Abstract—We propose weakly-constrained stream and block codes with tunable pattern-dependent statistics and demonstrate that the block code capacity at large block sizes is close to the prediction obtained from a simple Markov model published earlier. We demonstrate the feasibility of the code by presenting original encoding and decoding algorithms with a complexity log-linear in the block size and with modest table memory requirements. We also show that when such codes are used for mitigation of patterning effects in optical fibre communications, a gain of about 0.5dB is possible under realistic conditions, at the expense of small redundancy (∼10%).

Index Terms—Patterning effects, weakly-constrained codes, coding theory, pre-encoding.

I. INTRODUCTION
THE patterning effect due to inter-symbol interference (ISI) manifests itself in digital communication as dependence of the transmission result for an information bit on the surrounding pattern, i.e. the neighbouring bits. ISI imposes one of the most severe restrictions on the speed of data transmission and can be due to physical mechanisms of widely varying origins. For instance, in fibre optic digital communication, pattern dependence of errors can be caused by the gain saturation of a semiconductor optical amplifier (see, e.g., [1]) or by resonance interactions between pulses in bit-overlapping transmission regimes [2], [3]. Note that the current trend of ever-increasing channel rate makes it necessary to use increasingly shorter carrier pulses thus also increasing the effects of dispersive broadening of the signal and consequently bit-overlapping. Therefore the negative impact of the patterning effects may become a very serious factor in future optical systems.

One well-researched example of ISI is optical fibre communication at high bit rates limited by intra-channel four-wave-mixing (ICFWM) [2], [3], which generates “ghost” pulses. Under such conditions the main contribution to the bit error rate comes from the ghosts pulses that appear in logical-zero time slots surrounded by symmetric patterns of logical-ones [4]. Mitigation of intra-channel nonlinear effects by channel coding has been proposed in [5], [6]. Suppression of the ICFWM effects by employing a pre-encoding was studied in [4], [7]-[9]. Direct modeling of bit-error-rate in high-speed wave-division-multiplexing optical communications, see [10], [11], has shown the utility of skewed statistics pre-encoding of the type discussed in [7]. In the present paper we will extend the analysis begun in [7] with consideration and evaluation of specific coding schemes.

It should be noted that practical optical transmission lines have to rely on forward-error correction techniques for the provision of a low-BER communication channel. When the BER is small by itself, the FEC can cope with the errors due to both the channel noise and patterning well enough not to require any additional measures. However, as the patterning effects grow stronger, at some point the FEC scheme starts to deteriorate rapidly and it is at that point that error prevention becomes essential in maintaining the low-BER regime. The solution that we proposed in [7], and which we continue to study here, is to use a weakly constrained code to adjust the incidence of undesirable patterns down in order to reduce the part of the BER that is due to patterning and, consequently, bring the combined BER back under the FEC break-down threshold. Since the FEC threshold is usually very sharply defined, the most economical pre-encoding scheme ought to be tunable: any extra redundancy below the threshold is more effective if utilised by the FEC itself. Weakly constrained codes are codes that reduce the incidence of patterns of a certain kind; the amount of reduction is directly linked with the code redundancy and is controllable by a parameter that can be varied almost continuously [7]. This makes them ideally suited to the task of controllable pattern reduction for the purposes of FEC.

We will present specific block and stream codes for weakly-constrained coding and will demonstrate that for some of these codes, which can be realized in practice, the redundancy at a given level of pattern elimination is close to the Markov model obtained in [7]. Finally, for our block codes we propose a log-linear complexity encoding/decoding algorithm and evaluate the code gain under various strengths of the patterning effect.

II. FROM “GHOST” PULSES TO PATTERN REDUCTION
Denote as \(b_1 b_2 \ldots b_n\) the transmitted sequence of bits. If \(b_k = b_l = b_m = 1\) and the bit \(b_{k+l-m} = 0\), then the “ghost” pulse effect establishes a false 1 in the position \(k + l - m\) upon reception (see [4], [12] for more details). The patterns 1101, 1011, and 11011 are especially prone to “ghost” pulses. For example, if \(b_7 b_8 b_{p+1} b_{p+2} b_{p+3} = 11011\), then \(k = p, l = p + 3, m = p + 1\) and the bit \(b_{k+l-m} = b_{p+2} = 0\) can be received as 1. Since the capacity of a “ghost”-constrained channel is zero (see [12] for the proof), the complete suppression of the effect is impossible. However, for any finite sequence complete
elimination of “ghost” pulses can be achieved by constrained coding (see [12]).

It is noteworthy that some “resonance” patterns are more hazardous than others. It is proven that the 101 pattern, and, of course, patterns containing it, such as 11011, 1101 etc., make the main contribution to the channel bit error rate. This was observed experimentally and described in [4], [10] and [13]. The papers [10] and [13] describe a channel simulation and report some error statistics obtained in it.

In contrast to the complete elimination of “ghost” pulses, which is extremely expensive in terms of the channel capacity, a mere reduction of 101 can be achieved considerably more cheaply. Indeed, by using our proposed codes 50% of the undesirable pattern occurrences can be removed at 6% redundancy. Such a major reduction in strong-patterning situations leads to a similar reduction in the BER, which in turn can return the FEC procedure to the normal, subcritical, regime.

