Sparse Data Blocks and Multi-User Channels

Edward A. Ratzer
Cavendish Laboratory, Madingley Road, Cambridge CB3 0HE, UK
Phone: +44 (0) 1233 337238   Email: ear23@cam.ac.uk

October 25, 2002

Abstract

The creation of sparse data blocks of fixed size is studied and new algorithm introduced based on constant weight codes. The use of sparse data blocks on multiuser noisy channels (with noise models similar to CDMA and optical WDM) is studied.

1 Introduction

Sparse data (consisting of independent identically distributed (i.i.d.) bits with $p_1 \equiv \Pr(\text{digit} = 1) \neq 1/2$) is frequently used in information theory. Applications in communication theory include MN codes [4] and watermark codes [2].

This paper addresses the problem of mapping dense data to sparse data blocks of fixed size. This is often solved for small block sizes by using a look-up table [2]. An algorithm is presented in [4] for larger blocks and is studied further below. A new, more efficient, algorithm is then introduced.

We then present a new application for sparse data blocks in conjunction with linear error-correcting codes for multi-user channels.

2 An arithmetic coding based sparsifier

An algorithm for the generation of sparse blocks based on arithmetic coding [5] is presented in [4]. The algorithm uses standard arithmetic coding but reverses the usual role of compression and decompression. The probabilistic model used is:

$$\Pr(s_i = 1 | s_1, \ldots, s_{i-1}) = \Pr(s_i = 1) = p_1 \tag{1}$$

where s_i is the ith bit of the sparse data.

The algorithm is presented without details of initialization or termination. For initialization one can sets the standard arithmetic coding start conditions. The algorithm can be used without termination if there is an infinite stream of data being transmitted – the output sparse data is then split into blocks of the correct size. However if a single bit is corrupted it will corrupt all following bits in all blocks.

This property is not desirable in a communication theory context – to avoid this we need each block to be created independently. Different weight blocks have different probabilities of occurring, hence for fixed size sparse blocks the input sequence length is variable. We therefore need to

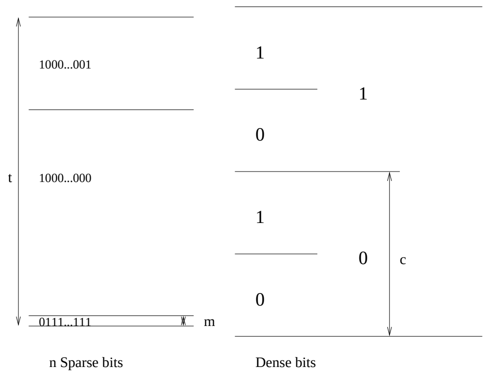


Figure 1: An example of a worst case for the arithmetic coding based sparsifier. If the next dense bit is `0` there is a ratio between the sparse interval size to dense interval size that varies with block size.

ensure that the termination has the property of forming a prefix code in the dense data. To do this we need to check for the existence of a valid sparse division as if the input digit is `0` and `1` (to maintain the prefix code tree structure) before reading in each binary digit. If there is not such a division for both `0` and `1` we stop reading for that block and transmit the sequence corresponding to a valid interval.

Arithmetic coding algorithms are efficient for compressing data as, with a suitable probabilistic model, the overhead is at worst 2 bits over the entire compression. However this constant, and small, inefficiency does not carry over to the sparsifier case above. Worst cases can be found where the information loss (δI) scales with the block size. A possible scenario is shown in figure 1 which gives the following information loss:

$$\begin{aligned}
t &= p_1^2 p_0^{n-2} + p_1 p_0^{n-1} + p_0 p_1^{n-1} && (2)\\
c &= t/2 \text{ for worst case} && (3)\\
m &= p_0 p_1^{n-1} && (4)\\
\delta I &= \log_2(c) - \log_2(m) && (5)\\
&= \log_2(3) - 1 + \frac{4(2n-5)}{3\ln(2)}(1/2 - p_1) + \\
&\qquad \mathcal{O}\left((1/2 - p_1)^2\right) && (6)
\end{aligned}$$

One might hope that these cases are sufficiently rare to be unimportant. However a simulation of the algorithm shows that this is not the case, in figure 2 it can be seen that the inefficiency is significantly worse than a constant.

