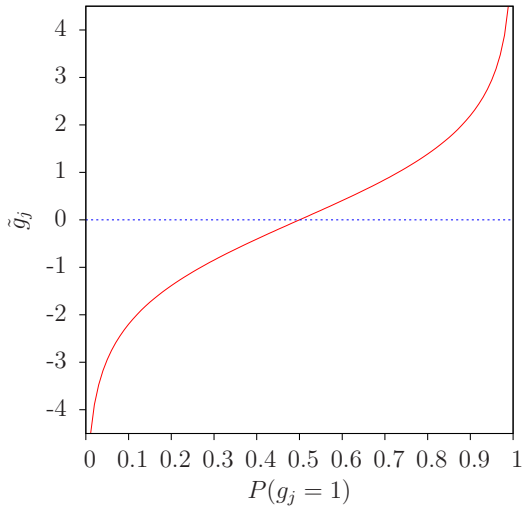

Figure 2.15: The LLR $\tilde{g}_j$ versus the probability $P(g_j = 1)$.

expresses confidence that we have $g_j = 1$. By contrast, when $\tilde{g}_j \ll 0$, we can be confident that $g_j = 0$ is true with a high probability.

2.8.2 Soft Demodulator

Having introduced the LLR, this section discusses how the soft QPSK demodulator works. After receiving the symbol vector $\tilde{\mathbf{g}}$ through the uncorrelated Rayleigh channel, we can derive the LLR corresponding to each bit in the transmitted bit vector $\mathbf{f}$ [32,127], as follows

$$\tilde{f}_{2j-1} = -\frac{4|h_j|^2}{N_0} \cdot \mathrm{Im}\left(\frac{\tilde{g}_j}{h_j}\right), \tag{2.7}$$

$$\tilde{f}_{2j} = -\frac{4|h_j|^2}{N_0} \cdot \mathrm{Re}\left(\frac{\tilde{g}_j}{h_j}\right), \tag{2.8}$$

where $\tilde{g}_j$ is the j-th symbol in the received symbol vector $\tilde{\mathbf{g}}$, h_j is the corresponding channel gain from the vector $\mathbf{h}$, N_0 is the power spectral density of the AWGN, $\tilde{f}_{2j-1}$ is the LLR for the first bit in symbol $\tilde{g}_j$ and $\tilde{f}_{2j}$ is the LLR for the second bit, as well as $Re(\cdot)$ and $Im(\cdot)$ are the real part and the imaginary part of $(\cdot)$, respectively. Here, it is assumed that the receiver has perfect knowledge of the channel gain vector $\mathbf{h}$ and of the power spectral density N_0.

Again, we consider the second symbol $\tilde{g}_2 = -0.7582 - 0.0339\mathrm{i}$ of the received symbol vector $\tilde{\mathbf{g}}$ as an example. According to Eq. (2.7) and Eq. (2.8), we have the LLR values of $f_3 = -2.2034$ and $f_4 = -2.0884$. These LLR values may be combined with those obtained from the other four received symbols of $\tilde{\mathbf{g}}$, in order to obtain the vector $\tilde{\mathbf{f}}$ of $J = 10$ LLRs, which may be passed to the URC decoder as the soft input, as it will be discussed in Section 2.11. Note that if a hard decision was made based on Figure 2.15

using the values of $\tilde{f}_{2j-1}$ and $\tilde{f}_{2j}$, this would suggest that the correct symbol is the 00 point in the constellation diagram, which indeed matches with the transmitted symbol g_2.

2.9 Deinterleaver Operation

As shown in Figure 2.2(b), the QPSK demodulated LLRs of $\tilde{\mathbf{f}}$ may be deinterleaved by π_2^{-1}, for the sake of rearranging the order of LLRs to produce the vector $\tilde{\mathbf{e}}$ pertaining to the corresponding bits in the vector $\mathbf{e}$. Since the deinterleaving operation in the receiver is the reverse of the interleaver operation in the transmitter, we can simply use the same interleaver vector. More specifically, $\tilde{\mathbf{e}}_{\pi_2(j)} = \tilde{\mathbf{f}}_j$. Similarly, the other deinterleaver π_1^{-1} of Figure 2.2(b) is employed in the same manner. Note that the deinterleaver π_1^{-1} also has another role, which is to mitigate the dependencies between neighboring LLRs in $\tilde{\mathbf{c}^{\mathrm{a}}}$ and $\tilde{\mathbf{a}^{\mathrm{a}}}$ of Figure 2.2(b), as required by the Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm's associated assumption of exploiting independent *a priori* information. These aspects will be described in Section 2.11 and Section 2.13.

For example, let us assume that the extrinsic LLR vector provided by the inner URC decoder is $\tilde{\mathbf{d}}^{\mathrm{e}} = [0.5497, 0.3906, -0.8309, -0.6987, -0.2616, -0.6861, 1.4936, 0.4513, 0.5186, -0.6613]$. This has to be deinterleaved in order to obtain the *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ for the outer URC decoder of Figure 2.2(b). Reusing the interleaver vector $\pi_1 = [10, 2, 4, 7, 1, 6, 5, 9, 3, 8]$ results in the *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}} =$ [-0.6613, 0.3906, -0.6987, 1.4936, 0.5497, -0.6861, -0.2616, 0.5186, -0.8309, 0.4513], where $\tilde{\mathbf{c}}^{\mathrm{a}}_{\pi_1(j)} = \tilde{\mathbf{d}}^{\mathrm{e}}_j$. Both of the *a priori* and the extrinsic LLR vectors are obtained using the BCJR algorithm, as it will be discussed in Section 2.11.

2.10 Demultiplexer Operation

In contrast to the multiplexer of Figure 2.2, a demultiplexer (or DeMUX) is a device taking a single input data stream and selecting one of many data-output-lines, which are connected to each output data stream. A simple demultiplexer schematic is depicted in Figure 2.16, in accordance with the multiplexer in Figure 2.10. At the receiver, the data stream D is split into three streams A, B and C. Note that a demultiplexer at the receiver is often used as a complement of a multiplexer at the transmitter, so they are typically used in pairs. Note that the signal selection operations of multiplexer and demultiplexer must be complementary.

2.11 BCJR Algorithm

In Figure 2.2(b), the iterative operation of the inner and outer URC decoders may be invoked after demodulation and deinterleaving. The URC decoder can be implemented by

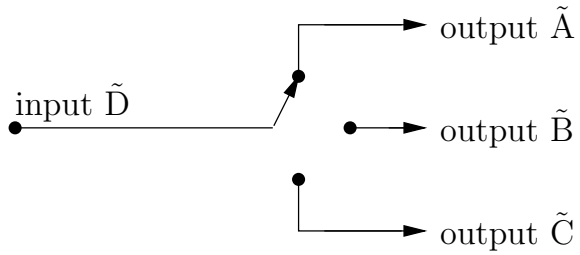

Figure 2.16: An example of demultiplexer schematic, where there is one input data stream D, and three output data streams A, B and C.

the so-called Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm [91], which is an *A Posteriori* Probability (APP) Soft-In Soft-Out (SISO) algorithm. In practice however, the logarithmic versions of the BCJR algorithm, the Log-BCJR algorithm and the Approx-Log-BCJR algorithm [108], have wider employment, as a benefit of their implementation simplicity. In this section, we will briefly describe the Log-BCJR algorithm and Approx-Log-BCJR algorithm. We commence in Section 2.11.1 by introducing the fundamental Add, Compare and Select (ACS) operations that form the basis of these algorithms. Following this, Section 2.11.2 describes the steps that are employed in the Log-BCJR and Approx-Log-BCJR algorithms.

2.11.1 ACS Operations

The original BCJR algorithm relies on a pair of basic operations, namely the addition and the multiplication of bit probabilities. However, these multiplication operations lead to complex hardware. Furthermore, the bit probabilities may have a very high dynamic range, requiring the employment of complex floating-point processing. The Log-BCJR algorithm avoids these problems by transforming the algorithm into the logarithmic domain, where multiplications become additions. For example, assuming that $A = \ln(a)$ and $B = \ln(b)$, we have

$$\ln(a \cdot b) = \ln(e^A \cdot e^B) = A + B. \tag{2.9}$$

In this way, the multiplication $(a \cdot b)$ in the classic linear domain becomes an addition in the logarithmic domain of the Log-BCJR algorithm. Furthermore, additions in the linear domain may be carried out by invoking the Jacobian logarithm in the logarithmic domain, which we denote using the $\max^*$ operator according to [128]

$$\ln(a + b) = \ln(e^A + e^B) = \max(A, B) + \ln(1 + e^{-|A-B|}) = \max^*(A, B). \tag{2.10}$$

Note that the $\max^*$ operation may be computed using successive pairwise operations, when it has to process more than two operands, where we have $\max^*(A, B, C) = \max^*(\max^*(A, B), C)$, for example. In the Approx-Log-BCJR algorithm, the function $f_c(|A - B|) =$

$\ln(1 + e^{-|A-B|})$ can be implemented using a Look-Up-Table (LUT), which compares $|A - B|$ to the elements in the LUT, in order to select the best approximation available in the LUT for $f_c(|A - B|)$. Likewise, the $\max(A, B)$ operation can be implemented by comparing A and B, and then selecting the larger one. Hence, all operations required in the Approx-Log-BCJR algorithm are "Addition", "Comparison" and "Selection", which are the so-called ACS operations. In the following chapters, we will employ the number of ACS operations as a metric to quantify the complexity of the decoders in different schemes, in order to make fair comparisons between them.

2.11.2 γ, α, β and δ Calculations

In order to present the Log-BCJR algorithm step by step, we consider the outer URC BCJR decoder of Figure 2.2. Here, the Log-BCJR algorithm imposes the URC^1 code constraints on the input *a priori* LLR vectors $\tilde{\mathbf{a}}^{\mathrm{a}}$ and $\tilde{\mathbf{b}}^{\mathrm{a}}$, outputting the improved extrinsic LLR vectors $\tilde{\mathbf{a}}^{\mathrm{e}}$ and $\tilde{\mathbf{b}}^{\mathrm{e}}$.

The calculation of the extrinsic LLR vectors $\tilde{\mathbf{a}}^{\mathrm{e}}$ and $\tilde{\mathbf{b}}^{\mathrm{e}}$ may be completed following the calculation of four sets of internal variables, namely $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$. For the sake of consistency, we continue using the notions in the transition diagram of Figure 2.8 and those in the trellis diagram of Figure 2.9. Next, we will demonstrate how to calculate these four sets of internal variables. Meanwhile, a detailed example is illustrated in Figure 2.17 to augment our discussions.

- $\boldsymbol{\gamma}$ calculation

  The $\boldsymbol{\gamma} = [\gamma_j(m, m')]$ values provide the *a priori* probability for each transition in the trellis, where $m, m' \in \mathbf{M}$ and $1 \leq j \leq J$. In the example of Figure 2.17, $\mathbf{M} \in \{1, 2\}$ is the set of all the legitimate states of the trellis, and $J = 5$ is the length of the bit vectors $\mathbf{a}$ and $\mathbf{b}$. For the j-th bit, each $\gamma_j(m, m')$ value denotes the transition probability that the decoder traverses from the previous state m' to the next state m. Clearly, there is a one-to-one mapping between the $\boldsymbol{\gamma} = [\gamma_j(m, m')]$ values and the trellis transitions, as shown by the labelling of each transition in Figure 2.17(e).

  As observed in Figure 2.17(e), the values of $\boldsymbol{\gamma}$ depend on the input *a priori* LLR vectors $\tilde{\mathbf{a}}^{\mathrm{a}}$ and $\tilde{\mathbf{b}}^{\mathrm{a}}$ that are shown in Figure 2.17(a). More particularly, for the given states m and m', $\gamma_j(m, m')$ can be calculated by

$$\gamma_j(m, m') = \tilde{a}_j^{\mathrm{a}} \cdot a(m, m') + \tilde{b}_j^{\mathrm{a}} \cdot b(m, m') + \ln[P(m|m')], \tag{2.11}$$

  where $P(m|m')$ is the probability that trellis transitions from the previous state m' to the next state m in the encoder, while $a(m, m')$ and $b(m, m')$ are the corresponding bit values for a and b, when the trellis transitions from state m' to state m, as

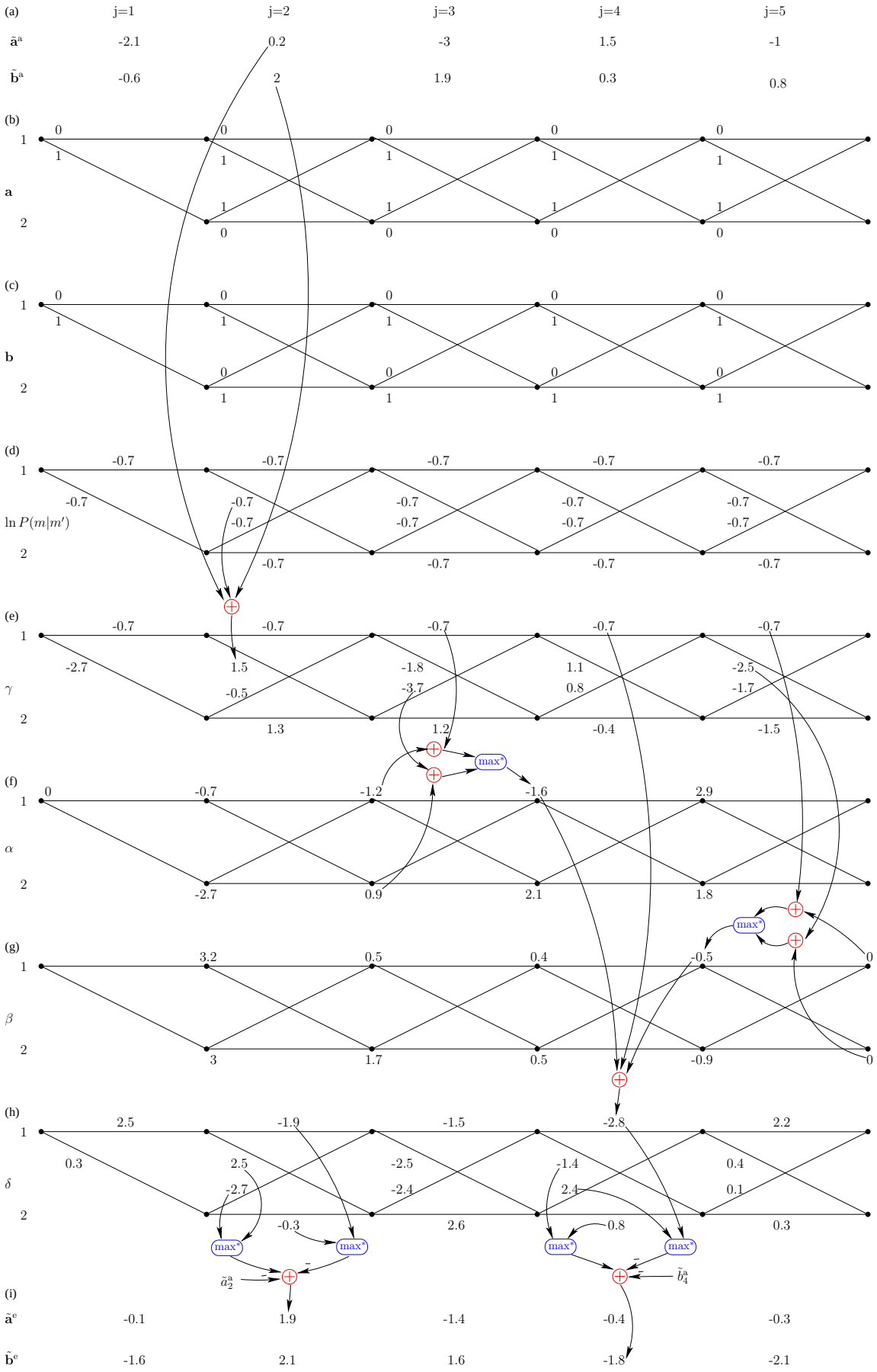

Figure 2.17: An example of applying the Log-BCJR algorithm to the outer URC decoder of Figure 2.2, where the inputs are the *a priori* LLR vectors $\tilde{\mathbf{a}}^a$ and $\tilde{\mathbf{b}}^a$, while the outputs are the extrinsic LLR vectors $\tilde{\mathbf{a}}^e$ and $\tilde{\mathbf{b}}^e$.

shown in Figure 2.17(b) and Figure 2.17(c), respectively. Since the URC^{1} code of Figure 2.6 is employed in the schematic of Figure 2.2, we have $P(m|m') = 0.5$ for all transitions, as shown in Figure 2.17(d), hence $\ln(P(m|m') \approx -0.7$. For example, as demonstrated in Figure 2.17(e), we have $\gamma_2(2,1) = \tilde{a}_2^{\mathrm{a}} \cdot a(2,1) + \tilde{b}_2^{\mathrm{a}} \cdot b(2,1) + \ln P(2|1) = 0.2 + 2 - 0.7 = 1.5$.

- $\boldsymbol{\alpha}$ calculation

  The $\boldsymbol{\alpha} = [\alpha_j(m)]$ values correspond to the forward transition probabilities, where $m \in \mathbf{M}$ and $0 \leq j \leq J-1$. There is a one-to-one mapping between the $\boldsymbol{\alpha} = [\alpha_j(m)]$ values and the states of the trellis, as shown by the labelling of the states in Figure 2.17(f). Each $\alpha_j(m)$ value depends on $\alpha_{j-1}(m')$ for all previous states $m' \in \mathbf{M}$ and all $\gamma_j(m,m')$ for the current transitions where $m' \in \mathbf{M}$. Hence, it requires forward recursions in the trellis to obtain all the $\alpha_j(m)$ values. More particularly, given the state m, $\alpha_j(m)$ can be obtained by

$$\alpha_j(m) = \max_{m' \in \mathbf{M}}{}^{*}[\alpha_{j-1}(m') + \gamma_j(m,m')]. \tag{2.12}$$

  For $j = 0$, we have the boundary condition $\alpha_0(1) = 0$, since the encoding is always initialised from the state $m_0 = 1$, as described in Section 2.3.3.

  For example, as illustrated in Figure 2.17(f), we have $\alpha_4(1) = \max^{*}[\alpha_3(1)+\gamma_4(1,1), \alpha_3(2) + \gamma_4(1,2)] = \max^{*}(-1.2 - 0.7, 0.9 - 3.7) = -1.6$. For simplicity, the result of -1.6 has been rounded to a single decimal place.

- $\boldsymbol{\beta}$ calculation

  The $\boldsymbol{\beta} = [\beta_j(m')]$ values correspond to the backward transition probabilities, where $m' \in \mathbf{M}$ and $1 \leq j \leq J$. There is a one-to-one mapping between the $\boldsymbol{\beta} = [\beta_j(m')]$ values and the states of the trellis, as shown by the labelling of the states in Figure 2.17(g). Each $\beta_j(m')$ value depends on $\beta_{j+1}(m)$ for all next states $m \in \mathbf{M}$ and all $\gamma_j(m',m)$ for the current transitions, where we have $m \in \mathbf{M}$. Hence, it requires backward recursions in the trellis to obtain all the $\beta_j(m')$ values. More particularly, given the state m', $\beta_j(m')$ can be obtained by

$$\beta_j(m') = \max_{m \in \mathbf{M}}{}^{*}[\beta_{j+1}(m) + \gamma_j(m',m)]. \tag{2.13}$$

  For $j = J$, we have the boundary conditions of $\beta_J(m) = 0$ for all $m \in \mathbf{M}$.

  For example, as illustrated in Figure 2.17(g), we have $\beta_4(1) = \max^{*}[\beta_5(1) + \gamma_4(1,1), \beta_5(2) + \gamma_4(2,1)] = \max^{*}(0 - 0.7, 0 - 1.5) = -0.5$.

- $\boldsymbol{\delta}$ calculation

  In the fourth step, the three sets of the intermediate variables, $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, can be used for calculating the *a posteriori* probability that the encoder traversed through each

transition in the trellis. More particularly, we denote the probability of transition from state m' to state m by $\delta_j(m, m')$, and all the transitions in the trellis by $\boldsymbol{\delta} = [\delta_j(m, m')]$, where $m, m' \in \mathbf{M}$ and $1 \leq j \leq J$. More particularly, each $\delta_j(m, m')$ value can be calculated by

$$\delta_j(m, m') = \alpha_j(m') + \gamma_j(m, m') + \beta_j(m). \tag{2.14}$$

For example, as demonstrated in Figure 2.17(h), we have $\delta_4(1,1) = \alpha_4(1)+\gamma_4(1,1)+\beta_4(1) = -2.1 - 0.7 - 0.7 = -3.5$.

Finally, the extrinsic LLRs of $\tilde{\mathbf{a}}^{\mathrm{e}}$ and $\tilde{\mathbf{b}}^{\mathrm{e}}$ can be obtained based on the $\boldsymbol{\delta}$ values. More particularly, for the values $\tilde{a}_j^{\mathrm{e}}$ and $\tilde{b}_j^{\mathrm{e}}$, we have

$$\tilde{a}_j^{\mathrm{e}} = \max^*_{m,m'|a(m,m')=1}[\delta_j(m, m')] - \max^*_{m,m'|a(m,m')=0}[\delta_j(m, m')] - \tilde{a}_j^{\mathrm{a}}. \tag{2.15}$$

$$\tilde{b}_j^{\mathrm{e}} = \max^*_{m,m'|b(m,m')=1}[\delta_j(m, m')] - \max^*_{m,m'|a(m,m')=0}[\delta_j(m, m')] - \tilde{b}_j^{\mathrm{a}}. \tag{2.16}$$

For example, as demonstrated in Figure 2.17(i), we have $\tilde{a}_2^{\mathrm{e}} = \max^*[\delta_2(2,1), \delta_2(2,1)] - \max^*[\delta_2(1,1), \delta_2(2,2)] - \tilde{a}_2^{\mathrm{a}} = \max^*(2.5, -2.7) - \max^*(-1.9, 0.3) - 0.2 = 1.9$, and $\tilde{b}_4^{\mathrm{e}} = \max^*[\delta_4(2,1), \delta_4(2,2)] - \max^*[\delta_4(1,1), \delta_4(1,2)] - \tilde{b}_4^{\mathrm{a}} = \max^*(-1.4, 0.8) - \max^*(-2.8, 2.4) - 0.3 = -1.8$.

This step completes the Log-BCJR algorithm. Throughout this treatise, we will invoke the Log-BCJR algorithm for soft decoding.

2.12 Iterative Decoding

In this section, we discuss the employment of the BCJR algorithm for iterative decoding. The so-called iterative decoding method is widely used in the concatenated systems of Section 2.2. As seen in the schematic of Figure 2.2(b), the extrinsic LLR vectors $\tilde{\mathbf{c}}^{\mathrm{e}}$ and $\tilde{\mathbf{d}}^{\mathrm{e}}$ are exchanged iteratively between the inner and outer URC decoders. For the first iteration, the inner URC decoder may be activated, given the inputs LLR vectors $\tilde{\mathbf{e}}$ and $\tilde{\mathbf{d}}^{\mathrm{a}}$. Note that the LLR vector $\tilde{\mathbf{e}}$ is the soft information generated from the demodulator, while the *a priori* LLR vector $\tilde{\mathbf{d}}^{\mathrm{a}}$ may be initialized to be an all-0 vector during the first iteration. After running the BCJR algorithm, the inner URC decoder outputs the extrinsic LLR vector $\tilde{\mathbf{d}}^{\mathrm{e}}$, which is deinterleaved by π_1^{-1} of Figure 2.2 to obtain the *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ as the input of the outer URC decoder. Similarly, the output extrinsic LLR vector $\tilde{\mathbf{c}}^{\mathrm{e}}$ of the outer URC decoder in Figure 2.2 is interleaved by π_1 to obtain the *a priori* LLR vector $\tilde{\mathbf{d}}^{\mathrm{a}}$ used as the input for the second iteration of the inner URC decoder. With the aid of this improved *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$, the inner decoder generates an improved extrinsic

LLR vector $\tilde{\mathbf{c}}^{\mathrm{e}}$. This process may continue for a certain amount number of iterations. In this way, gradually increasingly reliable *a priori* and extrinsic information is fed back to both decoders of Figure 2.2, in order to successively improve the decoding performance of the whole scheme.

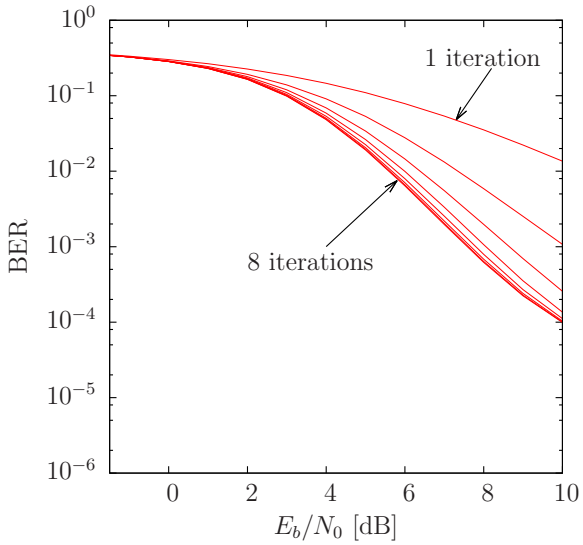

Figure 2.18: The BER performance that is obtained during the achievement of iterative decoding convergence in the serially concatenated scheme of Figure 2.2, when the bit vector $\mathbf{a}$ has a length of $J = 50$ and is transmitted over an uncorrelated narrow-band Rayleigh fading channel.

Figures 2.18, 2.19 and 2.20 characterize the BER performance of the scheme shown in Figure 2.2, when employing iterative decoding for frames having different lengths of $J \in \{50, 500, 5000\}$. It can be seen that an improved BER performance is achieved, when more decoding iterations are employed. Nevertheless, there is limit for how much the BER performance can be improved, whereupon no increased-confidence information can be obtained, even if more decoding iterations are carried out. This can be explained using the EXIT chart, as it will be discussed in Section 2.13.

Note that a superior BER performance is achieved, upon using longer frame lengths J. Observe in Figure 2.18 that for the case of $J = 50$, a BER below 10^{-2} is only achieved after $I = 8$ decoding iterations, when the channel SNR is around 6 dB. By contrast, as shown in Figure 2.19 and Figure 2.20 for the cases of $J = 500$ and $J = 5000$, respectively, a BER below 10^{-2} can be obtained when the SNR is around 4 dB, after $I = 8$ decoding iterations. Moreover, Figure 2.20 shows that when a relatively long bit vector length of $J = 5000$ is employed, the BER rapidly reduces as the channel SNR increases towards 5 dB. This is the so-called 'waterfall' or 'cliff' region. It becomes clear from these three BER plots that the turbo cliff is much steeper, when the bit vector length is higher.

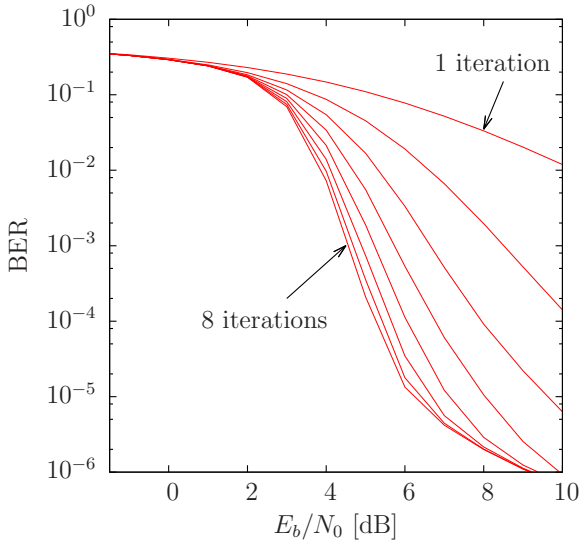

Figure 2.19: The BER performance that is obtained during the achievement of iterative decoding convergence in the serially concatenated scheme of Figure 2.2, when the bit vector $\mathbf{a}$ has a length of $J = 500$ and is transmitted over an uncorrelated narrow-band Rayleigh fading channel.

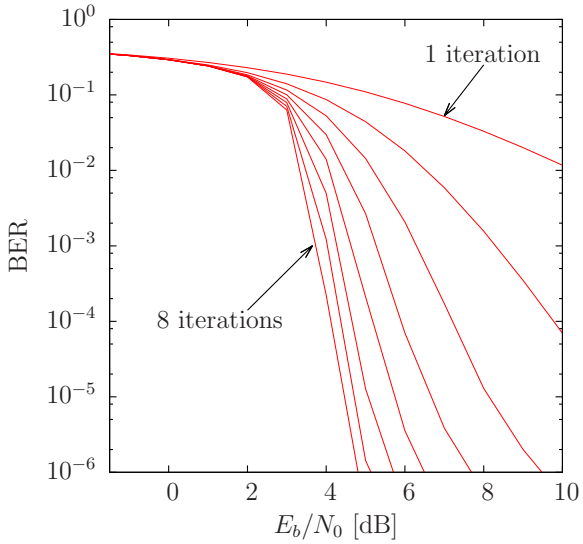

Figure 2.20: The BER performance that is obtained during the achievement of iterative decoding convergence in the serially concatenated scheme of Figure 2.2, when the bit vector $\mathbf{a}$ has a length of $J = 5000$ and is transmitted over an uncorrelated narrow-band Rayleigh fading channel.

2.13 EXIT Chart

As discussed in the previous section, the BER plot is indeed capable of characterizing the decoding performance of iterative decoding schemes. However, it does not explicitly characterise their convergence behaviours. This motivates the conception of a more sophisticated analysis tool, namely the EXIT chart [99], which may be employed to predict the convergence behaviour of an iteratively decoded scheme. In particular, an EXIT chart allows the outer and inner decoders of Figure 2.2(b) to be characterised independently of each other, before also cosidering their iterative decoding interaction. For example, the schematic depicted in Figure 2.21 may be used for generating the EXIT curve for the outer URC code of Figure 2.2. In this section, we will briefly introduce how to generate an EXIT chart and how to use it for analysing the decoding convergence behaviour.

The Mutual Information (MI) [129] is employed by the EXIT chart to quantify the reliability of the soft information contained in the iteratively exchanged LLR vectors. As described in [99], the MI between an LLR vector and the corresponding vector of bit values depends on the distribution of the LLR values. Based on the discussions in Section 2.8.1, if the distribution of the LLR values that corresponds to 0-bits is equal to that of the LLR values pertaining to 1-bits, then the MI will be zero. This implies that the LLR values are totally unreliable and the selection of a bit's value based upon the sign of the corresponding LLR will only give the correct answer with a probability of 50%. By contrast, as the reliability of the LLRs increases, the two LLR distributions will move apart and will only partially overlap, giving a MI that is higher than zero. When the distributions become completely separated, the resultant MI becomes equal to one, in which case the correct bit values will always be obtained by making decisions based upon the sign of the corresponding LLRs. In this treatise, the MI is denoted by $I(\cdot,\cdot)$. For example, $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c})$ in Figure 2.21 denotes the MI between the LLRs in the vector $\tilde{\mathbf{c}}^{\mathrm{e}}$ and the corresponding bits in the vector $\mathbf{c}$, where we have $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c}) \in [0, 1]$.

The schematic shown in Figure 2.21 may be used for generating the EXIT curve for the outer UEC decoder of Figure 2.2(b). In order to calculate the extrinsic MI $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c})$, we provide the URC BCJR decoder of Figure 2.2(b) with an *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ having a particular MI $I(\tilde{\mathbf{c}}^{\mathrm{a}}, \mathbf{c})$. This can be realized by considering the bit values in the vector $\mathbf{c}$ and generating a corresponding synthetic vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ of uncorrelated Gaussian distributed *a priori* LLRs [99], as illustrated in Figure 2.21. This synthetic LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ may then be input to the URC decoder and the MI $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c})$ of the resultant extrinsic LLR vector $\tilde{\mathbf{c}}^{\mathrm{e}}$ can be recorded. In our work, we employ the averaging method of [130] to calculate the MI, which is capable of calculating the MI of an LLR vector without having to consider the ground truth contained in the corresponding bit vector. However, this assumes that

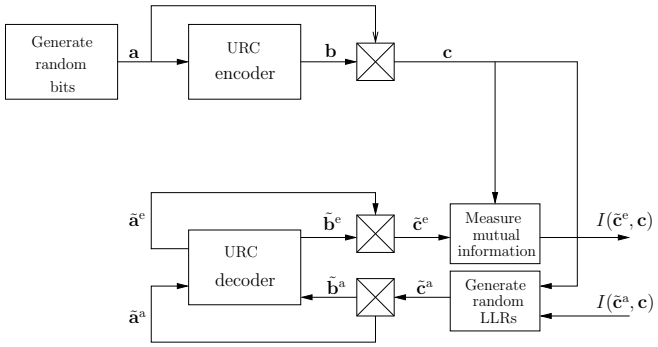

Figure 2.21: Schematic for generating the inverted EXIT curve for the outer URC BCJR decoder of Figure 2.2(b). Here, the *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ is generated using a particular MI $I(\tilde{\mathbf{c}}^{\mathrm{a}}, \mathbf{c})$ that is plotted on the vertical axis of the EXIT chart, while the extrinsic LLR vector $\tilde{\mathbf{c}}^{\mathrm{e}}$ is used to calculate the MI $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c})$ that is plotted on the horizontal axis.

the extrinsic LLR vector is generated by an optimal APP decoder, such as the Log-BCJR algorithm of Section 2.11. By contrast, the other method that may be used for calculating the MI is the histogram method [99], which uses knowledge of the bit values in the vector to avoid having to "believe" what the LLRs indicate. This produces a more reliable measurement of the MI, when the decoder is sub-optimal, such as when using the Approx-Log-BCJR algorithm of Section 2.11.

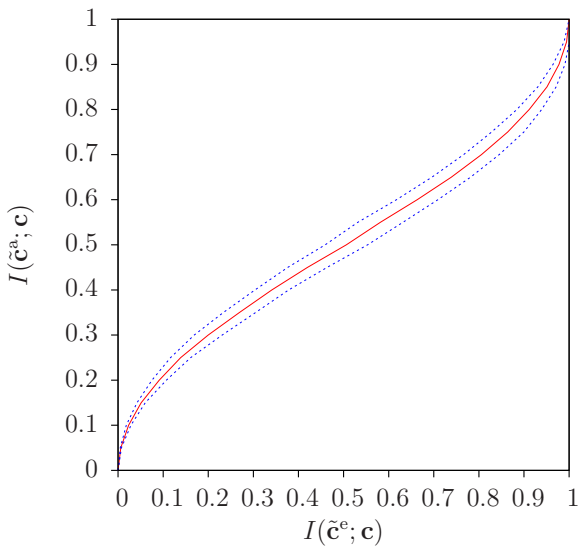

Figure 2.22: The solid line denotes the inverted EXIT curve of the outer URC decoder that is generated by the schematic of Figure 2.21. The dashed lines denote the average inverted EXIT curve measurement plus and minus its standard deviation. Here, the bit vector $\mathbf{c}$ has a length of $2J = 100$.

For example, the inverted EXIT curve of the outer URC decoder of Figure 2.2(b) is demonstrated in Figure 2.22, where the MI $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c})$ of the extrinsic LLR vector $\tilde{\mathbf{c}}^{\mathrm{e}}$ is plotted on the horizontal axis and the MI $I(\tilde{\mathbf{c}}^{\mathrm{a}}, \mathbf{c})$ of the *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ is plotted on the

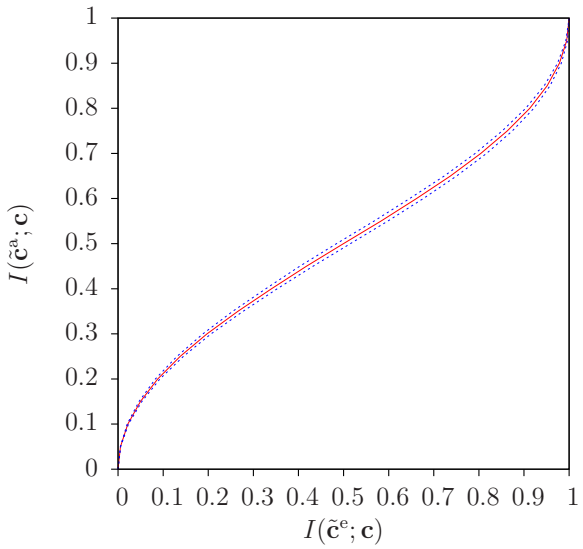

Figure 2.23: The solid line denotes the inverted EXIT curve of the outer URC decoder that is generated by the schematic of Figure 2.21. The dashed lines denote the average inverted EXIT curve measurement plus and minus its standard deviation. Here, the bit vector $\mathbf{c}$ has a length of $2J = 1000$.

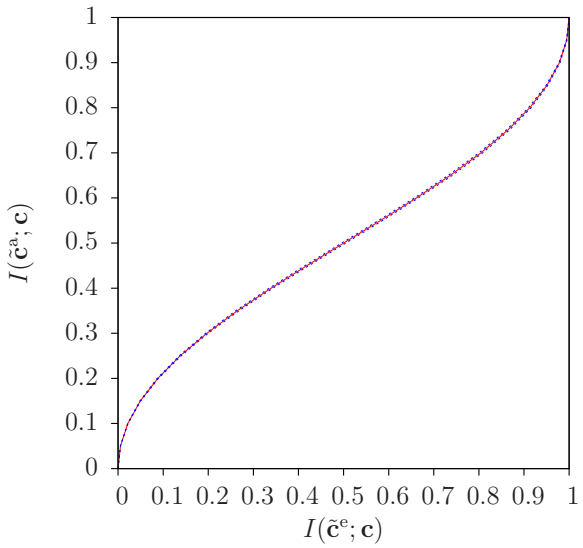

Figure 2.24: The solid line denotes the inverted EXIT curve of the outer URC decoder that is generated by the schematic of Figure 2.21. The dashed lines denote the average inverted EXIT curve measurement plus and minus its standard deviation. Here, the bit vector $\mathbf{c}$ has a length of $2J = 10000$.

vertical axis. More particularly, a specific point in the inverted EXIT curve of Figure 2.22 quantifies the MI $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c}) \in [0, 1]$ between the bit vector $\mathbf{c}$ and the extrinsic LLR vector $\tilde{\mathbf{c}}^{\mathrm{e}}$ that is output by the outer URC decoder, when it is provided with the *a priori* LLR vector $\tilde{\mathbf{c}}^{\mathrm{a}}$ having a particular MI $I(\tilde{\mathbf{c}}^{\mathrm{a}}, \mathbf{c}) \in [0, 1]$. In Figure 2.22, it is clearly seen that $I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c})$ increases from 0 to 1, when provided with increasing $I(\tilde{\mathbf{c}}^{\mathrm{a}}, \mathbf{c})$ in the range 0 to 1, which means that the output of the outer URC decoder becomes more and more reliable with increasingly reliable input. The EXIT chart finally reaches the $(1, 1)$ point at the top right corner, which is where iterative decoding convergence to a low BER is facilitated. This is because MIs of $I(\tilde{\mathbf{c}}^{\mathrm{a}}, \mathbf{c}) = I(\tilde{\mathbf{c}}^{\mathrm{e}}, \mathbf{c}) = 1$ implies perfect knowledge of the bits in the vector $\mathbf{c}$.

Furthermore, two more EXIT curves are provided in Figures 2.23 and 2.24, where the bit vector $\mathbf{c}$ has a length of 1000 and 10000, respectively. Clearly, the MI measurements of Figure 2.24 exhibit a lower standard deviation, compared to those of Figures 2.22 and 2.23. This is because the average values of large sets of random variables exhibit a lower standard deviation than those of small sets. This reduced variation from frame-to-frame when employing long vectors of bits explains the steep turbo 'waterfall' region shown in Figure 2.20. More explicitly, with low variation from frame-to-frame, either all frames will decoded with a low BER, if the E_b/N_0 is sufficiently high, or none of them will be decoded with a low BER, if the E_b/N_0 is low. By contrast, the slower waterfalls of Figures 2.18 and 2.19 are explained by the greater variation from frame-to-frame that is exhibited when using shorter bit vector $\mathbf{c}$. More explicitly, at a particular E_b/N_0, some frames may be decoded with a low BER, while some others may not be, where the ratio between these outcomes depends on how high the E_b/N_0 is.

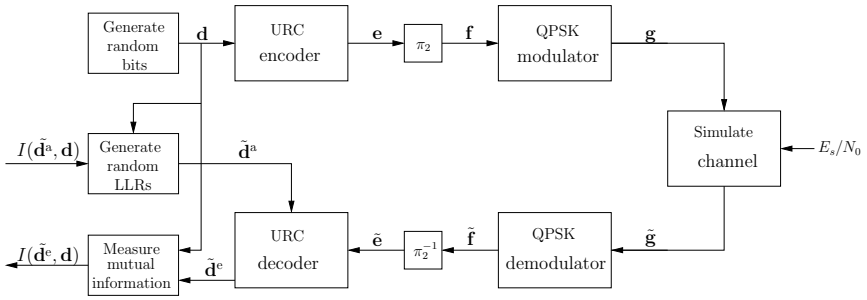

Figure 2.25: Schematic for generating the EXIT curve for the inner URC BCJR decoder of Figure 2.2. Here, the *a priori* LLR vector $\tilde{\mathbf{d}}^{\mathrm{a}}$ is generated using a particular MI $I(\tilde{\mathbf{d}}^{\mathrm{a}}, \mathbf{d})$ that is plotted on the horizontal axis of the EXIT chart, and the extrinsic LLR vector $\tilde{\mathbf{d}}^{\mathrm{e}}$ is used to calculate the MI $I(\tilde{\mathbf{d}}^{\mathrm{e}}, \mathbf{d})$ that is plotted on the vertical axis.

Similarly, the schematic used for generating the EXIT curve of the inner URC decoder is shown in Figure 2.25. In addition to the *a priori* LLR vector $\tilde{\mathbf{d}}^{\mathrm{a}}$ that is generated to have a particular MI $I(\tilde{\mathbf{d}}^{\mathrm{a}}, \mathbf{d})$, the inner URC decoder also has an extra input LLR vector $\tilde{\mathbf{e}}$ that

is received from the soft QPSK demodulator, which has a quality that is dictated by the symbol SNR E_s/N_0 at the output of the channel. Owing to this, the MI of the extrinsic LLR vector $\tilde{\mathbf{d}}^{\mathrm{e}}$ produced by the inner URC decoder of Figure 2.2(b) depends on both the LLR vectors $\tilde{\mathbf{d}}^{\mathrm{a}}$ and $\tilde{\mathbf{e}}$. In Figure 2.26, the EXIT function of the inner URC decoder is depicted for different SNRs, while the dashed lines denote the average plus or minus one standard deviation. As for the inverted outer EXIT curves of Figure 2.22, the standard deviation becomes wider, when employing shorter bit vectors, as shown in Figure 2.26 for the case where $\mathbf{d}$ has a length of $2J = 100$. The standard deviation reduces for longer bit vectors $\mathbf{d}$, as demonstrated in Figures 2.27 and 2.28. Note that when the SNR E_s/N_0 increases, the EXIT curve of the inner URC decoder moves upwards, which is associated with producing higher-quality extrinsic LLR vectors. This may be explained by the area properties of the EXIT chart, as it will be further discussed below.

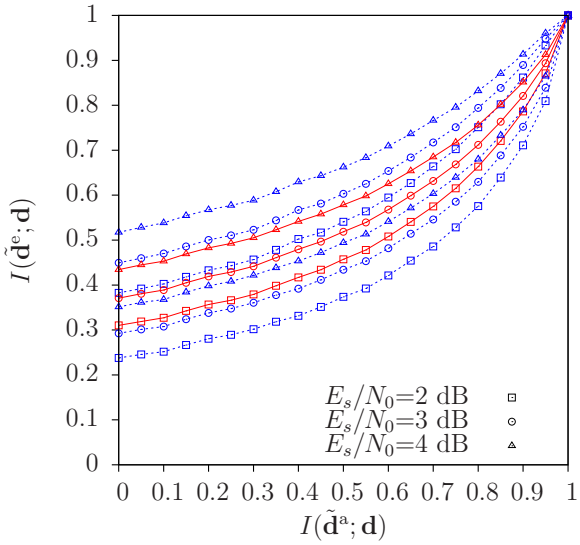

Figure 2.26: The solid lines denote the EXIT curves of the inner URC decoder that is generated by the schematic of Figure 2.25 for the case where $E_s/N_0 = 2, 3, 4$ dB. The dashed lines denote the average EXIT curve measurements plus and minus their standard deviations. Here, the bit vector $\mathbf{d}$ has a length of $2J = 100$.

As alluded to above, EXIT charts can be employed for predicting the iterative decoding convergence behaviour of the serially concatenated scheme shown in Figure 2.2. As described in Section 2.12, the two URC decoders of Figure 2.2(b) may be operated iteratively one after another, sequentially providing each other with increasingly reliable vectors of LLRs. This process may be characterised by plotting the so-called Monte-Carlo simulation-based iterative decoding trajectory, which advances up the tunnel created between the EXIT curves of the two URC decoders in a staircase fashion, until the pair of EXIT curves intersect, as depicted in Figures 2.29 and 2.30. Observe in Figure 2.29 that when the channel SNR is $E_s/N_0 = 4$ dB, the EXIT charts do not intersect before reaching the $(1, 1)$ point, hence creating an open EXIT chart tunnel [131] and therefore

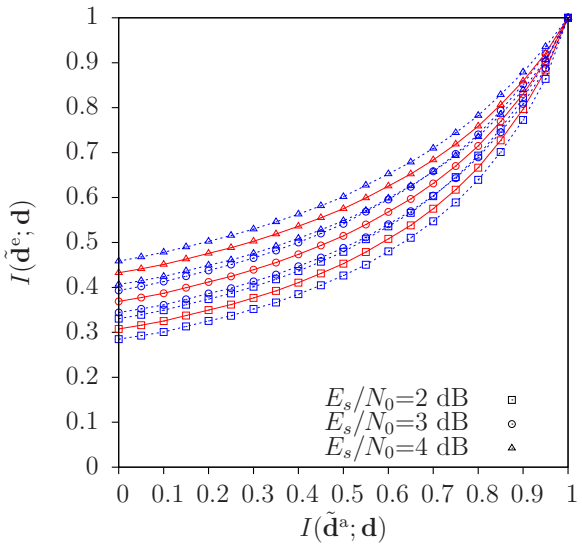

Figure 2.27: The solid lines denote the EXIT curves of the inner URC decoder that is generated by the schematic of Figure 2.25 for the case where $E_s/N_0 = 2, 3, 4$ dB. The dashed lines denote the average EXIT curve measurements plus and minus their standard deviations. Here, the bit vector $\mathbf{d}$ has a length of $2J = 1000$.

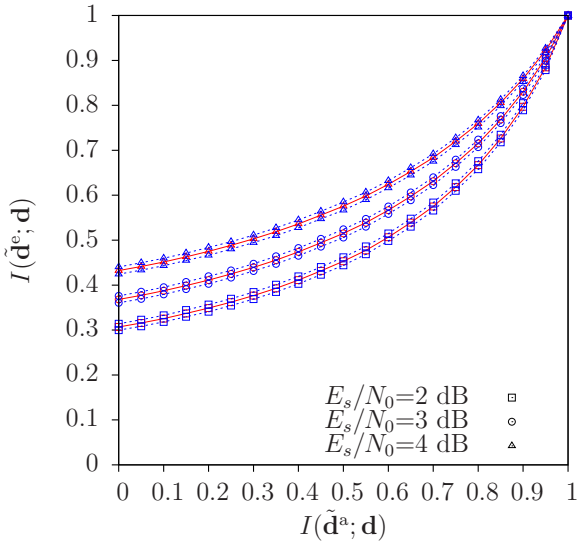

Figure 2.28: The solid lines denote the EXIT curves of the inner URC decoder that is generated by the schematic of Figure 2.25 for the case where $E_s/N_0 = 2, 3, 4$ dB. The dashed lines denote the average EXIT curve measurements plus and minus their standard deviations. Here, the bit vector $\mathbf{d}$ has a length of $2J = 10000$.

potentially enabling absolutely reliable soft information to be obtained. This indicates that iterative decoding convergence to an infinitesimally low BER may be obtained at $E_b/N_0 = E_s/N_0 = 4$ dB, which is in accordance with the BER curves of Figure 2.20.

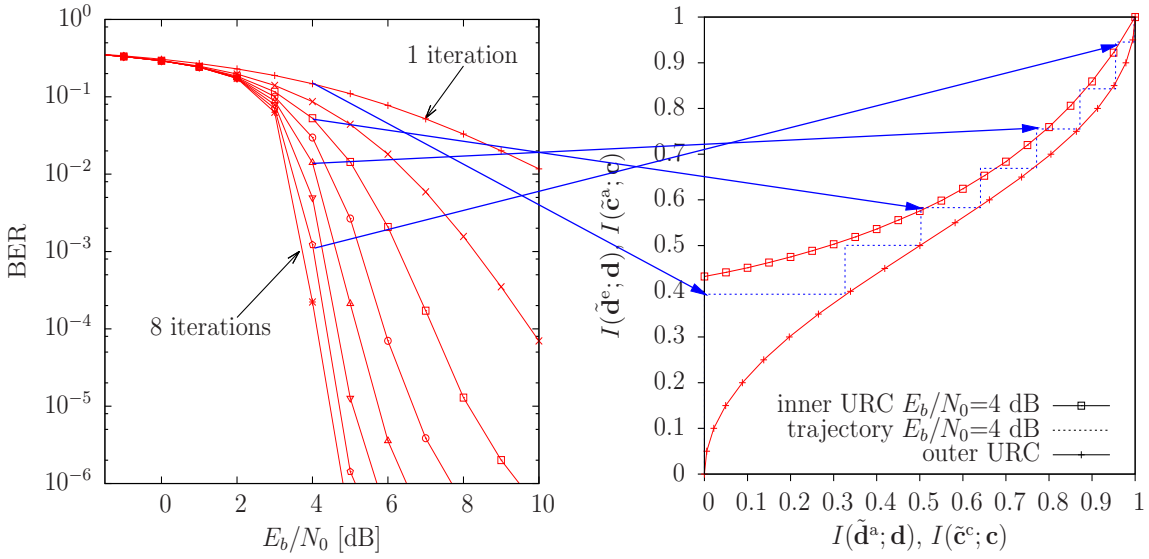

Figure 2.29: EXIT charts and iterative decoding trajectories for the serially concatenated URC decoders of Figure 2.2. Here the scheme is employed for the transmission over a QPSK-modulated Rayleigh fading channel having $E_b/N_0 = 4$ dB. The bit vectors $\mathbf{c}$ and $\mathbf{d}$ have a length of $2J = 10000$.

However, if the channel SNR is reduced to $E_s/N_0 = 2$ dB, the inner UEC decoder's EXIT curve of Figure 2.30 moves downwards to intersect with the inverted outer URC EXIT curve, before reaching the $(1, 1)$ point. In this case, the tunnel is closed, hence preventing the iterative decoding trajectory from approaching the $(1, 1)$ point and therefore resulting in a relatively high BER that is commensurate with the MI achieved. This can also be seen in the BER results of Figure 2.20, where the BER is above 10^{-1}, when we have $E_b/N_0 = E_s/N_0 = 2$ dB. Note that the number of steps in the iterative decoding trajectory indicates the number of decoding iterations that are required to achieve iterative decoding convergence. Also note that the variation between the different trajectories shown in Figures 2.29 and 2.30 may be explained by the standard deviations discussed above, hence resulting in a reduced variation for longer bit vectors.

As mentioned above, the EXIT curve of the inner URC decoder moves upwards, as the channel SNR increases, which may be explained by the area properties of the EXIT charts. Here, we denote the area beneath the inverted outer EXIT curve and the area beneath the inner EXIT curve by A_{outer} and A_{inner}, respectively. In [131, 132], the area A_{outer} of an optimal outer APP SISO decoder having a coding rate of R_{outer} was shown to be given by

$$A_{\text{outer}} = R_{\text{outer}}. \tag{2.17}$$

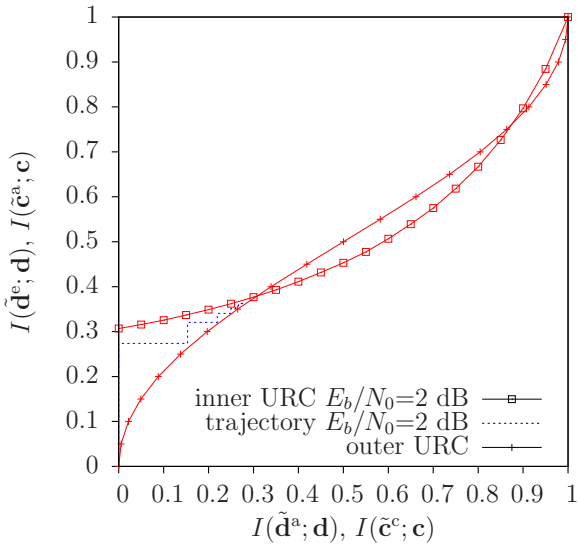

Figure 2.30: EXIT charts and iterative decoding trajectories for the serially concatenated URC decoders of Figure 2.2. Here the scheme is employed for the transmission over a QPSK-modulated Rayleigh fading channel having $E_b/N_0 = 2$ dB. The bit vectors $\mathbf{c}$ and $\mathbf{d}$ have a length of $2J = 10000$.

For example, the outer URC code of Figure 2.2 has a coding rate of $R_{\text{outer}} = 0.5$, which matches the area beneath the inverted EXIT curves of Figures 2.22, 2.23 and 2.24. As described in Section 2.6, the effective throughput η in bits of source information per channel symbol is given by

$$\eta = R_{\text{outer}} \cdot R_{\text{inner}} \cdot \log_2(M) = A_{\text{outer}} \cdot R_{\text{inner}} \cdot \log_2(M). \tag{2.18}$$

Furthermore, [131, 132] also showed that the DCMC capacity C [125] expressed in bits of source information per channel symbol is related to A_{inner} according to

$$C = A_{\text{inner}} \cdot R_{\text{inner}} \cdot \log_2(M), \tag{2.19}$$

when $R_{\text{inner}} = 1$ and when an optimal inner APP SISO decoder is employed. Based on these observations, we have the following three SNR bounds that constitute necessary conditions for iterative decoding convergence to an infinitesimally low probability of error to be supported.

- 1. *Capacity bound*

  As Shannon stated in his seminal publication of 1948 [133], $\eta < C$ constitutes a necessary condition for a reliable communication system. Therefore, the E_b/N_0 or E_s/N_0 SNR value where the capacity C is equal to the effective throughput η provides a lower bound on the SNR, where a low BER can be obtained. We refer to this as the *capacity bound*.

- 2. *Area bound*
  In order to facilitate the creation of an open EXIT chart tunnel, it is necessary, but not sufficient, for the area A_{outer} beneath the inverted outer decoder's EXIT function to exceed the area A_{inner} beneath the inner decoder's EXIT function [132]. Therefore, the *area bound* provides the specific E_b/N_0 or E_s/N_0 SNR value, where we have $A_{\text{outer}} = A_{\text{inner}}$, which would theoretically allow the creation of an open EXIT chart tunnel [134] and the achievement of a low BER, if the outer and inner EXIT functions were shaped to match each other.
- 3. *Tunnel bound*
  Depending on how well the EXIT curve shapes match each other, a narrow but open EXIT chart tunnel can only be created at a specific E_b/N_0 or E_s/N_0 SNR value, which we refer to as the *tunnel bound*. Therefore, if the SNR exceeds the tunnel bound, the EXIT curves will not intersect each other before reaching the $(1, 1)$ point in the EXIT chart and consequently the step-wise iterative decoding trajectory will also reach the $(1, 1)$ point. This results in a low BER, provided that the bit vector lengths are sufficiently high for this to be consistently achieved.

Here, we may conclude that near-capacity transmissions are facilitated, when a narrow, yet marginally open EXIT chart tunnel can be created for facilitating convergence to an infinitesimally low BER. Based on these observations of the bounds, the difference between the capacity bound and the area bound quantifies the loss that is mitigated by JSCC, while the difference between the area bound and the tunnel bound quantifies the capacity loss that is mitigated by irregular coding [135], as it will be discussed in the following section. Finally, the difference between the specific SNR, where a low BER is achieved and the tunnel bound quantifies the loss that can be mitigated by using longer bit vectors.

