Exponential Golomb and Rice Error Correction Codes for Generalized Near-Capacity Joint Source and Channel Coding

Matthew F. Brejza, Tao Wang, Wenbo Zhang, David Al-Khalili, Robert G. Maunder, Bashir M. Al-Hashimi and Lajos Hanzo
Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK
Email: {mfb2g09, tw1106, wz4g11, rm, bmah, lh}@ecs.soton.ac.uk

Abstract*—The recently proposed Unary Error Correction (UEC) and Elias Gamma Error Correction (EGEC) codes facilitate the near-capacity Joint Source and Channel Coding (JSCC) of symbol values selected from large alphabets at a low complexity. Despite their large alphabet, these codes were only designed for a limited range of symbol value probability distributions. In this paper, we generalize the family of UEC and EGEC codes to the class of Rice and Exponential Golomb (ExpG) Error Correction (RiceEC and ExpGEC) codes, which have a much wider applicability, including the symbols produced by the H.265 video codec, the letters of the English alphabet and in fact any arbitrary monotonic unbounded source distributions. Furthermore, the practicality of the proposed codes is enhanced to allow a continuous stream of symbol values to be encoded and decoded using only fixed-length system components. We explore the parameter space to offer beneficial trade-offs between error correction capability, decoding complexity, as well as transmission-energy, -duration and -bandwidth over a wide range of operating conditions. In each case, we show that our codes offer significant performance improvements over the best of several state-of-the-art benchmarkers. In particular, our codes achieve the same error correction capability, as well as transmission-energy, -duration and -bandwidth as a Variable Length Error-Correction (VLEC) code benchmarker, while reducing the decoding complexity by an order of magnitude. In comparison with the best of the other JSCC and Separate Source and Channel Coding (SSCC) benchmarkers, our codes consistently offer E_b/N_0 gains of between 0.5 dB and 1.0 dB which only appear to be modest, because the system operates close to capacity. These improvements are achieved for free, since they are not achieved at the cost of increasing transmission-energy, -duration, -bandwidth or decoding complexity.

NOMENCLATURE

ACS	Add-Compare-Select
CC	Convolutional Code
CRC	Cyclic Redundancy Check
DCMC	Discrete-Input Continuous-Output Memoryless Channel
ExpGEC	Exponential Golomb Error Correction
EG	Elias Gamma
EGEC	Elias Gamma Error Correction
EXIT	EXtrinsic Information Transfer
FLC	Fixed Length Code
IID	Independent and Identically Distributed
JSCC	Joint Source and Channel Coding
LLR	Logarithmic Likelihood Ratio
Log-BCJR	Logarithmic Bahl-Cocke-Jelinek-Raviv
QPSK	Quadrature Phase Shift Keying
RiceEC	Rice Error Correction
RV	Random Variable
SBSD	Soft Bit Source Decoding
SER	Symbol Error Ratio
SSCC	Separate Source and Channel Coding
UEC	Unary Error Correction
URC	Unity Rate Coding
VLEC	Variable Length Error-Correction

LIST OF SYMBOLS

A_1^o, A_2^o	Area beneath the inverted UEC or FLC-CC EXIT function.
a, b, c	Bit or LLR vector lengths.
C	DCMC capacity.
$\mathbf{d}$	Stream of symbols at the transmitter.
$\mathbf{x}$, $\mathbf{t}$	Stream of sub-symbols at the transmitter.
$\mathbf{y}$, $\mathbf{z}$, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$	Bit vectors at the transmitter.
$\hat{\mathbf{d}}$	Stream of decoded symbols.
$\hat{\mathbf{x}}$, $\hat{\mathbf{t}}$	Stream of decoded sub-symbols.
$\tilde{\mathbf{y}}$	Vector of *a posteriori* LLRs at the receiver.
$\tilde{\mathbf{z}}^\mathrm{a}$, $\tilde{\mathbf{u}}^\mathrm{a}$, $\tilde{\mathbf{v}}^\mathrm{a}$, $\tilde{\mathbf{w}}^\mathrm{a}$	Vector of *a priori* LLRs at the receiver.
$\tilde{\mathbf{z}}^\mathrm{e}$, $\tilde{\mathbf{u}}^\mathrm{e}$, $\tilde{\mathbf{v}}^\mathrm{e}$, $\tilde{\mathbf{w}}^\mathrm{e}$	Vector of extrinsic LLRs at the receiver.
H	Symbol entropy.
k_ExpG	Parameter of the ExpG code.
l_1, l_2	Average codeword length of the unary or FLC.
L	Symbol alphabet cardinality.
M_Rice	Parameter of the Rice code.
n_1, n_2	Codeword length of the UEC trellis and CC trellis.
p_1	Probability of the value 1 in a zeta distribution.
R_1^i, R_2^i	Puncturing or doping rate or the UEC or FLC-CC.
R_1^o, R_2^o	Average coding rate of the UEC or FLC-CC.
r_1, r_2	Number of states in the UEC trellis and CC trellis.
s	Parameter of the zeta function.
x_max, d_max	Maximum value of sub-symbol considered by the FLC decoder.
η	Effective throughput.

The authors wish to gratefully acknowledge the financial support of the ERSPC, Swindon UK under the auspices of grant EP/J015520/1 and EP/L010550/1, as well as the TSB, Swindon UK under the auspices of grant TS/L009390/1. The research data for this paper is available at DOI:10.5258/SOTON/393928

I. INTRODUCTION

The encoding of multimedia information such as video and audio typically results in symbol values that are se-

lected from large or infinite alphabets. For example, the H.265 video encoder represents source video information using transform coefficients and motion vectors [1], which correspond to a large alphabet of symbol values in the range spanning from 1 to around $L = 1000$, as shown in Figure 1a. It may be observed that these symbols obey Zipf's law [2], with low-valued symbols occurring frequently and high valued symbols occurring infrequently, as shown in Figure 1a. Owing to this, the occurrence of these symbol values may be modeled by a zeta probability distribution, as was previously observed for the H.264 video encoder in [3].

In order to facilitate the reliable and bandwidth-efficient transmission of multimedia information, both source coding and channel coding is required. where the state-of-the-art has evolved with the contributions listed in Table I. Shannon [4] postulated that near-capacity operation may be achieved using Separate Source and Channel Coding (SSCC). Here, a near-capacity channel code such as a turbo code [5] or Low-Density Parity-Check (LDPC) code [6] may be combined with a separate near-entropy source code, such as an arithmetic code [7] or Lempel-Ziv code [8]. However, these near-entropy source codes are typically impractical, since they assume that infinite complexity and/or latency can be afforded. For example, both the arithmetic code and Lempl-Ziv code require accurate knowledge of the probability of occurrence of each symbol value at both the transmitter and receiver. This imposes both an excessive complexity and memory requirement when the symbols are selected from an alphabet having a large or infinite cardinality, as it is typical for the encoding of multimedia information. In adaptive schemes, the transmitter and receiver learn the symbol value probabilities from the transmitted symbols, but any transmission errors result in de-synchronization between the transmitter and receiver, introducing a large number of decoding errors from that point onwards [9].

On the other hand, universal codes can encode source symbol values selected from large and infinite alphabets, without incurring the issues associated with near-entropy source codes. More specifically, a source code is said to be universal if it represents the source symbol vector using a bit vector that is guaranteed to have a finite average length for any monotonic source symbol probability distribution. These universal codes include the Elias Gamma code [13], Elias omega code [13], Stout code [15], Fibonacci code [18] and the Exponential Golomb (ExpG) [23] code. These codes are capable of operating without any knowledge of the source symbol value probabilities, granting them immunity to the deleterious synchronization problems that detrimentally affect both the arithmetic and Lempl-Ziv codes. However, typically non-negligible redundancy remains in the encoded bitstreams produced by these source codes, which leads to capacity loss, when combined with a separate channel code. Motivated by this, JSCC [24] may be employed for exploiting the residual redundancy that remains after source encoding for enhancing the attainable error correction capability. A particular example of JSCC is constituted by the classic family of Variable Length Error-Correction (VLEC) codes [25], [26], [27], [28], although they exhibit a decoding complexity, which increases rapidly as the cardinality of the source symbol value alphabet increases, preventing its application for large or infinite alphabets. Against this background, the recently proposed UEC code [3] facilitates the JSCC of source symbol values selected from alphabets having large or infinite cardinalities, while maintaining near-capacity operation and imposing only a modest decoding complexity.

However the UEC code is only practical for a subset of zeta distributions, which does not include those which best model the H.264 and H.265 symbol distributions. More specifically, since a UEC is not a universal code [9], for some zeta distributions the UEC code results in a bit vector having an infinite average length. Motivated by this deficiency of the UEC code, we previously proposed the class of EGEC codes [22]. Since the EGEC code is a universal code created from the class of Elias Gamma (EG) source codes, it produces bit vectors having a finite length for any monotonic probability distribution, including all zeta distributions. Despite its finite-length nature, it still produces long bit vectors for some zeta distributions, hence potentially resulting in low coding rates. To circumvent this problem, excessive puncturing may be required, if a high coding rate is desired, which potentially leads to increased complexity relative to codes having higher original coding rates, as well as to a degraded error correction performance [29]. As a result, the EGEC code family was only characterized for a limited set of zeta distributions in [22]. These limitations of the UEC and EGEC codes imply that neither of them exhibits general applicability, hence indicating that further generalization is required.

TABLE I: Major contributions in source and channel coding.

Year	Author	Contribution
1948	Shannon, C. [4]	The foundations of information theory.
1952	Huffman, D.A. [10]	Huffman source code.
1960	Reed, I.S.; Solomon, G. [11]	Reed-Solomon channel code.
1962	Gallager, R. [6]	The original paper on LDPC channel code.
1967	Viterbi, A.J. [12]	The Viterbi decoding algorithm.
1975	Elias, P. [13]	Elias Gamma code, Elias Omega code.
1977	Massey, J.L. [14]	Non-iterative Joint Source and Channel Coding (JSCC)
1978	Ziv, J.; Lempel, A. [8]	Lempel-Ziv variable rate source code.
1979	Rissanen, J. et al. [7]	Arithmetic near-entropy source coding.
1980	Stout, Q.F. [15]	The Stout source code.
1988	Bernard, M.A. et al. [16]	The VLEC JSCC.
1993	Berrou, C. et al. [17]	The turbo code.
1996	Fraenkel, A.S. et al. [18]	The Fibonacci code.
2000	Bauer, R.; Hagenauer, J. [19]	Introduces iterative decoding of VLECs.
2000, 2001	Görtz, N. [20] [21]	The introduction of iterative JSCC decoding.
2013	Maunder, R.G. et al. [3]	The Unary Error Correction (UEC) JSCC.
2014	Wang, T. et al. [22]	The Elias Gamma Error Correction (EGEC) JSCC.

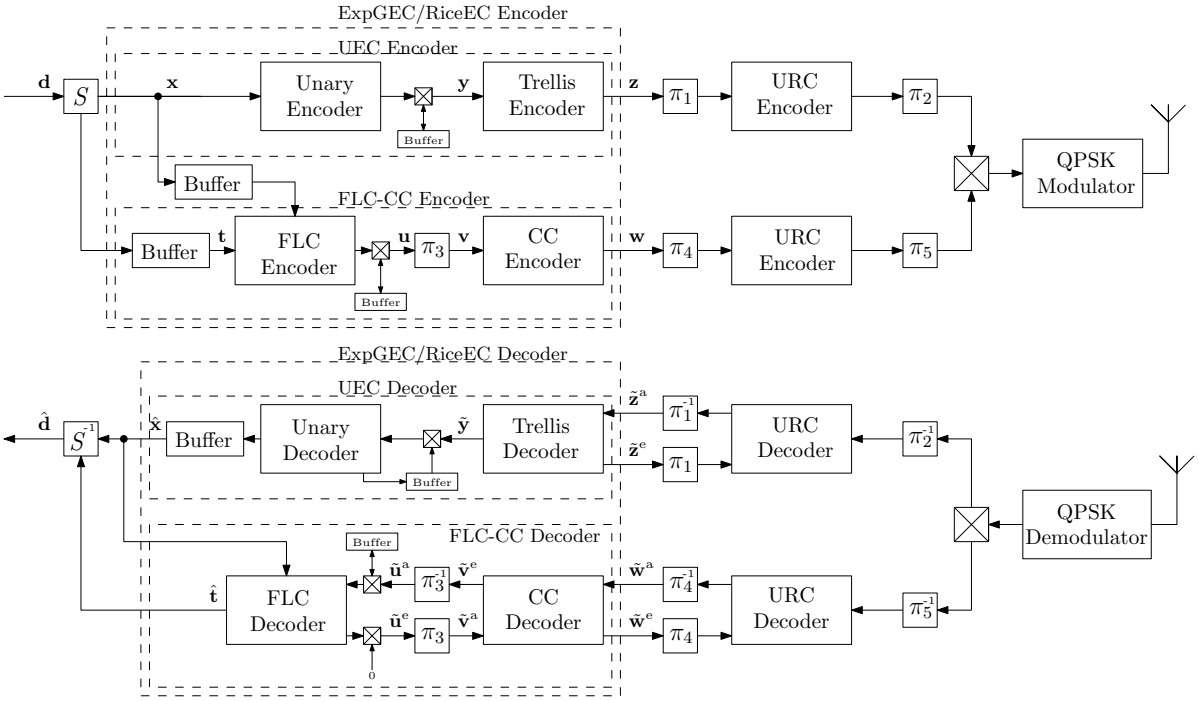


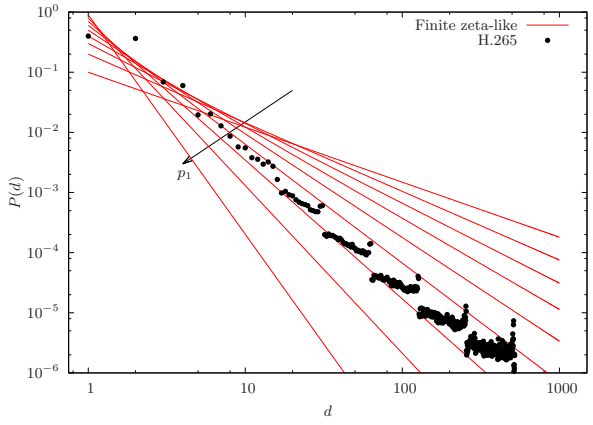
Fig. 2: The proposed ExpGEC/RiceEC schemes. Here, the buffers facilitate operation on the basis of a stream of source symbols, while maintaining fixed-length designs for the interleavers π_1–π_5.

Against this background, this paper extends and generalizes the UEC code family of [3] and the EGEC code of [22], so that they become attractive for encoding symbols produced by *all* zeta distributions, hence granting them general applicability. More specifically, we generalize the EGEC code family by extending its schematic to produce the class of Exponential Golomb Error Correction (ExpGEC) codes, as shown in Figure 2. We also extend the EGEC code to produce the Rice Error Correction (RiceEC) code family, which represents a generalization of the UEC code. We show that the proposed ExpGEC and RiceEC codes offer a better error correction performance over a wider range of source symbol distributions than that of its benchmarkers. In particular, we consider the full range of zeta distributions for the first time, allowing us to best represent a wider range of source symbols, including the distribution of the English alphabet, which has a cardinality of $L = 27$ when including the space character, as characterized in Figure 1b. Furthermore, we consider the specific distribution of symbols produced by the H.265 video encoder, which we have not previously considered. Motivated by this, we critically appraise the infinite-cardinality zeta distribution of our previous work in the specific scenario of finite-cardinality zeta-like distributions, where symbols can assume values in the range $\{1, ..., L\}$. We investigate how a specific choice of the parameter L affects the source symbol distribution, as well as how the parameters of the proposed codes may be best selected for optimizing their error correction performance. For the first time, we also compare the proposed codes and our previous codes to the classic VLEC code, although this benchmarker only has a moderate complexity for low values of L. Additionally, we further improve upon our previous work by improving the practicality of the proposed codes. More specifically, the EGEC scheme of [22] requires five different interleavers, each having different lengths and therefore designs, which change from frame to frame. Not only is this arrangement challenging to implement, but the overall error correction performance is dominated by the worse-case performance, associated with the shortest inverleaver lengths. By contrast, our proposed ExpGEC and RiceEC schemes are designed for encoding continuous streams of symbols, while maintaining constant interleaver-lengths in every frame, hence significantly improving the associated practicality and performance. Furthermore, we prove that these modifications guarantee synchronization between the transmitter and receiver.

