On the Complexity of Unary Error Correction Codes for the Near-Capacity Transmission of Symbol Values from an Infinite Set

Wenbo Zhang, Robert G. Maunder, Lajos Hanzo

***Abstract*—Unary Error Correction (UEC) codes have recently been proposed for the near-capacity Joint Source and Channel Coding (JSCC) of symbol values that are selected from a set having an infinite cardinality. In this paper, we characterize the computational complexity of UEC decoders and use complexity analysis for striking a desirable trade-off between the contradictory requirements of low complexity and near-capacity operation. We investigate a wide range of application scenarios and offer a deep insight into their beneficial parameterizations. In particular, we introduce puncturing for controlling the scheme's throughput and for facilitating fair comparisons with a Separate Source and Channel Coding (SSCC) benchmarker. The UEC scheme is found to offer almost 1.3 dB gain, when operating within 1.6 dB of the capacity bound. This is achieved without any increase in transmission energy, bandwidth, transmit duration or decoding complexity.**

***Index Terms*—Source coding, Video coding, Channel coding, Channel capacity, Iterative decoding, EXIT chart, Unary Error Correction Codes**

I. Introduction

MULTIMEDIA codecs such as the H.264 video codec [1] often employ universal source codes such as the Elias Gamma (EG) code [2]. These universal codes are designed for representing symbol values that are selected using a particular probability distribution from a set having an infinite cardinality, such as the set of all positive integers. However, following the application of these source codes, typically some residual redundancy is retained, which imposes capacity loss and therefore prevents near-capacity operation [3]. As a potential remedy, Joint Source and Channel Coding (JSCC) has been proposed for exploiting the residual redundancy and avoiding capacity loss. However, infinite-cardinality symbol value sets are impractical for existing JSCC schemes [4], since they operate on the basis of trellis and graph structures that are infinitely complex, when the cardinality of the symbol value set is infinite.

Against this background, [3] proposed a novel JSCC scheme, which is referred to as the Unary Error Correction (UEC) code. The UEC encoder generates a bit sequence by concatenating unary codewords [5], while the decoder exploits the residual redundancy using a trellis that has only a modest complexity, even when the cardinality of the symbol value set is infinite. An iteratively-decoded serial concatenation of the proposed UEC code and an Irregular Unity Rate Code (IrURC) [6] is capable of asymptotically eliminating the capacity loss, as the complexity of the UEC is increased. In [3], a JSCC UEC code was compared to a Separate Source and Channel Coding (SSCC) benchmarker, which employs an EG source code and a separate Convolutional Code (CC). The JSCC UEC scheme was found to eliminate the 1.11 dB capacity loss incurred by the SSCC EG-CC benchmarker, as well as offer 1.8 dB of gain in a particular practical scenario.

In this paper, we characterize the computational complexity of the UEC scheme and use this to strike a desirable trade-off between the contradictory requirements of low complexity and near-capacity operation. Section II begins by reviewing the operation of the UEC encoder and decoder, as well as that of the EG-CC benchmarker. The computational complexity of the UEC and EG-CC schemes is characterized in Section III. While in [3] we only investigated the relative performance of these schemes for a single source symbol probability distribution, here we consider three different scenarios in Section IV. In order to facilitate this, we introduce the employment of puncturing for controlling the schemes' throughputs and for facilitating fair comparisons in each scenario considered. Furthermore, we offer a deeper insight into the parametrization of the schemes, detailing the design of IrURCs that may be concatenated with the UEC and EG-CC schemes for achieving near-capacity operation. Section V compares the Symbol Error Ratio (SER) performance that can be achieved by the UEC-IrURC and EG-CC-IrURC parametrizations, when operating within a particular practical complexity limit. The UEC scheme is found to offer almost 1.3 dB gain, when operating within 1.6 dB of the capacity bound. This is achieved without any increase in transmission energy, bandwidth, transmit duration or decoding complexity. Finally, we conclude the paper in Section VI.

II. Transmitter and Receiver Operation

The UEC code of [3] was designed for conveying a vector $\mathbf{x} = [x_i]_{i=1}^{a}$ comprising a number of symbols. The value of each symbol $x_i \in \mathbb{N}_1$ is an Independent and Identically Distributed (IID) Random Variable (RV) X_i with probability $Pr(X_i = x) = P(x)$, where $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$ is the infinite-cardinality set comprising all positive integers. In this paper,

The authors are with Electronics and Computer Science, University of Southampton, SO17 1BJ, United Kingdom, e-mail: {wz4g11,rm,lh}@ecs.soton.ac.uk

The financial support of the EPSRC, Swindon UK under the grant EP/J015520/1, that of the RCUK under the India-UK Advanced Technology Centre (IU-ARC) as well as of the EU under the CONCERTO project is gratefully acknowledged. The fiscal support of the European Research Council's Advanced Fellow Grant is also gratefully acknowledged.