2.14 Irregular Design Philosophy

Irregular coding has been proposed for the reliable transmission of information at channel SNRs that are close to the channel's capacity bound [125] without imposing an excessive decoding complexity and latency. This concept was originally introduced [136] in the context of LDPC codes in [14, 137, 138], and then it was followed by applications in the context of turbo codes [139] and CCs [116].

Considering the serially concatenated scheme of Figure 2.2, the irregular design may be applied to the inner URC code, resulting in an Irregular Unity-Rate Convolutional (IrURC) as the inner code. Here, URCs having different parametrizations are used for the coding of different fractions of the bit vector $\mathbf{d}$ of Figure 2.2. As mentioned in Section 2.3.1, the $T = 10$ URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ [98] of Figure 2.6 can be employed as the component

codes of the IrURC code, since they have a variety of EXIT function shapes $I_t(\tilde{\mathbf{d}}^{\mathrm{e}}, \mathbf{d}) = f_{\mathrm{URC}^t}[I_t(\tilde{\mathbf{d}}^{\mathrm{a}}, \mathbf{d})]$. The specific fractions $\{\beta^t\}_{t=1}^{T=10}$ of the bit vector $\mathbf{d}$ that are encoded by each component code may be carefully chosen in order to shape the irregular EXIT function $I_t(\tilde{\mathbf{d}}^{\mathrm{e}}, \mathbf{d}) = f_{\mathrm{IrURC}}[I_t(\tilde{\mathbf{d}}^{\mathrm{a}}, \mathbf{d})]$ according to

$$f_{\mathrm{IrURC}}[I_t(\tilde{\mathbf{d}}^{\mathrm{a}}, \mathbf{d})] = \sum_{t=1}^{T} \beta^t \cdot f_{\mathrm{URC}^t}[I_t(\tilde{\mathbf{d}}^{\mathrm{a}}, \mathbf{d})], \tag{2.20}$$

where we have to obey

$$\sum_{t=1}^{T} \beta^t = 1. \tag{2.21}$$

Furthermore, the overall IrURC coding rate R_{IrURC} may be obtained according to

$$R_{\mathrm{IrURC}} = \sum_{t=1}^{T} \beta^t \cdot R_{\mathrm{URC}^t}. \tag{2.22}$$

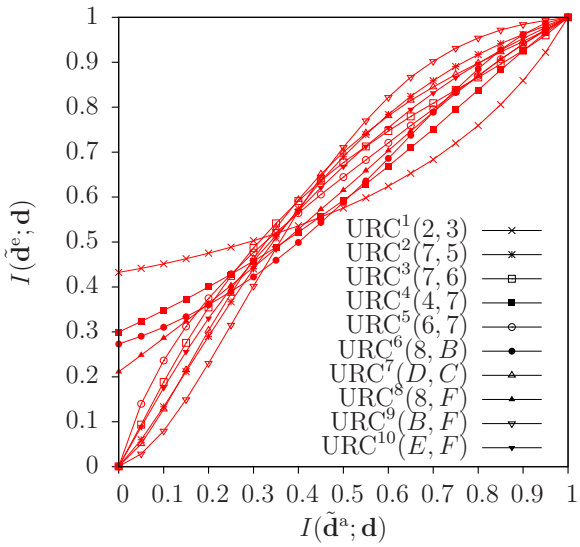

Figure 2.31: EXIT charts for the $T = 10$ URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ of Figure 2.6 for a QPSK-modulated Rayleigh fading channel having $E_s/N_0 = 4$ dB. Curves are labeled using the format $(g(\mathrm{URC}), f(\mathrm{URC}))$, where $g(\mathrm{URC})$ and $f(\mathrm{URC})$ are the hexadecimal generator and feedback polynomials of the URC code [98], respectively.

The EXIT functions of the $T = 10$ component URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ are illustrated in Figure 2.31, for the case where they are employed for transmission over an uncorrelated narrowband Rayleigh fading channel having an SNR of $E_s/N_0 = 4$ dB. Since the coding rate of each component URC code URC^t is $R_{\mathrm{URC^t}} = 1$, we have the overall coding rate of $R_{\mathrm{IrURC}} = 1$ for any IrURC code obtained by combining these component codes. By carefully selecting the fractions $\{\beta^t\}_{t=1}^{T=10}$, the IrURC EXIT function can be shaped to

create a narrow, but 'just' open EXIT chart tunnel with the inverted outer URC decoder's EXIT curve of Figures 2.22, 2.23 and 2.24. In this way, the tunnel bound will closely approach the capacity bound and near-capacity operation will be facilitated, when the bit vector length is sufficiently high. Note that the ability of EXIT chart matching to create a narrow, but 'just' open EXIT chart tunnel depends on the availability of a suite of component codes having a wide variety of EXIT function shapes, like that of Figure 2.31. In Chapter 5, we will design the candidate component codes for the UEC codes, and employ a novel double-sided EXIT chart matching algorithm to iteratively design the fractions for both the outer IrUEC code and the inner IrURC code. This results in a very narrow but still open EXIT chart tunnel that facilitates near-capacity operation.

2.15 Summary and Conclusions

In this chapter, we commenced our discussions in Section 2.2 by introducing two basic structures for iteratively decoded concatenated codes, namely the serially concatenated and parallel concatenated philosophy. After that, the chapter focussed on the serially concatenated scheme that is depicted in Figure 2.2 and successively described how each module operates in the transmitter, including the URC encoder, multiplexer, interleaver, QPSK modulator, as well as the corresponding modules in the receiver, including the URC decoder, demultiplexer, deinterleaver and QPSK demodulator. We also described how the uncorrelated narrow-band Rayleigh channel affects the transmitted symbols. Both the magnitude and the phase of the modulated symbols in the constellation diagram are influenced by the channel in terms of amplitude fading, phase rotation and additive noise.

When introducing the soft URC decoder of Section 2.11, we highlighted the APP SISO decoding algorithm, namely the BCJR algorithm that was invented by Bahl, Cocke, Jelinek and Raviv in 1972. For the sake of avoiding a complex circuit implementation due to the multiplications employed by the BCJR algorithm, as well as for the sake of reducing the memory requirement and allowing fixed-point processing to be used, the Log-BCJR algorithm and the Approx-Log-BCJR algorithm have found wider applications in practice. Following this, Section 2.12 described how the iterative decoding process exchanges extrinsic information between the two URC decoders by exchanging increasingly reliable vectors of LLRs and thus gradually improving the achievable error correction performance. After a hard-decision is made for the value of each message bit based upon the vector of LLRs output by the URC BCJR decoders, we characterised the BER curves of the serially concatenated scheme, when employing various parameter values, such as the number of iterations and bit vector length. Observe in the BER plots of Figures 2.18, 2.19 and 2.20, that lower BERs can be achieved when longer bit vectors are employed or when more

decoding iterations are performed. However, there is a limit as to how much the BER performance can be improved by performing more iterations, owing to the iterative decoding convergence behaviour of the concatenated codes

Although the BER plot is capable of characterizing the error correction performance of iteratively decoded concatenated codes, it is unable to closely characterise their convergence behaviour. This requires the EXIT charts [99] detailed in Section 2.13. An EXIT chart uses the MI for quantifying the quality of the extrinsic information exchanged between the constituent decoders in an iterative decoder. As discussed Section 2.13, we plotted the inverted EXIT curves for the outer URC code according the schematic of Figure 2.21, as well as the EXIT curves for the inner URC code according to the schematic of Figure 2.25. Having an open tunnel between the EXIT curves of the outer and inner codes facilitates a successful iterative decoding process towards a low BER. Finally, we considered irregular designs in Section 2.14, which may be invoked for facilitating near-capacity operation at low SNRs. In the following chapters, we will frequently refer back to the knowledge that has been introduced in this chapter.

Chapter 3

Unary Error Correction Codes and Their Complexity

3.1 Introduction

In this chapter, we introduce the structure of Unary Error Correction (UEC) codes [92, 96] as well as their encoding and decoding operations. Near-capacity operation is facilitated when our outer UEC code is serially concatenated with an inner code, such as the Unity-Rate Convolutional (URC) code of Section 2.3, which then iteratively exchange their extrinsic soft information. As highlighted in Figure 3.1, this chapter also relates to the probability distribution of the source symbols, the parametrization of the UEC code and the irregular design of the concatenated URC code.

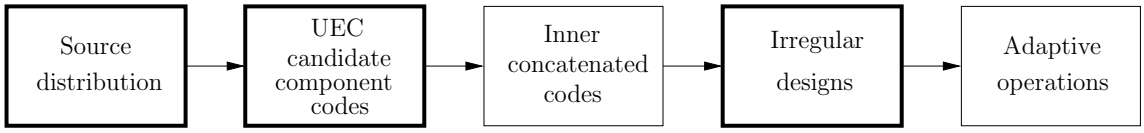

Figure 3.1: The design-flow of a UEC coded scheme. This chapter deals with the design aspects in the order indicated using the bold boxes.

3.1.1 Background and Motivation

In mobile wireless scenarios, multimedia transmission has to be bandwidth efficient and resilient to transmission errors, motivating both source and channel coding [140–142]. Classic Separate Source and Channel Coding (SSCC) [2] may be achieved by combining a near-entropy source code with a near-capacity channel code. In this scenario, it is theoretically possible to reconstruct the source information with an infinitesimally low probability of error, provided that the effective throughput of the transmission does not exceed the

channel's capacity [2]. However, SSCC is only capable of approaching the capacity in the general case by imposing both infinite complexity and infinite latency, provided that the error statistics are random, rather than bursty. For example, arithmetic coding [143] and Lempel-Ziv coding [22] are capable of encoding a sequence of symbols using a near-entropy number of bits per symbol. However, these schemes require both the transmitter and receiver to accurately estimate the occurrence probability of every symbol value that the source produces. In practice, the occurrence probability of rare symbol values can only be accurately estimated, if a sufficiently large number of symbols has been observed, hence potentially imposing an excessive latency.

This motivates the design of universal codes, such as the Elias Gamma (EG) codes [19], which facilitate the binary encoding of symbols selected from infinite sets, without requiring any knowledge of the corresponding occurrence probabilities at either the transmitter or receiver. The H.264 video codec [89] employs the EG code and this may be concatenated with classic channel codes, such as the serially concatenated pair of Convolutional Code (CC) [12] of Section 2.3, which provide a separate error correction capability. Nevertheless, this SSCC typically suffers from a capacity loss, owing to the residual redundancy that is typically retained during EG encoding, which results in an average number of EG-encoded bits per symbol that exceeds the entropy of the symbols.

In order to exploit the residual redundancy and hence to achieve near-capacity operation, the classic SSCC schemes may be replaced by Joint Source and Channel Coding (JSCC) arrangements [1] in many applications. As we will demonstrate in Section 3.2, the symbols that are EG encoded in H.264 are approximately zeta probability distributed [144], resulting in most symbols having low values, but some rare symbols having values around 1000. Until recently, the decoding complexity of all previous JSCCs, such as Reversible Variable Length Code (RVLC) [56] and Variable Length Error Correction (VLEC) code [57], increased rapidly with the cardinality of the symbol set, so much so that it became excessive for the H.264 symbol probability distribution. Indeed, the complexity asymptotically tends to infinity, when the cardinality of the source symbol set is infinite.

Against this background, a novel JSCC scheme was proposed in [92], which is referred to as the UEC code[1]. The UEC encoder generates a bit sequence by concatenating unary codewords [88] and then further encoding them using a trellis encoder, for the sake of protecting them against transmission errors. The decoder exploits the residual redundancy of the source sequence using a corresponding trellis that has only a modest complexity,

[1]The UEC scheme was originally proposed in [92], where the author of this thesis contributed the IrURC code design and Symbol Error Ratio (SER) simulations. This chapter is mainly based on [96] which further developed [92] by characterising the complexity of UEC codes primarily based on the contribution of the author of this thesis, so that aspects of [92] are also summarised for the sake of completeness.

even when the cardinality of the symbol value set is infinite. This trellis is designed so that the transitions between its states are synchronous with the transitions between the consecutive unary codewords in the concatenated bit sequence. Therefore, the UEC decoder is also able to exploit the residual redundancy using the classic Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm [91] of Section 2.11. In addition to detailing the operations of the UEC encoder and decoder, this chapter also quantifies the computational complexity of the UEC decoder in terms of the number of the Add, Compare and Select (ACS) operations discussed in Section 2.11.1, in order to facilitate a fair comparison between our UEC scheme and some suitably designed benchmarkers.

3.1.2 Novel Contributions

The novel contributions of this chapter are summarised as follows:

- An iteratively-decoded serial concatenation of the UEC code and an Irregular Unity-Rate Convolutional (IrURC) [98] is designed, which is capable of achieving near-capacity operation.
- Three application scenarios associated with different source distribution parametrizations are considered. In order to facilitate this, we introduce the employment of puncturing for controlling the effective throughputs of the proposed UEC scheme and of the benchmarkers, for the sake of facilitating fair comparisons in each of the scenarios considered.
- We quantify the computational complexity of the Log-BCJR algorithm used in the UEC decoder in terms of the number of the ACS operations required for the sake of providing fair comparisons between our UEC scheme and the benchmarkers. The ACS metric will also be used in all of the following chapters.

3.1.3 Chapter Organisation

The rest of this chapter is organised as follows:

- In Section 3.2, we introduce symbol value sets that have an infinite cardinality, such as the geometric distribution, the zeta distribution and the H.264 distribution.
- In Section 3.3, we review the operations of the UEC encoder, including the unary encoder and the UEC trellis encoder. The operations of the concatenated Irregular URC encoder, interleaver and modulator will be presented as well.
- In Section 3.4, we elaborate on how the UEC trellis decoder operates, and detail the iterative decoding operations exchanging extrinsic information between it and the IrURC decoder. The unary decoder makes a final decision on the values of the recovered symbols.

- In Section 3.5, we illustrate the EXIT chart concept and summarize the so-called area properties [108] of UEC codes, and mathematically prove that near-capacity operation can be achieved with the aid of an increased UEC trellis decoder complexity, regardless of the specific symbol value distributions.
- In Section 3.6, we introduce an SSCC EG-CC-IrURC benchmarker and discuss its encoding and decoding operations.
- In Section 3.7, we consider three different scenarios and offer a deeper insight into the parametrizations of both the proposed scheme as well as of the benchmarker, and detail the design of an IrURC that may be concatenated with the UEC and EG-CC schemes for the sake of achieving near-capacity operation.
- In Section 3.8, we characterize the computational complexity of the UEC and EG-CC schemes in terms of the number of ACS operations, which can be used to strike a desirable trade-off between the conflicting requirements of low complexity and near-capacity operation.
- In Section 3.10, we conclude this chapter.

3.2 Symbols Value Sets Having an Infinite Cardinality

The UEC code is designed for conveying a vector $\mathbf{x} = [x_i]_{i=1}^{a}$ comprising a number of symbols. Each symbol $x_i \in \mathbb{N}_1$ of the vector is obtained by an Independent and Identically Distributed (IID) Random Variable (RV) X_i with probability $Pr(X_i = x) = P(x)$, where $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$ is the infinite-cardinality set comprising all positive integers. Here, the symbol entropy is given by $H_X = \sum_{x \in \mathbb{N}_1} H[P(x)]$, where $H[p] = p \log_2(1/p)$.

For example, Figure 3.2 depicts the geometric distribution [144], which is defined as

$$P(x) = p_1(1 - p_1)^{x-1}, \tag{3.1}$$

where $p_1 = \Pr(X_i = 1)$ and the symbol entropy is given by

$$H_X = \frac{H[p_1] + H[1 - p_1]}{p_1}. \tag{3.2}$$

By contrast, the zeta distribution [144] of Figure 3.2 is defined as

$$P(x) = \frac{x^{-s}}{\zeta(s)}, \tag{3.3}$$

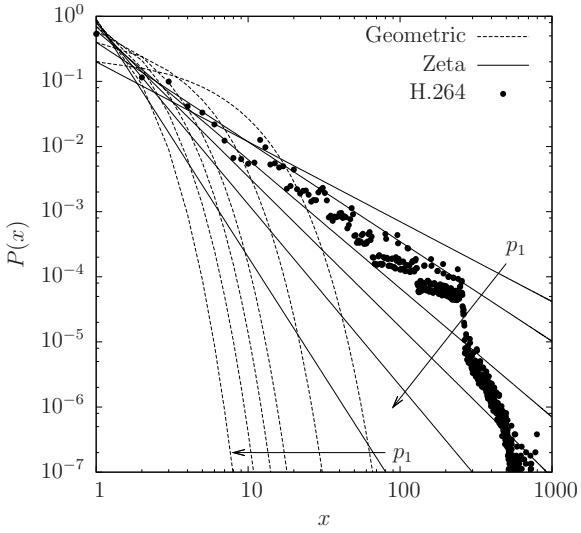

Figure 3.2: The geometric and zeta probability distributions for $p_1 \in \{0.2, 0.4, 0.6, 0.7, 0.8, 0.9\}$, as well as the H.264 distribution, where $p_1 = 0.536$. This was obtained by recording the values of the 44.6 million symbols that are EG-encoded when the JM 18.2 H.264 video encoder employs the '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.

where $\zeta(s) = \sum_{x \in \mathbb{N}_1} x^{-s}$ is the Riemann zeta function[2] and $s > 1$. In this case, $p_1 = 1/\zeta(s)$ and the symbol entropy is given by

$$H_X = \frac{\ln\left(\zeta(s)\right)}{\ln(2)} - \frac{s\zeta'(s)}{\ln(2)\zeta(s)}, \tag{3.4}$$

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

Figure 3.2[3] also exemplifies the distribution of the symbol values that are EG encoded by the H.264 video encoder, corresponding to an entropy of $H_X = 2.980$ bits per symbol. Note that EG code is a special case of the Exponential Golomb (ExpG) code, as shown in Table 7.2. Here, the symbols also include motion vectors, DCTs and all other EG-encoded parameters. Note that these symbol values appear to obey Zipf's law [144], since their distribution may be approximated by the zeta distribution. As a result, we mainly focus our attention on the zeta probability distribution in our work.

3.3 UEC Encoder Operation

Figure 3.3 illustrates the UEC scheme, which performs the JSCC encoding and decoding of symbols values that are selected from a set having an infinite cardinality. Explicitly,

[2]Expressing the zeta distribution as $P(x) = x^{-s} / \sum_{x \in \mathbb{N}_1} x^{-s}$ offers an attractive parallel with the geometric distribution, which may also be expressed in the form $P(x) = s^{-x} / \sum_{x \in \mathbb{N}_1} s^{-x}$, giving $p_1 = (s-1)/s$.

[3]The H.265 [90] software was not ready for use when [92] and [96] were written. However, the H.265 distribution is considered in the most recent work of Chapter 6.

the UEC encoder consists of two parts, the unary encoder and the trellis encoder, both of which are introduced in this section.

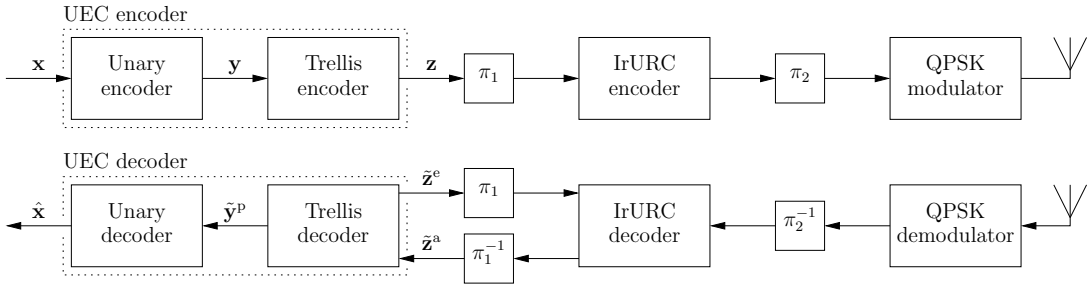

Figure 3.3: Schematic of the UEC JSCC scheme, when serially concatenated with IrURC coding and Gray-coded QPSK modulation schemes. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. Puncturing and depuncturing may also be performed by π_2 and π_2^{-1}, respectively.

3.3.1 Unary Encoder

As shown in Figure 3.3, the UEC scheme firstly encodes the symbol vector $\mathbf{x}$ using a unary encoder, which represents each symbol x_i by the corresponding unary codeword $\mathbf{y_i}$ that comprises x_i bits, as exemplified in Table 3.1. These codewords are then concatenated to obtain the b-bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$ shown in Figure 3.3. For example, the source symbol vector $\mathbf{x} = [4, 1, 2, 1, 3, 1, 1, 1, 2, 2]$ of $a = 10$ symbols yields the $b = 18$-bit vector $\mathbf{y} = [111001001100001010]$. Note that the average length of bit vector $\mathbf{y}$ is $a \cdot l$.

x_i	$P(x_i)$			$\mathbf{y}_i$	
	$p_1 = 0.7$	$p_1 = 0.8$	$p_1 = 0.9$	Unary	EG
1	0.7000	0.8000	0.9000	0	1
2	0.1414	0.1158	0.0717	10	010
3	0.0555	0.0374	0.0163	110	011
4	0.0286	0.0168	0.0057	1110	00100
5	0.0171	0.0090	0.0025	11110	00101
6	0.0112	0.0054	0.0013	111110	00110
7	0.0079	0.0035	0.0007	1111110	00111
8	0.0058	0.0024	0.0004	11111110	0001000
9	0.0044	0.0017	0.0003	111111110	0001001
10	0.0034	0.0013	0.0002	1111111110	0001010

Table 3.1: The first ten symbol probabilities for a zeta distribution having the parameter $p_1 = 0.7, 0.8, 0.9$, as well as the corresponding unary and EG codewords.

Note that the bit vector length b of $\mathbf{y}$ has an average value al that is finite, provided that the unary codewords have a finite average length

$$l = \sum_{x \in \mathbb{N}_1} P(x)x. \tag{3.5}$$

This is guaranteed [144] when the symbol values obey the geometric probability distribution of Eq. (3.1), in which case we have

$$l = \frac{1}{p_1}. \tag{3.6}$$

By contrast, when the symbols adopt the zeta distribution of Eq. (3.3), the average unary codeword length only remains finite[4] for $s > 2$ and hence for $p_1 > 0.608$, in which case we have

$$l = \zeta(s-1)/\zeta(s). \tag{3.7}$$

Note that for the H.264 symbol value distribution of Figure 3.2, we obtain the finite average unary codeword length of $l = 7.191$ bits per symbol.

3.3.2 Trellis Encoder

Following unary encoding, the bit sequence $\mathbf{y}$ is forwarded to the trellis encoder of Figure 3.3. This employs a UEC trellis of the sort depicted in Figure 3.4 to encode each bit y_j in the vector $\mathbf{y}$, in order of increasing bit-index j.

As shown in the generalized UEC trellis given in Figure 3.4, 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 $m_{j-1} \in \{1, 2, \ldots, r\}$ to its next state $m_j \in \{1, 2, \ldots, r\}$, 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

$$m_j = \begin{cases} 1 + \mathrm{odd}(m_{j-1}) & \text{if } y_j = 0 \\ \min[m_{j-1} + 2, r - \mathrm{odd}(m_{j-1})] & \text{if } y_j = 1 \end{cases}, \tag{3.8}$$

where the number of possible states r has to be even and the encoding process always begins from the state $m_0 = 1$. The function $\mathrm{odd}(\cdot)$ 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 $\mathbf{m} = [m]_{j=0}^{b}$ comprising $(b+1)$ state values.

For example, Figure 3.5 and Figure 3.6 exemplify two UEC trellises having $r = 4$ and $r = 6$ states, respectively. The bit vector $\mathbf{y} = [111001001100001010]$ yields the path $\mathbf{m} = [1, 3, 3, 3, 2, 1, 3, 2, 1, 3, 3, 2, 1, 2, 1, 3, 2, 4, 1]$ through the $r = 4$-state trellis of Figure 3.5, while yields the path $\mathbf{m} = [1, 3, 5, 5, 2, 1, 3, 2, 1, 3, 5, 2, 1, 2, 1, 3, 2, 4, 1]$ through the $r = 6$-state trellis of Figure 3.6. Note that 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 Eq. (3.11).

[4]Note that for zeta distributions having $p_1 \leq 0.608$, our Elias Gamma Error Correction (EGEC) code of [145] may be employed in order to achieve a finite average codeword length, as it will be detailed in Chapter 7.

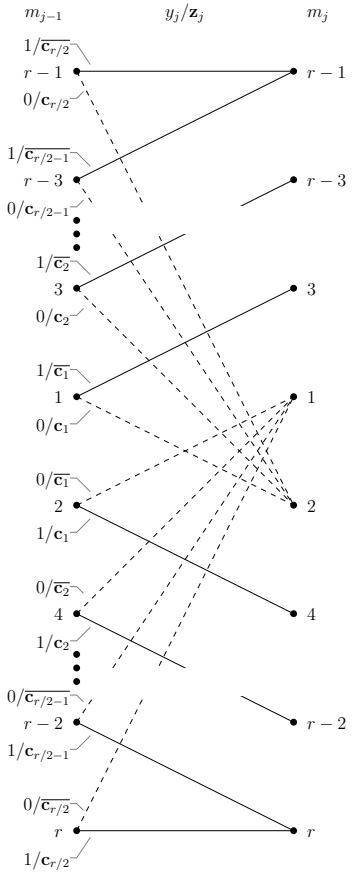

Figure 3.4: The generalized UEC trellis, having r states and n-bit codewords, where $\mathbb{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{r/2-1}, \mathbf{c}_{r/2}\}$.

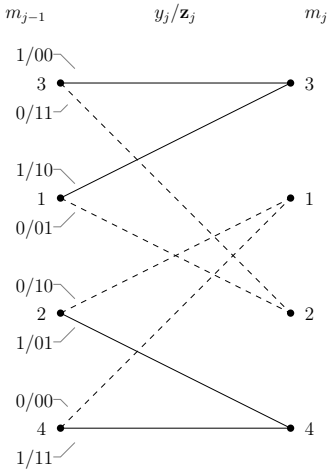

Figure 3.5: An $r = 4$-state $n = 2$-bit UEC trellis, where $\mathbb{C} = \{01, 11\}$.

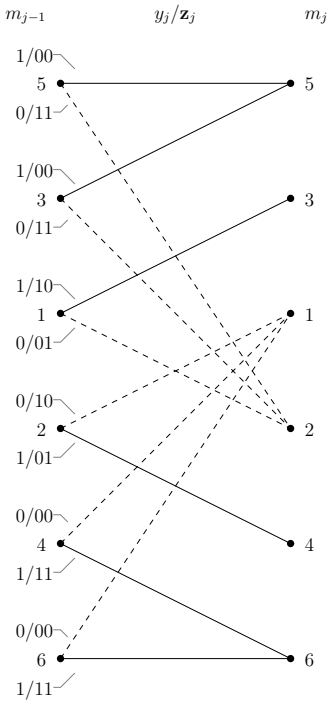

Figure 3.6: An $r = 6$-state $n = 2$-bit UEC trellis, where $\mathbb{C} = \{01, 11, 11\}$.

The transition path $\mathbf{m}$ may be modeled as particular realization of a RV vector $\mathbf{M} = [M_j]_{j=0}^{b}$, which are associated with the transition probabilities $\Pr(M_j = m, M_{j-1} = m') = P(m, m')$ is given in Eq. (3.11). Thus, the conditional transition probabilities $\Pr(M_j = m|M_{j-1} = m') = P(m|m')$ are given by

$$P(m|m') = \frac{P(m, m')}{\sum_{\check{m}=1}^{r} P(\check{m}, m')}, \tag{3.9}$$

while the entropy of the RV vector $\mathbf{M}$ is given by

$$H_{\mathbf{M}} = b \sum_{m'=1}^{r} \sum_{m=1}^{r} P(m, m') \log_2 \left(\frac{1}{P(m|m')} \right). \tag{3.10}$$

Note that the UEC trellis is designed to ensure that the transitions between its states are synchronous with the unary codewords of Table 3.1. More particularly, just as each symbol x_i in the symbol vector $\mathbf{x}$ corresponds to an x_i-bit codeword $\mathbf{y}_i$ in the bit vector $\mathbf{y}$, the symbol x_i also corresponds to a section $\mathbf{m}_i$ of the trellis path $\mathbf{m}$ comprising x_i transitions. A symbol having the value of $x_i < r/2$ and an odd index i results in a path section comprising the sequence of states $\mathbf{m}_i = [1, 3, 5, \ldots, 2x_i - 1, 2]$, while $\mathbf{m}_i = [2, 4, 6, \ldots, 2x_i, 1]$ is resulted when i is even. By contrast, for symbols having the value of $x_i \geq r/2$, an odd index i results in the sequence of x_i transitions $\mathbf{m}_i = [1, 3, 5, \ldots, r-1, r-1, \ldots, r-1, 2]$, while $\mathbf{m}_i = [2, 4, 6, \ldots, r, r, \ldots, r, 1]$ is yielded when i is even. As a result, a path sequence $\mathbf{m}_i$

$$P(m,m') = \begin{cases} \frac{1}{2l}\left[1 - \sum_{x=1}^{\lceil \frac{m'}{2} \rceil} P(x)\right] & \text{if } m' \in \{1,2,3,\ldots,r-2\},\ m = m'+2 \\ \frac{1}{2l} P(x)\Big|_{x=\lceil \frac{m'}{2} \rceil} & \text{if } m' \in \{1,2,3,\ldots,r-2\},\ m = 1+\text{odd}(m') \\ \frac{1}{2l}\left[1 - \sum_{x=1}^{\frac{r}{2}-1} P(x)\right] & \text{if } m' \in \{r-1,r\},\ m = 1+\text{odd}(m') \\ \frac{1}{2l}\left[l - \frac{r}{2} - \sum_{x=1}^{\frac{r}{2}-1} P(x)\left(x - \frac{r}{2}\right)\right] & \text{if } m' \in \{r-1,r\},\ m = m' \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

having an odd index i always starts from state 1 and ends at state 2, while sequences having an even index i start from state 2 and end at state 1. Owing to this, the trellis path $\mathbf{m}$ is guaranteed to be terminated at the state $m_b = 1$, when the symbol vector $\mathbf{x}$ has an even length a, while $m_b = 2$ is guaranteed, when a is odd.

The trellis encoder represents each bit y_j in the vector $\mathbf{y}$ by an n-bit codeword $\mathbf{z_j}$. This 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 $\overline{\mathbb{C}} = \{\overline{\mathbf{c}_1}, \overline{\mathbf{c}_2}, \ldots, \overline{\mathbf{c}_{r/2-1}}, \overline{\mathbf{c}_{r/2}}\}$, which is achieved according to

$$\mathbf{z}_j = \begin{cases} \overline{\mathbf{c}_{\lceil m_{j-1}/2 \rceil}} & \text{if } y_j = \text{odd}(m_{j-1}) \\ \mathbf{c}_{\lceil m_{j-1}/2 \rceil} & \text{if } y_j \neq \text{odd}(m_{j-1}) \end{cases} . \tag{3.12}$$

Finally, the selected codewords are concatenated to obtain the $(b \cdot n)$-bit vector $\mathbf{z} = [z_k]_{k=1}^{b \cdot n}$ of Figure 3.3. For example, when the $n = 2$-bit codewords $\mathbb{C} = \{01, 11\}$ are employed in the $r = 4$-state UEC trellis of Figure 3.5, the path $\mathbf{m} = [1,3,3,3,2,1,3,2,1,3,3,2,1,2,1,3,2,4,1]$ through the $r = 4$-state $n = 2$-bit trellis of Figure 3.5 corresponds to the encoded bit vector $\mathbf{z} = [1000001110100011100110101101 00]$. Similarly, when the $n = 2$-bit codewords $\mathbb{C} = \{01, 11, 11\}$ are employed in the $r = 6$-state UEC trellis of Figure 3.6, the path $\mathbf{m} = [1,3,5,5,2,1,3,2,1,3,5,2,1,2,1,3,2,4,1]$ through the $r = 6$-state $n = 2$-bit trellis of Figure 3.6 corresponds to the same encoded bit vector $\mathbf{z} = [1000001110100011100110101101 00]$. This is because the codewords $\mathbb{C} = \{01, 11, 11\}$ represent an extension of the codewords $\mathbb{C} = \{01, 11\}$, as it will be discussed in Section 4.2.

The bit vector $\mathbf{z}$ may be modeled as a specific realization of a vector $\mathbf{Z} = [Z_k]_{k=1}^{b \cdot n}$ comprising $b \cdot n$ binary RVs. Furthermore, the UEC trellis of Figure 3.4 has been designed to obey symmetry and to rely on complementary codewords, so that it produces mostly equiprobable bits, where $\Pr(Z_k = 0) = \Pr(Z_k = 1)$ and the bit entropy is $H_Z = 1$.

Therefore, it is possible for the UEC scheme to avoid capacity loss, as detailed in Section 3.4. 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, in contrast to those in the middle of the vector. In practice however, this is only apparent for bits 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 $b \cdot n$ and will be disregarded throughout the remainder of our work.

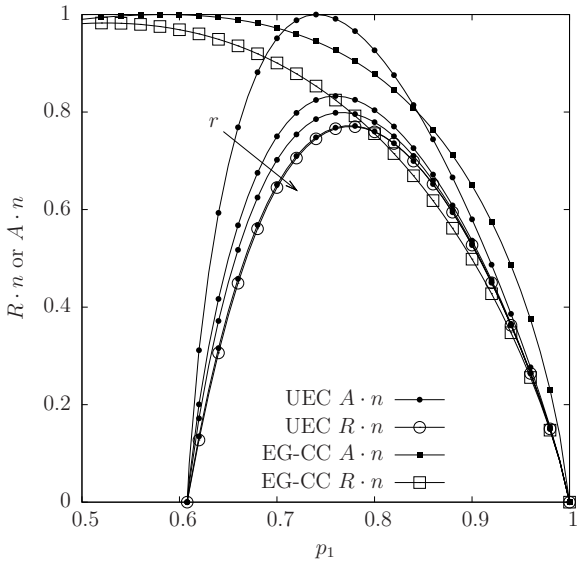

Figure 3.7: Plots of $R \cdot n$ and $A \cdot n$ that are obtained for the UEC and EG-CC schemes of Figures 3.3 and 3.10, in the case where the symbol values obey the zeta distribution of Eq. (3.3) having the parameter p_1. The value of $A \cdot n$ is provided for UEC codes having various numbers of states, namely $r \in \{2, 4, 6, 30\}$.

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

$$R = \frac{H_X}{l \cdot n} = \frac{1}{l \cdot n} \sum_{x \in \mathbb{N}_1} H[P(x)]. \tag{3.13}$$

In the case where the RVs in the vector $\mathbf{X}$ obey a geometric distribution, we can substitute Eq. (3.6) and Eq. (3.6) into Eq. (3.13) in order to obtain [146]

$$R = \frac{H[p_1] + H[1 - p_1]}{n}. \tag{3.14}$$

Figure 3.7 provides the corresponding plot for the case where the symbol values obey the zeta distribution of Eq. (3.7). Here, the coding rate R_o becomes 0 when $p_1 \leq 0.608$, since the average unary codeword length l becomes infinite in this circumstance.

3.3.3 IrURC Encoder, Interleaver, Puncturer and Modulator

As shown in Figure 3.3, the UEC-encoded bit vector $\mathbf{z}$ may be interleaved in the block π_1, IrURC encoded [98, 100] as detailed in Section 2.14 and then interleaved as well as

punctured in the block π_2. In accordance with convention, the coding rate R_i of this process is given by the number of input bits per output bit. More particularly, the IrURC encoder is comprised of $T = 10$ component codes, which are provided by the component URC codes $\{\mathbf{URC}^t\}_{t=1}^{T}$. As listed in Table 3.3, each URC component code is employed for encoding a particular fraction β of the bits provided by the interleaver π_1 of Figure 3.3. In Section 2.3, we have investigated the structures and coding process of these $T = 10$ component URC codes $\{\mathbf{URC}^t\}_{t=1}^{T}$, which are elaborately selected, since they can provide diverse choices of the EXIT functions, as discussed in Section 2.13. Moreover, the fractions β listed in Table 3.3 were chosen using the algorithm of [135] for the sake of generating EXIT chart curves that match those of the UEC schemes, as detailed in Section 2.14.

Note that in the absence of puncturing, the IrURC coding rate is given by the value of H_Z. If the UEC was not designed to produce equiprobable bits, then H_Z and the inner coding rate R_i would be less than unity, introducing capacity loss [131]. Following this, Quadrature Phase-Shift Keying (QPSK) modulation may be employed for transmission, as shown in Figure 3.3. Note that the puncturer may be adjusted for controlling the throughput $\eta = R_\mathrm{o} \cdot R_\mathrm{i} \cdot \log_2(M)$, where we have $M = 4$ for QPSK. The operations of interleaver, puncture and QPSK modulator have been already introduced in Chapter 2.

3.4 UEC Decoder Operation

In the receiver of Figure 3.3, QPSK demodulation, depuncturing π_2^{-1}, IrURC decoding and deinterleaving π_1^{-1} may be performed before invoking the proposed UEC decoder. In line with the UEC encoder, the UEC decoder also consists of two parts, which are the trellis decoder and the unary decoder.

3.4.1 Trellis Decoder

The trellis decoder of Figure 3.3 is provided with a vector of *a priori* LLRs $\tilde{\mathbf{z}}^\mathrm{a} = [\tilde{z}_k^\mathrm{a}]_{k=1}^{b\cdot n}$ that pertain to the corresponding bits in the vector $\mathbf{z}$. The trellis decoder applies the BCJR algorithm [91] to a UEC trellis of the sort shown in Figure 3.4, in order to consider every legitimate realization of $\mathbf{Z}$ having the particular length $b \cdot n$. Here, the value of $b \cdot n$ 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. The synchronization between the UEC trellis and the unary codewords is exploited during the BCJR algorithm's γ_t calculation of Eq. (9) in [91], by employing the conditional transition probabilities $P(m|m')$ of Eq. (3.9). As discussed in Section 3.3.2, the UEC trellis should be terminated at $m_0 = 1$ and either at $m_b = 1$ or $m_b = 2$, depending on whether the length a of the symbol vector $\mathbf{x}$ is even or odd, respectively. As shown in Figure 3.3, the BCJR decoder generates the vector of extrinsic LLRs $\tilde{\mathbf{z}}^\mathrm{e} = [\tilde{z}_k^\mathrm{e}]_{k=1}^{b\cdot n}$. Note that this BCJR algorithm has only a modest

complexity, since the trellis of Figure 3.4 may comprise as few as two states and because exact knowledge of the symbol probability distribution $P(x)$ is not required. Rather, the only knowledge required is that of the average unary codeword length l and the probabilities of the first $r/2-1$ symbol values, which may be accurately estimated heuristically, when $P(x)$ is unknown.

3.4.2 Iteratively Decoding

The extrinsic Logarithmic Likelihood Ratio (LLR) vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ of Figure 3.3 may be iteratively exchanged with the serially concatenated IrURC decoder. In-turn, the IrURC decoder may also iteratively exchange extrinsic LLR with the demodulator [147], 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 IrURC decoder and demodulator will also have an EXIT curve that reaches the $(1,1)$ point in the top right corner of the EXIT chart [134], iterative decoder convergence towards the Maximum Likelihood (ML) performance is facilitated [148]. At this point, the trellis decoder may invoke the BCJR algorithm for generating the vector of *a posteriori* LLR $\tilde{\mathbf{y}}^{\mathrm{p}} = [\tilde{y}_j^{\mathrm{p}}]_{j=1}^{b}$ that pertain to the corresponding bits in the vector $\mathbf{y}$.

3.4.3 Unary Decoder

The unary decoder of Figure 3.3 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 zero. A hard decision vector $\hat{\mathbf{y}}$ is then obtained by setting the value of these bits to zero and the value of all other bits to one. 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 obtain the symbol vector $\hat{\mathbf{x}}$ of Figure 3.3, which is guaranteed to comprise a number of symbols.

3.5 Near-capacity Performance of UEC Codes

Near-capacity operation is achieved, when reliable communication can be maintained at transmission throughputs that approach the capacity of the channel. When the UEC code of Figure 3.3 is serially concatenated with an IrURC code, near-capacity operation is facilitated, provided that the area A beneath the UEC code's inverted EXIT curve is equal to its coding rate R [131].

3.5.1 EXIT Curves

The transformation of $\tilde{\mathbf{z}}^{\mathrm{a}}$ into $\tilde{\mathbf{z}}^{\mathrm{e}}$ may be characterized by plotting the inverted UEC EXIT curve in an EXIT chart [149], as exemplified in Figure 3.8. 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 its EXIT curve will reach the $(1, 1)$ point in the top right corner of the EXIT chart [150].

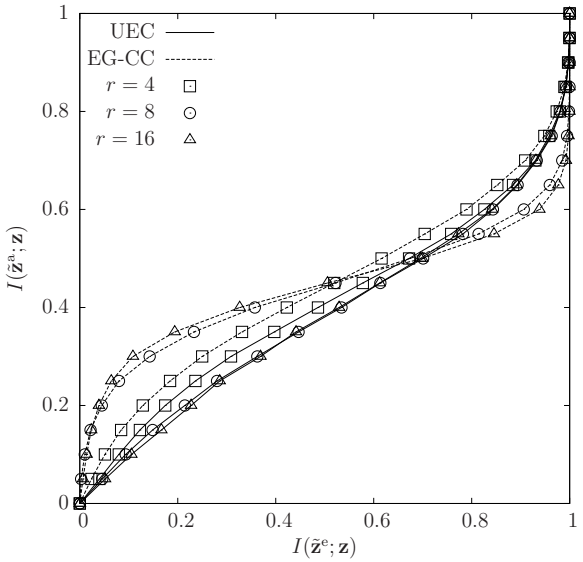

Figure 3.8: Inverted EXIT curves of the r-state BCJR decoders employed in the UEC and EG-CC schemes of Figures 3.3 and 3.10, for the case where the *a priori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{a}}$ is modeled by the transmission of $\mathbf{z}$ over a BEC and the symbol values obey a zeta probability distribution of Eq. (3.3) having the parameter value $p_1 = 0.797$. In the UEC scheme, the first codeword in the set $\mathbb{C}$ has the value 01 and the remaining $r/2 - 1$ codewords have the value 11, giving a free-distance of $d_{\mathrm{free}} = 4 \; \forall \; r \in \{4, 6, 8, \ldots\}$. The EG-CC scheme employs the $n = 2$-bit CCs of Table 3.3.

As mentioned in Section 2.13, in the case where the *a priori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{a}}$ is modeled by the transmission of $\mathbf{z}$ over a Binary Erasure Channel (BEC), the EXIT chart area A that is situated below the inverted UEC EXIT curve is given by

$$A = \frac{1}{n} \sum_{m'=1}^{r} \sum_{m=1}^{r} P(m, m') \log_2 \left(\frac{1}{P(m|m')} \right). \tag{3.15}$$

This result may be obtained from Eq. (23) of [131], where the notation may be converted according to $\mathcal{A} = 1 - A$, since $\mathcal{A}$ is defined as the area *above* the inverted EXIT curve. Furthermore, we employ $I^2_{A,\max} = 1$ for the total area enclosed within the EXIT chart, while the sum of the entropies of the RVs in the vector $\mathbf{Z}$ is given by $\sum_{i=1}^{m} H(V_i) = b \cdot n \cdot H_Z$. Here, we employ $H(\underline{V}|\underline{Y}) = H(\underline{V})$ as in Eq. (27) of [131], since the UEC decoder is employed as an outer decoder, which has no access to channel information.

Finally, we employ $H(\underline{V}) = H_{\mathbf{M}}$, which is given in Eq. (3.10). 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 $\mathbf{M}$. Note that the EXIT chart of Figure 3.8 and area calculation of Eq. (3.15) offer good approximations for cases where the LLRs of $\tilde{\mathbf{z}}^{\mathrm{a}}$ obey distributions other than that modeled by a BEC [131]. Substituting Eq. (3.11) and Eq. (3.9) into Eq. (3.15) and then rearranging the result yields [146]

$$\begin{aligned} A &= \tfrac{1}{ln}\textstyle\sum_{x=1}^{\frac{r}{2}-1} H[P(x)] + \tfrac{2}{ln} H\left[1 - \sum_{x=1}^{\frac{r}{2}-1} P(x)\right] \\ &+ \tfrac{1}{ln} H\left[l - \tfrac{r}{2} - \textstyle\sum_{x=1}^{r/2-1} P(x)\left(x - \tfrac{r}{2}\right)\right] \\ &- \tfrac{1}{ln} H\left[1 + l - \tfrac{r}{2} - \textstyle\sum_{x=1}^{r/2-1} P(x)\left(1 + x - \tfrac{r}{2}\right)\right]. \end{aligned} \tag{3.16}$$

In the case where the RVs in the vector $\mathbf{X}$ obey the geometric distribution of Eq. (3.1), the product of the UEC EXIT chart area A and the codeword length n is related to the distribution parameter p_1, as shown in Figure 3.7. Note that this relationship is symmetric about $p_1 = 0.5$, where $A(p_1) = A(1-p_1)$. Figure 3.7 provides the corresponding plots for the case where the symbol values obey the zeta distribution of Eq. (3.3) and where various numbers of states r are employed.

3.5.2 Area Property

In the schematic of Figure 3.3, near-capacity operation will be achieved if the IrURC decoder's EXIT curve has a shape closely matching with that of the UEC decoder, hence creating a narrow but still open EXIT chart tunnel and facilitating iterative decoding convergence towards the ML performance.

When the RVs in the vector $\mathbf{X}$ obey the geometric distribution of Eq. (3.1), the UEC EXIT area A is indeed equal to the UEC coding rate R, as shown in Figure 3.13. In fact, this is the case, regardless of both the specific number of states r employed and of the codeword length n. This may be shown by substituting Eq. (3.1) and Eq. (3.6) into Eq. (3.16) and then rearranging the result, yielding [146]

$$A = \frac{H[p_1] + H[1-p_1]}{n}, \tag{3.17}$$

which is identical to the expression provided for the coding rate R in Eq. (3.14).

In the case of the zeta distribution of Eq. (3.3), Figures 3.7 and 3.8 suggest that the UEC EXIT area A asymptotically approaches the UEC coding rate R as the number of states r is increased. In fact, this is the case, regardless of which particular symbol value distribution is adopted by the RVs in the vector $\mathbf{X}$. This may be proved by observing that the last three terms in Eq. (3.16) tend to zero as r is increased, leaving only the first term

of

$$\lim_{r\to\infty} A = \frac{1}{ln} \sum_{x\in\mathbb{N}_1} H[P(x)], \tag{3.18}$$

which is equal to the expression provided for the coding rate R in Eq. (3.13). Figure 3.9 plots the discrepancy between $R \cdot n$ and $A \cdot n$ as a function of the number of UEC states r for zeta distributions employing various values for the parameter p_1. More particularly, the lines in Figure 3.9 are obtained when $P(x)$ of Eq. (3.16) adopts the zeta distribution of Figure 3.2. Note that in all the cases considered, the discrepancy becomes less than 10^{-2}, when at least $r = 30$ states are employed.

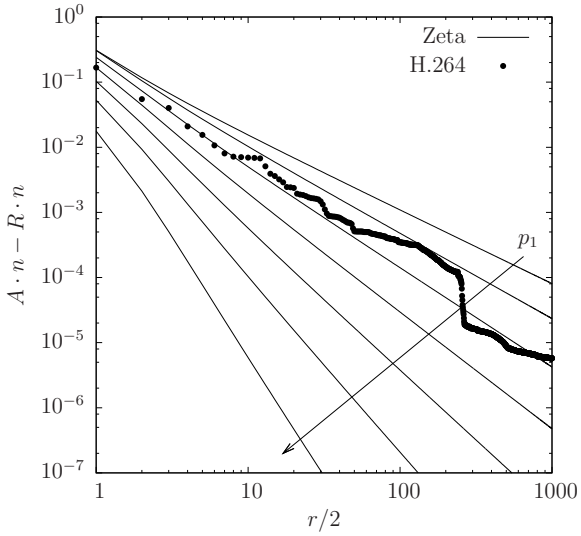

Figure 3.9: The discrepancy between $R \cdot n$ and $A \cdot n$ that results when UEC codes having various numbers of states r are employed to encode symbol values having zeta distributions with the parameters $p_1 \in \{0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$, as well as symbol values having the H.264 distribution of Figure 3.2.

Figure 3.9 also provides the corresponding plot for the case where the RVs in the vector $\mathbf{X}$ obey the H.264 distribution of Figure 3.2. More particularly, the dots in Figure 3.9 are obtained when $P(x)$ of Eq. (3.16) adopts the H.264 distribution of Figure 3.2. In this case $r = 14$ states are required for reducing the discrepancy below 10^{-2}. Here, the discrepancy between $R \cdot n$ and $A \cdot n$ quantifies the capacity loss. Note that the analysis of this section is specific to the case where the LLRs of $\tilde{\mathbf{z}}^{\mathrm{a}}$ adopt distributions that may be modeled by transmission over a BEC, since this is assumed by Eq. (3.16). However, the results of Section 3.9 demonstrate that the proposed UEC code is also capacity-approaching in cases where the LLRs of $\tilde{\mathbf{z}}^{\mathrm{a}}$ obey other distributions, for example when communicating over an uncorrelated narrowband Rayleigh fading channel.

3.6 An SSCC Benchmarker

The JSCC UEC-IrURC scheme of Figure 3.3 may be compared to the SSCC EG-CC-IrURC benchmarker, as shown in Figure 3.10. This is devised by replacing the components of the UEC code with the state-of-the-art SSCC components, on a like-for-like basis.

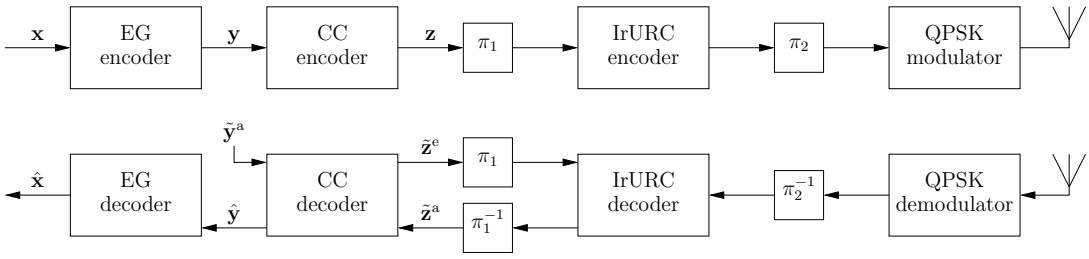

Figure 3.10: Schematic of the EG-CC SSCC scheme,when serially concatenated with IrURC coding and Gray-coded QPSK modulation schemes. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. Puncturing and depuncturing may also be performed by π_2 and π_2^{-1}, respectively.

3.6.1 EG-CC Encoder

In the transmitter of Figure 3.10, the unary encoder of the UEC scheme is replaced by an EG encoder. The first ten EG codewords are illustrated in Table 3.1. For example, the vector of $a = 8$ symbols $\mathbf{x} = [2, 1, 4, 2, 1, 3, 1, 1]$ yields the $b = 18$-bit vector $\mathbf{y} = [010100100010101111]$. The average EG codeword length is given by $l = \sum_{x \in \mathbb{N}_1} P(x) \left(2\lfloor \log_2(x) \rfloor + 1\right)$, which is guaranteed to be finite for any monotonic symbol value distribution having $P(x) \geq P(x+1) \, \forall \, x \in \mathbb{N}_1$, including the zeta distribution of Eq. (3.3) for all values of p_1. Even though the H.264 distribution of Figure 3.2 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. However in the EG-CC scheme, these RVs do not adopt equiprobable values $\Pr(Y_j = 0) \neq \Pr(Y_j = 1)$ in the general case, giving a value of less than unity for the corresponding bit entropy $H_{Y_j} = H[\Pr(Y_j = 0)] + H[\Pr(Y_j = 1)]$.

Similarly, the trellis encoder of the UEC scheme is replaced by a $1/n$-rate r-state CC encoder. We recommend the CCs that are described by the generator and feedback polynomials provided in Table 3.2. More specifically, we found that these non-systematic recursive CCs offer the optimal distance properties [151] subject to the constraint of producing equiprobable bits $\Pr(Z_k = 0) = \Pr(Z_k = 1)$. As described in Section 3.3, this $H_Z = 1$ constraint is necessary for avoiding a capacity loss, when the EG-CC scheme is serially concatenated with an IrURC code. Note that these non-systematic recursive CCs will also be employed as the basis of the benchmarkers in the following chapters.

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)

Table 3.2: 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.

The average coding rate R of the EG-CC encoder is given by Eq. (3.13). When the RVs in the vector $\mathbf{X}$ obey the geometric distribution of Eq. (3.1), the product of the EG-CC coding rate R and the codeword length n is related to the distribution parameter p_1, as shown in Figure 3.16. Similarly, Figure 3.7 provides the corresponding plot for symbol values obeying the zeta distribution of Eq. (3.3). The product becomes $R \cdot n = 0.983$ in the case of the H.264 symbol value distribution of Figure 3.2.

3.6.2 EG-CC Decoder

In the receiver, the trellis decoder of the UEC scheme is replaced by a CC decoder, as shown in Figure 3.10. This employs the BCJR algorithm during the iterative decoding process, in order to convert the *a priori* LLR vector $\tilde{\mathbf{z}}^{\text{a}}$ into the extrinsic LLR vector $\tilde{\mathbf{z}}^{\text{e}}$. As shown in Figure 3.10, the BCJR algorithm can exploit some of the residual redundancy present within the bit vector $\mathbf{y}$ by employing a corresponding vector of *a priori* LLRs $\tilde{\mathbf{y}}^{\text{a}} = [\tilde{y}_j^{\text{a}}]_{j=1}^{b}$. Here, we have

$$\tilde{y}_j^{\text{a}} = \ln\left(\frac{\Pr(Y_j = 0)}{\Pr(Y_j = 1)}\right), \tag{3.19}$$

where the bit value probabilities may be obtained heuristically.

As in the UEC scheme, the BCJR algorithm may be characterized by the corresponding inverted EG-CC EXIT curve, as exemplified in Figure 3.8. When the *a priori* LLR vector $\tilde{\mathbf{z}}^{\text{a}}$ may be modeled by the transmission of $\mathbf{z}$ over a BEC, it can be shown that the EXIT chart area A that is situated below the inverted EG-CC EXIT curve is given by

$$A = \frac{\sum_{j=1}^{b} H_{Y_j}}{b \cdot n}. \tag{3.20}$$

Note that unlike in the UEC scheme, the EG-CC EXIT chart area A is independent of the number of states r employed in the CC, as exemplified in Figure 3.9. If the RVs in the vector $\mathbf{X}$ obey the geometric distribution of Eq. (3.1), the product of the EG-CC EXIT

chart area A and the codeword length n is related to the distribution parameter p_1, as shown in Figure 3.9. Similarly, Figure 3.7 provides the corresponding plots for symbol values obeying the zeta distribution of Eq. (3.3). The product is $An = 0.998$ in the case of the H.264 symbol value distribution of Figure 3.2.

Note that in the EG-CC scheme, the values of $A \cdot n$ and $R \cdot n$ are separated by significant discrepancies of up to about 0.2, preventing near-capacity operation, as discussed in Section 3.5. This represents a significant disadvantage compared to the proposed UEC scheme, which can eliminate the discrepancy, when the symbol values obey the geometric distribution of Eq. (3.1). Note that if the EG code of the EG-CC scheme is replaced with a Golomb code, then the discrepancy between $A \cdot n$ and $R \cdot n$ is eliminated when the symbol values obey the geometric distribution of Eq. (3.1) [88]. However, a significant discrepancy can still be observed for other distributions. For these other symbol value distributions, 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 3.5.