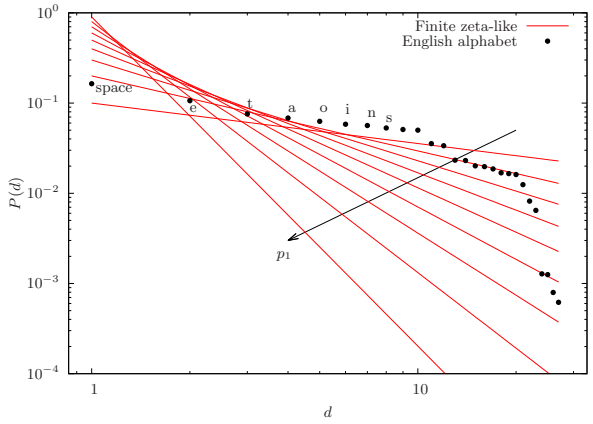
- **II. ExpGEC and RiceEC encoder**
  - Source symbols and their decomposition into sub-symbols
  - UEC sub-symbol encoder
  - FLC sub-symbol encoder
  - Integration into a transmitter
- **III. ExpGEC and RiceEC decoder**
  - Integration into a receiver
  - UEC sub-symbol decoder
  - FLC sub-symbol decoder
- **IV. Fixed length interleavers**
  - Fixed length interleavers for the UEC sub-code
  - Fixed length interleavers for the FLC-CC sub-code
  - Matching sub-symbols
  - Interleavers
- **V. Performance comparison with benchmarkers**
  - Scenarios and benchmarkers
  - Near capacity analysis
  - EXIT chart matching
  - SER performance
- **V. Conclusions**

Fig. 3: The structure of the paper

As shown in Figure 3, we commence our discourse by detailing the operation and characteristics of the proposed



(a) Distribution of H.265 symbols



(b) Distribution of English letters and the space character

Fig. 1: The probability distribution of (a) letters of the English alphabet and the space character, ordered according to descending probability of occurrence, and (b) symbols output from a H.265 encoder. Also shown is the finite zeta-like distribution of (1) for $p_1 = \{0.1, ..., 0.9\}$ and with $L = 27$ and $L = 1000$, respectively.

ExpGEC and RiceEC codes, where Sections II and III consider the operation of the encoders and decoders, respectively. Section IV describes the arrangement proposed for processing streams of symbols, while maintaining fixed interleaver lengths. Section V characterizes the error correction performance of the proposed codes, demonstrating that they are superior to the best of several benchmakers for a wide variety of source distributions, including the full range of zeta distributions, as well as the H.265 and English alphabet distributions. Finally, we offer our conclusions in Section VI.

II. ExpGEC and RiceEC encoder

In this section, we introduce the ExpGEC and RiceEC encoders, which are shown in Figure 2. We commence in Section II-A by describing how the proposed ExpGEC and RiceEC encoders decompose the source symbols into sub-symbols. Following this, Sections II-B and II-C describe how the sub-symbols are encoded by the UEC encoder and the Fixed Length Code-Convolutional Code (FLC-CC) encoder, respectively. Finally, Section II-D describes how the ExpGEC and RiceEC encoders may be integrated into a transmitter, as shown in Figure 2.

A. Source symbols and their decomposition into sub-symbols

As portrayed in Figure 2, the proposed ExpGEC and RiceEC encoders operate on the basis of a stream of source symbols $\mathbf{d} = [d_i]$, which may be modeled as a realization of a stream of Independent and Identically Distributed (IID) Random Variables (RVs) $\mathbf{D} = [D_i]$. In contrast to the infinite symbol alphabet of our previous work [22], we consider source symbols which are randomly selected from a finite symbol set, as described in Section I. This introduces an additional parameter, namely the cardinality L of the source symbol set. More specifically, each RV D_i adopts a value in the set $\{1, 2, 3, ..., L\}$, according to a particular source symbol distribution. Since Figure 1 shows that the H.265 symbols and the letters of the English alphabet obey Zipf's law, we consider a finite-cardinality zeta-like source symbol distribution, where the probability $\Pr(D_i = d) = P(d)$ is given by

$$P(d) = \frac{d^{-s}}{\sum_{\hat{d}=1}^{L} \hat{d}^{-s}} = \frac{d^{-s}}{\bar{\zeta}(s, L)}, \tag{1}$$

with $\bar{\zeta}(s, L) = \sum_{\hat{d}=1}^{L} \hat{d}^{-s}$ being the finite Riemann zeta-like function. Here, the variable $s > 1$ is related to the probability of an RV D_i adopting the value 1 according to $p_1 = \Pr(D_i = 1) = 1/\bar{\zeta}(s, L)$, which parameterizes the finite zeta-like distribution. The entropy of the source symbols is given by

$$H_D = \sum_{d=1}^{L} P(d) \log_2 \frac{1}{P(d)}. \tag{2}$$

Table II shows the first 18 unary, Rice, EG and ExpG codewords, demonstrating how each of the corresponding source encoders maps each symbol d_i from the stream $\mathbf{d}$ to the codeword Unary(d_i), Rice(d_i), EG(d_i) or ExpG(d_i), respectively. Here, the Rice code is parametrized by $M_{\text{Rice}} \in \{1, 2, 4, 8, 16, ...\}$, where $M_{\text{Rice}} = 1$ gives a special case identical to the unary code. Likewise, the ExpG code is parametrized by $k_{\text{ExpG}} \in \{0, 1, 2, 3, 4, ...\}$, where $k_{\text{ExpG}} = 0$ gives a special case identical to the EG code.

The length of an ExpG codeword $\text{ExpG}(d_i)$ may be expressed as $l_{\text{ExpG}}(d_i) = 2\lfloor \log_2(d_i + 2^{k_{\text{ExpG}}} - 1) \rfloor + 1 - k_{\text{ExpG}}$, while the length of a Rice codeword $\text{Rice}(d_i)$ is $l_{\text{Rice}}(d_i) = \lceil d_i / M_{\text{Rice}} \rceil + \log_2(M_{\text{Rice}})$. Note that in the special case of the unary code where $M_{\text{Rice}} = 1$, the length of each corresponding codeword becomes equal to the value of the encoded symbol $l_{\text{Unary}}(d_i) = d_i$. When the source symbols obey the finite zeta-like distribution, the average length $l = \sum_{d=1}^{L} P(d) \cdot l(d)$ of the encoded ExpG, Rice, and unary codewords is given by

$$l_{\text{ExpG}} = 1 - k_{\text{ExpG}} + \frac{1}{\bar{\zeta}(s, L)} \sum_{d=1}^{L} d^{-s} \lfloor \log_2(d + 2^{k_{\text{ExpG}}} - 1) \rfloor, \tag{3}$$

$$l_{\text{Rice}} = \log_2(M_{\text{Rice}}) + \frac{1}{\bar{\zeta}(s, L)} \sum_{d=1}^{L} d^{-s} \lceil d / M_{\text{Rice}} \rceil, \tag{4}$$

$$l_{\text{Unary}} = \frac{\bar{\zeta}(s-1, L)}{\bar{\zeta}(s, L)}. \tag{5}$$

As shown using the dashed lines in Table II, each ExpG and Rice codeword can be viewed as a concatenation of a

TABLE II: The decomposition of symbols d_i into sub-symbols x_i and t_i for the ExpG and Rice codes. The sub-codewords $\mathbf{y}_i$ 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 concatenation of $\mathbf{y}_i$ and $\mathbf{u}_i$ provides the codewords ExpG(d_i) or Rice(d_i), while the dashed line shows how these codewords are decomposed into the sub-codewords.

	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_{ExpG}=0, EG(d_i)				ExpG(d_i) k_{ExpG}=1				ExpG(d_i) k_{ExpG}=2			
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
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
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
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
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
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
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
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
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
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
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
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
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	0010	2
16									4	0001	11	3	2	01	111	7	5	00001	0000	0	4	0001	0001	1	3	001	0011	3
17									5	00001	00	0	3	001	000	0	5	00001	0001	1	4	0001	0010	2	3	001	0100	4
18									5	00001	01	1	3	001	001	1	5	00001	0010	2	4	0001	0011	3	3	001	0101	5

unary prefix $\mathbf{y}_i = \text{Unary}(x_i)$ and Fixed Length Code (FLC) suffix $\mathbf{u}_i = \text{FLC}(t_i)$, where x_i and t_i are the sub-symbols derived from a particular symbol d_i. For each symbol d_i, the value of the sub-symbol x_i is given by

$$\text{ExpG: } x(d) = \lfloor \log_2(d + 2^{k_{\text{ExpG}}} - 1) \rfloor + 1 - k_{\text{ExpG}}, \quad (6)$$

$$\text{Rice: } x(d) = \left\lceil \frac{d}{M_{\text{Rice}}} \right\rceil. \quad (7)$$

Here, the sub-codeword $\mathbf{y}_i = \text{Unary}(x_i)$ comprises $(x_i - 1)$ zeros, followed by a single logical one-valued bit. Likewise, the value of the sub-symbol t_i is given by

$$\text{ExpG: } t(d) = d - 2^{\lfloor \log_2(d + 2^{k_{\text{ExpG}}} - 1) \rfloor} + 2^{k_{\text{ExpG}}} - 1, \quad (8)$$

$$\text{Rice: } t(d) = \text{mod}((d-1), M_{\text{Rice}}), \quad (9)$$

where $\text{mod}(a, b)$ provides the modulo of a when divided by b. Here, the sub-codeword $\mathbf{u}_i = \text{FLC}(t_i)$ is obtained by representing the sub-symbol t_i in a binary form having a particular length. In the case of the ExpG code, the length of this suffix $\mathbf{u}_i = \text{FLC}(t_i)$ depends on the corresponding value of x_i, where $l_{\text{FLC}}(t) = x + k_{\text{ExpG}} - 1$. For the Rice code, this suffix $\mathbf{u}_i = \text{FLC}(t_i)$ has a fixed length $l_{\text{FLC}}(t) = \log_2(M_{\text{Rice}})$, which is independent of the value of x_i or t_i. Given the expressions (6) – (9), we may express d_i as functions of x_i and t_i, according to

$$\text{ExpG: } d(x,t) = 2^{(x + k_{\text{ExpG}} - 1)} - 2^{k_{\text{ExpG}}} + 1 + t, \quad (10)$$

$$\text{Rice: } d(x,t) = M_{\text{Rice}}(x-1) + t + 1. \quad (11)$$

Motivated by the observation that each ExpG and Rice codeword comprises a unary prefix and an FLC suffix, the ExpGEC/RiceEC encoder of Figure 2 employs a splitter S. This uses (6)-(9) to decompose each symbol d_i in the stream $\mathbf{d} = [d_i]$ into the sub-symbols x_i and t_i, which are concatenated to form the streams $\mathbf{x} = [x_i]$ and $\mathbf{t} = [t_i]$. For example, given the symbol vector $\mathbf{d} = [6, 1, 9, 3, 12, 4, 1, 2]$, the $k_{\text{ExpG}} = 1$ ExpGEC splitter yields the sub-symbol vectors $\mathbf{x} = [2, 1, 3, 2, 3, 2, 1, 1]$ and $\mathbf{t} = [3, 0, 2, 0, 5, 1, 0, 1]$, while the $M_{\text{Rice}} = 4$ RiceEC splitter would yield the sub-symbol vectors $\mathbf{x} = [2, 1, 3, 1, 3, 1, 1, 1]$ and $\mathbf{t} = [1, 0, 0, 2, 3, 3, 0, 1]$. Following this, each sub-symbol x_i in the stream $\mathbf{x}$ is encoded by the UEC encoder of Figure 2, which is based on the unary code, as described in Section II-B. Meanwhile, each sub-symbol t_i in the stream $\mathbf{t}$ is encoded by the FLC-CC encoder, which is based on the FLC code, as described in Section II-C.

B. UEC sub-symbol encoder

As shown in Figure 2, the input of the UEC encoder is provided by the stream of sub-symbols $\mathbf{x} = [x_i]$. This stream may be modeled as a realization of a stream of IID RVs $\mathbf{X} = [X_i]$. In the scenario where the RV D_i obeys the finite zeta-like distribution of (1), the probability $\Pr(X_i = x) = P(x)$ is given by

$$\text{ExpG: } P(x) = \frac{1}{\bar{\zeta}(s, L)} \sum_{d=2^{x+k_{\text{ExpG}}-1}-2^{k_{\text{ExpG}}}+1}^{\min(2^{x+k_{\text{ExpG}}}-2^{k_{\text{ExpG}}}, L)} d^{-s}, \quad (12)$$

$$\text{Rice: } P(x) = \frac{1}{\bar{\zeta}(s, L)} \sum_{d=M_{\text{Rice}}(x-1)+1}^{\min(Mx_i, L)} d^{-s}, \quad (13)$$

while the entropy of each RV X_i is given by

$$\text{ExpG: } H_X = \sum_{x=1}^{\lfloor \log_2(L + 2^{k_{\text{ExpG}}} - 1) \rfloor + 1 - k_{\text{ExpG}}} P(x) \log_2 \left(\frac{1}{P(x)} \right), \quad (14)$$

$$\text{Rice: } H_X = \sum_{x=1}^{\left\lceil \frac{L}{M_{\text{Rice}}} \right\rceil} P(x) \log_2 \left(\frac{1}{P(x)} \right). \quad (15)$$

Each sub-symbol x_i in the stream $\mathbf{x}$ is encoded by the unary encoder, which outputs the corresponding x_i-bit unary sub-codeword $\mathbf{y}_i = \text{Unary}(x_i)$, according to Table II. The

average length l_1 of these unary sub-codewords is given by

$$l_1 = \sum_{d=1}^{L} P(d) \cdot x(d), \tag{16}$$

$$\text{ExpG: } l_1 = 1 - k_{\text{ExpG}} + \frac{1}{\bar{\zeta}(s,L)} \sum_{d=1}^{L} \lfloor \log_2(d + 2^{k_{\text{ExpG}}} - 1) \rfloor \cdot d^{-s}, \tag{17}$$

$$\text{Rice: } l_1 = \frac{1}{\bar{\zeta}(s,L)} \sum_{d=1}^{L} \left\lceil \frac{d}{M_{\text{Rice}}} \right\rceil \cdot d^{-s}. \tag{18}$$

In [3], the unary code was employed for encoding the source symbols d_i directly, but this produces long average codeword lengths when p_1 is low, according to (5). However, in the scheme of Figure 2, the unary encoder is used for encoding the sub-symbols x_i instead. Since the sub-symbol probability distributions $P(x)$ of (12) and (13) are skewed towards the most likely symbol value $x = 1$, the UEC code may be used for their encoding without suffering from an excessive average codeword lengths, when p_1 is low. As shown in Figure 2, the codewords in the stream produced by the unary encoder are concatenated and partitioned into a succession of bit-vectors $\mathbf{y} = [y_j]_{j=1}^{b}$, having a fixed length of b bits, as it will be described in Section IV-A. For example, given the sequence $\mathbf{x} = [2, 1, 3, 2, 3, 2, 1, 1]$ comprising 8 sub-symbols, the unary encoder produces the sequence $\mathbf{y} = 011001010010111$ comprising $b = 15$ bits.

The bit vector $\mathbf{y}$ is entered into the trellis encoder of Figure 2, which operates on the basis of the UEC trellis of Figure 4. Here, UEC trellises comprising only $r_1 = 4$ states are adopted, since our previous work [22] showed that this is sufficient for avoiding any significant capacity loss despite its low complexity, as it will be characterized for the codes proposed in Section V-B. However, the option to extend the trellis to more states remains open [3], facilitating the elimination of even more capacity loss, at the cost of increasing the associated complexity. For each successive input bit y_j in $\mathbf{y} = [y_j]_{j=1}^{b}$, the trellis transition emerges from a previous state $m_{j-1} \in \{1, 2, 3, ..., r_1\}$ to a next state $m_j \in \{1, 2, 3, ..., r_1\}$. The path pursued through the UEC trellis when encoding the bit vector $\mathbf{y}$ may be represented as $\mathbf{m} = [m_j]_{j=0}^{b}$, comprising $b + 1$ state values, where m_0 and m_b are the start and end states of the trellis. Each zero-valued bit y_i in the unary-encoded bit vector triggers transitions to states towards the outside of the trellis. By contrast, each one-valued bit causes the trellis to transition back to one of the central states, according to

$$m_j = \begin{cases} 1 + \text{odd}(m_{j-1}) & \text{if } y_j = 1 \\ \min[m_{j-1} + 2, r_1 - \text{odd}(m_{j-1})] & \text{if } y_j = 0 \end{cases}. \tag{19}$$

Here, the function $\text{odd}(\cdot)$ returns zero, if its operand is even, or one if it is odd. Note that each unary codeword comprises a sequence of zero-valued bits which is terminated by a single one-valued bit. The trellis structure of Figure 4 exploits this for maintaining synchronization between the unary-encoded symbols and the path through trellis. In particular, the final one-valued bit y_j in each unary codeword is guaranteed to trigger a transition either to the state $m_j = 1$ or to $m_j = 2$. In the case where the unary-encoded bit vector

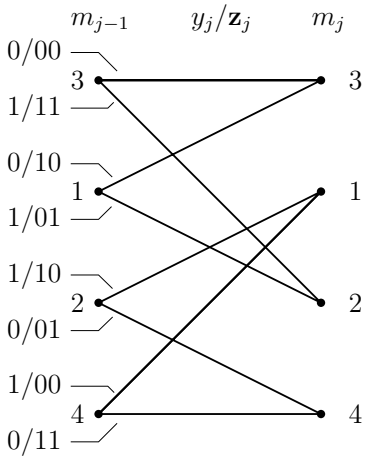


Fig. 4: The UEC trellis utilizing r_1 = 4 states, n_1 = 2 output bits and the codewords $\mathbb{C} = [01; 11]$

$\mathbf{y}$ comprises a complete sequence of unary codewords and the start state is $m_0 = 1$, the path traversed by the trellis encoder will always end in either the state $m_b = 1$ or $m_b = 2$, depending on whether the number of sub-symbols in $\mathbf{x}$ is odd or even. For example, given the unary-encoded bit vector comprising the $b = 15$ bits of $\mathbf{y} = 011001010010111$, the path through the trellis can be formulated as the vector $\mathbf{m} = [1, 3, 2, 1, 3, 3, 2, 4, 1, 3, 3, 2, 4, 1, 2, 1]$ of $b + 1 = 16$-states. The path $\mathbf{m}$ may be modeled as a realization of a vector of RVs $\mathbf{M} = [M_j]_{j=0}^{b}$, where the probability of each state being selected $\Pr(M_j = m | M_{j-1} = m') = P(m|m')$, is given by [3, Eqn. (9)]. The knowledge of these conditional transition probabilities $P(m|m')$ may be exploited to aid the receiver, as described in Section III-B.

