Almost-certainly Runlength-limiting Codes

David J. C. MacKay

Department of Physics, University of Cambridge
Cavendish Laboratory, Madingley Road,
Cambridge, CB3 0HE, United Kingdom.
mackay@mrao.cam.ac.uk
http://wol.ra.phy.cam.ac.uk/mackay/

Abstract. Standard runlength-limiting codes – nonlinear codes defined by trellises – have the disadvantage that they disconnect the outer error-correcting code from the bit-by-bit likelihoods that come out of the channel. I present two methods for creating transmissions that, with probability extremely close to 1, both are runlength-limited and are codewords of an outer linear error-correcting code (or are within a very small Hamming distance of a codeword). The cost of these runlength-limiting methods, in terms of loss of rate, is significantly smaller than that of standard runlength-limiting codes. The methods can be used with any linear outer code; low-density parity-check codes are discussed as an example.
The cost of the method, in terms of additional redundancy, is very small: a reduction in rate of less than 1% is sufficient for a code with blocklength 4376 bits and maximum runlength 14.

This paper concerns noisy binary channels that are also constrained channels, having maximum runlength limits: the maximum number of consecutive `1`s and/or `0`s is constrained to be r. The methods discussed can also be applied to channels for which certain other long sequences are forbidden, but they are not applicable to channels with minimum runlength constraints such as maximum transition-run constraints.

I have in mind maximum runlengths such as $r = 7$, 15, or 21. Such constraints have a very small effect on the capacity of the channel. (The capacity of a noiseless binary channel with maximum runlength r is about $1 - 2^{-r}$.)

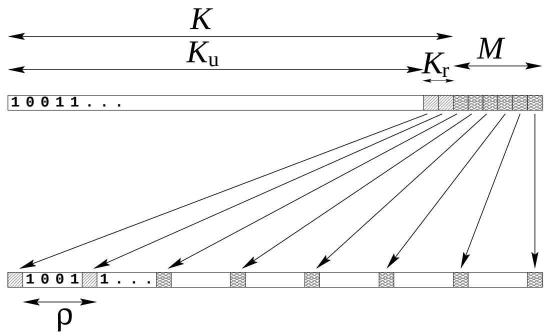
There are two simple ways to enforce runlength constraints. The first is to use a nonlinear code to map, say, 15 data bits to 16 transmitted bits [8]. The second is to use a linear code that is guaranteed to enforce the runlength constraints. The disadvantage of the first method is that it separates the outer error-correcting code from the channel: soft likelihood information may be available at the channel output, but once this information has passed through the inner decoder, its utility is degraded. The loss of bit-by-bit likelihood information can decrease the performance of a code by about 2 dB [3]. The second method may be feasible, especially if low-density parity-check codes are used, since they are built out of simple parity constraints, but it only gives a runlength limit r smaller than 16 if the outer code's rate is smaller than the rates that are conventionally required for magnetic recording (0.9 or so) [7].

I now present two simple ideas for getting the best of both worlds. The methods presented involve only a small loss in communication rate, *and* they are compatible with the use of linear error-correcting codes. The methods do not give an absolute guarantee of reliable runlength-limited communication; rather, as in a proof of Shannon's noisy channel coding theorem, we will be able to make a statement like 'with probability $1-10^{-20}$, this method will communicate reliably and satisfy the $r = 15$ runlength constraint'. This philosophy marries nicely with modern developments in magnetic recording, such as (a) the use of low-density parity-check codes, which come without cast-iron guarantees but work very well empirically [7]; and (b) the idea of digital fountain codes [1], which can store a large file on disc by writing thousands of packets on the disc, each packet being a random function of the original file, and the original file being recoverable from (almost) any sufficiently large subset of the packets – in which case occasional packet loss is unimportant. The ideas presented here are similar to, but different from, those presented by Immink [4–6], Deng and Herro [2], and Markarian *et al.* [9].

1 Linear method

The first idea is a method for producing a codeword of the linear outer code that, with high probability, satisfies the runlength-limiting constraints.

