Lossless Sliding-Block Compression of Constrained Systems[*]

John L. Fan[†] Brian H. Marcus[‡] Ron M. Roth[§]

September 28, 1999

Abstract

A method is presented for designing lossless sliding-block compression schemes that map constrained sequences onto unconstrained ones. The new compression scheme is incorporated into a coding technique for noisy constrained channels, which has applications to magnetic and optical storage. As suggested recently by Immink, the use of a lossless compression code can improve the performance of a modified concatenation scheme where the positions of the error-correcting code and constrained code are reversed (primarily in order to eliminate error-propagation due to the constrained code). Examples are presented that demonstrate the advantage of using sliding-block compression over block compression in a noisy constrained setting.

Keywords: Compression; Concatenated codes; Constrained systems; Noisy constrained channels; Sliding-block codes; Run-length-limited codes; State splitting.

1 Introduction

Constrained coding is a special kind of channel coding in which unconstrained user sequences are encoded in a lossless manner into a set S of sequences that satisfy certain hard constraints.

[*]This work was supported in part by grant Nos. 95-522 and 98-199 from the United-States–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

[†]Information Systems Laboratories, Department of Electrical Engineering, Stanford University, Stanford, CA 94309-2586, USA. e-mail: `jfan@isl.stanford.edu`.

[‡]IBM Research Division, Almaden Research Center K65/802, 650 Harry Road, San Jose, CA 95120, USA. e-mail: `marcus@almaden.ibm.com`.

[§]Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel. e-mail: `ronny@cs.technion.ac.il`.

The set S is called a *constrained system* (or simply a *constraint*) and is defined through a labeled finite directed graph G whose edges are labeled by elements of a finite alphabet Σ: the elements of S, referred to as the *constrained sequences*, are the finite words over Σ obtained from reading the labels of paths in G; we say that G is a *presentation* of S and G *generates* the words of S. Throughout this work, the term 'graph' will mean a labeled finite directed graph. A special case of constraints is the (d, k)*-run-length-limited* (RLL) constraint, which is defined as the set of binary words whose 1's are separated by runs of zeros of lengths between d and k [5], [12], [13]. For example, the $(2, \infty)$-RLL constraint is generated by the graph $G_{2,\infty}$ in Figure 1.

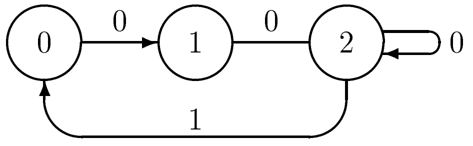


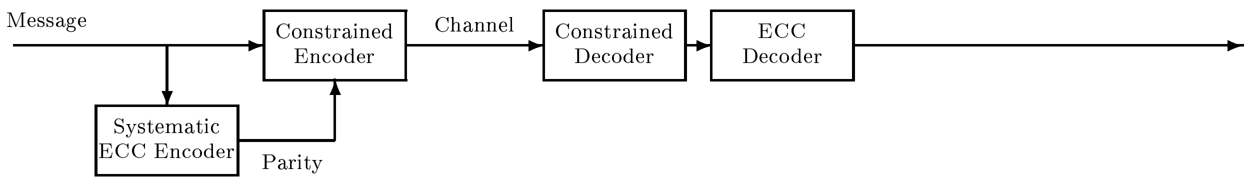
Figure 1: Graph presentation $G_{2,\infty}$ of the $(2, \infty)$-RLL constraint.

In *lossless data compression*, the roles of encoder and decoder are reversed: constrained sequences from a source are encoded into unconstrained sequences where no distortion is permitted upon decompression. This duality between constrained coding and lossless data compression has been noted by several authors (see for example [2], [7], [11], [16]). In these papers, data compression techniques such as arithmetic coding have been applied to constrained coding.

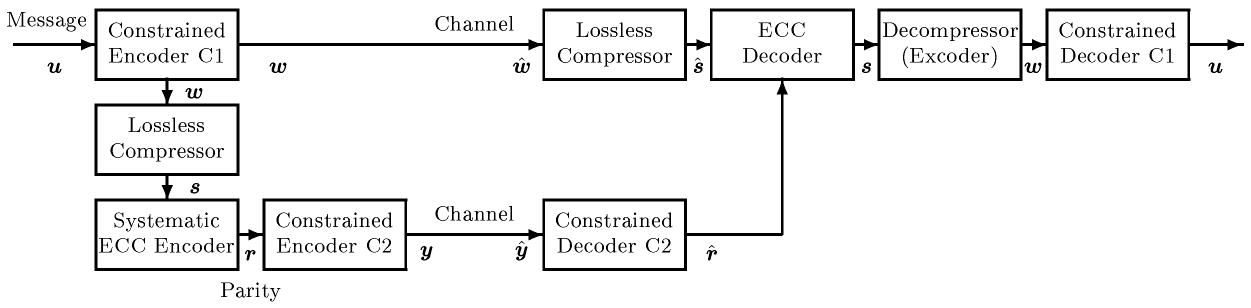
In this work, we transform a constrained coding construction (namely, a type of "reverse"-ACH algorithm [1], [10]) into a new lossless data compression construction. The ACH algorithm is guided by an "approximate eigenvector" which satisfies a certain inequality. Our new construction involves a similar kind of vector which satisfies a reversed inequality. This leads to some interesting consequences (e.g., see Proposition 3(c)).

The new construction for lossless data compression can be incorporated in a scheme for combining constrained coding and error-correcting codes (ECC), as suggested recently by Immink in [6]. This scheme has applications in magnetic and optical recording, where RLL and other constrained codes are commonly used. First, we review the standard concatenation scheme, in which information is first passed through an error-correction encoder and then a modulation encoder before being transmitted across the channel. At the receiver, the data is decoded via the modulation decoder and then the error-correction decoder. This is illustrated in Figure 2(a).

In the modified concatenation scheme (which was invented earlier than [6]; see Bliss [3] and Mansuripur [8]), the order of concatenation is reversed as shown in Figure 2(b). A user data sequence $\boldsymbol{u}$ is first encoded via a high rate-code C1 into a constrained sequence $\boldsymbol{w}$. In order to achieve the high rate (very close to capacity), long block lengths must



(a)



(b)

Figure 2: (a) Standard concatenation. (b) Modified concatenation with lossless compression.

be used. If $\boldsymbol{w}$ is then transmitted across a noisy (binary) channel, a small burst error or even a single isolated error in the received sequence $\hat{\boldsymbol{w}}$ could affect much or possibly all of the demodulated sequence $\hat{\boldsymbol{u}}$, yielding enormous error propagation. To avoid this, error correction is incorporated and used to correct all errors before decoding $\hat{\boldsymbol{w}}$. This is done by computing a sequence $\boldsymbol{r}$ of parity symbols on $\boldsymbol{w}$, and then encoding $\boldsymbol{r}$ into a constrained sequence $\boldsymbol{y}$ via a second constrained encoder C2 which is less efficient (namely, has a lower rate) than the first encoder; yet it operates on shorter blocks. Both constrained sequences $\boldsymbol{w}$ and $\boldsymbol{y}$ are then transmitted across the noisy channel. The sequence of parity symbols should be chosen to allow correction of a prescribed typical channel error event in $\boldsymbol{w}$, such as bursts of errors up to a certain length. The decoder attempts to recover $\boldsymbol{u}$ from the possibly corrupted versions $\hat{\boldsymbol{w}}$ of $\boldsymbol{w}$ and $\hat{\boldsymbol{y}}$ of $\boldsymbol{y}$. Since the constrained encoder C2 uses short block lengths, it is presumably subject to very little error propagation. Then the decoded version, $\hat{\boldsymbol{r}}$, of $\hat{\boldsymbol{y}}$ can be used to correct $\hat{\boldsymbol{w}}$, without fear of error propagation (i.e., the error events in $\hat{\boldsymbol{w}}\hat{\boldsymbol{r}}$ look roughly like the raw channel error events in $\hat{\boldsymbol{w}}\hat{\boldsymbol{y}}$). In this way, $\boldsymbol{w}$ is recovered error-free; decoding $\boldsymbol{w}$ via the first constrained decoder recovers $\boldsymbol{u}$ error-free.

