Windowed Erasure Codes

Chris Studholme
Department of Computer Science
University of Toronto
Email: cvs@cs.utoronto.ca

Ian Blake
Department of Electrical and Computer Eng.
University of Toronto
Email: ifblake@comm.utoronto.ca

***Abstract*—The design of erasure correcting codes and their decoding algorithms is now at the point where capacity achieving codes are available with decoding algorithms that have complexity that is linear in the number of information symbols. One aspect of these codes is that the overhead (number of coded symbols beyond the number of information symbols required to achieve decoding completion with high probability) is linear in k. This work considers a new class of random codes which have the following advantages: (i) the overhead is constant (in the range of 5 to 10) (ii) the probability of completing decoding for such an overhead is essentially one (iii) the codes are effective for a number of information symbols as low as a few tens. The price for these properties is that the decoding complexity is greater, on the order of $k^{3/2}$. However, for the lower values of k where these codes are of particular interest, this increase in complexity might be outweighed by other significant advantages. The parity check matrices of these codes are chosen at random as windowed matrices i.e. for each column an initial starting position of a window of length w is chosen and the succeeding w positions are chosen at random by zero or one. It can be shown that it is necessary that $w = O(k^{1/2})$ for the probabilistic matrix rank properties to behave as a non-windowed random matrix. The sufficiency of the condition has so far been established by extensive simulation, although other arguments strongly support this conclusion.**

I. Introduction

The construction and analysis of good (capacity achieving) erasure codes have been intensively investigated over the past few years following the ground breaking work of Luby, Shokrollahi, Spielman, Mitzemacher and many others. Such codes can be used very effectively for large downloads over the Internet in a multicast scenario in the presence of packet losses, where there is no feedback channel, or for distributing and reconstructing files in the presence of server crashes, among other applications. The end result of this work, due to Shokrollahi and Maymounkov, is that we know how to construct erasure codes that can be decoded in linear (in number of data symbols) time with high probability of successful decoding. Such results are asymptotic in nature requiring a large number of data symbols to achieve the promised performance. Additionally, the number of extra coded symbols required to ensure a high probability of decoding for block lengths in the few thousands, might typically be on the order of 3 to 4% of the number of data symbols.

One aspect of this work that is considered here, is to propose a new class of codes with a decoding algorithm that is Gaussian elimination. The class of codes will have an encoder complexity of $O(\log k)$ block operations per coded symbol and a decoder complexity of $O(k^{3/2})$, where k is the number of data symbols. In return for this increased decoder complexity it is shown that several advantages accrue: (i) the relationship between the probability of decoding success and overhead (the number of extra coded symbols) is quite precise. Indeed, the overhead is independent of the block length and can be very low (typically, only a few extra coded symbols). (ii) it is shown experimentally, that the codes are effective for numbers of data symbols down to less than one hundred. (iii) additionally, the parity checks for these codes can be obtained from a uniform distribution over subsets of the data symbols, rather than a more complicated distribution used in [9] and [10]. Thus while the codes have a significantly higher decoding complexity, there may well be applications where the added benefits of these codes are of interest. For lower block lengths, the additional decoding complexity is not very significant and these windowed codes become more attractive.

In the next section we briefly review the properties of LT, Raptor and Online codes. The following section reviews known rank properties of random binary matrices and introduces the windowed matrices on which the new class of codes will be based. At this point, the principal property required for the construction of our codes can only be established as a necessary condition. The sufficiency of the condition is only established through extensive simulation.

Section V gives the new encoding and decoding algorithm and relies on the work of section IV to establish their properties.

II. Previous Work

A very limited review of the remarkable recent work on erasure correcting codes is given. While work on low density parity check codes originates from the sixties [5], it was only more recent work that showed how such codes on the binary symmetric channel with errors (a transmitted binary 1 received as a zero or vice-versa) were capable of achieving the elusive goal of capacity. However, on the binary erasure channel a further step was taken to actually show how codes that achieve capacity can be constructed [7], [8], [9], [10]. Only this more limited case of coding for erasure channels is of interest here. Much of this work was concerned with designing (random) codes by constructing a bipartite graph whose incidence matrix is the parity check matrix of a linear code. A requirement of the construction [10] was to design two distributions on the degrees of the left and right vertices of this graph in such a way as to achieve capacity for a given decoding process.