We assume that a good outer (N, K) linear error-correcting code has been chosen, and that it is a systematic code. We divide the K source bits into $K_{\rm u}$ user bits and $K_{\rm r} > \log_2 M$ additional special source bits that we will set so as to satisfy the runlength constraints. The code has $M = N - K$ parity bits. We choose the special source bits such that all M of the parity bits are influenced by at least one of the $K_{\rm r}$ special bits. When transmitting the codeword we order the bits as shown below, such that the $K_{\rm r} + M$ special bits and parity bits appear uniformly throughout the block. We call these $K_{\rm r} + M$ bits the pad bits.



We modify the code by adding an offset $\mathbf{o}$ to all codewords. The random vector $\mathbf{o}$ is known to the sender and receiver and satisfies the runlength constraints. As we will see later, it may be useful for $\mathbf{o}$ to change from block to block; for example, in magnetic recording, it might depend on a seed derived from the sector number.

The method we will describe is appropriate for preventing runs of length greater than r for any r greater than or equal to ρ, the distance between pad bits. Two examples of the code parameters are given in table 1. R is the overall code rate from the user's perspective. We intend the number of special source bits

Table 1. Some feasible parameter settings.

K_{u}	4096	4096
K_{r}	20	24
M	296	432
N	4412	4552
$R = K_{\mathrm{u}}/N$	0.928	0.9
$\rho = N/(M + K_{\mathrm{r}})$	14	10

K_{r} to be very small compared with the blocklength. For comparison, a standard rate 15/16 runlength-limiting inner code would correspond to $K_{\mathrm{r}} \simeq 256$ bits. If instead we use $K_{\mathrm{r}} = 24$ then we are using one tenth the number of bits to achieve the runlength constraint.

The idea of the linear runlength-limiting method is that, once the user bits have been set, there are very likely to be codewords that satisfy the runlength constraints among the $2^{K_{\mathrm{r}}}$ codewords corresponding to the $2^{K_{\mathrm{r}}}$ possible settings of the special source bits.

Encoding method

We write the user data into the K_{u} bits and note which, if any, of the $K_{\mathrm{r}} + M$ pad bits are forced by the runlength constraints to take on a particular state. [If the maximum runlength, r, is greater than the spacing between pad bits, ρ, then there may be cases where we are free to choose which of two adjacent pad bits to force. We neglect this freedom in the following calculations, noting that this means the probability of error will be overestimated.]

For a given user block, the probability, averaging over offset vectors $\mathbf{o}$, that a particular pad bit is forced to take state `0` is the probability that it lies in or adjacent to a run of r `1`s, which is approximately

$$\beta \equiv r 2^{-r}. \tag{1}$$

The probability that a particular pad bit is forced to take state `1` is also β. The expected number of forced pad bits in a block is thus $2\beta(K_{\mathrm{r}} + M)$. Table 2 shows that, for $r \geq 14$, it will be rare that more than one or two pad bits are forced.

Having identified the forced bits, we use linear algebra to find a setting of the K_{r} special bits such that the corresponding codeword satisfies the forced bits, or, in the rare cases where no such codeword exists, to find a codeword that violates the smallest number of them.

Table 2. Examples of the expected number of forced pad bits, $2\beta(M + K_r)$.

r	14	16	20
β	8×10^{-4}	2×10^{-4}	2×10^{-5}
$K_r + M$	316	316	316
$2\beta(M + K_r)$	0.5	0.15	0.01

1.1 Probability of failure

The probability that this scheme will fail to find a satisfactory codeword depends on the details of the outer code. We first make an estimate for the case of a low-density parity-check code; later, we will confirm this estimate by numerical experiments on actual codes.

Let the columns of the parity-check matrix corresponding to the $K_r + M$ pad bits form a submatrix $\mathbf{F}$. In the case of a regular low-density parity-check code, this binary matrix will be sparse, having column weight 4, say, or about 8 in the case of a code originally defined over $GF(16)$. If the code is an irregular low-density parity-check code, these columns might be chosen to be the higher weight columns; we will see that this would reduce the probability of error.