One of Immink's key contributions in [6] was the realization that $\boldsymbol{w}$, being an encoded version of $\boldsymbol{u}$, is longer than $\boldsymbol{u}$, so that it may be necessary to increase the number of parity symbols $\boldsymbol{r}$ for the error-correcting code to achieve the same performance. In addition, for long bursts, the effect of a burst of channel errors is magnified relative to the standard concatenation scheme, since the bursts are not first decoded by the constrained decoder. Immink's solution to this problem was to *compress* the sequence $\boldsymbol{w}$ in a lossless (one-to-one) manner into a sequence $\boldsymbol{s}$, and then compute the sequence of parity symbols $\boldsymbol{r}$ based on $\boldsymbol{s}$; for instance, $\boldsymbol{s}$ (respectively, $\boldsymbol{r}$) could be the information (respectively, parity) portions of a Reed-Solomon code. So the parity sequence $\boldsymbol{r}$, and therefore also the modulated parity sequence $\boldsymbol{y}$, can be made shorter, thereby lowering the overhead of the error-correction scheme. At the channel output, the received sequence $\hat{\boldsymbol{w}}$ is compressed to a string $\hat{\boldsymbol{s}}$, and the ECC decoder recovers $\boldsymbol{s}$ from $\hat{\boldsymbol{r}}$ and $\hat{\boldsymbol{s}}$. Then the decompressor recovers $\boldsymbol{w}$, and the constrained decoder C1 recovers $\boldsymbol{u}$.

At one extreme, one could compress $\boldsymbol{w}$ back to $\boldsymbol{u}$ (in which case $\boldsymbol{s}$ would be the same as $\boldsymbol{u}$). But then a small channel error in $\hat{\boldsymbol{w}}$ could corrupt all of $\hat{\boldsymbol{s}}$ *before* error correction. Instead, the compression scheme will guarantee that such a channel error can corrupt only a limited number of bytes in $\hat{\boldsymbol{s}}$.

Examples of *block* compression codes for this purpose were given by Immink [6] and Fan and Calderbank [4]. Immink noted that the comparison of different block compression codes involves a non-trivial analysis of trading off of various factors. Compression codes were treated briefly in a precise conceptual context in Section II.C of [4]. Both of these papers served as inspiration for our work. The main difference between those papers and ours is that we allow the use of *sliding-block* compression codes.

2 Block codes

In Immink's scheme [6], the compression code is a simple *lossless block code* at rate $p : q$; this means that constrained words of length q, called *q-codewords*, are mapped one-to-one to unconstrained binary words of length p, called *p-frames*. Of course, a necessary and sufficient condition for such a code is

$$|S_q| \leq 2^p \, , \tag{1}$$

where S_q denotes the set of constrained words of length q in a given constraint S.

Immink gives in [6] two simple examples: a rate $8 : 11$ block code for the $(1, 12)$-RLL constraint and a rate $8 : 13$ block code for the $(2, 15)$-RLL constraint. For $p = 8$, these values of q are optimal: a simple computation reveals that condition (1) would be violated for any rate $8 : 12$ block code for the $(1, 12)$-RLL constraint and any rate $8 : 14$ block code for the $(2, 15)$-RLL constraint.

Clearly, $p = 8$ is a good choice owing to the availability of high performance, high efficiency, off-the-shelf Reed-Solomon codes. But allowing other values of p can give added flexibility in the choice of compression schemes (provided that p and the symbol alphabet of the ECC are somewhat compatible). Clearly, it is desirable to have a small compression rate p/q, and smaller compression rates can be achieved by larger block lengths p and q. But the capacity of the constraint imposes a lower bound on the compression rates, as we show next.

Recall that the capacity, $\mathsf{cap}(S)$, of a constraint S is defined by

$$\mathsf{cap}(S) = \lim_{\ell\to\infty} (1/\ell) \cdot \log |S_\ell| \ , \tag{2}$$

where the limit is known to exist [12, Section 3.2.1] (hereafter all logarithms are taken to base 2). Since $|S_{qm}| \le |S_q|^m$ for any choice of positive integers q and m, it follows that

$$\mathsf{cap}(S) = \lim_{m\to\infty} (1/(qm)) \cdot \log |S_{qm}| \le (1/q) \cdot \log |S_q| \ ;$$

that is, the limit in the right-hand side of (2) is taken over elements each of which is an upper bound on $\mathsf{cap}(S)$. Combining this with (1) yields

$$\mathsf{cap}(S) \le (1/q) \cdot \log |S_q| \le p/q \ . \tag{3}$$

Thus, to obtain compression rates p/q close to capacity, we need to take q (and hence p) sufficiently large so that $(1/q) \cdot \log |S_q|$ is close enough to capacity. This approach has several drawbacks. First, such schemes can be rather complex. Secondly, if the typical burst error length is short relative to q, then the compression code may actually expand the burst. Third, even if the typical burst error is of length comparable to q, it may be aligned so as to affect two or more consecutive q-codewords, and therefore two or more consecutive unconstrained p-frames; this "edge-effect" can counteract the benefits of using compression codes.

3 Sliding-block compressible codes

The foregoing discussion leads us to consider a more general class of compression codes, in particular *lossless sliding-block compression codes*. Such a code consists of a *compressor* and an *expanding coder* (in short, *excoder*). The compressor is a sliding-block code from sequences of q-codewords of S to unconstrained sequences of p-frames over $\{0, 1\}$; that is, a q-codeword $\boldsymbol{w}$ is compressed into a p-frame $\boldsymbol{s}$ as a time-invariant function of $\boldsymbol{w}$ and perhaps some m preceding and a upcoming q-codewords. The excoder, on the other hand, will have the form of a finite-state machine. The *sliding-block window length* is defined as the sum $\mathsf{m} + \mathsf{a} + 1$.

We next present a precise definition of the model of compressors and excoders considered in this work. For the sake of convenience, we start with excoders and then base the definition of compressors on that of the matching excoders. Let S be a constraint over an alphabet Σ, let Φ be a set of size n, and let m and a be nonnegative integers. An (m, a)*-sliding-block compressible* (S, n)*-excoder* is a graph $\mathcal{E}$ in which the edges are labeled by elements of Σ and, in addition, each edge is endowed by a *tag* from Φ so that the following holds:

(X1) the outgoing edges from each state are assigned distinct tags from Φ; in particular, each state will have at most n outgoing edges;

(X2) S is contained in the constraint (over Σ) that is presented by $\mathcal{E}$; and —

(X3) if $\boldsymbol{w}$ is a word in $S_{\mathsf{m}+\mathsf{a}+1}$, and $e_{-\mathsf{m}}e_{-\mathsf{m}+1}\ldots e_0\ldots e_{\mathsf{a}}$ and $e'_{-\mathsf{m}}e'_{-\mathsf{m}+1}\ldots e'_0\ldots e'_{\mathsf{a}}$ are sequences of edges that form two paths in $\mathcal{E}$ both labeled by $\boldsymbol{w}$, then the tags of e_0 and e'_0 agree.

Often we will apply this definition to a constrained system S whose alphabet consists of q-codewords in another constraint S', in which case $S_{\mathsf{m}+\mathsf{a}+1}$ will consist of words of length $(\mathsf{m}+\mathsf{a}+1)q$ in S' (see the definition below of a rate $p : q$ excoder).

The definition of an (m, a)-sliding-block compressible (S, n)-excoder $\mathcal{E}$ bears similarity to that of a tagged (m, a)-sliding-block decodable (S, n)-encoder as defined in [12, Section 3.6.1]: the main difference is in the containment relationship between S and the constraint presented by $\mathcal{E}$. Here, $\mathcal{E}$ must generate every word in S and, in addition, it may generate words that are not in S; note, however, that condition (X3) applies only to those paths in $\mathcal{E}$ that generate words in S.

Condition (X3) induces a mapping $\mathcal{C} : S_{\mathsf{m}+\mathsf{a}+1} \to \Phi$, which, in turn, defines the *sliding-block compressor* of $\mathcal{E}$ as follows. For any positive integer ℓ, the compressor maps every word

$$\boldsymbol{w} = w_{-\mathsf{m}}w_{-\mathsf{m}+1}\ldots w_0w_1\ldots w_{\ell-1}w_\ell\ldots w_{\ell+\mathsf{a}-1}$$

in $S_{\mathsf{m}+\mathsf{a}+\ell}$ into a pair $(v, \boldsymbol{s})$, where v is an *initial state* in $\mathcal{E}$, which can be the initial state of any path in $\mathcal{E}$ labeled by $w_0w_1\ldots w_{\ell+\mathsf{a}-1}$, and

$$\boldsymbol{s} = s_0s_1\ldots s_{\ell-1}$$

is a tag sequence in Φ^ℓ defined by

$$s_i = \mathcal{C}(w_{i-\mathsf{m}}w_{i-\mathsf{m}+1}\ldots w_i\ldots w_{i+\mathsf{a}}) \ , \quad 0 \le i < \ell \ .$$

Observe that the excoder can recover the sub-word $w_0w_1\ldots w_{\ell-1}$ of $\boldsymbol{w}$ by reading the labels along the (unique) path of length ℓ in $\mathcal{E}$ that starts at v and is tagged by $\boldsymbol{s}$.