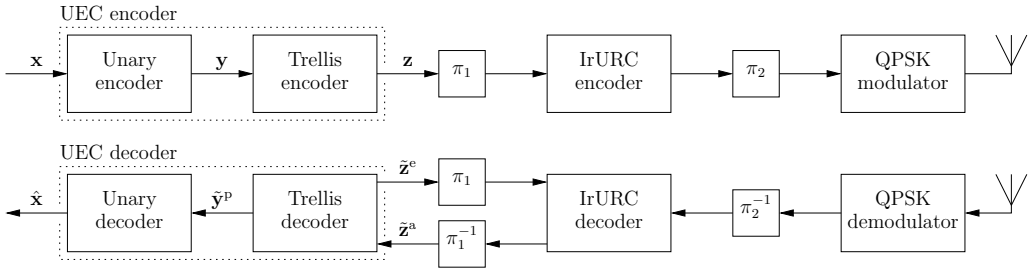

Fig. 1. Schematic of the UEC JSCC scheme, serially concatenated with IrURC coding and QPSK modulation schemes.

we focus our attention on the zeta probability distribution [7], which is defined as

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

where $\zeta(s) = \sum_{x \in \mathbb{N}_1} x^{-s}$ is the Riemann zeta function, $s > 1$, and $p_1 = Pr(X_i = 1) = 1/\zeta(s)$. Table I lists the first ten symbol probabilities $P(x_i)$ for zeta distributions having $p_1 \in \{0.7, 0.8, 0.9\}$.

TABLE I
THE FIRST TEN CODEWORDS OF THE UNARY AND EG CODES.

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

As shown in Figure 1, the UEC scheme encodes the source vector $\mathbf{x}$ using a unary encoder. Each symbol x_i in the vector $\mathbf{x}$ is represented by a corresponding codeword $\mathbf{y_i}$ that comprises x_i bits, as exemplified in Table I. When the symbols adopt the zeta distribution of (1), the average unary codeword length l is only finite for $s > 2$ and hence for $p_1 > 0.608$, in which case we have

$$l = \sum_{x \in \mathbb{N}_1} P(x) \cdot x = \frac{\zeta(s-1)}{\zeta(s)}. \tag{2}$$

The output of the unary encoder is generated by concatenating the selected codewords $[\mathbf{y}_i]_{i=1}^{a}$, in order to form the b-bit vector $\mathbf{y} = [y_j]_{j=1}^{b}$. For example, the source vector $\mathbf{x} = [4, 1, 2, 1, 3, 1, 1, 1, 2, 2]$ of $a = 10$ symbols yields the $b = 18$-bit vector $\mathbf{y} = [111001001100001010]$. Note that the average length of bit vector $\mathbf{y}$ is $a \cdot l$.

Following unary encoding, the trellis encoder of Figure 1 is employed encoding the bit sequence $\mathbf{y}$. Figure 2 exemplifies a UEC trellis having $r = 6$ states, while the generalized r-state UEC trellis was detailed in [3, Figure 3(a)]. Each bit y_j of the input bit sequence $\mathbf{y} = [y_j]_{j=1}^{b}$ forces the trellis encoder to traverse from its previous state $m_{j-1} \in \{1, 2, \ldots, r\}$ to its next state $m_j \in \{1, 2, \ldots, r\}$, in order of increasing bit-index j. Each next state m_j is selected from two legitimate

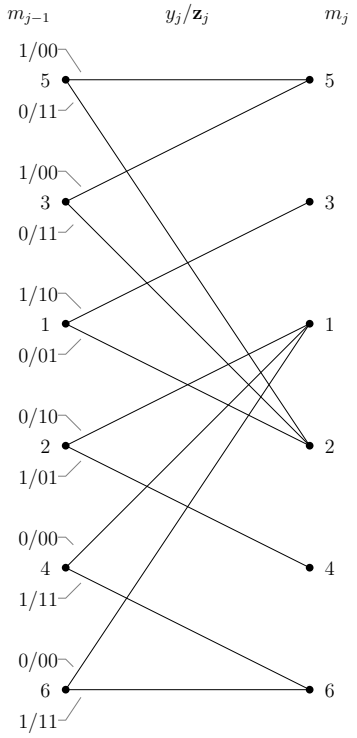

Fig. 2. An $r = 6$-state $n = 2$-bit UEC trellis, where $\mathbb{C} = \{01, 11, 11\}$.