Depending on the path selected through the UEC trellis, each of the bits in the unary-encoded bit vector $\mathbf{y}$ is encoded using a n_1-bit codeword $\mathbf{z}_j$, which are concatenated to form the bn_1-bit UEC-encoded vector $\mathbf{z} = [z_k]_{k=1}^{bn_1}$. Each codeword $\mathbf{z}_j$ is selected from the set of $r_1/2$ codewords $\mathbb{C} = [\mathbf{c}_1; \mathbf{c}_2; \mathbf{c}_3; ...; \mathbf{c}_{r_1/2}]$ or from the complementary set $\overline{\mathbb{C}} = [\overline{\mathbf{c}_1}; \overline{\mathbf{c}_2}; \overline{\mathbf{c}_3}; ...; \overline{\mathbf{c}_{r_1/2}}]$, according to

$$\mathbf{z}_j = \begin{cases} \overline{\mathbf{c}_{\lceil m_{j-1}/2 \rceil}} & \text{if } y_j \neq \text{odd}(m_{j-1}) \\ \mathbf{c}_{\lceil m_{j-1}/2 \rceil} & \text{if } y_j = \text{odd}(m_{j-1}) \end{cases}. \tag{20}$$

For example, the trellis of Figure 4 uses the set of codewords $\mathbb{C} = [01; 11]$, as well as the complementary set $\mathbb{C} = [10; 00]$. In the case of the example path through the trellis $\mathbf{m}$ provided above, the UEC-encoded bit vector $\mathbf{z} = 101110100011010010001101000110$ comprising $bn_1 = 30$ bits is produced, when using the $r_1 = 4$-state, $n_1 = 2$-bit trellis of Figure 4. Since the top and bottom halves of the UEC trellis use complementary codewords, the UEC-encoded bits of $\mathbf{z}$ are guaranteed to have equiprobable binary values. Due to this equiprobablity, the average coding rate of the UEC encoder is given by

$$R_1^{\text{o}} = \frac{H_X}{l_1 n_1}. \tag{21}$$

Here the superscript 'o' is used to indicate this coding rate relates to the outer code of a serial concatenation, namely the UEC code of Figure 2.

C. FLC-CC sub-symbol encoder

As shown in Figure 2, the FLC-CC sub-symbol encoder is used for encoding the sub-symbol stream $\mathbf{t} = [t_i]$. Here, the FLC encoder of Figure 2 represents each sub-symbol t_i using a codeword $\mathbf{u}_i$, which is given by the fixed-point binary representation of t_i, having a particular length that may depend on the particular value of the corresponding sub-symbol x_i, as described in Section II-B. Motivated by this, we may model the sub-symbol stream $\mathbf{t}$ as a realization of a stream of RVs $\mathbf{T} = [T_i]$, where each RV T_i is dependent on the corresponding RV X_i. More specifically, in the case where the RV D_i obeys the finite zeta-like distribution of (1), the joint probability $\Pr(T_i = t \cap X_i = x) = P(t \cap x)$ may be obtained according to

$$\text{ExpG: } P(t \cap x) = \frac{1}{\bar{\zeta}(s, L)} \left(2^{x+k_{\text{ExpG}}-1} - 2^{k_{\text{ExpG}}} + 1 + t\right)^{-s} \tag{22}$$

$$\text{Rice: } P(t \cap x) = \frac{1}{\bar{\zeta}(s, L)} \left(M_{\text{Rice}}(x-1) + t + 1\right)^{-s}, \tag{23}$$

where $0 \leq t < 2^{x-1+k_{\text{ExpG}}}$ for the case of the ExpG and $0 \leq t < M_{\text{Rice}}$ for the case of the RiceEC code.

These joint probabilities may be used for obtaining expressions for the corresponding conditional probabilities, which may be exploited in the receiver to aid FLC-CC decoding. In the case of the finite zeta-like distribution, the conditional probability $\Pr(T_i = t | X_i = x) = P(t|x)$ is given by

$$P(t|x) = \frac{P(t \cap x)}{P(x)}, \tag{24}$$

$$\text{ExpG: } P(t|x) = \frac{\left(2^{x+k_{\text{ExpG}}-1} - 2^{k_{\text{ExpG}}} + 1 + t\right)^{-s}}{\sum_{d=2^{x+k_{\text{ExpG}}-1}-2^{k_{\text{ExpG}}}+1}^{\min(2^{x+k_{\text{ExpG}}}-2^{k_{\text{ExpG}}}, L)} d^{-s}}, \tag{25}$$

$$\text{Rice: } P(t|x) = \frac{\left(M_{\text{Rice}}(x-1) + t + 1\right)^{-s}}{\sum_{d=M_{\text{Rice}}(x-1)+1}^{\min(Mx, L)} d^{-s}}, \tag{26}$$

where $0 \leq t < 2^{x-1+k_{\text{ExpG}}}$ for the case of the ExpGEC code and $0 \leq t < M_{\text{Rice}}$ for the RiceEC code. Finally, the conditional entropy of the RV T_i is given by

$$H_{T|X} = \sum_{d=1}^{L} P(t \cap x) \cdot \log_2 \left(\frac{1}{P(t|x)}\right). \tag{27}$$

As described in Section II-B, each FLC codeword $\mathbf{u}_i$ has the length $x_i + k_{\text{ExpG}} - 1$ in the case of the ExpGEC code, where x_i is the corresponding sub-symbol in the stream $\mathbf{x}$. Owing to this, the FLC-CC encoder requires knowledge of the symbol stream $\mathbf{x}$, as shown in Figure 2. In the case of the RiceEC code, the length of the FLC codewords is fixed at $\log_2(M_{\text{Rice}})$ and so the knowledge of $\mathbf{x}$ is not required during FLC-CC encoding in this case. When the sub-symbols of $\mathbf{d}$ obey the finite zeta-like distribution of (1), the average length of the FLC codewords l_2 is given by

$$\text{ExpG: } l_2 = \frac{1}{\bar{\zeta}(s, L)} \sum_{d=1}^{L} \lfloor \log_2(d + 2^{k_{\text{ExpG}}} - 1) \rfloor \cdot d^{-s}, \tag{28}$$

$$\text{Rice: } l_2 = \log_2(M_{\text{Rice}}). \tag{29}$$

Following FLC encoding, the resultant FLC codewords are then concatenated together to form a bit-stream, which is then partitioned into bit-vectors $\mathbf{u} = [u_j]_{j=1}^{c}$, comprising c number of bits, as it will be detailed in Section IV-B. In our example of using the $k_{\text{ExpG}} = 1$ ExpGEC code to encode the sub-symbol vectors $\mathbf{x} = [2, 1, 3, 2, 3, 2, 1, 1]$ and $\mathbf{t} = [3, 0, 2, 0, 5, 1, 0, 1]$, Table II may be used for producing the $c = 15$-bit FLC-encoded vector $\mathbf{u} = 110010001010101$. In the case of the $M_{\text{Rice}} = 4$ RiceEC code, the FLC encoding of the sub-symbol vector $\mathbf{t} = [1, 0, 0, 2, 3, 3, 0, 1]$ produces the $c = 16$-bit vector $\mathbf{u} = 0100001011110001$.

As shown in Figure 2, the FLC-encoded bit vector $\mathbf{u}$ is interleaved in the block π_3, in order to provide the bit vector $\mathbf{v} = [v_j]_{j=1}^{c}$. This is then encoded by a r_2-state, n_2-bit non-systematic recursive CC encoder having a design selected from [3, Table II], in order to obtain the bit vector $\mathbf{w} = [w_k]_{k=1}^{cn_2}$. The CC codes of [3, Table II] were shown in our previous work [22] to mitigate capacity loss. More explicitly, while the bits of $\mathbf{u}$ are not guaranteed to have equiprobable values, the non-systematic recursive nature of these CCs guarantees equiprobable values for the bits of $\mathbf{w}$, which mitigates capacity loss [3]. For example, the interleaver $\pi_3 = [6, 14, 11, 10, 3, 1, 5, 8, 13, 16, 15, 2, 4, 7, 12, 9]$ may be employed for transforming the example $c = 16$-bit vector $\mathbf{u}$ provided above into the $c = 16$-bit vector $\mathbf{v} = 0011000001010111$, where $v_j = u_{\pi_3(j)}$. When the $r_2 = 4$-state, $n_2 = 2$-bit CC encoder of [3, Table II] is employed, this bit vector $\mathbf{v}$ is encoded into the $cn_2 = 32$-bit FLC-CC-encoded bit vector $\mathbf{w} = 00001101010000000011100001110110$. The octally represented generator polynomials invoked for this CC encoder are [4,7], while the octal feedback polynomial is 6. Finally, the average coding rate of the FLC-CC encoder is given by

$$R_2^{\text{o}} = \frac{H_{T|X}}{l_2 n_2}. \tag{30}$$

D. Integration into a transmitter

The RiceEC and ExpGEC encoders may be integrated into a transmitter by serially concatenating them with an inner code. In the scheme of Figure 2, the bit vectors $\mathbf{z}$ and $\mathbf{w}$ provided by the UEC and FLC-CC encoders are interleaved in the blocks π_1 and π_4, before being encoded by 2-state Unity Rate Coding (URC) encoders, which behave as accumulators. As discussed in Section III-A, these URC codes will facilitate iterative decoding in the receiver, enabling near capacity operation [30].

Following URC encoding, the pair of resultant bit vectors are interleaved and optionally punctured or doped in the blocks π_2 and π_5 of Figure 2, before they are multiplexed and QPSK modulated onto the channel using Gray mapping. Here, puncturing or doping may be employed for achieving a particular effective target throughput η for the scheme. As we will discuss further in Section V-C, puncturing and doping may also be employed to provide unequal error protection for the UEC and FLC-CC parts of the schemes, in order to facilitate operation at lower channel SNRs. Puncturing is achieved in the blocks π_2 and π_5 of Figure 2 and by removing the appropriate number of bits from the end of the interleaved bit vector. By contrast, doping is achieved in the blocks π_2 or π_5 by duplicating the appropriate number of bits from the end of the interleaved bit vector and concatenating them. The puncturing and doping rates of π_2 and π_5 are represented by R_1^{i} and R_2^{i} respectively, which

quantifies their ratio of input to output bits. Here, a value of $R^{\mathrm{i}} > 1$ represents puncturing, where $R^{\mathrm{i}} < 1$ represents doping. For example, $R^{\mathrm{i}} = 0.75$ implies that one third of the original bits have been duplicated, while $R^{\mathrm{i}} = 1.25$ implies that one fifth of the original bits have been discarded. Here, the superscript i indicates relevance to the inner code. The transmitter's overall effective throughput in bits per symbol is given by

$$\eta = \frac{H_D \log_2(M_{\mathrm{mod}})}{l_1 n_1 / R_1^{\mathrm{i}} + l_2 n_2 / R_2^{\mathrm{i}}}, \tag{31}$$

where M_{mod} is the number of constellation points used by the modulator, which is $M_{\mathrm{mod}} = 4$ in the case of QPSK.

III. ExpGEC and RiceEC decoder

In this section, we detail the operation of the ExpGEC and RiceEC decoders of Figure 2. Section III-A discusses the integration of the UEC and FLC-CC sub-symbol decoders of the ExpGEC and RiceEC decoders into the receiver of Figure 2. Following this, the operation of the UEC sub-symbol decoder is described in Section III-B, while the operation of the FLC-CC sub-symbol decoder is described in Section III-C.

A. Integration into a receiver

In the receiver of Figure 2, the QPSK demodulator converts each received QPSK symbol into a pair of Logarithmic Likelihood Ratios (LLRs), which pertain to the bits at the transmitter. These LLRs convey the likelihood of the corresponding bits being 1-valued or 0-valued. More specifically, each LLR is defined by $\mathrm{LLR} = \ln \frac{Pr(\mathrm{bit}=1)}{Pr(\mathrm{bit}=0)}$, where a large positive valued LLR represents a high confidence that the bit has the value of logical one, while a large negative-valued LLR represents a high confidence that the bit is logical zero-valued. The LLRs are then demultiplexed and entered into the blocks π_2^{-1} and π_5^{-1} of Figure 2. These blocks undertake de-puncturing or de-doping as appropriate. De-puncturing is achieved by replacing each of the bits removed during puncturing with a zero-valued LLR, which reflects the absence of any knowledge about the value of the corresponding bit. By contrast, de-doping is achieved by removing the LLRs pertaining to the replicas of the duplicated bits and summing them into the LLRs pertaining to the corresponding duplicated bits. Following de-puncturing or de-doping the resultant LLRs are deinterleaved and forwarded to the URC decoders, which iteratively exchange their vectors of LLRs with the UEC decoder or the FLC-CC decoder, as appropriate. More specifically, the UEC trellis decoder and first URC decoder perform iterative decoding by exchanging the LLR vectors $\tilde{\mathbf{z}}^{\mathrm{a}}$ and $\tilde{\mathbf{z}}^{\mathrm{e}}$, which pertain to the bits in $\mathbf{z}$, as shown in Figure 2. Likewise, the CC decoder and second URC decoder perform iterative decoding by exchanging the LLR vectors $\tilde{\mathbf{w}}^{\mathrm{a}}$ and $\tilde{\mathbf{w}}^{\mathrm{e}}$, which pertain to the bits of $\mathbf{w}$. The UEC trellis decoder, CC decoder and URC decoder of Figure 2 invoke the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm [5] for converting the input *a priori* LLR vector into the extrinsic LLR output vector, as may be characterized by their EXtrinsic Information Transfer (EXIT) functions, exemplified in Figure 5.

Before initiating the first decoding iteration, the LLR vectors $\tilde{\mathbf{w}}^{\mathrm{e}}$ and $\tilde{\mathbf{z}}^{\mathrm{e}}$ are populated with zero-valued LLRs.

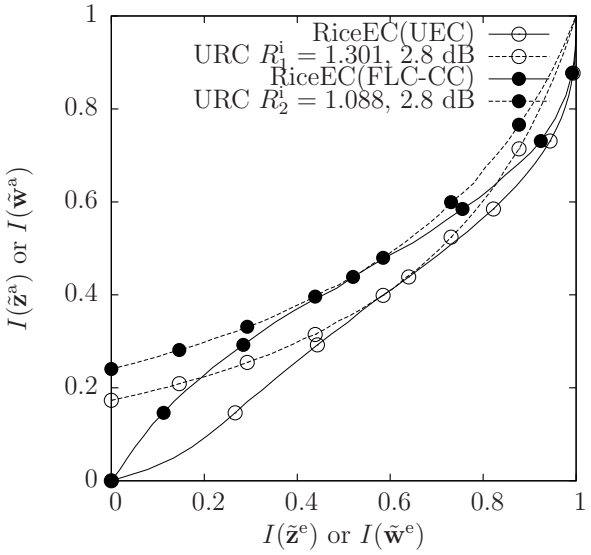


Fig. 5: EXIT charts for the UEC and FLC-CC sub-codes of the proposed RiceEC scheme, when the symbols of $\mathbf{d}$ obey a finite zeta-like distribution having the parameters $p_1 = 0.2$ and $L = 1000$. The RiceEC scheme adopts the parameters $M_{\mathrm{Rice}} = 32$, $n_1 = 2$, $n_2 = 2$ and $\eta = 0.9$, for the case of using QPSK modulation over an uncorrelated narrowband Rayleigh fading channel. Here, the unequal error protection of the UEC and FLC-CC sub-codes relies on different puncturing rates R_1^{i} and R_2^{i}, in order to ensure that the corresponding EXIT charts open at the same E_{b}/N_0 value of 2.8 dB.