A *block (S, n)-excoder* is an (S, n)-excoder with one state. Such an excoder is necessarily $(0, 0)$-sliding-block compressible.

In the examples of this paper, $\mathcal{E}$ will be (m, a)*-definite on S*: if $\boldsymbol{w}$ is a word in $S_{\mathsf{m}+\mathsf{a}+1}$, and $e_{-\mathsf{m}}e_{-\mathsf{m}+1}\ldots e_0\ldots e_{\mathsf{a}}$ and $e'_{-\mathsf{m}}e'_{-\mathsf{m}+1}\ldots e'_0\ldots e'_{\mathsf{a}}$ are two paths in $\mathcal{E}$ both generating $\boldsymbol{w}$, then $e_0 = e'_0$; note that (m, a)-definiteness is stronger than condition (X3). Moreover, the excoders in our examples will be $(0, \mathsf{a})$-sliding-block compressible where $\mathsf{a} \le 1$.

Next, we show how rate $p : q$ compression codes can be described through (S, n)-excoders and compressors. Let S be a constraint that is generated by a graph G. The *q-th power of S*, denoted S^q, is the set of all words in S whose lengths are divisible by q, where we group the symbols in each word into non-overlapping blocks from S_q. The set S^q is in fact a constraint which is generated by the *q-th power graph*, G^q, defined as the graph having the same set of states as G and one edge for each path of length q (with its labeling) in G [12, Section 2.3.1].

We now define an (m, a)*-sliding-block compressible excoder for S at rate $p : q$* to be an (m, a)-sliding-block compressible $(S^q, 2^p)$-excoder. The tag set Φ is taken as $\{0,1\}^p$, namely, the set of all possible values of any p-frame. So, a rate $p : q$ excoder for S maps p-frames into q-codewords in a state-dependent manner; the respective compressor, in turn, maps a sequence of q-codewords into a sequence of p-frames, where the i-th q-codeword is compressed into a p-frame through a time-invariant mapping $\mathcal{C} : S_{q(\mathsf{m}+\mathsf{a}+1)} \to \{0,1\}^p$ applied to the i-th q-codeword, as well as m preceding and a upcoming q-codewords. A *block excoder for S at rate $p : q$* is a rate $p : q$ excoder for S with one state. Note that Immink's scheme in [6] uses block excoders.

We next prove a necessary condition for the existence of (S, n)-excoders. Our proof makes use of the notion of *irreducibility* of graphs and constraints. A graph G is irreducible if for every ordered pair of states (u, v) in G there is a path in G from u to v. A constraint is irreducible if it has an irreducible graph presentation. It is known that for every constraint S there is an irreducible constraint $S' \subseteq S$ such that $\mathsf{cap}(S') = \mathsf{cap}(S)$ [12, Theorem 3.12]. An irreducible constraint is called *non-degenerate* if it contains words other than the empty word. Note that in an irreducible graph presentation of a non-degenerate irreducible constraint, each state must have at least one incoming edge and at least one outgoing edge.

Proposition 1 *Let S be a constraint. There is an (m, a)-sliding-block compressible (S, n)-excoder only if $\mathsf{cap}(S) \le \log n$.*

Proof. Let $\mathcal{E}$ be an (m, a)-sliding-block compressible (S, n)-excoder and let V and Φ be the set of states and the set of tags of $\mathcal{E}$, respectively. Suppose that S' is an irreducible constraint contained in S such that $\mathsf{cap}(S') = \mathsf{cap}(S)$. We assume that S' is non-degenerate, or else the result is obvious.

Let $\boldsymbol{w}$ be a word in S'_ℓ (and hence in S_ℓ). Since S' is irreducible and non-degenerate, the word $\boldsymbol{w}$ can be extended to a word $\boldsymbol{w}'\boldsymbol{w}\boldsymbol{w}'' \in S'_{\mathsf{m}+\mathsf{a}+\ell}$. The compressor of $\mathcal{E}$ maps the word $\boldsymbol{w}'\boldsymbol{w}\boldsymbol{w}''$ into a pair $(v, \boldsymbol{s})$, where $v \in V$ and $\boldsymbol{s} \in \Phi^\ell$, and the (unique) path in $\mathcal{E}$ that starts at v and is tagged by $\boldsymbol{s}$ generates the word $\boldsymbol{w}$. We have thus obtained through the compressor a one-to-one mapping from S'_ℓ into $V \times \Phi^\ell$; so,

$$|S'_\ell| \le |V| \cdot n^\ell \ .$$

The result follows by taking the limit as $\ell \to \infty$ and using the definition (2) of capacity. $\square$

Proposition 1 implies that there is a sliding-block compressible excoder for S at rate $p : q$ only if $\mathsf{cap}(S^q) \le p$ or, equivalently, $\mathsf{cap}(S) \le p/q$ (see [12, Proposition 3.13]). The latter inequality is exactly the reverse of Shannon's bound on the rate of (conventional) constrained encoders [12, Theorem 3.21]. The bound (3) is a special case of Proposition 1 for excoders with one state.

A constraint S has *finite memory* (or is of *finite type*) if there is a graph presentation G of S with the following property: there is an integer m such that all paths in G that generate a given word in S_m terminate in the same state of G. The smallest such m for a given graph presentation G is called the *memory* of G, and the memory of S is the smallest memory of any graph presentation of S. A (d, k)-RLL constraint is a constraint with memory k.

The next result, which we establish in Section 5, states that the condition in Proposition 1 is not only necessary, but also sufficient for constraints with finite memory.

Proposition 2 *Let S be a constraint with finite memory m and let n be a positive integer such that $\mathsf{cap}(S) \leq \log n$. Then there is an (m, a)-sliding-block compressible (S, n)-excoder; in fact, this excoder is (m, a)-definite on S.*

Finally, we make some remarks regarding the inclusion of the initial state in the information conveyed from the compressor to the excoder. The cost of transmitting this initial state is quite minimal. Typically, there will be a small number of states and the number of bits required to represent a state is only the logarithm of that number. Also, in Immink's scheme [6], one does not really need to expand the entire tag sequence after error correction: since channel decoding takes place after error correction and since the channel decoder has full knowledge of the received (uncompressed) constrained sequence, it need only re-expand the corrected p-frames in the compressed tag sequence; since a previously corrected portion of the received sequence is very likely to contain state information (for example if the excoder $\mathcal{E}$ is (m, a)-definite on S), no extra state information may be needed at all. Another alternative is to simply compress only those constrained sequences that can be generated from one fixed state of the excoder. When incorporated into Immink's scheme this would entail a loss in capacity, but the loss is very small since the block lengths are so long. A third solution, which is applicable to (m, a)-definite excoders, is to include the first $\mathsf{m}+\mathsf{a}+1$ q-codewords of the (non-compressed) constrained sequence in the bit stream that is protected by the ECC. Thus, the ECC decoder of the receiving end will reconstruct the correct prefix of the constrained sequence, thereby allowing us to recover the state information.

4 Application to burst correction

When used in conjunction with Immink's scheme [6] for combining constrained coding and error correction, there are a number of factors that may affect the choice of a compressor-excoder pair such as the complexity of the compression and expansion and the error-propagation associated with the application of the compression on the receiving end. In particular, we are concerned with how compressors handle raw channel bursts, and their suitability for use with a symbol-based ECC.

The compressor is applied to the channel bit sequence right after the channel, so that a benefit of using a low-rate excoder is that the length of a raw channel burst will be roughly decreased by the compression factor p/q, when the length of the burst is long (relative to q). On the other hand, edge effects in the use of a compressor can expand the error length, and this error propagation may dominate for short bursts. In addition, for sliding-block compressible excoders, the sliding-block window length will also extend the burst. Ultimately, the choice of a compressor-excoder pair involves a balance of these four factors:

1. compression rate $p : q$;
2. edge effects (how many extra p-frames are affected by the phasing of a burst);
3. effect of the sliding-block window length $\mathsf{m} + \mathsf{a} + 1$ (i.e., how many extra p-frames may be affected by each error); and —
4. compatibility between the frame length p and the symbol alphabet of the ECC.

We consider here the effect of a channel burst of length L bits on the maximum number of affected p-frames. Hereafter, by a length of a burst in a sequence over a given symbol alphabet we mean the number of symbols between (and including) the first and last erroneous symbols. Our computation will mainly concentrate on the simplified model where any error in the q-codeword will result in an entirely erroneous p-frame upon compression, although in practice it might be possible to mitigate this effect by a proper tag assignment to the excoder (see Section 5.4).