alternatives, depending on the bit value y_j, according to

$$m_j = \begin{cases} 1 + \text{odd}(m_{j-1}) & \text{if } y_j = 0 \\ \min[m_{j-1} + 2, r - \text{odd}(m_{j-1})] & \text{if } y_j = 1 \end{cases}, \tag{3}$$

where the number of possible states r has to be even and the encoding process always begins from the state $m_0 = 1$. The function $\text{odd}(\cdot)$ yields 1 if the operand is odd or 0 if it is even. In this way, the bit vector $\mathbf{y}$ identifies a path through the trellis, which may be represented by a vector $\mathbf{m} = [m]_{j=0}^{b}$ comprising $(b + 1)$ state values. For example, the bit vector $\mathbf{y} = [111001001100001010]$ yields the path $\mathbf{m} = [1, 3, 5, 5, 2, 1, 3, 2, 1, 3, 5, 2, 1, 2, 1, 3, 2, 4, 1]$ through the $r = 6$-state trellis of Figure 2. Note that the path $\mathbf{m}$ may be modeled as a particular realization of a vector $\mathbf{M} = [M_j]_{j=0}^{b}$ comprising $(b+1)$ RVs, which are associated with the transition probabilities $\Pr(M_j = m, M_{j-1} = m') = P(m, m')$ of [3, (8)].

The trellis encoder represents each bit y_j in the vector $\mathbf{y}$ by an n-bit codeword $\mathbf{z_j}$. This is selected from the set of $r/2$ codewords $\mathbb{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{r/2-1}, \mathbf{c}_{r/2}\}$ or from the

complementary set $\overline{\mathbb{C}} = \{\overline{\mathbf{c}_1}, \overline{\mathbf{c}_2}, \ldots, \overline{\mathbf{c}_{r/2-1}}, \overline{\mathbf{c}_{r/2}}\}$, which is achieved according to

$$\mathbf{z}_j = \begin{cases} \overline{\mathbf{c}_{\lceil m_{j-1}/2 \rceil}} & \text{if } y_j = \text{odd}(m_{j-1}) \\ \mathbf{c}_{\lceil m_{j-1}/2 \rceil} & \text{if } y_j \neq \text{odd}(m_{j-1}) \end{cases} . \quad (4)$$

For example, the $n = 2$-bit codewords $\mathbb{C} = \{01, 11, 11\}$ are employed in the $r = 6$-state UEC trellis of Figure 2. Finally, the selected codewords are concatenated to obtain the $b \cdot n$-bit vector $\mathbf{z} = [z_k]_{k=1}^{bn}$ of Figure 1. The average length of the bit vector $\mathbf{z}$ is equal to $al \cdot n$, while the average coding rate R_o of the UEC encoder is given by [3, (12)]. For example, the path $\mathbf{m} = [1, 3, 5, 5, 2, 1, 3, 2, 1, 3, 5, 2, 1, 2, 1, 3, 2, 4, 1]$ through the $r = 6$-state $n = 2$-bit trellis of Figure 2 corresponds to the encoded bit vector $\mathbf{z} = [100000111010111010001110011001110100]$.

As shown in Figure 1, the UEC-encoded bit vector $\mathbf{z}$ may be interleaved in the block π_1 of Figure 1, IrURC encoded [6], [8] and then interleaved as usually punctured in the block π_2. In accordance with convention, the coding rate R_i of this process is given by the number of input bits per output bit. Here, we have $R_\text{i} \geq 1$, since puncturing uses more than one input bit per output bit, while the number of input and output bits remains unaltered for each interleaving and IrURC encoding. Following this, Quaternary Phase Shift Keying (QPSK) modulation may be employed for transmission, as shown in Figure 1. Note that the puncturer may be adjusted for controlling the throughput $\eta = R_\text{o} \cdot R_\text{i} \cdot \log_2(M)$, where we have $M = 4$ for QPSK.

In the receiver, QPSK demodulation, depuncturing π_2^{-1}, IrURC decoding and deinterleaving π_1^{-1} may be performed before invoking the proposed UEC decoder of Figure 1. The trellis decoder applies the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [9], which exploits the synchronization between the UEC trellis and the unary codewords in its γ_t calculation of [9, Equation (9)]. This employs the conditional transition probability $\Pr(M_j = m | M_{j-1} = m')$, where we have

$$P(m|m') = \frac{P(m, m')}{\sum_{\check{m}=1}^{r} P(\check{m}, m')} \quad (5)$$

and $P(m, m')$ is given in [3, (8)].