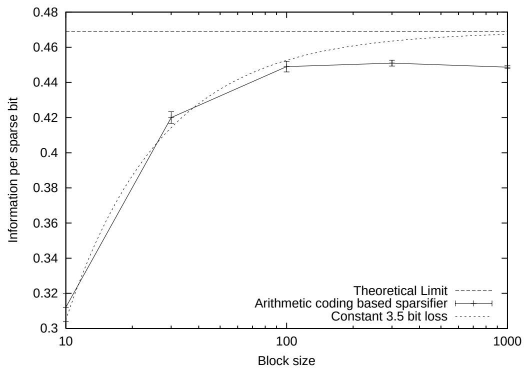


Figure 2: The efficiency of the arithmetic coding based sparsifier. The efficiency is plotted as a function of block size, N, for $p_1 = 0.1$. For comparison purposes the efficiency of a hypothetical scheme which has a constant 3.5 bit loss is shown as well.

3 The new sparsifier

A block generated from independent sparse bits is indistinguishable from a block generated by a fixed weight block generator, if the fixed weight (w) is chosen from the correct binomial distribution:

$$\begin{aligned} \Pr(\text{block}|p_1) &= \Pr(\text{block}|w, p_1)\Pr(w|p_1) && (7)\\ &= \Pr(\text{block}|w)\Pr(w|p_1) && (8)\\ &= \Pr(\text{block}|w)p_1^w(1-p_1)^{N-w} && (9) \end{aligned}$$

We can therefore split the task into two steps – generating blocks of a particular weight and encoding that weight.

The first task is a well studied field. M-out-of-N codes (also called constant weight codes) satisfy the constraint that for each codeword exactly M bits are `1` and the other $N-M$ bits are `0`. They are widely used in various applications as they can at least detect all unidirectional errors [1]. Applications include VLSI circuits and memories and optical channels [8]. Efficient create of both small and large blocks has been investigated [8, 6].

In [6] an algorithm based on a technique similar to arithmetic coding is presented. It can create an M-out-of-N code with at worst a one bit inefficiency. The idea is to use the same type of arithmetic coding algorithm as used before, but instead use it to generate constant weight codewords. The probabilistic model now becomes dependent on the history.

$$\Pr(s_i = 1|s_1, \ldots, s_{i-1}) = \frac{M - \text{weight}(s_1, \ldots, s_{i-1})}{N - i + 1} \qquad (10)$$

This probabilistic model has the effect that all blocks of weight M are equiprobable. Therefore we just need to choose a point on the division rather than the entire division to fairly represent it. This is illustrated in figure 3.

We now just need to generate an encoder for weights. This needs to be a prefix code to maintain unique codability and decodability. As an example, some of a prefix code generated using Huffman's algorithm [3] is shown in the middle column of Table 1. One can immediately see a problem that this algorithm has – the very low probability sequences will occur more often than expected if given a random stream of bits. For this paper we will disallow weights with low probability. This is illustrated in the last column of Table 1. The truncation was chosen to be the largest symmetric interval such that the error in the length was ≤ 1.

To create a sparse block from a $p_1 = 0.5$ i.i.d. data stream we read off a Huffman codeword to set the weight and then read off one codeword of the corresponding fixed weight sparsifier. The efficiency of this algorithm is illustrated in figure 4, assuming the maximum 1 bit loss of the arithmetic coding stage (so the effect of the Huffman coding stage can be more easily seen). In figure 5, the actual sparsification achieved is shown for example parameter values.

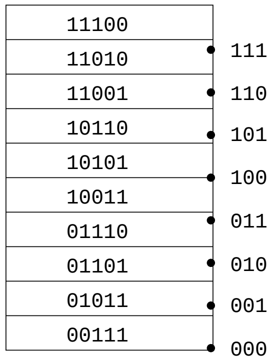


Figure 3: An example of the 3-out-of-5 constant weight code (after [6]). The inefficiency comes from the sequences `10011` and `11100` not being used.