3.7 Parametrization of the UEC-IrURC and EG-CC-IrURC schemes

In this section, we detail a number of parametrizations conceived for the UEC-IrURC and EG-CC-IrURC schemes. As shown in Table 3.4, we consider source symbols that obey zeta distributions having $p_1 \in \{0.7, 0.8, 0.9\}$. In all cases, we opted for employing UEC and CC codes having $n = 2$-bit codewords, since this is the minimum number required for facilitating iterative decoding convergence to the ML error correction performance [92]. Note that for each of the p_1 values considered, the UEC and EG-CC schemes have different outer coding rates R_o, as may be calculated using Eq. (3.14). For the sake of fair comparisons, the scheme having the lower outer coding rate R_o may be compensated using an inner coding rate of $R_\mathrm{i} > 1$, which is achieved by applying puncturing within the block π_2 of Figure 3.3. As shown in Table 3.4, this approach can be employed for achieving the same throughput of $\eta = R_\mathrm{o} \cdot R_\mathrm{i} \cdot \log_2(M)$ for both the UEC-IrURC and the EG-CC-IrURC scheme for each value of $p_1 \in \{0.7, 0.8, 0.9\}$.

Note that the approach described here is in contrast to that of [92], which only considered source symbols that obey zeta distributions having $p_1 = 0.797$. This value was chosen since it results in identical outer coding rates R_o for the UEC and EG-CC schemes, avoiding the requirement for puncturing in order to obtain fair comparisons. Table 3.4 provides the E_b/N_0 values at which the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity C of QPSK modulation in a uncorrelated narrowband Rayleigh fading channel becomes equal to the throughput η of the various schemes considered, as mentioned in Section 2.13.

As shown in Table 3.4, we consider UEC codes having $r \in \{4, 6, 8, 10, 32\}$ states. In each case, the first codeword in the set $\mathbb{C}$ has the value 01 and the remaining $(r/2 - 1)$ codewords have the value 11. By contrast, the EG-CC scheme employs the $n = 2$-bit $r = 4$-state CC of Table 3.2, since this was found to offer superior error correction performance over that of CCs having a higher number of states r. Figure 3.11, 3.12 and 3.13 provides the inverted EXIT curves [134] of the r-state UEC and EG-CC BCJR decoders, for the scenario where the symbol values obey zeta probability distributions having $p_1 \in \{0.7, 0.8, 0.9\}$. For each case, Table 3.4 provides the area A_{o} beneath the UEC and EG-CC EXIT curves, as may be calculated according to Eq. (3.16). In the case where the RVs in the vector $\mathbf{X}$ obey the geometric distribution, the product of the UEC EXIT chart area A and the codeword length n is related to the distribution parameter p_1. Figure 3.7 provides the corresponding plots for the case where the symbol values obey the zeta distribution of Eq. (3.3) and where various numbers of states r are employed.

It may be observed that the discrepancy between A_{o} and R_{o} diminishes, as the number of states r employed in the UEC scheme is increased. This may be expected since [92] showed that the discrepancy tends to zero, as r approaches infinity. By contrast, a significant discrepancy can be observed between A_{o} and R_{o} for the EG-CC scheme, for each value of p_1. Note that the discrepancy is independent of r in the case of the EG-CC scheme, as discussed in UEC.

p_1	Scheme	URC component code fractions β										ACS
		$r = 2$	$r = 4$				$r = 8$					
		(2,3)	(7,5)	(7,6)	(4,7)	(6,7)	(8,B)	(D,C)	(8,F)	(B,F)	(E,F)	
0.7	UEC	0	0	0.44	0	0.44	0	0.10	0	0.02	0	120.9
	EG-CC	0.35	0	0	0.18	0.17	0.05	0	0.25	0	0	121.5
0.8	UEC	0.18	0	0.71	0.10	0.01	0	0	0	0	0	96.6
	EG-CC	0.30	0	0.33	0.27	0.10	0	0	0	0	0	89.6
0.9	UEC	0	0	0.33	0	0	0.09	0.58	0	0	0	184.7
	EG-CC	0	0	0.85	0	0	0.02	0.13	0	0	0	124.4

Table 3.3: As mentioned in Section 2.3.1, the $T = 10$ URC component codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ [98] are labeled using the format (g, f), where g and f are the hexadecimal generator and feedback polynomials of the URC code, respectively.

As shown in Figure 3.3, the R_{o}-rate UEC and EG-CC schemes form the outer code in a serial concatenation with an R_{i}-rate inner code, which is comprised of an IrURC code and a puncturer π_2. In order to achieve iterative decoding convergence towards a vanishingly low error probability, it is necessary for the area beneath the inner EXIT curve A_{i} to exceed A_{o}, so that an open EXIT chart tunnel can be created. It can be shown that we have $A_{\text{i}} = C/[R_{\text{i}} \cdot \log_2(M)]$ for this inner code, regardless of how the IrURC code is parametrized. Table 3.4 provides the E_{b}/N_0 values at which A_{i} becomes equal to

A_o for the various schemes considered. These *area bounds* represent the lowest E_b/N_0 values, where it would be possible to create an open EXIT chart tunnel, if the IrURC code was particularly well parametrized. Note that the discrepancies between the area bound and the capacity bound represent the *capacity loss* of each scheme considered. As shown in Table 3.4, the EG-CC schemes suffer from a significant capacity loss, which can be eliminated by employing a UEC scheme having a sufficiently high number of states r.

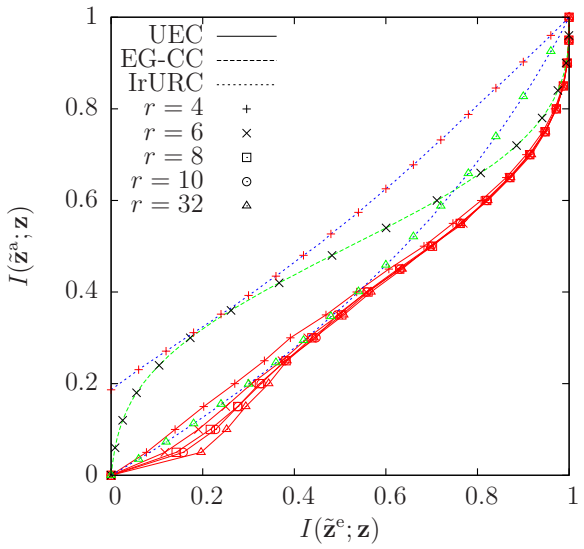

Figure 3.11: Inverted EXIT curves for the UEC BCJR decoder having $r \in \{4, 6, 8, 10, 32\}$ states and EG-CC BCJR decoder having $r = 4$ states, where $p_1 = 0.7$. Corresponding EXIT curves are provided for the IrURC schemes of Table 3.3, at the lowest E_b/N_0 values that facilitates the creation of an open tunnel with the EXIT curves of the $r = 32$-state UEC and the $r = 4$-state EG-CC, as listed in Table 3.4.

An IrURC parametrization was designed to be serially concatenated with the UEC and EG-CC schemes for each value of $p_1 \in \{0.7, 0.8, 0.9\}$. These IrURCs were constrained to the choice of the 10 URC component codes in Figure 4 of [98], since their decoders employ no more than $r = 8$ states and therefore do not have an excessive computational complexity compared to the UEC and EG-CC decoders, as shown in Table 3.5. As listed in Table 3.3, each URC component code is employed for encoding a particular fraction β of the bits provided by the interleaver π_1 of Figure 3.3. These fractions β were chosen using the algorithm of [135] for the sake of generating EXIT curves that match those of the $r = 32$ UEC scheme and of the $r = 4$ EG-CC scheme. As shown in Figure 3.11, 3.12 and 3.13, this facilitated the creation of open EXIT chart tunnels at the lowest possible E_b/N_0 values, as listed in Table 3.4. These E_b/N_0 values maybe deemed to represent *the open tunnel bounds*, which must be exceeded for the sake of creating an open tunnel in practice, hence facilitating iterative decoding convergence to a low error probability, when the number of symbols a in the source sequence $\mathbf{a}$ is infinite. Note that the discrepancies between the open tunnel bounds and area bounds may be attributed to the constraining

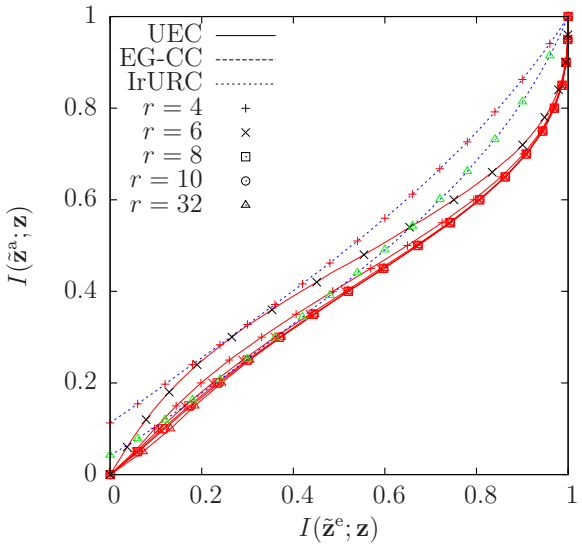

Figure 3.12: Inverted EXIT curves for the UEC BCJR decoder having $r \in \{4, 6, 8, 10, 32\}$ states and EG-CC BCJR decoder having $r = 4$ states, where $p_1 = 0.8$. Corresponding EXIT curves are provided for the IrURC schemes of Table 3.3, at the lowest E_{b}/N_0 values that facilitates the creation of an open tunnel with the EXIT curves of the $r = 32$-state UEC and the $r = 4$-state EG-CC, as listed in Table 3.4.

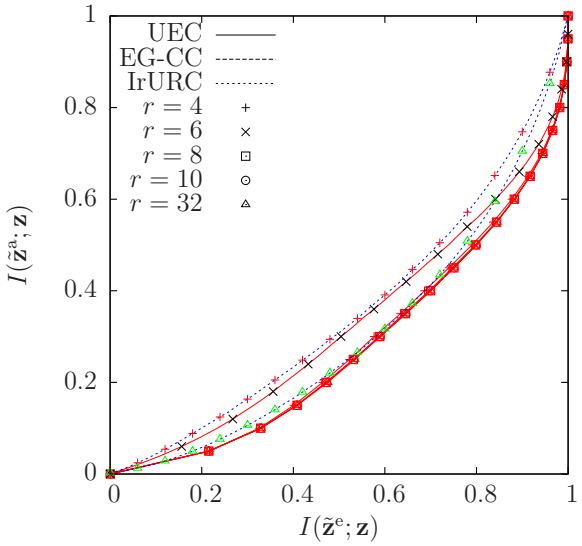

Figure 3.13: Inverted EXIT curves for the UEC BCJR decoder having $r \in \{4, 6, 8, 10, 32\}$ states and EG-CC BCJR decoder having $r = 4$ states, where $p_1 = 0.9$. Corresponding EXIT curves are provided for the IrURC schemes of Table 3.3, at the lowest E_{b}/N_0 values that facilitates the creation of an open tunnel with the EXIT curves of the $r = 32$-state UEC and the $r = 4$-state EG-CC, as listed in Table 3.4.

of the IrURC parametrization to using only the low-complexity URC component codes in Figure 4 of [98]. As shown in Table 3.4, the UEC schemes may be seen to offer significant gains over the EG-CC schemes, when $p_1 \in \{0.8, 0.9\}$. By contrast, for $p_1 = 0.7$ the UEC and EG-CC schemes may be seen to have similar open tunnel bounds. Table 3.4 quantifies the number of ACS operations that are performed per bit of $\mathbf{z}$. As described in Section 3.8, these values are obtained by employing the fractions β as weights, when averaging the number of ACS operations that are listed in Table 3.5 for URCs having $r \in \{2, 4, 8\}$ states.

3.8 Decoding Complexity Analysis

In this section, we characterize the computational complexity of the various decoders that are employed in the UEC-IrURC and EG-CC-IrURC schemes. As mentioned in Section 2.11, this is achieved by observing that each of the decoders listed in Table 3.5 operates on the basis of only using the $\max^*$ [152] and addition operations, where

$$\max^*(\tilde{z}_1, \tilde{z}_2) = \max(\tilde{z}_1, \tilde{z}_2) + \ln(1 + e^{-|\tilde{z}_1 - \tilde{z}_2|}). \tag{3.21}$$

The complexity of each type of decoder scales linearly with the number of bits $b \cdot n$ in the encoded vector $\mathbf{z}$ of Figure 3.3. Therefore, Table 3.5 lists the number of $\max^*$ and addition operations that are performed per bit of $\mathbf{z}$, when each type of decoder employs a trellis having r states. Note that the computational complexity of the trellis and CC decoder depends on whether it is used for generating an output pertaining to the bit vector $\mathbf{y}$ or to $\mathbf{z}$.

In practice, the term $f_c = \ln(1 + e^{-|\tilde{z}_1 - \tilde{z}_2|})$ in the $\max^*$ operation can be implemented at a low computational complexity by employing a Look-Up-Table (LUT) [152]. When employing an 8-entry LUT, a total of $\log_2(8) = 3$ compare operations are required for selecting the particular LUT entry that best approximates f_c. Furthermore, a compare operation is required for computing $\max(\tilde{z}_1, \tilde{z}_2)$ and an addition operation is required evaluating $\max(\tilde{z}_1, \tilde{z}_2) + f_c$. Therefore, each $\max^*$ operation can be considered to be equivalent to five Add, Compare and Select (ACS) arithmetic operations. By contrast, each addition operation corresponds to a single ACS operation. Using this logic, Table 3.5 lists the total number of ACS operations that are performed by each type of decoder. Note that since an IrURC is formed as a combination of URC components having different numbers of states r, the computational complexity of an IrURC decoder can be quantified as a weighted average of the URC complexities given in Table 3.5. The weights that should be employed in this weighted average are given by the fraction of bits β that are encoded by URC component codes having each number of states r, as it will be exemplified in Section 3.7.

p_1	Scheme	Figure	r	R_o	A_o	R_i	η	$E_\mathrm{b}/N_0\ [dB]$ capacity bound	$E_\mathrm{b}/N_0\ [dB]$ area bound	$E_\mathrm{b}/N_0\ [dB]$ tunnel bound
0.7	EG-CC	3.10	4	0.4503	0.4861	1	0.9006	1.39	2.03	3.5
	UEC	3.3	4	0.3226	0.3751	1.3958			2.70	3.8
			6		0.3510				2.09	3.7
			8		0.3412				1.85	3.7
			10		0.3361				1.72	3.6
			32		0.3253				1.46	3.4
0.8	EG-CC	3.10	4	0.3779	0.4387	1.0048	0.7594	0.83	1.96	3.1
	UEC	3.3	4	0.3797	0.4019	1			1.24	2.4
			6		0.3896				1.01	2.0
			8		0.3853				0.92	1.8
			10		0.3833				0.90	1.8
			32		0.3801				0.84	1.8
0.9	EG-CC	3.10	4	0.2492	0.3247	1.0578	0.5272	0.01	1.72	2.2
	UEC	3.3	4	0.2636	0.2682	1			0.11	0.9
			6		0.2651				0.04	0.9
			8		0.2642				0.02	0.8
			10		0.2639				0.01	0.8
			32		0.2636				0.01	0.7

Table 3.4: Outer coding rate R_o, inner coding rate R_i and total throughput η for UEC-IrURC scheme of Figure 3.3 and EG-CC-IrURC scheme of Figure 3.10 with different states. Three categories of E_b/N_0 bounds where $C = \eta$ and $A_\mathrm{i} = A_\mathrm{o}$ in theory, and where tunnel is open in simulation, respectively.

Decoder	r	$\max^*$	add	ACS
$n = 2$-bit CC Viterbi decoder $\hat{\mathbf{y}}$	4	2	8	18
$n = 2$-bit CC BCJR decoder $\tilde{\mathbf{z}}^{\mathrm{e}}$	4	10	22	72
$n = 2$-bit Trellis BCJR decoder $\tilde{\mathbf{y}}^{\mathrm{p}}$	4	7	20	55
	6	11	30.5	85
	8	15	40.5	115.5
	10	19	50.5	145.5
$n = 2$-bit Trellis BCJR decoder $\tilde{\mathbf{z}}^{\mathrm{e}}$	4	10	22	72
	6	16	32	112
	8	22	42	152
	10	28	52	192
URC BCJR decoder	2	6	19	49
	4	14	37	107
	8	30	73	223

Table 3.5: Number of $\max^*$ and addition operations that are performed per bit of $\mathbf{z}$, for various types of decoder employing trellises having r states. Note that CC is also a trellis code. In this chapter, the CC has a similar trellis to the UEC trellis code when $r = 4$.

In the case where I decoding iterations are performed between the $n = 2$-bit $r = 4$-state trellis decoder and the $r = 2$-state URC decoder, the total number of ACS operations per symbol in the sequence $\mathbf{x}$ is given by

$$N_{\mathrm{ACS}} = \frac{b \cdot n}{a}(49 \cdot I + 72 \cdot (I - 1) + 55) = 2 \cdot l \cdot (121 \cdot I - 17). \tag{3.22}$$

3.9 Simulation Results

In this section, we characterize the SER performance of the UEC-IrURC scheme of Figure 3.3 and EG-CC-IrURC scheme of Figure 3.10. We consider the transmission of source symbol sequences $\mathbf{x}$ comprising $a = 10^4$ symbols, since this frame length is typical of multimedia applications [92]. In order to facilitate fair comparison between schemes having different computational complexities per iteration, we impose a limit of 10^4 ACS operations for the decoding of each symbol in $\mathbf{a}$. This facilitates an average of $t = 6$ decoding iterations in the $r = 10$ UEC scheme for $p_1 = 0.7$ and $t = 11$ iterations in the $r = 4$ scheme for $p_1 = 0.9$, which are the most and least complex schemes considered, respectively.

As shown in Figure 3.15 and 3.16, our UEC-IrURC scheme outperforms the EG-CC-IrURC benchmarker for transmission over uncorrelated Rayleigh channel, when $p_1 \in \{0.8, 0.9\}$. More particularly, when $p_1 = 0.9$, a 1.3 dB gain is offered by the UEC-IrURC scheme, while operating within 1.6 dB of the capacity bound. By contrast, in Figure 3.14, the EG-CC-IrURC benchmarker outperforms the UEC-IrURC scheme for $p_1 = 0.7$. This

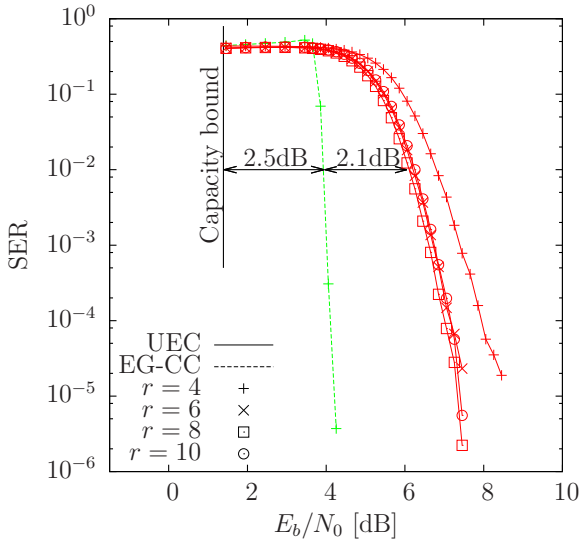

Figure 3.14: The SER performance of the UEC-IrURC scheme having $r \in \{4, 6, 8, 10\}$ states and the EG-CC-IrURC scheme having $r = 4$ states of Table 3.4, where $p_1 = 0.7$. Communication is over an uncorrelated narrowband Rayleigh fading channel and a complexity limit of 10^4 ACS operations is imposed for decoding each of the $a = 10^4$ symbols in $\mathbf{x}$. The schematics of Figures 3.3 and 3.10 were used.

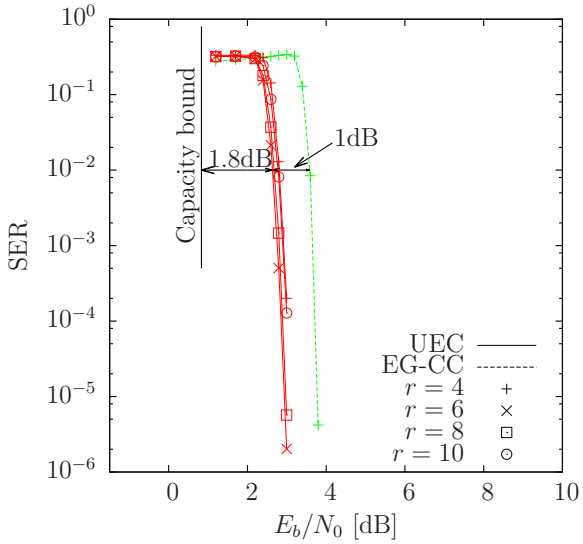

Figure 3.15: The SER performance of the UEC-IrURC scheme having $r \in \{4, 6, 8, 10\}$ states and the EG-CC-IrURC scheme having $r = 4$ states of Table 3.4, where $p_1 = 0.8$. Communication is over an uncorrelated narrowband Rayleigh fading channel and a complexity limit of 10^4 ACS operations is imposed for decoding each of the $a = 10^4$ symbols in $\mathbf{x}$. The schematics of Figures 3.3 and 3.10 were used.

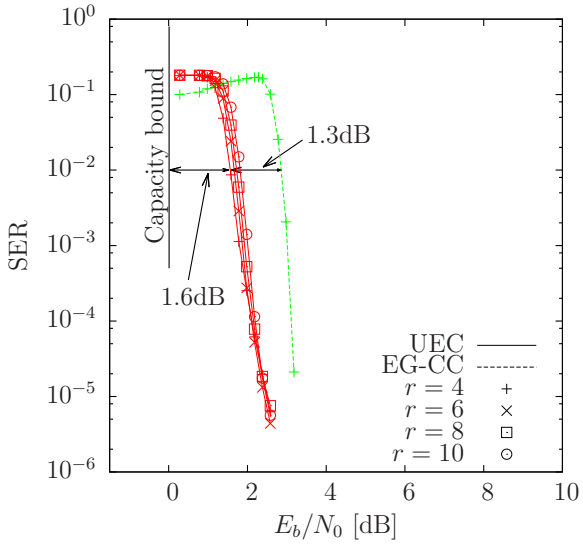

Figure 3.16: The SER performance of the UEC-IrURC scheme having $r \in \{4, 6, 8, 10\}$ states and the EG-CC-IrURC scheme having $r = 4$ states of Table 3.4, where $p_1 = 0.9$. Communication is over an uncorrelated narrowband Rayleigh fading channel and a complexity limit of 10^4 ACS operations is imposed for decoding each of the $a = 10^4$ symbols in $\mathbf{x}$. The schematics of Figures 3.3 and 3.10 were used.

may be attributed to the high number of bits that are punctured in the UEC-IrURC scheme for $p_1 = 0.7$. Owing to this, a relatively high number of decoding iterations are required for achieving iterative decoding convergence in this scheme, even when the E_b/N_0 value is high. As a result, a severe performance degradation is incurred, when imposing a limit of 10^4 ACS operations for decoding each symbol in $\mathbf{a}$. As shown in Figure 3.16, a UEC decoder employing $r = 6$ states offers the optimal trade-off between complexity and near-capacity operation when $p_1 = 0.8$, while $r = 8$ and $r = 4$ are the optimal choices for $p_1 = 0.7$ and $p_1 = 0.9$, respectively.

3.10 Summary and Conclusions

In this chapter, we described the encoding and decoding operations of the UEC code, which is a novel JSCC scheme designed for encoding symbol values that are selected from a set having an infinite cardinality, such as the set of all positive integers. The serially concatenated UEC-IrURC scheme of Figure 3.3 was proposed for facilitating practical near-capacity operation. Moreover, we quantified the performance and computational complexity of the UEC scheme in order to strike a desirable trade-off between the conflicting requirements of low complexity and near-capacity operation. Finally, our simulation results of Figures 3.14, 3.15 and 3.16 demonstrated that the UEC-IrURC scheme of Figure 3.3 outperforms the EG-CC-IrURC benchmarker of Figure 3.10, offering a 1.3 dB gain, when operating within 1.6 dB of the capacity bound. Note that this gain is achieved

for 'free', without increasing either the decoding complexity, or the transmission-duration, -bandwidth or -energy.

In Section 3.2, we characterised a pair of probability distributions that generate symbol values selected from an infinite cardinality, namely the geometric distribution and the zeta distribution of Eq. (3.3). We have mainly focused our attention on the zeta distribution, since it can be used for approximating the H.264 distribution.

Section 3.3 and Section 3.4 described the operations of the UEC encoder and decoder, respectively. Explicitly, the UEC encoder consists of two parts, the unary encoder and the trellis encoder of Figure 3.3. Owing to the synchronization between the unary codewords and the trellis transitions, we found that the UEC decoder is capable of invoking the Log-BCJR algorithm for exploiting all the residual redundancy left by the source encoder and facilitates near-capacity operation at a moderate complexity.

In Section 3.5, we provided the EXIT chart analysis and characterized area properties of the UEC code. We demonstrated that in the case of arbitrary symbol value distributions [92], the capacity loss asymptotically approaches zero, as the complexity of the UEC trellis is increased.

Section 3.6 introduced an SSCC EG-CC-IrURC benchmarker and detailed the encoding and decoding operations of the EG-CC scheme. We showed that unless a recursive non-systematic CC is employed, the output of the EG-CC scheme is not equiprobable, hence resulting in a capacity loss.

In Section 3.7, we offered a deeper insight into the parametrization of the schemes, detailing the design of IrURCs that may be concatenated with the UEC and EG-CC schemes for achieving near-capacity operation.

In Section 3.8, we quantified the computational complexity of the UEC and EG-CC schemes in terms of the number of ACS operations, in order to strike a desirable trade-off between the conflicting requirements of low complexity and near-capacity operation. Moreover, we considered three different scenarios in Section 3.7. In order to facilitate this, we introduced puncturing for controlling the schemes' throughputs and for facilitating fair comparisons in each scenario considered.

In Section 3.9, the SER performance was characterized and compared between the UEC-IrURC and EG-CC-IrURC parametrizations of Table 3.4, when operating within a particular practical complexity limit. The UEC scheme was found to offer gains of up to 1.3 dB in Figure 3.16, when operating within 1.6 dB of the capacity bound. This is achieved without any increase in transmission energy, bandwidth, transmit duration or decoding complexity.

As depicted in Figure 3.1, there are also some other aspects that we have to consider, when designing a UEC scheme. Owing to the synchronous transitions between the trellis states and the unary codewords as well as due to the symmetry of the trellis structure, it is possible for us to beneficially exploit our novel trellis design, including the adaptive and iterative decoding, irregular trellis structure and heuristic learning technique. Motivated by this, in Chapter 4, we investigate an adaptive decoder exchanging extrinsic information between the UEC code and concatenated code, for the sake of expediting the decoding convergence and for striking an attractive trade-off between near-capacity operation and a low complexity. In Chapter 5, we consider the UEC trellis encoder and carefully select the component UEC candidate codes, in order to develop an irregular UEC scheme capable of near-capacity operation at even lower $E_{\rm b}/N_0$ values. In Chapter 6, we then consider non-stationary source distributions, where our UEC scheme learns the unknown source statistics and dynamically adapts to the non-stationary statistics, so that the attainable decoding performance can be gradually improved.

Chapter 4

Adaptive UEC Codes for Expediting Iterative Decoding Convergence

4.1 Introduction

In this chapter, we propose a novel serially concatenated structure, namely *Adaptive* UEC-Turbo scheme [93], which performs adaptive iterative decoding in order to expedite the convergence of the Unary Error Correction (UEC) codes. As highlighted in Figure 4.1, this chapter addresses the inner concatenated code design and adaptive decoding aspects of the scheme.

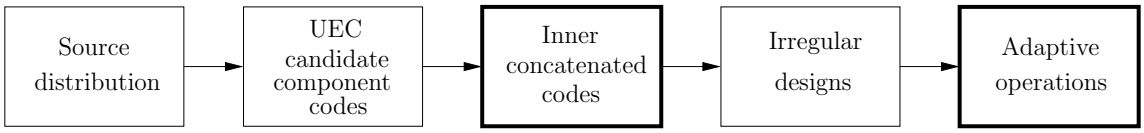

Figure 4.1: The design-flow of a UEC coded scheme. This chapter deals with the design aspects in the order indicated using the bold boxes.

4.1.1 Background and Motivation

The UEC code of Figure 3.3 constitutes a novel Joint Source and Channel Coding (JSCC) scheme capable of mitigating the capacity loss and facilitating near-capacity operation, even when the source symbol values are selected from a set having an infinite cardinality, such as the set of all positive integers. As discussed in Section 3.3, the UEC encoder in the context of Figure 3.3 generates a bit sequence by concatenating unary codewords, while the decoder employs the trellis of Figure 3.4 that has only a modest complexity. The UEC decoder is capable of exploiting the residual redundancy left by the source encoder in the

unary encoded bit sequence using the Log-BCJR algorithm, since the UEC trellis of Figure 3.4 was designed for ensuring that the transitions between its states are synchronous with the transitions between the consecutive unary codewords in the concatenated bit sequence. More specifically, Section 3.5 showed that when the symbol values have an arbitrary distribution, the capacity loss asymptotically approaches zero, as the number of states in the UEC trellis increases, albeit at the cost of increased decoding complexity. Motivated by this, in this chapter we conceive an adaptive UEC scheme that studies an attractive trade-off between its decoding complexity and its error correction capability.

Observe in the generalised UEC trellis of Figure 3.4 that an r-state n-bit UEC code is parametrized by a codebook $\mathbb{C}$ comprising $r/2$ number of codewords, each having n bits. A particular codebook $\mathbb{C}_1$ represents an extension of another codebook $\mathbb{C}_2$, if $\mathbb{C}_1$ can be obtained by repeating the last element in $\mathbb{C}_2$. For example, the $r = 6$ codebook $\mathbb{C} = \{01, 11, 11\}$ of Figure 3.6 can be considered to be an extension of the $r = 4$ codebook $\mathbb{C} = \{01, 11\}$ of Figure 3.5. In this case, the UEC-encoded bit vector $\mathbf{z}$ will always be identical, regardless of whether codebook $\mathbb{C} = \{01, 11, 11\}$ or codebook $\mathbb{C} = \{01, 11\}$ is employed, as observed in Section 3.3.2. In fact, regardless of how much a codebook $\mathbb{C}$ is extended, the UEC-encoded bit vector $\mathbf{z}$ will be identical, when encoding the same symbol vector $\mathbf{x}$. Moreover, when the number of states in the UEC trellis is increased, the capacity loss of the UEC code can be asymptotically reduced to zero, albeit at the cost of increased decoding complexity. We will exploit this property in Section 4.3 for dynamically adjusting the number of states employed in the UEC decoder by carefully adapting the decoding complexity.

As mentioned in Section 2.14, turbo codes [138], as well as the family of more general schemes employing the 'turbo principle' [153, 154] facilitate near-capacity operation without imposing an excessive decoding complexity or latency. This is achieved using iterative decoding by exchanging LLRs between a number of decoders. We may employ a carefully constructed hybrid combination of the serially concatenated scheme of Figure 2.2 and the parallel concatenated scheme of Figure 2.3, in order to construct a novel UEC-Turbo scheme, which is capable of providing a substantial design freedom for the iterative decoding between the UEC decoder and the two component decoders. In this way, a Three-Dimensional (3D) EXtrinsic Information Transfer (EXIT) chart may be conceived for quantifying the benefit of activating each decoder at each stage of the iterative decoding process, allowing us to activate the the specific component offering the greatest benefit at each stage. Owing to this, a novel adaptive activation order of the decoders is achieved, which expedites the convergence of iterative decoding.

Using these principles, this chapter proposes an adaptive iterative decoding technique

for expediting the iterative decoding convergence of UEC codes, which is referred to as the Adaptive UEC-Turbo scheme. We also propose the employment of 3D EXIT charts for controlling the dynamic adaptation of the UEC trellis decoder, as well as for controlling the decoder activation order between the UEC decoder and the turbo decoder. Our simulation results show that the novel adaptive decoding technique advocated provides an improved performance compared to a state-of-the-art JSCC benchmarker employing the non-adaptive UEC scheme and an Separate Source and Channel Coding (SSCC) benchmarker using the Elias Gamma (EG)-Convolutional Code (CC) scheme of Section 3.6.

4.1.2 Novel Contributions

The novel contributions of this chapter are summarised as follows:

- A three-stage concatenation of the UEC code and a half-rate turbo code is proposed, in which there are three decoders that facilitate a 3D EXIT chart analysis and an adaptive iterative decoding algorithm.
- We employ our 3D EXIT chart to estimate the potential quantitative benefits of activating each decoder at each specific stage of the iterative decoding process. Furthermore, a Two-Dimensional (2D) EXIT chart projection is employed to provide insights into whether or not any capacity loss is expected for the scheme.
- Based on the EXIT chart analysis, we dynamically adapt the activation order of the three component decoders of the UEC-Turbo scheme, in order to expedite the decoding convergence of the entire scheme. Moreover, the memory storage required for our 3D EXIT chart data is also quantified, which is shown to be readily acceptable in practice.
- As mentioned above, the more states are employed by the UEC trellis decoder, the nearer-capacity operation is achieved, but this is attained at the cost of an increased decoding complexity. We quantify the computational complexity of each turbo component decoder and of the UEC trellis decoder, when using different numbers of states in terms of the number of Add, Compare and Select (ACS) operations. Motivated by this, we also adaptively adjust the number of states that are employed in the UEC decoding trellis, in order to strike a desirable trade-off between the error correction performance attained and the complexity.
- A performance comparison is carried out between our Adaptive UEC-Turbo scheme and various non-adaptive state-of-the-art JSCC and SSCC benchmarkers. Our simulation results show that the proposed scheme outperforms the benchmarkers without requiring any additional decoding complexity, or transmission-duration, -bandwidth or -energy.

4.1.3 Chapter Organisation

The rest of this chapter is organised as follows:

- In Section 4.2, we propose our novel Adaptive UEC-Turbo scheme and describe the operations of its transmitter and receiver.
- In Section 4.3, we conceive the proposed adaptive iterative decoding algorithm, allowing not only the dynamic adjustment of the UEC decoder's operation, but also the dynamic adjustment of its iterative activation order exchanging extrinsic information with the two turbo decoder components. We also analyse the corresponding decoding complexity and storage requirements.
- In Section 4.4, both the capacity loss and the Symbol Error Ratio (SER) performance of the proposed Adaptive UEC-Turbo scheme is compared to those of the state-of-the-art benchmarkers employing either conventional SSCC or the non-adaptive UEC scheme of Figure 3.3.
- In Section 4.5, we conclude the chapter.

4.2 System Overview

As shown in Figure 4.2, our UEC code is serially concatenated with a turbo code that having two component Unity-Rate Convolutional (URC) codes. In this section, we detail the operation of the UEC-Turbo scheme of Figure 4.2. The transmitter's operation is described in Section 4.2.1, while the receiver is considered in Section 4.2.2, where we show that the number of states employed by the UEC decoder can be adapted independently of the number of states employed by the UEC encoder. Owing to the novelty of the proposed UEC-Turbo scheme, we show that the activation order of the three decoders may be dynamically adapted in order to strike an expediting decoding convergence.

4.2.1 Transmitter

The UEC encoder of [92] is designed for conveying a vector $\mathbf{x} = [x_i]_{i=1}^{a}$ comprising a number of symbols. The value of each symbol $x_i \in \mathbb{N}_1$ may be selected from a probability distribution $\Pr(X_i = x) = P(x)$ having an infinite cardinality, such as the set $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$ comprising all positive integers. In this chapter, we also assume that the symbol values obey a zeta probability distribution [144], since this is typical of the symbols produced by multimedia encoders, as described in Section 3.2. Without loss of generality, Table 4.1 exemplifies the first ten symbol probabilities $P(x_i)$ for the zeta distribution of Eq. (3.3) having the parameter of $p_1 = 0.797$, which corresponds to $s = 2.77$. We will frequently use $p_1 = 0.797$ throughout the remainder of this thesis, since it is found that

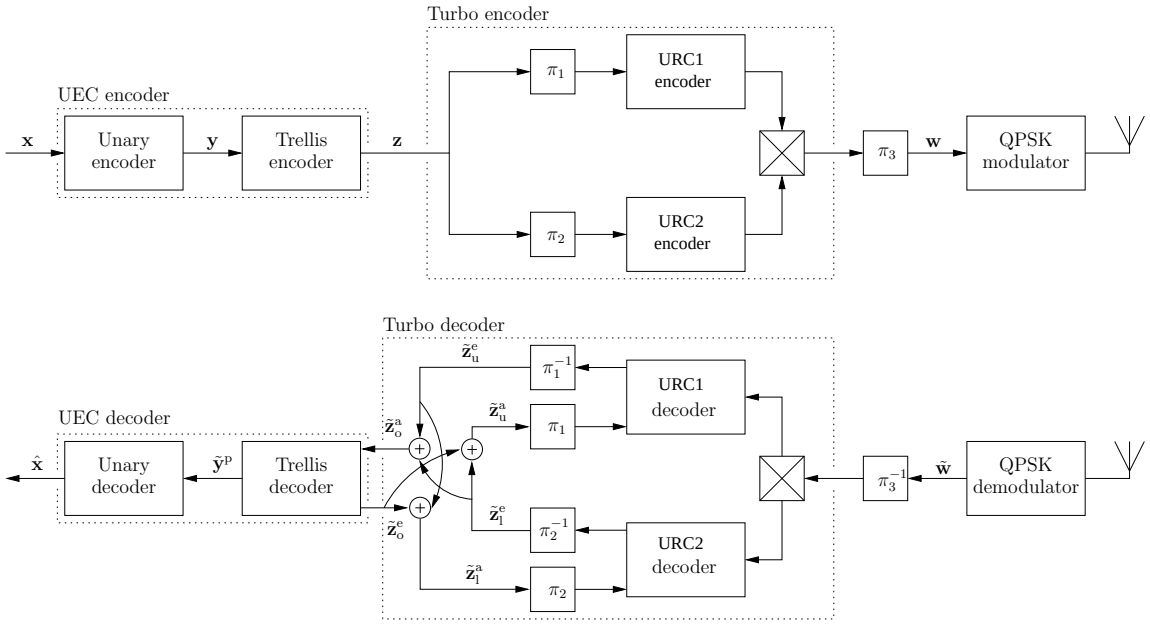

Figure 4.2: Schematic of the UEC-Turbo scheme, in which a UEC code is serially concatenated with turbo coding and Gray-coded QPSK modulation schemes. The subscripts 'o', 'u' and 'l' represent the outer UEC code, the upper turbo component code and the lower turbo component code, respectively. Here, π_1, π_2 and π_3 represent interleavers, while π_1^{-1}, π_2^{-1} and π_3^{-1} represent the corresponding deinterleavers. Multiplexer and de-multiplexer operations are also employed before π_3 and after π_3^{-1}, respectively. By contrast to the UEC-IrURC scheme of Figure 3.3, the IrURC code is replaced by a turbo code.

a fair comparison between unary- and EG-based schemes is achieved in this case. More specifically, the average unary and EG codeword lengths are equal for $p_1 = 0.797$.

The basic operations of the UEC encoder, including the unary encoder and trellis encoder, have been introduced in Section 3.3. Here, we employ a new example and a different trellis structure for illustrating that the number of states employed by the UEC trellis decoder may be independent of that employed by the UEC trellis encoder. The output of the unary encoder is generated by concatenating the selected codewords $[\mathbf{y}_i]_{i=1}^{a}$ of Table 4.1, in order to form the b-bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$. For example, the source vector $\mathbf{x} = [1, 1, 5, 1, 1, 2, 3, 1, 1, 6]$ of $a = 10$ symbols yields the $b = 22$-bit vector $\mathbf{y} = [0011110001011000111110]$. Note that the average length of the bit vector $\mathbf{y}$ is given by $(a \cdot l)$.

Following unary encoding, the UEC encoder employs a trellis to encode the bit sequence $\mathbf{y}$, as shown in Figure 4.2. The generalized UEC trellis of Figure 3.4 is parametrized by the codebook $\mathbb{C}$, which comprises $r/2$ number of n-bit codewords, where r is the number of trellis states employed. For example, the codebook $\mathbb{C} = \{1\}$ corresponds to the $r = 2$-state $n = 1$-bit UEC trellis of Figure 4.3, while $\mathbb{C} = \{1, 1, 1\}$ yields the $r = 6$-state $n = 1$-bit UEC trellis of Figure 4.4, respectively.

x_i	$P(x_i)$	$\mathbf{y}_i$	
		Unary	EG
1	0.797	0	1
2	0.117	10	010
3	0.038	110	011
4	0.017	1110	00100
5	0.009	11110	00101
6	0.006	111110	00110
7	0.004	1111110	00111
8	0.003	11111110	0001000
9	0.002	111111110	0001001
10	0.001	1111111110	0001010

Table 4.1: The first ten symbol probabilities for a zeta distribution having the parameter $p_1 = 0.797$, as well as the corresponding unary and EG codewords. By contrast to Table 3.1, the parameter p_1 has a different value in this table.

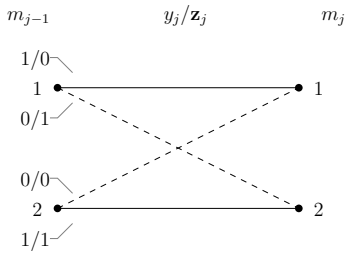

Figure 4.3: An $r = 2$-state $n = 1$-bit UEC trellis, where the codebook $\mathbb{C} = \{1\}$. The codebook is different from that in the trellis of Figure 3.5.

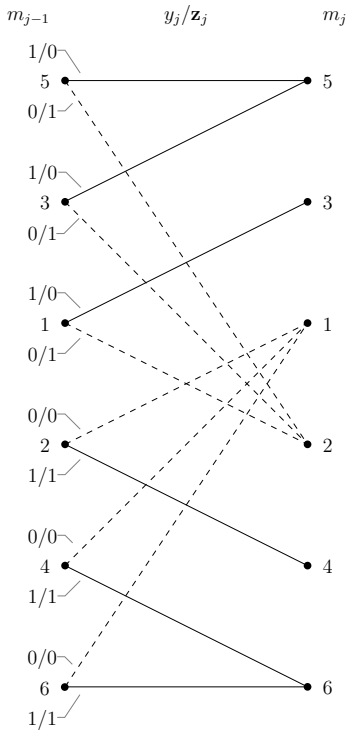

Figure 4.4: An $r = 6$-state $n = 1$-bit UEC trellis, where the codebook $\mathbb{C} = \{1, 1, 1\}$. The codebook is different from that in the trellis of Figure 3.6.

As mentioned in Section 3.3, 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 $m_{j-1} \in \{1, 2, \ldots, r\}$ to its next state $m_j \in \{1, 2, \ldots, r\}$, in order of increasing bit-index j. Therefore, the bit vector $\mathbf{y}$ identifies a path through the trellis, which may be represented by a vector $\mathbf{m} = [m]_{j=0}^{b}$ comprising $(b + 1)$ state values. Note that the example bit vector $\mathbf{y}$ provided above yields the path $\mathbf{m} = [1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1]$ through the $r = 2$-state trellis of Figure 4.3. Similiarly, the same bit vector $\mathbf{y}$ yields the path $\mathbf{m} = [1, 2, 1, 3, 5, 5, 5, 2, 1, 2, 4, 1, 3, 5, 2, 1, 2, 4, 6, 6, 6, 6, 1]$ through the $r = 6$-state trellis of Figure 4.4. Note that the trellis path $\mathbf{m}$ is guaranteed to terminate in the state $m_b = 1$, when the symbol vector $\mathbf{x}$ has an even length a, while $m_b = 2$ is guaranteed, when a is odd [92].

The UEC trellis encoder represents each bit y_j in the vector $\mathbf{y}$ by an n-bit codeword $\mathbf{z}_j$. This is selected from the codebook $\mathbb{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{r/2-1}, \mathbf{c}_{r/2}\}$ or from the complementary codebook $\overline{\mathbb{C}} = \{\overline{\mathbf{c}_1}, \overline{\mathbf{c}_2}, \ldots, \overline{\mathbf{c}_{r/2-1}}, \overline{\mathbf{c}_{r/2}}\}$. Hence, the selected codewords are concatenated to obtain the $(b \cdot n)$-bit vector $\mathbf{z} = [z_k]_{k=1}^{b \cdot n}$ of Figure 4.2. Recall that the $r = 2$- and $r = 6$-state trellises of Figure 4.3 and Figure 4.4 yield the different paths $\mathbf{m}$, when encoding the same example bit vector $\mathbf{y}$ provided above. Despite this however, both result in the same encoded bit vector $\mathbf{z} = [10000010110001011111110]$.

Note that the UEC-encoded bit vector $\mathbf{z}$ will always be identical, regardless of whether the $r = 2$-state UEC trellis of Figure 4.3 or the $r = 6$-state UEC trellis of Figure 4.4 is employed. This is because the $r = 6$ codebook $\mathbb{C} = \{1, 1, 1\}$ may be considered to be an extension of the $r = 2$ codebook $\mathbb{C} = \{1\}$. More specifically, a codebook $\mathbb{C}$ may be extended by appending replicas of the final codeword in the codebook. For example, the $r = 2$ codebook $\mathbb{C} = \{1\}$ can also be extended both to the $r = 4$ codebook $\mathbb{C} = \{1, 1\}$ and to the $r = 8$ codebook $\mathbb{C} = \{1, 1, 1, 1\}$. As a further example, the $r = 6$ codebook $\mathbb{C} = \{01, 11, 11\}$ of Figure 3.6 can be considered to be an extension of the $r = 4$ codebook $\mathbb{C} = \{01, 11\}$ of Figure 3.5. In fact, regardless of how much a codebook $\mathbb{C}$ is extended, the UEC-encoded bit vector $\mathbf{z}$ will be identical, when encoding the same symbol vector $\mathbf{x}$. We will exploit this property in Section 4.3 for dynamically adjusting the number of states employed in the UEC decoder.

The bit vector $\mathbf{z}$ may be modeled as a particular realization of a vector $\mathbf{Z} = [Z_k]_{k=1}^{b \cdot n}$ comprising $b \cdot n$ binary RVs. Each binary RV Z_k adopts the values 0 and 1 with the probabilities $\Pr(Z_k = 0)$ and $\Pr(Z_k = 1)$ respectively, corresponding to a bit entropy of $H_{Z_k} = H[\Pr(Z_k = 0)] + H[\Pr(Z_k = 1)]$. The overall average coding rate R_o of the UEC encoder is given by

$$R_\mathrm{o} = \frac{a \cdot H_X}{\sum_{k=1}^{b \cdot n} H_{Z_k}}. \tag{4.1}$$

where H_X is the symbol entropy of each symbol in vector $\mathbf{x}$. Note that the average coding rate of the UEC encoder depends upon the parameter p_1 of the zeta distribution, as shown in Figure 3.2 of [92]. Furthermore, the UEC trellis is designed to be symmetric and to rely on complementary codewords, so that it produces equiprobable bit values, where $\Pr(Z_k = 0) = \Pr(Z_k = 1) = 0.5$ and giving a bit entropy of $H_{Z_k} = 1$. In this case, Eq. (4.1) reduces to $R_\mathrm{o} = H_X/(l \cdot n)$, which is identical to Equation 3.13. When the source symbols obey a zeta distribution having $p_1 = 0.797$, we obtain a UEC coding rate of $R_\mathrm{o} = 0.762$, as shown in Table 4.3, which is in agreement with Figure 3.7.

As shown in Figure 4.2, the UEC-encoded bit vector $\mathbf{z}$ is then forwarded to the turbo encoder. This encodes the bit vector $\mathbf{z}$ twice, employing the pair of interleavers π_1 and π_2 to make the two encoded data sequences approximately statistically independent of each other. Here, the pair of turbo component encoders are constituted by identical $r = 8$-state URC encoders, which are labeled URC1 and URC2 in Figure 4.2. These employ $(8, \mathrm{F}) = [1001, 1111]$ as the octally represented generator and feedback polynomials respectively, as depicted in the eighth schematic of Figure 2.6. The coding rate R_i of the inner turbo code is given by

$$R_\mathrm{i} = \frac{\sum_{k=1}^{b \cdot n} H_{Z_k}}{2 \cdot b \cdot n}, \tag{4.2}$$

where the coding rate of each individual URC encoder is equal to $2 \cdot R_\mathrm{i}$. In the scenario, where $H_{Z_k} = 1$, we obtain $R_\mathrm{i} = 1/2$, as shown in Table 4.3 and as discussed above.

After multiplexing the two resultant encoded bit sequences, the channel interleaver π_3 is employed in Figure 4.2 for the sake of dispersing burst errors. Following this, Gray-coded $M = 4$-ary Quadrature Phase-Shift Keying (QPSK) modulation may be employed for transmitting the resultant bit vector $\mathbf{w}$, as shown in Figure 4.2. The effective throughput of the UEC-Turbo scheme is given by $\eta = R_\mathrm{o} \cdot R_\mathrm{i} \cdot \log_2(M)$, which has the value of $\eta = 0.762$ information bits/symbol, when the source symbols obey a zeta distribution having $p_1 = 0.797$, as shown in Table 4.3. Alternatively, a mapping scheme other than Gray coding or a modulation scheme having a higher order M can be employed, although this may require a higher complexity receiver design [92, 96].

4.2.2 Receiver

In the receiver of Figure 4.2, QPSK demodulation is employed in order to obtain the LLR vector $\tilde{\mathbf{w}}$. This is deinterleaved π_1^{-3} and demultiplexed, before iterative decoding commences by exchanging extrinsic information among the URC1, URC2 and UEC decoders of Figure 4.2. Note that higher-order modulation schemes may be readily employed, although this would require the iterative exchange of extrinsic information between the demodulator and the URC decoders, in order to avoid capacity loss [145]. This four-stage

concatenation scheme is considered in our work of [97], using the proposed adaptive iterative decoding techniques in this chapter.

In the three-state concatenation of Figure 4.2, the interleavers π_1, π_1^{-1}, π_2 and π_2^{-2} are employed for mitigating the correlation within the *a priori* LLR vectors of $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$, $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}$ and $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}}$. All three decoders apply the Log-BCJR algorithm [91], which has a complexity directly dependent on the number of states employed. We assume perfect synchronization between the UEC trellis and the unary codewords during the Log-BCJR algorithm's γ_t calculation of Equation (9) [91], as introduced in Section 2.11. This employs the conditional transition probability $\Pr(M_j = m|M_{j-1} = m')$ of Equation 3.9. During Log-BCJR decoding, the UEC trellis should emerge from $m_0 = 1$ and be terminated either at $m_b = 1$ or $m_b = 2$, depending on whether the length a of the symbol vector $\mathbf{x}$ is even or odd, respectively. Note that since extending a UEC trellis does not change the UEC-encoded bit vector $\mathbf{z}$, the UEC decoder may decode it using an extended, larger-complexity version of the specific UEC trellis employed by the UEC encoder. As explained in Chapter 3, increasing the number of states r employed by the UEC trellis decoder in this way has the benefit of improving its error correction capability, at the cost of increasing its complexity [92, 96]. In Section 4.3, we will exploit this to dynamically adjust r for the sake of striking an attractive the trade-off between its decoding complexity and error correction capability.

Observe in Figure 4.2 that the *a priori* LLR vectors provided for each of the three decoders are obtained as the sum of the extrinsic LLR vectors most recently generated by the other two decoders, namely we have $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}} = \tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$, $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}} = \tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$ and $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}} = \tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$. These LLR vectors comprise b number of LLRs, which pertain to the corresponding bits of $\mathbf{z}$. At the start of the iterative decoding process, the extrinsic LLR vectors $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}$, $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$ and $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$ are initialized with zero-valued LLRs.

During the iterative decoding process, the iterative operation of the URC1, URC2 and UEC decoders of Figure 4.2 may be performed using a wide variety of different decoder activation orderings. For the sake of conceptional simplicity, a fixed decoder activation order may be employed, in which the URC1, URC2 and UEC decoders of Figure 4.2 are activated using a regular activation order repeated periodically. For example, the decoders may be consecutively operated in turn, according to {URC1, URC2, UEC; URC1, URC2, UEC; ...}, where each ordered consecutive operation of the URC1, URC2 and UEC decoders represents a full system iteration of the decoding process. Alternatively, we may employ a non-periodic decoder activation order, in which an on-line decision is made at each stage of the iterative decoding process, in order to adaptively and dynamically select which of the URC1, URC2 and UEC decoders of Figure 4.2 to activate next. This is

exploited in Section 4.3 using a novel 3D EXIT chart aided technique, in order to expedite iterative decoding convergence towards an approximation of the ML error correction performance.

Following the achievement of iterative decoding convergence, the UEC trellis decoder of Figure 4.2 may invoke the Log-BCJR algorithm for generating the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}^{\mathrm{p}}$ that pertain to the corresponding unary-encoded bits in the vector $\mathbf{y}$. Then, the unary decoder sorts the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}^{\mathrm{p}}$, in order to identify the a number of bits in the vector $\mathbf{y}$ that are most likely to have values of zero. A hard decision bit vector $\hat{\mathbf{y}}$ is then obtained by setting the value of these bits to zero and the value of all other bits to one. Finally, the bit vector $\hat{\mathbf{y}}$ can be unary decoded in order to obtain the symbol vector $\hat{\mathbf{x}}$, which is guaranteed to comprise a number of symbols.

4.3 Adaptive Iterative Decoding

In this section, we propose our novel adaptive iterative decoding technique for the UEC-Turbo scheme of Figure 4.2. As described in Section 4.1, this evaluates the benefit and cost associated with activating each of the URC1, URC2 and UEC trellis decoders of Figure 4.2 at each stage of the iterative decoding process, in order to decide as to which decoder to activate next. In the case of the UEC trellis decoder, the number of states r to employ is also considered, as described in Section 4.2.

In Section 4.3.1, we employ 3D EXIT chart analysis [129, 134] in order to quantify the error correction benefit [155] that is offered by each decoder, if activated for the next decoding operation. Meanwhile, the computational complexity cost of each option is quantified in terms of the number of the associated ACS arithmetic operations [96] in Section 4.3.3. Hence, by jointly considering the benefit and cost, we can dynamically adapt the iterative decoding process of the UEC-Turbo scheme of Figure 4.2, as described in Section 4.3.2. Finally, Section 4.3.3 shows that the storage requirements of the proposed technique are modest, compared to those of the interleavers of Figure 4.2.

4.3.1 EXIT Chart Analysis

In this section, we analyze the iterative decoding convergence of the UEC-Turbo scheme introduced in Figure 4.2, which serially concatenates the UEC code [92] with a turbo code. In Section 4.3.1.1, we consider the 2D EXIT curves of the URC1, URC2 and UEC trellis decoders of Figure 4.2 separately. Then Section 4.3.1.2 shows that the three 2D EXIT curves can be converted into 3D surfaces and plotted in the same 3D EXIT chart, allowing the explicit visualization of the iterative decoding trajectory, when employing any arbitrary decoder activation order. Furthermore, the 3D EXIT chart may be employed

for quantifying the benefit, offered by each decoder in terms of the achievable Mutual Information (MI) improvement. Finally, Section 4.3.1.3 shows that the 3D EXIT chart may be projected into two dimensions, in order to demonstrate that the proposed UEC-Turbo scheme facilitates near-capacity operation.

4.3.1.1 2D EXIT Curves

As shown in Figure 4.2, the URC1 decoder generates the extrinsic LLR vector $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$ with the aid of the *a priori* LLR vector $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}$, which is combined with the LLRs received over the channel. Therefore, the MI $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z})$ of $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$ is related to the MI $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z})$ of $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}$, as well as to the MI $I(\tilde{\mathbf{w}};\mathbf{w})$ of the LLRs $\tilde{\mathbf{w}}$ provided by the demodulator, which depends on the channel's Signal to Noise Ratio (SNR) per bit $E_{\mathrm{b}}/N_0 = \mathrm{SNR}/\eta$.