Note that the UEC trellis should be terminated at $m_0 = 1$ and either at $m_b = 1$ or $m_b = 2$, depending on whether the length a of the symbol vector $\mathbf{x}$ is even or odd, respectively. As shown in Figure 1, the BCJR decoder generates the vector of extrinsic Logarithmic Likelihood Ratios (LLRs) $\tilde{\mathbf{z}}^\text{e} = [\tilde{z}_k^\text{e}]_{k=1}^{bn}$, which may be iteratively exchanged with the serially concatenated IrURC decoder of Figure 1. In the last iteration, the trellis decoder may invoke the BCJR algorithm for generating the vector of *a posteriori* LLRs $\tilde{\mathbf{y}}^\text{p} = [\tilde{y}_j^\text{p}]_{j=1}^{b}$ that pertain to the corresponding bits in the vector $\mathbf{y}$.

The unary decoder of Figure 1 sorts the values in this LLR vector in order to identify the a number of bits in the vector $\mathbf{y}$ that are most likely to have values of zero. A hard decision vector $\hat{\mathbf{y}}$ is then obtained by setting the value of these bits to zero and the value of all other bits to one. Finally, the bit vector $\hat{\mathbf{y}}$ can be unary decoded in order to obtain the symbol vector $\hat{\mathbf{x}}$ of Figure 1, which is guaranteed to comprise a number of symbols.

The JSCC UEC-IrURC scheme of Figure 1 may be compared to the SSCC EG-CC-IrURC benchmarker of [3, Figure 2(b)]. In the EG-CC-IrURC transmitter, the unary encoder is replaced by an EG encoder, while the trellis encoder is replaced by a CC encoder. The first ten EG codewords are illustrated in Table I, where the overall average codeword length is given by $l = \sum_{x \in \mathbb{N}_1} P(x) \left(2\lfloor \log_2(x) \rfloor + 1\right)$. Meanwhile, the CC encoder may be described by the generator and feedback polynomials provided in [3, Table II]. In the EG-CC-IrURC receiver, the CC decoder decodes the *a priori* LLRs $\tilde{\mathbf{z}}^\text{a}$ and $\tilde{\mathbf{y}}^\text{a}$ by utilizing the BCJR algorithm [9]. It outputs extrinsic information $\tilde{\mathbf{z}}^\text{e}$ in each iteration, which can be exchanged with that provided by the IrURC decoder. During the last iteration, the CC decoder uses the Viterbi algorithm [10] for generating the bit sequence $\hat{\mathbf{y}}$, which may then be decoded by the EG decoder.

III. Decoding Complexity Analysis

In this section, we characterize the computational complexity of the various decoders that are employed in the UEC-IrURC and EG-CC-IrURC schemes. This is achieved by observing that each of the decoders listed in Table II operates on the basis of only using the $\max^*$ [11] and addition operations, where

$$\max^*(\tilde{z}_1, \tilde{z}_2) = \max(\tilde{z}_1, \tilde{z}_2) + \ln(1 + e^{-|\tilde{z}_1 - \tilde{z}_2|}). \quad (6)$$

The complexity of each type of decoder scales linearly with the number of bits $b \cdot n$ in the encoded vector $\mathbf{z}$ of Figure 1. Therefore, Table II lists the number of $\max^*$ and addition operations that are performed per bit of $\mathbf{z}$, when each type of decoder employs a trellis having r states. Note that the computational complexity of the trellis and CC decoder depends on whether it is used for generating an output pertaining to the bit vector $\mathbf{y}$ or to $\mathbf{z}$.

TABLE II
Number of $\max^*$ and addition operations that are performed per bit of $\mathbf{z}$, for various types of decoder employing trellises having r states.

Decoder	r	$\max^*$	add	ACS
$n = 2$-bit CC Viterbi decoder $\hat{\mathbf{y}}$	4	2	8	18
$n = 2$-bit CC BCJR decoder $\tilde{\mathbf{z}}^\text{e}$	4	10	22	72
$n = 2$-bit Trellis BCJR decoder $\tilde{\mathbf{y}}^\text{p}$	4	7	20.5	55.5
	6	11	30.5	85.5
	8	15	40.5	115.5
	10	19	50.5	145.5
$n = 2$-bit Trellis BCJR decoder $\tilde{\mathbf{z}}^\text{e}$	4	10	22	72
	6	16	32	112
	8	22	42	152
	10	28	52	192
URC BCJR decoder	2	6	19	49
	4	14	37	107
	8	30	73	223