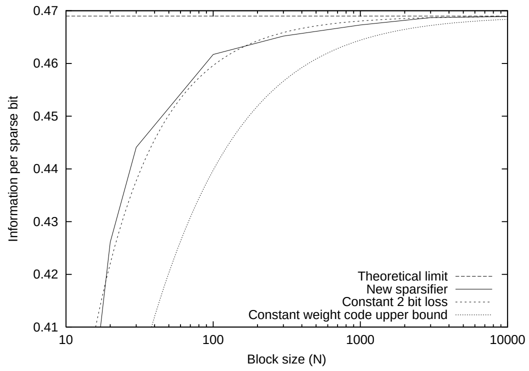


Figure 4: Efficiency of the proposed sparsification algorithm for $p_1 = 0.1$ assuming the worst performance of the arithmetic coding stage.

4 Sparse-Dense Codes on Multiuser channels

Some noisy channels can benefit from using an asymmetric code where zeros and ones are not equiprobable. For a single user channel the maximum gain over a traditional code is $(e/2)\ln 2 = 0.9421$ [7], but for a multiuser channel larger gains can be achieved.

The method of creation sparse data blocks here does not include the facility for any error correction. However feed-

Weight	$log_2 Pr$	Full Code	Length	Truncated Code	Length
0	3.04	010	3	010	3
1	1.89	10	2	10	2
2	1.81	11	2	11	2
3	2.4	00	2	00	2
4	3.48	0111	4	0111	4
5	4.97	01101	5	0110	4
6	6.82	011001	6		
$\vdots$	$\vdots$	$\vdots$	$\vdots$		
18	52.5	011000000000000001	18		
19	58.9	0110000000000000001	19		
20	66.4	0110000000000000000	19		

Table 1: Two Huffman codes generated for $N = 20$, $p_1 = 0.1$

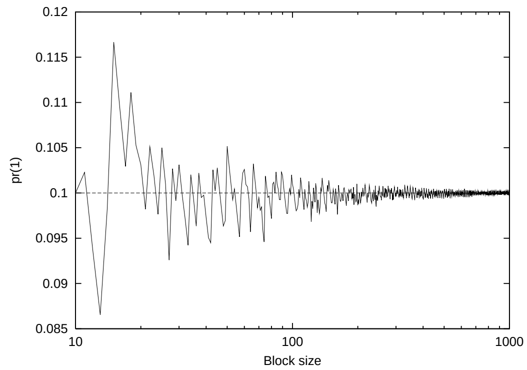
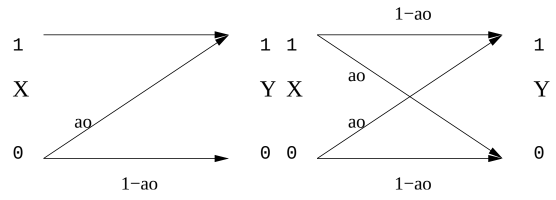


Figure 5: The actual sparsification achieved as a function of block size for a requested sparsification of $p_1 = 0.1$.



(a) Parallel Z channel (b) Parallel BSC

Figure 6: The two channel models studied. o is the number of ones being transmitted on the other channels at the current time and a is the noise level parameter.

ing sparse bits to a systematic error-correcting code encoder will lead to a code word with lower weight than usual. This is due to the systematic bits being sparse and the parity bits being dense (for a good code).

We will look at two simple channel models as an example.

Parallel Z Channel Each user's channel model is a Z-channel where the probability of corrupting 0s is proportional to the number of 1s being transmitted on the other channels, figure 6(a). This is similar to some optical channels (eg wave-length division multiplexing or a bundle of fibres with cross talk between them).

Parallel Binary Symmetric Channel (BSC) Each user's channel model is a BSC where the flip probability is proportional to the number of 1s being transmitted on the other channels, figure 6(b). This example is similar to noise on code-division multiple access (CDMA) channels.

4.1 Theoretical Capacity

Under the assumption of each user using the same bandwidth three different capacities can be calculated for the single user.

Dense transmissions The capacity achievable if each bit transmitted is i.i.d. with $p_1 = 0.5$. This capacity can be approached using linear error-correcting codes with no sparsification.

Sparse transmission This is the capacity achievable if each bit is i.i.d. with $p_1 \neq 0.5$.

Sparse-Dense transmission This is the capacity achievable if each data bit is transmitted in a sparse form and each check digit is dense.