The URC decoders then invoke the Log-BCJR algorithm for converting the LLR vectors provided by the demodulator into extrinsic LLR vectors that can be used for iterative decoding. The deinterleavers π_1^{-1} and π_4^{-1} of Figure 2 convert these extrinsic LLR vectors into the *a priori* LLR vectors $\tilde{\mathbf{z}}^{\mathrm{a}} = [\tilde{z}_k^{\mathrm{a}}]_{k=1}^{bn_1}$ and $\tilde{\mathbf{w}}^{\mathrm{a}} = [\tilde{w}_k^{\mathrm{a}}]_{k=1}^{cn_2}$, respectively. The UEC decoder and FLC-CC decoder then operate as described in Sections III-B and III-C, in order to generate the extrinsic LLR vectors $\tilde{\mathbf{z}}^{\mathrm{e}} = [\tilde{z}_k^{\mathrm{e}}]_{k=1}^{bn_1}$ and $\tilde{\mathbf{w}}^{\mathrm{e}} = [\tilde{w}_k^{\mathrm{e}}]_{k=1}^{cn_2}$. The interleavers π_1 and π_4 of Figure 2 convert these extrinsic LLR vectors into *a priori* LLR vectors that can be provided to the URC decoders. The iterations between the decoders continue until a fixed complexity limit has been reached or until the error-free decoding of $\hat{\mathbf{x}}$ and $\hat{\mathbf{t}}$ have been achieved, which may be detected using a Cyclic Redundancy Check (CRC) in practice. Note that the iterative UEC decoding must be completed before the iterative FLC-CC decoding can begin, since the latter takes the output $\hat{\mathbf{x}}$ of the former as its input, as will be discussed in Section III-C. Finally, the symbols $\hat{\mathbf{d}}$ are recovered using the de-splitter S^{-1}, which operates according to Equations (10) and (11).

B. UEC sub-symbol decoder

In this section, we describe the operation of the UEC sub-symbol decoder of Figure 2. Here, the trellis decoder invokes the Log-BCJR algorithm for the trellis of Figure 4 in order to convert the vector of *a priori* LLRs $\tilde{\mathbf{z}}^{\mathrm{a}}$ into the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}$, as well as the vector of extrinsic LLRs $\tilde{\mathbf{z}}^{\mathrm{e}}$. This extrinsic LLR vector $\tilde{\mathbf{z}}^{\mathrm{e}}$ is exchanged with the URC decoder in order to facilitate iterative

decoding, as described in Section III-A. The performance of the UEC trellis decoder may be enhanced by exploiting the knowledge of the conditional transition probabilities $P(m_j|m_{j-1})$ of the trellis, as discussed in Section II-B. More specifically, the decoder requires the knowledge of the average unary codeword length l_1 and of the first $r_1/2-1$ values of $P(x)$ in order to compute the transition probabilities $P(m_j|m_{j-1})$ according to [3, Eqn. (9)]. These conditional transition probabilities $P(m_j|m_{j-1})$ may be exploited during the computation of the *a priori* transition probabilities γ, which are employed by the logarithmic version of the BCJR algorithm [17]. Note that if the source distribution is unknown at the receiver, the trellis decoder may initially operate without the transition probabilities $P(m_j|m_{j-1})$, at the cost of a reduced error correction performance [22]. Once a sufficiently high number of frames have been decoded, the value of l_1 and the first $r_1/2-1$ values of $P(x)$ may be estimated heuristically, and then exploited for improving the error correction performance.

The transformation of $\tilde{\mathbf{z}}^{\mathrm{a}}$ into $\tilde{\mathbf{z}}^{\mathrm{e}}$ may be characterized by the UEC trellis decoder's inverted EXIT function [31], as shown in Figure 5. Note that for both the ExpGEC and RiceEC codes, the area A_1^{o} beneath the inverted UEC EXIT function may be closely approximated by [3], [32]

$$\begin{aligned} A_1^{\mathrm{o}} = &\frac{1}{l_1 n_1}\sum_{x=1}^{r_1/2-1}\big(H[P(x)]\big) + \frac{2}{l_1 n_1} H\left[1-\sum_{x=1}^{r_1/2-1} P(x)\right] \\ &+ \frac{1}{l_1 n_1} H\left[l_1 - R_1/2 + \sum_{x=1}^{r_1/2-1} P(x)(r_1/2-x)\right] \\ &- \frac{1}{l_1 n_1} H\left[l_1 + 1 - r_1/2 + \sum_{x=1}^{r_1/2-1} P(x)(r_1/2-1-x)\right]. \end{aligned} \tag{32}$$

Once a sufficiently high number of iterations have been completed between the UEC trellis decoder and the URC decoder, the trellis decoder outputs the *a posteriori* LLR vector $\tilde{\mathbf{y}} = [\tilde{y}_j]_{j=1}^{b}$, which is forwarded to the unary decoder, as shown in Figure 2. Note that since each unary codeword contains only a single logical one-valued bit, there are guaranteed to be a number of one-valued bits in the unary-encoded bit vector $\mathbf{y}$. The unary decoder exploits this observation by making logical one-valued hard decisions for the a highest LLR values in the *a posteriori* LLR vector $\tilde{\mathbf{y}}$, since high LLR values indicate that the corresponding bit in $\mathbf{y}$ is likely to have a logical value of one. Following this, zero-valued hard decisions are selected for the remaining $(b-a)$ LLRs in $\tilde{\mathbf{y}}$. Note that in practice the value of a may be reliably conveyed to the receiver using a small amount of side information. Alternatively, a logical one-valued hard decision may be selected for all positive LLRs in $\tilde{\mathbf{y}}$, while 0 may be selected for all the other LLRs, hence avoiding the requirement of side information, but degrading the error correction performance. Finally, the unary decoder then converts the hard decision bit vector $\tilde{\mathbf{y}}$ into sub-symbols $\tilde{\mathbf{x}}$ according to Table II.

C. FLC sub-symbol decoder

This section describes the FLC-CC decoder, which iteratively exchanges LLRs with the corresponding URC decoder of Figure 2. More specifically, the CC decoder invokes the Log-BCJR algorithm for processing the *a priori* LLR vectors $\tilde{\mathbf{w}}^{\mathrm{a}} = [\tilde{w}_k^{\mathrm{a}}]_{k=1}^{cn_2}$ and $\tilde{\mathbf{v}}^{\mathrm{a}} = [\tilde{v}_j^{\mathrm{a}}]_{j=1}^{c}$, where the latter adopts all zero values at the start of the iterative decoding process. In response, the CC decoder generates the extrinsic LLR vectors $\tilde{\mathbf{w}}^{\mathrm{e}} = [\tilde{w}_k^{\mathrm{e}}]_{k=1}^{cn_2}$ and $\tilde{\mathbf{v}}^{\mathrm{e}} = [\tilde{v}_j^{\mathrm{e}}]_{j=1}^{c}$, where the latter is iteratively exchanged with the FLC decoder of Figure 2. More specifically, $\tilde{\mathbf{v}}^{\mathrm{e}}$ is de-interleaved in the block π_3^{-1} for providing the *a priori* LLR vector $\tilde{\mathbf{u}}^{\mathrm{a}} = [\tilde{u}_j^{\mathrm{a}}]_{j=1}^{c}$ for the FLC decoder. The FLC decoder employs the Soft Bit Source Decoding (SBSD) algorithm of [33] for converting the *a priori* LLR vector $\tilde{\mathbf{u}}^{\mathrm{a}}$ into the decoded sub-symbol vector $\hat{\mathbf{t}} = [t_i]_{i=1}^{a}$ and the extrinsic LLR vector $\tilde{\mathbf{u}}^{\mathrm{e}} = [\tilde{u}_j^{\mathrm{e}}]_{j=1}^{c}$, which is interleaved in the block π_3 to provide the *a priori* LLR vector $\tilde{\mathbf{v}}^{\mathrm{a}} = [\tilde{v}_j^{\mathrm{a}}]_{j=1}^{c}$ for the next iteration of the CC decoder.

As shown in Figure 2, the FLC decoder requires the knowledge of the decoded sub-symbol vector $\hat{\mathbf{x}} = [\hat{x}_i]_{i=1}^{a}$, which is provided by the UEC decoder, as described in Section III-B. This allows the SBSD algorithm employed by the FLC decoder to exploit the knowledge of the conditional sub-symbol probabilities $P(t|x)$ [33], where the corresponding decoded sub-symbol $\hat{x}_i$ is employed as a substitute for x_i, when decoding the sub-symbol t_i. Note that if the source distribution is unknown at the receiver, the SBSD algorithm may be initially operated without the conditional sub-symbol probabilities $P(t|x)$, at the cost of a reduced error correction performance. Once a sufficient number of frames have been decoded, these probabilities may be heuristically estimated and exploited to restore the error correction performance. In the case of the ExpGEC code, the SBSD algorithm also requires the decoded sub-symbols of $\hat{\mathbf{x}}$ in order to determine the number of LLRs of $\tilde{\mathbf{u}}^{\mathrm{a}}$ that should be used to recover each of the sub-symbols of $\hat{\mathbf{t}}$. More specifically, the number of LLRs corresponding to the sub-symbol t_i is given by $l_{\mathrm{FLC}(t_i)} = x_i - 1 + k_{\mathrm{ExpG}}$, as previously described in Section II-C. For the RiceEC code, by contrast, this length is fixed at $l_{\mathrm{FLC}}(t_i) = \log_2(M_{\mathrm{Rice}})$. Note that the complexity of the SBSD algorithm increases exponentially with the length $l_{\mathrm{FLC}}(t_i)$ of the codeword it is tasked to decode, since it considers each of the codeword's $2^{l_{\mathrm{FLC}}(t_i)}$ possible values. Since $l_{\mathrm{FLC}}(t_i)$ grows with x_i in the case of the ExpGEC code, we may limit the FLC decoding complexity by only invoking the SBSD algorithm for calculating the extrinsic LLRs of $\tilde{\mathbf{u}}^{\mathrm{e}}$ for the specific symbols satisfying $\hat{x} \geq x_{\max} = \lfloor \log_2(d_{\max} + 2^{k_{\mathrm{ExpG}}}) \rfloor + 1 - k_{\mathrm{ExpG}}$, where we recommend a parameter value of $d_{\max} = 18$ for striking an attractive trade-off between the decoding complexity imposed and the error correction capability attained. This approach also has the benefit of limiting the number of conditional sub-symbol probabilities $P(t_i|x_i)$ that are exploited by the SBSD algorithm and that must therefore be known and stored by the receiver. For this reason, the SBSD algorithm is only applied in the case of the RiceEC code for the particular symbols satisfying $\hat{x}_i \geq x_{\max} = \lceil d_{\max}+1)/M_{\mathrm{Rice}} \rceil$, where we also recommend $d_{\max} = 18$. The exception to this is found in the cases where we have $M_{\mathrm{Rice}} = 32$ or $M_{\mathrm{Rice}} = 64$, when we recommend the choice of $d_{\max} = M_{\mathrm{Rice}}$, for the sake of offering an attractive error correction capability. The SBSD algorithm is not applied for all other symbols where $\hat{x}_i > x_{\max}$. Instead, zero-values are used for the corresponding extrinsic LLRs of

$\tilde{\mathbf{u}}^{\mathrm{e}}$, while the corresponding decoded sub-symbols of $\hat{\mathbf{t}}$ are obtained by applying hard decisions to the corresponding LLRs of $\tilde{\mathbf{u}}^{\mathrm{a}}$.

During each decoding iteration, the URC, CC, and FLC decoders are each activated in turn, according to the schedule {URC, CC, FLC, URC, CC, FLC, ... }. For symbols satisfying $\hat{x}_i \leq x_{\max}$, this schedule represents a three-stage iterative decoding process, but for symbols where $\hat{x}_i > x_{\max}$, this is effectively a two-stage iterative decoding process between the CC and URC decoders.

The transformation of $\tilde{\mathbf{w}}^{\mathrm{a}}$ into $\tilde{\mathbf{w}}^{\mathrm{e}}$ may be characterized by the FLC-CC decoder's inverted EXIT function, as shown in Figure 5. Note that the area beneath the inverted FLC-CC EXIT function may be closely approximated by (33) and (34)[1] [22].

IV. Fixed length interleavers

As described in Section II-D, the EGEC scheme of [22] employs five interleavers having specific lengths and therefore particular designs that change from frame-to-frame, hence limiting the practicality of the EGEC scheme. This may be attributed to the EGEC scheme's partitioning of the sub-symbol streams $\mathbf{x}$ and $\mathbf{t}$ into fixed length vectors, which are encoded using variable length codewords to produce bit vectors having lengths that change from frame-to-frame, Motivated by this, this section describes a novel approach that allows our proposed ExpGEC and RiceEC schemes of Figure 2 to use single fixed length designs for the interleavers π_1 to π_5 for each frame, significantly improving its practicality and error correction capability, as discussed in Section I. More specifically, the proposed approach encodes the sub-symbol streams $\mathbf{x}$ and $\mathbf{t}$ into bit-streams, which are partitioned into bit-vectors $\mathbf{y} = [y_j]_{j=1}^{b}$ and $\mathbf{u} = [u_j]_{j=1}^{c}$ having fixed lengths of b and c for the UEC and FLC-CC sub-codes, respectively. Our proposed solution is designed for accommodating unary and FLC codewords that span across frames, as well as for maintaining synchronization between the UEC and FLC-CC sub-codes of the decoder. We commence in Section IV-A by detailing how the UEC sub-code partitions the corresponding bit stream into the fixed length bit vector $\mathbf{y}$, in order to ensure that π_1 and π_2 of Figure 2 have fixed lengths. Likewise, Section IV-B describes how the FLC-CC sub-code partitions the corresponding bit stream into the fixed length bit vector $\mathbf{u}$, in order to ensure that π_3, π_4 and π_5 of Figure 2 have fixed lengths. Section IV-C proves how synchronization is maintained between the UEC and FLC-CC sub-codes, while Section IV-D discusses the specific design of the interleavers π_1 to π_5.

A. Fixed length interleavers for the UEC sub-code

As described in Section II-B, the unary encoder of Figure 2 converts each sub-symbol x_i in the stream $\mathbf{x}$ into the corresponding unary codeword $\mathbf{y}_i$. These codewords are concatenated to form a bit stream, which is partitioned into fixed length vectors $\mathbf{y} = [y_j]_{j=1}^{b}$, which may cause some unary codewords to be split between consecutive frames. In order to address this, the scheme of Figure 2 employs a buffer to store the last bits in the codeword that do not fit into the current frame, so that they can be concatenated onto the start of the next frame. For example, when unary encoding the sub-symbol vector $\mathbf{x} = [1, 2, 1, 1, 4, 1, 3, 1, 2]$ to form bit vectors comprising $b = 8$ bits, we obtain $\mathbf{y} = [1, 0, 1, 1, 1, 0, 0, 0]$ and $\mathbf{y} = [1, 1, 0, 0, 1, 1, 0, 1]$ for two consecutive frames. Note that the unary codeword corresponding to the fifth sub-symbol of $\mathbf{x}$ is split between the two bit-vectors of $\mathbf{y}$. Owing to this, the first and last bits of the bit vector $\mathbf{y}$ are not guaranteed to be the first and last bits of a unary codeword,which is in contrast to the usual UEC operation. The UEC trellis encoder is capable of accommodating this change in two ways.

In a **first method** for accommodating codewords split between two consecutive frames, the UEC trellis encoder may carry over its state between successive frames. The transmitter then uses a small amount of additional side information for reliably conveying this state to the receiver. The UEC trellis decoder may use this side information to initialize the end state of one frame, as well as the initial state of the next. For example, when employing this approach for the two successive bit vectors $\mathbf{y} = [1, 0, 1, 1, 1, 0, 0, 0]$ and $\mathbf{y} = [1, 1, 0, 0, 1, 1, 0, 1]$ from our previous example, the paths transversed through the UEC trellis are given by $\mathbf{m} = [1, 2, 4, 1, 2, 1, 3, 3, 3]$ and $\mathbf{m} = [3, 2, 1, 3, 3, 2, 1, 3, 2]$, where the state $m = 3$ is sent as side information between the transmitter and reciever. This method maintains all of the UEC code's near-capacity capability, although this is achieved at the cost of requiring additional side information to be transmitted.