Subsequent to this, the work of Luby [7], [8] introduced the notion of a *rateless* code. Again, this can be thought of in terms of a bipartite graph where the left vertices are identified with, say, k data or input symbols (which we may take as binary symbols for simplicity although the extension to strings of binary symbols is trivial). The right vertices are thought of as parity checks on the input symbols or as coded symbols. These are generated as follows: for a given degree distribution on the integers $1, 2, \cdots, k, \;\; \rho(k)$, the distribution is sampled to give an integer $d, \;\; 1 \leq d \leq k$. The corresponding code symbol is formed by choosing d input symbols at random and XOR'ing them. For decoding, if a sufficient number of coded symbols are obtained, the process starts by choosing a coded symbol of degree 1 i.e. a code symbol corresponding to a right vertex of degree 1. The value of the code symbol is transferred to the corresponding input symbol, whose value is then transferred to all coded symbols containing it, and all the corresponding edges are removed from the graph. If a new right vertex is now of degree 1, the process continues.

Clearly the decoding continues until it completes with all input symbols recovered or there are no right vertices of degree 1 at some stage, in which case the decoding fails. To minimize the probability of the latter, the distribution $\rho(i)$ is chosen carefully. Initially, it was suggested [7] to use the *soliton* distribution given by $\rho(i) = 1/i(i-1), \;\; i = 2, 3, \cdots, k, \;\; \rho(1) = 1/k$. The theoretical reasoning for this distribution is sound, but it turns out in practice that the probability, at each stage of the decoding, of having a vertex of degree 1 is too sensitive, giving too high a probability the decoding would not complete. To remedy the situation, a *robust* soliton distribution was suggested that proved more robust in practice. These codes are referred to as rateless since the decoder can continue to gather coded symbols from any source, on the same input data, until decoding completes – there is no fixed *a priori* rate associated with the code. In conventional coding, it would first be necessary to estimate the erasure probability to allow proper code design and this rateless feature is very useful..

For an original set of k data or input symbols, the number of extra (more than k) coded symbols required to give a high probability of decoding completion is referred to as the *overhead*. It is shown in [7] that if δ is the allowable decoding failure probability then by using the robust soliton distribution for the generation of the coded symbols, the overhead required to decode is $O(\sqrt{k}\log^2(k/\delta))$ and the average degree of the right vertices is $D = \log(k/\delta))$. Since the complexity of decoding is proportional to the number of edges in the graph, this last result shows the decoding complexity to be $O(k\log(k/\delta))$.

To improve the situation further Shokrollahi [10] and independently Maymounkov [9] introduced the idea of *precoding*. Here one uses a good erasure correcting code to code the original input symbols resulting in a certain number of *auxiliary* symbols. These are then added to the input symbols (on the left hand side of the bipartite graph) and the combined set of left hand side symbols are coded using a truncated soliton-like encoder. This latter (outer) code has a mean right hand side degree that is independent of k, giving linear time encoding and decoding. In exchange for this complexity reduction, overhead is increased to $O(k)$ extra coded symbols. For convenience, we note Theorem 2 from [9], using our terminology. A similar result is found in [10].

Theorem 1 *(Maymounkov): For any message of size k input symbols, and any parameter $\epsilon > 0$, there is a rateless locally encodable code (right distribution) that can recover the input symbols from any $(1+\epsilon)k$ coded symbols with high probability in time proportional to $k\log(1/\epsilon)$.*

In the presence of such a powerful result one might wonder why further work on this problem is of interest. One aspect of the above is that they are essentially asymptotic results. In practice, with our simulations and others available in the references, a typical overhead for the LT decoding might be as much as 15% for a number of input symbols on the order of a thousand and this drops to perhaps 3% to 4% for a much higher number of input symbols. The Online or Raptor codes can be constructed to have an overhead of 3% to 4%, of k for large k.

The class of codes presented here will have a decoding complexity of $O(k^{3/2})$ for k input symbols. Although significantly higher than the previous work, for low k the increase is not so large when all the constants in the estimates are taken into account. In return for this increased complexity, one obtains codes that are effective for values of k as low as 100, are easily generated requiring only a uniform distribution, and have a very high and quantifiable probability of decoding completion with an overhead that is constant, independent of k. It is likely there will be applications where such properties are of interest.