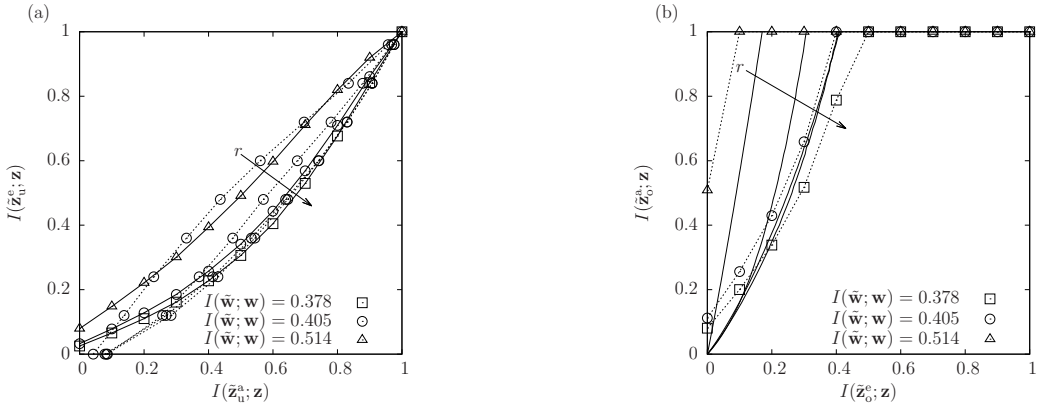

Figure 4.5: 2D EXIT charts for the UEC-Turbo scheme of Figure 4.2, for UEC trellises having various numbers of states $r \in \{2, 4, 6, 8\}$. These EXIT charts are projected in terms of (a) the URC1 decoder and (b) the UEC trellis decoder. In (a), the solid lines represent the URC1 EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}) = f_{\mathrm{u}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})]$ for a variety of MI values $I(\tilde{\mathbf{w}};\mathbf{w})$. Meanwhile the dotted lines represent the projected UEC-URC2 EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}) = f_{\mathrm{o,l}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w}), r]$, for the case where $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.405$. In (b), the solid lines denote the inverted UEC EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}};\mathbf{z}), r]$, while the dotted lines denote the projected URC1-URC2 EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}};\mathbf{z}) = f_{\mathrm{u,l}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})]$. Here, MI values of $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.378, 0.405$ and 0.514 correspond to uncorrelated narrowband Rayleigh fading channel E_{b}/N_0 values of 0.8 dB, 1.3 dB and 3.3 dB, respectively. Different random designs are employed for the interleavers π_1, π_2 and π_3 in each simulated frame.

As detailed in [99], this relationship may be visualized using the 2D EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}) = f_{\mathrm{u}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})]$ of the URC1 decoder, which is plotted for various $I(\tilde{\mathbf{w}};\mathbf{w})$ values in the 2D EXIT chart of Figure 4.5(a). Note that because the URC1 decoder is recursive [156], and hence has an infinite impulse response, it may be referred to as Maximal Mutual Information Achieving (MMIA) [129]. As a result of this, the URC1 EXIT function reaches the $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}) = I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}) = 1$ point of pefect decoding convergence associated with the top-right corner of the EXIT chart of Figure 4.5(a), where

low decoding error rates are facilitated [156]. More explicitly, a low decoding error rate is achieved, because in the presence of perfect *a priori* information, perfect extrinsic information is generated, again, as represented by reaching the $(1, 1)$ point of Figure 4.5(a). Likewise, since URC2 of Figure 4.2 is identical to URC1, its EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{l}}[I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})]$ is identical to that of URC1, therefore also reaching the $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}; \mathbf{z}) = I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}) = 1$ point.

Similarly, the UEC decoder's transformation of the *a priori* LLR vector $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ into the extrinsic LLR vector $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}$ may be characterized by the UEC EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}), r]$. This EXIT function is shown inverted - i.e. with the abscissa and ordinate axes swapped - in the 2D EXIT chart of Figure 4.5(b) for the cases of employing the UEC codebooks $\mathbb{C} = \{1\}$, $\mathbb{C} = \{1, 1\}$, $\mathbb{C} = \{1, 1, 1\}$ and $\mathbb{C} = \{1, 1, 1, 1\}$, which correspond to $r = 2$, 4, 6 and 8 UEC trellis states, respectively. Note that when employing these $n = 1$-bit UEC codebooks, a free-distance of $d_{\mathrm{free}} = 1$ results for the UEC-encoded bit vector $\mathbf{z}$. Note that because we have $d_{\mathrm{free}} < 2$, these UEC codebooks are not MMIA [150]. Owing to this distance-limitation, the inverted UEC EXIT functions of Figure 4.5(b) do not reach the $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}) = I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = 1$ point, where low decoding error rates are facilitated. Nevertheless, this does not prevent iterative decoding convergence towards a low decoding error rate, since the presence of the two MMIA URC1 and URC2 decoders is sufficient for achieving this [129]. Note that inverted EXIT curves corresponding to a selection of $n = 2$-bit UEC codebooks can be seen in Figure 3.8.

4.3.1.2 3D EXIT Chart

Observe in Figure 4.2 that the receiver employs an iterative exchange of LLRs that pertain to the bit sequence $\mathbf{z}$ among three decoding components, i.e., the URC1 decoder, the URC2 decoder and the UEC's trellis decoder of Figure 4.2. The *a priori* LLR sequence $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ is obtained as a sum of the extrinsic LLR sequences generated by the other two components $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}} = \tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$. Hence, the inverted EXIT function of the UEC component, which is denoted by $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}), r]$ may be expressed as a 3D function of two arguments, i.e. as $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}), r]$. Note that for the scenario, where the extrinsic LLRs in the sequences $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$ and $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$ are Gaussian distributed, the MI of the *a priori* LLR sequence $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ can be obtained analytically, according to [129, 149]

$$I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}) = J\left(\sqrt{J^{-1}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z})]^2 + J^{-1}[I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z})]^2}\right), \tag{4.3}$$

where

$$J(\sigma) = 1 - \int_{-\infty}^{\infty} \frac{e^{-(\xi - \sigma^2/2)^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}} \cdot \log_2[1 + e^{-\xi}] d\xi$$

and $\sigma = J^{-1}(I)$ is the inverse function. In practice, the approximations of [149] may be used for these functions. This approach allows the 3D EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}), r]$ to be obtained from the 2D EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}), r]$, without requiring time-consuming Monte-Carlo simulations.

In a similar manner, the URC1 component's EXIT function of $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{u}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})]$ seen in Figure 4.5(a) may be expressed as $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{u}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})]$. Note that since the URC1 and URC2 components of the turbo code shown in Figure 4.2 are identical, we may directly obtain the URC2 component's 3D function $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{l}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}),\ I(\tilde{\mathbf{w}}; \mathbf{w})]$, where we have $f_{\mathrm{u}} = f_{\mathrm{l}}$. Since all three of the 3D EXIT functions $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z})$, $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z})$ and $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z})$ have the same arguments, they can all be plotted in a single 3D EXIT chart having axes labelled with these three MIs. This approach is shown in Figure 4.6 for our UEC-Turbo scheme of Figure 4.2, when communicating over uncorrelated narrowband Rayleigh fading channels having a range of $I(\tilde{\mathbf{w}}; \mathbf{w})$ values. Here, the complex channel gain has a mean of 0, a variance of 1 and a coherence time of 0, such that each fading coefficient affects one transmitted symbol. Note that we employ this channel model, since it is representative of various wireless channels. More specifically, the presence of the interleaver π_3 means that the EXIT functions of URC1 and URC2 remain unaffected by the coherence time of the Rayleigh fading channel, provided that it is short compared to the transmit duration of each frame. Owing to this, our results apply not only to uncorrelated Rayleigh fading channels, but also to fast fading channels.

For example, as shown in Figure 4.6(b), the surfaces $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{u}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})]$ and $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{l}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})]$, where $I(\tilde{\mathbf{w}}; \mathbf{w}) = 0.405$, represent the extrinsic information produced by the URC1 and URC2 components of the turbo decoder for $E_{\mathrm{b}}/N_0 = 1.3$ dB. Note that the intersection between these two surfaces and the plane $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = 0$ gives the 2D EXIT chart for the turbo decoder alone. This 2D EXIT chart in Figure 4.5(a) shows that without the aid of the UEC trellis decoder, the decoding trajectory would converge at about $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}) = I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}) = 0.08$. By contrast, when the $r = 6$-state UEC trellis decoder is introduced, an open 3D tunnel can be created, as shown in Figure 4.6(b). This allows the iterative decoding trajectory to reach the extrinsic MIs of $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}) = I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}) = 1$, where low decoding error rates are facilitated, as described in Section 4.3.1.1. Note that the trajectories shown in Figure 4.6 rely on the fixed, periodic decoder activation order of {URC1, URC2, UEC; URC1, URC2, UEC; ...}, as described in Section 4.2.2. However, the 3D EXIT chart in Section 4.3.2 will be used for comparing the quantitative potential benefits associated with activating each decoder at each stage of the iterative decoding process, in order to dynamically adapt the decoder activation order for expediting the attainable convergence.

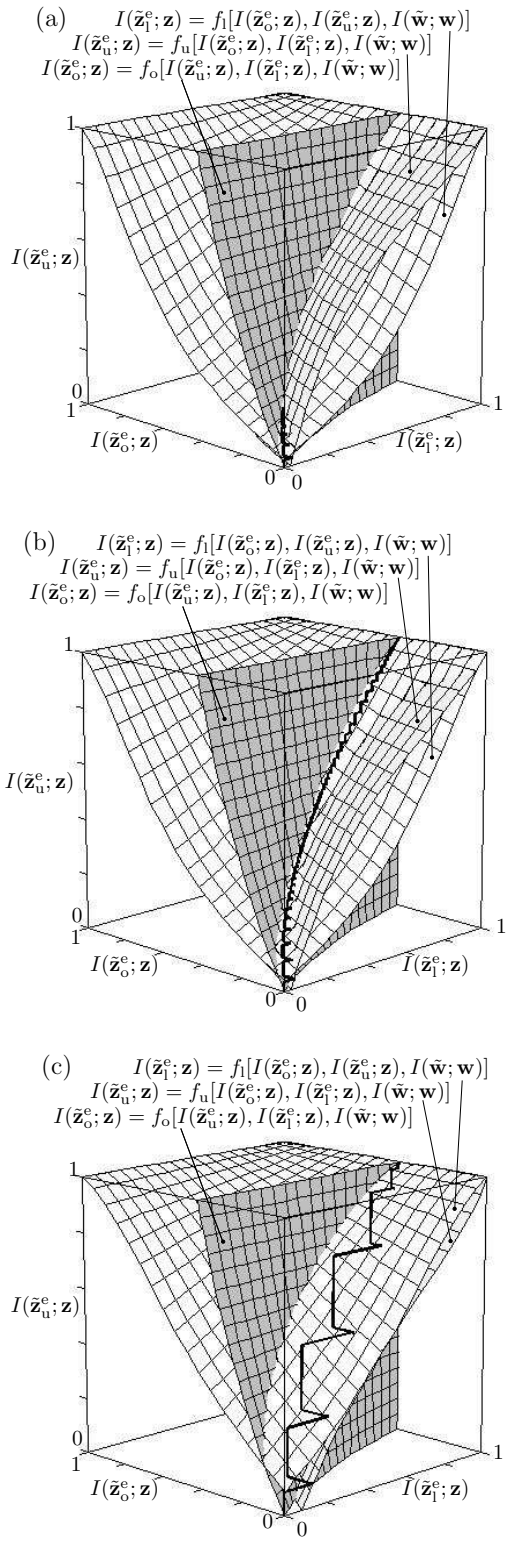

Figure 4.6: 3D EXIT surfaces for the UEC-Turbo scheme of Figure 4.2 when employing $r = 6$ states in the UEC trellis decoder, for communicating over an uncorrelated narrowband Rayleigh fading channel, having MI values of (a) $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.378$, (b) 0.405 and (c) 0.514, which correspond to uncorrelated Rayleigh fading channel E_{b}/N_0 values of 0.8 dB, 1.3 dB and 3.3 dB, respectively. In each case, a bold line is used to indicate the iterative decoding trajectory, when employing the fixed decoder activation order {URC1, URC2, UEC; URC1, URC2, UEC; ...}. Note that these iterative decoding trajectories reside *below* the URC1 EXIT surface f_{u} and *above* the two other EXIT surfaces f_{o} and f_{l}. Different random designs are employed for the interleavers π_1, π_2 and π_3 in each simulated frame.

As shown in Figure 4.6(c), a wider EXIT chart tunnel is created at $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.514$, requiring fewer activations of the UEC, URC1 and URC2 decoders in order to achieve a low decoding error rate. By contrast, the EXIT chart tunnel is closed for $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.378$, preventing the achievement of a low decoder error rate, as shown in Figure 4.6(a). In Section 4.3.1.3, our 3D EXIT charts are projected into two dimensions, in order to characterize the effect of the $I(\tilde{\mathbf{w}};\mathbf{w})$ value upon the iterative decoding convergence of the proposed UEC-Turbo scheme, as well as to evaluate its capacity-achieving capability.

4.3.1.3 2D EXIT Chart Projections

The 3D EXIT surfaces of Figure 4.6 can be projected into 2D [129, 147, 157], which were plotted in the 2D EXIT charts of Figure 4.5. More explicitly, Figure 4.5(a) complements the URC1 component's EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}) = f_{\mathrm{u}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})]$ with the EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}) = f_{\mathrm{o,l}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w}), r]$ where $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.405$, which represents a 2D projection of the 3D UEC and URC2 EXIT surfaces of Figure 4.6. This EXIT function characterizes the process in which the LLR vector $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$ is fed into the UEC and URC2 decoders, where these components are iteratively operated until convergence is achieved, and then they generate the LLR vector $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}$, as shown in Figure 4.2. As a result, Figure 4.5(a) characterizes a decoder activation order of {URC1, URC2, UEC, URC2, UEC, ..., URC2, UEC; URC1, URC2, UEC, URC2, UEC, ..., URC2, UEC; ...}. Note that the 2D projection of the 3D UEC and URC1 EXIT surfaces is identical to that of the 3D UEC and URC2 EXIT surfaces, owing to the symmetry of URC1 and URC2. Figure 4.5(a) shows that an open EXIT chart tunnel is created at $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.405$, when $r = 6$ states are employed in the UEC trellis decoder, which is in agreement with the 3D EXIT chart of Figure 4.6(b). This facilitates iterative decoding convergence to the $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}) = I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}) = 1$ point at the top-right corner of the EXIT chart of Figure 4.5(a), where a low decoding error rate is achieved.

Similarly, the inverted UEC EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}};\mathbf{z}), r]$ of Figure 4.5(b) can be complemented with the EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}};\mathbf{z}) = f_{\mathrm{u,l}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})]$, which represents a 2D projection of the 3D URC1 and URC2 EXIT surfaces of Figure 4.6. This EXIT function characterizes the process in which the LLR vector $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}$ is forwarded to the URC1 and URC2 components of Figure 4.2, where these components are iteratively operated until convergence is achieved, and then they generate the LLR vector $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$, as shown in Figure 4.2. As a result, Figure 4.5(b) characterizes a component activation order of {URC1, URC2, URC1, URC2, ..., URC1, URC2, UEC; URC1, URC2, URC1, URC2, ..., URC1, URC2, UEC; ...}. Note that in agreement with the 3D EXIT chart of Figure 4.6(b), an open EXIT chart tunnel is created in Figure 4.5(b) at $I(\tilde{\mathbf{w}};\mathbf{w}) = 0.405$, when

$r = 6$ states are employed in the UEC trellis decoder. This open tunnel facilitates iterative decoding convergence to the top edge of the EXIT chart, where we have $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}) = 1$.

The area properties of 2D EXIT charts[1] [132] can be employed to determine whether an iteratively decoded scheme suffers from any capacity loss, which prevents its near-capacity operation. As discussed in Section 4.2.1 and shown in Table 4.3, the effective throughput of the proposed UEC-Turbo scheme is $\eta = 0.762$ information bits/symbol. When communicating over an uncorrelated narrowband Rayleigh fading channel, the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity C becomes equal to η, when the E_{b}/N_0 value is equal to the *capacity bound* of 0.84 dB, as shown in Table 4.3. This implies that reliable communication is possible for E_{b}/N_0 values above 0.84 dB, provided that an open tunnel can be created in the EXIT chart of Figure 4.5(b).

However, in order to facilitate an open EXIT chart tunnel, it is necessary, but not sufficient, for the area A_{o} beneath the inverted UEC EXIT function $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}) = f_{\mathrm{o}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}), r]$ to exceed the area A_{i} beneath the 2D projection of the URC1 and URC2 EXIT surfaces $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}) = f_{\mathrm{u,l}}[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})]$ [132], as discussed in Section 2.13. For the cases where the proposed UEC-Turbo scheme of Figure 4.2 employs $r \in \{2, 4, 6, 8\}$ UEC trellis states, Table 4.3 quantifies the *area bound* as the E_{b}/N_0 value at which $A_{\mathrm{o}} = A_{\mathrm{i}}$. These area bounds represent the lowest E_{b}/N_0 values, where it would be possible to achieve a low decoding error rate, provided that the EXIT functions matched each other sufficiently well.

Note that the area bound exceeds the capacity bound for all cases considered in Table 4.3, where the discrepancy represents *capacity loss*. However, this capacity loss can be seen to approach zero as the number of UEC trellis states r is increased. This is because the expression for A_{o} of Eq. (3.16) in Section 3.5.1 asymptotically approaches the UEC coding rate R_{o} as the number of trellis states r is increased [92], where $A_{\mathrm{o}} = R_{\mathrm{o}}$ is a necessary condition for avoiding capacity loss [131]. Furthermore, the other necessary condition $A_{\mathrm{i}} = C/[R_{\mathrm{i}} \cdot \log_2(M)]$ is also satisfied, since the URC1 and URC2 codes of Figure 4.2 each have an individual coding rate of $2R_{\mathrm{i}} = 1$ [131].

In practice, the EXIT functions of Figure 4.5(b) do not match each other perfectly and hence no an open tunnel emerges, until the E_{b}/N_0 value exceeds the *tunnel bound*. As shown in Figure 4.5(b) and confirmed in Figure 4.5(a) and Figure 4.6(b), the proposed UEC-Turbo scheme has a tunnel bound of $E_{\mathrm{b}}/N_0 = 1.3$ dB, when employing $r = 6$ UEC trellis states. The tunnel bounds for $r \in \{2, 4, 6, 8\}$ are summarised in Table 4.3.

[1] It was shown by Land [130] that the size of the open area between a pair of components exchanging extrinsic information is commensurate with the associated capacity loss.

4.3.2 Dynamic Adjustment of the Decoder Activation Order

In this section, we propose a novel adaptive iterative decoding technique for dynamically adjusting the decoder activation order in the UEC-Turbo scheme of Figure 4.2. In contrast to the fixed decoder activation orders that have been discussed in Sections 4.2.2, 4.3.1.2 and 4.3.1.3, these dynamic decoder activation orders can vary between frames. More specifically, following the activation of a URC decoder, our novel adaptive iterative decoding technique can either activate the other URC decoder, or it can activate the UEC trellis decoder and select the number $r \in \{2, 4, 6, 8\}$ of trellis states to employ. Similarly, following the activation of the UEC trellis decoder, our novel adaptive iterative decoding technique can then activate either the URC1 or the URC2 decoder. As we will show in Section 4.4, this expedites iterative decoding convergence, facilitating near-capacity operation, with reduced computational complexity. We employ the EXIT charts of Section 4.3.1 to quantify the benefit associated with activating the UEC, URC1 and URC2 decoders at each stage of the iterative decoding process, as well as the computational complexity of Section 4.3.3 to quantify the corresponding cost.

At each stage of the iterative decoding process, our proposed technique estimates the quality of the LLR vectors $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}$, $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$ and $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$ that were obtained after the most recent activation of the respective decoders. More specifically, the averaging method of measuring MI [158] may be employed to estimate $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z})$, $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z})$ and $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z})$, without requiring knowledge of the bit values in the vector $\mathbf{z}$. Likewise, the averaging method may be employed to estimate the MI $I(\tilde{\mathbf{w}}; \mathbf{w})$ of the LLRs $\tilde{\mathbf{w}}$ provided by the QPSK demodulator. Following this, the 3D EXIT charts of Section 4.3.1.2 may be employed to predict the extrinsic MI that is offered by activating each decoder. Then, the corresponding MI improvements can be obtained as the discrepancies between these predicted MIs and the current MIs, in order to quantify the benefit associated with activating each decoder. For example, the benefit associated with activating the UEC trellis decoder using r states may be quantified by the MI improvement of

$$\Delta I_{\mathrm{o}}^{r} = f_{\mathrm{o}}\left[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}), r\right] - I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}). \tag{4.4}$$

Similarly, the benefits associated with the URC1 and URC2 decoders may be quantified as

$$\Delta I_{\mathrm{u}} = f_{\mathrm{u}}\left[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})\right] - I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}), \tag{4.5}$$

$$\Delta I_{\mathrm{l}} = f_{\mathrm{l}}\left[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})\right] - I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}), \tag{4.6}$$

respectively.

As discussed in Section 4.3.1.3, when the number of states r employed by the UEC

trellis decoder is increased, a MI improvement $\Delta I_{\rm o}^{r}$ may be expected. Therefore, our novel adaptive iterative decoding technique also considers the computational complexity cost $C_{\rm o}^{r}$, $C_{\rm u}$ and $C_{\rm l}$ of the UEC decoder having $r \in \{2, 4, 6, 8\}$ states, as well as the 8-state URC1 and URC2 decoders, respectively. As discussed in Section 4.3.3, Table 4.2 states the number of ACS operations required by each decoder. Note that while our adaptive technique makes *on-line* decisions about which decoder to activate at each stage of the iterative decoding process, this is performed using the complexity figures of Table 4.2, which were calculated in an *off-line* fashion. These complexity figures remain constant throughout each decoding iteration and for each transmitted frame. For example, when employing the $r = 6$-state $n = 1$-bit trellis shown in Figure 4.4, the complexity cost is $\mathrm{C}_{\rm o}^{6} = 166$ ACS operations per bit of $\mathbf{z}$.

After each stage of the iterative decoding process, the benefit-to-cost ratios $\Delta I_{\rm o}^{r}/C_{\rm o}^{r}$, $\Delta I_{\rm u}/C_{\rm u}$ and $\Delta I_{\rm l}/C_{\rm l}$ are calculated for the UEC trellis decoder having $r \in \{2, 4, 6, 8\}$ states, as well as for the URC1 and URC2 decoders, respectively. Then, the particular decoder offering the highest benefit-to-cost ratio is selected for activation in the next stage of the iterative decoding process.

As shown in Algorithm 4.1, the adaptive iterative decoding process continues, until the accumulated complexity C of the activated components reaches the pre-defined complexity limit $C_{\rm limit} = 273$ ACS operations. Note that these 273 ACS operations are reserved, so that even if the component having the highest complexity is selected in the final stage of the iterative decoding process, there will be 49 ACS operations remaining. This then allows the $r = 2$ UEC trellis decoder to obtain the *a posteriori* LLR vector $\tilde{\mathbf{y}}^{\rm p}$, as shown in Table 4.2. However, if more ACS operations are available following the final stage of iterative decoding, then a UEC trellis having a higher number of states r can be selected for obtaining $\tilde{\mathbf{y}}^{\rm p}$ at a reduced probability of error. Furthermore, the iterative decoding process will be terminated early, if the *a posteriori* MI $I(\tilde{\mathbf{z}}_{\rm o}^{\rm e} + \tilde{\mathbf{z}}_{\rm u}^{\rm e} + \tilde{\mathbf{z}}_{\rm l}^{\rm e}; \mathbf{z})$ reaches 0.999, which implies that error free decoding can be achieved without operating any further component decoders.

4.3.3 Complexity and Storage Analysis

In this section, we quantify the complexity of activating the UEC trellis decoder when employing r number of trellis states, in addition to quantifying the complexity associated with activating the URC1 and URC2 decoders of Figure 4.2. Moreover, we consider the storage requirements of the adaptive iterative decoding technique introduced in Section 4.3.2.

The complexity of each component decoder was quantified in the context of an iterative decoding scheme relying on the number of trellis transitions and states in their respective

Algorithm 4.1: Adaptive iterative decoding algorithm

$\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}} \leftarrow \mathbf{0}$, $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}} \leftarrow \mathbf{0}$, $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}} \leftarrow \mathbf{0}$, $C \leftarrow 0$.
while $C \leq C_{\mathrm{limit}} - 273$ **and** $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z}) < 0.999$ **do**
    $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}} \leftarrow \tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$, $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}} \leftarrow \tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$ and $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}} \leftarrow \tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$.
    Measure MI: $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z})$, $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}; \mathbf{z})$ and $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}}; \mathbf{z})$.
    for $r \in \{2, 4, 6, 8\}$ **do**
        $\Delta I_{\mathrm{o}}^{r} \leftarrow f_{\mathrm{o}}\left[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}; \mathbf{z}), r\right] - I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}; \mathbf{z})$. {}Predict MI improvement
    end for
    $\Delta I_{\mathrm{u}} \leftarrow f_{\mathrm{u}}\left[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})\right] - I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}; \mathbf{z})$.
    $\Delta I_{\mathrm{l}} \leftarrow f_{\mathrm{l}}\left[I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}}; \mathbf{z}), I(\tilde{\mathbf{w}}; \mathbf{w})\right] - I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}; \mathbf{z})$.
    $best \leftarrow max(\Delta I_{\mathrm{o}}^{r}/C_{\mathrm{o}}^{r}, \Delta I_{\mathrm{u}}/C_{\mathrm{u}}, \Delta I_{\mathrm{l}}/C_{\mathrm{l}}), r \in \{2, 4, 6, 8\}$.
    for $r \in \{2, 4, 6, 8\}$ **do**
        if $\Delta I_{\mathrm{o}}^{r}/C_{\mathrm{o}}^{r} = best$ **then**
            Activate r-state UEC trellis decoder to convert $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ into $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}$,
            $C \leftarrow C + C_{\mathrm{o}}^{r}$.
        end if
    end for
    if $\Delta I_{\mathrm{u}}/C_{\mathrm{u}} = best$ **then**
        Activate URC1 decoder to convert $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}}$ into $\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}}$,
        $C \leftarrow C + C_{\mathrm{u}}$.
    end if
    if $\Delta I_{\mathrm{l}}/C_{\mathrm{l}} = best$ **then**
        Activate URC2 decoder to convert $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}}$ into $\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$,
        $C \leftarrow C + C_{\mathrm{l}}$.
    end if
end while
$\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}} \leftarrow \tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}} + \tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}}$
if $C \leq C_{\mathrm{limit}} - 223$ **then**
    Activate $r = 8$-state UEC trellis decoder to convert $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ into $\tilde{\mathbf{y}}^{\mathrm{p}}$,
else if $C \leq C_{\mathrm{limit}} - 165$ **then**
    Activate $r = 6$-state UEC trellis decoder to convert $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ into $\tilde{\mathbf{y}}^{\mathrm{p}}$,
else if $C \leq C_{\mathrm{limit}} - 107$ **then**
    Activate $r = 4$-state UEC trellis decoder to convert $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ into $\tilde{\mathbf{y}}^{\mathrm{p}}$,
else
    Activate $r = 2$-state UEC trellis decoder to convert $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}}$ into $\tilde{\mathbf{y}}^{\mathrm{p}}$.
end if

trellises [159]. This determines the number of ACS arithmetic operations. As discussed in Section 3.8, our approach is to decompose each of the Log-BCJR calculations detailed in Section 2.11 into their constituent addition and $\max^*$ operations, allowing a fair comparison between the complexities of the components having different types of operations. Again, we employ the Look-Up-Table (LUT) based technique of [152] to convert the $\max^*$ operation into five ACS arithmetic operations, as described in Section 2.11.1. Hence, we know that the complexity of each type of decoder scales linearly with the number of bits $b \cdot n$ in the encoded vector $\mathbf{z}$ of Figure 4.2. Table 4.2 lists the number of $\max^*$ and addition operations that are performed per bit of $\mathbf{z}$, when each type of decoder employs a trellis having r states. Here, the computational complexity of the UEC trellis decoder depends on whether it is used for generating the extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ or for the *a posteriori*

Decoder	r	$\max^*$	add	ACS
$n = 1$-bit Trellis BCJR decoder $\tilde{\mathbf{y}}^{\mathrm{p}}$	2	6	19	49
	4	14	37	107
	6	22	55	165
	8	30	73	223
	10	38	91	281
$n = 1$-bit Trellis BCJR decoder $\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}}$	2	6	20	50
	4	14	38	108
	6	22	56	166
	8	30	74	224
	10	38	92	282
URC BCJR decoder	2	6	19	49
	8	30	73	223
URC Viterbi decoder	2	2	8	18

Table 4.2: Number of addition and $\max^*$ operations that are performed per bit of $\mathbf{z}$, for the various types of decoders of Figure 4.2 employing trellises having r states.

LLR vector $\tilde{\mathbf{y}}^{\mathrm{p}}$. Note that the results of Table 4.2 complement those listed in Table 3.5 for $n = 2$-bit codes.

Now we consider the storage requirements of the novel adaptive iterative decoding technique introduced in Section 4.3.2. In order to quantify the benefit associated with activating each decoder, in comparison to that of the others, the value of the functions $f_{\mathrm{o}}\left[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}};\mathbf{z}), r\right]$, $f_{\mathrm{u}}\left[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})\right]$ and $f_{\mathrm{l}}\left[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})\right]$ is required. Rather than storing these 3D EXIT functions, the associated storage requirements can be reduced by instead storing the 2D EXIT functions $f_{\mathrm{o}}\left[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}};\mathbf{z}), r\right]$, $f_{\mathrm{u}}[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})]$ and $f_{\mathrm{l}}\left[I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})\right]$. In this way, Eq. (4.4) – Eq. (4.6) may be reformulated as

$$\Delta I_{\mathrm{o}}^{r} = f_{\mathrm{o}}\left[I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}};\mathbf{z}), r\right] - I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z}), \tag{4.7}$$

$$\Delta I_{\mathrm{u}} = f_{\mathrm{u}}\left[I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})\right] - I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z}), \tag{4.8}$$

$$\Delta I_{\mathrm{l}} = f_{\mathrm{l}}\left[I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}};\mathbf{z}), I(\tilde{\mathbf{w}};\mathbf{w})\right] - I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}};\mathbf{z}), \tag{4.9}$$

where $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{a}};\mathbf{z})$, $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{a}};\mathbf{z})$ and $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{a}};\mathbf{z})$ may be obtained using Eq. (4.3).

Furthermore, an infinite amount of storage would be required for storing the values of the 2D EXIT functions for all possible values of $I(\tilde{\mathbf{z}}_{\mathrm{o}}^{\mathrm{e}};\mathbf{z})$, $I(\tilde{\mathbf{z}}_{\mathrm{u}}^{\mathrm{e}};\mathbf{z})$, $I(\tilde{\mathbf{z}}_{\mathrm{l}}^{\mathrm{e}};\mathbf{z})$ and $I(\tilde{\mathbf{w}};\mathbf{w})$. Instead, we recommend storing the 2D EXIT functions for only a limited set of carefully selected quantised values of these parameters. During the adaptive iterative decoding process, the values of the 2D EXIT functions may be approximated by interpolating between the values stored.

In particular, based on the observations from Figures 3.14, 3.15 and 3.16, we recommend storing the values of the 2D UEC EXIT function for only the 4 values of r in the set $\{2, 4, 6, 8\}$, since having $r \geq 10$ states offers only a marginal additional MI improvement compared to $r = 8$, at the cost of a significantly higher computational complexity. Additionally, it is only necessary to store one set of 2D URC EXIT functions, since the URC1 and URC2 decoders are identical. We recommend storing the values of the 2D URC EXIT function for only the 24 $I(\tilde{\mathbf{w}}; \mathbf{w})$ values in the set of $\{$0.337, 0.362, 0.389, 0.415, 0.443, 0.456, 0.465 0.476, 0.487, 0.498, 0.510, 0.521, 0.532, 0.543, 0.554, 0.565, 0.576, 0.587, 0.598, 0.609, 0.635, 0.661, 0.686, 0.710$\}$, which correspond to the E_{b}/N_0 values of $\{0.0, 0.5, 1.0, 1.5\}$, $\{2.0, 2.2, \ldots, 4.8, 5.0\}$ and $\{5.5, 6.0, 6.5, 7.0\}$. This is motivated by the observation that at very low $I(\tilde{\mathbf{w}}; \mathbf{w})$ values, iterative decoding convergence is impossible and hence any efforts to optimise the process are futile. Furthermore, at very high $I(\tilde{\mathbf{w}}; \mathbf{w})$ values, iterative decoding is unnecessary, since error free decoding can typically be achieved by activating each of the UEC, URC1 and URC2 decoders only once. Similarly, we recommend storing the values of the 2D EXIT functions for only the 26 values of *a priori* MI in the set $\{0.00, 0.04, \ldots, 0.96, 1.00\}$. Finally, we suggest using 8 bits to store the extrinsic MI values that are obtained by the 2D EXIT functions. In Section 4.4, we will show that using more storage would not give significantly improved performance, while reducing it would degrade the performance.

Following the above recommendations, a total of 5824 bits memory is required for storing the 2D EXIT functions. We consider this amount of storage to be practical, since it is comparable to the number of bits required to store the set of 3GPP LTE interleaver parameters [120].

4.4 Comparison with Benchmarkers

In this section, we compare our Adaptive UEC-Turbo scheme of Section 4.3 with four benchmarkers, which are listed in Table 4.3.

1. Non-adaptive UEC-Turbo
   In the first benchmarker, we consider the non-adaptive UEC-Turbo scheme, employing the fixed, periodic decoder activation order of $\{$URC1, URC2, UEC; URC1, URC2, UEC; $\ldots\}$. This benchmarker has the same schematic as that in Figure 4.2. However, in both the transmitter and receiver of this non-adaptive scheme, we employ an $n = 1$-bit UEC trellis having a fixed number of states $r = 2$, 4, 6 or 8, corresponding to the UEC codebooks $\mathbb{C} = \{1\}$, $\mathbb{C} = \{1, 1\}$, $\mathbb{C} = \{1, 1, 1\}$ and $\mathbb{C} = \{1, 1, 1, 1\}$, respectively.
2. EG-URC-Turbo

Additionally, the JSCC UEC-Turbo scheme of Figure 4.2 may be compared to a classic SSCC benchmarker, which we refer to as the EG-URC-Turbo scheme, as shown in Figure 4.8. Separately from channel coding, this scheme employs an EG code for source coding, where the first ten EG codewords are illustrated in Table 4.1 and the average EG codeword length is given by $l = \sum_{x \in \mathbb{N}_1} P(x) \left(2\lfloor \log_2(x) \rfloor + 1\right)$.

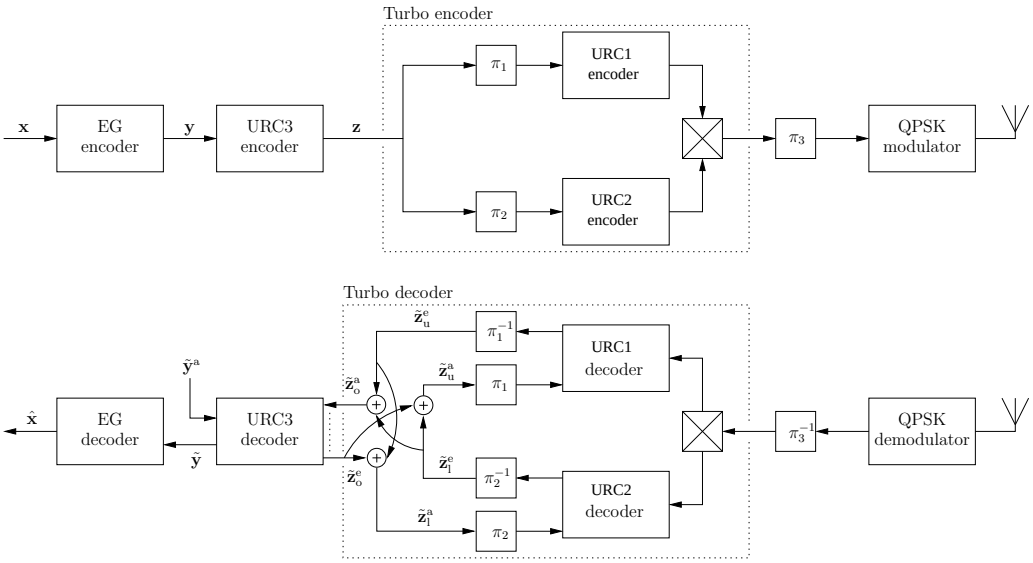

Figure 4.8: Schematic of the EG-URC-Turbo scheme, in which unary coding is serially concatenated with turbo coding and Gray-coded QPSK modulation schemes. Here, π_1, π_2 and π_3 represent interleavers, while π_1^{-1}, π_2^{-1} and π_3^{-1} represent the corresponding deinterleavers. Multiplexer and de-multiplexer operations are also employed before π_3 and after π_3^{-1}, respectively. By contrast to the UEC-Turbo scheme of Figure 4.2, the unary encoder is replaced by an EG encoder, while the trellis encoder is replaced by a URC encoder.

In analogy to Figure 3.10, the EG-URC-Turbo transmitter of Figure 4.8 may be obtained by replacing the unary encoder with an EG encoder. Furthermore, the trellis encoder is replaced by an accumulator, which we refer to as the $r = 2$-state $n = 1$-bit URC3 encoder, as illustrated in Figure 2.6. This accumulator is included in order to convert the vector of non-equiprobable bits $\mathbf{y}$ into the vector of equiprobable bits $\mathbf{z}$.

More explicitly, the b-bit vector $\mathbf{y} = [y]_{j=1}^{b}$ may be modeled as a realization of a vector $\mathbf{Y} = [Y_j]_{j=1}^{b}$ comprising b binary RVs, where $\Pr(Y_j = 0) \neq \Pr(Y_j = 1)$ and the bit entropy is $H_{Y_j} = H[\Pr(Y_j = 0)] + H[\Pr(Y_j = 1)] < 1$ in general. Owing to the recursive infinite impulse response nature of the accumulator, the encoded bits $\mathbf{z}$ have a bit entropy of $H_{Z_k} = 1$, which is necessary for avoiding capacity loss, as we will show below. In the receiver, the trellis decoder of Figure 4.2 is replaced by the URC3 decoder.

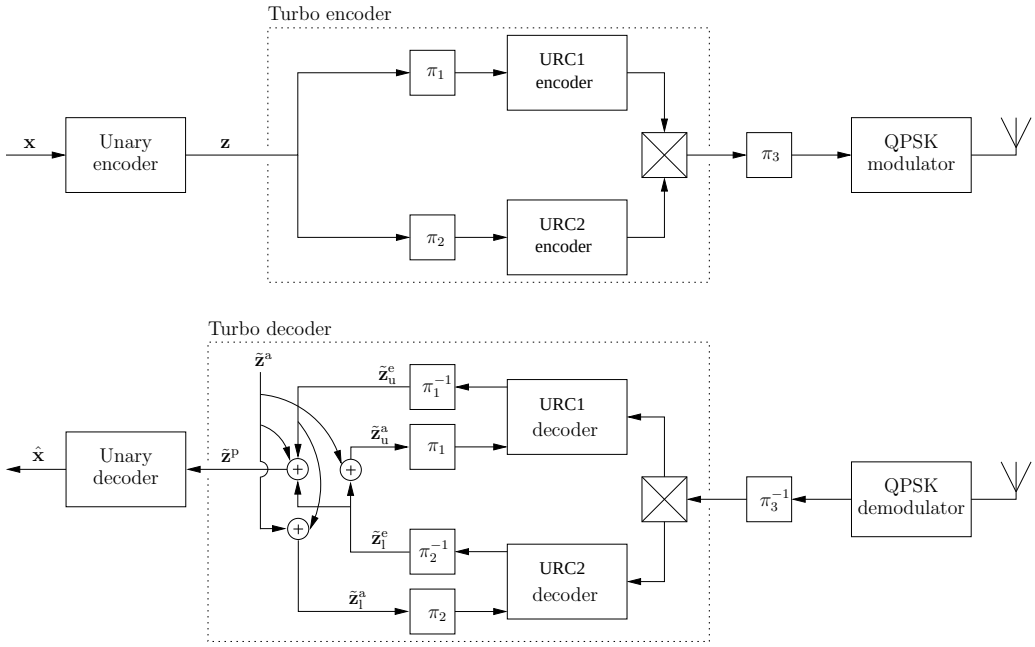

Figure 4.9: Schematic of the Unary-Turbo scheme, in which unary coding is serially concatenated with turbo coding and Gray-coded QPSK modulation schemes. Here, π_1, π_2 and π_3 represent interleavers, while π_1^{-1}, π_2^{-1} and π_3^{-1} represent the corresponding deinterleavers. Multiplexer and de-multiplexer operations are also employed before π_3 and after π_3^{-1}, respectively. By contrast to the UEC-Turbo scheme of Figure 4.2, the UEC encoder is replaced by a unary encoder.

During iterative decoding, the fixed decoder activation order {URC1, URC2, URC3; URC1, URC2, URC3; ...} is employed, which is justified since the $r = 2$-state URC3 decoder has a significantly lower decoding complexity than the $r = 8$-state URC1 and URC2 decoders. As shown in Figure 4.8, the URC3 decoder is employed to convert the *a priori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{a}}$ into the extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ using the Log-BCJR decoding algorithm. However, in analogy to Figure 3.10, this process is assisted by additionally considering the *a priori* LLR vector $\tilde{\mathbf{y}}^{\mathrm{a}} = [\tilde{y}_j^{\mathrm{a}}]_{j=1}^{b}$, where $\tilde{y}_j^{\mathrm{a}} = \ln[\Pr(Y_j = 0)/\Pr(Y_j = 1)]$. In the final decoding iteration, the URC3 decoder uses the Viterbi algorithm [28] for generating the bit sequence $\hat{\mathbf{y}}$, which may be subsequently decoded by the EG decoder.

3. Unary-Turbo

   In order to illustrate that satisfying the $H_{Z_k} = 1$ constraint is necessary for avoiding capacity loss, we consider additional SSCC benchmarkers, in which the bits input to the turbo encoder are not equiprobable, giving $H_{Z_k} < 1$.

   In the Unary-Turbo benchmarker of Figure 4.9, a unary source encoder is concatenated with a separate turbo channel encoder. This benchmarker may be considered to be an adaptation of the UEC-Turbo scheme of Figure 4.2, in which the UEC trellis encoder is omitted. Owing to this, we have $\mathbf{z} = \mathbf{y}$, $\Pr(Z_k = 0) = \Pr(Y_j = 0)$,

$\Pr(Z_k = 1) = \Pr(Y_j = 1)$ and $H_{Z_k} = H_{Y_j}$, where the bit indices k and j are interchangeable, since we have $n = 1$ bit in $\mathbf{z}$ per bit in $\mathbf{y}$.

In the receiver of Figure 4.9, the iterative decoding process exploits some of the residual redundancy present within the bit vector $\mathbf{z}$ by adding the *a priori* LLRs vector $\tilde{\mathbf{z}}^{\mathrm{a}} = [\tilde{z}_k^{\mathrm{a}}]_{k=1}^{b \cdot n}$ to the extrinsic LLR vectors, where we have $\tilde{z}_k^{\mathrm{a}} = \ln[\Pr(Z_k = 0)/\Pr(Z_k = 1)]$. Following the final decoding iteration, the *a posteriori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{p}}$ of Figure 4.9 can be soft-decision unary decoded, as described in Section 4.2.2.

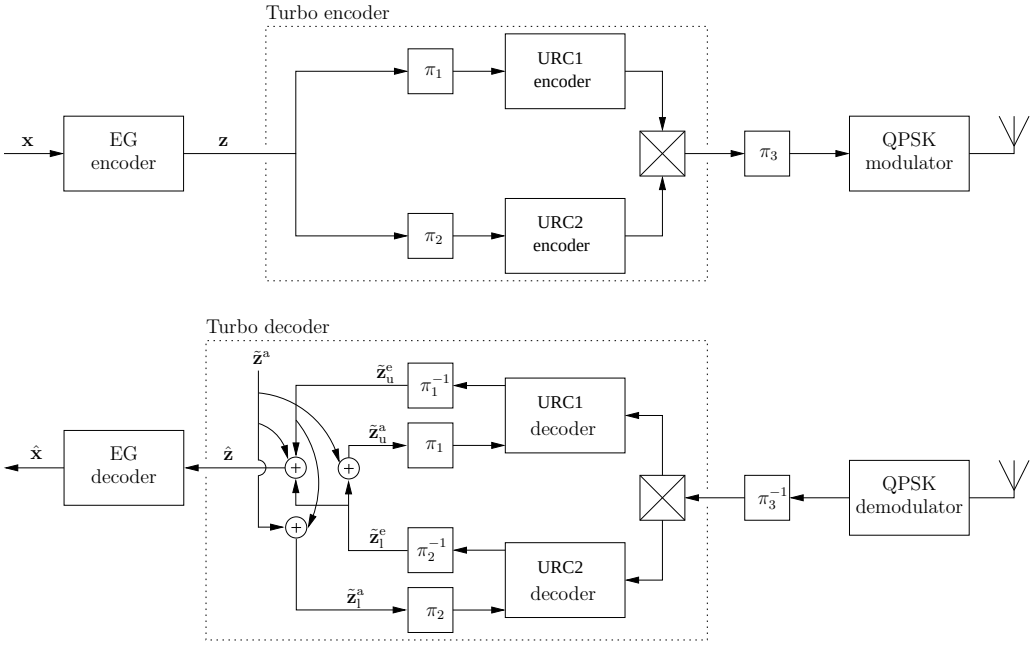

Figure 4.10: Schematic of the EG-Turbo scheme, in which unary coding is serially concatenated with turbo coding and Gray-coded QPSK modulation schemes. Here, π_1, π_2 and π_3 represent interleavers, while π_1^{-1}, π_2^{-1} and π_3^{-1} represent the corresponding deinterleavers. Multiplexer and de-multiplexer operations are also employed before π_3 and after π_3^{-1}, respectively. By contrast to the UEC-Turbo scheme of Figure 4.2, the UEC encoder is replaced by an EG encoder.

4. EG-Turbo

   Similarly, our final SSCC benchmarker can be obtained by replacing the unary code of Figure 4.9 with an EG code, as depicted in Figure 4.10. The resultant EG-Turbo scheme represents a specific version of the EG-URC-Turbo benchmarker, in which the bits forwarded to the turbo encoder are not equiprobable, giving $H_{Z_k} < 1$. Note that following the final decoding iteration in the EG-Turbo receiver, it is necessary to use hard decisions to convert the *a posteriori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{p}}$ into the bit vector $\hat{\mathbf{z}}$, before performing EG decoding.

For all five schemes, symbol values $\mathbf{x}$ that obey a zeta distribution having a parameter value of $p_1 = 0.797$ are employed. This value offers a fair comparison, since Figure 3.9 shows that unary and EG codes provide an equal average codeword length l for $p_1 = 0.797$.

As a result, all schemes have the same effective throughput of $\eta = 0.762$ information bits/symbol, when $n = 1$-bit codewords and $M = 4$-ary QPSK are employed, as shown in Table 4.3. Note that while all considered schemes have the same effective throughput η, they have different outer and inner coding rates $R_{\rm o}$ and $R_{\rm i}$, according to Eq. (4.1) and Eq. (4.2), respectively. This is because the various schemes considered have different bit entropies H_{Z_k}.

Table 4.3 compares the DCMC capacity, area and tunnel bounds of the various considered schemes, when employing QPSK modulation for communication over an uncorrelated narrowband Rayleigh fading channel. Note that all considered schemes have the same capacity bound of 0.84 dB, since they all have the same effective throughput of 0.762 information bits/symbol. The area bounds of the UEC-Turbo scheme were determined as detailed in Section 4.3.1.3. This technique was also employed for determining the area bound of the EG-URC-Turbo schemes, although Eq. (3.20) of Section 3.6.2 was employed for determining the area beneath the inverted EG-CC EXIT function, rather than Eq. (3.16) in Section 3.5.1. The area bounds of the Unary-Turbo and EG-Turbo schemes were obtained by plotting the corresponding EXIT charts, in which the measured MI can assume values in the range $[0, H_{Z_k}]$. Similarly, plots of the corresponding EXIT charts were employed for determining the tunnel bounds in the case of all the schemes considered.

As described in Section 4.3.1.3, the capacity loss imposed by the proposed UEC-Turbo scheme becomes vanishingly small as the number of UEC trellis states r is increased. Owing to this benefit of JSCC, the proposed UEC-Turbo scheme employing $r \geq 6$ UEC trellis states suffers from less capacity loss than the SSCC EG-URC-Turbo benchmarker, as shown in Table 4.3. Furthermore, the proposed UEC-Turbo scheme can be seen to impose a lower capacity loss than the SSCC Unary-Turbo benchmarker. As described in Section 4.3.1.3, the proposed UEC-Turbo scheme accrues this benefit by conditioning the bits in the vector $\mathbf{z}$ so that they adopt equiprobable values, giving $H_{Z_k} = 1$ and $2R_{\rm i} = 1$. By contrast, the bits in the vector $\mathbf{z}$ do not adopt equiprobable values in the Unary-Turbo benchmarker, giving $H_{Z_k} < 1$. As described in Section 4.3.1.3, this corresponds to URC1 and URC2 individual coding rates of $2R_{\rm i} < 1$, which imposes a capacity loss [131]. This phenomenon also explains why the $H_{Z_k} = 1$ EG-URC-Turbo benchmarker imposes a 0.91 dB lower capacity loss than the $H_{Z_k} < 1$ EG-Turbo benchmarker, as shown in Table 4.3.

The SER of the schemes considered is compared in Figure 4.11 for the case where vectors comprising $a = 10^2, 10^3, 10^4$ number of $p_1 = 0.797$ zeta-distributed symbols are

Scheme	Figure	r	R_{o}	A_{o}	R_{i}	η	E_{b}/N_0 [dB] capacity bound	E_{b}/N_0 [dB] area bound	E_{b}/N_0 [dB] tunnel bound
UEC-Turbo	4.2	2	0.762	0.934	0.500	0.762	0.84	2.48	2.7
		4		0.808				1.27	1.8
		6		0.783				1.04	1.3
		8		0.774				0.95	1.3
EG-URC-Turbo	4.8	2	0.762	0.882	0.500			1.27	2.4
Unary-Turbo	4.9	N/A	0.816	N/A	0.467			2.48	3.1
EG-Turbo	4.10	N/A	0.864	N/A	0.441			2.18	2.8

Table 4.3: Characteristics of the various schemes considered, including the outer coding rate R_{o}, the inner coding rate R_{i} and the effective throughput η. E_{b}/N_0 bounds are given for the case of Gray-coded QPSK transmission over an uncorrelated narrowband Rayleigh fading channel.

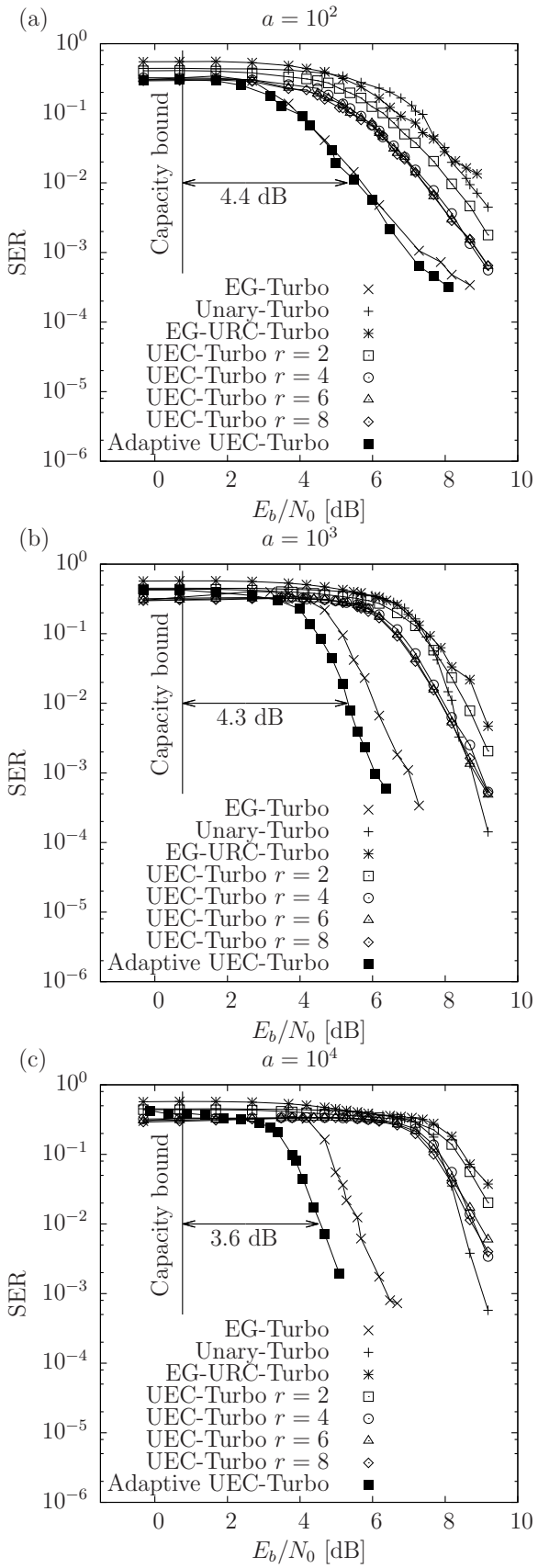

Figure 4.11: The SER performance of the five schemes of Table 4.3, when conveying vectors comprising (a) $a = 10^2$, (b) $a = 10^3$ and (c) $a = 10^4$ symbols, which obey a zeta distribution having the parameter $p_1 = 0.797$. Communication is over an uncorrelated narrowband Rayleigh fading channel and a complexity limit of $C_{\text{limit}} = 3000$ ACS operations is imposed for decoding each of the symbols in the vector $\mathbf{x}$. Different random designs are employed for the interleavers π_1, π_2 and π_3 in each simulated frame.

transmitted over an uncorrelated narrowband Rayleigh fading channel. In order to facilitate fair comparisons amongst the schemes having different computational complexities, their iterative decoding was always terminated, when the complexity limit of 3000 ACS operations for each of the $b \cdot n$ bits in the vector $\mathbf{z}$ was reached by each scheme, as show in Algorithm 4.7. Figure 4.11 shows that our proposed Adaptive UEC-Turbo scheme outperforms each of the four benchmarkers, regardless of whether $a = 10^2$, 10^3 or 10^4 symbols are conveyed by each frame. In conclusion, our proposed Adaptive UEC-Turbo scheme offers an SER of 10^{-2} at an E_b/N_0 value, which is nearly 1.2 dB lower than that required by the best benchmarker, when $a = 10^4$. This scheme is capable of operating within 3.6 dB of the DCMC capacity bound, as shown in Figure 4.11.

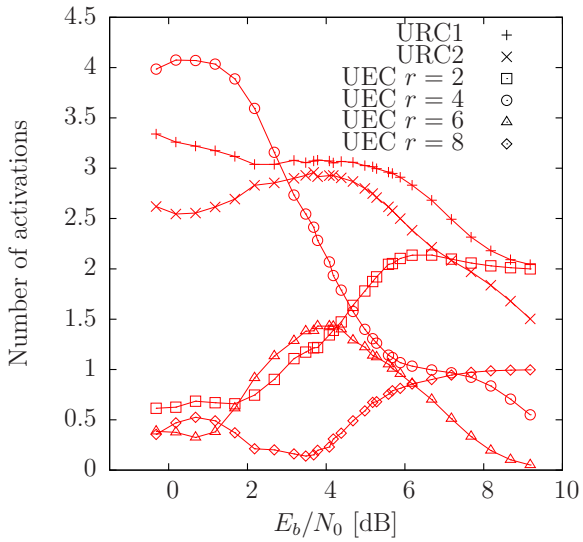

Figure 4.12: Average number of activations of the component decoders in the proposed adaptive UEC-Turbo scheme of Figure 4.2, when conveying vectors comprising $a = 10^2$ symbols, which obey a zeta distribution having the parameter $p_1 = 0.797$. Communication is over an uncorrelated Rayleigh fading channel and a complexity limit of $C_{\text{limit}} = 3000$ ACS operations is imposed for decoding each of the symbols in the vector $\mathbf{x}$. Different random designs are employed for the interleavers π_1, π_2 and π_3 in each simulated frame.