Consider one row of $\mathbf{F}$ of weight w. If the w corresponding pad bits are *all* constrained, then there is a probability of 1/2 that the parity constraint corresponding to this row will be violated. In this situation, we can make a codeword that violates one of the w runlength constraints and satisfies the others. The probability of this event happening is $(2\beta)^w$. For every row of $\mathbf{F}$, indeed *for every non-zero codeword of the dual code* corresponding to $\mathbf{F}$, there is a similar event to worry about. The expected number of runlength constraints still violated by the best available codeword is thus roughly

$$\sum_w \frac{1}{2} A_{\mathbf{F}}(w)(2\beta)^w, \tag{2}$$

where $A_{\mathbf{F}}(w)$ is the weight enumerator function of the code whose generator matrix is $\mathbf{F}$. For small β, this expectation is dominated by the low-weight words of $\mathbf{F}$, so, if there are M words of lowest weight $w_{\min}$, the expected number of violations is roughly

$$\frac{1}{2} M(2\beta)^{w_{\min}} \tag{3}$$

Table 3 shows this expected number for $w_{\min} = 4$ and 8.

For example, assuming $w_{\min} = 8$ (which requires a matrix $\mathbf{F}$ whose columns have weight 8 or greater), for a maximum runlength r of 14 or more, we can get the probability of failure of this method below 10^{-20}.

What the above analysis has not pinned down is the relationship between the column weight of $\mathbf{F}$ and $w_{\min}$. We now address this issue, assuming $\mathbf{F}$ is a low-density parity check matrix. If $\mathbf{F}$ has a row of weight w, then the dual code has a word of weight w. Any linear combination of rows also gives a dual

Table 3. The expected number of violations for $w_{\min} = 4$ and 8.

r	14	16	20
M	296	296	296
β	8×10^{-4}	2×10^{-4}	2×10^{-5}
$\frac{1}{2}M(2\beta)^4$	1×10^{-9}	8×10^{-12}	3×10^{-16}
$\frac{1}{2}M(2\beta)^8$	1×10^{-20}	5×10^{-25}	7×10^{-34}

codeword. Is it likely that the dual code has words of lower weight than the already sparse rows that make up the parity check matrix? It would be nice to know it is not likely, because then we could approximate the expected number of violations (2) by

$$\frac{1}{2}\sum_{w} g(w)(2\beta)^{w}, \tag{4}$$

where $g(w)$ is the number of rows of $\mathbf{F}$ that have weight w. However, as $\mathbf{F}$ becomes close to square, it becomes certain that linear combinations of those low-weight rows will be able to make even lower weight dual words. The approximation (4) would then be a severe underestimate.

We now test these ideas empirically.

1.2 Explicit calculation of the probability of conflict

I took a regular low-density parity-check code with blocklength $N = 4376$ bits and $M = 282$ (the true number of independent rows in the parity check matrix was 281). The column weight was $j = 4$. In four experiments I allocated $K_{\mathrm{r}} = 11, 21, 31, 41$ of the source bits to be special bits and found experimentally the conditional probability, given w, that a randomly selected set of w pad bits constrained by the runlength constraints would conflict with the code constraints.

[Method: given w randomly chosen pad bits, Gaussian elimination was attempted to put the generator matrix into systematic form with respect to the chosen bits. This was repeated for millions of choices of the pad bits, and the probability of failure of Gaussian elimination was estimated from the results. The actual probability of failure will be smaller than this quantity by a factor between 1 and 2, because it is possible, even though the pad bits are not independent, that the runlength constraints will be compatible with the code constraints.]

Under a union-type approximation (like (4)) that only counts the dual codewords that are rows of $\mathbf{F}$, we would predict this conditional probability to be

$$P(\text{conflict}|w) \simeq \sum_{w'} g(w') \frac{\binom{N-w'}{w-w'}}{\binom{N}{w}}. \tag{5}$$

The empirically-determined conditional probability of failure, as a function of the weight w of the constraint, is shown in figure 1, along with the approximation (5), shown by the lines without datapoints.