III. Random Matrices

Properties of binary random matrices that are of interest for this work are briefly reviewed here using only a few of the large number of references [1], [2], [3], [4], [6] that have considered this interesting problem. The following theorem, due to Kolchin [6] but adapted to our terminology, is a remarkable result and more general than our interests:

Theorem 2 *(Kolchin): Let $m \geq 0$, be a fixed integer. Let the elements of a random $k \times (k+m), \; m \geq 0$, matrix $M = (m_{ij})$ be independent and suppose there is a positive constant δ such that the inequalities $\delta \leq P[m_{ij} = 1] \leq 1 - \delta$ hold for $i = 1, \ldots, k$, and $j = 1, \ldots, k+m$. Then as $k \to \infty$,*

$$P[\mathit{rank}(M) = k] \to Q_m := \prod_{i=m+1}^{\infty} \left(1 - \frac{1}{2^i}\right). \qquad (1)$$

It can also be shown that for $n \leq k$, the probability that n random columns have full rank (for $p = 0.5$, but also more generally) is given by

$$\prod_{i=1}^{n}\left(1-\frac{1}{2^{k-i+1}}\right).$$

The difference between this result and theorem when $n = k$ lies in the asymptotic nature of the former.

Although the theorem is an asymptotic result, we find that as long as δ is close enough to $1/2$, and in particular, if $p = 1/2$ is the probability of a 1 for all matrix elements, then Q_m is a very good estimate of the probability of full rank for k as low as 10. In fact, this expression also holds for values of p down to about $(2\log k)/k$. Below this, one can modify the above expression for Q_m to take into account the likelihood of all zero columns and rows. Such a modification is beyond the scope of this document, but it is interesting to note that a result of Cooper [4] indicates that this lower limit is actually of the form $(\log(k) + d(k))/k$, where $d(k)$ is a function that tends to zero sufficiently slowly as $k \to \infty$.

When $p = 1/2$ the expected number of 1's per column is $k/2$; however, the same probability of full rank can be achieved for p as low as $(2\log k)/k$, and in this case the expected number of 1's per column is only $2\log k$. The observation we make here is that one need not distribute these $2\log k$ 1's uniformly throughout the rows of the column. Instead, one can confine these 1's to a small number of consecutive rows, called a *window*. In the next section it is shown that results essentially the same as for random matrices can be achieved for window sizes down to about $2\sqrt{k}$.

Using Q_m, one can show the expected number of extra columns needed to obtain rank k as

$$\bar{m} = \sum_{m=0}^{\infty} mP_m = \sum_{i=0}^{\infty}(1-Q_i) = 1.60669515\ldots$$

where $P_m = Q_m - Q_{m-1}$ is the probability of requiring exactly m columns beyond k for the matrix to acquire full rank. We see that, with only 2 extra columns (independent of k), a random binary matrix has a better than 50% chance of having full rank. With 7 or 8 extra columns, the probability of full rank is very close to 1. Since our windowed matrices (with window length at least $2\sqrt{k}$) have a probability of full rank that is virtually identical to Q_m, these matrices will also have a high probability of full rank with only a few extra columns. This very low overhead is a significant advantage of the erasure code construction we give in section V.

IV. Rank properties of windowed matrices

We now describe the construction of windowed matrices and give an upper bound on the probability of full rank. The proof of our main theorem establishes a minimum window length of $\sqrt{k}$ to achieve a reasonably high probability of full rank, while the results of our simulations show that $2\sqrt{k}$ is sufficient. In future work we intend to show how one might calculate the exact probability of full rank as a function of window length.

Before stating our main theorem we wish to be precise about how a windowed column is generated. First, a starting row is chosen randomly and uniformly from among the k rows and a 1 is placed in this starting row. This 1 is referred to as the *initial 1*. Then, place 1's and 0's in the w rows immediately following the initial 1. We will either choose each row independently with p being the probability of a 1, or decide on a fixed column weight of σ and place $\sigma - 1$ 1's in the w rows. All other rows simply contain 0. Note that if the initial 1 appears within w rows of the bottom of the column, the window will wrap back to the top. The following theorem does not depend on which strategy is used.

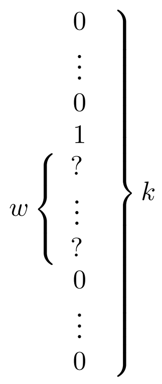

Fig. 1. Column structure for windowed column.

Theorem 3 *For sufficiently large k, the probability that a $k \times k$ random windowed binary matrix with window length $w = \delta\sqrt{k}/2$ has rank k is at most $2\Phi(\delta)Q_0$, where $\Phi(z)$ is the normal distribution function and $Q_0 = 0.288788\ldots$ as given by (1).*

Proof: Let A be a matrix consisting of k independently generated windowed columns with $w = \delta\sqrt{k}/2$. Without loss of generality, assume k is even. For each column, if the initial 1 falls in the first $k/2$ rows, label the column a *top* column. Similarly, if the initial 1 falls in the last $k/2$ rows, label it a *bottom* column.