The maximum number of q-codewords (including the edge effect) that can be affected by a channel burst of length L bits is either $\lfloor (L-1)/q \rfloor + 1$ or $\lceil (L-1)/q \rceil + 1$, depending on the phasing within a q-codeword where the channel burst starts. For (m, a)-sliding-block compressible excoders, the effect of the memory and anticipation is to expand the number of affected p-frames by $\mathsf{m}+\mathsf{a}$, so that we get a maximum of

$$N = N(L) = \lceil (L-1)/q \rceil + \mathsf{m}+\mathsf{a}+1 \tag{4}$$

affected p-frames.

Next, we need to translate from a number of erroneous p-frames to a corresponding number of symbol errors for the ECC. Let the symbol alphabet of the ECC in Figure 2(b) be the finite field $\mathrm{GF}(2^B)$; namely, the sequence of p-frames is regarded as a long bit-stream and sub-divided into non-overlapping blocks of length B bits, each such block being a symbol of the ECC. We make the assumption that the boundaries between p-frames align with the boundaries between ECC symbols as often as possible, in particular, every $(pB)/\gcd(p, B)$

bits. We can then calculate the maximum number of ECC symbols that are in error due to a channel burst of length L bits.

Consider our basic unit to be of size $\gcd(p,B)$ bits, so that we are starting with a burst of length $(Np)/\gcd(p,B)$, and looking for the maximum number of affected blocks of length $B/\gcd(p,B)$. This is analogous to finding the maximum number of q-codewords affected by a burst of channel bits, and we obtain the following expression for the number of ECC symbols in error as a function of L and B:

$$D(L,B) = \left\lceil \frac{(Np)/\gcd(p,B) - 1}{B/\gcd(p,B)} \right\rceil + 1 = \left\lceil \frac{Np - \gcd(p,B)}{B} \right\rceil + 1 \; .$$

Putting this together with (4) yields

$$D(L,B) = \left\lceil \frac{(\lceil (L-1)/q \rceil + \mathsf{m}+\mathsf{a}+1)p - \gcd(p,B)}{B} \right\rceil + 1 \; .$$

Example 1 Consider a $(0,1)$-sliding-block compressible excoder for the $(2,\infty)$-RLL constraint at rate $4:7$ (such as the excoder that we will present in Example 5 in Section 5.3).

Table 1 contains the respective values of $D(L,B)$ for $L=40$ and $B=4,5,6,7,8$.

B	$D(40,B)$	ν	r	κ
4	8	544	64	840
5	8	1,320	(80)	2,170
6	6	2,340	72	3,969
7	6	5,418	(84)	9,334
8	5	10,280	80	17,850

Table 1: Parameters of an ECC used for burst lengths of up to 40 bits in conjunction with a $(0,1)$-sliding-block compressible excoder for the $(2,\infty)$-RLL constraint at rate $4:7$.

The last three columns in Table 1 show the parameters of an ECC that consists of $D(L,B)$ interleaving levels of a one-error-correcting extended Reed-Solomon code of length 2^B+1 over GF(2^B) [9, Ch. 10]. The overall block length (in bits) of this ECC scheme equals $\nu = \nu(L,B) = (2^B+1)\cdot B \cdot D(L,B)$; the values of ν are listed in the third column of the table. Since Reed-Solomon codes are maximum distance separable [9, Ch. 11], each symbol of GF(2^B) can be corrected using two redundancy symbols. Therefore, the total number of redundancy symbols is $2D(L,B)$, thereby attaining the Reiger bound [14, p. 110]. The redundancy (in bits) of the coding scheme thus equals $r = r(L,B) = 2\cdot B\cdot D(L,B)$; this is the length of “Parity” in Figure 2(b). The values of r are listed in the fourth column of the table, where numbers in parentheses indicate that smaller redundancy values (and larger ECC block lengths) can be obtained by using a larger value of B.

A block of ν bits at the output of the ECC encoder in Figure 2(b) corresponds to $\nu - r$ bits at the input of that encoder; those bits, in turn, correspond to $\kappa = \kappa(L, B) = \lfloor (\nu - r) \cdot q/p \rfloor$ channel bits at the input of the lossless compressor. The values of κ are listed in the fifth column of Table 1. Clearly, a smaller value of the ratio r/κ means a smaller overhead introduced by the ECC (when combined with the compression). $\square$

Using the ECC scheme and the notations of Example 1, we can fix a number, κ_0, of channel bits and compute for each (maximal) burst length L the respective redundancy r obtained by optimizing over all values of B for which $\kappa(L, B) \geq \kappa_0$. As an example, consider a message of 512 user bytes (4,096 bits) in Figure 2(b). Selecting the rate 256 : 466 code of [6] as the constrained encoder C1, the message is mapped into 7,456 channel bits. Figure 3 shows the best redundancy values attained for $\kappa_0 = 7,456$ and $L \leq 300$ using a $(0, 1)$-sliding-block compressible excoder at rate 4 : 7 for the $(2, \infty)$-RLL constraint. The figure shows the redundancy values also for two other block excoders for this constraint at rates 8 : 13 and 4 : 6; note that 8 : 13 is the rate of the excoder presented in [6]. Thus we see that for longer bursts, the sliding-block excoder requires less redundancy due to a better compression of the burst length, in spite of the longer sliding-block window. (We point out that Figure 3 is the same also for $\kappa_0 = 7,427$, which is the number we get when we divide 4,096 by the capacity ($\approx .5515$) of the $(2, \infty)$-RLL constraint.)

5 Construction of sliding-block compressible excoders

Our construction of excoders (and respective compressors) follows the lines of the *state-splitting algorithm* (also known as the Adler-Coppersmith-Hassner algorithm, in short, ACH) for constructing conventional finite-state encoders for constraints [1], [12, Section 4]. In order to describe the construction, we need some concepts from constrained coding. We will review these below, but for more details the reader may consult Sections 2, 3 and 4 of [12].

5.1 State splitting

A *state splitting* of a graph G (called an out-splitting in [12, Section 4.1] and sometimes called a *round of state splitting*) is obtained by partitioning the set, E_u, of outgoing edges from each state u of G into $N = N(u)$ disjoint sets,

$$E_u = E_u^{(1)} \cup E_u^{(2)} \cup \cdots \cup E_u^{(N)} , \tag{5}$$

replacing u by *descendant states* $u^{(1)}, u^{(2)}, \ldots, u^{(N)}$, assigning $E_u^{(r)}$ as outgoing edges from $u^{(r)}$, and replicating all edges incoming to a state v to each of its descendants $v^{(r)}$.

In the conventional state-splitting algorithm, we begin with a graph presentation G of the given constraint S. Typically, we assume that G is *deterministic*, i.e., at each state,

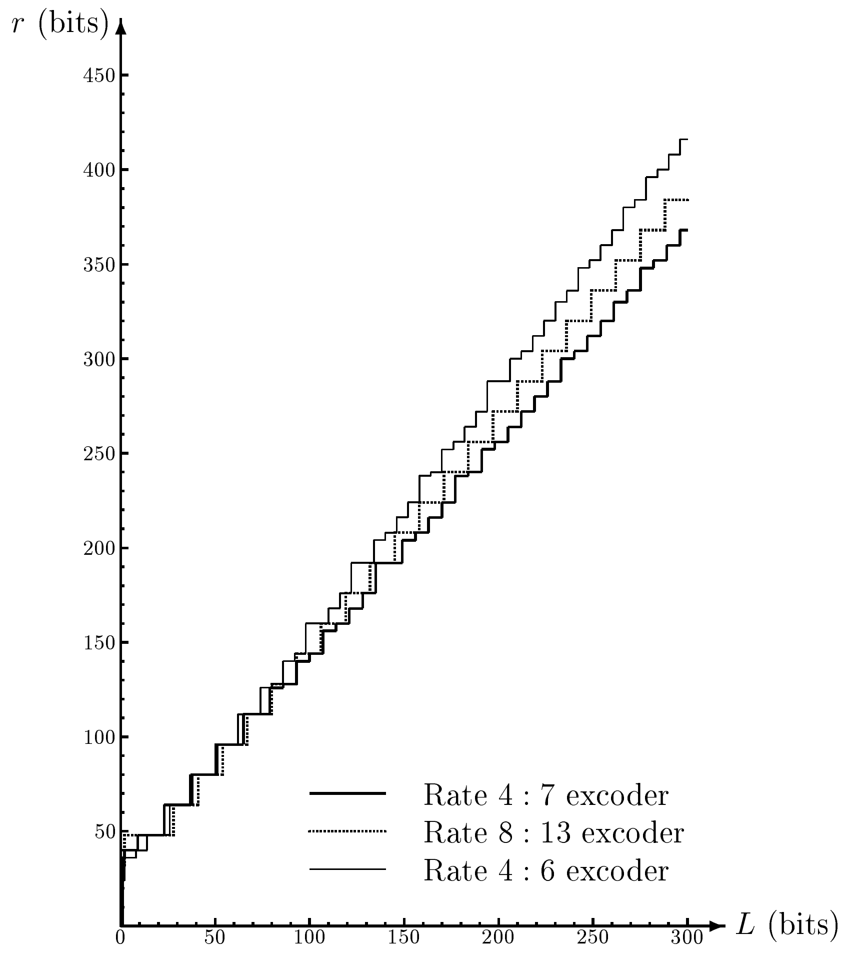


Figure 3: Redundancy for 4, 096 user bits and various burst lengths using three different excoders for the $(2, \infty)$-RLL constraint.