Figure 4.12 characterises the average number of times that each of the component decoders is activated, when the proposed adaptive UEC-Turbo scheme is employed to decode vectors comprising $a = 100$ symbols. This shows that at low E_b/N_0 values, the proposed adaptive iterative decoder relies on the URC decoders to a greater extent, but that as the E_b/N_0 value increases, the UEC decoder is selected more frequently. More particularly, since the $r = 2$-state UEC trellis has a lower complexity and hence a higher benefit-to-cost ratio, it is selected more frequently than the other UEC trellises. Note that similar observations were found when employing vectors comprising $a = 10^3$ and $a = 10^4$ symbols.

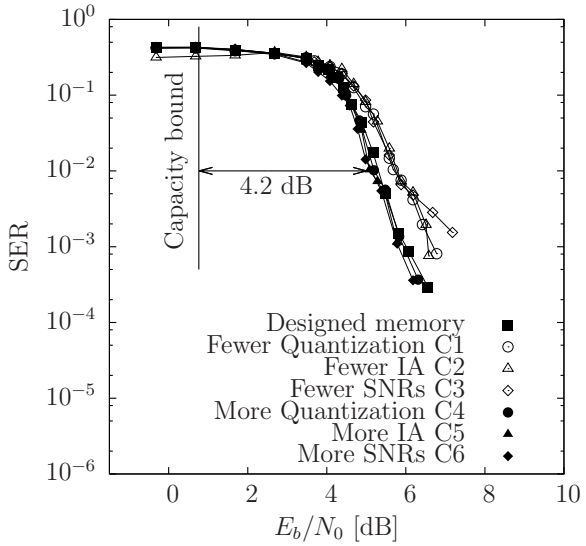

Figure 4.13: The SER performance of the proposed adaptive UEC-Turbo scheme of Figure 4.2 having various storage requirements, when conveying vectors comprising $a = 10^3$ symbols, which obey a zeta distribution having the parameter $p_1 = 0.797$. Communication is over an uncorrelated narrowband Rayleigh fading channel and a complexity limit of 3000 ACS operations is imposed for decoding each of the symbols in the vector $\mathbf{x}$. Different random designs are employed for the interleavers π_1, π_2 and π_3 in each simulated frame.

In Figure 4.13, we show how the storage requirements of the proposed adaptive UEC-Turbo scheme affect the attainable SER performance. Here, we repeat the SER performance provided in Figure 4.11(b) for vectors comprising $a = 10^3$ symbols, when employing the adaptive UEC-Turbo scheme having the 5824-bit storage requirement discussed in Section 4.3.3. This scheme relies on the storage parameters of 8-bit quantization, 26 values of *a priori* MI and 24 values of $I(\tilde{\mathbf{w}}; \mathbf{w})$. Figure 4.13 compares this SER performance to those of three schemes having lower storage requirements, namely schemes C1, C2 and C3. Additionally, the attainable SER performance is compared to those of three schemes having higher storage requirements, namely schemes C4, C5 and C6. Each of these schemes employs a different setting for one of the three storage parameters, as follows:

C1: 5-bit quantization, giving a total storage requirement of 3640 bits.

C2: 11 values of *a priori* MI in the set $\{0.00, 0.10, \ldots, 0.90, 1.00\}$, having a total storage requirement of 2464 bits.

C3: 8 values of $I(\tilde{\mathbf{w}}; \mathbf{w})$ in the set {0.337, 0.389, 0.443, 0.498, 0.554, 0.609, 0.661, 0.710}, which correspond to the E_b/N_0 values of $\{0.0, 1.0, \ldots, 6.0, 7.0\}$, exhibiting a total storage requirement of 2496 bits.

C4: 9-bit quantization, giving a total storage requirement of 6552 bits.

C5: 51 values of *a priori* MI in the set $\{0.00, 0.02, \ldots, 0.98, 1.00\}$, imposing a total storage requirement of 11424 bits.

C6: 36 values of $I(\tilde{\mathbf{w}}; \mathbf{w})$ in the set {0.337, 0.347, 0.357, 0.367, 0.378, 0.389, 0.399, 0.410, 0.421, 0.432, 0.443, 0.456, 0.465 0.476, 0.487, 0.498, 0.510, 0.521, 0.532, 0.543, 0.554, 0.565, 0.576, 0.587, 0.598, 0.609, 0.620, 0.630, 0.640, 0.651, 0.661, 0.671, 0.681, 0.691, 0.700, 0.710}, which corresponds to the E_b/N_0 values of $\{0.0, 0.2, \ldots, 6.8, 7.0\}$, giving a total storage requirement of 8320 bits.

As shown in Figure 4.13, the schemes having lower storage requirements than our recommendation in Section 4.3.3 Section 4.3.2 suffer from a degraded SER performance. By contrast, the schemes having higher storage requirements do not offer a significantly improved SER performance, motivating our recommendations. Note that similar observations were found, when employing vectors comprising $a = 10^2$ and $a = 10^4$ symbols.

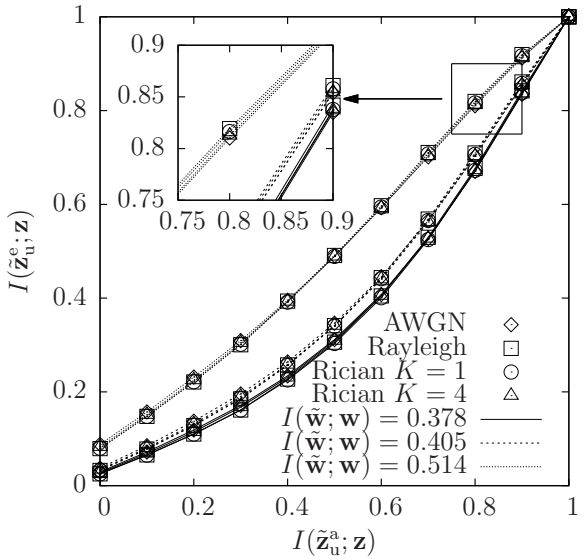

Figure 4.14: 2D EXIT functions for the URC1 decoder of the UEC-Turbo scheme of Figure 4.2, when transmitting over various uncorrelated narrowband channels, having the particular E_b/N_0 values that yield MI values of $I(\tilde{\mathbf{w}}; \mathbf{w}) = 0.378, 0.405$ and 0.514.

Furthermore, in order to generalise our adaptive decoding technique beyond the uncorrelated narrowband Rayleigh fading channel, we consider three additional channels models, namely the Additive White Gaussian Noise (AWGN) channel, as well as the Rician channel with K-factors of $K = 1$ and $K = 4$. Regardless of the channel model, Figure 4.14 illustrates that the URC EXIT functions are quite similar to each other, when the MI $I(\tilde{\mathbf{w}}; \mathbf{w})$ happens to be the same. Owing to this, the proposed adaptive algorithm can be seamlessly employed for any of the above listed channels. In these cases, our technique can continue operating on the basis of the EXIT functions obtained using the Rayleigh fading model, with no performance penalty. This is demonstrated in Figure 4.15,

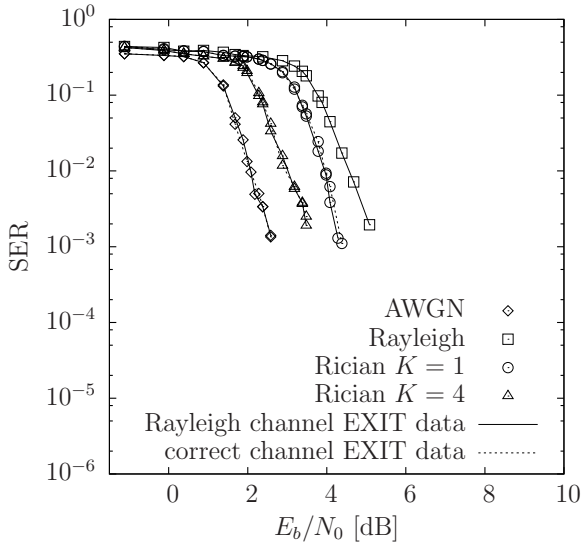

Figure 4.15: The SER performance of the proposed Adaptive UEC-Turbo scheme of Figure 4.2 in various channels, when conveying vectors comprising $a = 10^4$ symbols, which obey a zeta distribution having the parameter $p_1 = 0.797$. Plots are provided for the case where the proposed adaptive iterative decoding technique employs 2D EXIT functions that were obtained using the correct channel model. Plots are also provided for the case of employing 2D EXIT functions that were obtained using the Rayleigh fading channel model.

which compares the SER performance of our adaptive UEC-Turbo scheme, when employing EXIT functions obtained using the correct channel model and when employing those obtained using the Rayleigh fading model. As shown in Figure 4.15, the SER performance is almost identical, regardless of whether the correct channel model is employed or not.

4.5 Summary and Conclusions

In this chapter, we proposed the Adaptive UEC-Turbo scheme of Figure 4.2, which is a three-stage concatenation that applies an adaptive iterative decoding technique conceived for expediting the iterative decoding convergence. In our scheme, 3D EXIT chart analysis was employed for controlling the dynamic adaptation of the UEC trellis decoder, as well as for controlling the decoder activation order exchanging extrinsic information between the UEC decoder and the turbo decoder of Figure 4.2. Our simulation results showed that this novel adaptive decoding technique provided an improved performance compared to a conventional JSCC benchmarker employing the non-adaptive UEC scheme of Figure 3.3 and a SSCC benchmarker employing the EG-CC scheme of Figure 3.10 that was first introduced in Section 3.6.

We commenced in Section 4.2 by summarising the operation of the UEC code of Figure 4.2 in new context, which is concatenated with a turbo code. Both the UEC encoder and decoder operate on the basis of trellises, in which the transitions between the states are

synchronous with the transitions between consecutive codewords in a unary-encoded bit sequence. Based on the observation that extending the UEC codebook employed by the UEC decoder maintains compatibility with the UEC encoder, we proposed to dynamically reduce or increase the number of states employed in the trellis decoder in order to carefully balance the performance versus complexity requirements.

This idea was extended in Section 4.3, allowing not only the dynamic adjustment of the UEC decoder's operation, but also that of its activation order exchanging extrinsic information with the two turbo decoder components. We proposed the employment of 3D EXIT chart for quantifying the specific benefit of activating each decoding component at each particular stage of the iterative decoding process. At the same time, we quantified the corresponding cost in terms of the number of ACS operations performed by each decoding component. By activating the specific decoding component offering the highest benefit-to-cost ratio at each stage, we demonstrated that the convergence of the iterative decoding process may be significantly expedited. In this way, we are able to control this dynamic adaptation of the number of states employed in UEC trellis decoder, as well as to control the decoder activation order, in order to strike an attractive trade-off between the scheme's decoding complexity and its error correction capability. Furthermore, we demonstrated that the storage required for implementing the proposed adaptive iterative decoding scheme is modest compared to the storage required by the interleavers, for example.

In Section 4.4, we compared our Adaptive UEC-Turbo scheme to other JSCC and SSCC benchmarkers. The simulation results of Figure 4.11 demonstrated that the proposed scheme offers gains up to 1.2 dB, when the zeta-distributed symbols are transmitted over QPSK-modulated uncorrelated narrowband Rayleigh fading channel. Note that these gains are achieved for 'free', without imposing any additional decoding complexity, or transmission-duration, -bandwidth or -energy. Furthermore, we generalised our adaptive decoding technique beyond the uncorrelated narrowband Rayleigh fading channel scenario, by considering three additional channels models, including the AWGN channel, as well as the Rician channel having K-factors of $K = 1$ and $K = 4$. In these cases, our technique continues operating on the basis of the stored EXIT charts, without any performance penalty, regardless of which channel model is applied.

As shown in Figure 4.1, the concepts of this chapter may be extended also by considering component UEC candidate codes and irregular designs of the UEC codes, so that near-capacity operation can be achieved at even lower SNR values. Motivated by this, the following chapter will propose a novel irregular UEC trellis that is capable of can further improving the attainable decoding performance.

Chapter 5

Irregular UEC Codes for 'Nearer-Capacity' Operation

5.1 Introduction

In this chapter, we propose a novel irregular trellis design for the Unary Error Correction (UEC) code, which we refer to as the *Irregular* Unary Error Correction (IrUEC) code [94]. Unlike the conventional irregular codes of Section 2.14, the proposed IrUEC operates on the basis of a single irregular trellis, rather than a set of separate regular trellises. As highlighted in Figure 5.1, this chapter also introduces a novel EXIT chart matching algorithm for jointly matching the shapes of both the EXtrinsic Information Transfer (EXIT) functions of the proposed IrUEC and the concatenated Irregular Unity-Rate Convolutional (IrURC) codes, which we refer to as 'double-sided' matching.

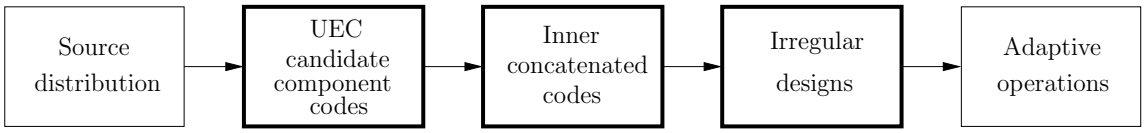

Figure 5.1: The design-flow of a UEC coded scheme. This chapter deals with the design aspects in the order indicated using the bold boxes.

5.1.1 Background and Motivation

The previous chapters proposed a novel Joint Source and Channel Coding (JSCC) scheme referred to as the UEC code [92], which was the first JSCC that mitigates capacity loss and incurs only a moderate decoding complexity, even when the cardinality of the symbol set is infinite. In a particular practical scenario, Section 4.4 showed that an iteratively-decoded serial concatenation of the UEC code with an IrURC code offers a 1.3 dB gain compared

to a conventional Separate Source and Channel Coding (SSCC) benchmarker, without incurring an increased transmission energy, duration, bandwidth or decoding complexity. Motivated by the IrURC design of Chapter 3, this chapter conceives an irregular design for the UEC code, in order to extend the regular UEC of our previous chapters.

Observe in Figure 3.4 that the generalized UEC trellis structure is parametrized by an even number of states r and by the UEC codebook $\mathbb{C}$, which comprises $r/2$ binary codewords of a particular length n. Each bit y_j of the unary-encoded bit sequence $\mathbf{y} = [y_j]_{j=1}^{b}$ corresponds to a transition in the UEC trellis from the previous state $m_{j-1} \in \{1, 2, \ldots, r\}$ to the next state $m_j \in \{1, 2, \ldots, r\}$. Each next state m_j is selected from a pair of legitimate alternatives, depending both on the previous state m_{j-1} and on the bit value y_j. More specifically, regardless of how the UEC trellis is parametrized, a unary-coded bit of $y_j = 1$ engenders a transition towards state $m_j = r-1$ or $m_j = r$ of the generalised UEC trellis of Figure 3.4 in Section 3.3.2, while the $y_j = 0$-valued bit at the end of each unary codeword resulting a transition to state $m_j = 1$ or $m_j = 2$, depending on whether the current symbol x_i has an odd or even index i.

Based on this common feature of all UEC trellises, the synchronisation between the unary codewords and trellis transitions allows the residual redundancy that remains following unary encoding to be exploited for error correction. Furthermore, this common treatment of the unary-encoded bits in $\mathbf{y}$ between all UEC trellises allows their merger in order to form a single irregular trellis. By contrast, the classic Convolutional Code (CC), Variable Length Code (VLC) and Unity-Rate Convolutional (URC) codes having different parametrizations do not generally exhibit the required similarity in their trellises.

In this chapter, we will exploit these properties of UEC codes in order to facilitate reliable operation even closer to the capacity bound. More specifically, we propose an IrUEC code. This IrUEC code employs different UEC parametrizations for the coding of different subsets of each message frame, in analogy with previous irregular codes, such as the IrURC of [100], the Irregular Convolutional Code (IrCC) of [116] and the Irregular Variable Length Code (IrVLC) of [59]. However, these previously designed irregular codes operate on the basis of a number of separate trellises, each of which has a different but uniform structure and is used for the coding of a different subset of the message frame. By contrast, our new IrUEC code operates on the basis of a single irregular trellis having a novel design, which exploits the common features of all UEC trellises, as described above. This irregular trellis has a non-uniform structure that applies different UEC parametrizations for different subsets of the frame on a bit-by-bit basis. This allows the irregularity of the proposed IrUEC code to be controlled on a fine-grained bit-by-bit basis, rather than

on a symbol-by-symbol basis. By exploiting this fine-grained control of the IrUEC irregularity, the IrUEC EXIT function may be shaped to create a narrow, but marginally open EXIT chart tunnel, hence facilitating nearer-to-capacity operation.

5.1.2 Novel Contributions

The novel contributions of this chapter are summarised as follows:

- We propose a novel IrUEC code that operates on the basis of a single irregular trellis having a novel design, which is referred to as the Irregular Trellis (IrTrellis). This IrTrellis has a non-uniform structure that applies different UEC parametrizations for different subsets of the transmitted frame on a bit-by-bit basis. This allows the irregularity of the proposed IrUEC code to be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis, hence facilitating nearer-to-capacity operation.
- Since an r-state n-bit UEC code is parametrized by a codebook $\mathbb{C}$ comprising $r/2$ number of codewords each having n bits, there are many candidates for the component codes of our proposed IrUEC code. Therefore, we analyse the candidate UEC codebooks and categorize them in terms of EXIT function shape. This allows the selection of candidate codebooks to be reduced for the sake of reducing the complexity of EXIT chart matching.
- In order to further select UEC candidate codes having good performance, the free-distance properties of the UEC codebooks are characterised for the first time, using a heuristic method that is able to obtain an approximate measurement of the free-distance.
- In our serially concatenated IrUEC-IrURC scheme, we also propose a novel extension to the double-sided EXIT chart matching algorithm of [100] that can be employed for jointly designing the EXIT function matching between the IrUEC and IrURC codes.
- We construct two versions of SSCC Elias Gamma (EG)-IrCC-IrURC benchmarker, which are EG-IrCC(sys)-IrURC benchmarker and EG-IrCC(nonsys)-IrURC benchmarker, respectively. In order to avoid the capacity loss introduced by the systematic recursive CC component codes in the EG-IrCC(sys)-IrURC benchmarker, we improve its performance by employing the non-systematic recursive CC component codes in the EG-IrCC(nonsys)-IrURC benchmarker.
- A Parallel IrUEC benchmarker is proposed in analogy with the conventional irregular codes, such as the IrCC and the IrVLC codes. However, our simulation results show that our IrUEC outperforms the Parallel IrUEC benchmarker without any increase in transmission energy, bandwidth, latency or decoding complexity.

5.1.3 Chapter Organisation

The rest of this chapter is organised as follows:

- In Section 5.2, we propose a serially concatenated IrUEC-IrURC scheme, as well as describe its transmitter operations, including the novel IrTrellis encoder.
- In Section 5.3, we discuss the corresponding receiver operations of the proposed IrUEC-IrURC scheme, where iterative decoding is employed for exchanging extrinsic information between the IrUEC decoder and the IrURC decoder.
- In Section 5.4, we investigate the free-distance properties of the UEC code for the first time, in order to obtain a range of component UEC candidate codes that offer sufficient design freedom for the parametrization of the proposed IrUEC code. We also introduce a novel double-sided EXIT chart matching algorithm that facilitates matching between the IrUEC EXIT function and the IrURC EXIT function.
- In Section 5.5, we introduce a pair of conventional SSCC schemes and a JSCC scheme as our benchmakers, which are the EG-IrCC(sys)-IrURC benchmarker, EG-IrCC(nonsys)-IrURC benchmarker and Parallel IrUEC-IrURC benchmarker.
- In Section 5.6, the performances of the proposed IrUEC-IrURC scheme and of the benchmarkers are compared. We demonstrate that our IrUEC-IrURC scheme outperforms the benchmarkers without requiring any increase in transmission energy, bandwidth, latency or decoding complexity.
- In Section 5.7, we conclude this chapter.

5.2 IrUEC-IrURC Encoder

In this section, we introduce the transmitter of the proposed IrUEC-IrURC scheme of Figure 5.2. The IrURC encoder employs T number of component URC encoders $\{\mathrm{URC}^t\}_{t=1}^{T}$, each having a distinct independent trellis structure. By contrast, the IrUEC employs a unary encoder and a novel IrTrellis encoder with a single irregular trellis. However, in analogy with the IrURC code, we note that this irregular trellis comprises a merging of S component UEC trellis structures $\{\mathrm{UEC}^s\}_{s=1}^{S}$, where UEC^s is the s-th component UEC trellis structure that is defined by the corresponding codebook $\mathbb{C}_s$, as illustrated in Figure 3.4. In Section 5.2.1 and Section 5.2.2, the two components of the IrUEC encoder in Figure 5.2, namely the unary encoder and the novel IrTrellis encoder are detailed. The IrURC encoder and the modulator are introduced in Section 5.2.3.

5.2.1 Unary Encoder

Similar to the UEC encoder of Section 3.3.1, the IrUEC encoder is also designed for conveying a vector $\mathbf{x} = [x_i]_{i=1}^{a}$ comprising a number of symbols, as shown in Figure 5.2.

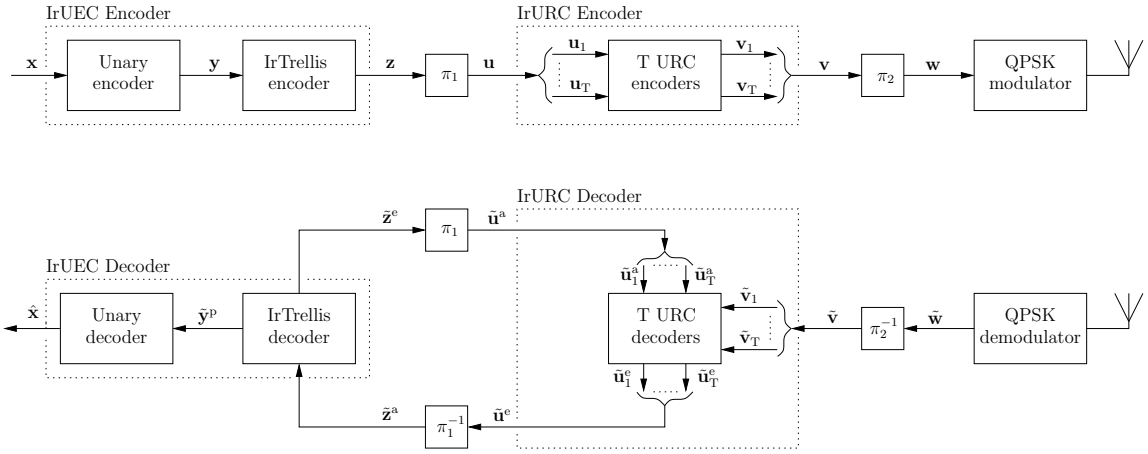

Figure 5.2: Schematic of the proposed IrUEC-IrURC scheme, in which an IrUEC code is serially concatenated with IrURC code and Gray-coded QPSK modulation schemes. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. In the UEC-IrURC scheme of Figure 3.3, the regular UEC code is replaced by an irregular UEC code.

The value of each symbol $x_i \in \mathbb{N}_1$ may be modeled by an Independent and Identically Distributed (IID) Random Variable (RV) X_i, which adopts the value x with a probability of $\Pr(X_i = x) = P(x)$. The value of each symbol in $\mathbf{x}$ can be selected from a set having an infinite cardinality, such as the set $\mathbb{N}_1 = \{1, 2, 3, \ldots, \infty\}$ comprising all positive integers. Throughout this chapter we assume that the symbol values obey a zeta probability distribution [144], since this models the symbols produced by multimedia encoders, as described in Section 3.2. Without loss of generality and as exemplified in Table 4.1, we employ a zeta distribution having the parameter of $p_1 = 0.797$ in Eq. 3.3, which was found to allow a fair comparison between unary- and EG-based schemes [92], as described in Section 4.2.1.

As shown in Figure 5.2, the IrUEC encoder represents the source vector $\mathbf{x}$ using a unary encoder. More specifically, each symbol x_i in the vector $\mathbf{x}$ is represented by a corresponding codeword $\mathbf{y}_i$ that comprises x_i bits, namely $(x_i - 1)$ binary ones followed by a zero, as exemplified in Table 4.1. The output of the unary encoder is generated by concatenating the selected codewords $\{\mathbf{y}_i\}_{i=1}^{a}$, in order to form the b-bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$. For example, the source vector $\mathbf{x} = [4, 1, 2]$ of $a = 3$ symbols yields the $b = 7$-bit vector $\mathbf{y} = [1110010]$. Note that the average length of the bit vector $\mathbf{y}$ is given by $(a \cdot l)$.

5.2.2 IrTrellis Encoder

Following unary encoding, the IrTrellis encoder of Figure 5.2 employs a single new *irregular* trellis to encode the bit vector $\mathbf{y}$, rather than using a selection of separate trellis structures, as is necessary for the IrCC [116], IrVLC [59] and IrURC [100] coding schemes. Our novel irregular trellis structure is facilitated by the properties of the generalised trellis

structure of Figure 3.4, which was the basis of our previous work on regular UEC codes. This trellis structure is parametrized by an even number of states r and by the UEC codebook $\mathbb{C}$, which comprises $r/2$ binary codewords of a particular length n. Each bit y_j of the unary-encoded bit sequence $\mathbf{y} = [y_j]_{j=1}^{b}$ corresponds to a transition in the UEC trellis from the previous state $m_{j-1} \in \{1, 2, \ldots, r\}$ to the next state $m_j \in \{1, 2, \ldots, r\}$. Each next state m_j is selected from two legitimate alternatives, depending both on the previous state m_{j-1} and on the bit value y_j, according to Eq. (3.8). More specifically, regardless of how the UEC trellis is parametrized, a unary-coded bit of $y_j = 1$ causes a transition towards state $m_j = r - 1$ or r of the generalised UEC trellis of Figure 3.4, while the $y_j = 0$-valued bit at the end of each unary codeword causes a transition to state $m_j = 1$ or 2, depending on whether the current symbol x_i has an odd or even index i.

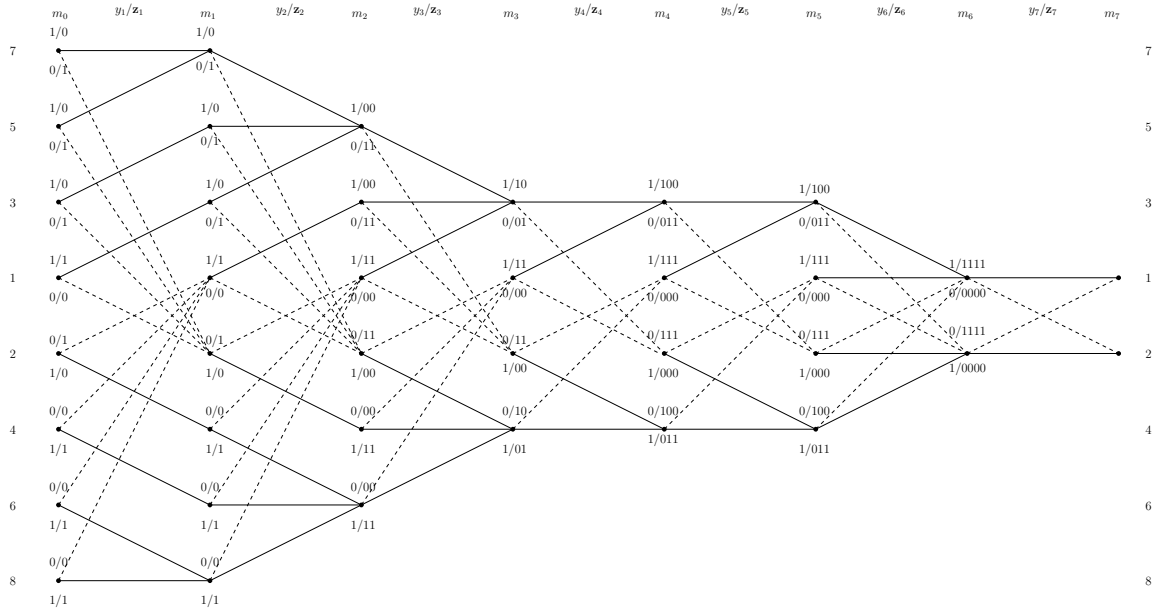

Figure 5.3: An example of the proposed irregular UEC trellis, which is obtained by amalgamating seven different UEC trellises. Here, the component UEC codebooks $\mathbb{C}_1 = \{0, 1, 1, 1\}, \mathbb{C}_2 = \{0, 1, 1, 1\}, \mathbb{C}_3 = \{000, 000, 000\}, \mathbb{C}_4 = \{00, 01\}, \mathbb{C}_5 = \{000, 011\}, \mathbb{C}_6 = \{000, 011\}$ and $\mathbb{C}_7 = \{0000\}$ are employed. By contrast, the regular trellis of Figure 3.4 only has a single trellis.

This common feature of all UEC trellises maintains synchronisation with the unary codewords and allows the residual redundancy that remains following unary encoding to be explicated for error correction. Furthermore, this common treatment of the unary-encoded bits in $\mathbf{y}$ between all UEC trellises allows them to merge in order to form our novel irregular trellis. More specifically, our novel irregular trellis can be seen as concatenation of a number of individual UEC trellis structures with different numbers of states r and different codebooks $\mathbb{C}$. By contrast, CCs, VLCs and URC codes having different parametrizations do not generally exhibit the required similarity in their trellises. More specifically, the final state of a particular component encoder has no specific relationship with the initial state

of the subsequent component encoder, hence preventing their amalgamation into IrCC, IrVLC and IrURC trellises, respectively.

The IrTrellis encoder of Figure 5.2 encodes the b-bit unary-encoded bit sequence $\mathbf{y} = [y_j]_{j=1}^{b}$ using an irregular trellis that is obtained by concatenating b number of regular UEC trellis structures. The proposed IrTrellis can be constructed using diverse combinations of component regular UEC trellises, having any parametrization. However, the component regular trellises may be strategically selected in order to carefully shape the EXIT function of the IrUEC code, for the sake of producing a narrow EXIT chart tunnel and for facilitating near-capacity operation, as it will be detailed in Section 5.4. Without loss of generality, Figure 5.3 provides an example of the irregular trellis for the example scenario where we have $b = 7$. Each bit y_j in the vector $\mathbf{y}$ is encoded using the corresponding one of these b trellis structures, which is parametrized by an even number of states r_j and the codebook $\mathbb{C}_j = \{\mathbf{c}_1^j, \mathbf{c}_2^j, \ldots, \mathbf{c}_{r_j/2-1}^j, \mathbf{c}_{r_j/2}^j\}$, which comprises $r_j/2$ binary codewords of a particular length n_j. Note that successive trellis structures can have different numbers of states, subject to the constraint $r_j \leq r_{j-1} + 2$, as it will be demonstrated in the following discussions. Note that this constraint does not restrict the generality of the IrUEC trellis, since the IrUEC EXIT function shape is independent of the ordering of the component trellis structures.

Observe in Figure 5.3, the encoding process always emerges from the state $m_0 = 1$, as in the regular UEC trellis of [92]. The unary-encoded bits of $\mathbf{y}$ are considered in order of increasing index j and each bit y_j causes the novel IrTrellis to traverse from the previous state $m_{j-1} \in \{1, 2, \ldots, r_{j-1}\}$ to the next state $m_j \in \{1, 2, \ldots, r_j\}$, which is selected from two legitimate alternatives according to Equation 3.8 of Section 3.3.2. More specifically,

$$m_j = \begin{cases} 1 + \mathrm{odd}(m_{j-1}) & \text{if } y_j = 0 \\ \min[m_{j-1} + 2, r_j - \mathrm{odd}(m_{j-1})] & \text{if } y_j = 1 \end{cases}, \tag{5.1}$$

where the function $\mathrm{odd}(\cdot)$ yields 1 if the operand is odd or 0 if it is even. Note that the next state m_j in the irregular trellis is confined by the number of states r_j in the corresponding trellis structure, rather than by a constant number of states r, as in the regular UEC trellis of [92]. In this way, the bit sequence $\mathbf{y}$ identifies a path through the single irregular trellis, which may be represented by a vector $\mathbf{m} = [m_j]_{j=0}^{b}$ comprising $b + 1$ state values.

As in the regular UEC trellis, the transitions of the proposed irregular trellis are also synchronous with the unary codewords of Table 4.1. More specifically, just as each symbol x_i in the vector $\mathbf{x}$ corresponds to an x_i-bit codeword $\mathbf{y}_i$ in the vector $\mathbf{y}$, the symbol x_i also corresponds to a section $\mathbf{m}_i$ of the trellis path $\mathbf{m}$ comprising x_i transitions between (x_i+1) states. Owing to this, the path $\mathbf{m}$ is guaranteed to terminate in the state $m_b = 1$, when the

$$P(m_j, m_{j-1}) = \begin{cases} \frac{1}{2l}\left[1 - \sum_{x=1}^{\lceil \frac{m_{j-1}}{2} \rceil} P(x)\right], \\ \text{if } m_{j-1} \in \{1, 2, 3, \ldots, r_{j-1} - 2\},\ m_j = m_{j-1} + 2. \\ \\ \frac{1}{2l} P(x)\Big|_{x=\lceil \frac{m_{j-1}}{2} \rceil}, \\ \text{if } m_{j-1} \in \{1, 2, 3, \ldots, r_{j-1} - 2\},\ m_j = 1 + \text{odd}(m_{j-1}). \\ \\ \frac{1}{2l}\left[1 - \sum_{x=1}^{\frac{r_{j-1}}{2}-1} P(x)\right], \\ \text{if } m_{j-1} \in \{r_{j-1} - 1, r_{j-1}\},\ m_j = 1 + \text{odd}(m_{j-1}). \\ \\ \frac{1}{2l}\left[l - \frac{r_{j-1}}{2} - \sum_{x=1}^{\frac{r_{j-1}}{2}-1} P(x)\left(x - \frac{r_{j-1}}{2}\right)\right], \\ \text{if } m_{j-1} \in \{r_{j-1} - 1, r_{j-1}\},\ m_j \in \{r_j - 1, r_j\}. \\ \\ 0. \qquad \text{otherwise} \end{cases} \tag{5.2}$$

symbol vector $\mathbf{x}$ has an even length a, while $m_b = 2$ is guaranteed when a is odd [92]. Note that the example unary-encoded bit sequence $\mathbf{y} = [1110010]$ corresponds to the path $\mathbf{m} = [1, 3, 5, 3, 2, 1, 1, 2]$ through the irregular UEC trellis of Figure 5.3.

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. Note that the probability $\Pr(M_j = m_j, M_{j-1} = m_{j-1}) = P(m_j, m_{j-1})$ of the transition from the previous state m_{j-1} to the next state m_j can be derived by observing the value of each symbol in the vector $\mathbf{x}$ and simultaneously its corresponding index. The state transition $\mathbf{M} = \{M_j\}_{j=0}^{b}$ follows the same rule shown in Eq. (5.1), and all the transitions can be categorised into four types, as illustrated in Eq. (3.11) of Section 3.3.2. Owing to this, the probability of a transition $P(m_j, m_{j-1})$ in the irregular trellis is associated with the transition probabilities $\Pr(M_j = m, M_{j-1} = m') = P(m, m')$ in Eq. (5.2). Note that these transition probabilities are generalized, allow their application to any IrUEC trellis and to any source probability distribution $P(x)$.

The proposed IrTrellis encoder represents each bit y_j in the vector $\mathbf{y}$ by a codeword $\mathbf{z}_j$ comprising n_j bits. This is selected from the corresponding set of $r_j/2$ codewords $\mathbb{C}_j = \{\mathbf{c}_1^j, \mathbf{c}_2^j, \ldots, \mathbf{c}_{r_j/2-1}^j, \mathbf{c}_{r_j/2}^j\}$ or from the complementary set $\overline{\mathbb{C}}_j = \{\overline{\mathbf{c}_1^j}, \overline{\mathbf{c}_2^j}, \ldots, \overline{\mathbf{c}_{r_j/2-1}^j}, \overline{\mathbf{c}_{r_j/2}^j}\}$, which is achieved according to

$$\mathbf{z}_j = \begin{cases} \overline{\mathbf{c}_{\lceil m_{j-1}/2 \rceil}^j} & \text{if } y_j = \text{odd}(m_{j-1}) \\ \mathbf{c}_{\lceil m_{j-1}/2 \rceil}^j & \text{if } y_j \neq \text{odd}(m_{j-1}) \end{cases}. \tag{5.3}$$

Finally, the selected codewords are concatenated to obtain the bit vector $\mathbf{z} = [z_k]_{k=1}^{b\bar{n}}$ of Figure 5.2, where $\bar{n} = \frac{1}{b}\sum_{j=1}^{b} n_j$ is the average codeword length. For example, the path $\mathbf{m} = [1, 3, 5, 3, 2, 1, 1, 2]$ through the irregular UEC trellis of Figure 5.3 yields the encoded bit sequence $\mathbf{z} = [1000011111110000]$, which comprises $b \cdot \bar{n} = 16$ bits, where we have $\bar{n} = \frac{16}{7}$.

Note that the bit vector $\mathbf{z}$ may be modeled as a specific realization of a vector $\mathbf{Z} = [Z_k]_{k=1}^{b\bar{n}}$ comprising $b\bar{n}$ binary RVs. Observe in Figure 5.3 that each of the b component trellis structures in the irregular UEC trellis of the IrTrellis encoder is designed to obey symmetry and to rely on complementary codewords. Hence, bits of the encoded bit vector $\mathbf{Z}$ have equiprobable values, where $\Pr(Z_k = 0) = \Pr(Z_k = 1) = 0.5$, and the bit entropy obeys $H_{Z_k} = H[\Pr(Z_k = 0)] + H[\Pr(Z_k = 1)] = 1$. Owing to this, in contrast to some of the benchmarkers to be considered in Section 5.5, the proposed IrUEC scheme of Figure 5.2 does not suffer from additional capacity loss.

We assume that each of the b trellis structures in the proposed irregular UEC trellis is selected from a set of S component UEC trellis structures $\{\mathrm{UEC}^s\}_{s=1}^{S}$, corresponding to a set of S component codebooks $\{\mathbb{C}_s\}_{s=1}^{S}$. More specifically, we assume that each codebook $\mathbb{C}_s$ is employed for generating a particular fraction α_s of the bits in $\mathbf{z}$, where we have $\sum_{s=1}^{S} \alpha_s = 1$. Here, the number of bits generated using the codebook $\mathbb{C}_s$ is given by $b \cdot \bar{n} \cdot \alpha_s$. We will in Section 5.4 show that the fractions $\alpha = \{\alpha_s\}_{s=1}^{S}$ may be designed in order to appropriately shape the IrUEC EXIT function. Moreover, the IrUEC coding rate is given by $R_{\mathrm{IrUEC}} = \sum_{s=1}^{S} \alpha_s \cdot R_{\mathrm{UEC}^s}$, where the corresponding coding rate R_{UEC^s} of the regular UEC^s code depends on the codebook $\mathbb{C}_s$ and is given by Eq. (3.13) of Section 3.3.2.

5.2.3 IrURC Encoder and Modulator

As shown in Figure 5.2, the IrUEC-encoded bit sequence $\mathbf{z}$ is interleaved in the block π_1 in order to obtain the bit vector $\mathbf{v}$, which is encoded by an IrURC encoder [98, 100] comprising T component URC codes $\{\mathrm{URC}^t\}_{t=1}^{T}$. Unlike our IrUEC code, each component URC code URC^t of the IrURC code employs a separate trellis structure. This is necessary, since the final state of each component URC code has no relation to the initial state of the subsequent component URC code, as described in Section II-B. Therefore, the interleaved IrURC-encoded bit vector $\mathbf{u}$ is decomposed into T sub-vectors $\{\mathbf{u}_t\}_{t=1}^{T}$, each having a length given by $b \cdot \bar{n} \cdot \beta_t$, where β_t represents the specific fraction of the bits in $\mathbf{v}$ that are encoded by the component URC^t code, which obeys $\sum_{t=1}^{T} \beta_t = 1$. In Section 5.4, we also show that the fractions $\beta = \{\beta_t\}_{t=1}^{T}$ may be designed in order to shape the IrURC EXIT function.

In common with each of its T number of component URC codes, the IrURC code has a coding rate of $R_{\mathrm{IrURC}} = 1$, regardless of the particular irregular code design. Owing to this, each of the T number of binary sub-vectors $\{\mathbf{v}_t\}_{t=1}^{T}$ that result from IrURC encoding has the same length as the corresponding sub-vector $\mathbf{u}_t$. The set of these sub-vectors $\{\mathbf{v}_t\}_{t=1}^{T}$ are concatenated to obtain the bit-vector $\mathbf{v}$, which comprises $b\bar{n}$ bits.

Finally, the IrURC-encoded bit vector $\mathbf{v}$ is interleaved by π_2 in order to obtain the bit vector $\mathbf{w}$, which is modulated onto the uncorrelated non-dispersive Rayleigh fading channel using Gray-mapped Quadrature Phase-Shift Keying (QPSK). The overall effective throughput of the proposed scheme is given by $\eta = R_{\mathrm{IrUEC}} \cdot R_{\mathrm{IrURC}} \cdot \log_2(M)$, where we have $M = 4$ for QPSK.

5.3 IrUEC-IrURC Decoder

In this section, we introduce the receiver of the proposed IrUEC-IrURC scheme shown in Figure 5.2. In analogy with the IrURC encoder, the IrURC decoder employs T number of component URC decoders $\{\mathrm{URC}^t\}_{t=1}^{T}$, each having a distinct independent trellis structure. By contrast, the IrUEC employs a unary decoder and a novel IrTrellis decoder relying on a single irregular trellis. In Section 5.3.1, the demodulator and the iterative operation of the IrURC and IrUEC decoders will be discussed, while in Sections 5.3.2 and 5.3.3 we will detail the internal operation of two components of the IrUEC decoder, namely of the IrTrellis decoder and of the unary decoder, respectively.

5.3.1 Demodulator and Iterative Decoding

As shown in Figure 5.2, QPSK demodulation is employed by the receiver in order to obtain the vector $\tilde{\mathbf{w}}$ of LLRs, which pertain to the bits in the vector $\mathbf{w}$. This vector is deinterleaved by π_2^{-1} for the sake of obtaining the Logarithmic Likelihood Ratio (LLR) vector $\tilde{\mathbf{v}}$, which is decomposed into the T sub-vectors $\{\tilde{\mathbf{v}}_t\}_{t=1}^{T}$ that have the same lengths as the corresponding sub-vectors of $\{\mathbf{v}_t\}_{t=1}^{T}$. Here, we assume that a small amount of side information is used for reliably conveying the lengths of all vectors in the IrUEC-IrURC transmitter to the receiver. The sub-vectors $\{\tilde{\mathbf{v}}_t\}_{t=1}^{T}$ are then input to the corresponding component URC decoders $\{\mathrm{URC}^t\}_{t=1}^{T}$ of the IrURC decoder.

Following this, iterative exchanges of the vectors of extrinsic LLRs [160] commences between the SISO IrUEC and IrURC decoders. In Figure 5.2, the notation $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{z}}$ represent vectors of LLRs pertaining to the bit vectors $\mathbf{u}$ and $\mathbf{z}$, which are related to the inner IrURC decoder and the outer IrUEC decoder, respectively. Additionally, a subscript of this notation denotes the dedicated role of the LLR, with *a*, *e* and *p* indicating *a priori*, *extrinsic* and *a posteriori* LLR, respectively.

At the beginning of iterative decoding, the *a priori* LLR vector $\tilde{\mathbf{u}}^{\mathrm{a}}$ is initialised with a vector of zeros, having the same length as the corresponding bit vector $\mathbf{u}$. As shown in the IrURC decoder of Figure 5.2, the vector $\tilde{\mathbf{u}}^{\mathrm{a}}$ is decomposed into the T sub-vectors $\{\tilde{\mathbf{u}}_t^{\mathrm{a}}\}_{t=1}^{T}$, which have the same lengths as the corresponding sub-vectors of $\{\mathbf{u}_t\}_{t=1}^{T}$. Together with $\{\tilde{\mathbf{v}}_t^{\mathrm{a}}\}_{t=1}^{T}$, the sub-vectors $\{\tilde{\mathbf{u}}_t^{\mathrm{a}}\}_{t=1}^{T}$ are fed to the corresponding URC decoder URC^t, which then outputs the resulting extrinsic LLR vectors $\{\tilde{\mathbf{u}}_t^{\mathrm{e}}\}_{t=1}^{T}$ by employing the logarithmic Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm [91]. These vectors are combined for forming the extrinsic LLR vector $\tilde{\mathbf{u}}^{\mathrm{e}}$ that pertains to the vector $\mathbf{u}$, which is sequentially deinterleaved by the block π_1^{-1} in order to obtain the *a priori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{a}}$ that pertains to the bit vector $\mathbf{z}$. Similarly, the IrTrellis decoder is provided with the *a priori* LLR vector $\tilde{\mathbf{z}}^{\mathrm{a}}$ and generates the vector of extrinsic LLRs $\tilde{\mathbf{z}}^{\mathrm{e}}$, which are interleaved in the block π_1 to obtain the *a priori* LLR vector $\tilde{\mathbf{u}}^{\mathrm{a}}$ that is provided for the next iteration of the IrURC decoder.

5.3.2 IrTrellis Decoder

As discussed in Section 5.2, our IrUEC code employs a novel bit-based irregular trellis, while the IrURC code employs a selection of independent trellises. The novel IrTrellis decoder within the IrUEC decoder applies the BCJR algorithm to the irregular trellis. The synchronization between the novel irregular trellis and the unary codewords is exploited during the BCJR algorithm's γ_t calculation of Section 2.11.2 [91]. This employs the conditional transition probability $\Pr(M_j = m_j | M_{j-1} = m_{j-1})$, where we have

$$P(m_j|m_{j-1}) = \frac{P(m_j, m_{j-1})}{\sum_{\check{m}=1}^{r_j} P(\check{m}, m_{j-1})} \tag{5.4}$$

and $P(m_j, m_{j-1})$ is given in Eq. (5.2).

Note that the IrUEC decoder will have an EXIT function [161] that reaches the $(1, 1)$ point of perfect convergence to an infinitesimally low Symbol Error Ratio (SER), provided that all component codebooks in the set $\{\mathbb{C}_s\}_{s=1}^{S}$ have a free-distance of at least 2 [150], as characterised in Section 5.4. Since the combination of the IrURC decoder and demodulator will also have an EXIT curve that reaches the $(1, 1)$ point in the top right corner of the EXIT chart, iterative decoding convergence towards the ML performance is facilitated [148]. At this point, the IrTrellis decoder may invoke the BCJR algorithm for generating the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}^{\mathrm{p}}$ that pertain to the corresponding bits in the vector $\mathbf{y}$.

5.3.3 Unary Decoder

As described in Section 3.4.3, the unary decoder of Figure 5.2 sorts the values in the 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 zero. A hard decision vector $\hat{\mathbf{y}}$ is then obtained by setting the value of these bits to zero and the value of all other bits to one. Finally, the bit vector $\hat{\mathbf{y}}$ can be unary decoded in order to obtain the symbol vector $\hat{\mathbf{x}}$ of Figure 5.2, which is guaranteed to comprise a number of symbols.

5.4 Algorithm for the Parametrization of the IrUEC-IrURC Scheme

The performance of the IrUEC-IrURC scheme depends on how well it is parametrized. A good parametrization is one that results in a narrow but still open EXIT chart tunnel, although achieving this requires a high degree of design freedom, when shaping the IrUEC and IrURC EXIT functions. Therefore, we begin in Section 5.4.1 by characterising the free-distance property of the UEC codes and selecting a set of UEC component codes having a wide variety of different inverted EXIT function shapes. This maximises the degree of freedom that is afforded, when matching the IrUEC EXIT function to that of the IrURC code. In Section 5.4.2, we propose a novel extension to the double-sided EXIT chart matching algorithm of [100], which we employ for jointly matching the EXIT functions of the IrUEC and the IrURC codes. However, in contrast to the algorithm of [100], which does not allow a particular coding rate to be targeted for the IrUEC-IrURC scheme, our algorithm designs both the fractions α and β to achieve a particular target coding rate. In Section 5.5, this will be exploited to facilitate a fair comparison with benchmarkers having particular coding rates.

5.4.1 Design of UEC Component Codes

Since an r-state n-bit UEC code is parametrized by a codebook set $\mathbb{C}$ comprising $r/2$ number of codewords each having n bits, there are a total of $2^{n \cdot r/2}$ number of candidates for $\mathbb{C}$. It is neither possible nor necessary to employ all these $2^{n \cdot r/2}$ codebooks as the component codes in our IrUEC code, because some of the codebooks will have identical or similar inverted EXIT function shapes, offering no additional degree of freedom, when performing EXIT chart matching. Therefore, it is desirable to eliminate these candidate codebooks.

The generalised UEC trellis structure associated with the codebook $\mathbb{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{r/2-1}, \mathbf{c}_{r/2}\}$ is depicted in Figure 3.4 of Section 3.3.2. Note that the upper half and the lower half of the trellis is symmetrical in terms of the output codewords z_j generated in response to a given input bit value y_j, as shown in Eq. (5.3). More specifically, for the states in the upper half of the trellis, the output codewords z_j are selected from the codebook $\mathbb{C}$ when $y_j = 0$, while the codewords from its complementary codebook $\overline{\mathbb{C}} = \{\overline{\mathbf{c}_1}, \overline{\mathbf{c}_2}, \ldots, \overline{\mathbf{c}_{r/2-1}}, \overline{\mathbf{c}_{r/2}}\}$ are selected when $y_j = 1$. For the states in the lower half of the trellis, the output codewords z_j are selected from the codebook $\mathbb{C}$ when $y_j = 1$ and from the complementary codebook

$\overline{\mathbb{C}}$ when $y_j = 0$. Intuitively, if any particular subset of the n bits at the same positions within each codeword of $\mathbb{C}$ are inverted, this would not change the distance properties of the output bit vector $\mathbf{z}$, hence resulting in an identical inverted EXIT function. For example, inverting the first bit of each codeword in the codebook $\mathbb{C}_0 = \{00, 01\}$ will give a new codebook $\mathbb{C}_1 = \{10, 11\}$ having an identical EXIT function. Likewise, inverting both bits of the codewords in $\mathbb{C}_0$ will give $\mathbb{C}_2 = \{11, 10\}$, which also has an identical EXIT function. Similarly, swapping any pair of the n bits at the same positions between each pair of codewords will not affect the distance properties or the shape of the inverted EXIT function either. For example, swapping the two bits in the codebook $\mathbb{C}_0$ results in a new codebook $\mathbb{C}_3 = \{00, 10\}$, having an identical inverted UEC EXIT function shape. Therefore, each of these four codebooks, $\mathbb{C}_0$, $\mathbb{C}_1$, $\mathbb{C}_2$ and $\mathbb{C}_3$, as well as their conversions created by bit-inversion and swapping, have identical inverted EXIT functions. Consequently, all but one of these codebooks can be eliminated as candidates for the sake of reducing the complexity of EXIT chart matching.

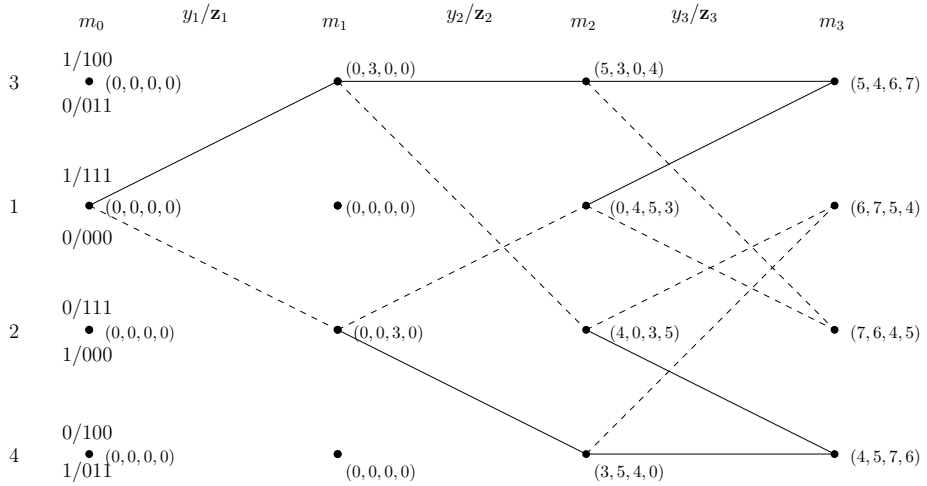

Figure 5.4: The legitimate paths through the first three stages in UEC trellis having the codewords $\mathbb{C} = \{000, 011\}$.

The number of candidate UEC codebooks may be further reduced by characterising their free-distance properties. Since no analytic method has been developed for calculating the free-distance d_f of a UEC code, we propose a heuristic method for obtaining an approximate measure of d_f. The free-distance represents the minimum distance between any pair of encoded bit vectors produced by different paths through the trellis. The total number of possible pairings of paths emerging from a particular state in a UEC trellis of length b is given by $2^{b-1} \cdot (2^b - 1)$, which grows exponentially. However, considering the symmetry of a regular UEC trellis, it is possible to use a step-by-step directed search for determining the free-distance, rather than using a brute force exhaustive search. Note that in the regular UEC trellis as generalised in Figure 3.4 of Section 3.3.2, 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 $\mathbf{z} = [z_k]_{k=1}^{b\bar{n}}$. By exploiting this observation, the free-distance d_f can be obtained by computing the Hamming Distance (HD) between each pair of paths and then selecting the pair having the minimum HD, whenever two paths merge at a particular state in the trellis.

When the bit sequence length considered satisfies $b > r/2$, the paths form complete trellis stages, as exemplified in Figure 5.4. Therefore, in order to reduce the search complexity, we consider all permutations of the b-bit unary-encoded vector $\mathbf{y}$ bit-by-bit, considering all paths that emerge from state $m_0 = 1$ and terminate at each particular state $m_b = 1, 2, \ldots, r$, on a step-by-step basis. For a pair of states $m_j, m'_j \in \{1, 2, 3, \ldots, r\}$, we define $d^j_{m_j,m'_j}$ as the minimum HD between the set of all paths that terminate at state m_j and the set that ends at state m'_j, given the input bit sequence $[y_1, y_2, \ldots, y_j]$, where $j \in \{0, 1, \ldots, b\}$. Each state m_j is labelled as $(d^j_{m_j,1}, d^j_{m_j,2}, d^j_{m_j,3}, \ldots, d^j_{m_j,r})$, where we have $d^j_{m_j,m'_j} = d^j_{m'_j,m_j}$. For each state $m_0 \in \{1, 2, 3, \ldots, r\}$, the minimum HDs are initialized to 0s. Therefore, the distance $d^j_{m_j,m'_j}$ can be calculated by

$$d^j_{m_j,m'_j} = \min_{m_{j-1},m'_{j-1}} [d^{j-1}_{m_{j-1},m'_{j-1}} + h(\mathbf{z}_{m_{j-1},m_j}, \mathbf{z}_{m'_{j-1},m'_j})]. \tag{5.5}$$

Here, $\mathbf{z}_{m_{j-1},m_j}$ is the codeword in the set $\mathbb{C}$ or in the complementary set $\overline{\mathbb{C}}$ that is generated by the transition from state m_{j-1} to state m_j, while the function $h(\cdot, \cdot)$ denotes the HD between the two operands. For each pair of states $m_j, m'_j \in \{1, 2, 3, \ldots, r\}$, we perform $r(r-1)/2$ number of comparisons, which are repeated b number of times. Owing to this, our method conceived for determining the free-distance of a UEC code has a complexity order of $O[b \cdot r(r-1)]$, where r is the number of states in the trellis and b is the length of the bit vector $\mathbf{y}$ considered. Let $\mathbb{Y}_{b_1}$ be the bit sequence set associated with the set of all paths $\mathbb{M}_{b_1}$ having a length of b_1, while $\mathbb{Y}_{b_2}$ is the bit sequence set associated with the path set $\mathbb{M}_{b_2}$ having a length of b_2. Therefore, all sequences in $\mathbb{Y}_{b_1}$ are prefix of sequences in $\mathbb{Y}_{b_2}$, when we have $b_1 < b_2$. For example, when $b_1 = 2$ and $b_2 = 3$, the bit sequence $\mathbf{y}_2 = \{111011\}$ is a prefix of the bit sequence $\mathbf{y}_3 = \{111011111\}$, where $\mathbf{y}_2$ is associated with the path vector $\mathbf{m}_2 = \{1, 3, 2\}$ and $\mathbf{y}_2$ is associated with the path vector $\mathbf{m}_2 = \{1, 3, 2, 1\}$, respectively. Note that according to Lemma 1 of [162], the minimum HD $d_f(\mathbb{Y}_{b_1})$ among all bit sequences in $\mathbb{Y}_{b_1}$ is an upper bound on the minimum HD $d_f(\mathbb{Y}_{b_2})$ of $\mathbb{Y}_{b_2}$, when we have $b_1 < b_2$. Owing to this, the approximate free-distance d_f calculated using our method converges to the true free-distance, as the lengths of the paths considered are extended towards infinity. In our experiments, we considered bit vector lengths of up to $b = 10r$. In all cases, we found that the free-distance has converged before that point, regardless of how the UEC code is parametrised, owing to the common features of all UEC codes described in Section II-B."