Could the patterns 101 be reduced or eliminated using standard coding techniques? For example, the codes RLL\((d, k)\) with \(d \geq 2\) are suitable for such elimination as they ensure that at least two 0s are present between any pair of 1s. Indeed, the codes RLL\((1, \infty)\) and RLL\((2, \infty)\) have been proposed for the worst pattern suppression (see [14]). Unfortunately, RLL-codes have very small capacities \((C(2, \infty) = 0.5515, C(1, \infty) = 0.6942)\) and are therefore rather expensive for long-haul optical transmission systems. Other codes were suggested as a cheaper alternative, following exactly the same logic as ours (small patterns associated with resonances instead of all possible resonances), for example, a simple code for avoiding the pattern “11011” is described in paper [4], with a redundancy of only 20%; it completely eliminates the 11011 pattern. We have been guided by the same level of acceptable redundancy (20% or less) but our purpose has been rather different: as mentioned in section I, we wish to propose a code with tuneable redundancy that eliminates just enough occurrences of the undesirable pattern to bring the BER down to a level that FEC can process. For long patterns that have a small rate of occurrence, e.g. 11011, which would occur in about 3% of quintuplets in a random packet, tuning could be compatible with complete elimination: indeed, one could control the number of different undesirable patterns to eliminate. However, if one were to completely eliminate short patterns, in our case triplets, those contribute from about 15% to 30% redundancy each, which makes fine tuning difficult.

The code presented below achieves near continuous tuneability using a version of the enumeration method [17]. We must emphasise that the code in question can not be regarded as an instance of the general RLL code, nor of Vasic’s graph model[4]. Since it reduces, rather than eliminates, the incidence of a pattern, it is a weakly-constrained code rather than a constrained code.

The next section will lay theoretical foundations of our approach.

---

1 the best redundancy we have found in [12] was about 30%, but they also quote redundancies up to 85%, see [12] p. 70.

2 We enumerate triplets according to their binary code, e.g. triplet 6 is 110.
for different \( \varepsilon \) the minimum-redundancy code for a given 101 skew ties the were found. They are shown in Table I. Ref [8] states that this language, by introducing a skew similarly to the Markov model discussed earlier. Let us reduce the probability of \( z_1 \), which is equal to the frequency of the 101 triplet, and increase the probability of every other \( z_k \), \( k > 1 \) to compensate:

\[
P(z_k) = \varepsilon \cdot \theta(k - 2)2^{-(k-1)} + (1 - \varepsilon)2^{-k}.
\]

Here \( \theta(m) = 0 \) for negative \( m \) and 1 otherwise. This distribution is correctly normalised for any \( \varepsilon \). A direct calculation of the entropy gives

\[
H(Z) = 1 + \varepsilon + g(\varepsilon),
\]

where \( g(\varepsilon) \) is exactly as defined earlier. The average length of one \( Z \) is

\[
L(Z) = 2 + \varepsilon.
\]

Remembering that \( Zs \) account for half of the string symbols and the other half has \( H(U) = L(U) = 2 \) the redundancy

\[
R = 1 - \frac{H(Z) + H(U)}{L(Z) + L(U)} = 1 - \frac{g(\varepsilon)}{4 + \varepsilon}
\]

exactly as suggested by the Markov model. The frequency of the 101 triplet in a long \( ZU \) string is the probability of the symbol \( z_1 \), \( (1 - \varepsilon)/2 \), normalised by the length of the \( ZU \) pair, \( 4 + \varepsilon \), which gives \( (1 - \varepsilon)/(8 + 2\varepsilon) \), again in full agreement with the Markov model.

It is notable that the \( ZU \)-language is a handy instrument for constrained code analysis. Other state frequencies can be reformulated in the \( ZU \) language very easily. For example, the frequency

\[
P_7 = \frac{1}{L(ZU)} \sum_{k=3}^{\infty} (k - 3)P(u_k).
\]

The symmetries inherent in the Markov solution, such as \( P_3 = P_4 \) can now be proven in the general case by observing that the pattern 100 can only be detected in the \( Uz_kU \) situation with \( k \geq 2 \) but then the same \( Uz_kU \) combination would deliver the 001 pattern. Similarly, \( P_3 = P_6 \).

The Markov model is still very useful because, as the aforementioned example of \( P_7 \) shows, a finite calculation in the Markov chain corresponds to an infinite summation over the \( ZU \) language, which may or may not be possible analytically. Nevertheless, unlike the Markov chain, the \( ZU \) approach empowers us to make the next step. Below we use the \( ZU \)-language for calculation of the bit-stuffing code properties.

### IV. ZU LANGUAGE

Let us now consider a slightly different representation of a bit string. Imagine a language based on two sets of symbols: \( z_k \) representing a string of zeroes of size \( k \) and similarly \( u_k \), a string of ones sized \( k \), \( k > 0 \). Any bit string \( s \) of nonzero length represents the following structure:

\[
s = \ldots U ZU Z U Z \ldots
\]

where each \( Z \) stands for \( z_k \) with some \( k \), generally different for different \( Z \), and each \( U \) similarly for a \( u_k \). The string can begin and/or end with either \( Z \) or \( U \), but since the end-effects are not important for statistical calculations on a long string, we can limit ourselves to strings of \( UZ \)-pairs without loss of generality.

In an uncorrelated random bit string, the frequency of each of the symbols is \( P(u_k) = P(z_k) = 2^{-k} \). It is easy to calculate the entropy of the string given those probabilities. Since the order of \( Zs \) and \( Us \) is fixed, the entropy is the sum of the entropies of the \( Z \)- and \( U \)-substrings, and furthermore, the sum of the entropies of individual \( Z \) and \( U \) symbols. Consequently,

\[
H(U) = - \sum_{k=1}^{\infty} P(u_k) \log_2 P(u_k) = 2
\]

and the average length in bits of a symbol \( u_k \) is

\[
L(U) = \sum_{k=1}^{\infty} kP(u_k) = 2
\]

and the same for symbols \( z_k \). Thus the information density of the uncorrelated string is precisely 1 bit/digit as expected.