all outgoing edges are labeled distinctly. For example, the graph $G_{2,\infty}$ in Figure 1 is a deterministic presentation of the $(2, \infty)$-RLL constraint. Note that the q-th power graph of any deterministic graph is also deterministic. Given a (deterministic) presentation G of the constraint S, the state-splitting algorithm generates an encoder for S through a sequence of state splittings, which are guided by certain nonnegative integer vectors called *approximate eigenvectors.*

The algorithm we present here is very similar; the main difference is that instead of approximate eigenvectors, we will use what we call *super-vectors.* Such a vector will lead us through a sequence of state-splitting operations beginning with the graph G and ending with a graph H with out-degree at most n (i.e., each state has at most n outgoing edges). Then one assigns to the edges of H tags taken from the tag alphabet Φ (of size n) such that at each state all outgoing edges have distinct tags; typically, $n = 2^p$ and $\Phi = \{0, 1\}^p$. The tagged version of H will be the (S, n)-excoder $\mathcal{E}$.

In order to guarantee the sliding-block compressibility of $\mathcal{E}$, we will assume that S has finite memory m, in which case we take G as a (necessarily deterministic) graph presentation of S that has memory m. When state splitting is applied to this graph, we are guaranteed to end up with an excoder $\mathcal{E}$ which is (m, a)-definite on S, where a is the number of state splittings; in particular, $\mathcal{E}$ will be (m, a)-sliding-block compressible.

5.2 Super-vectors

As is the case with approximate eigenvectors, super-vectors will be computed using the *adjacency matrix* A_G of the graph presentation G of S: the rows and columns of A_G are indexed by the states of G, and entry (u, v) in A_G is the number of edges from state u to state v. For a deterministic presentation, the adjacency matrix can be used to compute the capacity of S; namely $\mathsf{cap}(S) = \log \lambda(A_G)$, where $\lambda(A_G)$ is the largest (absolute value of any) eigenvalue of A_G [12, Theorem 3.12]. Note that $(A_G)^q = A_{G^q}$ and, so, $(\lambda(A_G))^q = \lambda(A_{G^q})$.

Example 2 The adjacency matrix of the graph $G_{2,\infty}$ in Figure 1 is

$$A_{G_{2,\infty}} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} ,$$

and the capacity of the $(2, \infty)$-RLL constraint is given by $\log \lambda(A_{G_{2,\infty}}) \approx \log 1.4656 \approx .5515$. By Proposition 1, we will be able to construct a sliding-block compressible excoder for the $(2, \infty)$-RLL constraint only if its rate $p : q$ satisfies $(\lambda(A_{G_{2,\infty}}))^q \leq 2^p$. In the running example of this section, we will choose $p = 4$ and $q = 7$, in which case $(\lambda(A_G))^q \approx 14.5227 \leq 16 = 2^p$. $\square$

Let A be a nonnegative integer square matrix and let n be a positive integer; e.g., $A = A_G^q$ and $n = 2^p$. An (A, n)-*super-vector* is a nonnegative integer vector $\boldsymbol{x}$, not identically zero, such that

$$A\boldsymbol{x} \leq n\boldsymbol{x} .$$

Note that approximate eigenvectors used in constrained coding are defined the same way except that the inequality is reversed (with the benefit of hindsight, approximate eigenvectors should probably have been called "sub-vectors").

By a straightforward modification of the proof of Theorem 3.6 of [12], it follows that for any nonnegative integer square matrix A, there exists an (A, n)-super-vector if and only if $\lambda(A) \leq n$. The proof suggests ways of finding such a vector, but a simpler algorithm is presented in Figure 4; this algorithm is the analogue of the Franaszek Algorithm [12, Section 3.1.4] for finding approximate eigenvectors.

```
$\boldsymbol{x} \leftarrow (1\,1\,\ldots\,1)^T$;
while ($A\boldsymbol{x} \not\leq n\boldsymbol{x}$)
        $\boldsymbol{x} \leftarrow \max\left\{\left\lceil \frac{1}{n}A\boldsymbol{x}\right\rceil, \boldsymbol{x}\right\}$;      /* apply $\lceil\cdot\rceil$ and $\max\{\cdot,\cdot\}$ componentwise */
return $\boldsymbol{x}$;
```

Figure 4: Reversed Franaszek algorithm for computing (A, n)-super-vectors.

Next, we summarize the properties of this algorithm—in particular, the uniqueness of a minimum (A, n)-super-vector $\boldsymbol{x}^*$ and the fact that the algorithm always converges to $\boldsymbol{x}^*$. Parts (b) and (d) of the following are analogous to Proposition 3.8 of [12] for approximate eigenvectors. On the other hand, for approximate eigenvectors, there is no analogous uniqueness result and there is no clear choice of initial vector for the algorithm. For convenience we assume that A is irreducible, i.e., for any pair of row and column indexes (u, v) there is a positive integer m such that $(A^m)_{u,v} > 0$; this assumption holds for the adjacency matrix of an irreducible graph, which is the typical (and practical) case.

Proposition 3 *Let A be an irreducible nonnegative integer square matrix and let n be a positive integer with $\lambda(A) \leq n$. Then the following holds:*

(a) Any (A, n)-super-vector is strictly positive.

(b) If $\boldsymbol{x}$ and $\boldsymbol{x}'$ are (A, n)-super-vectors, then the vector defined by $\boldsymbol{z} = \min\{\boldsymbol{x}, \boldsymbol{x}'\}$ is also an (A, n)-super-vector (here, $\min\{\cdot,\cdot\}$ is applied componentwise).

(c) There is a unique minimum (A, n)-super-vector, i.e., a unique (A, n)-super-vector $\boldsymbol{x}^*$ such that for any (A, n)-super-vector $\boldsymbol{x}$ we have $\boldsymbol{x}^* \leq \boldsymbol{x}$.

(d) The algorithm in Figure 4 eventually halts and returns the vector $\boldsymbol{x}^*$.

Proof. (a) Let $\boldsymbol{x} = (x_u)_u$ be an (A, n)-super-vector. If some entry x_u is 0 then, according to the inequality $A\boldsymbol{x} \leq n\boldsymbol{x}$, we have $x_v = 0$ for all v such that $A_{u,v} \neq 0$. By irreducibility of A, this implies that $\boldsymbol{x}$ is identically zero, contrary to the definition of an (A, n)-super-vector.

(b) Since A is nonnegative, $A\boldsymbol{z} \leq A\boldsymbol{x} \leq n\boldsymbol{x}$ and $A\boldsymbol{z} \leq A\boldsymbol{x}' \leq n\boldsymbol{x}'$; so, $A\boldsymbol{z} \leq n\boldsymbol{z}$. Clearly $\boldsymbol{z}$ has only integer entries. By (a), $\boldsymbol{x}$ and $\boldsymbol{x}'$ are strictly positive, and thus so is $\boldsymbol{z}$; in particular, it is not identically zero.

(c) Let $\boldsymbol{x}^*$ be obtained by taking the componentwise minimum of all (A, n)-super-vectors. By (b), $\boldsymbol{x}^*$ is an (A, n)-super-vector, and it is clearly the unique minimum such vector.

(d) Denote by $\boldsymbol{x}^i$ the value of $\boldsymbol{x}$ after the i-th iteration of the `while` loop in Figure 4, with $\boldsymbol{x}^0 = (1\,1\,\ldots\,1)^T$. We show by induction on i that $\boldsymbol{x}^i \leq \boldsymbol{x}^*$. The induction base $i = 0$

follows from (a). Now, suppose that $\boldsymbol{x}^i \leq \boldsymbol{x}^*$ for some i. Since A is nonnegative, we have

$$\frac{1}{n} A\boldsymbol{x}^i \leq \frac{1}{n} A\boldsymbol{x}^* \leq \boldsymbol{x}^* \, .$$

Therefore, $\boldsymbol{x}^{i+1} = \max\left\{ \left\lceil \frac{1}{n} A\boldsymbol{x}^i \right\rceil , \boldsymbol{x}^i \right\} \leq \boldsymbol{x}^*$, thereby establishing the induction step.