For example, Figure 5.4 shows all of the legitimate paths through an $r = 4$-state trellis employing the codebook $\mathbb{C} = \{000, 011\}$ that may be caused by the first three bits in a bit vector $\mathbf{y} = \{y_j\}_{j=1}^{b}$, having a length $b > 3$. Particularly, the minimum HD $d_{2,3}^{1}$ between states $m_1 = 2$ and $m_1' = 3$ is given by $d_{2,3}^{1} = d_{1,1}^{0} + h(111, 000) = 3$. Since there are no legitimate paths leading to the states $m_1 = 1$ or $m_1 = 4$, we do not update the associated distances, as shown in Figure 5.4. Similarly, we have $d_{1,2}^{2} = d_{2,3}^{1} + h(111, 011) = 4$, and $d_{1,2}^{3} = \min(d_{1,2}^{2} + h(000, 111), d_{1,4}^{2} + h(000, 100), d_{2,3}^{2} + h(111, 011), d_{3,4}^{2} + h(011, 100)) = 4$. Once the forward recursion has considered a sufficient number of trellis stages for $\min(d_{1,1}^{j}, d_{1,2}^{j}, d_{2,2}^{j}) = \min(d_{1,1}^{j-1}, d_{1,2}^{j-1}, d_{2,2}^{j-1})$, then the approximate free-distance becomes $d_f = \min(d_{1,1}^{j}, d_{1,2}^{j}, d_{2,2}^{j})$.

n	$r = 2$	$r = 4$
2		($\{00, 01\}$, 3)
3	($\{000\}$, 3)	($\{000, 011\}$, 4)
4	($\{0000\}$, 4)	($\{0000, 0111\}$, 5)

Table 5.1: After inverting and swapping, we select the IrUEC component UEC codebooks $\{\mathbb{C}_s\}_{s=1}^{5}$ with n bits and r states both up to 4. All the codebooks are in the format $(\mathbb{C}_s, d_f)$, where d_f is the approximate free-distance.

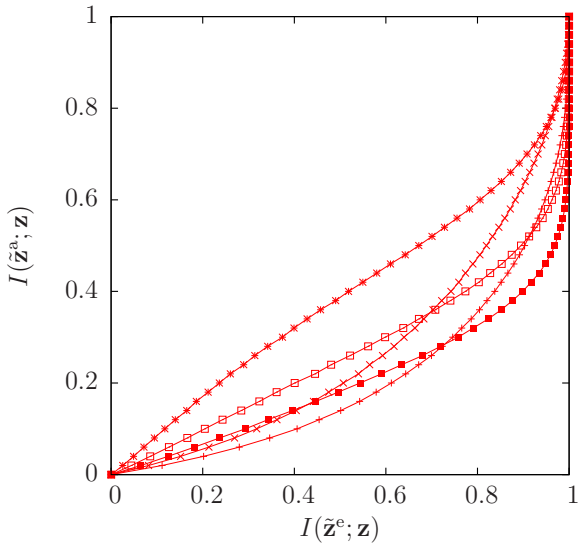

Figure 5.5: Inverted EXIT functions for the $S = 5$ component UEC codes $\{\text{UEC}^s\}_{s=1}^{5}$ of Table 5.1, when extended to $r = 10$ states codebooks, and when the symbol values obey a zeta probability distribution having the parameter value $p_1 = 0.797$ in Eq. 3.3.

Our set of candidate component UEC codes was further reduced by considering their free distances. More specifically, in order to achieve a wide variety of EXIT function shapes, we retained only UEC codebooks having the maximal or minimal free distances for each combination of $n \in \{2, 3, 4\}$ and $r \in \{2, 4\}$, where a free-distance of 3 is the minimal value that facilitates convergence to the $(1, 1)$ point [150] and avoids an error floor. We

drew the EXIT functions for all remaining candidate component UEC codes and selected the five codebooks offering the largest variety of EXIT function shapes, as listed in Table 5.1. Our experiments revealed that only insignificant EXIT function shape variations are obtained, when considering more than $r = 4$ states. Without loss of generality, our irregular trellis example of Figure 5.3 is constructed by concatenating the five UEC codebooks of Table 5.1. In the following simulations, we will consider irregular trellises that are constructed using these codebooks. However, as mentioned in Chapter 3, the number of states r employed by our five UEC component codes can be optionally and independently increased in the receiver, in order to facilitate nearer-to-capacity operation at the cost of an increased decoding complexity [92]. This is achieved by repeating the last element in the codebook. For example, while the transmitter may use the codebook $\mathbb{C} = \{00, 01\}$, the receiver may extend this to the $r = 10$-state codebook $\mathbb{C} = \{00, 01, 01, 01, 01\}$. Figure 5.5 plots the inverted EXIT functions of the component UEC codes $\{\mathrm{UEC}^s\}_{s=1}^{5}$, when extended to $r = 10$ states. Note that, similar to the IrURC EXIT function, the composite IrUEC EXIT function f_{IrUEC} is given as a weighted average of the component EXIT functions $\{f_{\mathrm{UEC}^s}\}_{s=1}^{5}$, where we have

$$f_{\mathrm{IrUEC}} = \sum_{s=1}^{5} \alpha_s \cdot f_{\mathrm{UEC}^s}. \tag{5.6}$$

5.4.2 Double-sided EXIT Chart Matching Algorithm

As depicted in the data-flow diagram of Figure 5.6, the algorithm commences by selecting the fractions α, in order to yield an IrUEC code design having a particular coding rate R_{IrUEC} and a composite IrUEC EXIT function that is shaped to match the average of T URC EXIT functions that correspond to a particular E_{b}/N_0 value. The technique of [100] may be employed for selecting the fractions β, in order to yield a composite IrURC EXIT function that is shaped to match that of the IrUEC code. Following this, the algorithm alternates between the matching of the composite IrUEC EXIT function to the composite IrURC EXIT function and vice versa, as shown in Figure 5.6.

In order to facilitate near-capacity operation, we use a 0.1 dB E_{b}/N_0 decrement per iteration for the component URC EXIT functions, when designing the fractions β for the IrURC code, until we find the lowest E_{b}/N_0 value that achieves a marginally open EXIT tunnel. Note that the double-sided EXIT chart matching algorithm allows the design of an IrUEC code having a specific coding rate R_{IrUEC}. This enables us to design the IrUEC code to have a coding rate of $R_{\mathrm{IrUEC}} = 0.254$, which provides a fair performance comparison with the regular UEC-IrURC scheme of [92] and with other benchmarkers, as detailed

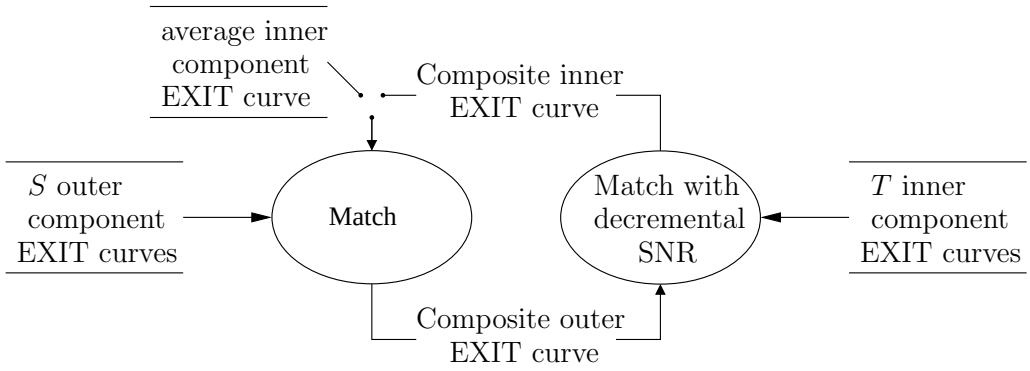

Figure 5.6: Data-flow diagram of the proposed double-sided EXIT chart matching algorithm. By contrast to the algorithm of [100], our double-sided EXIT chart matching algorithm provides a fine-control of the overall coding rate.

in Section 5.5. More specifically, this results in the same overall effective throughput of $\eta = R_{\text{IrUEC}} \cdot R_{\text{IrURC}} \cdot \log_2(M) = 0.508$ bit/s/Hz, as listed in Table 5.2.

For the IrURC encoder, we employ the $T = 10$-component URC codes $\{\text{URC}^t\}_{t=1}^{10}$ of [98,108], as discussed in Section 2.3. After running the double-sided EXIT chart matching algorithm of Figure 5.6 until the E_b/N_0 value cannot be reduced any further without closing the EXIT chart tunnel, the composite EXIT functions of the IrUEC and IrURC schemes are obtained, as depicted in Figure 5.7(a). Here, the E_b/N_0 value is 0.3 dB, which is 0.35 dB away from the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity bound of -0.05 dB and was found to be the lowest one that creates an open EXIT chart tunnel. More specifically, the fractions of the bit vector $\mathbf{z}$ that are generated by the constituent UEC codes $\{\text{UEC}^s\}_{s=1}^{5}$ of the IrUEC encoder are α = [0 0.7240 0.0924 0 0.1836], respectively. Similarly, the fractions of the bit vector $\mathbf{u}$ that encoded by the constituent URC codes $\{\text{URC}^t\}_{t=1}^{10}$ of the IrURC encoder are β = [0.1767 0 0.8233 0 0 0 0 0 0 0], respectively.

5.5 Benchmarkers

In this section, we compare the SER performance of the proposed IrUEC-IrURC scheme of Figure 5.2 to that of various SSCC and JSCC benchmarkers. As mentioned in Section 5.4, the proposed IrUEC-IrURC scheme and all benchmarkers are designed to have the same effective overall throughput of $\eta = 0.508$ bit/s/Hz, for the sake of fair comparison. A pair of benchmarkers are constituted by the UEC-IrURC and EG-CC-IrURC schemes of our previous work [92]. Furthermore, a new benchmarker is created by replacing the unary encoder and the IrTrellis encoder in the transmitter of Figure 5.2 with an EG encoder and an IrCC encoder, respectively. This results in the SSCC benchmarker of Figure 5.8, which we refer to as the EG-IrCC-IrURC scheme. Table 4.1 shows the first ten codewords of the EG code, which are used for encoding the symbol vector $\mathbf{x}$.

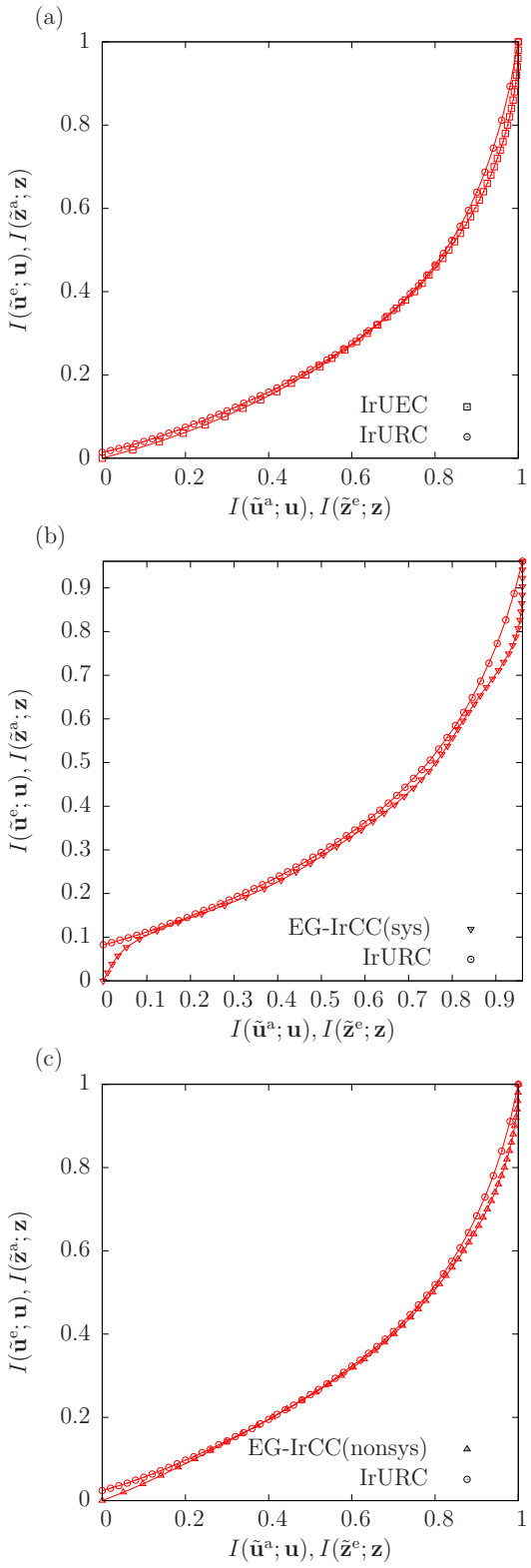

Figure 5.7: Composite EXIT functions of (a) the IrUEC decoder of Figure 5.2 employing $S = 5$ component UEC codes $\{\mathrm{UEC}^s\}_{s=1}^{5}$, (b) the EG-IrCC decoder of Figure 5.8 employing the $S = 13$ component recursive systematic CC codes $\{\mathbf{CC}_{\mathrm{sys}}^s\}_{s=1}^{13}$ and (c) the EG-IrCC scheme employing the $S = 11$ component non-systematic CC codes $\{\mathbf{CC}_{\mathrm{ns}}^s\}_{s=1}^{11}$, and the IrURC scheme employing the $T = 10$ component URC codes $\{\mathrm{URC}^t\}_{t=1}^{10}$, when conveying symbols that obeying a zeta distribution having the parameter $p_1 = 0.797$, and communicating over a QPSK-modulated uncorrelated narrowband Rayleigh fading channel. The EXIT chart tunnel is marginally open when $E_{\mathrm{b}}/N_0 = 0.3$, 2.0 and 1.1 dB, respectively. By contrast the EXIT functions of Figures 3.11, 3.12 and 3.12, an even narrower but still open tunnel is obtained.

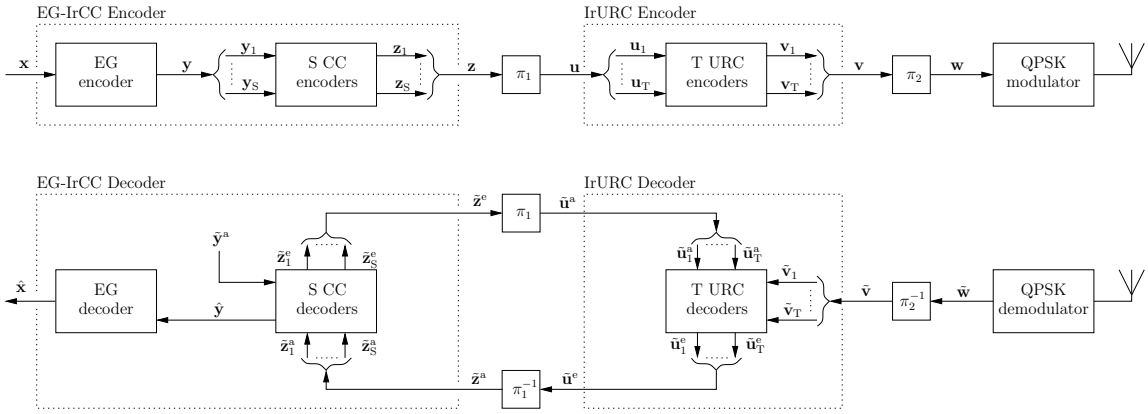

Figure 5.8: Schematic of the EG-IrCC-IrURC benchmarker, in which an EG-IrCC code is serially concatenated with IrURC code and Gray-coded QPSK modulation schemes. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. By contrast to the IrUEC-IrURC scheme of Figure 5.2, the IrUEC code is replaced by an EG-IrCC code.

As in the IrUEC-IrURC scheme, the bit vector $\mathbf{y}$ output by the EG encoder may be modeled as a realization of vector $\mathbf{Y} = [Y_j]_{j=1}^{b}$ having binary RVs. However, as observed in Section 3.6, these RVs do not adopt equiprobable values $\Pr(Y_j = 0) \neq \Pr(Y_j = 1)$, hence giving a less than unity value for the corresponding bit entropy H_{Y_j}. Similarly, the bit vector $\mathbf{z}$ of Figure 5.8 may be modeled as a particular realization of a vector $\mathbf{Z} = [Z_k]_{k=1}^{b\bar{n}}$ comprising $b\bar{n}$ binary RVs. Each binary RV Z_k adopts the values 0 and 1 with the probabilities $\Pr(Z_k = 0)$ and $\Pr(Z_k = 1)$ respectively, corresponding to a bit entropy of H_{Z_k}. In the case where the IrCC code employs systematic component codes, the bits of $\mathbf{y}$ having the entropy $H_{Y_j} < 1$ will appear in $\mathbf{z}$, resulting in a bit entropy of $H_{Z_k} < 1$. However, a bit entropy of $H_{Z_k} < 1$ is associated with a capacity loss, as described in [92].

Hence, for the sake of avoiding any capacity loss, it is necessary to use non-systematic recursive component codes, so that the bits in the resultant encoded vector $\mathbf{z}$ have equiprobable values [92]. In order to demonstrate this, we introduce two versions of the EG-IrCC-IrURC benchmarker. Firstly, the $N = 13$ recursive systematic component CC codes [116] $\{\mathbf{CC}_{\text{sys}}^{s}\}_{s=1}^{13}$ that were originally proposed for IrCC encoding are adopted in the EG-IrCC-IrURC encoder, as it will be described in Section 5.5.1. Secondly, Section 5.5.2 employs the $S = 11$ non-systematic recursive CC codebooks $\{\mathbf{CC}_{\text{ns}}^{s}\}_{s=1}^{11}$ proposed in [98], in order to offer an improved version of the EG-IrCC benchmarker. Meanwhile, the 10 component URC codebooks $\{\text{URC}^{t}\}_{t=1}^{10}$ employed by the IrURC encoder in both versions of the benchmarker of Figure 5.8 are identical to those in the IrURC encoder of Figure 5.2.

5.5.1 Recursive Systematic Component CC Codes

The recursive systematic CC codes $\{\mathbf{CC}_{\text{sys}}^{s}\}_{s=1}^{13}$ employed in [116] were designed to have coding rates of $R_{\text{CC}_{\text{sys}}^{s}} \in \{0.1, 0.15, \ldots, 0.65, 0.7\}$. However, since the EG-encoded bits

in the vector $\mathbf{y}$ are not equiprobable, none of the systematic bits in the bit vector $\mathbf{z}$ will be equiprobable either. As a result, the coding rate $R_{\mathrm{CC}^s_{\mathrm{sys}}} = \dfrac{H_{Y_j}}{n_{\mathrm{CC}^s_{\mathrm{sys}}} \cdot H_{Z_k}^{\mathrm{CC}^s_{\mathrm{sys}}}}$ of each systematic CC will be lower than the above-mentioned values. Since each CC code $\mathbf{CC}^s_{\mathrm{sys}}$ produces a different number of systematic bits, each will have a different bit entropy $H_{Z_k}^{\mathrm{CC}^s_{\mathrm{sys}}}$, and the EXIT function of each CC code will converge to a different point $(H_{Z_k}^{\mathbf{CC}^s_{\mathrm{sys}}}, H_{Z_k}^{\mathbf{CC}^s_{\mathrm{sys}}})$ in the EXIT chart [156]. The composite IrCC EXIT function will converge to a point $(H_{Z_k}^{\mathrm{IrCC}}, H_{Z_k}^{\mathrm{IrCC}})$, where $H_{Z_k}^{\mathrm{IrCC}}$ is given by a weighted average of $\{H_{Z_k}^{\mathbf{CC}^s_{\mathrm{sys}}}\}_{s=1}^{13}$, according to

$$H_{Z_k}^{\mathrm{IrCC}} = \sum_{s=1}^{13} \alpha_s \cdot H_{Z_k}^{\mathbf{CC}^s_{\mathrm{sys}}}. \tag{5.7}$$

Since the vector $\mathbf{z}$ is interleaved to generate the bit vector $\mathbf{u}$ as the input of the IrURC encoder, the IrURC EXIT function will also converge to $(H_{Z_k}^{\mathrm{IrCC}}, H_{Z_k}^{\mathrm{IrCC}})$. However, this presents a particular challenge, when parametrizing the fractions α and β of the EG-IrCC(sys)-IrURC scheme. More specifically, the fractions α vary as our double-sided EXIT chart matching algorithm progresses, causing the entropy $H_{Z_k}^{\mathrm{IrCC}}$ to vary as well. This in turn causes the IrURC EXIT function to vary, creating a cyclical dependency that cannot be readily resolved. More specifically, the fractions α must be selected to shape the EG-IrCC EXIT function so that it matches the IrURC EXIT function, but the IrURC EXIT function depends on the fractions α selected for the EG-IrCC EXIT function.

Owing to this, we design the fractions α and β by assuming that the bits of $\mathbf{y}$ are equiprobable and by plotting the inverted EXIT functions for the $S = 13$ recursive systematic CC codes accordingly, giving convergence to the $(1, 1)$ point in Figure 5.7(b). Then we invoke our double-sided EXIT matching algorithm to design the fractions α and β for the IrCC(sys) and IrURC codes, which we apply to the EG-IrCC(sys)-IrURC scheme. For the case where the bits of the vector $\mathbf{y}$ have the non-equiprobable values that result from EG encoding, the composite EXIT functions are shown in Figure 5.7(b). Here, the effective throughput is $\eta = 0.508$ bit/s/Hz and the E_{b}/N_0 value is 2.0 dB, which is the lowest value for which an open EXIT chart tunnel can be created. This E_{b}/N_0 tunnel bound is 2.05 dB away from the DCMC capacity bound of -0.05 dB, owing to the above-mentioned capacity loss. Furthermore, the EG-IrCC(sys)-IrURC scheme has an area bound of 1.72 dB, which corresponds to a capacity loss of 1.77 dB, relative to the capacity bound. The designed fractions for the EG-IrCC scheme are $\alpha =$ [0.0620 0.2997 0.0497 0.0004 0.1943 0 0.0984 0.1285 0 0 0 0.0002 0.1668], while the fractions for the IrURC code are $\beta =$ [0.6548 0 0.3452 0 0 0 0 0 0 0 0], respectively.

5.5.2 Recursive Non-Systematic Component CC Codes

In order to avoid the capacity loss introduced by the recursive systematic CC codes, we advocate the recursive non-systematic CC codebooks $\{\mathrm{CC}_{\mathrm{ns}}^{s}\}_{s=1}^{11}$, which are described by the generator and feedback polynomials provided in Table 3.2. More specifically, of the 12 codes presented in Table 3.2, we use all but the $r = 2$, $n = 2$ code, for the sake of avoiding an error floor. These recursive non-systematic CC codes attain the optimal distance properties [151] subject to the constraint of producing equiprobable bits $\Pr(Z_j = 0) = \Pr(Z_j = 1)$, which is necessary for avoiding any capacity loss. The inverted EXIT functions are plotted in Figure 5.9.

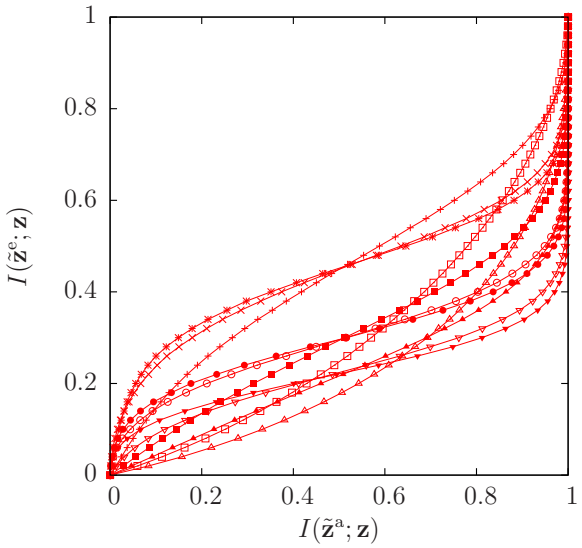

Figure 5.9: Inverted EXIT functions for EG-CC code, for the case where the $S = 11$ component recursive non-systematic CC codes $\{\mathrm{CC}_{\mathrm{ns}}^{s}\}_{s=1}^{11}$ are employed, and the symbol values obey a zeta probability distribution having the parameter value $p_1 = 0.797$.

For the sake of a fair comparison, we apply the double-sided EXIT chart matching algorithm of Figure 5.6 again to design the EG-IrCC(nonsys)-IrURC scheme having a coding rate of $R_{\mathrm{EG-IrCC}} = 0.254$ and an effective throughput of $\eta = 0.508$ bit/s/Hz. The composite EXIT functions of the EG-IrCC(nonsys) and IrURC schemes are shown in Figure 5.7(c). Here, the fractions of the EG-IrCC scheme are α = [0.8101 0 0.0643 0 0 0 0 0.1256 0 0 0], while the fractions of the IrURC code are β = [0.2386 0 0.7614 0 0 0 0 0 0 0], respectively. The EXIT chart of Figure 5.9 is provided for an E_{b}/N_0 value of 1.1 dB, which is the lowest value for which an open EXIT chart tunnel is created. As shown in Table 5.2, this E_{b}/N_0 tunnel bound is just 1.15 dB away from the DCMC capacity bound of -0.05 dB. This improvement relative to the EG-IrCC(sys)-IrURC scheme may be attributed to the non-systematic nature of the EG-IrCC(nonsys)-IrURC scheme, which has reduced the capacity loss to 1.07 dB, as quantified by considering the difference between the E_{b}/N_0 area bound of 1.02 dB and the capacity bound.

5.5.3 Parallel Component UEC Codes

In order to make a comprehensive comparison, we also consider a Parallel IrUEC-IrURC scheme. As shown in Figure 5.10, this scheme employs a parallel concatenation of S number of separate UEC trellis encoders to encode the bit vector $\mathbf{y}$, in analogy with the structure of the EG-IrCC scheme of Figure 5.8.

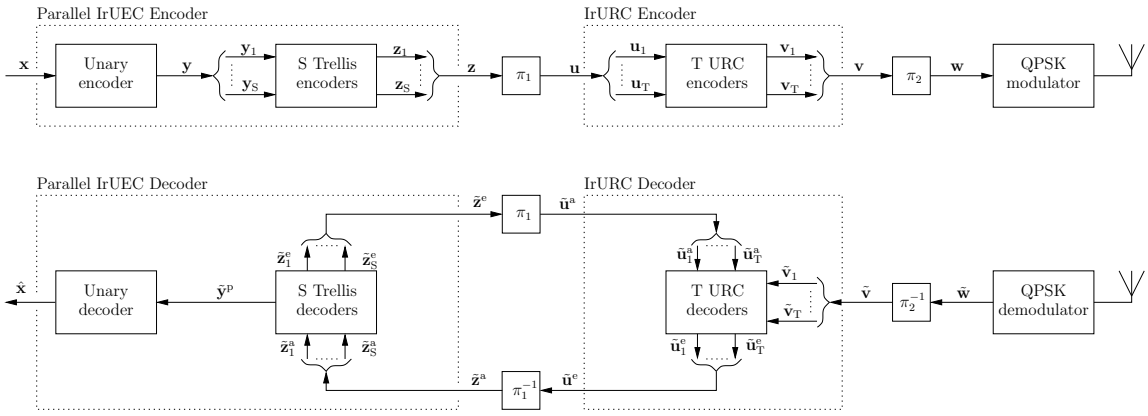

Figure 5.10: Schematic of the Parallel IrUEC-IrURC benchmarker, in which a parallel IrUEC code is serially concatenated with IrURC code and Gray-coded QPSK modulation schemes. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. By contrast to the IrUEC scheme of Figure 5.2 employing the IrTrellis of Figure 5.3, the parallel IrUEC scheme of this figure employs a parallel concatenation of separate trellises, in analogy with the structure of the EG-IrCC scheme of Figure 5.8.

The component UEC codes of the Parallel IrUEC encoder are selected from the five constituent codes provided in Table II, while the component UEC codes of the Parallel IrUEC decoder are extended to $r = 10$ states. The irregular fractions employed by the Parallel IrUEC scheme are the same as those used in our proposed IrUEC scheme. However, in order for each component UEC trellis encoder to remain synchronized with the unary codewords in the bit vector $\mathbf{y}$, it is necessary for each component trellis to commence its encoding action from state $m_0 = 1$ and end at state $m_b = 1$ or $m_b = 2$. Owing to this, the subvectors of $\mathbf{y}$ input to each component UEC must comprise an integer number of complete unary codewords. The irregular coding fractions can only be controlled at the symbol level in the case of the parallel IrUEC scheme, rather than at the bit level, as in the proposed IrUEC scheme. Therefore, the corresponding EXIT chart of the parallel IrUEC scheme is not guaranteed to have an open tunnel, when the E_b/N_0 value approaches the tunnel bound of Table 5.2, hence resulting in a degraded SER performance. However, if the frame length a was orders of magnitude higher, the difference between the symbol-based and bit-based segmentations of the bit vector $\mathbf{y}$ would become insignificantly small. As a result, a similar SER performance may be expected for the parallel IrUEC scheme

in this case. In the following section, we will compare the performances of the Parallel IrUEC and the proposed IrUEC schemes, using different values for the frame length a.

5.6 Simulation Results

The SER performance of the IrUEC-IrURC, the EG-IrCC(sys)-IrURC and the EG-IrCC(nonsys)-IrURC, UEC-IrURC and EG-CC-IrURC schemes is characterised in Figure 5.11. In each case, the source symbol sequence $\mathbf{x}$ comprises $a = 10^4$ symbols, the values of which obey a zeta distribution having a parameter value of $p_1 = 0.797$. As shown above, the parametrizations of the irregular codes in each scheme are designed to achieve the closest possible matching of EXIT charts, while giving the same overall effective throughput of $\eta = 0.508$ bit/s/Hz. Transmission is performed over a Gray-coded QPSK-modulated uncorrelated narrowband Rayleigh fading channel, resulting in the DCMC capacity bound of -0.05 dB. We select two parametrizations of the schemes in [92] to create two of our four benchmarkers, namely the $r = 4$-state UEC-IrURC and the $r = 4$-state EG-CC-IrURC schemes. Note that the $r = 4$-state EG-CC-IrURC scheme was found to outperform other parametrizations of the same scheme having higher number of states, owing to its superior EXIT chart matching accordingly. With the same effective throughput η, a fair comparison is provided between our proposed IrUEC-IrURC scheme and the four benchmarkers.

Note that the practical implementation of the time-variant IrTrellis used in our IrUEC-IrURC scheme follows the same principles as the parallel time-invariant trellises of the benchmarker schemes, such as the EG-IrCC-IrURC scheme and the regular UEC-IrURC scheme. Once the irregular coding fractions have been determined, the specific portions of message that should be encoded and decoded by the corresponding trellises are also determined. In both time-variant and parallel time-invariant trellises, the hardware is required to support different trellis structures, which may be implemented by appropriately changing the connections among the states of a single hardware implementation of a trellis. Although the proposed time-invariant trellis has some peculiarities at the interface between its different sections, these can also be implemented using the same hardware at either side of the interface. As an example platform for hardware implementation, the computation unit of [152] performs one ACS arithmetic operation per clock cycle, which are the fundamental operations used in BCJR decoders [96]. Therefore, the implementational complexity depends only on the computational complexity, as quantified per decoding iteration in Table 5.2. Since a common computational complexity limit is used in our comparisons of the various schemes, they can be deemed to have the same implementational complexity. Although the routing and control of the proposed IrTrellis may be expected to be more

Scheme	Figure	Codebooks	R_{o}	R_{i}	η	E_{b}/N_0 [dB] capacity bound	E_{b}/N_0 [dB] area bound	E_{b}/N_0 [dB] tunnel bound	Complexity
IrUEC-IrURC	5.2	$\{\mathrm{UEC}^s\}_{s=1}^{5}$	0.254	1	0.508	-0.05	0.21	0.3	258
IrUEC(med)-IrURC							0.30	0.6	192
IrUEC(low)-IrURC							1.14	1.2	157
UEC-IrURC	3.3	$\{000, 011\}$					0.49	1.7	120
EG-IrCC-IrURC	5.8	$\{\mathbf{CC}_{\mathrm{sys}}^s\}_{s=1}^{13}$	0.667	0.576			1.72	2.0	341
		$\{\mathbf{CC}_{\mathrm{ns}}^s\}_{s=1}^{11}$	0.254	1			1.02	1.1	146
EG-CC-IrURC	3.10	([4,7,7],6,6)					1.00	2.2	132

Table 5.2: Characteristics of the various schemes considered, including outer coding rate R_{o}, inner coding rate R_{i} and effective throughput η. E_{b}/N_0 bounds are given for the case of Gray-coded QPSK transmission over an uncorrelated narrowband Rayleigh fading channel. Complexity is quantified by the average number of Add, Compare and Select (ACS) operations incurred per decoding iteration and per bit in the vector $\mathbf{z}$.

complicated than in the parallel time-invariant trellises of the benchmarkers, it may be expected that the associated overhead is negligible compared to the overall implementational complexity.

As shown in Table 5.2, our IrUEC-IrURC scheme imposes a complexity of 258 ACS operations per iteration per bit, when employing $r = 10$ states for each component UEC code in the IrTrellis decoder. We also consider alternative parametrizations of our IrUEC-IrURC scheme, which employ an IrTrellis having fewer states, in order to achieve lower complexities. The IrUEC(med)-IrURC scheme relies on $r = 6$ trellis states for different stages of the IrTrellis, which results in a total complexity of 192 ACS operations per iteration per bit. This matches that of the UEC-IrURC benchmarker. At the same time, the IrUEC(low)-IrURC scheme employs the minimal number of states for each stage of the IrTrellis, namely either $r = 4$ states, as listed in Table 5.1, hence resulting in a complexity of 157 ACS operations per iteration per bit.

During the simulation of each scheme, we recorded both the SER and the complexity incurred after each decoding iteration, resulting in a Three-Dimensional (3D) plot of SER versus E_b/N_0 and versus complexity. Figure 5.11 presents Two-Dimensional (2D) plots of SER versus E_b/N_0 relationship, which were obtained by slicing through these 3D plots at a particular complexity. More specifically, we select the complexity limits of $10,000$ and $5,000$ ACS operations per iteration per bit in Figure 5.11(b) and (c), respectively. Meanwhile, Figure 5.11(a) characterizes the SER performance achieved after iterative decoding convergence, regardless of the complexity.

As shown in Table 5.2, the proposed IrUEC-IrURC scheme has an area bound of 0.21 dB, which is the E_b/N_0 value where the area A_o beneath the inverted IrUEC EXIT function equals that beneath the IrURC EXIT function. Although the UEC-IrURC benchmarker has a similar area bound of $E_\mathrm{b}/N_0 = 0.49$ dB, it has an inferior EXIT chart matching capability owing to its employment of regular UEC constituent codes. By contrast, the employment of two irregular codes in the proposed IrUEC-IrURC scheme facilitates an open EXIT chart tunnel at an E_b/N_0 value of 0.3 dB, which is 1.4 dB lower than the open tunnel bound of the UEC-IrURC benchmarker. Note that the area and tunnel bounds are degraded in the context of the lower complexity versions of the proposed IrUEC-IrURC scheme, which have fewer states in the IrTrellis. This may be explained by the increased capacity loss encountered when the number of UEC states is reduced [92]. Note however that even with a reduced complexity, the proposed IrUEC-IrURC scheme tends to exhibit superior area and tunnel bounds, when compared to the EG-IrCC-IrURC and EG-CC-IrURC benchmarkers, as shown in Table 5.2. This may be attributed to the large capacity loss that is associated with SSCC scheme [92].

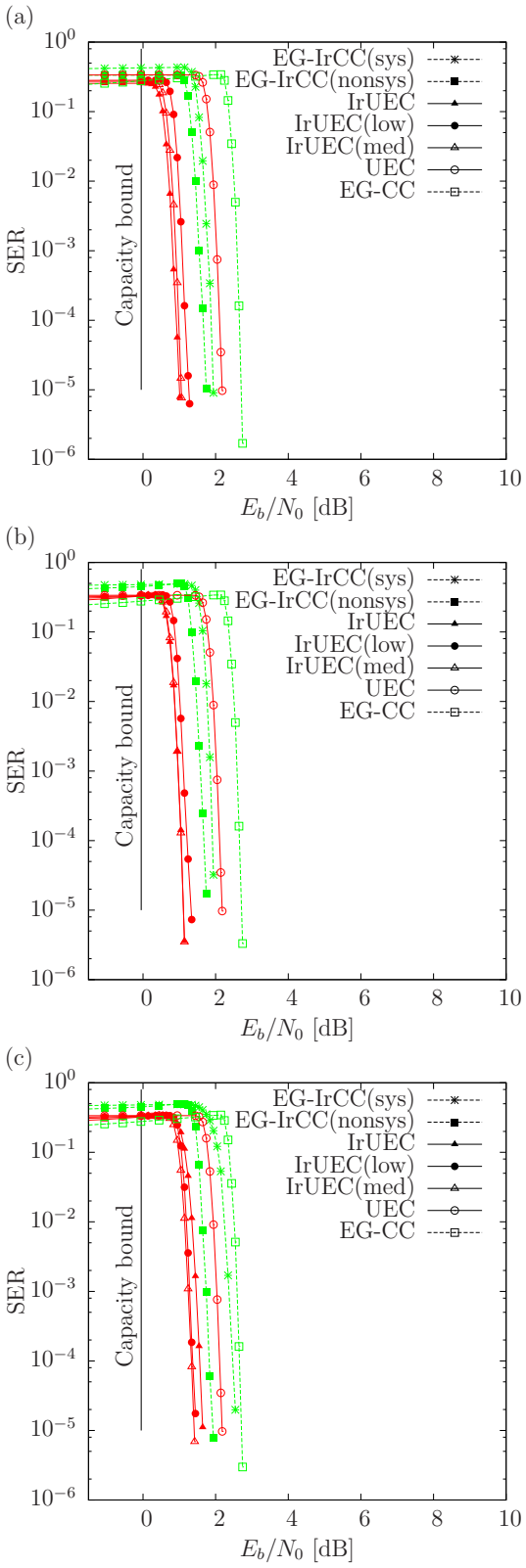

Figure 5.11: SER performance for various arrangements of the proposed IrUEC-IrURC scheme of Figure 5.2, the EG-IrCC-IrURC of Figure 5.8, the Parallel IrUEC-IrURC scheme of Figure 5.10, as well as the UEC-IrURC scheme of Figure 3.3 and the EG-IrURC scheme of Figure 3.10, when conveying $a = 10^4$ number of symbols obey a zeta distribution having the parameter $p_1 = 0.797$, and communicating over a QPSK-modulated uncorrelated narrowband Rayleigh fading channel having a range of E_b/N_0 values. A complexity limit of (a) unlimited, (b) $10,000$ and (c) $5,000$ ACS operations per decoding iteration is imposed for decoding each of the bits in $\mathbf{z}$.

Figure 5.11 demonstrates that our proposed IrUEC-IrURC scheme has a superior SER performance compared to all other benchmarkers, regardless of which complexity limit is selected in this particular scenario. For example, as shown in Figure 5.11(a), our IrUEC-IrURC scheme facilitates operation within 0.4 dB of the capacity bound, offering a 0.8 dB gain compared to the EG-IrCC(nonsys)-IrURC scheme, which is the best-performing of the SSCC benchmarkers. This is achieved without any increase in transmission energy, bandwidth, transmit duration or decoding complexity. Note that the EG-IrCC(nonsys)-IrURC benchmarker offers a 0.9 dB gain over the EG-IrCC(sys)-IrURC benchmarker, which is owing to the capacity loss that is associated with systematic IrCC component codes. As expected, the reduced complexity versions of the proposed IrUEC-IrURC scheme exhibit a degraded SER performance. However, the IrUEC(low)-IrURC scheme can be seen to offer up to 0.5 dB gain over the UEC-IrURC benchmarker, which has a close decoding complexity per bit per iteration.

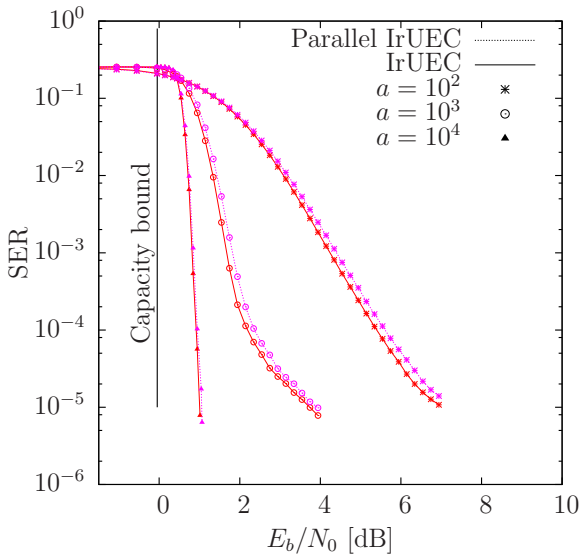

Figure 5.12: SER performance for various frame lengths $a \in \{10^2, 10^3, 10^4\}$ of the proposed IrUEC-IrURC scheme of Figure 5.2 and the Parallel IrUEC-IrURC scheme of Figure 5.10, when conveying symbols obeying a zeta distribution having the parameter $p_1 = 0.797$, and communicating over a QPSK-modulated uncorrelated narrowband Rayleigh fading channel having a range of E_b/N_0 values.

Since the Parallel IrUEC-IrURC scheme can only provide a symbol-level control of the irregular coding fractions, the EXIT chart tunnel is not guaranteed to be open at low E_b/N_0 values. As a result, Figure 5.12 shows that the Parallel IrUEC-IrURC scheme of Figure 5.10 performs relatively poorly compared to the proposed IrUEC-IrURC scheme, particularly when the frame length has values of $a = 10^2$ and $a = 10^3$ symbols. Note that this performance gain offered by the proposed scheme is obtained without imposing any additional decoding complexity and without requiring any additional transmission-energy, -bandwidth, or -duration. In analogy with Figure 5.11(a), an additional set of SER

results is provided in Figure 5.13 for the various schemes considered, where the source symbols obey a zeta distribution having the parameter $p_1 = 0.9$, where the complexity is potentially unlimited. It can be seen that the proposed IrUEC-IrURC scheme also outperforms all other benchmarkers in this situation, offering a 1 dB gain compared to the EG-IrCC(nonsys)-IrURC scheme, which is the best-performing one of the set of SSCC benchmarkers.

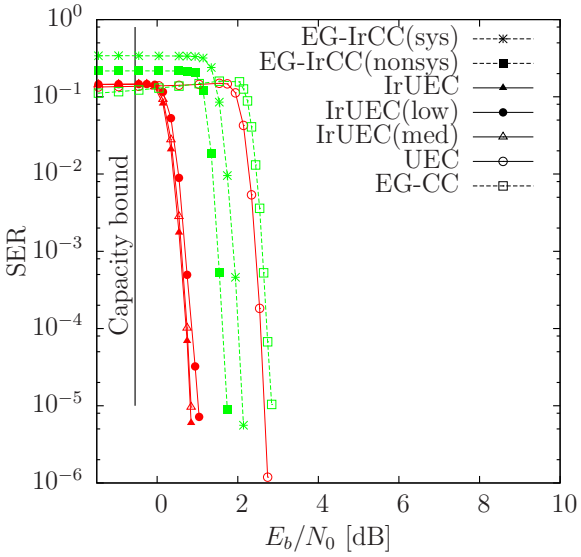

Figure 5.13: SER performance for various arrangements of the proposed IrUEC-IrURC scheme of Figure 5.2, the EG-IrCC-IrURC of Figure 5.8, the Parallel IrUEC-IrURC scheme of Figure 5.10, as well as the UEC-IrURC scheme of Figure 3.3 and the EG-IrURC scheme of Figure 3.10, when conveying $a = 10^4$ number of symbols obey a zeta distribution having the parameter $p_1 = 0.9$, and communicating over a QPSK-modulated uncorrelated narrowband Rayleigh fading channel having a range of E_b/N_0 values. The complexity is unlimited for decoding each of the bits in $\mathbf{z}$.

Note that the performance gain of the proposed IrUEC-IrURC scheme is obtained by elaborately designing the IrUEC EXIT function, in order to create a narrow but marginally open EXIT chart tunnel at a low E_b/N_0 value that is close to the area bound and capacity bound, as discussed in Section 5.4.2. Since the benchmarker schemes suffer from capacity loss which separates their tunnel, area and capacity bounds, the performance gain of the proposed IrUEC-IrURC scheme depicted in Figure 5.11 and 5.13 may be expected in the general case, regardless of the specific source probability distribution and the parametrization of the scheme. As an additional benefit of the proposed IrUEC-IrURC scheme, a single bit error within a particular codeword can only result in splitting it into two codewords, or into merging it with the next codeword, since every unary codeword contains only a single 0. Fortunately, the decoding of the other unary codewords will be unaffected. Owing to this, a single bit error in the IrUEC-IrURC scheme can only cause a Levenshtein distance [163] of 2, hence preventing error propagation. By contrast, in the EG-based

benchmarkers, a single bit error can cause error propagation, resulting in a Levenshtein distance that is bounded only by the length of the message.

5.7 Summary and Conclusions

In this chapter, we proposed a novel near-capacity JSCC scheme, which we refer to as the IrUEC code. Like the regular UEC code of [92], it employs a unary code, but replaces the UEC's trellis code with a novel IrTrellis code. Unlike the conventional irregular codes of Figures 5.8 and 5.10, the IrTrellis code of Figure 5.2 operates on the basis of the single amalgamated irregular trellis of Figure 5.3, rather than a number of separate trellises. This allows the irregularity of the proposed IrUEC code to be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis, hence facilitating nearer-to-capacity operation. More specifically, our results demonstrate that controlling the IrUEC irregularity on a bit-by-bit basis offers gains of up to 0.5 dB over the best-performing symbol-by-symbol approach, without imposing any increase in transmission energy, bandwidth, latency or decoding complexity.

In Section 5.2 and Section 5.3, we introduced the transmitter and receiver of the proposed IrUEC-IrURC scheme of Figure 5.2, respectively. More particularly, our proposed IrUEC encoder employs a unary encoder and a novel IrTrellis encoder relying on a single irregular trellis, while the IrURC encoder employs T component URC encoders $\{\mathrm{URC}^t\}_{t=1}^{T}$, each having a distinct independent trellis structure. In analogy to the IrURC code of Figure 5.2, we denote the IrTrellis code of Figure 5.2 by S merged component UEC trellises $\{\mathrm{UEC}^s\}_{s=1}^{S}$, where UEC^s is the s-th component UEC trellis structure defined by the corresponding codebook $\mathbb{C}_s$. At the receiver, both the IrUEC and IrURC decoder apply the BCJR algorithm and perform iterative decoding for exchanging increasingly reliable soft information.

In order to parametrize the IrUEC-IrURC scheme of Figure 5.2, Section 5.4 characterises the free-distance properties of the UEC trellis for the first time, so that a suite of UEC codes having a wide variety of EXIT curve shapes can be selected for the component codes of our IrUEC code. Furthermore, we introduced the new double-sided EXIT chart matching algorithm of Figure 5.6. On the one hand, the component UEC codes having a wide variety of EXIT chart shapes provide design freedom for the IrUEC EXIT chart. On the other hand, the novel double-sided EXIT chart matching algorithm exploit this design freedom for parametrizing the IrUEC-IrURC scheme for the sake of creating a narrow but marginally open EXIT chart tunnel at a low E_b/N_0 value, which is close to the area bound and the capacity bound.

The simulation results of Section 5.6 demonstrated that near-capacity operation is facilitated at $E_{\rm b}/N_0$ values that are within 0.4 dB of the DCMC capacity bound, when achieving an effective throughput of $\eta = 0.508$ bit/s/Hz and employing QPSK for transmission over an uncorrelated narrowband Rayleigh fading channel. This corresponds to a gain of 0.8 dB compared to the best of several SSCC benchmarkers, which is achieved without any increase in transmission energy, bandwidth, transmit duration or decoding complexity.

As shown in Figure 5.1, the contribution of this chapter may be complemented by also considering non-stationary and unknown source distributions. In the following chapter, we will design a dynamic learning-aided version of the UEC scheme, which can learn the unknown source statistics and gradually improve its decoding performance during a transient phase, then dynamically adapt to the non-stationary statistics and maintain its decoding performance during a steady state phase.

Chapter 6

Learning-aided UEC Codes for Non-Stationary and Unknown Sources

6.1 Introduction

In this chapter, we propose a dynamic version of the Unary Error Correction (UEC) scheme [95] of Figure 6.4 for non-stationary and unknown source distributions, which we refer to as the *Learning-aided* UEC scheme. It is capable of learning the unknown source statistics and hence gradually improves its decoding performance during its learning phase of operation, then dynamically adapts to the non-stationary statistics and maintains reliable near-capacity operation during its steady-state phase. As highlighted in Figure 6.1, this chapter considers the the source probability distribution and adaptive decoding operations.

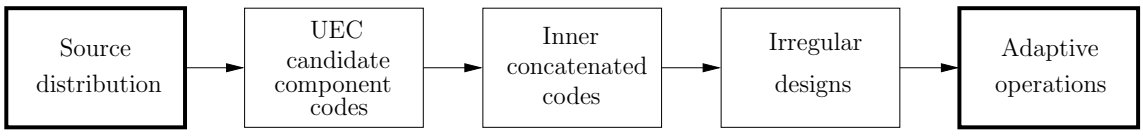

Figure 6.1: The design-flow of a UEC coded scheme. This chapter deals with the design aspects in the order indicated using the bold boxes.

6.1.1 Background and Motivation

Chapter 3 has considered a novel Joint Source and Channel Coding (JSCC) scheme, which was referred to as a UEC code [92] was proposed and it was the first JSCC that mitigates capacity loss and incurs only a moderate decoding complexity, even when the cardinality of the symbol set is infinite. Based on the UEC scheme, we proposed an *Adaptive*

UEC scheme and *Irregular* UEC scheme in Chapters 4 and 5, respectively. For the Adaptive UEC scheme, we proposed an adaptive iterative decoding technique for expediting the iterative decoding convergence of UEC codes. We also proposed the employment of Three-Dimensional (3D) EXtrinsic Information Transfer (EXIT) charts for controlling the dynamic adaptation of the UEC trellis decoder, as well as for controlling the decoder activation order between the UEC decoder and the turbo decoder. In the Irregular UEC scheme of Chapter 5, we proposed a single irregular trellis, which operates on a bit-by-bit basis. By exploiting this fine-grained control of the *Irregular* Unary Error Correction (IrUEC) irregularity, the IrUEC EXIT function can be shaped to create a narrow, but marginally open EXIT chart tunnel, hence facilitating 'nearer-to-capacity' operation.

However, in the previous chapters, the UEC code was only capable of achieving near-capacity operation when the source distribution was known at the receiver. Hence, the applicability of the UEC code was limited to some particular scenarios and its application was prevented in the generalised case of unknown and non-stationary source probability distributions.

In this chapter, we propose a new *Learning-aided* UEC scheme. Like our previous UEC schemes, the proposed learning-aided scheme does not require any prior knowledge of the source distribution at the transmitter. However, in contrast to our previous UEC schemes of Chapters 3, 4 and 5, the proposed learning-aided scheme is also capable of achieving near-capacity operation without any prior knowledge of the source distribution at the receiver. This is achieved by learning the source distribution based on the received symbols. Starting from a situation of having no prior information about the source distribution, the proposed receiver becomes capable of recovering a first frame of symbols, albeit possibly with a relatively high Symbol Error Ratio (SER), if the channel's Signal to Noise Ratio (SNR) is close to the capacity bound. Nonetheless, this frame of symbols can be used for making a first estimate of the source distribution, which is stored in memory. This information can then be used to aid the recovery of a second frame of symbols, with an improved SER. This allows the estimate of the source distribution that is stored in memory to be improved. In this way, the SER and the estimate of the source distribution can be gradually improved in successive frames during the learning phase. Following this, the receiver enters a steady-state phase, during which reliable near-capacity operation can be maintained by continuing the learning process, even if the source is non-stationary.

6.1.2 Novel Contributions

The novel contributions of this chapter are summarised as follows:

- We analyse the characteristics of the H.265 codec's entropy-encoded symbol values.

Inspired by the distribution of these symbol values, we propose a non-stationary probability distribution model, which can be readily parametrized to represent the H.265 distribution.

- We propose a novel learning-aided UEC scheme, which does not require any prior knowledge of the source distribution at the receiver. The learning algorithm gradually infers the source distribution based on the received symbols and feeds this information back to the decoder in order to assist the decoding process. Hence, near-capacity operation is facilitated.
- In order to implement our learning algorithm, we employ a memory storage at the receiver, which is used for storing the source distribution statistics that have been observed from the successively recovered symbol vectors. Moreover, we quantify the size of this memory storage in terms of the number of most-recently recovered symbol vectors, and strike an attractive trade-off between the size of the memory and the error correction capability.
- We propose a pair of learning-aided Separate Source and Channel Coding (SSCC) benchmarkers, namely a learning-aided Elias Gamma (EG)-Convolutional Code (CC) scheme and a learning-aided Arithmetic-CC scheme, as well as the corresponding idealized but impractical versions of all schemes, in order to characterise the upper bounds on their performance.

6.1.3 Chapter Organisation

The rest of this chapter is organised as follows:

- In Section 6.2, we analyse the nature of non-stationary source symbol distributions. We extend the stationary zeta distribution model that was used in the previous chapters to create a non-stationary zeta distribution model of Section 3.2, which is inspired by the non-stationary nature of the symbols generated by the H.265 video codec.
- In Section 6.3, we detail the transmitter and receiver architecture of the proposed learning-aided UEC scheme, as well as detailing our proposed learning technique. We detail how the learning algorithm may be implemented by employing a memory storage at the receiver to infer the source distribution statistics, which improves the error correction capability by feeding these statistics back to the decoder.
- In Section 6.4, we propose a pair of SSCC benchmarkers that employ a similar learning technique, namely a learning-aided EG-CC scheme and a learning-aided Arithmetic-CC scheme, as well as their corresponding idealized but impractical versions.

- In Section 6.5, we compare the SER performance of the proposed learning-aided UEC scheme to those of the learning-aided EG-CC and the learning-aided Arithmetic-CC benchmarkers. The comparison is carried out by using different parameter sets for the non-stationary zeta distribution and using a common complexity limit that is sufficient for achieving iterative decoding convergence in all schemes.
- In Section 6.6, we conclude this chapter.

6.2 Nature of the Source

In this section, we introduce the source distributions considered in this paper, which are inspired by those of H.265. However, the particular distribution of the source symbols produced during H.265 encoding are sensitive to the specific selection of video encoding parameters and to the type of video sequences being encoded. Owing to this, we prefer to consider model source distributions, which can be readily parametrized to be representative of the distribution of the H.265 codec's entropy-encoded symbol values in various applications, as well as of a wide variety of other multimedia source distributions. We commence in Section 6.2.1 by introducing a stationary zeta distribution, which is inspired by the H.265 distribution. This is extended in Section 6.2.2, to conceive the non-stationary zeta source distribution that is applied throughout the rest of this paper. This non-stationary distribution is a direct consequence of the non-stationary nature of the H.265 distribution.