We are now prepared to analyse a 101-skewed code using this language, by introducing a skew similarly to the Markov model discussed earlier. Let us reduce the probability of \( z_1 \),...
$\varepsilon = 1$ the code is strongly-constrained (all 101s are removed from the message), and at $\varepsilon = 0$ the code has no constraints whatsoever.

A. Bit stuffing

Let us now consider a practical code. After the example of bit stuffing which is well-known in digital recording technology, let us insert an extra 0 in a Z-symbol to make it longer and thus reduce the probability of the pattern 101. For this to be practical, we need to be able to reverse the effect of stuffing, i.e. we need to know that an extra 0 has been inserted in the received bit sequence so that we may remove it. This can be achieved by making the stuffing decision conditional on the preceding U-symbol. Here is our proposed code $C_t$:

$$C_t(u_k z_m) = \begin{cases} u_k z_m, & \text{if } k < t \\ u_k z_{m+1}, & \text{if } k \geq t \end{cases}$$

The inverse code is, obviously,

$$C_t^{-1}(u_k z_m) = \begin{cases} u_k z_m, & \text{if } k < t \\ u_k z_{m-1}, & \text{if } k \geq t \end{cases},$$

and is undefined on any $u_k z_0$ combination with $k \geq t$, as such combinations are prevented by the direct code. Code $C_t$ is controllable via the value of $t$: since the frequency of $u_k$ falls exponentially with $k$, by choosing $t$ large enough one can achieve the smallest desirable degree of stuffing. As a limiting case, code $C_t$ always stuffs an extra zero into a Z, thus completely eliminating the symbol 1 and its corresponding pattern 101.\(^3\) We conclude that the code $C_t$ gives a good degree of control over the probability of the pattern 101 at large $t$.

Notice that the code we have described suffers from the same problem as convolution codes in FEC: it has the tendency to spread an error by making nonlocal coding decisions. Indeed the above code mapping is local only in the ZU representation as it maps a ZU pair on a ZU pair. However an error 1 $\rightarrow$ 0 disrupting a string of 1s within a single U will effectivly insert an extra ZU pair. This means that a code of the type described above would require some support from an FEC code to neutralise those types of errors; however, when the skew-related effects are much stronger than random errors, the bit-stuffing code could be quite effective in combination with FEC.

Let us now evaluate the effect of bit stuffing on redundancy and the frequency of 101 as we did with the Markov model. It is easy to see that the probability of the UZ pair is as follows:

$$P(u_k z_m) = \begin{cases} 2^{-(k+m)} & \text{if } k < t \\ 2^{-(k+m-1)} & \text{if } k \geq t, m \geq 2 \\ 0 & \text{if } k \geq t, m = 1 \end{cases},$$

since the stuffed symbol retains its frequency but becomes one bit longer. After a straightforward calculation we obtain the average length of the resulting UZ-pair:

$$L(C_t(U Z)) = 4 + 2^{-(t-1)}.$$

Unsurprisingly, the length is 5 when $t = 1$ as we always stuff one zero into the Z symbol, which corresponds to the 20% redundancy under the Markov chain with $\varepsilon = 1$. As for the entropy $H(C_t(U Z))$, it remains the same $H = 4$ since the code is reversible, hence the image under the code contains exactly the same information\(^4\). The output frequency $P(U z_1)$ is

$$P(U z_1) = \frac{t-1}{t} \left( \sum_{k=1}^{t-1} P(u_k)P(z_1) \right) = \frac{t-1}{t} \left( \sum_{k=1}^{t-1} 2^{-k-1} \right) = \frac{1}{2} \left( 1 - 2^{-(t-1)} \right)$$

which corresponds to the Markov model with $\varepsilon = 2^{-(t-1)}$.

B. Weakly-constrained block codes

One way of dealing with the error propagation of a bit-stuffing code is to use flexible blocks. The idea of a weakly-constrained block code is as follows.

Let $W_m$ be a list of all length-$m$ bit strings. Denote as $W_m(j)$ the $j$th entry of the list and as $L_{m,k}$ the number of length $m$ bit strings with no more than $k$ 101s. The initial members of $W_m$ with indices $0 \leq j < L_{m,0}$ have no triplets 101, those with indices $L_{m,0} \leq j < L_{m,1}$ have exactly one 101 etc. Finally codewords with indices $L_{m,k_{\text{max}}-1} \leq j < L_{m,k_{\text{max}}}$ contain $k_{\text{max}}$ triplets 101. The $k_{\text{max}}$ value can be found for each specific $m$ by construction (for example $k_{\text{max}} = 2$ if $m = 5$, because 5-bit strings with three or more 101s do not exist).

Let us represent the source bitstream $S$ as a sequence of small blocks $s_i$ with the length $n \leq m$: $S = (s_1, s_2, ..., s_p)$. Denote as $D = (d_0, d_1, ..., d_p)$ the encoded message, where $d_i$ is a length-$m$ data block. Now the block code $B(m,n)$ converts $s_i$ blocks into $d_i$ blocks thus: $d_i = W_m(s_i), i = 1..p$. This way only the codewords with indices less than $2^n$ are used in the coding process. For decoding we need a table $W_m^{-1}(j)$, $0 \leq j < 2^m$. $W_m^{-1}(j) = q$ if $W_m(j) = q$. Then the decoding process can be defined as $s_i = W_m^{-1}(d_i), i = 1..p$. This code has the redundancy $R = (m-n)/n$.