Let t be a random variable representing the number of top columns generated, and let b represent the number of bottom columns. Both t and b are sampled from the binomial distribution with the probability of generating each type of column being $1/2$, so the expected number of columns of each type is $k/2$ and the standard deviation is $\sqrt{k}/2$.

Now, since the top columns can have 1's in at most $k/2+w$ rows, this is also the maximum number of independent top columns A may have. Similarly, the maximum number of independent bottom columns is also $k/2+w$. Combining these two constraints, and noting that $t + b = k$, gives us

$$k/2 - w \ \leq\ t\ \leq\ k/2 + w\ . \tag{2}$$

The above constraint is satisfied as long as t is within δ standard deviations of the mean. If k is sufficiently large, the binomial distribution may be approximated using the normal distribution and $2\Phi(\delta)$ is the probability that t is within δ standard deviations of the mean.

Finally, we assume that if (2) is satisfied, A's probability of having rank k is no greater than that of a random matrix, Q_0. Therefore, the probability of full rank is at most $2\Phi(\delta)Q_0$. ■

For $\delta = 2$, $2\Phi(\delta)$ is roughly 0.95. The above theorem tells us that the probability of full rank for a matrix with window length $w = \delta\sqrt{k}/2 = \sqrt{k}$ will be at most $0.95Q_0$. Actually, our experiments show that the probability is significantly less than this bound. Despite this, we see from extensive simulation that with only a factor of 2 increase in the window length, to $2\sqrt{k}$, the bound can be achieved.

Figure 2 demonstrates this effectiveness. The open squares and triangles show the results for window sizes of $2\sqrt{k}$, $1.5\sqrt{k}$ and $\sqrt{k}$. Clearly, $w = 2\sqrt{k}$ is necessary for good results. In the $k = 2,500$ graph, the solid triangles show that for $w = 2\sqrt{k}$, a densely packed window having mean column weight $\sqrt{k}$ is unnecessary. The *low weight* data series was generated using $p = (\log k)/\sqrt{k}$ for a mean column weight of $2\log k$.

While our theorem gives a necessary condition for a windowed matrix to have a high probability of full rank, in future work we hope to prove that a window size of $2\sqrt{k}$ is sufficient. Thus far we have successfully reduced the problem of computing the probability of full rank as a function of window size to a cell occupancy problem. One considers throwing k balls in k cells, where the cells are viewed as the starting row of a column and the balls as columns. If the window size is w and the number of balls in any t consecutive cells, viewing the cells as arranged in a circle, exceeds $t + w$, the matrix cannot have full rank.

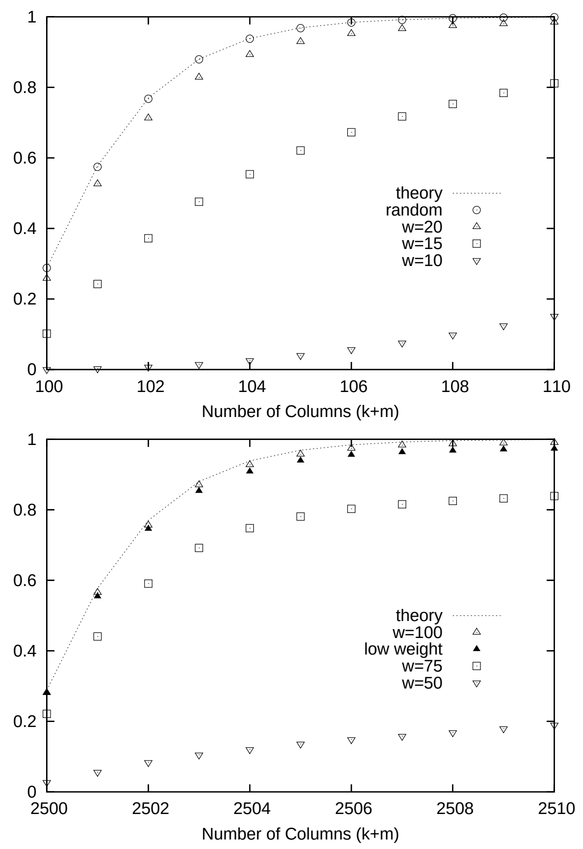

Fig. 2. Probability of rank k for a $k \times (k + m)$ matrix. The upper graph shows the results for $k = 100$, while the lower is for $k = 2500$. In both graphs, the broken line is Q_m, the open circles show results from random ($p = 1/2$) matrices, and the other symbols show results for windowed matrices. For clarity, no open circles are shown in the lower graph. Instead, the *low weight* closed triangles are the results for a window size of $w = 100$ but with the mean column weight fixed at $2\log k$. All data points are the results of testing at least $5,000$ matrices.