6.2.1 Stationary Zeta Distribution

As shown in Figure 6.4, the proposed learning-aided UEC scheme is designed to convey a sequence of successive symbol vectors, where each vector $\mathbf{x} = [x_i]_{i=1}^{a}$ can be obtained as the realization of a corresponding vector $\mathbf{X} = [X_i]_{i=1}^{a}$ of Independent and Identically Distributed (IID) Random Variable (RV). Each RV X_i adopts the symbol value $x \in \mathbb{N}_1$ according to the probability $\Pr(X_i = x) = P(x)$, where $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$ is the infinite-cardinality set comprising all positive integers. Here, the symbol entropy is given by

$$H_X = \sum_{x \in \mathbb{N}_1} H[P(x)], \tag{6.1}$$

where $H[p] = p \log_2(1/p)$ [92].

Figure 6.2 illustrates the logarithmically scaled probability distribution of the symbol values that are entropy encoded by the H.265 video encoder, for the case of a particular video encoder [164] parametrization and for a particular set of video sequences. The H.265 distribution of Figure 6.2 corresponds to a symbol entropy of $H_X = 2.348$ bits per symbol.

Note that these symbol values appear to obey Zipf's law [144], since the H.265 distribution may be approximated by the zeta distribution.

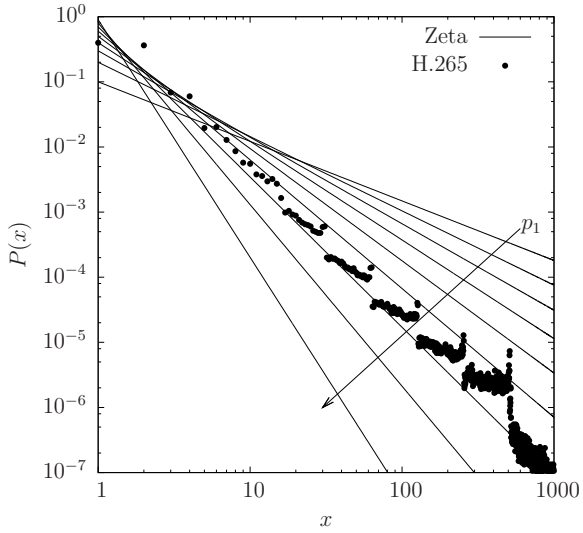

Figure 6.2: The zeta probability distributions for $p_1 \in \{0.1, 0.2, 0.3, \ldots, 0.9\}$, as well as the H.265 entropy-encoded symbol value distribution. This was obtained by recording the values of the symbols that are EG- and UnaryMax-encoded when the HM 13.0 H.265 video encoder employs the 'encoder_randomaccess_main.cfg' and 'encoder_lowdelay_main.cfg' configurations to encode the 112.9 million symbols that occur during the 220 seconds of video from the 24 video sequences that are commonly used in High Efficiency Video Coding (HEVC) [164]. By contrast, the H.264 distribution of Figure 3.2 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.

More specifically, the stationary zeta probability distribution [144] is defined as

$$P(x) = \frac{x^{-s}}{\zeta(s)}, \tag{6.2}$$

where $\zeta(s) = \sum_{x \in \mathbb{N}_1} x^{-s}$ is the Riemann zeta function and s parametrizes the distribution. Alternatively, the zeta distribution may be parametrized by $p_1 = \Pr(X_i = 1) = 1/\zeta(s)$, which is the occurrence probability of the most frequently encountered symbol value, namely of the symbol 1. Zeta distributions having the parameter of $p_1 \in \{0.1, 0.2, 0.3, \ldots, 0.9\}$ and the corresponding symbol entropy of $H_X \in \{17.458, 9.171, 6.104, 4.402, 3.267, 2.422, 1.740, 1.154, 0.612\}$ are depicted in Figure 6.2. Table 4.1 exemplifies the source symbol probabilities $P(x)$ for the case of a stationary zeta distribution, having the parameter of $p_1 = 0.797$, as was considered in our previous work [92]. This parametrization corresponds to a symbol entropy of 1.171 bits per symbol.

However, in contrast to the stationary zeta distributions that have been considered in our previous work, the H.265 source distribution is non-stationary, since it varies gradually with time, sometimes having a higher than average entropy and sometimes having a lower than average entropy. Motivated by this, we propose a non-stationary zeta distribution model in this paper, as detailed in Section 6.2.2.

6.2.2 Non-Stationary Zeta Distribution

In order to characterize how quickly the non-stationary source distribution varies in H.265, we segmented the sequence of 112.9 million entropy-encoded symbols characterized in Figure 6.2 into successive vectors having a fixed length of a symbols. Following this, we measured the symbol entropy in each vector using Eq. (6.1), in order to obtain a corresponding sequence of symbol entropies. Finally, we measure the autocorrelation of the consecutive pair of values in the entropy sequence, as plotted in Figure 6.3 as a function of the vector length a that quantified in terms of the number of the symbols. It may be seen that the largest first-order autocorrelation is achieved, when the vector length a is around 1000. Therefore, without loss of any generality, we assume that the symbol vector $\mathbf{x}$ of Figure 6.4 has a length of $a = 1000$ symbols, throughout the rest of this paper.

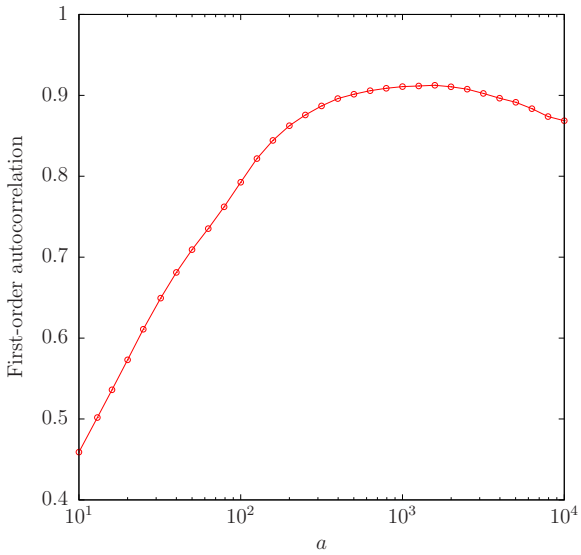

Figure 6.3: The first-order autocorrelation of symbol entropy sequence, when the 112.9 million H.265 entropy-encoded symbols are segmented into successive subvectors of length $a \in \{10, 10^{1.1}, 10^{1.2}, \ldots, 10^4\}$.

In contrast to the stationary zeta distribution of Section 6.2.1, the non-stationary zeta distribution produces successive symbol vectors $\mathbf{x}$ of $a = 1000$ symbols having a series of different but correlated p_1 values, as mentioned in the context of the autocorrelation results of Figure 6.3. This stream of correlated p_1 values is obtained by first generating an uncorrelated stream of Gaussian distributed random values. Note that this uncorrelated stream has a length of $a = 1000$. This random stream is then smoothened by using a

low-pass filter having a normalised cut-off frequency of $1/T$, where T is a parameter that specifies the number of successive symbol vectors $\mathbf{x}$ produced per cycle of variation among the correlated p_1 values. Following this, the mean and standard deviation of the filtered values are adjusted to be equal to the parameter values $\bar{p_1}$ and σ, respectively. Each successive value of p_1 from this stream may then be used to parametrize the zeta distribution of Eq. (6.2), which is used to generate each successive vector $\mathbf{x}$ of $a = 1000$ source symbols.

As listed in Table 6.1, the analysis of the following sections will consider seven different sets of parametrizations. In set (a) for example, the low pass filter is parametrized by $T = 40$ successive symbol vectors $\mathbf{x}$ per p_1 cycle, by a mean of $\bar{p_1} = 0.8$ and by a standard deviation of $\sigma = 1/30$, resulting in 99.7% of the non-stationary p_1 values falling in the range of $[0.7, 0.9]$. Note however that the proposed learning-aided UEC scheme may be applied to arbitrary non-stationary source distributions, without being limited to those adhering to the model described in this section.

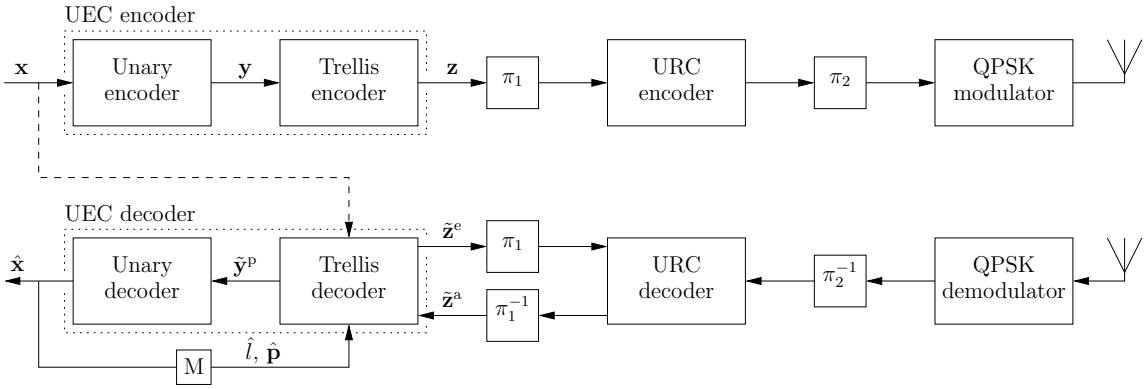

Figure 6.4: Schematic of the proposed learning-aided UEC scheme, in which a UEC code is serially concatenated with a URC code and Gray-mapped QPSK modulation. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. Block M represents the memory storage that is used to store the statistics observed from successive recovered symbol vectors $\hat{\mathbf{x}}$. Bold notation without a diacritic is used to denote a symbol or bit vector. A diacritical tilde represents an LLR vector pertaining to the bit vector with the corresponding notation. The superscripts 'a', 'e' and 'p' denote *a priori*, extrinsic and *a posteriori* LLRs, respectively. By contrast to the UEC-IrURC scheme of Figure 3.3, the irregular URC code is replaced by a regular URC code.

6.3 Learning-aided UEC Coding

The proposed learning-aided UEC scheme of Figure 6.4 performs the JSCC encoding and decoding of successive symbol vectors $\mathbf{x}$, in which the symbol values are selected from a set having an infinite cardinality, as described in Section 6.2. Like conventional UEC coding, the proposed learning-aided UEC scheme does not require any knowledge of the

symbol occurrence probabilities at the transmitter. However, in contrast to conventional UEC coding, the proposed scheme does not require this knowledge at the receiver either, since it can gradually estimate the source probability distribution from the recovered symbols. Note that since our focus is the learning algorithm of the UEC code, a simple URC code is selected for the inner code. In Section 6.3.1 and 6.3.2, we will introduce the operations of the transmitter and the receiver, respectively. Following this, Section 6.3.3 will discuss the operation of the proposed learning mechanism.

6.3.1 Transmitter Operation

Similar to the UEC encoder of Figure 3.3 in Section 3.3.1, the learning-aided UEC scheme encodes the source vector $\mathbf{x} = [x_i]_{i=1}^{a}$ using a unary encoder, as shown in Figure 6.4. Each symbol x_i in the vector $\mathbf{x}$ is firstly represented by the corresponding codeword $\mathbf{y}_i$ that comprises x_i bits, namely $(x_i - 1)$ logical one-valued bits followed by a single logical zero-valued bit, as exemplified in Table 4.1. Note that the average codeword length l of $\mathbf{y}_i$ of Figure 6.4 is given by Eq. (3.5). For example, the source vector $\mathbf{x} = [1, 4, 2, 1, 1, 3, 1, 2]$ of $a = 8$ symbols yields the $b = 15$-bit vector $\mathbf{y} = [011101000110010]$. Note that the average length of the bit vector $\mathbf{y}$ of Figure 6.4 is given by $a \cdot l$.

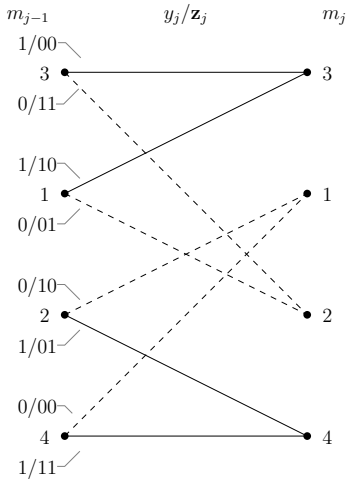

Figure 6.5: An $r = 4$-state $n = 2$-bit UEC trellis, having codeword $\mathbb{C} = \{01, 11\}$. This trellis is as same as the one of Figure 3.5.

Following unary encoding, the trellis encoder of Figure 6.4 is employed for encoding the bit vector $\mathbf{y}$. Figure 3.4 illustrates the generalized r-state UEC trellis, while Figure 6.5 of this chapter exemplifies an $r = 4$-state UEC trellis. As discussed in Section 3.3.2, 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 $m_{j-1} \in \{1, 2, \ldots, r\}$ to its next state $m_j \in \{1, 2, \ldots, r\}$, in the order of the increasing bit-index j. Each next state m_j is selected from two legitimate alternatives, depending on the bit value y_j, according to Eq. (3.8).

In this way, the bit vector $\mathbf{y}$ identifies a path through the trellis, which may be represented by a vector $\mathbf{m} = [m]_{j=0}^{b}$ comprising $(b+1)$ state values. For example, the bit vector $\mathbf{y} = [011101000110010]$ yields the path $\mathbf{m} = [1, 2, 4, 4, 4, 1, 3, 2, 1, 2, 4, 4, 1, 2, 4, 1]$ through the $r = 4$-state trellis of Figure 6.5. Note that the trellis path $\mathbf{m}$ through the trellis of Figure 6.5 remains synchronised with the unary codewords, since the zero-valued bit at the end of each codeword $\mathbf{y}_i$ returns the path $\mathbf{m}$ to either state 1 or state 2, depending on whether the codeword $\mathbf{y}_i$ represents a symbol x_i having an odd or even index i. 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 Eq. (3.11).

The trellis encoder represents each bit y_j in the vector $\mathbf{y}$ by an n-bit codeword $\mathbf{z_j}$. This 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 $\overline{\mathbb{C}} = \{\overline{\mathbf{c}_1}, \overline{\mathbf{c}_2}, \ldots, \overline{\mathbf{c}_{r/2-1}}, \overline{\mathbf{c}_{r/2}}\}$, which is achieved according to Eq. (3.12). For example, the $n = 2$-bit codewords $\mathbb{C} = \{01, 11\}$ are employed in the $r = 4$-state UEC trellis of Figure 6.5. Finally, the selected codewords are concatenated to obtain the $(b \cdot n)$-bit vector $\mathbf{z} = [z_k]_{k=1}^{bn}$ of Figure 6.4. For example, the path $\mathbf{m} = [1, 2, 4, 4, 4, 1, 3, 2, 1, 2, 4, 4, 1, 2, 4, 1]$ through the $r = 4$-state $n = 2$-bit trellis of Figure 6.5 corresponds to the encoded bit vector $\mathbf{z} = [01011111001011100101110001010 0]$. Note that the UEC encoder does not require any knowledge of the source distribution, since the output bit vector $\mathbf{z}$ of the UEC encoder only depends on the symbol vector $\mathbf{x}$ and the parametrization of the UEC trellis. This is true for both the conventional UEC scheme and for our proposed learning-aided UEC scheme.

When the source distribution is stationary, the overall average coding rate R_o of the UEC encoder is given by $R_\mathrm{o} = \dfrac{H_X}{nl}$. However, if the source distribution is non-stationary, then the symbol entropy H_X and the average unary codeword length l will vary from frame to frame. In this case, the average UEC coding rate R_o is given by the expectation

$$R_\mathrm{o} = E\left\{\frac{H_X}{nl}\right\}, \tag{6.3}$$

which may be estimated experimentally.

As shown in Figure 6.4, the UEC-encoded bit vector $\mathbf{z}$ is interleaved in the block π_1, encoded by the Unity-Rate Convolutional (URC) encoder and then interleaved again by the block π_2. Here, we recommend a 2-state URC having the generator polynomial of $[1, 0]$ and the feedback polynomial of $[1, 1]$, as characterized in Figure 9.6 of [98]. Following this, Gray-mapped Quadrature Phase-Shift Keying (QPSK) modulation may be employed for transmission, as shown in Figure 6.4. Consequently, the effective throughput is given

by $\eta = R_{\rm o} \cdot R_{\rm i} \cdot \log_2(M)$ bits per symbol, where we have $R_{\rm i} = 1$ for the URC coding rate and $M = 4$ for the QPSK modulation order.

6.3.2 Receiver Operation

In the receiver of Figure 6.4, Gray-mapped QPSK demodulation is followed by deinterleaving in the block π_2^{-1}, before commencing iterative information exchange between the URC and UEC decoders. Here, the two decoders exchange their vectors of LLRs, which are interleaved and deinterleaved in the blocks π_1 and π_1^{-1}, respectively. Both of these decoders apply the Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm [91] to their respective trellises, where the UEC trellis decoder may employ the trellis of Figure 6.5, which has only a modest complexity.

As shown in Figure 6.4, the UEC trellis decoder is provided with a vector of *a priori* LLRs $\tilde{\mathbf{z}}^{\rm a} = [\tilde{z}_k^{\rm a}]_{k=1}^{bn}$ that pertain to the corresponding bits in the vector $\mathbf{z}$. These *a priori* LLRs are used for generating the vector of extrinsic LLRs $\tilde{\mathbf{z}}^{\rm e} = [\tilde{z}_k^{\rm e}]_{k=1}^{bn}$, which also pertain to the corresponding bits in the vector $\mathbf{z}$. Here, the value of bn 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. The BCJR algorithm can exploit the synchronization between the UEC trellis and the unary codewords, in order to improve the receiver's error correction capability and to facilitate near-capacity operation. This is achieved by including the conditional transition probability $\Pr(M_j = m | M_{j-1} = m') = P(m|m')$ as an additional term during the BCJR algorithm's γ_t calculation of Eq. (2.11), where we have

$$P(m|m') = \frac{P(m, m')}{\sum_{\check{m}=1}^{r} P(\check{m}, m')}, \tag{6.4}$$

and $P(m, m')$ is given in Eq. (3.11), which depends on the symbol probability distribution $P(x)$ as described in Section 6.3.1.

Note that knowledge of the entire symbol probability distribution $P(x)$ is not required in order to exploit Eq. (6.4). Rather, the only knowledge required is that of the average unary codeword length l and the probabilities of the first $r/2 - 1$ symbol values $\mathbf{p} = [P(x)]_{x=1}^{r/2-1}$ [92]. In the conventional UEC scheme of [92], the average length l and the probability vector $\mathbf{p}$ are assumed to be stationary and known at the receiver, as represented by the dashed line in Figure 6.4. However, since $P(x)$ is non-stationary and unknown at the receiver of the proposed learning-aided UEC scheme, it must estimate l and $\mathbf{p}$ heuristically and iteratively, before this information can be exploited by Eq. (6.4). The mechanism proposed for this learning process will be discussed in Section 6.3.3. In the absence of this information, the $\Pr(M_j = m | M_{j-1} = m')$ term can be simply omitted from the γ_t calculation of Eq. (2.11), at the cost of degrading the receiver's error correction capability.

In each decoding iteration, the UEC trellis decoder may also invoke the 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 $\mathbf{y}$. The unary decoder of Figure 6.4 exploits the observation that each of the a unary codewords in the vector $\mathbf{y}$ contains only a single logical zero-valued bit. This is achieved by sorting the *a posteriori* LLRs in the 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 zero. A hard decision vector $\hat{\mathbf{y}}$ is then obtained by setting the value of these bits to zero and the value of all other bits to one. Finally, the bit vector $\hat{\mathbf{y}}$ can be unary decoded in order to obtain the symbol vector $\hat{\mathbf{x}}$ of Figure 6.4, which is guaranteed to comprise a number of symbols owing to the above-described technique. Note that the value of a is assumed to be known to the receiver. In practice, this may be achieved by either using a constant value for a that is hard-coded into the receiver or by reliably conveying the value of a from the transmitter to the receiver using a small amount of side information. The iterative exchange of LLRs between the UEC and URC decoders of Figure 6.4 continues until a particular number of iterations has been completed or until the correctly decoded symbol vector $\hat{\mathbf{x}}$ has been obtained, which may be detected using a Cyclic Redundancy Check (CRC) code in practice, for example.

6.3.3 Learning Algorithm

As discussed in Section 6.3.2, the only knowledge that the UEC trellis decoder requires in order to facilitate near-capacity operation is the average unary codeword length l and the probabilities of the first $r/2-1$ symbol values $\mathbf{p} = [P(x)]_{x=1}^{r/2-1}$. When the probability distribution $P(x)$ of source symbols $\mathbf{x}$ is non-stationary and unknown, our learning-aided UEC scheme is capable of heuristically and iteratively estimating l and $\mathbf{p}$ from the recovered symbol vectors $\hat{\mathbf{x}}$, which are then back to the trellis decoder as *a priori* information, in order to improve the receiver's error correction capability.

The estimation is implemented using the memory storage block labeled M in Figure 6.4. This memory storage is used to store source distribution statistics that have been observed from successively recovered symbol vectors $\hat{\mathbf{x}}$. We quantify the size of this memory storage in terms of the number M of the most-recently recovered symbol vectors $\hat{\mathbf{x}}$, from which the statistics are derived. As described in Section 6.3.2, the number a of symbols in each vector $\mathbf{x}$ and the number b of bits in each vector $\mathbf{y}$ are known to the decoder. Owing to this, following the decoding of each frame, the average unary codeword length l can be estimated by

$$\hat{l} = \frac{\sum_{m=1}^{\mathrm{M}} b_m}{\sum_{m=1}^{\mathrm{M}} a_m}, \tag{6.5}$$

which can then be used for assisting the decoding of the next frame. Here, a_m and b_m are the number of symbols and the number of bits in the m-th most-recently recovered frame, respectively. In this way, we can also count the occurrences of the first $r/2-1$ symbol values in each recovered symbol vector $\hat{\mathbf{x}}$. Therefore, following the decoding of each frame, the probability vector $\mathbf{p}$ can be estimated by

$$\hat{\mathbf{p}} = \left[\frac{\sum_{m=1}^{\mathrm{M}} N(\hat{\mathbf{x}}_m = x)}{\sum_{m=1}^{\mathrm{M}} a_m} \right]_{x=1}^{r/2-1}, \tag{6.6}$$

which can then also be used for assisting the decoding of the next frame. Here, $N(\hat{\mathbf{x}}_m = x)$ denotes the number of recovered symbols that has a value of x in the m-th most-recently recovered symbol vector $\hat{\mathbf{x}}_m$. Note that the memory will be empty during the decoding of the first frame. In this case, $\hat{l}$ and $\hat{\mathbf{p}}$ are not exploited during UEC decoding due to omitting the $\Pr(M_j = m|M_{j-1} = m')$ term from the γ_t calculation of Eq. (2.11), as described in Section 6.3.2. Following this, since at this stage M frames have not yet been received, the summations in Eq. (6.5) and Eq. (6.6) are adjusted accordingly.

Note that during this initial transient phase, the estimated $\hat{l}$ and $\hat{\mathbf{p}}$ may not have accurate values, particularly if transmission errors occur owing to encountering a low SNR. Nevertheless, this imperfect *a priori* information can still contribute towards improving the error correction capability of the trellis decoder, gradually resulting in fewer errors in $\hat{\mathbf{x}}$ and more reliable feedback of $\hat{l}$ and $\hat{\mathbf{p}}$ for the next frame. In this way, our learning-aided UEC scheme becomes capable of gradually learning the statistics of the source distribution, hence iteratively updating the estimated $\hat{l}$ and $\hat{\mathbf{p}}$ on a frame-by-frame basis. As a result, the performance of the decoder can be gradually improved until a steady-state phase is reached, where upon near-capacity operation is achieved.

For the stationary probability distribution, it is clear that the more memory storage is applied, the more accurate source distribution statistics can be obtained. In this case, the difference between the estimated values of $\hat{l}$ as well as $\hat{\mathbf{p}}$ and the real values of l as well as $\mathbf{p}$ will become infinitesimal, when an infinite memory is applied, giving

$$\lim_{\mathrm{M}\to\infty} \hat{l} = l \quad \text{and} \quad \lim_{\mathrm{M}\to\infty} \hat{\mathbf{p}} = \mathbf{p}. \tag{6.7}$$

However, in the case of non-stationary source distributions, it is not desirable to apply an infinite memory storage. This is not only because of the impracticality of infinite memory, but also because the source distribution of the current frame is only correlated to the recent frames, when the source is non-stationary. If the size of the memory storage is excessive, then the estimates $\hat{l}$ and $\hat{\mathbf{p}}$ will be contaminated by out-of-date source distribution statistics,

hence degrading the error correction capability of the scheme. Therefore, there is a trade-off between collecting sufficient statistics for accurate estimation and collecting too many out-of-date statistics. In the simulations of Section 6.4, we will investigate this trade-off, in order to optimize the overall performance of our proposed learning-aided scheme. We will also consider an idealized but impractical version of the proposed learning-aided UEC scheme, in which a 'genie' provides the receiver with perfect knowledge of l and $\mathbf{p}$ for each frame. As a result, this version provides a baseline, which characterizes the bound on the performance that the learning-aided UEC scheme is capable of achieving.

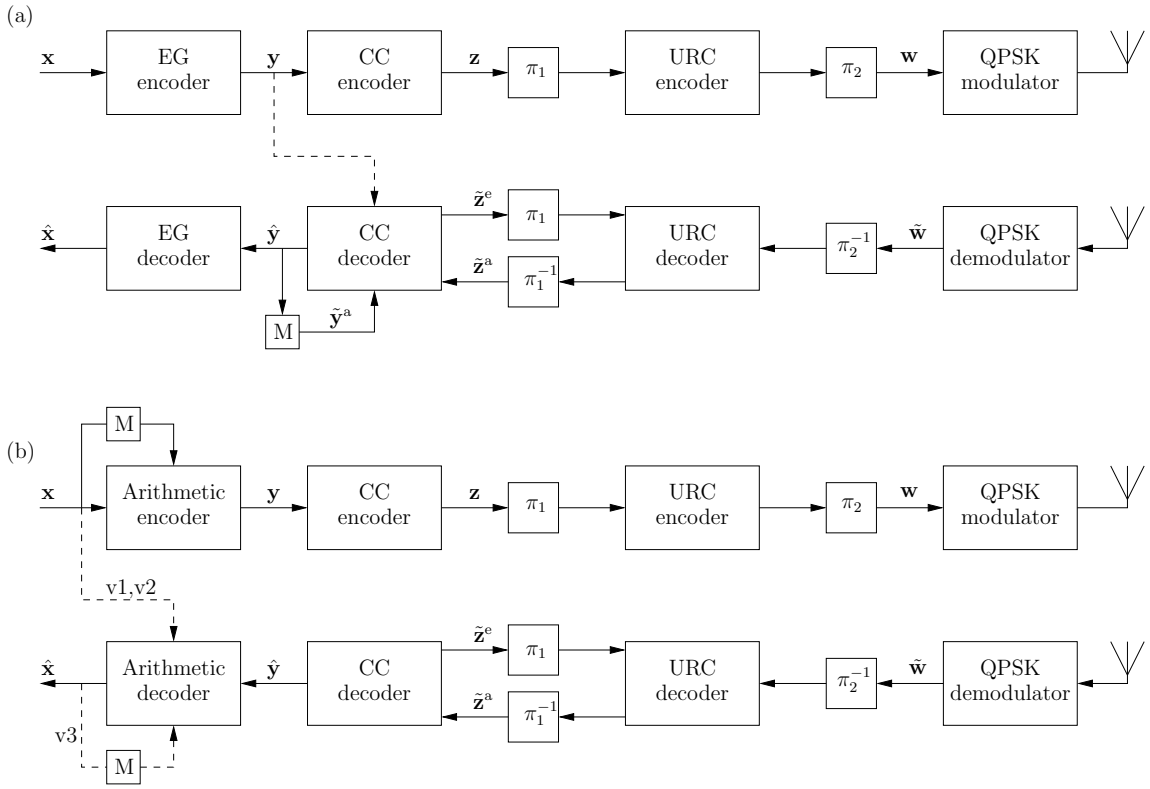

Figure 6.6: Schematics of (a) the learning-aided EG-CC and (b) the learning-aided Arithmetic-CC benchmarkers, which employ serial concatenation with a URC code and a Gray-mapped QPSK modulation scheme. Here, π_1 and π_2 represent interleavers, while π_1^{-1} and π_2^{-1} represent the corresponding deinterleavers. For the sake of controlling the effective throughput η, doping or puncturing may be performed by π_2. Block M represents the memory that is used to store the statistics observed from successive recovered bit vectors $\hat{\mathbf{y}}$ or symbol vectors $\hat{\mathbf{x}}$. By contrast to the learning-aided UEC-URC scheme of Figure 6.4, the UEC scheme is replaced by EG-CC and Arithmetic-CC schemes, respectively.

6.4 Benchmarkers

In this section, we compare the proposed learning-aided UEC scheme to a pair of SSCC benchmarkers that employ a similar learning strategy, as well as to the corresponding idealized but impractical versions of all schemes. In the learning-aided EG-CC benchmarker

of Figure 6.6(a), the unary code of Figure 6.4 is replaced by an EG code and the UEC trellis code is replaced by a CC code. As a further step, the learning-aided Arithmetic-CC benchmarker is obtained by replacing the EG code by an arithmetic code, as shown in Figure 6.6(b). The design and parametrizations of the two benchmarkers are detailed in Section 6.4.1 and Section 6.4.2, respectively.

6.4.1 Learning-aided EG-CC Benchmarker

In the transmitter of the learning-aided EG-CC scheme, an EG encoder is invoked for converting the symbol vector $\mathbf{x}$ into the bit vector $\mathbf{y}$, which typically has non-equiprobable bit values. Here, the first ten codewords of the EG code are given in Table 4.1. When the source distribution is stationary, the average EG codeword length is given by $l = \sum_{x \in \mathbb{N}_1} P(x) \left(2\lfloor \log_2(x) \rfloor + 1\right)$. However, when the source distribution is non-stationary, the average EG codeword length will vary from frame to frame, as described in Section 6.3.1. Following EG encoding, $\mathbf{y}$ is CC encoded to obtain the bit vector $\mathbf{z}$, as shown in Figure 6.6(a). The CC encoder may be described by the generator and feedback polynomials provided in Table 3.2. Here, we employ the $n = 2$-bit $r = 4$-state CC code, in order to facilitate a fair comparison with the scenario where the trellis encoder of the learning-aided UEC scheme employs the UEC trellis of Figure 6.5.

In the case of a non-stationary source distribution, the EG-CC coding rate R_{o} may be quantified by the expectation of Eq. (6.3) and will typically differ from that of the UEC scheme. For the sake of fair comparisons, either the technique of doping [165] or puncturing may be applied in the block π_2 of Figure 6.6 in order to achieve the same effective throughput η for all schemes. Explicitly doping is applied when the coding rate R_{o} of EG-CC encoding is higher than that of UEC encoding, while puncturing is applied, when the coding rate of EG-CC encoding is lower. More particularly, in the doping operation of π_2, a certain number of bits from the tail of the interleaved bit vector $\mathbf{w}$ are duplicated and appended to the end of $\mathbf{w}$. In the receiver, the 'de-doping' operation of π_2^{-1} is constituted by removing the corresponding LLRs of the vector $\tilde{\mathbf{w}}$, before adding them into the LLRs that now form the end of $\tilde{\mathbf{w}}$. By contrast, when puncturing is applied in π_2, a certain number of bits are truncated from the tail of the bit vector $\mathbf{w}$. In the corresponding 'de-puncturing' operation of π_2^{-1}, the positions in the Logarithmic Likelihood Ratio (LLR) vector $\tilde{\mathbf{w}}$ that correspond to the truncated bits are filled by zero-valued LLRs. The inner coding rate R_{i} is defined as the ratio of the number of bits entering π_2 to the number of bits emerging from π_2, where $R_{\text{i}} > 1$ for puncturing as $R_{\text{i}} < 1$ for doping, as listed in Table 6.1.

During iterative decoding, the CC decoder chooses the BCJR algorithm for converting the vector of *a priori* LLRs $\tilde{\mathbf{z}}^{\text{a}}$ into the vector of extrinsic LLRs $\tilde{\mathbf{z}}^{\text{e}}$, and it employs the Viterbi algorithm [3] for converting $\tilde{\mathbf{z}}^{\text{a}}$ into the vector of recovered bits $\hat{\mathbf{y}}$, which is then

Set	T	$\bar{p_1}$	σ	Scheme	Figure	R_o	R_i	η	E_b/N_0 [dB] capacity bound	E_b/N_0 [dB] area bound	E_b/N_0 [dB] tunnel bound	Complexity
a	40	0.8	1/30	UEC	6.4	0.3726	1	0.7452	0.767	1.26	2.5	189
				EG-CC	6.6(a)	0.3765	0.9897			1.95	3.0	186
				Arith-CC	6.6(b)	0.5000	0.7452			1.78	2.6	141
b	40	0.85	1/30	UEC	6.4	0.3330	1	0.6660	0.485	0.87	2.3	160
				EG-CC	6.6(a)	0.3201	1.0401			1.63	3.3	166
				Arith-CC	6.6(b)	0.5000	0.6660			1.60	2.4	105
c	40	0.75	1/30	UEC	6.4	0.3626	1	0.7252	0.695	1.58	2.7	245
				EG-CC	6.6(a)	0.4198	0.8638			2.61	3.1	209
				Arith-CC	6.6(b)	0.5000	0.7252			1.90	2.7	176
d	20	0.8	1/30	UEC	6.4	0.3727	1	0.7454	0.776	1.31	2.5	189
				EG-CC	6.6(a)	0.3786	0.9845			1.98	3.1	186
				Arith-CC	6.6(b)	0.5000	0.7454			1.78	2.6	141
e	160	0.8	1/30	UEC	6.4	0.3718	1	0.7436	0.757	1.27	2.5	188
				EG-CC	6.6(a)	0.3732	0.9962			1.98	3.1	186
				Arith-CC	6.6(b)	0.5000	0.7436			1.69	2.6	139
f	40	0.8	1/60	UEC	6.4	0.3767	1	0.7534	0.810	1.32	2.5	185
				EG-CC	6.6(a)	0.3759	1.0020			1.95	3.1	186
				Arith-CC	6.6(b)	0.5000	0.7534			1.73	2.5	139
g	40	0.8	1/20	UEC	6.4	0.3621	1	0.7242	0.681	1.18	2.4	195
				EG-CC	6.6(a)	0.3762	0.9626			2.02	3.1	187
				Arith-CC	6.6(b)	0.5000	0.7242			1.60	2.5	142

Table 6.1: The parametrization of the seven sets of non-stationary source distribution parameters, including frames per cycle T, as well as mean $\bar{p_1}$ and standard deviation σ of the zeta distribution parameter p_1. Characteristics are provided for the various schemes considered, including outer coding rate R_o, inner coding rate R_i and effective throughput η. E_b/N_0 bounds are given for the case of Gray-mapped QPSK transmission over an uncorrelated narrowband Rayleigh fading channel. Complexity is quantified by the average number of ACS operations incurred per symbol in the vector $\mathbf{x}$.

EG decoded in order to obtain the recovered symbol vector $\hat{\mathbf{x}}$. Note that in the receiver of the learning-aided EG-CC scheme, the CC decoder is unable to exploit knowledge of the average codeword length l or that of the symbol probability distribution $P(x)$, like the UEC trellis decoder. However, the BCJR algorithm employed by the CC decoder is able to exploit the knowledge of the probability of occurrence of the binary values in the bit vector $\mathbf{y}$ output by the EG encoder. More specifically, the CC decoder is provided with a vector $\tilde{\mathbf{y}}^{\mathrm{a}} = [y_j^{\mathrm{a}}]_{j=1}^{b}$ of *a priori* LLRs, having identical values. As shown in Figure 6.6(a), we consider two different versions of the learning-aided EG-CC benchmarker, depending on how the value used for all LLRs in the vector $\tilde{\mathbf{y}}^{\mathrm{a}}$ is obtained.

- 1. Idealized but impractical *a priori* information
  As indicated by the dashed line in Figure 6.6(a), the first version of the learning-aided EG-CC benchmarker relies on an impractical genie to provide the receiver with perfect knowledge of the bit value probabilities of each frame, which can be used to provide the idealized *a priori* LLR vector $\tilde{\mathbf{y}}^{\mathrm{a}}$. As a result, this version provides a baseline, which characterizes the bound on the performance that the learning-aided EG-CC benchmarker can achieve.
- 2. Estimated *a priori* information
  Based on the learning technique of Section 6.3.3, the second version of the learning-aided EG-CC benchmarker employs memory M at the receiver to store the bit probability statistics obtained from the M most-recently recovered bit vectors $\hat{\mathbf{y}}$, as shown in Figure 6.6(a). In this way, the *a priori* LLRs can be estimated as

$$\tilde{\mathbf{y}}^{\mathrm{a}} = \ln \frac{\sum_{m=1}^{\mathrm{M}} w(\hat{\mathbf{y}}_m)}{\sum_{m=1}^{\mathrm{M}} [l(\hat{\mathbf{y}}_m) - w(\hat{\mathbf{y}}_m)]}, \tag{6.8}$$

  where $w(\hat{\mathbf{y}}_m)$ and $l(\hat{\mathbf{y}}_m)$ are the weight and length of the m-th most-recently recovered bit vector $\hat{\mathbf{y}}_m$, respectively. As in the learning-aided UEC scheme, the number of frames M considered by the memory storage may be optimized in the case of non-stationary source probability distributions. Note that during the transient phase, the memory and the *a priori* LLR vector $\tilde{\mathbf{y}}^{\mathrm{a}}$ are operated in analogy with the technique employed by the proposed learning-aided UEC scheme during the learning phase.

6.4.2 Learning-aided Arithmetic-CC Benchmarker

As shown in Figure 6.6(b), the learning-aided Arithmetic-CC benchmarker can be obtained by replacing the EG code of Figure 6.6(a) by an arithmetic code. Both arithmetic encoding and decoding require knowledge of the entire source symbol probability distribution. However, this corresponds to an infinite amount of knowledge, when the cardinality

of the source alphabet is infinite. In order to overcome this problem, the learning-aided Arithmetic-CC scheme clips all symbol values in the vector $\mathbf{x}$ to a limit of 1000, before they are arithmetic encoded. This clipping leads to some arithmetic decoding errors, since the receiver will recover the clipped value, rather than the correct value of each symbol. However, this effect is small, since symbol values exceeding 1000 are extremely rare in the source distributions considered in Figure 6.2. In case of non-stationary source distributions, the arithmetic encoder and decoder's knowledge of the source distribution must be adaptively updated, in order to reflect the time-variant statistics of the source. As shown in Figure 6.6(b), depending on how this knowledge is updated and whether synchronization between the transmitter and the receiver is assumed, we consider three different versions of the learning-aided Arithmetic-CC scheme, as follows.

- v1. Idealized but impractical knowledge of the source distribution and idealized but impractical synchronization

  As indicated by the dashed line in Figure 6.6(b), the first version of the learning-aided Arithmetic-CC benchmarker relies on an impractical genie to provide perfect knowledge of the source distribution of each frame to both the transmitter and receiver, guaranteeing perfect synchronization between them. As a result, this version provides a baseline, which characterizes the bound on the performance that the learning-aided Arithmetic-CC scheme can only theoretically achieve.
- v2. Estimated knowledge of the source distribution and idealized but impractical synchronization

  Based on the learning technique of Section 6.3.3, the second version of the learning Arithmetic-CC benchmarker employs the memory M of Figure 6.6(b) at the transmitter for storing all the symbol probability statistics estimated from the M previous symbol vectors $\mathbf{x}$, as shown in Figure 6.6(b). Therefore, in analogy to Eq. (6.6), the symbol occurrence probabilities can be estimated as

$$\hat{\mathbf{p}} = \left[\frac{1 + \sum_{m=1}^{\mathrm{M}} N(\hat{\mathbf{x}}_m = x)}{1000 + \sum_{m=1}^{\mathrm{M}} a_m} \right]_{x=1}^{1000}. \tag{6.9}$$

  Here, the addition of 1 in the numerator of Eq. (6.9) ensures that no symbols are attributed an estimated probability of zero and naturally allows a uniform probability distribution to be assumed when encoding the first frame, for which the memory M of Figure 6.6(b) is empty. During the subsequent learning phase, the memory is operated in analogy with the learning-aided UEC scheme. This version of the

learning-aided Arithmetic-CC benchmarker relies on an impractical genie conveying the estimated symbol probabilities $\hat{\mathbf{p}}$ from the transmitter to the receiver, guaranteeing perfect synchronization.

- v3. Estimated knowledge of the source distribution and imperfect but practical synchronization
  In this practical version of the learning-aided Arithmetic-CC benchmarker, the learning process of Eq. (6.9) is performed independently in the transmitter and receiver, based on the symbol vector $\mathbf{x}$ and the recovered symbol vector $\hat{\mathbf{x}}$, respectively. This is achieved using independent memories M in the transmitter and receiver, as shown in Figure 6.6(b). Owing to this, synchronization between the memory continues storing the estimated symbol probabilities is not guaranteed in the presence of transmission errors, which may cause $\hat{\mathbf{x}}$ and $\mathbf{x}$ to differ.

In the case of a non-stationary source distribution, the arithmetic coding rate R_{o} may be quantified in analogy to Eq. (6.3), once the steady state has been reached. In order to facilitate fair comparisons, doping or puncturing may be applied in the learning-aided Arithmetic-CC scheme, in order to achieve the same steady-state effective throughput η as the learning-aided UEC and EG-CC schemes.

6.5 Simulation Results

In this section, we compare the steady-state SER performance of the proposed learning-aided UEC scheme to those of the learning-aided EG-CC and the learning-aided Arithmetic-CC benchmarkers of Figure 6.6. In each of Figures 6.7(a) to 6(g), the source symbol vector $\mathbf{x}$ obeys the non-stationary zeta distribution of Section 6.2.2, which is parametrized by the corresponding set shown in Table 6.1. For example, the simulation results of Figure 6.7(a) correspond to the parameter set (a), for which the parameters $T = 40$, $\bar{p_1} = 0.8$ and $\sigma = 1/30$ are employed to generate a sequence of p_1 values, as described in Section 6.2.2. Each of those p_1 values is used for generating a symbol vector $\mathbf{x}$ comprising $a = 10^3$ symbols, which is then encoded by the various schemes considered. Transmission is performed over a Gray-mapped QPSK-modulated uncorrelated narrowband Rayleigh fading channel. Table 6.1 quantifies the number of Add, Compare and Select (ACS) operations performed by the BCJR algorithm of each scheme per symbol of $\mathbf{x}$ [96]. In Figure 6.7, all SER results are obtained using a limit of 7500 ACS operations per symbol of $\mathbf{x}$, which is sufficient for achieving iterative decoding convergence in all schemes, and facilitates fair comparisons in terms of decoding complexity.

As described in Sections 6.4.1 and 6.4.2, we employ doping and puncturing in order to obtain the same overall effective throughput η for all schemes employing the same

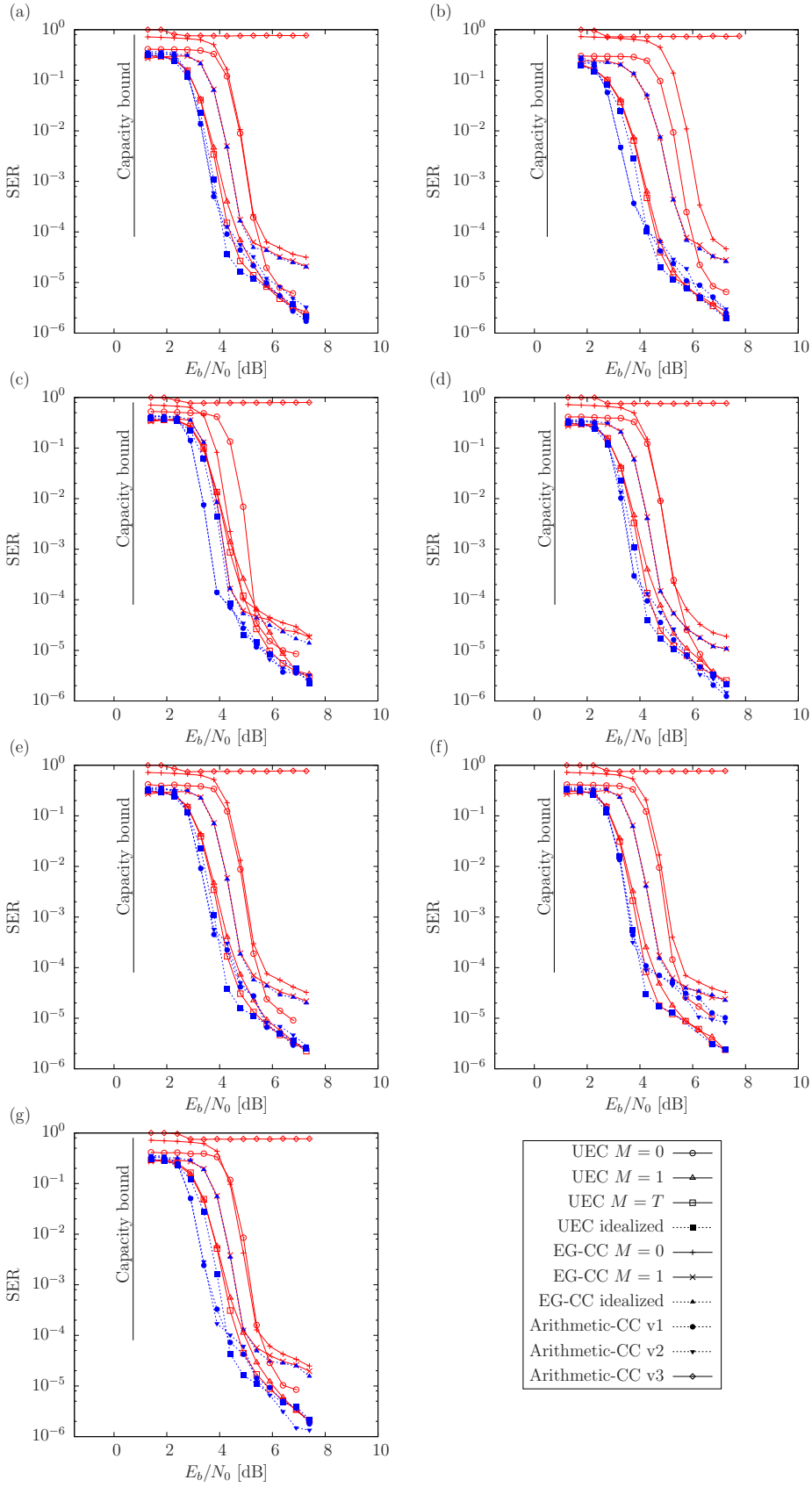

Figure 6.7: SER performance for various arrangements of the proposed learning-aided UEC scheme of Figure 6.4, the learning-aided EG-CC benchmarker of Figure 6.6(a) and the learning-aided Arithmetic-CC benchmarker of Figure 6.6(b), when conveying symbols obeying a non-stationary zeta distribution that is generated by the corresponding parameter sets (a) to (g) of Table 6.1, and communicating over a QPSK-modulated uncorrelated narrowband Rayleigh fading channel. A complexity limit of 7500 ACS operations is imposed for decoding each of the symbols in $\mathbf{x}$.

source distribution parameter set. The ninth column of Table 6.1 provides the specific E_b/N_0 values, where the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity becomes equal to the effective throughput η of each scheme considered. These E_b/N_0 values represent the *capacity bound*, above which it is theoretically possible to achieve reliable communication. For example, the parameter set (a) of Table 6.1, results in an effective throughput of $\eta = 0.7452$ bit/s/Hz for all the schemes considered, yielding a corresponding DCMC capacity bound of 0.767 dB. This bound is represented by a vertical line in Figure 6.7(a). However, in order to facilitate the creation of an open EXIT chart tunnel, it is necessary, but not sufficient, for the area A_o beneath the inverted outer EXIT function to exceed the area A_i beneath the inner EXIT function [132]. Therefore, the *area bound* provides the E_b/N_0 value, where we have $A_o = A_i$, which would theoretically allow the creation of an open EXIT chart tunnel [134]. Depending on how well the EXIT functions match each other, a narrow but open EXIT chart tunnel can only be created at a specific E_b/N_0 value, which we refer to as the *tunnel bound*. Based on these observations, the E_b/N_0 bounds may be used for characterizing the iterative decoding performance of our proposed scheme and the benchmarkers.

Furthermore, for the practical versions of our schemes that employ the memories shown in Figure 6.4 and 6.6, we consider different memory sizes M used for each parameter set. More specifically, Figure 6.7 considers the case of $M = 0$, where no memory is used at the receiver and hence the decoders never have any knowledge of the source symbol distribution. When $M = 1$, only the statistics derived from the previous recovered symbol vector $\hat{\mathbf{x}}$ are stored in the memory, providing the decoder with only limited knowledge about the source distribution. Despite this, the learning-aided UEC scheme benefits from an E_b/N_0 gain of up to 1 dB, when employing $M = 1$ instead of $M = 0$. As described in Section 6.3.3, the size M of the memory expressed in terms of the number of entropy-encoded symbols can be optimized to suit the nature of the non-stationary source. Without loss of generality and for the sake of simplicity, Figure 6.7 provides SER results for the proposed learning-aided UEC scheme, where the size of the memory M is set to the number of frames per cycle T, which parametrizes the non-stationary source distribution.

Our experiments reveal that the resultant SER performance is close to that offered by optimizing M. However, the optimized value of M may be adjusted accordingly in practice. As shown in Figure 6.7, the SER performance of the practical learning-aided UEC scheme having the optimized memory size M offers a performance that is within 0.1 dB from the idealized but impractical UEC scheme, at the SER of 10^{-3}. By contrast, the practical learning-aided EG-CC scheme only requires $M = 1$ in order to achieve the same performance as its idealized but impractical baseline. This is because the learning-aided EG-CC scheme only has to estimate a single source distribution, namely the value

of the LLR that is used for all elements of the vector $\tilde{\mathbf{y}}^{\mathrm{a}}$ provided to the CC decoder. Nevertheless, our proposed $M = 1$ based learning-aided UEC scheme offers up to 0.8 dB gain compared to the $M = 1$ learning-aided EG-CC scheme at the SER of 10^{-3}, and gains of up to 0.85 dB when employing the value of $M = T$. This may explained by the capacity loss that is incurred by the SSCC used in the learning-aided EG-CC scheme, which is characterized by the large discrepancies between the E_{b}/N_0 tunnel and capacity bounds shown in Table 6.1.

Note that the gains offered by the proposed learning-aided UEC scheme are achieved for free, with no increase to decoding complexity or to transmission-energy, -bandwidth or -duration. Note that the idealized versions of the Arithmetic-CC benchmarker offer similar SER performances to our optimized practical learning-aided UEC scheme. This may be explained by the high efficiency of the arithmetic code, which results in a smaller capacity loss, compared to that of the EG-CC benchmarker. Note that the results of Figure 6.7 are provided for the case, where the Arithmetic-CC scheme employs a memory size of $M = T$, which was found to offer a performance closely matching that of the optimized M. However, the practical version of the learning-aided Arithmetic-CC benchmarker performs poorly, since the memories in its transmitter and receiver become easily desynchronized in the presence of transmission errors.

6.6 Summary and Conclusions

In this chapter, we developed a novel learning-aided UEC scheme, which was designed for transmitting symbol values selected from unknown and non-stationary probability distributions. More specifically, the learning-aided UEC scheme is capable of heuristically inferring the source symbol distribution, based on the estimated symbol probabilities obtained from the recovered symbols that are stored in memory at the receiver. By iteratively feeding these estimated symbol probabilities back to the UEC decoder of Figure 6.4, it can dynamically adjust and maintain near-capacity operation, at the cost of only a moderate memory requirement at the receiver. The simulation results of Figure 6.7 show that our learning-aided UEC scheme outperforms the benchmarkers by up to 0.85 dB in a variety of scenarios, without imposing any additional decoding complexity or any additional transmission-energy, -bandwidth, or -duration.

In Section 6.2, we commenced by reviewing the stationary zeta distribution that was used in the previous chapters and which can be used for crudely modelling the symbols produced by the H.265 video codec. However, in the H.265 codec, the particular distribution of the source symbols produced during entropy encoding is sensitive to the specific selection of video encoding parameters and to the type of video sequences being encoded.

Owing to this, we prefer to consider model source distributions, which can be readily parametrized to be representative of the distribution of the H.265 codec's entropy-encoded symbol values in various applications, as well as of a wide variety of other multimedia source distributions. This idea was extended by conceiving the non-stationary zeta source distribution model that is applied throughout this chapter. This non-stationary distribution is inspired by the non-stationary nature of the H.265 distribution.

The transmitter and the receiver operations of the proposed learning-aided UEC scheme of Figure 6.2 were introduced in Sections 6.3.1 and 6.3.2, respectively. Like conventional UEC coding, the proposed learning-aided UEC scheme does not require any knowledge of the symbol occurrence probabilities at the transmitter. However, in contrast to conventional UEC coding, the proposed scheme does not require this knowledge at the receiver either, since it can gradually estimate the source probability distribution from the recovered symbols. The operation of our proposed learning mechanism was discussed in Section 6.3.3. When the probability distribution $P(x)$ of the source symbols $\mathbf{x}$ is non-stationary and unknown, our learning-aided UEC scheme is capable of iteratively estimating the average unary codeword length l and the probabilities of the first $r/2-1$ symbol values $\mathbf{p} = [P(x)]_{x=1}^{r/2-1}$ from the recovered symbol vectors $\hat{\mathbf{x}}$, which are then fed back to the trellis decoder as *a priori* information, in order to improve the receiver's error correction capability. The estimation is implemented using the memory storage block labeled M in Figure 6.4. This memory storage is used to store source distribution statistics that have been observed from the successively recovered symbol vectors $\hat{\mathbf{x}}$. Moreover, we quantified the size of this memory storage in terms of the number M of most-recently recovered symbol vectors $\hat{\mathbf{x}}$, allowing us to strike a desirable trade-off between the memory size and the error correction capability.

In Section 6.4, we also proposed two SSCC benchmarkers based on the same learning technique, namely a learning-aided EG-CC scheme and a learning-aided Arithmetic-CC scheme, as well as their idealized and impractical versions. For the sake of fair comparison, we applied doping or puncturing in order to obtain the same overall effective throughput η for all schemes, as well as selecting the same computational complexity limit in order to achieve iterative decoding convergence in all schemes.

Our simulation results of Section 6.5 showed that the proposed learning-aided UEC scheme benefits from an E_b/N_0 gain of up to 1 dB, when employing $M = 1$ instead of $M = 0$. The SER performance of the practical learning-aided UEC scheme having the optimized memory size M offers a performance that is within 0.1 dB from the idealized but impractical UEC scheme, at the SER of 10^{-3}. By contrast, the practical learning-aided EG-CC scheme only requires $M = 1$ in order to achieve the same performance as

its idealized but impractical baseline. This is because the learning-aided EG-CC scheme only has to estimate a single source distribution parameter, namely the value of the LLR that is used for all elements of the vector $\tilde{\mathbf{y}}^{\mathrm{a}}$ provided for the CC decoder. Nevertheless, our $M = 1$-based learning-aided UEC scheme offers up to 0.8 dB gain compared to the $M = 1$ learning-aided EG-CC scheme at the SER of 10^{-3}, and gains of up to 0.85 dB when employing the value of $M = T$. Note that the results of Figure 6.7 are provided for the case, where the Arithmetic-CC scheme employs a memory size of optimized M. The practical version of the learning-aided Arithmetic-CC benchmarker was found to perform poorly, since the memories in its transmitter and receiver become easily desynchronized in the presence of transmission errors.