**Code example:** Let us consider a code $B(9,8)$ that maps 8-bit strings onto 9-bit strings. It can be verified that $k_{\text{max}} = 4$. The values of $L_{9,k}$ are given in Table II.

It is clear from Table II that $B(9,8)$ codewords contain no more than one triplet 101, since, for this code, $0 \leq j < 2^m = 256$, $W_9(j)$ for $0 \leq j < 200$ does not contain 101, and $W_9(j)$ for $200 \leq j < 399$ contains exactly one 101.

The per-bit frequency of 101 in the output is approximately

$$\frac{1}{m L_{m,k}} \sum_{i=0}^{k} G_{m,i} \text{ where } G_{m,i} \text{ is the number of } m\text{-bit strings that contain exactly } i \text{ triplets 101.}$$

This is an approximation that neglects the triplets formed at the junctures of neighbouring blocks. These depend on the frequency of certain bits at the

\(^2\)The reader might think that at least $C_1$ is an instance of RLL, the more so that there is no tuning parameter left in the code. Still it is not the case: $C_1$ allows 1s to come in a series uninterrupted by 0s, while eliminating single 0s, and not multiple 0s, in the output, which is something quite impossible with RLL.

\(^3\)this can also be established by direct calculation
beginning and end of a block. Indeed, the combination 101 will arise around the starting bit of a block that starts with 01, provided that it follows a block ending with 1. Similarly, a 101 combination around the last bit of a block arises when the block ends with 10, and the next one starts with 1. The exact per-bit frequency of the 101 combinations is, consequently,

\[ \frac{1}{mL_{m,k}} \sum_{i=0}^{k} iG_{m,i} + \phi_{x1}\phi_{001} + \phi_{x10}\phi_{01}. \]

Here \( \phi_{001}, \phi_{01}, \phi_{x11}, \) and \( \phi_{x10} \) are frequencies of the blocks that begin with 01, begin with 1, end with 1, and end with 10, respectively. Note that due to the assumed randomness of the source bit stream, the neighbouring patterns are uncorrelated, which justifies the use of products of frequencies in the above formula.

We have constructed block codes in table form for \( m=8, 16, 28 \) by exhaustive search and computed all \( \phi \) and \( L_{m,k} \) for some small \( k \). For comparability, Fig. 2 shows \( R \) vs effective \( \varepsilon \), i.e., such at which \( P_0 \) of the Markov chain matches the value for the block code. Codes with \( m=8, 16, 28 \) are also practical due to their small table sizes while the 28-bit code is less so as it requires about 1Gb of table space, but even a code as large as this does not approach the redundancy level of the Markov chain. For larger block codes the table method is impractical, and an effective encoding algorithm is required.

VI. ANALYSIS OF THE WEAKLY-CONSTRAINED BLOCK CODE

In the previous section we introduced and discussed the block codes constructed numerically in table form. This gave us a chance to compare the statistics of constrained block coding with the Markov model, if only numerically and approximately. In this section we will give an analytical view of the block-code statistics which shows further deviations from the idealised Markov model and explains and quantifies them.

A. Frequency of 101 triplets

Consider a block of \( m \) bits. Out of the \( 2^m \) bit strings there will be some \( G_{m,0} \) strings that contain no triplet 101, \( G_{m,1} \), strings that contain exactly one such triplet, etc. Let us determine the value of \( G_{m,k} \) analytically.

Proposition 6.1: The following recurrence relation holds:

\[ G_{m,k} = 2G_{m-1,k} - G_{m-2,k} + G_{m-3,k} + G_{m-2,k-1} - G_{m-3,k-1}, \]

with the initial data

\[ G_{3,0} = 7, G_{3,1} = 1, G_{3,k} = 0 \text{ for any } k > 1; \]

\[ G_{4,0} = 12, G_{4,1} = 4, G_{4,k} = 0 \text{ for any } k > 1; \]

\[ G_{5,0} = 21, G_{5,1} = 10, G_{5,2} = 1, G_{5,k} = 0 \text{ for any } k > 2. \]

Proof: Denote as \( S_{m,k} \) a set of bit strings of length \( m \) having exactly \( k \) 101 triplets, \( |S_{m,k}| = G_{m,k} \), and denote as \( S_{m,k}^a \subset S_{m,k} \) the part made up of strings with the opening segment \( a \). Similarly introduce \( G_{m,k}^a = |S_{m,k}^a| \).

Consider an arbitrary \((m-1)\)-bit string and denote it as \( B \). Attach the next bit \( b \) from the data stream to the beginning of the string and denote the resulting string as \( bB \). Assume that the data stream is statistically unbiased and uncorrelated:

\[ P_{b=0} = P_{b=1} = 1/2. \]

Now determine the conditions under which \((bB) \in S_{m,k}\):

1) \( B \in S_{m-1,k} \) and \( B \notin S_{m-1,k}^1 \) (there are \( G_{m-1,k} - G_{m-1,k}^1 \) such strings);
2) \( B \in S_{m-1,k-1} \) (there are \( G_{m-1,k-1} \) such strings).

If \( b = 0 \), then obviously \((bB) \in S_{m,k}\) if \( B \in S_{m-1,k} \) (there are \( G_{m,k}^0 = G_{m-1,k} \) such strings). Since these cases are mutually exclusive, we have the following equation:

\[ G_{m,k} = G_{m,k}^0 + G_{m,k}^1. \]

Since prefixing 0 to a string does not increase the number of 101s, \( G_{m,k}^0 = G_{m-1,k} \). Consequently, \( G_{m,k}^1 = G_{m-1,k} - G_{m-1,k-1} = G_{m-2,k-1} - G_{m-3,k-1} \).