In a **second method** conceived for accommodating codewords split between two consecutive frames, the trellis encoder may be forced to restart the trellis path m from state 1, when encoding each bit vector $\mathbf{y}$. This does not compromise the near-capacity capability of the UEC code since synchronization is still maintained between the trellis path and the unary codewords despite the trellis encoder potentially starting from the middle of a codeword. More specifically, owing to the particular design of the UEC trellis, the trellis path returns to one of the two central states, whenever the final bit in a unary codeword is encountered, as described in Section II-B. For example, when employing this approach for the successive bit-vectors of $\mathbf{y} = [1, 0, 1, 1, 1, 0, 0, 0]$ and $\mathbf{y} = [1, 1, 0, 0, 1, 1, 0, 1]$ from the previous example, the paths traversed through the UEC trellis are given by $\mathbf{m} = [1, 2, 4, 1, 2, 1, 3, 3, 3]$ and $\mathbf{m} = [1, 2, 1, 3, 3, 2, 1, 3, 2]$, respectively. Observe that these paths are identical to those that result from the first method, with the only exception of the states corresponding to the end of the split codeword, demonstrating that synchronization has been maintained. Note that this approach causes the end state to be unknown to the receiver, since it may correspond to the middle of a unary codeword. This may be accommodated during the Log-BCJR algorithm, by not terminating the UEC trellis at its right-most end, like usual. Furthermore since the first state of each frame is forced to 1, the transition probabilities $P(m_j|m_{j-1})$ employed by the Log-BCJR algorithm for the first codeword may be slightly inaccurate. These two factors impose some error correction performance loss upon the UEC trellis decoder, although this effect is negligible in practice. Owing to this, the results of Section V will adopt this second method, since it does not require any side information to be sent between the transmitter and receiver.

[1]Note that (33) and (34) correct an error in the corresponding formulation of [22, Eq.(22)], which has a redundant summation in its second line.

$$\text{ExpGEC: } A_2^{\text{o}} = \frac{1}{n_2 l_2} \sum_{d=1}^{\min(L, 2^{(x_{\max}+k_{\text{ExpG}})}-2^{k_{\text{ExpG}}})} H\Big[P\big(t(d)|x(d)\big)\Big] P\big(x(d)\big) + \frac{1}{n_2}\left(1 - \sum_{d=1}^{\min(L, 2^{(x_{\max}+k_{\text{ExpG}})}-2^{k_{\text{ExpG}}})} \frac{(x-1+k_{\text{ExpG}}) P\big(t(d) \cap x(d)\big)}{l_2}\right) \tag{33}$$

$$\text{RiceEC: } A_2^{\text{o}} = \frac{1}{n_2 l_2} \sum_{d=1}^{\min(L, Mx_{\max})} H\Big[P\big(t(d)|x(d)\big)\Big] P\big(x(d)\big) + \frac{1}{n_2}\left(1 - \sum_{d=1}^{\min(L, Mx_{\max})} \frac{\log_2(M_{\text{Rice}}) P\big(t(d) \cap x(d)\big)}{l_2}\right) \tag{34}$$

In the receiver of Figure 2, the UEC trellis decoder and URC decoder perform iterative decoding, in order to obtain the *a posteriori* LLR vector $\tilde{\mathbf{y}}$, which pertains to the bit vector $\mathbf{y}$ of the transmitter, as described in Section III-B. Mirroring the buffer employed in the transmitter, the receiver employs a buffer to temporarily store LLRs from the end of the vector $\tilde{\mathbf{y}}$ which correspond to a unary codeword that was split between consecutive frames. More specifically, when the UEC trellis decoder generates the LLR-vector $\tilde{\mathbf{y}}$ for the next frame, it is concatenated on to the end of any LLRs stored in the buffer, forming part of the first codeword in $\tilde{\mathbf{y}}$. The concatenated LLR-vector is then provided to the unary decoder, which generates the vector $\hat{\mathbf{x}}$ comprising a sub-symbols, as described in Section III-B. Following this, any trailing LLRs of $\tilde{\mathbf{y}}$ that did not contribute to a complete unary codeword are placed into the buffer, ready to be concatenated to the beginning of the next LLR-vector $\tilde{\mathbf{y}}$. Note that since the number of sub-symbols a in the vector $\hat{\mathbf{x}}$ varies from frame to frame, it must be conveyed to the receiver using a small amount of side information, as previously discussed in Section III-B. More specifically, this side information conveys the number of logical one-valued bits in $\mathbf{y}$, as exploited by the unary decoder. While the proposed scheme of Figure 2 is capable of functioning without this side information, its inclusion guarantees synchronization between the UEC and FLC-CC parts, as it will be described in Section IV-C. Using the example given above, if the unary decoder is successful in decoding the LLR vector $\tilde{\mathbf{y}}$ of the first frame, it will output the sub-symbol vector $\hat{\mathbf{x}} = [1, 2, 1, 1]$, and will place the last three LLRs of $\tilde{\mathbf{y}}$ into the buffer. Following this, the unary decoder will output the sub-symbol vector $\hat{\mathbf{x}} = [4, 1, 3, 1, 2]$ if it is successful in decoding the LLR-vector $\tilde{\mathbf{y}}$ of the second frame, with no LLRs needed to be placed into the buffer. The decoded sub-symbols of $\hat{\mathbf{x}}$ are buffered until the corresponding sub-symbols of $\hat{\mathbf{t}}$ have also been decoded by the FLC decoder, as will be discussed in Section IV-C.

B. Fixed length interleavers for the FLC-CC sub-code

In a similar fashion to the UEC encoder, the FLC encoder of Figure 2 converts each sub-symbol t_i in the stream $\mathbf{t}$ into the corresponding FLC codeword $\mathbf{u}_i$, as described in Section II-C. These codewords are concatenated to form a stream of bits, which is then partitioned into fixed length bit-vectors $\mathbf{u} = [u_j]_{j=1}^{c}$, which may result in some codewords becoming split between consecutive frames. In order to address this, the scheme of Figure 2 employs a buffer for storing the last bits in the codeword that do not fit into the current frame, so that they can be concatenated onto the start of the next frame. For example, in the case of a RiceEC code associated with $M_{\text{Rice}} = 2$, when FLC encoding the sub-symbol vector $\mathbf{t} = [1, 3, 2, 0, 2]$ to form bit vectors comprising $c = 5$ bits, we obtain $\mathbf{u} = [0, 1, 1, 1, 1]$ and $\mathbf{u} = [0, 0, 0, 1, 0]$ for two consecutive frames. Note that the FLC codeword corresponding to the third sub-symbol of $\mathbf{t}$ is split across the two bit-vectors of $\mathbf{u}$. In the receiver of Figure 2, the CC decoder iteratively exchanges the LLR-vectors $\tilde{\mathbf{w}}^{\text{a}}$ and $\tilde{\mathbf{w}}^{\text{e}}$ with the URC decoder, as well as the LLR-vectors $\tilde{\mathbf{v}}^{\text{a}}$ and $\tilde{\mathbf{v}}^{\text{e}}$ with the FLC decoder, as described in Section III-C. More specifically, $\tilde{\mathbf{v}}^{\text{e}}$ is de-interleaved in the block π_3^{-1} to obtain $\tilde{\mathbf{u}}^{\text{a}}$, while $\tilde{\mathbf{u}}^{\text{e}}$ is interleaved to obtain $\tilde{\mathbf{v}}^{\text{a}}$. However, the FLC decoder can only generate a sub-symbol of $\hat{\mathbf{t}}$ and a codeword of extrinsic LLRs for $\tilde{\mathbf{v}}^{\text{e}}$ when the *a priori* LLR vector $\tilde{\mathbf{v}}^{\text{a}}$ contains all of the *a priori* LLRs for that codeword. Owing to this, a buffer is required for storing *a priori* LLRs for incomplete codewords that have been split between frames, as shown in Figure 2. When the CC decoder forwards an *a priori* LLR vector $\tilde{\mathbf{u}}^{\text{a}}$ to the FLC decoder, it is concatenated onto the end of any LLRs stored in the buffer from the previous frame. In the case of the ExpGEC scheme, the decoded sub-symbols of $\hat{\mathbf{x}}$ are used for determining the length of each of the codewords in $\tilde{\mathbf{u}}^{\text{a}}$, hence allowing the FLC decoder to determine how many excess trailing LLRs of $\tilde{\mathbf{u}}^{\text{a}}$ must be stored in the buffer for each frame. In the case of the RiceEC scheme, since the length of the FLC codewords is fixed at $\log_2(M_{\text{Rice}})$, the FLC decoder does not need any additional information for determining how many excess trailing LLRs to place in the buffer for each frame. The FLC decoder processes the *a priori* LLRs of $\tilde{\mathbf{u}}^{\text{a}}$ comprising complete codewords, in order to generate corresponding extrinsic LLRs for the vector $\tilde{\mathbf{u}}^{\text{e}}$, which is provided to the CC decoder. The extrinsic LLRs corresponding to the *a priori* LLRs that were concatenated from the buffer must be removed before $\tilde{\mathbf{u}}^{\text{e}}$ is provided to the CC decoder, since they are not relevant to the current frame. Likewise, zero-valued LLRs must be inserted at the end of $\tilde{\mathbf{u}}^{\text{e}}$ to pad the vector, in correspondence to the *a priori* LLRs of $\tilde{\mathbf{u}}^{\text{a}}$ that were placed into the buffer, rather than being decoded by the FLC decoder. As a result of this, the FLC decoder and CC decoder may not iteratively exchange LLRs pertaining to every bit in the vector $\mathbf{u}$. This may therefore result in a slight degradation of the FLC decoder's

near-capacity capability, in a similar manner to the effect of limiting the FLC decoding complexity using the parameter $x_{\max}$. However, since only a small number of LLRs in the vector $\tilde{\mathbf{u}}^{\mathrm{e}}$ are impacted in this way, this effect is negligible in practice.

C. Matching sub-symbols

As described in Section II, the ExpGEC and RiceEC codes decompose the symbol stream $\mathbf{d}$ into two sub-symbol streams $\mathbf{x}$ and $\mathbf{t}$. It is important to ensure that the two sub-symbol streams remain synchronized at the receiver, even in the presence of transmission errors. More specifically, if errors cause deleted or inserted sub-symbols to appear in $\hat{\mathbf{x}}$ or $\hat{\mathbf{t}}$, then the incorrect pairings of sub-symbols will be recombined for generating the reconstructed symbol stream $\hat{\mathbf{d}}$, hence resulting in a high SER for the remainder of the stream. This motivates the approach described in this section for avoiding the two sub-symbol streams becoming de-synchronized. We commence by describing how the transmitter multiplexes the UEC part and FLC-CC part onto the channel, then discuss how the receiver maintains synchronization.

As explained in Section III-C, the FLC decoder requires knowledge of the sub-symbol $\hat{x}_i$ in order to generate the corresponding sub-symbol $\hat{t}_i$. Motivated by this, the transmitter ensures that each sub-symbol x_i is encoded and transmitted in advance of its corresponding sub-symbol t_i, so that the reconstructed sub-symbol $\hat{x}_i$ is available for use by the FLC decoder at the appropriate time. Since the number of sub-symbols of $\mathbf{x}$ and $\mathbf{t}$ that are represented by each bit vector of $\mathbf{z}$ and $\mathbf{w}$ may vary from frame to frame, the transmitter has to ensure that a sufficiently high number of sub-symbols of $\mathbf{x}$ have been transmitted before transmitting a bit vector representing a selection of sub-symbols of $\mathbf{t}$. Since the transmitter is capable of maintaining a count of how many sub-symbols of $\mathbf{x}$ it has transmitted at any given time, it can infer how many sub-symbols of $\hat{\mathbf{x}}$ are buffered in the receiver, ready to be used by the FLC decoder. Likewise, the transmitter employs a buffer for storing the sub-symbols of $\mathbf{t}$ until they are ready for transmission, as shown in Figure 2. When the transmitter identifies that the number of sub-symbols of $\hat{\mathbf{x}}$ buffered in the receiver exceeds the number of sub-symbols of $\mathbf{t}$ required to generate the next bit vector $\mathbf{w}$, it is transmitted to the receiver.

The proposed ExpGEC and RiceEC schemes are designed to ensure that synchronization is maintained between the UEC and FLC-CC decoders. As described above, de-synchronization can occur if the unary decoder or FLC decoder output too many or too few sub-symbols for $\hat{\mathbf{x}}$ and $\hat{\mathbf{t}}$, when decoding the LLR vectors $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{u}}^{\mathrm{a}}$, respectively. However, owing to the side information used to indicate to the UEC decoder how many one-valued bits there are in each bit vector $\mathbf{y}$, the unary decoder is guaranteed to output the correct number of sub-symbols for $\hat{\mathbf{x}}$, since each unary codeword contains only a single logical one-valued bit, as described in Section IV-A. For the RiceEC code, the correct number of sub-symbols for $\hat{\mathbf{t}}$ is also guaranteed, since each FLC codeword comprises the same number of $\log_2(M_{\mathrm{Rice}})$ bits. In the case of the ExpGEC code, the properties of the UEC code may be exploited for ensuring synchronization. More specifically, the number of FLC encoded bits of $\mathbf{u}$ associated with each decoded vector $\hat{\mathbf{x}} = [\hat{x}_i]_{i=1}^{a}$ is given by $\sum_{i=1}^{a}(x_i + k_{\mathrm{ExpG}} - 1)$, where a is the length of a specific decoded vector of $\hat{\mathbf{x}}$, which can be correctly inferred from the side information, which conveys the number of logical one-valued bits in each vector $\mathbf{y}$. Since the length of each unary encoded codeword is given by $l_{\mathrm{Unary}(x_i)} = x_i$, the previous sum may be expressed as $\sum_{i=1}^{a}(l_{\mathrm{Unary}(x_i)} + k_{\mathrm{ExpG}} - 1) = a(k_{\mathrm{ExpG}} - 1) + \sum_{i=1}^{a} l_{\mathrm{Unary}(x_i)}$, where $\sum_{i=1}^{a} l_{\mathrm{Unary}(x_i)}$ is the number of LLRs in $\tilde{\mathbf{y}}$ constituting the sub-symbol vector $\hat{\mathbf{x}}$. Since the values of a and the number of LLRs in $\tilde{\mathbf{y}}$ are known to the receiver, the number of LLRs of $\tilde{\mathbf{u}}^{\mathrm{a}}$ associated with a decoded sub-symbol vector of $\hat{\mathbf{x}}$ does not depend on the received value of those sub-symbols. Owing to this, any errors in $\hat{\mathbf{x}}$ cannot cause the sub-symbols $\hat{\mathbf{x}}$ and $\hat{\mathbf{t}}$ to become permanently de-synchronized.

D. Interleavers

In contrast to the JSCC schemes of our previous work [22], the proposed ExpGEC and RiceEC schemes employ interleavers π_1 to π_5 of Figure 2 having fixed lengths, which do not change from frame to frame. Owing to this, these lengths and the corresponding interleaver designs may be hard-coded into the transmitter and receiver. This has the added benefit of avoiding the large memory requirement for storing a large number of interleaver designs, as well as the additional design effort required for ensuring that all the interleavers have desirable distance properties. It also avoids the requirement for using side information to signal which interleaver lengths are used for each frame.

The interleavers π_2 and π_5 of Figure 2 are required for evenly distributing the punctured or doped bits throughout the corresponding bit vector, as opposed to the interleavers π_1, π_3 and π_4 of Figure 2, which are required for maintaining desirable distance properties. As a result, we recommend the LTE sub-block interleaver [34] for the interleavers π_2 and π_5, in order to evenly distribute the doped or interleaved bits. Meanwhile, we recommend S-Random interleavers [35] for the interleavers π_1, π_3 and π_4.

V. Performance comparison with benchmarkers

In this section, we compare the performance of our proposed RiceEC and ExpGEC schemes with that of several appropriate benchmarkers, which are introduced in Section V-A. In Section V-B, we analyze the near-capacity potential of both the proposed codes and of the benchmarkers. Following this, Section V-C discusses our EXIT chart analysis invoked for selecting the doping or puncturing rates R_1^{i} and R_2^{i}, which control the unequal error correction of the proposed schemes. Finally, Section V compares the performance of the proposed schemes and the benchmarkers.

A. Scenarios and benchmarkers

Table III lists the considered parameterizations of the proposed schemes and of the benchmarkers, namely of the Rice-CC, ExpG-CC and VLEC schemes [26]. In analogy with the EG-CC scheme of [22], the Rice-CC and ExpG-CC benchmarkers are both SSCCs, which replace the EG source code of [22] by the Rice and ExpG codes, respectively. Note that the EG-CC scheme of [22] may be viewed as the special case of the ExpG-CC benchmarker, where $k_{\mathrm{ExpG}} = 0$. In these benchmarkers, separate channel coding is provided

by a serial concatenation of an $r = 4$-state non-systematic recursive CC of [3, Table II] with an $r = 2$-state URC and Grey-coded QPSK modulation. Note that this combination was shown to minimize (but not eliminate) the capacity loss of SSCC schemes like the Rice-CC and ExpG-CC benchmarkers [3], [22]. A further benchmarker is provided by the JSCC VLEC scheme of [25], [26] in which a VLEC code is serially concatenated with a $r = 2$ state URC and Gray-coded QPSK modulator. This scheme offers a near capacity operation, but has a rapidly increasing complexity as L increases, as described in Section I.