Chapter 7

Conclusions and Future Research

In this concluding chapter, a summary of the thesis and the main findings of our investigations will be presented in Section 7.1, followed by a range of tangible design guidelines in Section 7.2. Finally, we present a number of potential ideas for future work in Section 7.3.

7.1 Main Conclusions

As discussed in Chapter 1, in classic Separate Source and Channel Coding (SSCC), the source information may be reconstructed with an infinitesimally low probability of error, provided that the transmission rate does not exceed the channel's capacity. However, SSCC schemes require both the transmitter and receiver to accurately estimate the occurrence probability of every symbol value that the source produces. In practice, the occurrence probability of rare symbol values can only be accurately estimated, if a sufficiently large number of symbols has been observed, hence potentially imposing an excessive latency. This motivates the design of universal codes, which facilitate the binary encoding of symbols selected from infinite sets, without requiring any knowledge of the corresponding occurrence probabilities at either the transmitter or receiver. In order to exploit the residual redundancy and hence to achieve near-capacity operation, the classic SSCC schemes may be replaced by Joint Source and Channel Coding (JSCC) arrangements in many applications. However, the decoding complexity of all previous JSCCs increases rapidly with the cardinality of the symbol set, to the extent that it becomes excessive for the H.264/H.265 symbol probability distribution and asymptotically tends to infinity, when the cardinality is infinite.

Against this background, the novel concept of Unary Error Correction (UEC) coding is studied and its applications, characteristics and performance are investigated in this thesis. Before we introduced our proposed UEC family, Chapter 2 presented the background knowledge that is required when designing codes in the UEC family. Chapter 2 focused on

the state-of-the-art that is widely used in wireless communications, providing a foundation that is frequently referred to by the following chapters. In each of the following chapters, we focused our attention and made contributions to one or more aspects of Figure 1.3, which is repeated in Figure 7.1 for convenience. The important conclusions of this treatise are as follows.

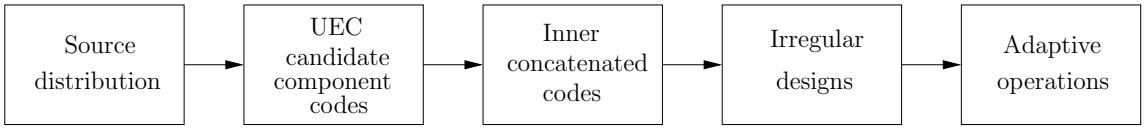

Figure 7.1: The design-flow of a UEC coded scheme.

In Chapter 3, we reviewed the encoding and decoding operations of the UEC code and characterized a serial concatenation scheme, namely the UEC-Irregular Unity-Rate Convolutional (IrURC) scheme proposed for facilitating practical near-capacity operation. We also quantified the computational complexity of the UEC scheme in order to strike a desirable trade-off between the contradictory requirements of low complexity and near-capacity operation. The main conclusions of this chapter were as follows.

- The UEC encoder consists of two parts, the unary encoder and the trellis encoder. Owing to the synchronization between the unary codewords and the trellis transitions, the UEC decoder can employ the Bahl, Cocke, Jelinek and Raviv (BCJR) algorithm to exploit all residual redundancy that remains following unary encoding, hence facilitating near-capacity operation at a moderate complexity.
- The EXIT chart and area properties of the UEC code were characterized and it was shown that in the case of arbitrary symbol value distributions, the capacity loss asymptotically approaches zero, as the complexity of the UEC trellis is increased.
- We showed that the SSCC Elias Gamma (EG)-Convolutional Code (CC) benchmarker suffers from capacity loss. Hence, our simulation results demonstrate that our UEC-IrURC scheme outperforms the EG-CC-IrURC benchmarker, offering as much as 1.3 dB gain and operating within 1.6 dB of the capacity bound.

In Chapter 4, we proposed an *Adaptive* UEC-Turbo scheme, which is a three-stage concatenation that applies an adaptive iterative decoding technique for expediting iterative decoding convergence. A Three-Dimensional (3D) EXIT chart analysis was proposed for controlling the dynamic adaptation of the UEC trellis decoder, as well as for controlling the decoder activation order between the UEC decoder and the turbo decoder. The main conclusions of this chapter were as follows.

- When a UEC trellis encoder operates on the basis of an extended codebook, the corresponding trellis decoder can still employ the original codebook with lower complexity. Based on this observation, we proposed dynamically reducing or increasing the number of states employed in the trellis decoder, in order to balance the performance versus complexity trade-off..
- A 3D EXIT chart was employed for quantifying the benefit of activating each decoding component at each stage of the iterative decoding process. We showed that the UEC decoder's operation can be dynamically adjusted, and additionally its activation order controlling the iterative soft information exchange with the two turbo decoder components may also be varied.
- We quantified the corresponding complexity cost in terms of the number of Add, Compare and Select (ACS) operations performed by each decoding component. By activating the specific decoding component offering the largest benefit-to-cost ratio at each stage, we demonstrated that the convergence of the iterative decoding process may be significantly expedited, resulting in an attractive trade-off between its decoding complexity and its error correction capability.

In Chapter 5, we proposed an *Irregular* UEC-IrURC scheme, which facilitates nearer-capacity operation. The IrUEC scheme employs different UEC parametrizations for the encoding of different subsets of each message frame, operating on the basis of a single irregular trellis having a novel design. The main conclusions of this chapter are as follows.

- The irregular trellis employed by an IrUEC has a non-uniform structure that applies different UEC parametrizations for different subsets of the frame on a bit-by-bit basis. This allows the irregularity of the proposed IrUEC code to be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis. Hence, nearer-to-capacity operation is facilitated by exploiting this fine-grained control of the IrUEC irregularity.
- The free-distance properties of the UEC trellis were characterised for the first time in this chapter, in order to conceive an attractive parametrization of the IrUEC scheme. Having characterized the free-distance of the UEC trellis using different codebooks, we carefully selected a suite of UEC codes having a wide variety of EXIT chart shapes for the component codes of our IrUEC code.
- A new double-sided EXIT chart matching algorithm was proposed for jointly matching the EXIT charts of the IrUEC and the IrURC codes. On the one hand, the component UEC codes having a wide variety of EXIT chart shapes provide design freedom for the IrUEC EXIT chart. On the other hand, the novel double-sided EXIT chart

matching algorithm exploits this design freedom, in order to parametrize the IrUEC-IrURC scheme for creating a narrow but marginally open EXIT chart tunnel at a low E_{b}/N_0 value that is close to the area bound and the capacity bound.

In Chapter 6, we developed a novel *Learning-aided* UEC scheme, which was designed for transmitting symbol values selected from unknown and non-stationary probability distributions. The learning-aided UEC scheme is capable of heuristically inferring the source symbol distribution, hence eliminating the requirement for any prior knowledge of the symbol occurrence probabilities at either the transmitter or the receiver. The main conclusions of this chapter were as follows.

- Inspired by the H.265 video codec, we extended the stationary zeta source symbol distribution of the previous chapters to conceive a non-stationary zeta source distribution, which is capable of modelling the non-stationary nature of the H.265 distribution.
- The proposed learning mechanism can heuristically and iteratively estimate the average unary codeword length l and the probabilities of the first $r/2-1$ symbol values $\mathbf{p} = [P(x)]_{x=1}^{r/2-1}$ from the recovered symbol vectors $\hat{\mathbf{x}}$, which are then fed back to the trellis decoder as *a priori* information, in order to improve the receiver's error correction capability.
- The proposed learning algorithm was implemented using a memory storage at the receiver. This memory storage is used for storing the source distribution statistics that have been observed from successively recovered symbol vectors. We quantified the size of this memory storage in terms of the M most-recently recovered symbol vectors, showing that a trade-off between the memory size and the error correction capability can be achieved, while maintaining near-capacity operation.

7.2 Design Guidelines

In this section, we will summarize the general design guidelines of the UEC scheme by examining the various schemes investigated throughout Chapter 3 to Chapter 6, which are summarized in Table 7.1. Figure 7.1 highlights the specific aspects that must be considered, when designing a UEC coding scheme.

- *Source distribution*
  The UEC code is designed for conveying a vector $\mathbf{x} = [x_i]_{i=1}^{a}$ comprising a number of symbols. Each symbol $x_i \in \mathbb{N}_1$ of the vector is obtained by an Independent and Identically Distributed (IID) Random Variable (RV) X_i, which adopts the value x

Chapter	Scheme	Figure	Source distribution	Outer codec			Inner codec			Modem
				Number of component codes	EXIT matched	Adaptive operation	Number of component codes	EXIT matched	Adaptive operation	
3	UEC-IrURC	3.3	Zeta distribution	1	No	No	10	Yes	No	Gray-coded QPSK
4	UEC-Turbo	4.2	Zeta distribution	4	No	Yes	1	No	Yes	
5	IrUEC-IrURC	5.2	Zeta distribution	5	Yes	No	10	Yes	No	
6	UEC-URC	6.4	Non-stationary zeta distribution	1	No	Yes	1	No	No	

Table 7.1: Comparison of the various schemes proposed in Chapters 3 to 6.

with the probability $Pr(X_i = x) = P(x)$, where $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$ is the infinite-cardinality set comprising all positive integers. Figure 3.2 depicts the geometric distribution, the zeta distribution and the distribution of the symbol values that are EG encoded by the H.264 video encoder. These symbol values appear to obey Zipf's law, since their distribution may be approximated by the zeta distribution. As shown in Table 7.1, we therefore focused our attention on the stationary zeta probability distribution in Chapters 3 to 5, considering its parametrization using the parameter p_1. In Chapter 6, we extended the stationary zeta distribution to create a non-stationary zeta distribution model, which is capable of modelling the non-stationary nature of the symbols generated by the H.265 video codec.

The design of a UEC scheme must consider, whether the source distribution is known as in Chapters 3 to 5, or unknown as in Chapter 6. Likewise, the design must consider whether the distribution is stationary as in Chapters 3 to 5, or non-stationary as in Chapter 6. If the distribution is unknown or non-stationary, then the learning-aided UEC scheme of Chapter 6 is motivated. Otherwise, the design of the UEC scheme must be parametrized according to the particular source distribution, as was the case for the adaptive and irregular UEC schemes of Chapters 4 and 5, respectively.

- *UEC candidate component codes*

  In Chapter 5, the free-distance properties of the UEC codebooks are characterised for the first time, using a heuristic method is capable of obtaining an approximate measurement of the free-distance. Having characterized the free-distance of the UEC trellis using different codebooks, a suite of UEC codes having a wide variety of EXIT chart shapes are selected for the candidate component codes of the IrUEC code, as shown in Table 7.1. Since this suite of codes facilitates the creation of a narrow but marginally open EXIT chart tunnel, we recommend its employment for near-capacity irregular UEC schemes.

  By contrast, only one UEC candidate code is selected for each of the regular UEC codes that are employed in Chapters 3, 4 and 6. For regular UEC schemes, the selection of the UEC codebook is based on a trade-off between (a) creating a narrow but marginally open EXIT chart tunnel for the sake of facilitating near-capacity operation and (b) having a low complexity.

- *Inner concatenated codes*

  Near-capacity operation is facilitated when our outer UEC code is serially concatenated and iteratively decoded with an inner code. In Chapter 3, a serial concatenation of the UEC-IrURC scheme is proposed, in which the IrURC code is employed as the inner code, as shown in Table 7.1. In Chapter 4, a three-stage concatenation of the

UEC-Turbo scheme is conceived, in which a turbo code is employed as the inner code, as shown in Table 7.1. More particularly, the turbo code is a half-rate code comprising two Unity-Rate Convolutional (URC) codes. Therefore, the UEC-Turbo scheme comprises a total of three decoders, requiring 3D EXIT chart analysis and an adaptive iterative decoding algorithm. An outer IrUEC code is concatenated with an inner IrURC code in Chapter 5, as shown in Table 7.1. Owing to our novel double-sided EXIT chart matching algorithm, 'nearer-capacity' operation is facilitated by the IrUEC-IrURC scheme. Table 7.1 also shows that a regular URC is employed as the inner code in Chapter 6, where we conceived a learning algorithm for allowing the UEC code to recover source symbols having an unknown or non-stationary distribution.

Where near-capacity operation is desired, we recommend irregular inner codes, although these must be parametrized according to the specific source distribution encountered. For adaptive or simple schemes, we recommend regular inner codes.

- *Irregular design*

  Irregular coding has been proposed for the reliable transmission of information at channel SNRs that are close to the channel's capacity bound without imposing an excessive decoding complexity and latency. The difference between the E_b/N_0 area bound and the E_b/N_0 tunnel bound quantifies the capacity loss that is mitigated by irregular coding. In Chapters 3 and 5, irregular design is applied to the inner URC code, resulting in an IrURC code. Here, the $T = 10$ URC codes $\{\mathbf{URC}^t\}_{t=1}^{T=10}$ of Figure 2.6 are employed as the component codes. As a further step, the novel IrUEC code of Chapter 5 operates on the basis of a single irregular trellis having an irregular design that is afforded by the common features of all UEC trellises. This irregular trellis has a non-uniform structure that applies different UEC parametrizations for different subsets of the frame on a bit-by-bit basis, which allows the irregularity of the proposed IrUEC code to be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis.

  Where operation extremely close to capacity is desired, we recommend the use of irregular outer and inner codes, although these must be parametrized according to the particular source distribution.

- *Adaptive operations*

  Owing to the three-stage UEC-Turbo concatenation of Chapter 4, we can employ a 3D EXIT chart for quantifying the benefit of activating each decoding component, in order to dynamically adjust the activation order of the UEC decoder and the two turbo decoder components. Furthermore, based on the observation that extending the UEC codebook employed by the UEC decoder maintains compatibility with

the UEC encoder, we can also dynamically reduce or increase the number of states in the trellis decoder of the UEC-Turbo scheme. Therefore, a desirable trade-off between complexity and error correction performance is obtained by the Adaptive UEC-Turbo scheme of Figure 4.2. Meanwhile, the learning-aided UEC scheme of Chapter 6 employs adaptive operations to avoid requiring any prior knowledge of the symbol occurrence probabilities at either the transmitter or at the receiver. This is because it can heuristically estimate the source probability distribution from the recovered symbols, as well as iteratively feeding the distribution-related information back to the trellis decoder as *a priori* information, in order to adjust the receiver's parametrization and to improve the receiver's error correction capability. Hence, adaptive operation constitutes a powerful tool, which may be exploited for enhancing the performance of the UEC schemes.

7.3 Future Work

The research illustrated in this treatise can be extended in several directions. In this section, we highlight a number of potential future research ideas.

7.3.1 Adaptive/Irregular/Learning-aided EGEC, RiceEG and ExpGEC schemes

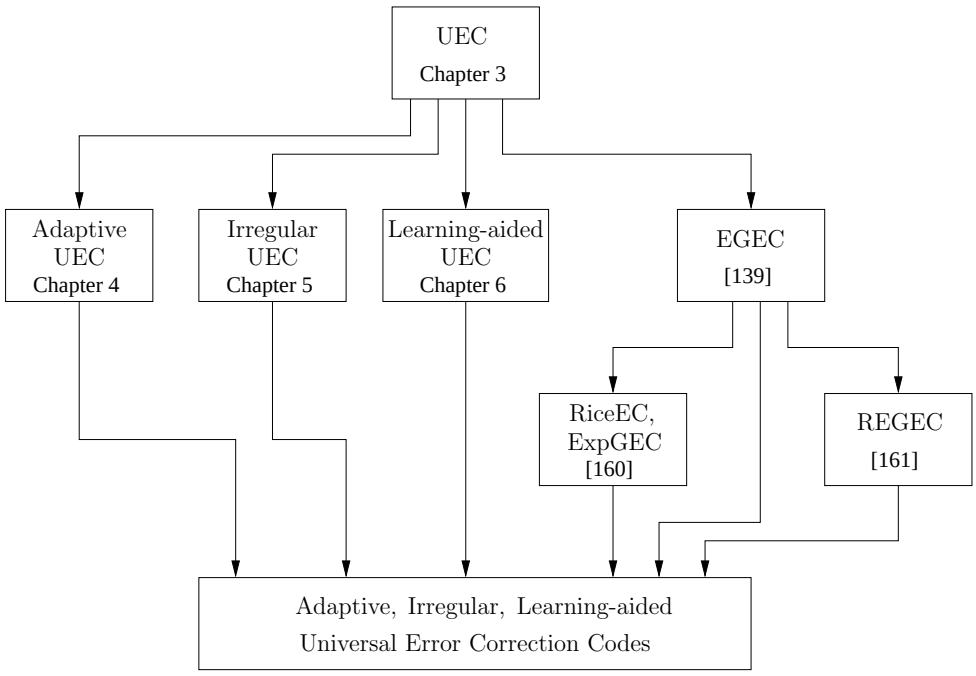

Figure 7.2: The enhancement of universal error correction codes by using the techniques developed in this treatise.

The proposed UEC code is based on the unary code, which is a special case of the Rice code [15]. More specifically, the Rice code is parametrized by M_{Rice}, where $M_{\text{Rice}} = 1$

yields the unary code, as shown in Table 7.2. By exploiting the similarities between the unary and Rice codes, our UEC code was extended to conceive a Rice Error Correction (RiceEC) code in [166][1]. Although RiceEC codes have shorter average codeword lengths than UEC codes for some source symbol distributions, neither of them may be deemed to constitute universal codes having finite average codeword lengths for all monotonic source symbol distributions. Owing to this, the UEC and RiceEC codes may suffer from having an average codeword length that tends to infinity for some source symbol distributions, including zeta distributions having $p_1 \leq 0.608$ in the case of the UEC code, as described in Section 3.3.

By contrast, the Elias Gamma (EG) code [12] and the Exponential Golomb (ExpG) code [15] are universal codes, providing a finite average codeword length for any monotonic source symbol distribution. Motivated by this, universal JSCCs were obtained by extending the UEC code to conceive the Elias Gamma Error Correction (EGEC) code of [145][2] and the Exponential Golomb Error Correction (ExpGEC) code of [166]. Both the codes are capable of facilitating the near-capacity transmission of infinite-cardinality symbol alphabets having any arbitrary monotonic probability distribution, hence having a much wider applicability. Note that the ExpG code is parametrized by k and the EG code is obtained in the special case of $k = 0$, as shown in Table 7.2. More specifically, Table 7.2 characterizes the Unary, Rice, EG and ExpG codes, which map each symbol d_i in the vector $\mathbf{d} = [d_i]_{i=1}^{a}$ to the respective codeword Unary(d_i), Rice(d_i), EG(d_i) or ExpG(d_i), where the Rice and ExpG encoders are parametrized by $M_{\text{Rice}} \in \{1, 2, 4, 8\}$ and $k \in \{0, 1, 2\}$, respectively.

Note that all codewords in Table 7.2 can be considered to be a concatenation of a unary codeword and a suffix codeword. All codewords having the same unary prefix start with a suffix having the same fixed length. Motivated by this, the EGEC, RiceEG and ExpGEC encoders decompose each input symbol into two sub-symbols, which are encoded separately by two distinct sub-encoders, as shown in the schematic of Figure 7.3. The first sub-encoder is referred to as the UEC sub-encoder, which operates in the same manner as the UEC encoder of this treatise. The second sub-encoder employs a serial concatenation of a Fixed length Code (FLC) and of a Convolutional Code (CC), which is referred to as the FLC-CC sub-encoder. More particularly, the splitter S decomposes each symbol d_i in the vector $\mathbf{d} = [d_i]_{i=1}^{a}$ into the sub-symbols x_i in the vector $\mathbf{x} = [x_i]_{i=1}^{a}$ and the sub-symbol t_i in the vector $\mathbf{t} = [t_i]_{i=1}^{a}$, according to the arrangement shown in Table 7.2 for the various

[1] In which the author of this thesis was co-authored and contributed to obtain the source distribution statistics.

[2] In which the author of this thesis was co-authored and contributed to the simulations.

	Rice(d_i) M_{Rice}=1, Unary(d_i)			Rice(d_i) M_{Rice}=2			Rice(d_i) M_{Rice}=4			Rice(d_i) M_{Rice}=8			ExpG(d_i) k=0, EG(d_i)			ExpG(d_i) k=1			ExpG(d_i) k=2			REGEC
d_i	x_i	$\mathbf{y}_i$	$\mathbf{u}_i$ t_i	x_i	$\mathbf{y}_i$	$\mathbf{u}_i$ t_i	x_i	$\mathbf{y}_i$	$\mathbf{u}_i$ t_i	x_i	$\mathbf{y}_i$	$\mathbf{u}_i$ t_i	x_i	$\mathbf{y}_i$	$\mathbf{u}_i$ t_i	x_i	$\mathbf{y}_i$	$\mathbf{u}_i$ t_i	x_i	$\mathbf{y}_i$	$\mathbf{u}_i$ t_i	
1	1	1	0	1	1	0	0	1	1	00	0	1	1	000	0	1	1		0	1	1	0	0	1	1	00	0	1
2	2	01	0	1	1	1	1	1	1	01	1	1	1	001	1	2	01	0	0	1	1	1	1	1	1	01	1	001
3	3	001	0	2	01	0	0	1	1	10	2	1	1	010	2	2	01	1	1	2	01	00	0	1	1	10	2	011
4	4	0001	0	2	01	1	1	1	1	11	3	1	1	011	3	3	001	00	0	2	01	01	1	1	1	11	3	00001
5	5	00001	0	3	001	0	0	2	01	00	0	1	1	100	4	3	001	01	1	2	01	10	2	2	01	000	0	00011
6	6	000001	0	3	001	1	1	2	01	01	1	1	1	101	5	3	001	10	2	2	01	11	3	2	01	001	1	01001
7	⋮	⋮	⋮	4	0001	0	0	2	01	10	2	1	1	110	6	3	001	11	3	3	001	000	0	2	01	010	2	01011
8				4	0001	1	1	2	01	11	3	1	1	111	7	4	0001	000	0	3	001	001	1	2	01	011	3	0000001
9				5	00001	0	0	3	001	00	0	2	01	000	0	4	0001	001	1	3	001	010	2	2	01	100	4	0000011
10				5	00001	1	1	3	001	01	1	2	01	001	1	4	0001	010	2	3	001	011	3	2	01	101	5	0001001
11				6	000001	0	0	3	001	10	2	2	01	010	2	4	0001	011	3	3	001	100	4	2	01	110	6	0001011
12				6	000001	1	1	3	001	11	3	2	01	011	3	4	0001	100	4	3	001	101	5	2	01	111	7	0100001
13				⋮	⋮	⋮	⋮	4	0001	00	0	2	01	100	4	4	0001	101	5	3	001	110	6	3	001	0000	0	
14								4	0001	01	1	2	01	101	5	4	0001	110	6	3	001	111	7	3	001	0001	1	⋮
15								4	0001	10	2	2	01	110	6	4	0001	111	7	4	0001	0000	0	3	001	0000	2	

Table 7.2: The decomposition of symbols d_i into sub-symbols x_i and t_i for the EGEC, RiceEC and ExpGEC codes [145, 166] of Figure 7.3. Each codeword can be considered to be a concatenation of $\mathbf{y}_i$ and $\mathbf{u}_i$, where the bit-vector is decomposed into the sub-bit-vectors. The bit-vectors $\mathbf{y}_i$ and $\mathbf{u}_i$ relate to the sub-symbols x_i and t_i according to $\mathbf{y}_i$ = Unary(x_i) and $\mathbf{u}_i$ = FLC(t_i), respectively. The first 12 codewords are also listed for the REGEC code [167].

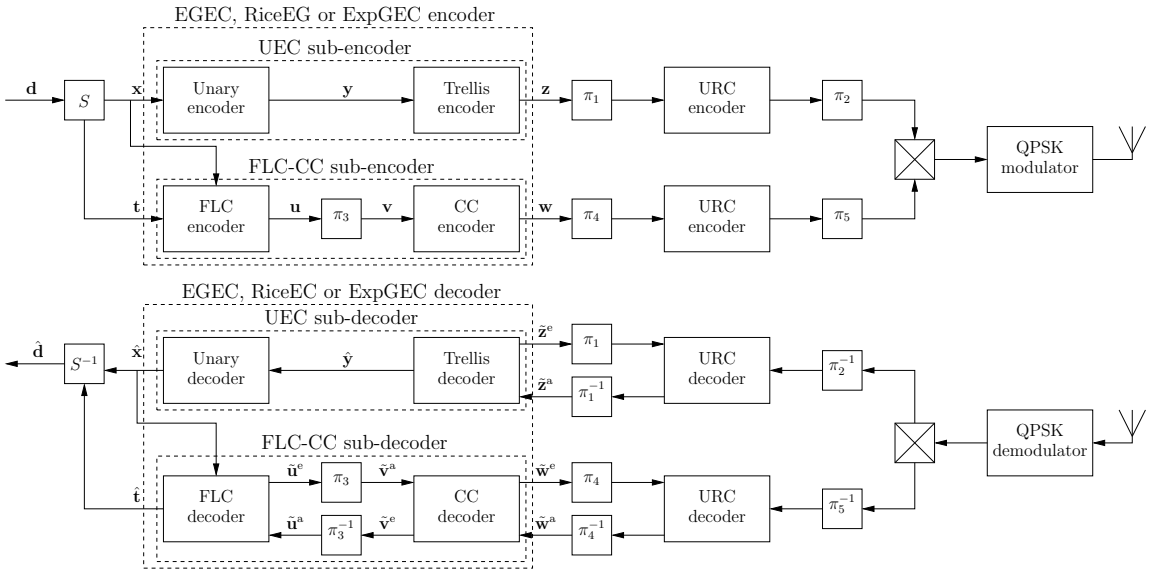

Figure 7.3: The schematic of the EGEC, RiceEG and ExpGEC codes [145, 166].

codes considered. Each sub-symbol x_i and each sub-symbol t_i is encoded by the UEC sub-encoder and the FLC-CC sub-encoder, respectively. An Unequal Error Protection (UEP) scheme was also proposed for optimizing the relative contribution of the two sub-codes to the encoding process, facilitating near-capacity operation at a low decoder complexity.

Owing to the similarity between the UEC scheme and the UEC sub-part of the EGEC, RiceEG and ExpGEC schemes, they may be readily enhanced by extending our adaptive, irregular and learning-aided techniques of Chapters 4, 5 and 6.

- For the adaptive design of Chapter 4, the upper URC code of Figure 7.3 may be replaced by a turbo code, providing a three-stage concatenation in the UEC sub-part. In this way, the EGEC, RiceEG or ExpGEC decoder can dynamically adjust the activation order of the three decoders in the UEC sub-part, as well as the number of states in the UEC trellis decoder in order to strike an attractive trade-off between the decoding complexity and the error correction capability. Likewise, the FLC-CC part of Figure 7.3 may be concatenated with a turbo code and the corresponding decoder activation order can be dynamically adapted.
- For the irregular design of Chapter 5, the UEC sub-part can provide a fine-grained bit-by-bit basis of the irregularity, rather than on a symbol-by-symbol basis. This allows an IrUEC sub-part to be designed in order to match the concatenated URC code. In the FLC-CC part, a conventional Irregular Convolutional Code (IrCC) may be used to obtain a similar benefit.
- For the learning-aided design of Chapter 6, a memory storage may be employed

in the scheme of Figure 7.3, in analogy to that of Figure 6.4. Therefore, the occurrence probabilities of the source symbols of the vector $\mathbf{x}$ can be estimated and hence can be fed back to the UEC trellis decoder as *a priori* information, facilitating reliable communication for unknown and non-stationary source probability distributions. Likewise, a similar mechanism may be employed in the FLC-CC part, in analogy to the learning-aided CC code used in the benchmarker of Chapter 6.

However, the EGEC, RiceEG and ExpGEC schemes have a complicated structure that comprises two parts, as shown in Figure 7.3. Owing to this, the FLC-CC sub-encoder cannot be operated until the operation of the UEC sub-encoder is completed, since the FLC-CC sub-decoder relies on the side information provided by the UEC sub-decoder. Therefore, the EGEC, RiceEG and ExpGEC schemes may suffer from the delay and loss of synchronization that are associated with the two parts. Furthermore, the UEP of the two parts must be tailored to the specific parametrization of the source distribution, leading to degraded performance, when the source distribution is unknown or non-stationary. Additionally, the puncturing used in this UEP scheme increases the complexity and can lead to capacity loss. Motivated by this, the so-called Reordered Elias Gamma Error Correction (REGEC) code was proposed in [167], as a universal JSCC having a simpler structure, as discussed in the following subsection.

7.3.2 Adaptive/Irregular/Learning-aided REGEC schemes

As shown in Figure 7.4, the REGEC code [167] combines a so-called Reordered Elias Gamma (REG) source code with a novel trellis-based channel code. As shown in Table 7.2, the REG codewords comprise a reordering of the bits in the EG codewords, allowing 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, facilitating near-capacity operation. In this way, the REGEC scheme avoids the complicated two-part structure that is used in the EGEC, RiceEG and ExpGEC schemes, as well as the associated drawbacks. In particular, the REGEC code also has a simple structure comprising only a single part, which does not suffer from the delay and loss of synchronization that are associated with the two parts of the EGEC, RiceEG and ExpGEC codes. Furthermore, since the REGEC code does not need UEP, its parametrization does not have to be tailored to the particular source distribution, giving a "one size fits all" scheme.

Owing to the similarity between the UEC scheme and the REGEC scheme, our adaptive, irregular and learning techniques may be readily extended to design the corresponding Adaptive-, Irregular- and Learning-aided REGEC schemes.

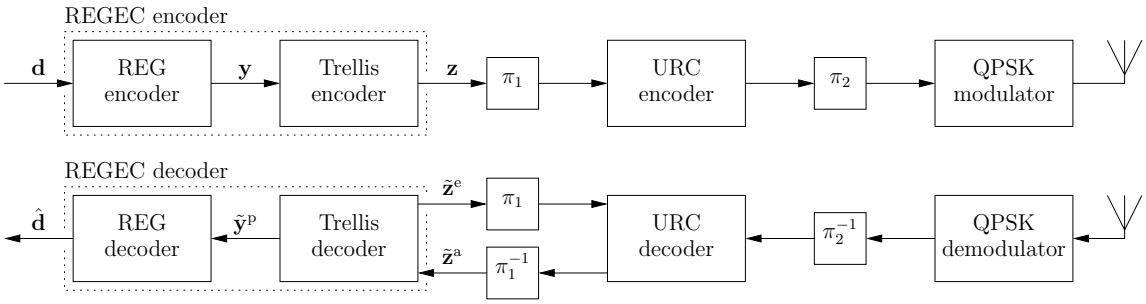

Figure 7.4: The schematic of the Reordered Elias Gamma Error Correction (REGEC) code [167].

- In case of the adaptive design of Chapter 4, the codebook extension technique is also applicable to the REGEC code, as discussed in [167]. This 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 URC code of Figure 7.4 may be replaced by a turbo code, providing a three-stage concatenation. Therefore, 3D EXIT chart analysis can be employed to quantify the potential benefits associated with activating each decoder, allowing hence the activation order of the three decoders to be dynamically adjusted.
- In case of the irregular design of Chapter 5, the REGEC trellis also exhibits common features regardless of its parametrization, in the same way as the UEC trellis. This may be exploited to create an irregular REGEC trellis having an irregularity that can be controlled on a fine-grained bit-by-bit basis, rather than on a symbol-by-symbol basis. Moreover, the URC code of Figure 7.4 may be replaced by an IrURC code, in order to create a narrow, but marginally open EXIT chart tunnel, hence facilitating 'nearer-to-capacity' operation.
- In case of the learning-aided design of Chapter 6, the memory storage that is employed in the scheme of Figure 6.4 may be also introduced into the REGEC scheme of Figure 7.4. In this way, the occurrence probabilities of the source symbols can be estimated and then fed back to the trellis decoder as *a priori* information. In this way, the REGEC code would become capable of reliable near-capacity communication for unknown and non-stationary source probability distributions.

7.4 Closing Remarks

Throughout this thesis we have introduced the novel UEC scheme and methodologies for its design in the context of digital telecommunications. Our UEC scheme is a JSCC arrangement conceived for performing both the compression and error correction of multimedia information during its transmission from an encoder to a decoder. The UEC code is capable of mitigating capacity loss, hence facilitating near-capacity operation, even when the source symbol values are selected from a set having an infinite cardinality, such as the set of all positive integers. In Chapter 4, we proposed an *Adaptive* UEC-Turbo scheme, which is a three-stage concatenation that applies an adaptive iterative decoding technique for expediting iterative decoding convergence. A 3D EXIT chart analysis was proposed for controlling the dynamic adaptation of the UEC trellis decoder, as well as for controlling the activation order of exchanging soft extrinsic information between the UEC decoder and the turbo decoder. In Chapter 5, we developed an *Irregular* UEC-IrURC scheme, which facilitates nearer-capacity operation. The IrUEC scheme employs different UEC parametrizations for the encoding of different subsets of each message frame, operating on the basis of a single irregular trellis having a novel design. In Chapter 6, we proposed a *Learning-aided* UEC scheme, which was designed for transmitting symbol values selected from unknown and non-stationary probability distributions. The learning UEC scheme is capable of heuristically inferring the source symbol distribution, hence eliminating the requirement for any prior knowledge of the symbol occurrence probabilities at either the transmitter or the receiver. Finally, in Chapter 7, we described the application of the proposed techniques to universal error correction codes, as potential opportunities for future work. With these benefits, this thesis may contribute to future standards for the reliable near-capacity transmission of multimedia information, having significant technical and economic impact.

Bibliography

[1] J. L. Massey, "Joint source and channel coding," in *Proceedings of Communication Systems and Random Process Theory*, (The Netherlands: sijthofoo and Noordhoff), pp. 279–293, December 1978.

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

[3] A. J. Viterbi and J. Omura, *Principles of Digital Communication and Coding*. New-York: McGraw-Hill, 1979.

[4] L. Hanzo, P. J. Cherriman, and J. Streit, *Video Compression and Communications: H.261, H.263, H.264, MPEG4 and Proprietary Codecs for HSDPA-Style Adaptive Turbo-Transceivers*. John Wiley and IEEE Press, September 2007.

[5] L. Hanzo, P. J. Cherriman, and J. Streit, *Video Compression and Communications: H.261, H.263, H.264, MPEG4 and Proprietary Codecs for HSDPA-Style Adaptive Turbo-Transceivers*. John Wiley and IEEE Press, September 2007.

[6] L. Hanzo, F. C. A. Somerville, and J. P. Woodard, *Voice and Audio Compression for Wireless Communications*. John Wiley and IEEE Press, 2007.

[7] A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Transactions on Information Theory*, vol. 22, pp. 1–10, January 1976.

[8] R. M. Fano, "The transmission of information," in *Research Laboratory of Electronics*, pp. Mass. Inst. of Techn. (MIT), Technical Report No. 65, March 1949.

[9] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, pp. 147–160, April 1950.

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

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

[12] P. Elias, "Coding for two noisy channels," *IRE Convention Record, pt.4*, pp. 37–47, 1955.

[13] J. Wozencraft, "Sequential decoding for reliable communication," *IRE National Converntion Record*, vol. 5, pp. 11–25, August 1957.

[14] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, January 1962.

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

[16] 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.

[17] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268–278, March 1973.

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

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

[20] H. Imai and S. Hirakawa, "A new multilevel coding method using error-correcting codes," *IEEE Transactions on Information Theory*, vol. 23, pp. 371–377, May 1977.

[21] J. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Transactions on Information Theory*, vol. 24, pp. 76–80, January 1978.

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

[23] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, pp. 149–162, March 1979.

[24] G. Ungerböck, "Channel coding with multilevel/phase signals," *IEEE Transactions on Information Theory*, vol. 28, pp. 55–67, January 1982.

[25] I. H. Witten, J. G. Cleary, and R. Neal, "Arithmetic coding for data compression," *Communications of the ACM*, pp. 520–540, June 1987.

[26] D. Divsalar and M. K. Simon, "Multiple trellis coded modulation (MTCM)," *IEEE Transactions on Communications*, vol. 36, pp. 410–419, April 1988.

[27] A. R. Calderbank, "Multilevel codes and multistage decoding," *IEEE Transactions on Communications*, vol. 37, pp. 222–229, March 1989.

[28] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proceedings of IEEE Global Telecommunications Conference*, vol. 3, pp. 1680–1686, November 1989.

[29] W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data disturbed by time-varying intersymbol interference [applicable to digital mobile radio receivers]," in *Proceedings of IEEE Global Telecommunications Conference*, vol. 3, pp. 1679–1684, December 1990.

[30] W. T. Webb, L. Hanzo, and R. Steele, "Bandwidth efficient QAM schemes for Rayleigh fading channels," *IEE Proceedings I, Communications, Speech and Vision*, vol. 138, pp. 169–175, June 1991.

[31] E. Zehavi, "8-PSK trellis codes for a rayleigh channel," *IEEE Transactions on Communications*, vol. 40, pp. 873–884, May 1992.

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

[33] Y. Kofman, E. Zehavi, and S. Shamai, "Performance analysis of a multilevel coded modulation system," *IEEE Transactions on Communications*, vol. 42, pp. 299–312, February 1994.

[34] S. L. Goff, A. Glavieux, and C. Berrou, "Turbo-codes and high spectral efficiency modulation," in *IEEE International Conference on Communications*, vol. 2, pp. 645–649, May 1994.

[35] 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, pp. 1009–1013, June 1995.

[36] X. D. Li and J. A. Ritcey, "Bit-interleaved coded modulation with iterative decoding," *IEEE Communications Letters*, vol. 1, pp. 169–171, November 1997.

[37] P. Robertson and T. Wörz, "Bandwidth-efficient turbo trellis-coded modulation using punctured component codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 206–218, February 1998.

[38] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Self-concatenated trellis coded modulation with self-iterative decoding," in *IEEE Global Telecommunications Conference*, vol. 1, (Sydney, NSW, Australia), pp. 585–591, 1998.

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

[40] M. Luby, "LT codes," in *Proceedings of 43rd Annunal IEEE Symposium Foundations of Computer Science*, (Vancouver, BC, Canada), pp. 271–280, November 2002.

[41] J. Hou and M. H. Lee, "Multilevel LDPC codes design for semi-BICM," *IEEE Communications Letters*, vol. 8, pp. 674–676, November 2004.

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

[43] G. Yue, B. Lu, and X. Wang, "Design of rate-compatible irregular repeat accumulate codes," *IEEE Transactions on Communications*, vol. 55, pp. 1153–1163, June 2007.

[44] M. Grangetto, B. Scanavino, G. Olmo, and S. Benedetto, "Iterative decoding of serially concatenated arithmetic and channel codes with JPEG 2000 applications," *IEEE Transactions on Image Processing*, vol. 16, pp. 1557–1567, June 2007.

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

[46] Z. Wang and J. Luo, "Error performance of channel coding in random-access communication," *IEEE Transactions on Information Theory*, vol. 58, pp. 3961–3974, June 2012.

[47] J. Luo, "A generalized channel coding theory for distributed communication," *IEEE Transactions on Communications*, vol. 63, pp. 1043–1056, April 2015.

[48] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. 28, pp. 84–95, January 1980.

[49] H. Kumazawa, M. Kasahara, and T. Namekawa, "A construction of vector quantizers for noisy channels," *Electronics and Engineering in Japan*, vol. 67-B, pp. 39–47, January 1984.

[50] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, March 1960.

[51] J. Max, "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, vol. 6, pp. 7–12, March 1960.

[52] N. Farvardin, "A study of vector quantization for noisy channels," *IEEE Transactions on Information Theory*, vol. 36, pp. 799–809, July 1990.

[53] K. Sayood and J. C. Borkenhagen, "Use of residual redundancy in the design of joint source/channel coders," *IEEE Transactions on Communications*, vol. 39, pp. 838–846, June 1991.

[54] D. J. Miller and M. Park, "A sequence-based approximate mmse decoder for source coding over noisy channels using discrete hidden markov models," *IEEE Transactions on Communications*, vol. 46, pp. 222–231, February 1998.

[55] B. L. Montgomery and J. Abrahams, "Synchronization of binary source codes," *IEEE Transactions on Information Theory*, vol. 32, pp. 849–854, November 1986.

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

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

[58] R. Thobaben and J. Kliewer, "Design considerations for iteratively-decoded source-channel coding schemes," in *Proceedings of Allerton Conference on Communications, Control, and Computing*, (Monticello, IL, USA), September 2006.

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

[60] V. B. Balakirsky, "Joint source-channel coding using variable-length codes," *Proceedings of IEEE International Symposium on Information Theory (ISIT'97)*, p. 419, January 1997.

[61] M. Park and D. J. 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, January 2000.

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

[63] R. Bauer and J. Hagenauer, "Symbol by symbol MAP decoding of variable length codes," in *Proceedings of ITG Conference on Source and Channel Coding*, (Munich, Germany), pp. 111–116, January 2000.

[64] 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.

[65] C. Weidmann, "Reduced-complexity soft-in-soft-out decoding of variable-length codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Yokohama, Japan), p. 201, June 2003.

[66] S. Malinowski, H. Jegou, and C. Guillemot, "Synchronization recovery and state model reduction for soft decoding of variable length codes," *IEEE Transactions on Information Theory*, vol. 53, pp. 368–377, January 2007.

[67] R. G. Maunder, J. Kliewer, S. X. 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.

[68] V. A. Kotelnikov, *The Theory of Optimum Noise Immunity*. New York, NY, USA: McGraw-Hill, 1959.

[69] A. Kurtenbach and P. Wintz, "Quantizing for noisy channels," *IEEE Transactions on Communications*, vol. 17, pp. 291–302, April 1969.

[70] N. Farvardin and V. Vaishampayan, "Optimal quantizer design for noisy channels: An approach to combined source-channel coding," *IEEE Transactions on Information Theory*, vol. 33, pp. 827–838, November 1987.

[71] R. R. Wyrwas and P. G. Farrell, "Joint source-channel coding for raster document transmission over mobile radio," *IEE Proceedings I on Communications, Speech and Vision*, vol. 136, pp. 375–380, December 1989.

[72] K. Ramchandran, A. Ortega, K. M. Uz, and M. Vetterli, "Multiresolution broadcast for digital HDTV using joint source/channel coding," *IEEE Journal on Selected Areas in Communications*, vol. 11, pp. 6–23, January 1993.

[73] I. Kozintsev and K. Ramchandran, "Robust image transmission over energy-constrained time-varying channels using multiresolution joint source-channel coding," *IEEE Transactions on Signal Processing*, vol. 46, pp. 1012–1026, April 1998.

[74] R. E. V. Dyck and D. J. Miller, "Transport of wireless video using separate, concatenated, and joint source-channel coding," *in Proceedings of the IEEE*, vol. 87, pp. 1734–1750, October 1999.

[75] J. f. Cai and C.-W. Chen, "Robust joint source-channel coding for image transmission over wireless channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, pp. 962–966, September 2000.

[76] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proceedings of the Second International Conference on Turbo Codes and Related Topics*, (Brest, France), pp. 1–5, September 2000.

[77] N. Görtz, "A generalized framework for iterative source-channel decoding," *Annals of Telecommunications, Special issue on "Turbo Codes: a wide-spreading technique"*, pp. 435–446, July 2001.

[78] T. A. Ramstad, "Shannon mappings for robust communication," *Telektronikk*, vol. 98, no. 1, pp. 114–128, 2002.

[79] J. Hagenauer and N. Görtz, "The turbo principle in joint source-channel coding," in *Proceedings of IEEE Information Theory Workshop*, (Paris, France), pp. 275–278, March 2003.

[80] 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.

[81] X. Jaspar, C. Guillemot, and L. Vandendorpe, "Joint source-channel turbo techniques for discrete-valued sources: From theory to practice," *Proceedings of the IEEE*, vol. 95, pp. 1345–1361, June 2007.

[82] Q. Xu, V. Stankovic, and Z.-X. Xiong, "Distributed joint source-channel coding of video using raptor codes," *IEEE Journal on Selected Areas in Communications*, vol. 25, pp. 851–861, May 2007.

[83] P. Minero, S. Lim, and Y.-H. Kim, "Joint source-channel coding via hybrid coding," *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pp. 781–785, July 2011.

[84] D. Persson, J. Kron, M. Skoglund, and E. G. Larsson, "Joint source-channel coding for the mimo broadcast channel," *IEEE Transactions on Signal Processing*, vol. 60, pp. 2085–2090, April 2012.

[85] V. Kostina and S. Verdu, "Lossy joint source-channel coding in the finite block-length regime," *IEEE Transactions on Information Theory*, vol. 59, pp. 2545–2575, May 2013.

[86] S. M. Romero, M. Hassanin, J. Garcia-Frias, and G. R. Arce, "Analog joint source channel coding for wireless optical communications and image transmission," *Journal of Lightwave Technology*, vol. 32, pp. 1654–1662, May 2014.

[87] S. Tridenski, R. Zamir, and A. Ingber, "The Ziv-Zakai-Rényi bound for joint source-channel coding," *IEEE Transactions on Information Theory*, vol. 61, pp. 4293–4315, August 2015.

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

[89] *Advanced video coding for generic audiovisual services*. ITU-T Std. H.264, March 2005.

[90] *High efficiency video coding*. ITU-T Rec. H.265, June 2013.

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

[92] 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. 6, pp. 1977–1987, May 2013.

[93] W. Zhang, Y. Jia, X. Meng, M. F. Brejza, R. G. 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, February 2015.

[94] 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 from an infinite set," *IEEE Transactions on Communications*, vol. 63, pp. 5073–5088, December 2015.

[95] 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 sources," *to be submitted*, December 2015.

[96] W. Zhang, R. G. Maunder, and L. Hanzo, "On the complexity of unary error correction codes for the near-capacity transmission of symbol values from an infinite set," *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2795–2800, April 2013.

[97] M. F. Brejza, W. Zhang, R. G. Maunder, B. M. 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," *IEEE International Conference on Communications (ICC)*, pp. 2603–2608, June 2015.

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

[99] S. ten Brink, "Convergence of iterative decoding," *IEEE Electronics Letters*, vol. 35, pp. 806–808, May 1999.

[100] R. G. Maunder and L. Hanzo, "Near-capacity irregular variable length coding and irregular unity rate coding," *IEEE Transactions on Wireless Communications*, vol. 8, pp. 5500–5507, November 2009.

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

[102] C. Douillard, M. Jezequel, C. Berrou, A. Picart, P. Didier, and A. Glavieux, "Iterative correction of intersymbol interference: Turbo equalization," *European Transactions on Telecommunications*, vol. 6, pp. 507–511, September 1995.

[103] R. Bauer and J. Hagenauer, "On variable length codes for iterative source/channel decoding," in *Proceedings of IEEE Data Compression Conference (DCC)*, (Snowbird, USA), pp. 273–282, April 2001.

[104] M. Adrat and P. Vary, "Iterative source-channel decoding: Improved system design using EXIT charts," *EURASIP Journal on Applied Signal Processing (Special Issue: Turbo Processing)*, pp. 928–941, May 2005.

[105] T. Richardson and R. Urbanke, "The capacity of LDPC codes under message passing decoding," *IEEE Transactions on Information Theory*, vol. 47, pp. 595–618, February 2001.

[106] D. Divsalar, S. Dolinar, and F. Pollara, "Serial concatenated trellis coded modulationn with rate-1 inner code," in *Proceedings of the IEEE Global Telecommunications Conference*, (San Francisco, CA, USA), pp. 777–782, November 2000.

[107] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 1st ed., 2005.

[108] L. Hanzo, T. H. Liew, B. L. Yeap, R. Y. S. Tee, and S. X. Ng, *Turbo Coding, Turbo Equalisation and Space-Time Coding: EXIT-Chart Aided Near-Capacity Designs for Wireless Channels*. John Wiley and Sons, 2rd ed., 2011.

[109] D. Divsalar and F. Pollara, "Turbo codes for deep space communications," *TDA Progress Report 42-120*, pp. 29–39, February 1995.

[110] S. Huettinger and J. Huber, "Design of multiple turbo codes with transfer characteristics of component codes," in *Proceedings of Conference on Information Sciences and Systems (CISS '02)*, (Princeton, NJ, USA), March 2002.

[111] D. Divsalar and F. Pollara, "Serial and hybrid concatenated codes with applications," in *International Symposium on Turbo Codes and Related Topics*, (Brest, France), September 1997.

[112] S. X. Ng, M. F. U. Butt, and L. Hanzo, "On the union bounds of self-concatenated convolutional codes," *IEEE Signal Processing Letters*, vol. 16, pp. 1070–9908, September 2009.

[113] D. Divsalar, S. Dolinar, and F. Pollara, "Serial concatenated trellis coded modulation with rate-1 inner code," in *Proceedings of IEEE Global Telecommunications Conference*, (San Francisco, CA, USA), pp. 777–782, November 2000.

[114] R. Y. S. Tee, R. G. Maunder, and L. Hanzo, "Exit-chart aided near-capacity irregular bit-interleaved coded modulation design," *IEEE Transactions on Wireless Communications*, vol. 8, pp. 32–37, January 2009.

[115] L. Hanzo, O. Alamri, M. E. Hajjar, and N. Wu, *Near-Capacity MutiFunctional MIMO Systems*. John Wiley and Sons, 2009.

[116] M. Tüchler, "Design of serially concatenated systems depending on the block length," *IEEE Transactions on Communications*, vol. 52, pp. 209–218, February 2004.

[117] T. Dean, *Network+ Guide to Networks*. Course Technology, 2009.

[118] B. Vucetic and J. Yuan, *Turbo Codes: Principles and Applications*. Kluwer Academic Publishers, Norwell, Mass, USA, 2002.

[119] O. Y. Takeshita, "Permutation polynomial interleavers: An algebraic-geometric perspective," *IEEE Transactions on Information Theory*, vol. 53, pp. 2116–2132, June 2007.

[120] *Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding*. 3rd Generation Partnership Project (3GPP), TS 36.212, September 2008.

[121] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *Telecommunications and Data Acquisition Progress Report*, vol. 122, pp. 56–65, April 1995.

[122] R. G. Maunder and L. Hanzo, "Evolutionary algorithm aided interleaver design for serially concatenated codes," *IEEE Transactions on Communications*, vol. 59, pp. 1753–1758, July 2011.

[123] F. Gray, *Pulse Code Communication*. U. S. Patent 2 632 058, March 1953.

[124] B. Sklar, "Rayleigh fading channels in mobile digital communication systems i. characterization," *IEEE Communications Magazine*, vol. 35, pp. 90–100, July 1997.

[125] J. G. Proakis and M. Salehi, *Digital Communications*. McGraw-Hill, 5th ed., 2008.

[126] 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.

[127] P. S. Ahmed, R. G. Maunder, and L. Hanzo, "Partial soft decode and forward," *IEEE Vehicular Technology Conference (VTC Fall)*, pp. 1–5, September 2011.

[128] L. Li, R. G. Maunder, B. M. 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, January 2013.

[129] R. G. Maunder and L. Hanzo, "Extrinsic information transfer analysis and design of block-based intermediate codes," *IEEE Transactions on Vehicular Technology*, vol. 60, pp. 762–770, March 2011.

[130] I. Land, P. Hoeher, and S. Gligorevic, "Computation of symbol-wise mutual information in transmission systems with LogAPP decoders and application to exit charts," in *Proceedings of International Conference on Source and Channel Coding (SCC)*, (Germany), pp. 195–202, January 2004.

[131] 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.

[132] A. Ashikhmin, G. Kramer, and S. ten Brink, "Code rate and the area under extrinsic information transfer curves," in *Proceedings of IEEE International Symposium on Information Theory*, (Lausanne, Switzerland), p. 115, June 2002.

[133] C. E. Shannon, *The mathematical theory of communication*. July 1948.

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

[135] M. Tüchler and J. Hagenauer, "Exit charts of irregular codes," in *Proceddings of Conference on Information Systems and Sciences (CISS)*, (Princeton, NJ), pp. 748–753, March 2002.

[136] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proceedings of the ACM symposium on Theory of Computing*, (El Paso, TX, USA), pp. 150–159, May 1997.

[137] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, Mass.: MIT Press, 1963.

[138] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *IEEE Electronics Letter*, vol. 32, pp. 457–458, August 1996.

[139] B. J. Frey and D. J. C. MacKay, "Irregular turbo-like codes," in *Proceedings of the International Symposium on Turbo Codes*, (Brest, France), pp. 67–72, September 2000.

[140] J. Zou, H. Xiong, C. Li, R. Zhang, and Z. He, "Lifetime and distortion optimization with joint source/channel rate adaptation and network coding-based error control in wireless video sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 60, pp. 1182–1194, March 2011.

[141] Y. Huo, C. Zhu, and L. Hanzo, "Spatio-temporal iterative source-channel decoding aided video transmission," *IEEE Transactions on Vehicular Technology*, vol. 62, pp. 1597–1609, May 2013.

[142] N. Othman, M. El-Hajjar, O. Alamri, S. X. Ng, and L. Hanzo, "Iterative amr-wb source and channel decoding using differential space-time spreading-assisted sphere-packing modulation," *IEEE Transactions on Vehicular Technology*, vol. 58, pp. 484–490, January 2009.

[143] B. Ryabko and J. Rissanen, "Fast adaptive arithmetic code for large alphabet sources with asymmetrical distributions," *IEEE Communications Letters*, vol. 7, pp. 33–35, January 2003.

[144] N. L. Johnson, A. W. Kemp, and S. Kotz, *Univariate Discrete Distributions*. John Wiley and Sons, Inc., Hoboken, NJ, USA, 2005.

[145] 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.

[146] R. G. Maunder, W. Zhang, T. Wang, and L. Hanzo, "Derivations for 'A unary error correction code for the near-capacity joint source and channel coding of symbol values from an infinite set'," [Online]. Available: http://eprints.soton.ac.uk/341736/.

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

[148] D. Divsalar, H. Jin, and R. J. McEliece, "Coding theorems for 'turbo like' codes," in *Proceedings of the 36th Allerton Conference on Communication, Control and Computing*, (Allerton House, USA), pp. 201–210, September 1998.

[149] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Transactions on Communications*, vol. 52, pp. 670–678, April 2004.

[150] J. Kliewer, N. Goertz, 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.

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

[152] L. Li, R. G. Maunder, B. M. 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, January 2013.

[153] J. Hagenauer, "The turbo principle - Tutorial introduction and state of the art," in *Proceedings of the International Symposium on Turbo Codes*, (Brest, France), pp. 1–11, September 1997.

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

[155] 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. 4306–1315, October 2009.

[156] 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*, (Seattle, WA, USA), pp. 2680–2684, July 2006.

[157] F. Brannstrom, L. K. Rasmussen, and A. J. Grant, "Convergence analysis and optimal scheduling for multiple concatenated codes," *IEEE Transactions on Information Theory*, vol. 51, pp. 3354–3364, September 2005.

[158] J. Hagenauer, "The EXIT chart - Introduction to extrinsic information transfer in iterative decoding," in *Proceedings of the European Conference on Signal Proceesing*, (Vienna, Austria), pp. 1541–1548, September 2004.

[159] H. Chen, R. G. Maunder, and L. Hanzo, "Low-complexity multiple-component turbo-decoding-aided hybrid ARQ," *IEEE Transactions on Vehicular Technology*, vol. 60, pp. 1571–1577, May 2011.

[160] S. Benedetto and G. Montorsi, "Iterative decoding of serially concatenated convolutional codes," *IEEE Electronics Letters*, vol. 32, pp. 1186–1188, June 1996.

[161] R. G. 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.

[162] A. Diallo, C. Weidmann, and M. Kieffer, "Efficient computation and optimization of the free distance of variable-length finite-state joint source-channel codes," *IEEE Transactions on Communications*, vol. 59, pp. 1043–1052, April 2011.

[163] D. Sankoff and J. B. Kruskal, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, 1983.

[164] M. Wien, *High Efficiency Video Coding - Coding Tools and Specification*. Berlin, Heidelberg: Springer, 2014.

[165] S. ten Brink, "Rate one-half code for approaching the Shannon limit by 0.1 dB," *IEEE Electronics Letters*, vol. 36, pp. 1293–1294, July 2000.

[166] M. F. Brejza, T. Wang, W. Zhang, D. A. 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.

[167] 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*, December 2015.

Subject Index

N

P

Q

R

S

T

U

V

X

Author Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z