We arrive at the following simultaneous equations:

\[ \begin{align*}
G_{m,k} &= 2G_{m-1,k} + G_{m-1,k-1} - G_{m-1,k}^1, \\
G_{m,k}^1 &= G_{m-1,k}, \\
G_{m,k}^0 &= G_{m-1,k} + G_{m-1,k-1} - G_{m-1,k}^1.
\end{align*} \]

After straightforward substitutions, we arrive at the equation from the Proposition. The initial data can be verified by direct calculation.

The above recurrence relation is useful because it has the ability to predict the redundancy of the weakly-constrained block code for large values of \( m \), for which direct numerical evaluation may be difficult (though arguably still possible using the Monte-Carlo method), while the cost of evaluating the recurrences is small.

Let us first restrict ourselves to the code with minimum redundancy: \( k = 0 \), and denote \( G_{m,0} \) as \( G_m \). Proposition 6.1 will now come to:

\[ G_m = 2G_{m-1} - G_{m-2} - G_{m-3} = 0 \]

with the initial data

\[ G_3 = 7, G_4 = 12, G_5 = 21. \]

For \( m = G_{m+3} \) \((m = 0, 1, \ldots)\), we have \( y_m = 2y_{m+2} + y_{m+1} - y_m = 0 \).

Using standard methods, the solution of this equation can be found:

\[ y_m = c_1\xi^m + c_2\sin m\varphi + c_3\cos m\varphi, \]

where \( c_1 \approx 0.6846, c_2 \approx 0.1514, c_3 \approx -0.0496, \varphi \approx 1.408, \rho \approx 0.7549. \)

Note that since \( \rho < 1 \), \( y_m = O(\xi^m) \) in the limit of large \( m \). Since the number of code words of length \( m \) is in fact \( G_m = y^{m-3} \) and the number of all strings of length \( m \) is \( 2^m \), the asymptotic code capacity \( 1/m \log_2 G_m \rightarrow \log_2 \xi \approx 0.8114 \) per digit. This is higher than the Markov chain result 0.8.

Remember that our block codes are optimised for minimum redundancy at a given skew by sorting the list of code words in the ascending order in the number of 101 triplets per word.
Corollary

\[ \varepsilon P \] will include all codewords containing zero triplets 101. Therefore, \( P_{100} \neq 0 \), because the zero codeword is included in the codeword space anyway. Similarly, \( P_{110} \neq 0 \).

**Lemma 6.2:** For any \( k \geq 0, m \geq 5 \), the equation \( P(0000) = P(0010) \) holds.

**Proof:** Since neither the replacement 00000 \( \rightarrow \) 00100, nor its inverse modifies the frequency of 101s in the sequence, these replacements exchange a codeword \( B \in S(m, k) \) with another codeword \( B' \in S(m, k) \). Similarly it can be proven that \( P(00001) = P(00011), P(01011), P(00100), P(01110) = P(01101), P(01100) = P(00110), P(000010) = P(000110) \).

Since the block code for each \( \varepsilon \) will contain at least one codeword with patterns 0000, 0010, 1000, 1001, 1100, 1110, 1111, 0111, 0111, the inequalities \( |\varepsilon_0| \neq 1, |\varepsilon_4| \neq 1, |\varepsilon_5| \neq 1 \) and \( |\varepsilon_7| \neq 1 \) are correct. Therefore according to Table I, \( P_1 = P_1 = P_0(1 + \varepsilon_0)/(1 - \varepsilon_0), P_3 = P_0 = P_7(1 + \varepsilon_7)/(1 - \varepsilon_7) \) for any skew-parameter vector \( \varepsilon = (\varepsilon_0, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \varepsilon_5, \varepsilon_6, \varepsilon_7) \) for any skew-parameter vector \( \varepsilon = (\varepsilon_0, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \varepsilon_5, \varepsilon_6, \varepsilon_7) \).

Let us find this vector using the above codeword properties.

\[ \varepsilon = \varepsilon_0 \left( 1, -1, \frac{2\varepsilon_0^2 - 2\varepsilon_0 + 5}{(1 + \varepsilon_0)^2}, -1, \frac{2\varepsilon_0^2 - 2\varepsilon_0 + 5}{(1 + \varepsilon_0)^2}, -1 \right), \]

where every skew parameter is expressed in terms of a single skew \( \varepsilon_0 \) (see Table III).

However, to be able to juxtapose the multi-skew block code and a single-skew Markov chain, we must first define the basis for comparison. Such a basis could be an effective skew value \( \varepsilon_{\text{eff}} \) such that the density of 101 in the block code at a given \( \varepsilon \) corresponds to the density of 101 in the single-skew Markov chain at the skew value \( \varepsilon_{\text{eff}} \). In other words, \( \varepsilon_{\text{eff}} \) indicates how much the single-skew Markov chain would have to be skewed to achieve the same density of the undesirable states. The latter is simply \( P_5 = (1 - \varepsilon_{\text{eff}})/(8 + 2\varepsilon_{\text{eff}}) \) whereas from the Markov chain density and the \( \varepsilon \) expression above we get

\[ P_5 = \frac{\varepsilon_0^4 + 6\varepsilon_0^2 + 8\varepsilon_0 + 1}{8(2\varepsilon_0^2 + \varepsilon_0 + 1)} = (1 - \varepsilon_{\text{eff}})/(8 + 2\varepsilon_{\text{eff}}). \]

This gives us \( \varepsilon_0 \) in terms of \( \varepsilon_{\text{eff}} \).