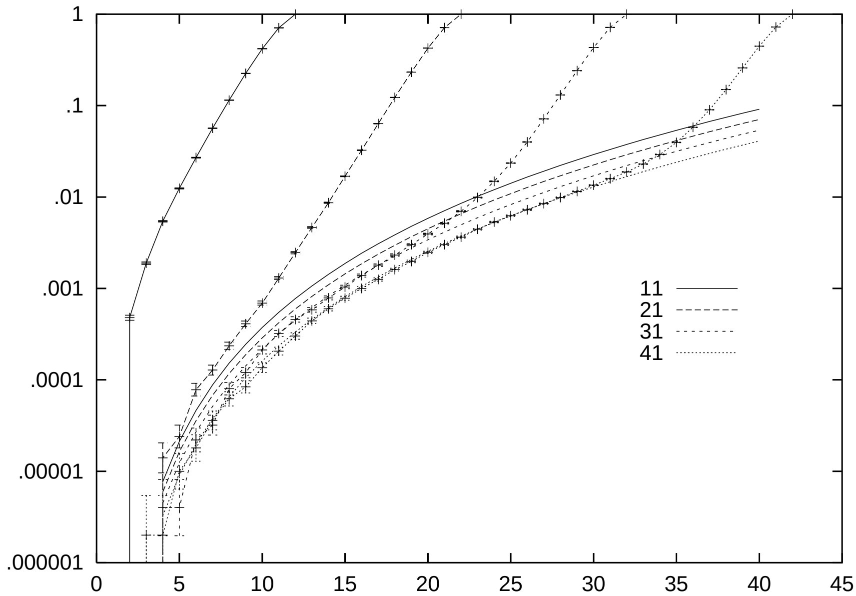


Fig. 1. Probability of failure of the linear runlength-limiting method as a function of number of constrained pad bits, w. The four curves with points and error bars show empirical results for $K_{\mathrm{r}} = 11$, 21, 31, and 41. The four lines show the approximation (5). All systems derived from the same regular low-density parity-check code with $M = 281$ constraints and column weight $j = 4$. The runlength limits for these four cases are $r = N/(K_{\mathrm{r}}+M) = 15$, 14.5, 14, and 13.6.

It can be seen that for $K_{\mathrm{r}} = 11$ and 21, the approximation is an underestimate, but for $K_{\mathrm{r}} = 31$ and 41, it gives a snug fit in the important area (*i.e.*, low w). From the empirical results we can also deduce the probability of failure, which is

$$P(\text{conflict}) = \sum_{w} P(w) P(\text{conflict}|w) \tag{6}$$

$$= \binom{N}{w} (2\beta)^{w} (1-2\beta)^{N-w} P(\text{conflict}|w). \tag{7}$$

Plugging in $\beta = 8 \times 10^{-4}$ (corresponding to constrained length $r = 14$), we find that for $K_{\mathrm{r}} = 21$, 31, and 41, the probability of a conflict is about 10^{-15}. We will discuss how to cope with these rare failures below.

1.3 Further experiments

I also explored the dependence on column weight by making three codes with identical parameters N, M, $K_{\rm r}$, but different column weights: $j = 3$, $j = 4$, and $j \simeq M/2$ (a random code). Figure 2 shows that for $j = 4$, the failure probability, at small w, is quite close to that of a random code.