In practice, the term $f_\text{c} = \ln(1 + e^{-|\tilde{z}_1 - \tilde{z}_2|})$ in the $\max^*$ operation can be implemented at a low computational complexity by employing a Look-Up-Table (LUT) [11]. When employing an 8-entry LUT, a total of $\log_2(8) = 3$ compare operations are required for selecting the particular LUT entry that best approximates f_c. Furthermore, a compare operation is required for computing $\max(\tilde{z}_1, \tilde{z}_2)$ and an addition operation

is required evaluating $\max(\tilde{z}_1, \tilde{z}_2) + f_c$. Therefore, each $\max^*$ operation can be considered to be equivalent to five Add, Compare and Select (ACS) arithmetic operations. By contrast, each addition operation corresponds to a single ACS operation. Using this logic, Table II lists the total number of ACS operations that are performed by each type of decoder. Note that since an IrURC is formed as a combination of Unity Rate Code (URC) components having different numbers of states r, the computational complexity of an IrURC decoder can be quantified as a weighted average of the URC complexities given in Table II. The weights that should be employed in this weighted average are given by the fraction of bits α that are encoded by URC component codes having each number of states r, as it will be exemplified in Section IV.

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

$$N_{\text{ACS}} = \frac{bn}{a}(49t + 72(t-1) + 55.5) = 2l(121t - 16.5). \quad (7)$$

IV. Parametrization of the UEC-IrURC and EG-CC-IrURC schemes

In this section, we detail a number of parametrizations conceived for the UEC-IrURC and EG-CC-IrURC schemes. As shown in Table III, we consider source symbols that obey zeta distributions having $p_1 \in \{0.7, 0.8, 0.9\}$. In all cases, we opted for employing UEC and CC codes having $n = 2$-bit codewords, since this is the minimum number required for facilitating iterative decoding convergence to the Maximum Likelihood (ML) error correction performance [3]. Note that for each of the p_1 values considered, the UEC and EG-CC schemes have different outer coding rates R_o, as may be calculated using [3, (12)]. For the sake of fair comparisons, the scheme having the lower outer coding rate R_o may be compensated using an inner coding rate of $R_i > 1$, which is achieved by applying puncturing within the block π_2 of Figure 1. As shown in Table III, this approach can be employed for achieving the same throughput of $\eta = R_o \cdot R_i \cdot \log_2(M)$ for both the UEC-IrURC and the EG-CC-IrURC scheme[1], for each value of $p_1 \in \{0.7, 0.8, 0.9\}$. Table III provides the E_b/N_0 values at which the Discrete-input Continuous-output Memoryless Channel (DCMC) capacity C of QPSK modulation in a uncorrelated narrowband Rayleigh fading channel becomes equal to the throughput η of the various schemes considered. These *capacity bounds* represent the lowest E_b/N_0 values, where it is theoretically possible to achieve an infinitesimally low error probability.

As shown in Table III, we consider UEC codes having $r \in \{4, 6, 8, 10, 32\}$ states. In each case, the first codeword in the set $\mathbb{C}$ has the value 01 and the remaining $(r/2-1)$ codewords have the value 11, as advocated in [3]. By contrast, the EG-CC scheme employs the $n = 2$-bit $r = 4$-state CC of [3, Table II], since this was found to offer superior error correction performance over that of CCs having a higher number of states r [3]. Figure 3 provides the inverted EXtrinsic Information Transfer (EXIT) curves [12] of the r-state UEC and EG-CC BCJR decoders, for the scenario where the symbol values obey zeta probability distributions having $p_1 \in \{0.7, 0.8, 0.9\}$. For each case, Table III provides the area A_o beneath the UEC and EG-CC EXIT curves, as may be calculated according to [3]. It may be observed that the discrepancy between A_o and R_o diminishes, as the number of states r employed in the UEC scheme is increased. This may be expected since [3] showed that the discrepancy tends to zero, as r approaches infinity. By contrast, a significant discrepancy can be observed between A_o and R_o for the EG-CC scheme, for each value of p_1. Note that the discrepancy is independent of r in the case of the EG-CC scheme, as discussed in UEC.