### Table III: Calculation of the \( \varepsilon_i \) Values

<table>
<thead>
<tr>
<th>Property</th>
<th>Corollary</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(000001) = P_7P_0 = P_0P_7 + 1 )</td>
<td>( \varepsilon_1 = -\varepsilon_0 )</td>
</tr>
<tr>
<td>( P(00011) = P_7P_0 )</td>
<td>( \varepsilon_4 = \varepsilon_0 )</td>
</tr>
<tr>
<td>( P(01001) = P_5P_2 )</td>
<td>( \varepsilon_5 = -\varepsilon_0 )</td>
</tr>
<tr>
<td>( P(00010) = P_7P_0 )</td>
<td>( \varepsilon_2 = 1 - \frac{\varepsilon_0^2}{4(1 + \varepsilon_0)} )</td>
</tr>
<tr>
<td>( P(00100) = P_7P_0 )</td>
<td>( \varepsilon_3 = \frac{\varepsilon_0^2(\varepsilon_0^2 - 2\varepsilon_0 + 5)}{(1 + \varepsilon_0)^2} )</td>
</tr>
<tr>
<td>( P(00110) = P_7P_0 )</td>
<td>( \varepsilon_4 = \frac{\varepsilon_0^2(\varepsilon_0^2 - 2\varepsilon_0 + 5)}{(1 + \varepsilon_0)^2} )</td>
</tr>
<tr>
<td>( P(11010) = P_5P_2 )</td>
<td>( \varepsilon_5 = \varepsilon_2 )</td>
</tr>
<tr>
<td>( P(11011) = P_5P_2 )</td>
<td>( \varepsilon_3 = \varepsilon_2 )</td>
</tr>
<tr>
<td>( P(11101) = P_5P_2 )</td>
<td>( \varepsilon_4 = \varepsilon_2 )</td>
</tr>
</tbody>
</table>

**C. Calculation of the redundancy**

The above-mentioned quartic equation is of the form:

\[ \varepsilon_0^4 + 2\varepsilon_0^2(3 - 8P_5) + 8\varepsilon_0(1 - P_5) - 8P_5 + 1 = 0 \]

and can be proven to have a single root in the interval \(-1 \leq \varepsilon_0 \leq 1 \) given that \( 0 \leq P_5 \leq 1/8 \). Obtaining the root numerically presents no technical problem, so \( \varepsilon_0 \) can be assumed to be functionally dependent on \( \varepsilon_{\text{eff}} \) in a known way.

Consequently, let us express the entropy in terms of \( \varepsilon_0 \).

Using the general formula from [8] and interpreting \( \varepsilon \) as a function of \( \varepsilon_0 \) we get

\[ h(\varepsilon_0) = -\sum_{k=0}^{7} P_k(\varepsilon(\varepsilon_0))f(\varepsilon(\varepsilon_0)), \]

where \( f(x) = \frac{1 + x}{2} \log_2 \frac{1 + x}{2} + \frac{1 - x}{2} \log_2 \frac{1 - x}{2} \). By simplifying the equations from Table I for four \( \varepsilon_0 \) as mentioned above and using 2 we establish that

\[ h(\varepsilon_0) = f(\varepsilon_0)(P_0(\varepsilon_0) + P_1(\varepsilon_0) + P_2(\varepsilon_0) + P_3(\varepsilon_0) + P_4(\varepsilon_0) + P_5(\varepsilon_0) + P_6(\varepsilon_0) + P_7(\varepsilon_0)) + f(\varepsilon_2(\varepsilon_0))(P_2(\varepsilon_0) + P_6(\varepsilon_0)). \]

By varying \( \varepsilon_{\text{eff}} \) (and via it, \( \varepsilon_0 \)) over its range, we obtain the curve \( B_{\infty} \) in Fig. 2. Observe that block size 128 is roughly as close to the result obtained from the single-skew Markov chain as the theoretical limit of all weakly-constrained codes with a single skew, i.e. the curve “Markov chain”. It seems likely that for any practical purpose block size 128 will be sufficient. Another consideration would be the required level of interleaving and/or FEC to mitigate significant error propagation that a block code longer than 128 would be prone to.

**D. Code gain**

We assume that the vector of error probabilities is \( \bar{Q} = q(1, 1, 1, 1, 1, M, 1, 1) \). Since the code constructed above is a
The value of \( \Gamma \) corresponds to the increase of SNR that delivers the same error probability reduction in the case of AGWN without patterning as the block code delivers under the patterning defined by the vector \( \tilde{Q} \).

Fig. 3 shows the code gain as a function of the BER for the block code Bk128 (see Fig. 2) taken at 10% redundancy. It is remarkable that a gain of about 0.5dB is possible for a channel with \( BLER = 0.3 \) \((BER = 2.7 \cdot 10^{-3})\) from the simple code with \( R = 10\% \). It should be borne in mind that the gain displayed in the figure does not take into account any processes associated with decoding (and long block codes are almost as prone to error propagation as convolution codes). Nevertheless, if adequately protected by FEC, the effects of error propagation with a moderate block size can be alleviated. Schemes for a constrained code and FEC concatenation are considered in [17]. Similar schemes can be used in our case, too. It may also be possible to devise a soft decoding scheme based on the algorithm presented in the next section, which is less prone to error propagation effects.

### E. Encoder/decoder algorithms for large block sizes

In contrast to the Markov model the weakly-constrained block code described above includes, by construction, the maximum number of code words satisfying the pattern constraint for a given \( \varepsilon \). Consequently it has the least redundancy among all codes with that \( \varepsilon \). It is therefore quite important to find algorithms that realise the encoder and decoder mappings efficiently and to get an idea of their complexity.