When calculating the capacities for the example channels one can take the average codeword weight and then use this to calculate the noise level – this is assuming a large number of users with independent interleavers and a consequence of the noise being linear.

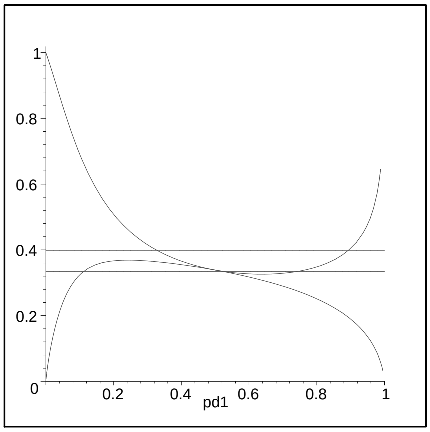
So to calculate the dense data capacity we simply calculate $\mathrm{I}(X;Y)|_{p_1=0.5}$. For the sparse calculation we evaluate $\max_{p_1} \mathrm{I}(X;Y)$, remembering that the noise level depends on X.

Calculation of the sparse-dense capacity is more complicated as we have two expressions for the capacity, one from Shannon and one from needing all the user data to be able to be contained within the sparse bits:

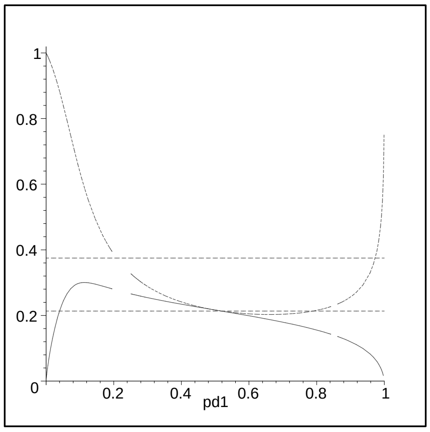
$$C = \max_{p_1} f(p_1) \quad (11)$$

$$f(p_1) = \mathrm{I}(X;Y) = R_d \mathrm{H}_2(p_1) \quad (12)$$

where R_d is the ratio of source bits to block size (which is the rate of the traditional error-correcting code being used). A symbolic maths package was used to solve the



(a) Parallel Z channel with 128 users and $a = 0.0074$



(b) Parallel BSC with 128 users and $a = 0.0037$

Figure 7: Plots $f(p_1)$ for a sparse-dense code. The lower curve shows $f(p_1)$ (the sparse-dense capacity is at the maximum) and the upper curve shows the corresponding dense code rate, R_d. The lower line indicates the dense transmission capacity and the upper line the sparse transmission capacity.

equations. For the parallel Z channel with 128 users and $a = 0.0074$, $f(p_1)$ is shown in figure 7(a). Similarly $f(p_1)$ is plotted for the parallel BSC channel, figure 7(b).

4.2 Results

The two channel models presented above were briefly studied by simulation.

4.2.1 Parallel Z Channel

Two codes of rate 0.3 were constructed. The first was an $N = 10000$, $R = 0.3$, $j = 3$ low-density parity-check (LDPC) code created using Construction 1A [4]. The second was a Sparse-Dense code created using a $p_1 = 0.25$ sparsifier combined with an $N = 10000$, $R = 0.37$, $j = 3$ LDPC code (also created using Construction 1A) so that the total code rate was also 0.3.

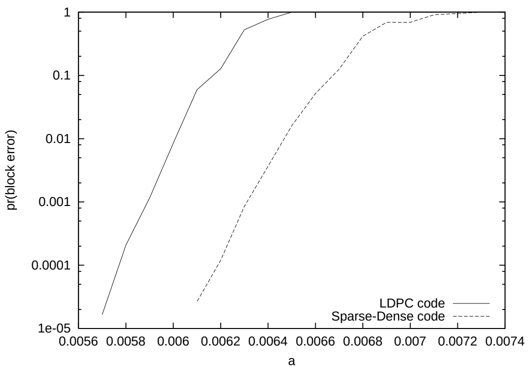


Figure 8: The performance of two $R = 0.3, N = 10000$ codes on a 128 user parallel Z channel