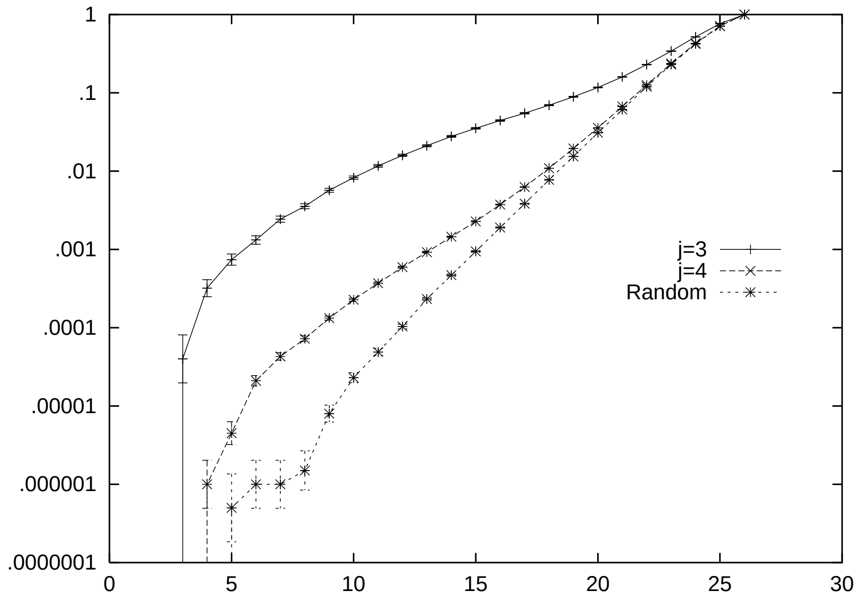


Fig. 2. Probability of failure of the linear runlength-limiting method for various column weights. The code parameters were $K_{\rm r} = 25$, $M = 220$.

1.4 How to make the probability of failure even smaller

We showed above that our runlength-limiting method can have probability of failure about 10^{-15}. What if this probability of failure is too large? And what should be done in the event of a failure? I can suggest two simple options. First, in discdrive applications, if the offset vector $\mathbf{o}$ is a random function of the sector number where the block is being written, we could have an emergency strategy: when the optimally encoded block has runlength violations, leave a pointer in the current sector and write the file in another sector, where the coset $\mathbf{o}$ is different. This strategy would incur an overhead cost at write-time on the rare occasions where the writer has to rearrange the blocks on the disc.

The second option is even simpler, as described in the following section.

2 Nonlinear method

If the linear method fails to satisfy all the runlength constraints, a really dumb option is to modify the corresponding pad bits so that the runlength constraints *are* satisfied, and transmit the modified word, which will no longer be a codeword of the outer error-correcting code. *As long as the number of flipped bits is not too great, the decoder of the outer code will be able to correct these errors.* The average probability of a bit's being flipped is raised by a very small amount compared with a typical noise level of 10^{-4}. The probability distribution of the number of flipped bits depends on the details of the code, but for a low-density parity-check code whose graph has good girth properties, we'd expect the probability of t flips to scale roughly as

$$p_0 p_1^{t-1}, \tag{8}$$

where $p_0 = \frac{1}{2}(2\beta)^{w_{\min}}$ and $p_1 = \frac{1}{2}(2\beta)^{w_{\min}-1}$, and no worse than

$$\binom{K_{\mathrm{r}} + M}{t} p_0 p_1^{t-1}, \tag{9}$$

which is roughly

$$\frac{(K_{\mathrm{r}} + M)^t}{t!}(2\beta)^{t(w_{\min}-1)} \tag{10}$$

For $w_{\min} = 4$, $K_{\mathrm{r}} + M = 316$, and $t = 6$, for example, the probability of t errors would be roughly as shown in table 4. Thus as long as the outer code is

Table 4.

r	14	16	20
$\frac{(K_{\mathrm{r}}+M)^t}{t!}(2\beta)^{6\times 3}$	2×10^{-44}	4×10^{-54}	5×10^{-74}

capable of correcting substantially more than 6 errors, the probability of failed transmission using this scheme is very low indeed.

2.1 Use of the nonlinear method alone

For some applications, the dumb nonlinear scheme by itself might suffice. At least in the case of a low-density parity-check code, it is simple to modify the decoder to take into account the fact that the pad bits are slightly less reliable than the user bits. [We could even include in the belief propagation decoding algorithm the knowledge that the pad bit is least reliable when it sits in the middle of run.]

Let the pad bits be the parity bits, *i.e.*, let $K_{\rm r} = 0$, use a random offset vector $\mathbf{o}$, and flip whichever pad bits need to be flipped to satisfy the runlength constraints. The probability of a pad bit's being flipped is β, which was given in table 2. If the ambient noise level is a bit-flip probability of 10^{-3}, then for runlength constraints r greater than or equal to 16, the increase in noise level for the pad bits (from 10^{-3} to $10^{-3} + \beta$) is small enough that the average effect on performance will be small. For a t-error-correcting code, this dumb method would suffer an absolutely uncorrectable error (*i.e.*, one that cannot be corrected at any noise level) with probability about $M^{(t+1)}\beta^{(t+1)}/(t+1)!$. For $r = 16$ and $M = 316$, this probability is shown in table 5.