For a small \( m \), the best coding strategy is table look-up: build a table of code words indexed by the source message block and use it for encoding. This way encoding is achieved by a single table reference per block. Decoding could be done similarly, by preparing the inverse table in advance. As the value of \( m \) increases (in practice already at \( m \geq 32 \)), storage requirements become prohibitive, necessitating a computational rather than purely table-based solution. As we have seen earlier, the redundancy curve saturates near \( m = 128 \), where table look-up is many orders of magnitude beyond any reasonable storage capacity, so the need for an efficient algorithm will not be eliminated by improvements in storage capacity in the foreseeable future. Besides low redundancy, an
advantage of a large \( m \) is the fact that the list of codewords, which is of the length \( L_{m,k} \), could be shortened to the length \( 2^m \) where \( n = \lfloor \log_2 L_{m,k} \rfloor \) increasing redundancy by a fraction of one bit for every \( n \) bits of the message, a negligible increase for \( m = 128 \) (corresponding to \( n > 0.8 \cdot 128 \approx 100 \)). The benefit of the shortening is that the shortened code replaces a string of \( n \) bits by a string of \( m \) bits and then applies the same function to the next portion of \( n \) bits. Below we shall only deal with a single block.

**Specification** Given the index of an \( m \)-bit code word in the (imaginary) table for block code \((m,k)\), compute the code word. Assume that \( m \geq 5 \), otherwise use actual table look-up. Similarly for decoding: given a valid \( m \)-bit code word for the block code \((m,k)\), i.e. any \( m \)-bit string with no more than \( k \) 101-triplets, find the index of this code word in the imaginary code table. The code table is assumed to have been sorted in ascending order within each \( m \)-bit string.

**Solution idea:** Generally, encoding/decoding procedures are based on enumerative techniques. In [17] enumerative algorithms for RLL-sequences were proposed. Below we develop a similar algorithm for a constrained code other than RLL, namely the weakly-constrained block code for the suppression of triplets 101.

Denote as \( N \) the index of the code word that contains no more than \( k \) 101-combinations. Then \( 0 \leq N < L_{m,k} \). The algorithm emulates a reference to the code look-up table, in which elements with indices \( 0 \leq N < G_{m,0} \) have no 101 triplets, those with indices \( G_{m,0} \leq N < G_{m,1} \) have exactly one 101 triplet, etc. and generally those with indices \( \sum_{j=0}^{k-1} G_{m,j} \leq N < \sum_{j=0}^{k} G_{m,j} = L_{m,k} \) have exactly \( k \) undesirable triplets. Consequently this virtual table is laid out in the increasing order of codeword values and nondecreasing order of triplets-per-word as required for the block code.

For encoding we need first to find the value of \( d \): the number of 101-triplets in the result code word, \( d = \max\{p \mid N \geq \sum_{j=0}^{p-1} G_{m,j}\} \). Next compute \( M = N - L_{m,d-1} \) the code word index in the section of the table with \( d \) 101-triplets, counting from the bottom of the segment; it is obvious that \( 0 \leq M < G_{m,d} \).

Denote the \( i \)-th bit of the code word as \( b_i \) \((0 \leq i < m-1)\) and the numbering is from the least significant bit. The code word is computed bit-by-bit starting with the most significant bit. The computation closely follows the recurrence relation (Eq. 1).

1. Find \( b_{m-1} \): Introduce a running \( M_i \), \( 0 \leq i \leq m-1 \) and set \( M_{m-1} = M \). If \( 0 \leq M < G_{m-1,d} \) we determine \( b_{m-1} = 0 \) and then we set \( M_{m-2} = M_{m-1} \). Otherwise, if \( G_{m-1,d} \leq M < G_{m,d} \) we determine \( b_{m-1} = 1 \) and set \( M_{m-2} = M_{m-1} - G_{m-1,d} \). Counting from the bottom of the segment; it is obvious that \( 0 \leq M < G_{m,d} \).

2. Assume that \( p \geq 1 \) most significant bits have been determined: \( b_{m-1}, b_{m-2}, \ldots, b_{m-p} \). These bits will include \( d_p \leq d \) 101-triplets. We determine that \( b_{m-p-1} = 0 \), and set \( M_{m-p-1} = M_{m-p} \), if \( 0 \leq M_{m-p} < A \), and otherwise we conclude that \( b_{m-p-1} = 1 \), and set \( M_{m-p-1} = M_{m-p} - A \). Here \( A \) is a threshold value, which depends on the two previously determined bits (we can legitimately assume \( b_m = 0 \) for the very first application of step 2).

A variable \( A \) is a threshold value, which depends on the two previously determined bits (we can legitimately assume \( b_m = 0 \) for the very first application of step 2).

The threshold values are summarised in Table IV. After elementary transformations that use previously established identities, we can eliminate the upper indices. Threshold values expressed in terms of \( G_{m,k} \) are given in Table V.

For any possible \( m-p \) and \( d-p \) combinations, the values of \( A_{0,1} \) can be obtained from a table prepared once beforehand, bearing in mind the obvious boundary values \( G_{m,0} = 1 \) for \( m < 0 \), \( G_{m,0} = 2^m \) for \( 0 \leq m < 3 \), \( G_{m,k} = 0 \) for \( k < 0 \), and \( G_{m,k} = 0 \) whenever \( k > 0 \) and \( m < 3 \).