As shown in Table III, our comparisons consider several scenarios, including the case where the source symbols represent the 26 letters of the English alphabet and the space character, which have a particular probability distribution, as shown in Figure 1b. Here, we map the letters and the space character to the symbol values of 1 to 27 according to descending probability of occurrence. Figure 1b compares this probability distribution to the finite zeta-like distribution of (1) having a parameter value of $L = 27$ and various values of p_1. The entropy of the letters probability distribution is $H_D = 4.1$ bits per symbol, which is equal to that of the finite zeta-like distribution having $p_1 = 0.45$. We also consider the scenario where the source symbols obey the same probability distribution as the transform coefficients and motion vectors produced by a H.265 encoder, as shown in Figure 1b. Note that while the H.265 encoder produces some symbols having values higher than 1000, these have a combined probability of less than 2×10^{-5}. Motivated by this, Figure 1a also shows the finite zeta-like distribution having $L = 1000$ and various values of p_1. The entropy of the H.265 probability distribution is $H_D = 2.4$ bits per symbol, which is equal to that of the finite zeta-like distribution having $p_1 = 0.6$.

As shown in Table III, we also consider scenarios, where the source symbols obey the finite zeta-like probability distribution of (1). Here, we consider the alphabet cardinality of $L = 27$ in order to match that of the English letter source set, as well as the cardinality $L = 1000$, since this is approximately equal to the highest symbol value produced by H.264 [3, Fig. 7] and H.265. This approach allows the performance of the various schemes considered to be compared for both small and large symbol alphabet cardinalities L. Furthermore, we parametrize the finite zeta-like probability distributions using $p_1 \in \{0.2, 0.4, 0.6\}$, which spans the range of p_1 values that best approximate the English alphabet and the H.265 probability distributions, as discussed above. Note that our previous work of [22] focused only on the complementary parameter value range of $p_1 \geq 0.6$.

As shown in Table III, the puncturing and doping of the proposed schemes and benchmarkers are specifically parameterized in each scenario for achieving the same effective throughput η, for the sake of facilitating fair comparisons. More specifically, the effective throughput of $\eta = 0.95$ was selected for the English letters distribution, while the effective throughput of $\eta = 0.85$ was selected for the H.265 distribution. Meanwhile, the target throughputs of $\eta \in \{0.90, 0.83, 0.85\}$ were selected for the scenarios using the finite zeta-like distribution having $p_1 \in \{0.2, 0.4, 0.6\}$, respectively. These target effective throughputs were selected since they are close to the native effective throughputs of all schemes considered, ensuring that none of them were excessively impacted by puncturing or doping.

As described in Section II, the proposed RiceEC and ExpGEC schemes are parameterized by M_{Rice} and k_{ExpG}, respectively. Table III shows the specific values of M_{Rice} and k_{ExpG} that were selected in each scenario considered. In each case, we selected the particular parameter values that give the best SER performance, as will be characterized in Section V-D. We also selected $M_{\mathrm{Rice}} = 1$ and $k_{\mathrm{ExpG}} = 0$, which reduce the RiceEC and ExpGEC schemes to the special cases of the UEC and EGEC schemes of our previous work [3], [22]. The latter arrangements served as additional benchmarkers. The only exception to this however is the omission of the UEC benchmarker in scenarios where excessive puncturing would be required for achieving the effective target throughput η. For example, the UEC benchmarker has very small outer coding rates of $R_1^{\mathrm{o}} = 0.0004$ for the case of a finite zeta-like distribution having $p_1 = 0.2$ and $L = 1000$, as well as $R_1^{\mathrm{o}} = 0.007$ for $p_1 = 0.4$ and $L = 1000$. This may be expected, since the UEC benchmarker has an average codeword length l_{Unary} which approaches infinity, as the source alphabet cardinality L tends to infinity, for the case of $p_1 < 0.608$. Despite this, the UEC benchmarker has only a moderately small outer coding rate of $R_1^{\mathrm{o}} = 0.246$ for the case of the finite zeta-like distribution having $p_1 = 0.6$ and $L = 1000$, as well as of $R_1^{\mathrm{o}} = 0.348$ for the case of the H.265 distribution, as shown in Table III. Since the Rice-CC and ExpG-CC benchmarkers employ the Rice and ExpG source codes respectively, they are also parameterized by M_{Rice} and k_{ExpG}. Table III shows the values of M_{Rice} and k_{ExpG} that were selected for the Rice-CC and ExpG-CC benchmarkers in each of the scenarios considered. In each case, this selection was made by finding the parameterization of M_{Rice} or k_{ExpG} that gives the best SER performance, as will be characterized in Section V-D.

Note that the VLEC benchmarker is only considered for scenarios having source symbol alphabets associated with the cardinality of $L = 27$, since it suffers from an excessive trellis complexity for larger values of L. The VLEC codewords were designed using the approach of [36], which was parameterized using a block distance $d_{\mathrm{b}} = 2$, divergence distance $d_{\mathrm{d}} = 2$ and convergence distance $d_{\mathrm{c}} = 1$. Here, the block distance d_b specifies the minimum Hamming distance required between all pairs of equal length VLEC codewords. Meanwhile, the divergence distance d_d and convergence distance d_c specify the minimum Hamming distances required between all pairs of un-equal length codewords, when they are left- and right-aligned, respectively [37], [25]. After a VLEC codebook was designed in this way, each codeword was then doubled in length by concatenating it with its inverse. For example, the codebook $\{101, 1001, 10001\}$ becomes $\{101010, 10010110, 1000101110\}$ using this approach. This enables a fair comparison by reducing the VLEC coding rate R^{o} to become comparable to those of the other schemes, while also ensuring that the VLEC-encoded bits have equiprobable values, which is a necessary condition for avoiding capacity loss [3].

The final column of Table III quantifies the complexity of each scheme, which was quantified in terms of the number of Add-Compare-Select (ACS) operations performed per bit entered into the QPSK modulator, per decoding iteration.

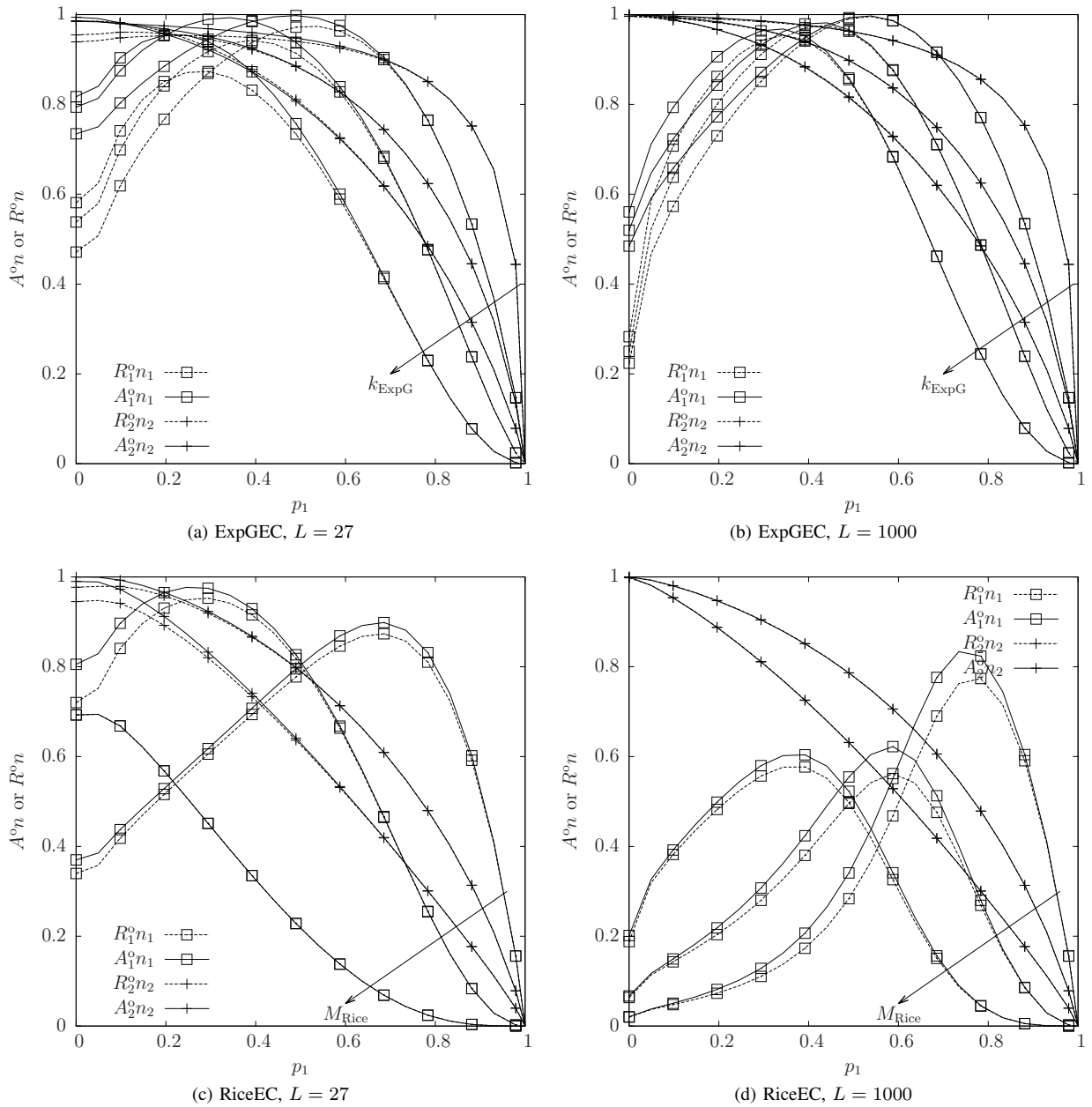
TABLE III: Table of the selected parameters for the proposed schemes and the benchmarkers. All schemes use $r_1 = 4$ and $r_2 = 4$ states. Likewise all schemes use $d_{\max} = 18$, except for the RiceEC schemes having $M_{\mathrm{Rice}} = 32$ and $M_{\mathrm{Rice}} = 64$, where we employ $d_{\max} = M_{\mathrm{Rice}}$.

L	p_1 or distribution	η	Scheme, parameter k_{ExpG}/ M_{Rice}		n_1	n_2	R_1^{o}	R_2^{o}	A_1^{o}	A_2^{o}	R_1^{i}	R_2^{i}	Capacity bound E_{b}/N_0	Area bound E_{b}/N_0	Tunnel bound E_{b}/N_0	Complexity
27	Letters	0.95	ExpGEC	0	2	2	0.374	0.466	0.434	0.490	1.186	1.116		2.29	3.50	304
				3	2	2	0.354	0.473	0.354	0.474	1.277	1.021		1.68	2.63	311
			RiceEC	4	2	2	0.484	0.489	0.494	0.492	0.972	0.987		1.69	2.67	252
				8	2	2	0.432	0.480	0.442	0.479	1.067	1.004	1.60	1.65	2.58	281
			ExpG-CC	1	2		0.443		0.491		1.072			2.07	3.42	270
			Rice-CC	4	2		0.485		0.500		0.979			1.73	2.70	245
			VLEC-URC		2		0.377		0.375		1.289			1.67	3.01	9641
	0.2	0.90	ExpGEC	0	2	2	0.386	0.476	0.444	0.497	1.093	1.028		1.94	3.08	278
				2	2	2	0.427	0.481	0.477	0.481	0.981	0.979		1.55	2.64	264
			RiceEC	1	2		0.260		0.266		1.732			1.49	3.71	435
				8	2	2	0.460	0.463	0.475	0.463	0.965	0.980	1.39	1.46	2.62	265
			ExpG-CC	0	2		0.422		0.475		1.066			1.89	3.52	267
			Rice-CC	4	2		0.472		0.500		0.954			1.63	2.70	240
			VLEC-URC		2		0.360		0.366		1.249			1.44	2.74	9717
	0.4	0.83	ExpGEC	0	2	2	0.471	0.474	0.493	0.488	0.872	0.892		1.26	2.20	228
				2	2	2	0.413	0.437	0.432	0.436	0.980	0.965		1.17	2.33	259
			RiceEC	1	2		0.350		0.358		1.184			1.17	3.01	251
				4	2	2	0.455	0.430	0.462	0.432	0.917	0.960	1.10	1.17	2.23	243
			ExpG-CC	0	3		0.315		0.329		1.318			1.26	3.10	409
			Rice-CC	4	2		0.442		0.469		0.940			1.31	2.67	237
			VLEC-URC		2		0.325		0.335		1.276			1.20	2.43	10693
	0.6	0.85	ExpGEC	0	2	2	0.479	0.461	0.485	0.469	0.886	0.916		1.22	2.18	229
				1	2	2	0.407	0.410	0.411	0.409	1.046	1.038		1.23	2.53	268
			RiceEC	1	2		0.426		0.438		0.997			1.27	2.65	297
				2	2	2	0.424	0.397	0.429	0.397	1.006	1.068	1.19	1.24	2.42	260
			ExpG-CC	0	3		0.316		0.332		1.344			1.36	3.45	417
			Rice-CC	2	2		0.413		0.465		1.028			1.63	3.27	260
			VLEC-URC		2		0.272		0.280		1.565			1.29	3.23	14007
1000	H.265	0.85	ExpGEC	0	2	2	0.464	0.386	0.465	0.375	0.926	1.071		1.25	2.35	248
				1	2	2	0.408	0.481	0.409	0.472	1.008	0.909		1.21	2.26	246
			RiceEC	1	2	2	0.348		0.427		1.222		1.19	2.03	4.66	307
				2	2	2	0.351	0.499	0.397	0.500	1.121	0.956		1.65	3.05	269
			ExpG-CC	1	2		0.445		0.489		0.956			1.56	2.78	241
			Rice-CC	4	2		0.351		0.446		1.210			2.20	4.70	305
	0.2	0.90	ExpGEC	0	2	2	0.368	0.495	0.388	0.498	1.161	0.959		1.63	2.85	273
				3	2	2	0.456	0.470	0.478	0.471	0.968	0.968		1.48	2.42	256
			RiceEC	64	2	2	0.390	0.405	0.395	0.405	1.148	1.114	1.39	1.42	2.65	494
			ExpG-CC	1	2		0.447		0.471		1.007			1.60	2.81	254
			Rice-CC	64	2		0.401		0.475		1.122			2.14	4.07	283
	0.4	0.83	ExpGEC	0	2	3	0.473	0.325	0.477	0.327	0.914	1.200		1.27	2.45	285
				3	2	2	0.425	0.405	0.429	0.405	0.969	1.029		1.12	2.31	277
			RiceEC	32	2	2	0.274	0.327	0.285	0.327	1.441	1.281	1.10	1.15	2.78	478
			ExpG-CC	0	3		0.319		0.323		1.300			1.15	2.95	403
			Rice-CC	16	2		0.334		0.466		1.244			2.63	5.67	313
	0.6	0.85	ExpGEC	0	2	2	0.491	0.470	0.491	0.473	0.872	0.892		1.21	2.09	224
				1	2	2	0.430	0.414	0.430	0.414	0.993	1.024		1.20	2.40	259
			RiceEC	1	2		0.246		0.291		1.712			1.83	5.70	430
				8	2	2	0.224	0.300	0.240	0.300	1.771	1.453	1.19	1.26	3.50	425
			ExpG-CC	0	3		0.282		0.333		1.316			1.29	2.21	408
			Rice-CC	4	2		0.315		0.454		1.347			2.98	6.69	339

This method of comparing complexity is motivated since logarithmic implementations of the algorithms used within each of the iterative decoding blocks can be decomposed into only the basic ACS operations. In particular, each max* operation employed by the Log-BCJR algorithm is assumed to be computed using several lookup table operations, requiring a total of 5 ACS operations [38].

B. Near capacity analysis

There are two requirements for an iterative decoding scheme to facilitate near-capacity operation, where reliable communication is achieved using an effective throughput η that approaches the Discrete-Input Continuous-Output Memoryless Channel (DCMC) capacity C [39]. Firstly, the inner coding rate should obey $A^{\mathrm{i}} = C/[R^{\mathrm{i}} \log_2(M_{\mathrm{mod}})]$, where C is the DCMC capacity. As discussed in [31], this condition is satisfied by the URC, regardless of the puncturing or doping rate R^{i} used for meeting the required effective throughput η. Secondly, the areas beneath the inverted EXIT outer functions A^{o} should approach the corresponding outer coding rates R^{o} [40]. Motivated by this, this section compares the areas A_1^{o} and A_2^{o} beneath the inverted EXIT functions of the UEC and FLC-CC subcodes of the proposed RiceEC and ExpGEC schemes with the corresponding coding rates R_1^{o} and R_2^{o}, in order to characterize their near-capacity operation.