Table 5. Probability of an absolutely uncorrectable error for the nonlinear method and a t-error-correcting code with $r = 16$ and $M = 316$.

t	5	9	19
$\frac{M^{(t+1)}\beta^{(t+1)}}{(t+1)!}$	2×10^{-10}	2×10^{-18}	2×10^{-41}

Thus, the dumb method appears to be feasible, in that it can deliver a failure probability smaller than 10^{-15} with a t=9-error-correcting code. If it is coupled with the trick of having the offset $\mathbf{o}$ vary from sector to sector, then it appears to offer a cheap and watertight runlength limiting method, even with a weaker outer code: on the rare occasions when more than, say, 5 bits need to be flipped, we move the data to another sector; this emergency procedure would be used of order once in every billion writes. The only cost of this method is a slight increase in the effective noise level at the decoder.

3 Discussion

In an actual implementation it would be a good idea to compute the weight enumerator function of the dual code defined by $\mathbf{F}$ and ensure that it has the largest possible minimum distance.

The ideas in this paper can be glued together in several ways.

- Special bits: various values of $K_{\rm r}$ can be used, including $K_{\rm r} = 0$ (*i.e.*, use the nonlinear method alone). The experiments suggest that increasing beyond $K_{\rm r} = 20$ or 30 gives negligible decrease in the probability of conflict.
- Nonlinear bitflipping. This feature could be on or off.
- Variable offset vector $\mathbf{o}$. If the offset vector can be pseudorandomly altered, it is very easy to cope with rare cases where either of the above methods fails.

If the variable offset vector is available, then either of the two ideas – the linear method or the nonlinear method – should work fine by itself. Otherwise, a belt-and-braces approach may be best, using the linear and nonlinear methods together.

Acknowledgements

This work was supported by the Gatsby Foundation and by a partnership award from IBM Zürich research laboratory. I thank Brian Marcus, Steven McLaughlin, Paul Siegel, Jack Wolf, Andreas Loeliger, Kees Immink, and Bane Vasic for helpful discussions, and Evangelos Eleftheriou of IBM Zürich for inviting me to the 1999 IBM Workshop on Magnetic Recording.

References

1. John Byers, Michael Luby, Michael Mitzenmacher, and Ashu Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM '98, September 2-4, 1998*, 1998.
2. R. H. Deng and M. A. Herro. DC-free coset codes. *IEEE Trans. Inf. Th.*, 34:786–792, 1988.
3. R. G. Gallager. *Low Density Parity Check Codes.* Number 21 in Research monograph series. MIT Press, Cambridge, Mass., 1963.
4. K. A. S. Immink. Constructions of almost block-decodable runlength-limited codes. *IEEE Transactions on Information Theory*, 41(1), January 1995.
5. K. A. S. Immink. A practical method for approaching the channel capacity of constrained channels. *IEEE Trans. Inform. Theory*, 43(5):1389–1399, Sept 1997.
6. K. A. S. Immink. Weakly constrained codes. *Electronics Letters*, 33(23), Nov. 1997.
7. D. J. C. MacKay and M. C. Davey. Evaluation of Gallager codes for short block length and high rate applications. In B. Marcus and J. Rosenthal, editors, *Codes, Systems and Graphical Models*, volume 123 of *IMA Volumes in Mathematics and its Applications*, pages 113–130. Springer-Verlag, New York, 2000.
8. B. H. Marcus, P. H. Siegel, and J. K. Wolf. Finite-state modulation codes for data storage. *IEEE Journal on Selected Areas in Communication*, 10(1):5–38, January 1992.
9. G. S. Markarian, M. Naderi, B. Honary, A. Popplewell, and J. J. O'Reilly. Maximum likelihood decoding of RLL-FEC array codes on partial response channels. *Electronics Letters*, 29(16):1406–1408, 1993.