Next we verify that the algorithm halts. Observe that $\boldsymbol{x}^0 \leq \boldsymbol{x}^1 \leq \boldsymbol{x}^2 \leq \ldots \leq \boldsymbol{x}^*$ and that $\boldsymbol{x}^i$ are integer vectors. Now, if the algorithm did not halt, there had to be an index i for which $\boldsymbol{x}^{i+1} = \boldsymbol{x}^i$. However, that would imply $\left\lceil \frac{1}{n} A\boldsymbol{x}^i \right\rceil \leq \boldsymbol{x}^i$ (in which case $\boldsymbol{x}^i$ would in fact be a super-vector), so the algorithm had to halt in the i-th iteration.

We therefore conclude that the algorithm halts and returns an (A, n)-super-vector $\boldsymbol{x} \leq \boldsymbol{x}^*$; in fact, we must have $\boldsymbol{x} = \boldsymbol{x}^*$, since $\boldsymbol{x}^*$ is the unique minimum (A, n)-super-vector. □

Example 3 The adjacency matrix of the graph $G = G^7_{2,\infty}$ is given by

$$A_G = A^7_{G_{2,\infty}} = \begin{pmatrix} 3 & 2 & 4 \\ 4 & 3 & 6 \\ 6 & 4 & 9 \end{pmatrix} .$$

Now,

$$\begin{aligned}
\boldsymbol{x}^0 &= (1\ 1\ 1)^T \, , \\
\boldsymbol{x}^1 &= \max\left\{ \left\lceil \tfrac{1}{16} A_G (1\ 1\ 1)^T \right\rceil , (1\ 1\ 1)^T \right\} = \max\left\{ (1\ 1\ 2)^T , (1\ 1\ 1)^T \right\} = (1\ 1\ 2)^T \, , \\
\boldsymbol{x}^2 &= \max\left\{ \left\lceil \tfrac{1}{16} A_G (1\ 1\ 2)^T \right\rceil , (1\ 1\ 2)^T \right\} = \max\left\{ (1\ 2\ 2)^T , (1\ 1\ 2)^T \right\} = (1\ 2\ 2)^T \, ,
\end{aligned}$$

and $A_G \boldsymbol{x}^2 \leq 16 \boldsymbol{x}^2$. Hence, by Proposition 3(d), the vector $(1\ 2\ 2)^T$ is the unique minimum (A_G, n)-super-vector $\boldsymbol{x}^*$. □

5.3 Consistent splitting

As mentioned before, the state-splitting construction of a sliding-block compressible (S, n)-excoder $\mathcal{E}$ starts with a deterministic presentation G of S. Typically, S will be a q-th power of a given constraint S' and G will be the q-th power of a graph presentation of S'; the integer n will be 2^p and $\mathcal{E}$ will thus be a sliding-block compressible excoder for S' at rate $p : q$.

Given S, G, and n, we compute an (A_G, n)-super-vector $\boldsymbol{x}$ using the algorithm in Figure 4. The entry x_u in $\boldsymbol{x}$ will be referred to as the *weight of state* u. Using the vector $\boldsymbol{x}$, we will transform the graph G through state-splitting operations into a graph H such that $(1\ 1\ \ldots\ 1)^T$ is an (A_H, n)-super-vector (i.e., the weights are all reduced to 1). It is easy to see that this is equivalent to saying that H has out-degree at most n. An (S,n)-excoder $\mathcal{E}$ will then be obtained by assigning tags to the edges of H.

Next we discuss the role of the (A_G,n)-super-vector in more detail. For an edge e in a graph, denote by $\tau(e)$ the terminal state of e.

Given a graph G, a positive integer n, and an (A_G,n)-super-vector $\boldsymbol{x} = (x_u)_u$, an $\boldsymbol{x}$-*consistent partition* of G is defined by partitioning the set, E_u, of outgoing edges from each state u in G as in (5) such that

$$\sum_{e \in E_u^{(r)}} x_{\tau(e)} \le n x_u^{(r)} \qquad \text{for} \qquad r = 1, 2, \ldots, N = N(u) \ , \tag{6}$$

where $x_u^{(r)}$ are nonnegative integers and

$$\sum_{r=1}^{N(u)} x_u^{(r)} \ = \ x_u \ . \tag{7}$$

The state splitting based upon such a partition is called an $\boldsymbol{x}$-*consistent splitting.* The splitting is called *non-trivial* if at least one state u has at least two descendants $u^{(r)}, u^{(t)}$ such that both $x_u^{(r)}$ and $x_u^{(t)}$ are strictly positive.

Let G' denote the graph after splitting. It is easy to see that the (A_G,n)-super-vector $\boldsymbol{x}$ gives rise to an *induced* $(A_{G'},n)$-*super-vector* $\boldsymbol{x}' = (x'_u)_u$; namely, set $x'_{u^{(r)}} = x_u^{(r)}$. Note that (7) asserts that the weights of the descendants of a state u sum to the weight of u.

Example 4 Let S be presented by the graph $G = G^7_{2,\infty}$. For each state $u \in \{0,1,2\}$ in G, denote by $\mathcal{L}_u$ the set of labels of the outgoing edges from state u in G. Note that the graph G has memory 1, since the label of an edge determines the terminal state of that edge: edges whose labels end with '1' terminate in state 0, edges whose labels end with '10' terminate in state 1, and the remaining edges terminate in state 2. Hence, each set $\mathcal{L}_u$ completely describes the set, E_u, of outgoing edges from state u. We have

$$\begin{aligned}
\mathcal{L}_0 &= \{0000000, 0000001, 0000010, 0000100, 0001000, 0001001, 0010000, 0010001, 0010010\} \ ; \\
\mathcal{L}_1 &= \{0000000, 0000001, 0000010, 0000100, 0001000, 0001001, 0010000, 0010001, 0010010 \\
&\qquad 0100000, 0100001, 0100010, 0100100\} \ ; \\
\mathcal{L}_2 &= \{0000000, 0000001, 0000010, 0000100, 0001000, 0001001, 0010000, 0010001, 0010010, \\
&\qquad 0100000, 0100001, 0100010, 0100100, 1000000, 1000001, 1000010, 1000100, 1001000, 1001001\} \ .
\end{aligned}$$

We have shown in Example 3 that $\boldsymbol{x} = (1\ 2\ 2)^T$ is an $(A_G,16)$-super-vector. We next perform an $\boldsymbol{x}$-consistent splitting on G which will result in a graph H in which each state has weight 1. That is, up to tagging, the graph H will be an $(S,16)$-excoder, namely, a rate $4:7$ excoder for the $(2,\infty)$-RLL constraint.

Since state 0 has weight 1, it will not be split (or more precisely, we split it trivially into one state, namely itself). Since each of the states 1 and 2 has weight 2, we would like to split each into two states of weight 1. Define the *weight of an edge* to be the weight of its terminal state. Now, $A_G(1\ 2\ 2)^T = (15\ 22\ 32)^T$, indicating that the total weights of outgoing edges from states 0, 1, and 2 are 15, 22, and 32, respectively. If we can partition the sets of outgoing edges from state 1 and state 2, each into two subsets of edges of total weight at most 16, then it follows that the weights in the graph H obtained from the corresponding state splitting will all be 1, as desired. This indeed can be done as follows, where each partition element $E_u^{(r)}$ is represented by the respective label set $\mathcal{L}_u^{(r)}$ (labels that correspond to edges with weight 2 are underlined):

$$\begin{array}{lcl}
\mathcal{L}_0 & = & \{\underline{0000000}, 0000001, \underline{0000010}, \underline{0000100}, \underline{0001000}, 0001001, \underline{0010000}, 0010001, \underline{0010010}\} \ ; \\
\mathcal{L}_1^{(1)} & = & \{\underline{0000000}, 0000001, \underline{0000010}, \underline{0000100}, \underline{0001000}, 0001001, \underline{0010000}, 0010001, \underline{0010010}\} \ ; \\
\mathcal{L}_1^{(2)} & = & \{\underline{0100000}, 0100001, \underline{0100010}, \underline{0100100}\} \ ; \\
\mathcal{L}_2^{(1)} & = & \{\underline{0000000}, 0000001, \underline{0000010}, \underline{0000100}, \underline{0001000}, 0001001, \underline{0010000}, 0010001, \underline{0010010}, 1001001\} \ ; \\
\mathcal{L}_2^{(2)} & = & \{\underline{0100000}, 0100001, \underline{0100010}, \underline{0100100}, \underline{1000000}, 1000001, \underline{1000010}, \underline{1000100}, \underline{1001000}\} \ .
\end{array}$$

The reader can verify that the sets E_0, $E_1^{(1)}$, $E_1^{(2)}$, $E_2^{(1)}$, and $E_2^{(2)}$ have total weights 15, 15, 7, 16, and 16, respectively, as desired (in fact, the weights of $E_2^{(1)}$ and $E_2^{(2)}$ are forced to be 16).