V. Erasure code construction

These windowed matrices can be used as the basis for an efficiently encodable and decodable erasure code. The code we describe here has an encoder complexity of about $2\log k$ block sums per output block and a decoder complexity of $1.3k^{3/2}$ block sums. This complexity is achieved while keeping the overhead constant at approximately 2 extra blocks.

A. Encoding

The encoder has k data symbols (input blocks) that need to be transmitted to the decoder. To create an output block, the encoder first generates a windowed column as described in the previous section. We suggest the encoder generate fixed weight columns with $\sigma = \lceil 2\log k \rceil_{odd}$. This notation is used to refer to the lowest odd integer greater than or equal to $2\log k$. Note that k columns each having even weight cannot have rank k, and therefore, σ must be odd or the decoder will always fail.

Along with generating a windowed column, the encoder sums (using bitwise exclusive or) the σ input blocks corresponding to the 1's in the column. The column (or some efficient encoding of it) along with the sum are then transmitted to the decoder.

Although setting $w = 2\sqrt{k}$ will work reasonably well, we suggest one instead set $w = 2(\sqrt{k} - 1)(\sigma - 1)/(\sigma - 2)$. This slightly larger value ensures that the expected number of rows between the first and last 1 (inclusive) in each column is $2\sqrt{k}$.

B. Decoding

The decoding algorithm is simply *Gaussian Elimination*; however, to ensure our discussion of decoder complexity is precise, we describe a specific decoding algorithm. Decoding has two phases:

1) **Column collection.** During column collection, matrix columns, along with their associated data blocks, are inserted into a hash table and reduced as necessary to ensure they are independent.
2) **Back filling.** At the conclusion of column collection, we have a lower triangular matrix with 1's along the diagonal. This matrix is non-singular. With a series of data block sums, the matrix is made diagonal and decoding is complete.

The hash table constructed during the first phase is to have exactly k bins and each column received hashes to the bin whose index is the index of the first row containing a 1. For the purposes of this algorithm, columns are not considered to wrap from bottom to top, and as a result, the first 1 in a column may not coincide with the initial 1. When this table is full, the k columns comprise a lower triangular matrix with 1's along the diagonal.

Hash collisions will occasionally happen during column collection. To resolve such a collision, simply add the two columns (and their associated data blocks) together to get a column that hashes to a different bin. A subtle but important detail of this algorithm is the choice of columns to keep after collision resolution. Obviously, the sum is to be kept. The other column to keep is the *shorter* of the two colliding columns. Here, the length of a column is the number of rows between the first 1 and the last 1 (inclusive). If the two columns are of equal length, either one may be kept. Two identical columns indicate a dependency; one is simply discarded and an extra column must be collected.

When the hash table is full, back filling can begin. Back filling is done starting at the last row and working up through the matrix. First, the data block associated with the 1 on the diagonal is added to all of the data blocks associated with 1's in the last row. Then, the second to last row is processed in a similar manner. At the completion of back filling, the data blocks will be the original input blocks.

Theorem 4 *Worst case decoder complexity is $\bar{\ell}k$ data block additions, where $\bar{\ell}$ is the mean column length. Column length, ℓ, as mentioned earlier in this section, is the number of rows between the first* 1 *and the last* 1, *inclusive.*

Proof: During the column collection phase, one data block addition is required each time there is a hash table collision. If two columns, one of length x and the other of length y, $x \leq y$, collide, their sum will be a column whose length is no greater than $y - 1$. Since $\bar{\ell}k$ is the sum of the length's of the columns and each collision reduces this total length by at least 1, there can be at most $\bar{\ell}k$ collisions.