As shown in Figure 1, the R_o-rate UEC and EG-CC schemes form the outer code in a serial concatenation with an R_i-rate inner code, which is comprised of an IrURC code and a puncturer π_2. In order to achieve iterative decoding convergence towards a vanishingly low error probability, it is necessary for the area beneath the inner EXIT curve A_i to exceed A_o, so that an open EXIT chart tunnel can be created. It can be shown that we have $A_i = C/[R_i \cdot \log_2(M)]$ for this inner code, regardless of how the IrURC code is parametrized. Table III provides the E_b/N_0 values at which A_i becomes equal to A_o for the various schemes considered. These *area bounds* represent the lowest E_b/N_0 values, where it would be possible to create an open EXIT chart tunnel, if the IrURC code was particularly well parametrized. Note that the discrepancies between the area bound and the capacity bound represent the *capacity loss* of each scheme considered. As shown in Table III, the EG-CC schemes suffer from a significant capacity loss, which can be eliminated by employing a UEC scheme having a sufficiently high number of states r.

An IrURC parametrization was designed to be serially concatenated with the UEC and EG-CC schemes for each value of $p_1 \in \{0.7, 0.8, 0.9\}$. These IrURCs were constrained to the choice of the ten URC component codes of [6, Figure 4], since their decoders employ no more than $r = 8$ states and therefore do not have an excessive computational complexity compared to the UEC and EG-CC decoders, as shown in Table II. As listed in Table IV, each URC component code is employed for encoding a particular fraction α of the bits provided by the interleaver π_1 of Figure 1. These fractions α were chosen using the algorithm of [13] for the sake of generating EXIT curves that match those of the $r = 32$ UEC scheme and of the $r = 4$ EG-CC scheme. As shown in Figure 3, this facilitated the creation of open EXIT chart tunnels at the lowest possible E_b/N_0 values, as listed in Table III. These E_b/N_0 values maybe deemed to represent *the open tunnel bounds*, which must be exceeded for the sake of creating an open tunnel in practice, hence facilitating iterative decoding convergence to a low error probability, when the number of symbols a in the source sequence $\mathbf{x}$ is infinite [14]. Note that the discrepancies between the open tunnel bounds and area bounds may be attributed to the constraining of the IrURC parametrization to using only the low-complexity URC component codes of

[1]Note that the approach described here is in contrast to that of [3], which only considered source symbols that obey zeta distributions having $p_1 = 0.797$. This value was chosen since it results in identical outer coding rates R_o for the UEC and EG-CC schemes, avoiding the requirement for puncturing in order to obtain fair comparisons.

TABLE III

OUTER CODING RATE $R_{\rm o}$, INNER CODING RATE $R_{\rm i}$ AND TOTAL THROUGHPUT η FOR TWO SCHEMES WITH DIFFERENT STATES. THREE CATEGORIES OF $E_{\rm b}/N_0$ WHERE $C = \eta$ AND $A_{\rm i} = A_{\rm o}$ IN THEORY, AND WHERE TUNNEL IS OPEN IN SIMULATION, RESPECTIVELY.