(a) ExpGEC, $L = 27$

(b) ExpGEC, $L = 1000$

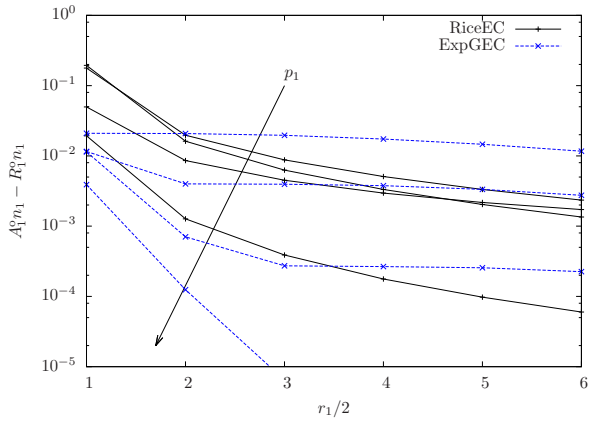
(c) RiceEC, $L = 27$

(d) RiceEC, $L = 1000$

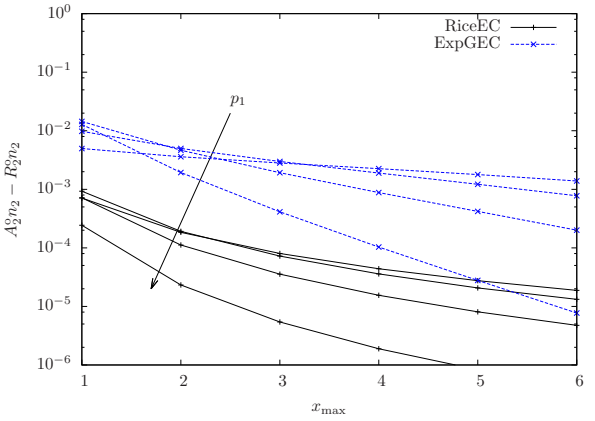
Fig. 6: Product of the coding rate $R^{\rm o}$ and inverted EXIT chart area $A^{\rm o}$ with the codeword length n for the UEC and FLC-CC sub-codes of the proposed ExpGEC and RiceEC schemes, for the case of the finite zeta-like source distribution with various p_1 values and $L \in \{27, 1000\}$. A parameter value of $r_1 = 4$ is used to obtain the inverted UEC EXIT chart area $A_1^{\rm o}$, while $d_{\max} = 18$ is used to obtain the inverted FLC-CC EXIT chart area $A_2^{\rm o}$. (a) and (b) characterize the proposed ExpGEC scheme for different values of the parameter $k_{\rm ExpG} \in \{0, 1, 2\}$, while (c) and (d) characterize the proposed RiceEC scheme for different values of the parameter $M_{\rm Rice} \in \{1, 4, 16\}$. Note that there are no plots of $A_2^{\rm o}$ and $R_2^{\rm o}$ for the RiceEG scheme having $M_{\rm Rice} = 1$, since there is no FLC-CC sub-code in this case.

Figure 6 shows the product of the coding rates $R^{\rm o}$ and inverted EXIT chart areas $A^{\rm o}$ with the corresponding codeword lengths n for the proposed schemes as functions of the finite zeta-like distribution parameter p_1. Here we have plotted the products $A^{\rm o}n$ and $R^{\rm o}n$ to eliminate the effect of different codeword lengths n on the analysis. Furthermore, the values of $R^{\rm o}$ and $A^{\rm o}$ for each of the considered schemes are listed in Table III for each scenario considered. For both the RiceEC and ExpGEC schemes, the coding rate $R_1^{\rm o}$ was obtained using (21), while the FLC-CC coding rate $R_2^{\rm o}$ was obtained using (30). Likewise, (32) is used to obtain the area beneath the inverted UEC EXIT function $A_1^{\rm o}$, for the case of employing $r_1 = 4$ states. Meanwhile, (33) and (34) are used for obtaining the area beneath the inverted FLC-CC EXIT function $A_2^{\rm o}$, where we use $d_{\max} = 18$. These values of r_1 and $d_{\max}$ were found to strike an attractive trade-off between the complexity and near-capacity operation, as discussed in Sections III-B and III-C, respectively.

The discrepancy between $A^{\rm o}n$ and $R^{\rm o}n$ represents capacity loss, as discussed in [3]. Figure 6 shows that the capacity loss $A^{\rm o}n - R^{\rm o}n$ depends on the particular scheme, scenario and parameterization considered. The capacity loss $A_1^{\rm o}n_1 - R_1^{\rm o}n_1$ of the UEC sub-code of the proposed RiceEG and ExpGEC schemes depends on the number of states r_1



(a) The UEC capacity loss, depending on the number of states r_1 employed by trellis decoder.



(b) The FLC-CC capacity loss, depending on the value of $x_{\max}$ employed by the FLC decoder.

Fig. 7: The capacity loss $A^o n - R^o n$ for the UEC and FLC-CC sub-codes with $L = 1000$ and $p_1 \in \{0.2, 0.4, 0.6, 0.8\}$, where $k_{\text{ExpG}} = 1$ in the case of the ExpGEC code and $M_{\text{Rice}} = 8$ in the case of the RiceEC code.

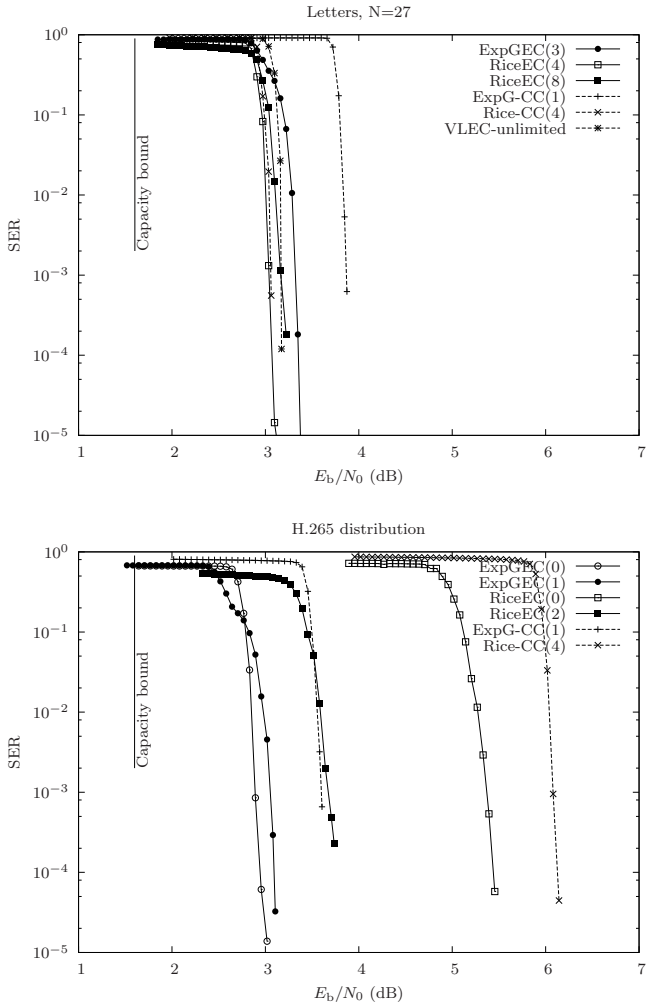


Fig. 9: SER performance of the proposed schemes and benchmarkers listed in Table III when the source symbols obey the probability distribution of letters in the English alphabet, as well as when the symbols obey the probability distribution of a H.265 encoder. Each scheme encodes an average $a = 20000$ symbols per frame and uses QPSK modulation for communication over an uncorrelated narrowband Rayleigh fading channel. For each scheme, the parameter k_{ExpG} or M_{Rice} is listed in the legend within brackets. A complexity limit of 5000 ACS operations per bit input into the QPSK modulator is imposed on each scheme, except in the case of the VLEC benchmarker where the ultimate unlimited complexity performance is shown.

employed by the UEC trellis decoder, as shown in Figure 7a. If more than $r_1 = 4$ states are employed, then the capacity loss shown in Figure 6 will be reduced accordingly, as it may also be seen in (21) and (32), which show that A_1^o approaches R_1^o as r_1 approaches $2L$. Figure 7a also shows that this capacity loss reduces as p_1 is increased, since this results in the less frequent occurrence of higher sub-symbol values in the stream $\mathbf{x}$, which would benefit from using more than $r_1 = 4$ states in the UEC trellis decoder to exploit knowledge of the corresponding occurrence probabilities. As characterized in Figure 7b, the capacity loss $A_2^o n_2 - R_2^o n_2$ of the FLC-CC sub-code is caused by symbols in the stream $\mathbf{d}$ having values that exceed the limit $d_{\max}$, which occur more frequently as p_1 is reduced. This explains why employing a value higher than $d_{\max} = 18$ reduces the capacity loss shown in Figure 7b. This can also be seen in (33) and (34), which show that A_2^o approaches R_2^o as $d_{\max}$ approaches L.

Table III also quantifies the E_b/N_0 values, where the performance of each scheme considered is limited by the *capacity bound*, *area bound* and *tunnel bound*. More specifically, the *capacity bound* is the E_b/N_0 value where the DCMC capacity C becomes equal to the effective throughput η of the scheme, representing theoretical minimum E_b/N_0 at which reliable communication is possible [31]. The *area bound* is the E_b/N_0 value at which $A^i = A^o$, implying that it is theoretically possible to create an open EXIT tunnel, providing that there is a good match between the shapes of the EXIT curves of the schemes' inner and outer decoders [22]. The discrepancy between the capacity bound and the area bound represents the capacity loss of the particular scheme. Table III shows that our best performing ExpGEC and RiceEC schemes have an area bound that is within 0.1 dB of the capacity bound, demonstrating their capability of near-capacity operation. By contrast, the discrepancies between the area and capacity bounds are significantly higher for the SSCC ExpG-CC and Rice-CC benchmakers, owing to the corresponding capacity loss. Finally the *tunnel bound* is the E_b/N_0 value at which an open tunnel is

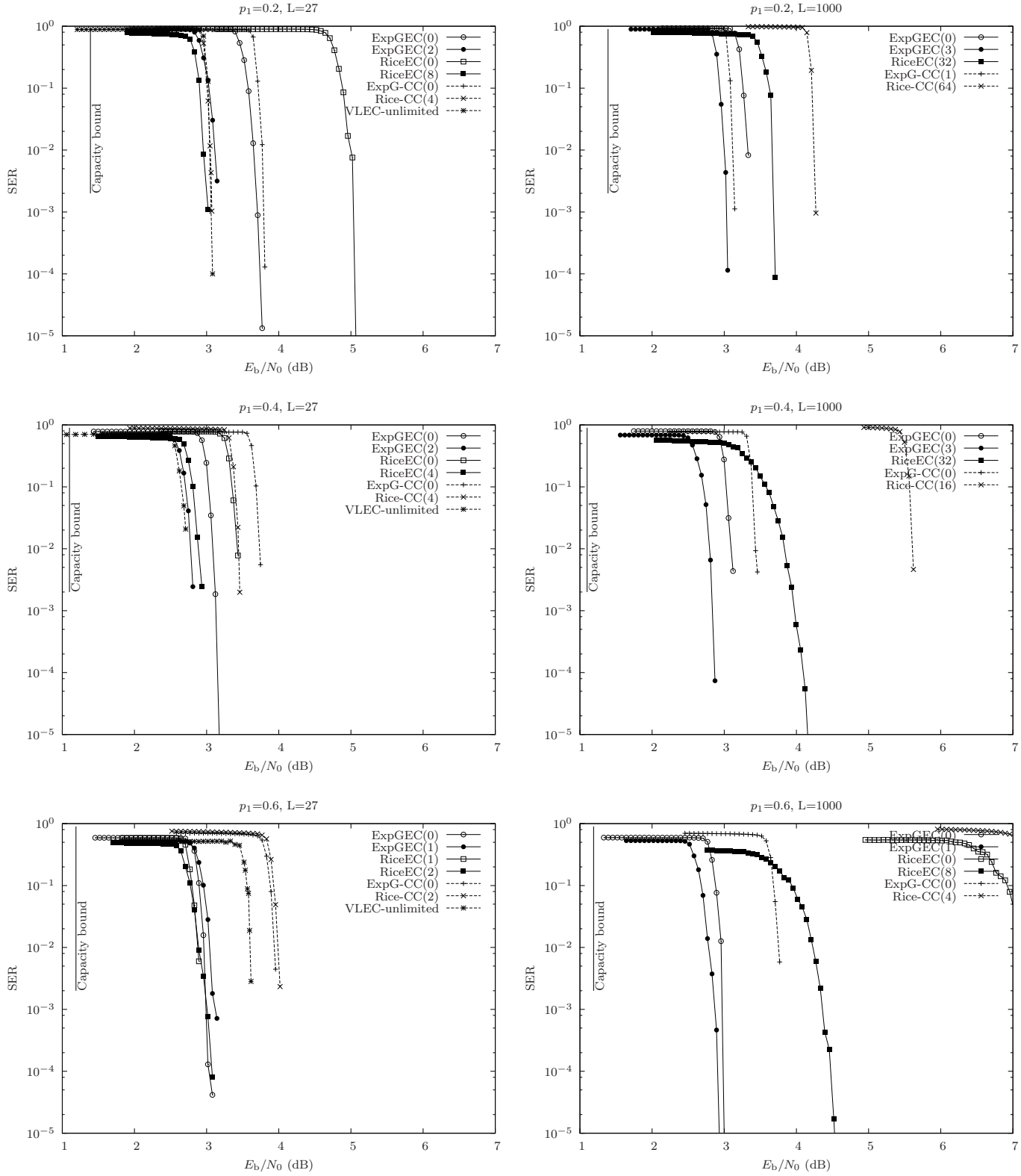


Fig. 8: SER performance of the proposed schemes and benchmarkers listed in Table III. Each scheme encodes an average of $a = 20000$ symbols per frame, which are generated using finite zeta-like probability distributions having different combinations of the parameters $L \in \{27, 1000\}$ and $p_1 \in \{0.2, 0.4, 0.6\}$. Each scheme uses QPSK modulation for communication over an uncorrelated narrowband Rayleigh fading channel. For each scheme, the adopted value of the parameter k_{ExpG} or M_{Rice} is listed in the legend within brackets. A complexity limit of 5000 ACS operations per bit input into the QPSK modulator is imposed on each scheme, except in the case of the VLEC benchmarker where the ultimate unlimited complexity performance is shown.

actually created between the EXIT curves of the scheme's inner and outer decoders [41]. Note that for all proposed schemes and for all benchmarkers, the 2-state URC was found to produce lower tunnel bounds than any 4-state or 8-state URCs. Furthermore, the 2-state URC exhibits the additional benefit of having the lowest complexity of these design options.

C. EXIT chart matching

This section discusses the employment of EXIT charts for designing the parameterization of the proposed ExpGEC and RiceEC schemes, as well as for characterizing their iterative decoding convergence and that of the benchmarkers. This facilitates the rapid characterization of these schemes, considering a wide range of values for the scheme parameters such as k_{ExpG}, M_{Rice}, R_1^{i}, R_2^{i}, n_1 and n_2, as well as for various combinations of the scenario parameters p_1 and L, without requiring time-consuming SER simulations. Separate EXIT charts may be used for characterizing the pair of iterative decoding processes corresponding to the UEC and FLC-CC sub-codes of our proposed RiceEC and ExpGEC schemes. As discussed in [31], codeword lengths of $n_1 \geq 2$ and $n_2 \geq 2$ are required in the proposed schemes in order to facilitate iterative decoding convergence to the (1,1) point in the EXIT chart, associated with a vanishingly low SER [31]. If both sub-codes correspond to equal inner coding rates of R_1^{i} and R_2^{i} and therefore equal amounts of puncturing or doping, then they may be said to adopt equal error protection. However, in this case, the EXIT charts corresponding to the UEC and FLC sub-codes may not form marginally open tunnels at the same channel E_{b}/N_0 value. This results in a range of E_{b}/N_0 values where the EXIT chart of one sub-code has an open tunnel allowing iterative decoding convergence to the (1,1) point and a low SER for the corresponding sub-symbols, while the other sub-code has a closed tunnel leading to a high SER for the corresponding sub-symbols and resulting in a high SER overall. To combat this, unequal error protection [22] may be employed to increase the E_{b}/N_0 value where the first EXIT chart tunnel becomes open, allowing the E_{b}/N_0 value where the second EXIT chart tunnel opens to be reduced to the same value. This enables a low overall SER to be achieved at this lower E_{b}/N_0 value. For example, Figure 5 provides the EXIT charts for the unequal error protection of a particular parameterization of the RiceEC scheme. In this case where $E_{\mathrm{b}}/N_0 = 2.8$ dB, both the UEC and FLC-CC sub-codes are associated with marginally open EXIT chart tunnels, facilitating iterative decoding convergence to the (1,1) point and a low overall SER.