During the back filling phase, the number of data block additions needed is exactly the weight of the matrix (after column collection) less k. Also, the weight of the matrix is no greater than the total length, and the total length after column collection no greater than the total length before column collection less the number of collisions. Therefore, the sum of the weight of the matrix after column collection and the number of collisions resolved during column collection is at most $\bar{\ell}k$. ■

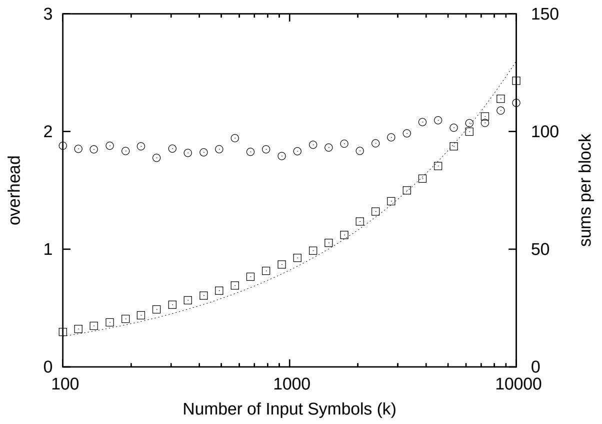

Fig. 3. Overhead and decoder performance for a windowed erasure code. The circles show the mean number of extra columns needed (left axis) while the squares show the mean number of column operations needed per input symbol (right axis). Each data point is the mean from 10,000 runs. The broken line is $1.3\sqrt{k}$.

The average case complexity is $\bar{\ell}k/2$. This follows from the fact that when columns of length x and y, $x \leq y$, are added, the expected length of the resulting column is $y - 2$. Furthermore, the expected weight of the matrix after column collection is half the total length.

To see how the complexity may be calculated from w, first notice that for columns that do not wrap, $\ell \leq w + 1$; however, for columns that do wrap, ℓ may be as large as k. Since the probability of generating a column that wraps is w/k, mean column length can be calculated as $\bar{\ell} = (w/k)k + (1 - w/k)(w + 1) \approx 2w$. When using $w \approx 2\sqrt{k}$ as suggested, expected decoder complexity is $wk \approx 2k^{3/2}$.

Figure 3 shows that for k between 100 and 10,000 we achieve a decoder complexity of $1.3k^{3/2}$ while maintaining an overhead of about 2 extra blocks. For an idea of real world performance we performed two tests involving a 32 MiByte file on an AMD Athlon XP 1800+. Decoding 8192 blocks (4 kiB each) requires about 10 seconds of CPU time, while decoding 65536 blocks (512 bytes each) takes 2 minutes.

Finally, we note that allowing columns to wrap from bottom to top increases decoder complexity by a factor of 2. We have developed a strategy to avoid such wrapping; however, we find that to maintain a high probability of full rank, w must be as large as $4\sqrt{k}$ for some columns. This negates most of the benefit of avoiding wrapping, and therefore, we do not describe the strategy here.

VI. Conclusions

An efficiently encodable and decodable class of erasure correcting codes, with decoding complexity $O(k^{3/2})$, based on the rank properties of windowed binary random matrices, has been formulated. They have the advantages, over other classes of erasure correcting codes, of a low and fixed overhead and effectiveness at much lower block lengths. The necessity of the rank properties required for the windowed matrices are established here, and while simulation and intuitive arguments show the conditions needed are clearly sufficient, it is hoped to establish these properties analytically in future work.

References

[1] Johannes Blömer, Richard Karp, and Emo Welzl, *The rank of sparse random matrices over finite fields*, Random Structures and algorithms **10** (1997), 407–419.
[2] R. Brent, S. Gao, and A. Lauder, *Random Krylov spaces over finite fields*, SIAM J. Dicsrete Math. **16** (2003), 276–287.
[3] Colin Cooper, *On the distribution of rank of a random matrix over a finite field*, Random Structures and Algorithms **17** (2000), 197–212.
[4] ______, *On the rank of random matrices*, Random Structures and Algorithms **16** (2000), no. 2, 209–232.
[5] R.G. Gallager, *Low density parity check codes*, MIT Press (1963).
[6] V.F. Kolchin, *Random graphs*, Cambridge University Press (1999).
[7] Michael Luby, *LT codes*, In The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002.
[8] Michael Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel Speilman, and Volker Stemann, *Practical loss-resilient codes*, STOC (1997), 150–159.
[9] Petar Maymounkov, *Online codes*, 2002, Technical Report TR2002-833, New York University, October 2002.
[10] Amin Shokrollahi, *Raptor codes*, 2003, Available at: `http://www.inference.phy.cam.ac.uk/mackay/DFountain.html`.