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

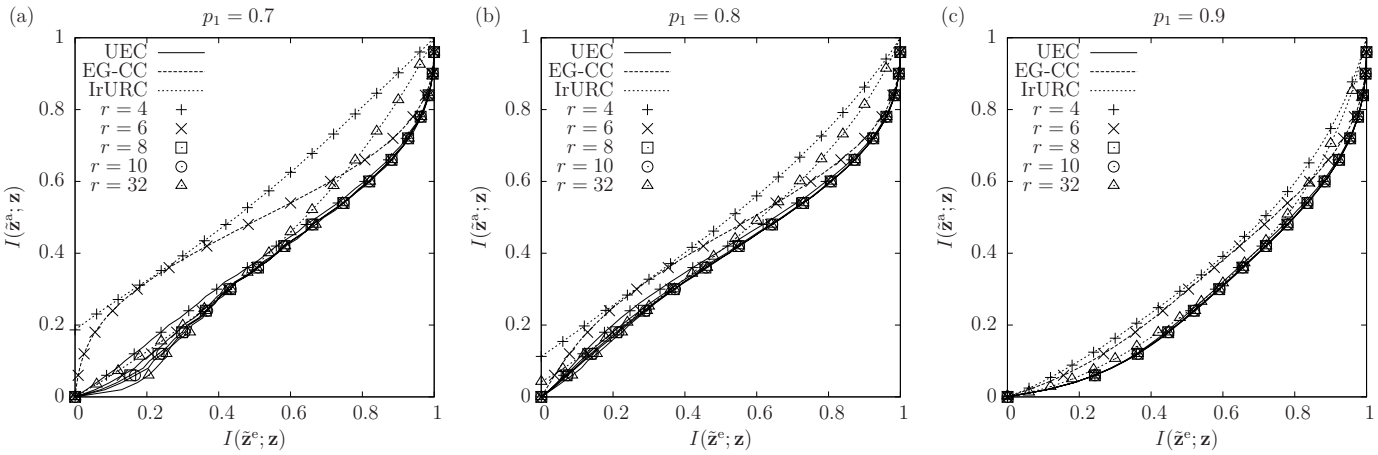

Fig. 3. Inverted EXIT curves for the UEC BCJR decoder having $r \in \{4, 6, 8, 10, 32\}$ states and EG-CC BCJR decoder having $r = 4$ states, where $p_1 \in \{0.7, 0.8, 0.9\}$. Corresponding EXIT curves are provided for the IrURC schemes of Table IV, at the lowest $E_{\rm b}/N_0$ values that facilitates the creation of an open tunnel with the EXIT curves of the $r = 32$-state UEC and the $r = 4$-state EG-CC, as listed in Table III.

[6, Figure 4]. As shown in Table III, the UEC schemes may be seen to offer significant gains over the EG-CC schemes, when $p_1 \in \{0.8, 0.9\}$. By contrast, for $p_1 = 0.7$ the UEC and EG-CC schemes may be seen to have similar open tunnel bounds. Table IV quantifies the number of ACS operations that are performed per bit of $\mathbf{z}$. As described in Section III, these values are obtained by employing the fractions α as weights, when averaging the number of ACS operations that are listed in Table II for URCs having $r \in \{2, 4, 8\}$ states.

V. SER PERFORMANCE

In this section, we characterize the SER performance of the UEC-IrURC and EG-CC-IrURC schemes of Section IV. We consider the transmission of source symbol sequence $\mathbf{x}$ comprising $a = 10^4$ symbols, since this frame length is typical of multimedia applications [3]. In order to facilitate fair comparison between schemes having different computational complexities per iteration, we impose a limit of 10^4 ACS operations for the decoding of each symbol in $\mathbf{x}$. This facilitates an average of $t = 6$ decoding iterations in the $r = 10$ UEC scheme for $p_1 = 0.7$ and $t = 11$ iterations in the $r = 4$ scheme for $p_1 = 0.9$, which are the most and least complex schemes considered, respectively.

As shown in Figure 4, our UEC-IrURC scheme outperforms the EG-CC-IrURC benchmarker for transmission over uncorrelated Rayleigh channel, when $p_1 \in \{0.8, 0.9\}$. More particularly, when $p_1 = 0.9$, a 1.3 dB gain is offered by the UEC-IrURC scheme, while operating within 1.6 dB of the capacity bound. By contrast, the EG-CC-IrURC benchmarker outperforms the UEC-IrURC scheme for $p_1 = 0.7$. This may be attributed to the high number of bits that are punctured in the UEC-IrURC scheme for $p_1 = 0.7$. Owing to this, a relatively high number of decoding iterations are required for achieving iterative decoding convergence in this scheme, even when the $E_{\rm b}/N_0$ value is high. As a result, a severe performance degradation is incurred, when imposing a limit of 10^4 ACS operations for decoding each symbol in $\mathbf{x}$. Note that a UEC decoder employing $r = 6$ states offers the optimal trade-

TABLE IV
TEN COMPONENT URC CODES ARE LABELED USING THE FORMAT (g, f), WHERE g AND f ARE THE HEXADECIMAL GENERATOR AND FEEDBACK POLYNOMIALS OF THE URC CODE, RESPECTIVELY.