The resulting split graph H will have five states, 0, $1^{(1)}$, $1^{(2)}$, $2^{(1)}$, and $2^{(2)}$, and the induced (A_H, n)-super-vector is $\boldsymbol{x}' = (1\ 1\ 1\ 1\ 1)^T$, implying that the row sums of the adjacency matrix

$$A_H = \begin{pmatrix} 3 & 2 & 2 & 4 & 4 \\ 3 & 2 & 2 & 4 & 4 \\ 1 & 1 & 1 & 2 & 2 \\ 4 & 2 & 2 & 4 & 4 \\ 2 & 2 & 2 & 5 & 5 \end{pmatrix}$$

are all at most 16. □

The following modification of Proposition 4.4 of [12] shows that in general there always is an $\boldsymbol{x}$-consistent splitting whenever we need one.

Proposition 4 *Let G be an irreducible graph which does not have out-degree at most n and let $\boldsymbol{x}$ be an (A_G, n)-super-vector. Then there is a non-trivial $\boldsymbol{x}$-consistent splitting of G.*

Proof. The proof is essentially contained in Proposition 5 of [10], but we outline it also here for the sake of completeness. By the assumption, some state u in G has out-degree greater than n. By the pigeon-hole principle, there is a subset E of E_u with at most n edges such that n divides $\sum_{e \in E} x_{\tau(e)}$. Partition E_u into two sets $E_u^{(1)} \cup E_u^{(2)}$ where

$$E_u^{(1)} = E \qquad \text{and} \qquad E_u^{(2)} = E_u \setminus E \ ,$$

and set

$$x_u^{(1)} = (1/n)\Big(\sum_{e \in E} x_{\tau(e)}\Big) \qquad \text{and} \qquad x_u^{(2)} = x_u - x_u^{(1)} \; .$$

It can be readily verified that the partition $E_u^{(1)} \cup E_u^{(2)}$ indeed implies a non-trivial $\boldsymbol{x}$-consistent splitting of state u. $\square$

Passage from the (A_G, n)-super-vector $\boldsymbol{x}$ to the induced $(A_{G'}, n)$-super-vector $\boldsymbol{x}'$ always preserves the super-vector sum and increases the number of states. Since a super-vector is always a positive integer vector, it follows that repeated applications of Proposition 4 beginning with G eventually yield a graph H with an (A_H, n)-super-vector $(1\,1\,\ldots\,1)^T$ and therefore with out-degree at most n. As mentioned earlier, if the original presentation G has finite memory m, then H will be (m, a)-definite on S. Finally, we assign tags to the edges of H, thereby obtaining an (m, a)-sliding-block compressible (S, n)-excoder $\mathcal{E}$. This establishes Proposition 2.

It may well be possible to merge states in H, resulting in a simpler excoder. Suppose that u and v are states in H such that every word that can be generated in H by paths starting at u can also be generated by paths starting at v. We can merge state u into state v by redirecting all incoming edges to u into v and then deleting state u with all its outgoing edges. This is the direct analogy of merging in constrained coding [12, Section 4.6.1].

Merging can add new words to those that are generated by H (and $\mathcal{E}$), and it may also give rise to new paths that present words of S. In particular, it may destroy definiteness on S. However, suppose that there are integers $\mathsf{m}' \geq \mathsf{m}$ and $\mathsf{a}' \geq \mathsf{a}$ such that the following holds: for every word $\boldsymbol{w} \in S_{\mathsf{m}'}$ that can be generated by a path in H that terminates in u and for every word $\boldsymbol{w}' \in S_{\mathsf{a}'+1}$ that can be generated in H from v but not from u, we have $\boldsymbol{w}\boldsymbol{w}' \notin S_{\mathsf{m}'+\mathsf{a}'+1}$. Under this condition, the merged graph is $(\mathsf{m}', \mathsf{a}')$-definite on S, since definiteness on S involves only words in S, which may be a proper subset of the constraint presented by the merged graph.

Example 5 Continuing the discussion in Example 4, we observe that $\mathcal{L}_1^{(2)} \subset \mathcal{L}_2^{(2)}$; furthermore, since G has memory 1, edges with the same label in $E_1^{(2)}$ and $E_2^{(2)}$ terminate in the same state of G. It follows that every word that can be generated in H from $1^{(2)}$ can also be generated from $2^{(2)}$.

Let $\boldsymbol{w}$ be a word that is generated by a path in H that terminates in state $1^{(2)}$. This word is also generated in G by a path that terminates in state 1 and, so, $\boldsymbol{w}$ ends with '10'. Let $\boldsymbol{w}'$ be a word that can be generated in H from $2^{(2)}$ but not from $1^{(2)}$. The word $\boldsymbol{w}'$ necessarily starts with '1' since

$$\mathcal{L}_2^{(2)} \setminus \mathcal{L}_1^{(2)} = \{1000000, 1000001, 1000010, 1000100, 1001000\} \; .$$

It follows that $\boldsymbol{w}\boldsymbol{w}'$ contains the sub-word '101'; as such, it violates the $(2, \infty)$-RLL constraint and, therefore, it does not belong to S. We conclude that the definiteness on S

will be preserved upon merging state $1^{(2)}$ into state $2^{(2)}$. Similarly, since $\mathcal{L}_0 = \mathcal{L}_1^{(1)} \subset \mathcal{L}_2^{(1)}$ and $\mathcal{L}_2^{(1)} \setminus \mathcal{L}_0 = \{1001001\}$, we see that states 0 and $1^{(1)}$ can be merged into state $2^{(1)}$ while preserving definiteness on S. Thus, the resulting merged graph, H', has only two states, $a \equiv 2^{(1)}$ and $b \equiv 2^{(2)}$, and it is $(1,1)$-definite on S. In fact, H' is $(0,1)$-definite on S: since $\mathcal{L}_2^{(1)} \cap \mathcal{L}_2^{(2)} = \emptyset$, the initial state of the edge that generates the current codeword is uniquely determined by that codeword.

Finally, we assign tags from $\{0,1\}^4$ to the edges of H' to obtain the $(0,1)$-sliding-block compressible excoder $\mathcal{E}_{2,\infty}$ whose transition table is shown in Table 2. In that table, the rows are indexed by the tags (4-frames) and the columns by the states of $\mathcal{E}_{2,\infty}$. Entry (s,u) in the table contains the label and terminal state of the outgoing edge from state u that is tagged by s. Observe that the tags have been assigned to the edges of $\mathcal{E}_{2,\infty}$ so that tags of edges that

	a	b
0000	0000000, a	0100000, a
0001	0000000, b	0100000, b
0010	0000010, a	0100100, a
0011	0000010, b	0100100, b
0100	0000100, a	1000000, a
0101	0000100, b	1000000, b
0110	0001000, a	1000100, a
0111	0001000, b	1000100, b
1000	0010000, a	1001000, a
1001	0010000, b	1001000, b
1010	0010010, a	0100001, a
1011	0010010, b	1000001, a
1100	0000001, a	0100010, a
1101	0001001, a	0100010, b
1110	0010001, a	1000010, a
1111	1001001, a	1000010, b

Table 2: Excoder $\mathcal{E}_{2,\infty}$.

have the same label (and initial state) will differ in only the last bit. Thus, whenever the compression of the current 4-frame depends on the upcoming 7-codeword, such dependency is limited only to determining the last bit of the 4-frame (see [15] for an application of a similar idea). $\square$

In Table 3, we list for each $p \leq 16$ the largest value of q which allows a block excoder at rate $p:q$ as well as the largest value of q which allows a $(0,1)$-sliding-block compressible excoder at rate $p:q$ for the $(2,\infty)$-RLL constraint. The values of q for the block excoder case can be obtained by (1), where the values of $|S_q|$ can be computed using the formulas in [5, Section 5.2] (it can be verified that the same values of q apply also to $(0,0)$-sliding-block compressible excoders). The values of q for the $(0,1)$-sliding-block compressible case

p	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
q (block)	1	3	4	6	8	10	11	13	15	17	19	21	22	24	26	28
q ((0, 1)-sliding block)	1	3	**5**	**7**	**9**	10	**12**	**14**	**16**	**18**	19	21	**23**	**25**	**27**	**29**

Table 3: Maximal values of q for existence of block excoders and $(0,1)$-sliding-block compressible excoders for the $(2,\infty)$-RLL constraint.

were obtained by actually computing $(A^q_{G_{2,\infty}}, 2^p)$-super-vectors and verifying that excoders can be obtained by one round of state splitting. In particular, the rate $4:7$ is attained by the excoder $\mathcal{E}_{2,\infty}$ in Example 5. It is worthwhile pointing out that a rate $5:9 = .5555\ldots$ is attainable by a $(0,1)$-sliding-block compressible excoder with ten states. This rate is above the capacity ($\approx .5515$) by less than 1%. The boldface numbers in Table 3 indicate values of q which are larger (by 1) than those attainable by block codes. The results in Table 3 remain unaffected if one were to replace the $(2,\infty)$-RLL constraint with the $(2,15)$-RLL constraint.