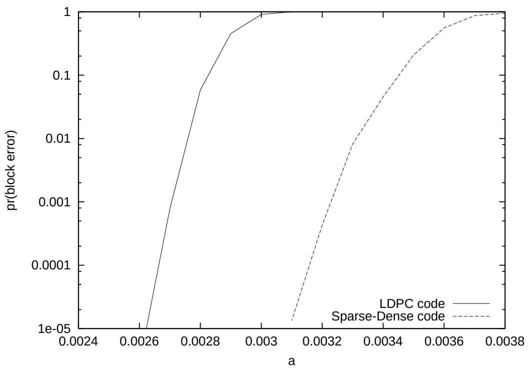


Figure 9: The performance of two $R = 0.2, N = 10000$ codes on a 128 user parallel BSC

The results from simulation are plotted in figure 8. It can be seen that the Sparse-Dense code outperforms a traditional LDPC code.

4.2.2 Parallel BSC

In a similar way as the parallel Z channel a simulation was carried out of the parallel BSC using two $N = 10000$, $R = 0.2$ codes. The two codes were an LDPC code and a sparse-dense code created with a $p_1 = 0.11$ for the data bits. The simulation results are shown in figure 9. A gain over the LDPC code was observed for the sparse-dense code.

5 Discussion

Despite outperforming other known algorithms for large blocks, the current sparsifier algorithm will not reach the theoretical limit. This is due to two reasons. Firstly the selection of the weight transmitted using a Huffman code is not optimal (unless $p_1 = 1/2$). Secondly the M-out-of-N code will never be optimal for all M (unless $N \leq 2$). The first can not be fixed by dividing up the probabilities of the weights as we need a one-to-one mapping between weights and prefix code words to maintain unique decodability. The second is not a concern for large block sizes as the loss (assuming computational arithmetic accuracy) is never more than 1 bit. It can be seen that the sparsifier is performing better than other algorithms and is reaching the theoretical limit more rapidly than solely using a constant weight code.

The codes presented show a gain of the order expected from the capacity calculations. For the parallel BSC, sparse codes are more advantageous than for the parallel Z channel; for the single user Z channel the optimum input distribution has $p_1 > 1/2$, so for the multiuser channel the two effects work against each other. This can be seen in both the simulations and capacity calculations. It is expected a parallel Z channel with the opposite asymmetric error would show a still larger gain.

The codes chosen were not optimized in detail for the channel model. The capacity graphs were used as a guide for the parameters to use. Optimization of R_d and p_1 could be carried out. Also irregular LDPC codes could be used with degree sequences optimized for the two different channel models.

Further gains on the channel models studied are liable to obtained by using a technique that creates sparse blocks with the ability to correct errors. This will allow us to investigate codes which can approach the sparse transmission capacity, in the two cases illustrated a further gain similar to the gain of a sparse-dense code over a dense code could be achieved. A technique that adds this directly to the sparse graph of an LDPC code could be investigated so that belief propagation decoding can extend over a sparsification algorithm too. Such codes could also be useful for single user asymmetric channels.

6 Conclusion

We have presented a technique for creating large sparse blocks of data and an example from error-correcting coding where they can be used to produce a significant gain over traditional linear codes.

References

[1] J.M. Berger. A note on error detection codes for asymmetric channels. *Information and Control*, 4:68–73, 1961.

[2] Matthew C. Davey and David J.C. MacKay. Reliable communication over channels with insertions, deletions, and substitutions. *IEEE Transactions on Information Theory*, 47(2):687–698, February 2001.

[3] David A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[4] David J.C. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, 1999.

[5] Alistair Moffat, Radford M. Neal, and Ian H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, July 1998.

[6] Tenkasi V. Ramabadran. A coding scheme for m-out-of-n codes. *IEEE Transactions on Communications*, 38(8):1156–1163, August 1990.

[7] Richard A. Silverman. On binary channels and their cascades. *IEEE Transactions on Information Theory*, 1(3):19–27, December 1955.

[8] Jong-Hoon Youn and Bella Bose. Some improved encoding and decoding schemes for balanced codes. In *IEEE Pacific Rim International Symposium on Dependable Computing*, pages 103–109, December 2000.