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

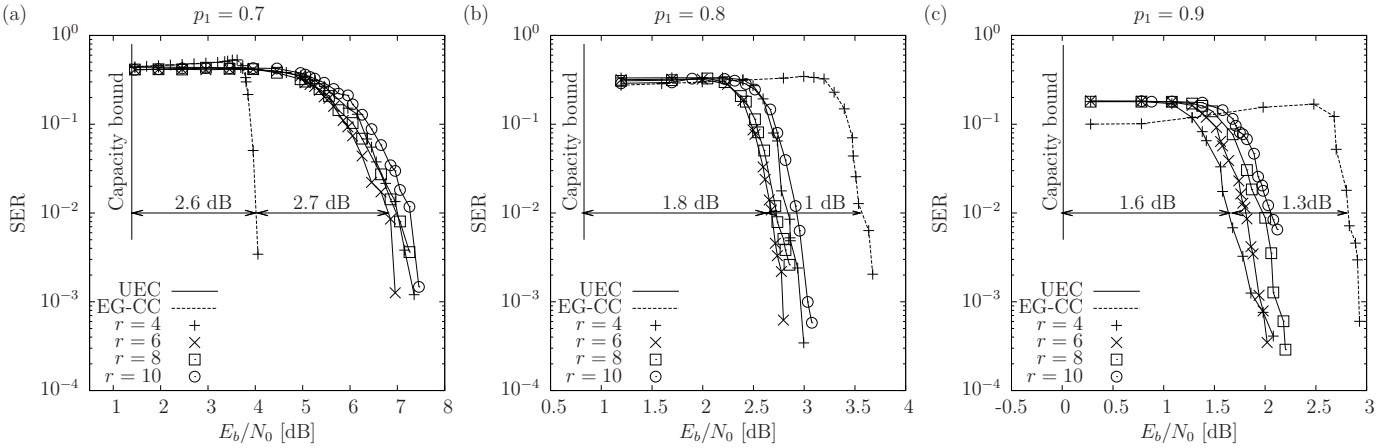

Fig. 4. The SER performance of the UEC-IrURC scheme having $r \in \{4, 6, 8, 10\}$ states and the EG-CC-IrURC scheme having $r = 4$ states of Table III, where $p_1 \in \{0.7, 0.8, 0.9\}$. Communication is over an uncorrelated narrowband Rayleigh fading channel and a complexity limit of 10^4 ACS operations is imposed for decoding each of the $a = 10^4$ symbols in $\mathbf{x}$.

off between complexity and near-capacity operation when $p_1 \in \{0.7, 0.8\}$, while $r = 4$ is the optimal choice for $p_1 = 0.9$, as shown in Figure 4.

VI. CONCLUSIONS

In this paper, we have characterized the computational complexity of UEC decoders and used this for striking a desirable trade-off between the contradictory requirements of low complexity and near-capacity operation. We have detailed UEC parametrizations that offer as much as 1.3 dB gain over the corresponding EG-CC benchmarker, when operating within 1.6 dB of the capacity bound. This was achieved without any increase in transmission energy, bandwidth, transmit duration or decoding complexity. In our future work, we will develop an *Elias Gamma Error Correction (EGEC)* code, which can achieve the same throughput η as the EG-CC scheme without puncturing when $p_1 = 0.7$, while avoiding any capacity loss.

REFERENCES

[1] *Advanced video coding for generic audiovisual services*. ITU-T Std. H.264, March 2005.

[2] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inform. Theory*, vol. 21, no. 2, pp. 194–203, March 1975.

[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 (Submitted)*, 2012 [Online]. Available: http://eprints.soton.ac.uk/341736/.

[4] J. L. Massey, "Joint source and channel coding," in *in Communication Systems and Random Process Theory*, The Netherlands: sijthofoo and Noordhoff, Dec. 1978, pp. 279–293.

[5] R. G. Gallarger and D. C. Van Voorhis, "Optimal source codes for gemotrically distributed integer alphabets," *IEEE Trans. Inform. Theory*, vol. 21, no. 2, pp. 228–230, Mar 1975.

[6] L. Hanzo, R. G. Maunder, J. Wang, and L.-L. Yang, *Near-Capacity Variable Length Coding*. Chichester, UK: Wiley, 2010. [Online]. Available: http://eprints.ecs.soton.ac.uk/20911/.

[7] Johnson, N. L., Kemp, A. W. and Kotz, S., *Univariate Discrete Distributions*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2005.

[8] R. G. Maunder and L. Hanzo, "Near-capacity irregular variable length coding and irregular unity rate coding," *IEEE Trans. Wireless Commun.*, vol. 8, no. 11, pp. 5500–5507, Nov. 2009.

[9] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate(Corresp.)," *IEEE Trans. Inform. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[10] L. Hanzo, T. H. Liew, B. L. Yeap, R. Y. S. Tee, S. X. Ng, *Turbo Coding, Turbo Equalisation and Space-Time Coding*. John Wiley & Sons, 2011. [Online]. Available: http://eprints.soton.ac.uk/258252/.

[11] Li, L. and Maunder, R. G. and Al-Hashimi, B. M. and Hanzo, L., "A Low-Complexity Turbo Decoder Architecture for Energy-Efficient Wireless Sensor Networks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–9, 2011.

[12] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1727–1737, Oct. 2001.

[13] M. Tuchler and J. Hagenauer, "EXIT charts of irregular codes," *Conference on Information Sciences and Systems*, vol. , no. , p. , Mar. 20-22 2002.

[14] J. W. Lee and R. E. Blahut, "Generalized EXIT chart and BER analysis of finite-length turbo codes," in *in Proc. IEEE Global Telecommunications Conf.*, San Fransisco, CA, USA, Dec. 2003, pp. 2067–2072.