We also mention here the existence of the following two $(0,1)$-sliding-block compressible excoders that might be of practical interest: a rate $8:9$ excoder for the $(0,2)$-RLL constraint, and a rate $5:7$ excoder for the $(1,7)$-RLL constraint; there are no block excoders at such rates for those constraints.

5.4 Reduction of edge effect in error propagation

Recall that in the analysis of Section 4, we made the conservative assumption that a p-frame is wholly corrupted even if only one bit in that p-frame is in error. Yet, as we know for block codes, and as we have demonstrated in Example 5 for sliding-block compressible excoders, special care in the assignment of tags can reduce the dependency of certain bits in p-frames on certain channel bits, thereby reducing the effect of error. Specifically, when using the excoder $\mathcal{E}_{2,\infty}$ of Example 5, a burst of $L = 40$ channel bits, while affecting up to eight 4-frames, can corrupt only up to 29 bits (and not 32 bits) in those frames. This, in turn, allows us to modify Table 1 to produce Table 4 (the modified entries are indicated by boldface numbers). It follows from Table 4 that for the range $841 \le \kappa \le 7,778$, the actual redundancy that will be required is strictly less than dictated by Table 1. In particular, for $3,970 \le \kappa \le 7,778$, the savings amount to reducing the redundancy from 80 to 70 bits.

Additional savings can be obtained by using an excoder with more states, as we demonstrate in the next example.

Example 6 Looking closely at Table 2, one can see that the compression of the last bit of the current 4-frame depends on the first, second, fourth, and seventh bits of the upcoming 7-codeword. The dependency on the seventh bit has a slight disadvantage in case of short

B	$D(40,B)$	ν	r	κ
4	8	544	64	840
5	**7**	$\mathbf{1,155}$	**70**	$\mathbf{1,898}$
6	6	$2,340$	(72)	$3,969$
7	**5**	$\mathbf{4,515}$	**70**	$\mathbf{7,778}$
8	5	$10,280$	80	$17,850$

Table 4: Modified Table 1.

bursts—in particular isolated *bit-shift errors*, where an occurrence of '1' in a constrained sequence is shifted by one position, thereby resulting in two adjacent erroneous bits in the constrained sequence. If those two bits cross the boundary of adjacent 7-codewords, the error may propagate through the compression to up to 9 bits in three 4-frames.

By allowing more states, we are able to present another rate 4 : 7 excoder for the $(2,\infty)$-RLL constraint, $\mathcal{E}'_{2,\infty}$, where the compression of the last bit of the current 4-frame depends on the first, second, fourth, and fifth bits of the upcoming 7-codeword (whereas the other bits of the 4-frame depend only on the current 7-codeword). Here, one bit-shift error in the constrained sequence may affect only two 4-frames. The excoder $\mathcal{E}'_{2,\infty}$ is obtained through a different splitting of state 2 in G (into states $2^{(1')}$ and $2^{(2')}$), resulting in four states, $\alpha \equiv 0 = 1^{(1)}$, $\beta \equiv 1^{(2)}$, $\gamma \equiv 2^{(1')}$, and $\delta \equiv 2^{(2')}$, with a transition table as shown in Table 5. The excoder $\mathcal{E}'_{2,\infty}$ is $(1,1)$-definite (but not $(0,1)$-definite) on the constraint;

	α	β	γ	δ
0000	0000000, γ	0100010, α	0000000, γ	0100010, α
0001	0000000, δ	0100010, β	0000000, δ	0100010, β
0010	0000010, α	0100000, γ	0000010, α	0100000, γ
0011	0000010, β	0100000, δ	0000010, β	0100000, δ
0100	0000100, γ	–	0000100, γ	1000010, α
0101	0000100, δ	–	0000100, δ	1000010, β
0110	0010000, γ	–	0010000, γ	1000000, γ
0111	0010000, δ	–	0010000, δ	1000000, δ
1000	0010010, α	–	0010010, α	1001001, α
1001	0010010, β	0100001, α	0010010, β	0100001, α
1010	0001000, γ	0100100, γ	0100100, γ	0001000, γ
1011	0001000, δ	0100100, δ	0100100, δ	0001000, δ
1100	–	–	1000100, γ	1000001, α
1101	0001001, α	–	1000100, δ	0001001, α
1111	0000001, α	–	0000001, α	1001000, γ
1111	0010001, α	–	0010001, α	1001000, δ

Table 5: Excoder $\mathcal{E}'_{2,\infty}$.

still, the particular tagging of the edges makes it $(0,1)$-sliding-block compressible. Note that states α and β have less than 16 outgoing edges and, so, certain elements of $\{0,1\}^4$ do not tag those edges. $\square$

References

[1] R. Adler, D. Coppersmith, M. Hassner, *Algorithms for sliding block codes — an application of symbolic dynamics to information theory, IEEE Trans. Inform. Theory,* 29 (1983), 5–22.

[2] E. Arikan, *An implementation of Elias coding for input-restricted channel, IEEE Trans. Inform. Theory,* 36 (1990), 162–165.

[3] W.G. Bliss, *Circuitry for performing error correction calculations on baseband encoded data to eliminate error propagation, IBM Tech. Discl. Bull.,* 23 (1981), 4633–4634.

[4] J.L. Fan, A.R. Calderbank, *A modified concatenated coding scheme, with applications to magnetic data storage, IEEE Trans. Inform. Theory,* 44 (1998), 1565–1574.

[5] K.A.S. Immink, *Coding Techniques for Digital Recorders,* Prentice Hall, New York, 1991.

[6] K.A.S. Immink, *A practical method for approaching the channel capacity of constrained channels, IEEE Trans. Inform. Theory,* 43 (1997), 1389–1399.

[7] K.J. Kerpez, *Runlength codes from source codes, IEEE Trans. Inform. Theory,* 37 (1991), 682–687.

[8] M. Mansuripur, *Enumerative modulation coding with arbitrary constraints and post-modulation error correction coding and data storage systems, Proc. SPIE,* 1499 (1991), 72–86.

[9] F.J. MacWilliams, N.J.A. Sloane, *The Theory of Error-Correcting Codes,* North-Holland, Amsterdam, 1977.

[10] B.H. Marcus, *Sofic systems and encoding data, IEEE Trans. Inform. Theory,* 31 (1985), 366–377.

[11] G.N.N. Martin, G.G. Langdon, S.J.P. Todd, *Arithmetic codes for constrained channels, IBM J. Res. Develop.,* 27 (1983), 94–106.

[12] B.H. Marcus, R.M. Roth, P.H. Siegel, *Constrained systems and coding for recording channels,* in *Handbook of Coding Theory,* V.S. Pless and W.C. Huffman (Eds.), Elsevier, Amsterdam, 1998, 1635–1764.

[13] B.H. Marcus, P.H. Siegel, J.K. Wolf, *Finite-state modulation codes for data storage, IEEE J. Sel. Areas Comm.,* 10 (1992), 5–37.

[14] W.W. Peterson, E.J. Weldon, Jr., *Error-Correcting Codes,* Second Edition, MIT Press, Cambridge, Massachusetts, 1972.

[15] R.M. Roth, *On runlength-limited coding with DC control,* submitted to *IEEE Trans. Communications.*

[16] S.J.P. Todd, G.N.N. Martin, G.G. Langdon, *A general fixed rate arithmetic coding method for constrained channels, IBM J. Res. Develop.,* 27 (1983), 107–115.