**F. Encoding/decoding algorithms**

The encoding/decoding algorithms are presented in the Table VI. The critical path for processing one bit through the encoding algorithm contains one left shift (\( t_s \)), four assignments (\( t_a \)), two bitwise comparisons (\( t_{bc} \)), one comparison (\( t_c \)) and one subtraction (\( t_s \)). Denoting the register transfer cost as \( T \), we get the total time per loop cycle: \( 9T + t_s + 4t_a + 2t_{bc} + t_c + t_s \). In reality the cost of comparisons and subtractions is generally dependent on \( m \), due to the carry propagation delay in subtractions and the result propagation towards the lower digits in comparisons. However, various solutions are available which accelerate the propagation, making the overall complexity estimate logarithmic in \( m \). The assignment cost, too, can be reduced: by speculatively executing both alternatives of the if-while unrolling the loop to increase its pipelined concurrency.

A microelectronic implementation of those measures can easily be obtained, but any technical details would go beyond the scope of the present paper. We only wish to note that the cost is generally \( O(m \log m) \) and that the constant in it can be made quite small by employing modern design principles in microelectronics.

Similarly, the critical path for the decoding algorithm includes one left shift (\( t_s \)), three assignments (\( t_a \)), two bitwise comparisons (\( t_{bc} \)), one addition (\( t_{add} \)) and one subtraction (\( t_s \)). The total time per loops cycle is thus \( 7T + t_s + 3t_a + 2t_{bc} + t_{add} + t_s \), which is similar to that for the encoding algorithm and the same considerations regarding implementation apply.

**G. Memory requirement**

Let us estimate the read-only storage size, which is required for \( m \)-bit block-coding, for any \( 0 \leq k \leq k_{\max} \). For this one needs to tabulate \( A_0(i,j) \) and \( A_1(i,j) \), where \( A_0(i,j) = \)

<table>
<thead>
<tr>
<th><strong>TABLE IV</strong></th>
<th><strong>THRESHOLD VALUES</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_{m-p+1} )</td>
<td>( b_{m-p} )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>TABLE V</strong></th>
<th><strong>THRESHOLD VALUES FOR ENCODING/DECODING ALGORITHM</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_{m-p+1} )</td>
<td>( b_{m-p} )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

VII. WEAKLY-CONSTRAINED BLOCK CODE FOR OTHER TRIPLETS

Since nonlinear effects in optical fibre communications can lead to intricate error statistics, not necessarily resulting from the "ghost" pulse phenomena alone, suppression of other triplets may be desirable in data transmission systems.

In this section we will denote as $G_{m,k}^a$ the number of length-$m$ sequences which contain no more than $k$ occurrences of the triplet $a$. For each triplet we have established recurrence relations similar to Eqs. 1. They are listed in Table VII and can be employed in an encoding/decoding algorithm. For calculation of $G_{m,k}^a$ the following initial data must be used:

\[
G_{1,j} = \frac{m!}{j!(m-j)!}, \quad \frac{m!}{(j-1)!(m-j)!} - \frac{m!}{(j-2)!(m-j)!} = 0, \quad 0 \leq j \leq m
\]

The total storage read-only capacity is $S \leq 2mn(k+1)(k+2)$ bits. In particular, block coding requires no more than 202 Kbytes of memory, if the block size is 128 bits, for any $k$.

To summarise the results of this section, the time complexity of the encoder/decoder is log-linear in $m$, $T = O(m \log m)$, and the read-only storage requirement is quadratic in $m$ and linear in $k$, $S = O(m^2 k)$.

VIII. CONCLUSIONS

A detailed analysis of block and stream versions of weakly-constrained codes generalizing the first theoretical results in
Fig. 4. Limiting block code redundancy for various triplets.

[8] has been presented. A redundancy measure is obtained against the effective skewing factor $\varepsilon$ for specific stream and block codes, and it turns out to be somewhat better than the theoretical model published previously. The effective code gain for practical block codes has been obtained and plotted against a range of bit error rates at various magnitudes of the patterning effect. It is shown that using these simple codes in a hard-decision situation provides an effective gain of about 0.5dB. For large blocks, which correspond to the codes in a hard-decision scheme for tedons, "sliding-block line codes to increase dispersion-limited distance of optical fiber channels," IEEE J. Sel. Areas Commun., vol. 13, no. 3, pp. 486-498, 1995.


**REFERENCES**


Alex Shafarenko received his graduate Diploma in Physics from the Novosibirsk State University Department of Physics in 1983, and a Ph.D. from the Siberian Branch of the Russian Academy of Sciences in 1990. He joined faculty at the University of Surrey, England as Senior Lecturer within the Department of Electronic and Electrical Engineering and was subsequently made a Reader. He was appointed to his current position Professor of Software Engineering in 2000 by the University of Hertfordshire as he joined the School of Computer Science. Dr Shafarenko has led several international research projects in the area of advanced signal processing and parallel computing.

Anton Skidin received his graduate Diploma in Electronic Engineering from Kargan State University in 2005. He is currently a Research Fellow of the Institute of Computational Technologies Russian Academy of Sciences Siberian Branch.

Sergei K. Turitsyn graduated from the Department of Physics of the Novosibirsk University, Russia in 1982 and received the Ph.D. degree in Theoretical and Mathematical Physics from the Institute of Nuclear Physics, Novosibirsk, Russia in 1986. From 1992 to 1998, he was with the Institute for Theoretical Physics I, Heinrich-Heine University, Duesseldorf, Germany, first as a Humboldt Fellow and later as Leader of the collaborative projects with Deutsche Telekom. He joined the Photonics Research Group in the School of Engineering and Applied Science, Aston University, United Kingdom in 1998 and is now one of the Photonics Group leaders. Professor Sergei Turitsyn was a recipient of the Royal Society Wolfson Research Merit Award in 2005.