To be specific, unequal error protection is achieved by carefully selecting the inner puncturing or doping rates R_1^{i} and R_2^{i} that are associated with the UEC or FLC-CC sub-codes. Note that when R_1^{i} or R_2^{i} is reduced in order to increase the error protection for the corresponding sub-code, the other one of R_1^{i} or R_2^{i} must be increased, in order to maintain the same overall throughput η, according to (31). As an alternative to excessive doping, the error protection of the FLC-CC sub-code may be increased by increasing the codeword length of the $r_2 = 4$-state CC from $n_2 = 2$ to $n_2 = 3$ bits, according to the design of [3, Table II].

Most of the schemes characterized in Table III have puncturing or doping rates R^{i} that are close to 1, avoiding the performance degradation that is associated with excessive puncturing or doping. However, schemes such as the UEC benchmarker that results for the $M_{\mathrm{Rice}} = 1$ special case of the RiceEC scheme, have puncturing rates that are as high as $R^{\mathrm{i}} = 1.7$, which partly contributes to a high E_{b}/N_0 tunnel bound. More specifically, excessive puncturing results in EXIT charts with narrower open tunnels, resulting in a gradual SER improvement with E_{b}/N_0, rather than a steep turbo cliff. Secondly, excessive puncturing negatively impacts the threshold E_{b}/N_0 value where a marginally open EXIT chart tunnel is created, leading to more capacity loss. Furthermore, a punctured code will also have a decoding complexity disadvantage, since the punctured bits must be decoded alongside the transmitted bits.

D. SER performance

In this section, we compare the SER performance of the proposed RiceEC and ExpGEC schemes to that of the benchmarkers for each of the scenarios listed in Table III. The SER performance of these schemes is characterized in Figures 8 and 9, where a complexity limit of 5000 ACS operations per QPSK input bit is imposed, as it was characterized per decoding iteration in Table III. This complexity limit was chosen in all scenarios considered, since we found that it only marginally impacts the SER performance of whichever scheme converges to its ultimate unlimited performance with the lowest complexity in each scenario. The employment of this criterion for selecting the complexity limit ensures that excessively high-complexity schemes are not favored over those which have a marginally worse SER performance but lower complexity. Due to the considerable complexity of the trellis employed in the VLEC benchmarker, it performs poorly when this complexity limit is imposed, even for the case of $L = 27$. Owing to this, Figures 8 and 9 characterize the SER of the VLEC benchmarker when the complexity limit is removed, in order to characterize its ultimate performance, although this is only achieved at the cost of potentially excessive complexity.

Figures 8 and 9 show that our family of schemes offer the best SER performance in all of the considered scenarios. In the finite zeta-like distribution scenarios having $p_1 = 0.6$, the best of the proposed schemes offers around 1 dB of gain compared to the best SSCC benchmarker for both considered values of L. Likewise, our schemes offer around 0.5 dB of gain for $p_1 = 0.4$, as well as around 0.75 dB of gain for the H.265 distribution. For $p_1 = 0.2$ and for the English letters distribution however, our schemes offer only a marginal SER performance gain over the Rice-CC benchmarker. Note however that this benchmarker suffers from poor performance in other scenarios, which prevents its general applicability. Note that the gains offered by the proposed schemes are achieved 'for free,' since they are achieved without increasing the required decoding complexity, or transmission-energy, -bandwidth, or -duration. The unlimited complexity VLEC benchmarker offers an SER performance very close to our proposed schemes for $p_1 \in \{0.2, 0.4\}$ and $L = 27$, however it should be noted that its complexity is more than an order of magnitude greater than that of our proposed schemes. Furthermore, the complexity of the VLEC benchmarker becomes impractical for values of L significantly greater than 27.

At higher values of p_1, lower values of the parameters of k_{ExpG} and M_{Rice} give higher coding rates R_1^{o} and R_2^{o}, enabling higher effective throughputs η without requiring excessive puncturing. This facilitates better SER performance than may be achieved using higher values of k_{ExpG} and M_{Rice} at these p_1 values. For example, in the scenario where we have $L = 27$ and $p_1 = 0.2$, the $k_{\mathrm{ExpG}} = 2$ parameterization of the ExpGEC scheme outperforms the $k_{\mathrm{ExpG}} = 0$ parameterization by 0.5 dB. By contrast, when $p_1 = 0.6$, the $k_{\mathrm{ExpG}} = 0$ parameterization offers a marginal improvement over the $k_{\mathrm{ExpG}} = 1$ parameterization. This is also the case in the RiceEC, where the $M_{\mathrm{Rice}} = 8$ parameterization offers the best performance at $p_1 = 0.2$ and $L = 27$, while the $M_{\mathrm{Rice}} = 1$ parameterization offers the best performance at $p_1 = 0.6$. In the case where $L = 1000$, the coding rates R_1^{o} and R_2^{o} of the RiceEC scheme are lower than those of the ExpGEC, requiring more puncturing to meet the effective target throughput η. Owing to this, the ExpGEC schemes are superior to the RiceEC scheme in these cases. By contrast, in the case where we have $L = 27$, the coding rates R_1^{o} and R_2^{o} of the RiceEC scheme are similar to those of the ExpGEC, with neither schemes requiring any significant puncturing or doping. This allows the RiceEC to provide similar performance to the ExpGEC when $L = 1000$.

VI. Conclusions

In this paper we have extended and generalized the EGEC code of [22] to give the proposed ExpGEC code. Similarly, we extended the UEC code of [3] to design the proposed RiceEC code. Both these novel codes facilitate operation over a significantly wider range of source symbol distributions. This paper has focused on the scenario where the cardinality L of the source symbol set is finite, allowing comparison to a VLEC benchmarker. We have shown that the proposed schemes achieve the same SER performance as the VLEC benchmarker, but at an order of magnitude lower complexity when we have $L = 27$. Furthermore, our proposed schemes maintain a moderate complexity for significantly higher L values, while that of the VLEC benchmarker may become excessive. Furthermore, we have proposed a technique for increasing the practicality of the proposed schemes, which allow fixed-length designs to be employed for all interleavers. We have shown across a wide range of application scenarios that our family of proposed schemes outperforms several SSCC benchmarkers by as much as 1 dB with no cost in terms of decoding complexity, transmission-energy, -bandwidth or -duration.

References

[1] ITU-T, "Series H: audiovisual and multimedia systems, infrastructure of audiovisual services coding of moving video, high efficiency video coding," 2015. [Online]. Available: www.itu.int/rec/T-REC-H.265-201504-I

[2] N. Johnson, A. Kemp, and S. Kotz, *Univariate discrete distributions*. John Wiley & Sons, 2005.

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

[4] C. E. Shannon, "A mathematical theory of communications," *The Bell System Technical Journal*, vol. 27, pp. 379–423,623–656, 1948.

[5] M. Brejza, L. Li, R. Maunder, B. Al-Hashimi, C. Berrou, and L. Hanzo, "20 years of turbo coding and energy-aware design guidelines for energy-constrained wireless applications," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 8–28, 2016. [Online]. Available: http://eprints.soton.ac.uk/378161/

[6] R. Gallager, "Low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, jan 1962.

[7] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, mar 1979.

[8] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, sep 1978.

[9] D. MacKay, *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

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

[11] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM Journal of Applied Math*, vol. 8, pp. 300–304, 1960.

[12] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. IT-13, pp. 493–497, 1967.

[13] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 194–203, mar 1975.

[14] J. Massey, "Joint source and channel coding," *Communication Systems and Random Process Theory*, 1977.

[15] Q. Stout, "Improved prefix encodings of the natural numbers," *IEEE Trans. Inform. Theory*, 1980.

[16] M. Bernard and B. Sharma, "Some combinatorial results on variable length error correcting codes," *Ars Combinatoria*, pp. 181–194, 1988.

[17] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error correcting coding and decoding: turbo codes," in *Proceedings of the IEEE International Conference on Communications*, vol. 2, Geneva, Switzerland, 1993, pp. 1064–1070.

[18] A. Fraenkel and S. Kleinb, "Robust universal complete codes for transmission and compression," *Discrete Applied Mathematics*, vol. 64, no. 1, pp. 31–55, 1996.

[19] R. Bauer and J. Hagenauer, "Symbol-by-symbol MAP decoding of variable length codes," in *Proc. 3. ITG Conf. on Source and Channel Coding*, 2000, pp. 111–116.

[20] N. Gortz, "Iterative source-channel decoding using soft-in/soft-out decoders," in *2000 IEEE Int. Symp. on Information Theory*, Sorrento, 2000, p. 173.

[21] ——, "On the iterative approximation of optimal joint source-channel decoding," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 9, pp. 1662–1670, 2001.

[22] T. Wang, W. Zhang, R. Maunder, and L. Hanzo, "Near-capacity joint source and channel coding of symbol values from an infinite source set using elias gamma error correction codes," *IEEE Transactions on Communications*, vol. 62, no. 1, pp. 280–292, jan 2014. [Online]. Available: http://eprints.soton.ac.uk/346658/

[23] D. Salomon, *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[24] M. Fresia, F. Perez-Cruz, H. Poor, and S. Verdu, "Joint source and channel coding," *IEEE Signal Processing Magazine*, vol. 27, no. 6, pp. 104–113, nov 2010.

[25] V. Buttigieg and P. Farrell, "Variable-length error-correcting codes," *IEE Proceedings - Communications*, vol. 147, no. 4, p. 211, aug 2000.

[26] M. Bernard and B. Sharma, "A lower bound on average codeword length of variable length error-correcting codes," *IEEE Transactions on Information Theory*, vol. 36, no. 6, pp. 1474–1475, 1990.

[27] R. Maunder and L. Hanzo, "Near-capacity irregular variable length coding and irregular unity rate coding," *IEEE Transactions on Wireless Communications*, vol. 8, no. 11, pp. 5500–5507, 2009. [Online]. Available: http://eprints.soton.ac.uk/264471/

[28] L. Hanzo, R. G. Maunder, J. Wang, and L.-L. Yang, *Near-capacity variable-length coding*. Chichester, UK: John Wiley & Sons, Ltd, oct 2010.

[29] D. Rowitch and L. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes," *IEEE Transactions on Communications*, vol. 48, no. 6, pp. 948–959, jun 2000.

[30] J. Kliewer, A. Huebner, and D. Costello, "On the achievable extrinsic information of inner decoders in serial concatenation," in *2006 IEEE Int. Symp. on Information Theory*, Seattle, jul 2006, pp. 2680–2684.

[31] J. Hagenauer, "The EXIT Chart – introduction to extrinsic information transfer in iterative processing," in *Proc. 12th European Signal Processing Conf.*, Vienna, 2004, pp. 1541–1548.

[32] R. Maunder and L. Hanzo, "Genetic Algorithm Aided Design of Component Codes for Irregular Variable Length Coding," *IEEE Transactions on Communications*, vol. 57, no. 5, pp. 1290–1297, may 2009. [Online]. Available: http://eprints.soton.ac.uk/264470

[33] M. Adrat, R. Vary, and J. Spittka, "Iterative source-channel decoder using extrinsic information from softbit-source decoding," in *2001 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing. Proc.*, 2001, pp. 2653–2656 vol.4.

[34] G. Group, "Multiplexing and channel coding (Release 8). 3GPP TS36. 212 V8. 4.0 Technical specification group radio access network," 2008.

[35] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *JPL Progress report 42-122*, pp. 56–65, 1995.

[36] J. Wang, L.-L. Yang, and L. Hanzo, "Iterative construction of reversible variable-length codes and variable-length error-correcting codes," *IEEE Communications Letters*, vol. 8, no. 11, pp. 671–673, nov 2004.

[37] R. G. Maunder and L. Hanzo, "Genetic algorithm aided design of component codes for irregular variable length coding," *IEEE Transactions on Communications*, vol. 57, no. 5, pp. 1290–1297, may 2009. [Online]. Available: http://eprints.soton.ac.uk/264470/

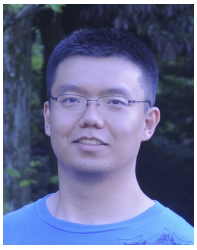
[38] L. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A low-complexity turbo decoder architecture for energy-efficient wireless sensor networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–9, 2011.

[39] J. Proakis, *Digital communications*. McGraw-Hill, 1983.

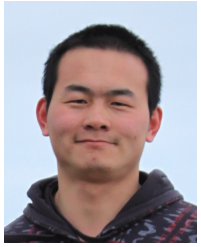
[40] A. Ashikhmin, "Extrinsic information transfer functions: model and erasure channel properties," *Information Theory, IEEE Transactions on*, vol. 50, no. 11, pp. 2657–2673, 2004.

[41] J. Lee and R. Blahut, "Generalized EXIT chart and BER analysis of finite-length turbo codes," *Global Telecommunications Conf. 2003*, pp. 2067–2072 vol.4, 2003.

Matthew F. Brejza recieved a first class honors BEng in Electronic Engeering from the University of Southampton, UK, in July 2012, where he is currently working toward the Ph.D. degree with the Communications Research Group, School of Electronics and Computer Science. His research interests include flexible hardware implementation, source and channel coding and their applications in low power data communications.

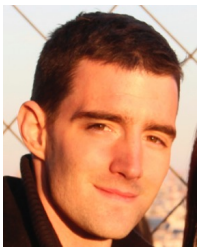


Tao Wang received the B.S. degree in information engineering from the University of Science and Technology of Beijing (USTB), Beijing, China, in 2006. He received M.Sc. degree in communication from University of Southampton, Southampton, U.K in 2008. He is currently working toward the Ph.D. degree with the Communications Research Group, Electronics and Computer Science, University of Southampton, Southampton, UK. His current research interests include joint source/channel coding and distributed video coding.



Wenbo Zhang received the M.E. degree in Information and Communication Engineering from the University of Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2011. He is currently working toward the Ph.D. degree with the Communications Research Group, Electronics and Computer Science, University of Southampton, Southampton, UK. His current research interests include joint source/channel coding and variable length coding.

David Al-Khalili graduated from the University of Southampton in 2014 with a first class honours MEng in Electronic Engineering. He is currently training to become a UK and European patent attorney, with particular areas of work including telecommunications, image and signal processing and medical devices.



Robert G. Maunder has studied with Electronics and Computer Science, University of Southampton, UK, since October 2000. He was awarded a first class honors BEng in Electronic Engineering in July 2003, as well as a PhD in Wireless Communications in December 2007. He became a lecturer in 2007 and an Associated Professor in 2013. Rob's research interests include joint source/channel coding, iterative decoding, irregular coding and modulation techniques. For further information on this research, please refer to http://users.ecs.soton.ac.uk/rm.

Bashir M. Al-Hashimi is an ARM Professor of Computer Engineering and Dean of the Faculty of Physical Sciences and Engineering, University of Southampton. In 2009, he was elected fellow of the IEEE for significant contributions to the design and test of low-power circuits and systems. He holds a Royal Society Wolfson Research Merit Award (2014-2019). He has published over 300 technical papers, authored or co-authored 5 books and has graduated 31 PhD students.

Lajos Hanzo (http://www-mobile.ecs.soton.ac.uk) FREng, FIEEE, FIET, Fellow of EURASIP, DSc received his degree in electronics in 1976 and his doctorate in 1983. In 2009 he was awarded an honorary doctorate by the Technical University of Budapest and in 2015 by the University of Edinburgh. In 2016 he was admitted to the Hungarian Academy of Science. During his 40-year career in telecommunications he has held various research and academic posts in Hungary, Germany and the UK. Since 1986 he has been with the School of Electronics and Computer Science, University of Southampton, UK, where he holds the chair in telecommunications. He has successfully supervised about 100 PhD students, co-authored 20 John Wiley/IEEE Press books on mobile radio communications totalling in excess of 10 000 pages, published 1500+ research entries at IEEE Xplore, acted both as TPC and General Chair of IEEE conferences, presented keynote lectures and has been awarded a number of distinctions. Currently he is directing a 60-strong academic research team, working on a range of research projects in the field of wireless multimedia communications sponsored by industry, the Engineering and Physical Sciences Research Council (EPSRC) UK, the European Research Council's Advanced Fellow Grant and the Royal Society's Wolfson Research Merit Award. He is an enthusiastic supporter of industrial and academic liaison and he offers a range of industrial courses. He is also a Governor of the IEEE VTS. During 2008 - 2012 he was the Editor-in-Chief of the IEEE Press and a Chaired Professor also at Tsinghua University, Beijing. His research is funded by the European Research Council's Senior Research Fellow Grant. For further information on research in progress and associated publications please refer to http://www-mobile.ecs.soton.ac.uk Lajos has 24 000 citations.