Efficient Erasure Codes

Michael G. Luby*   Michael Mitzenmacher†   M. Amin Shokrollahi‡

Daniel A. Spielman§   Volker Stemann¶

*International Computer Science Institute, Berkeley, California. Research supported in part by National Science Foundation operating grant NCR-9416101, and United States-Israel Binational Science Foundation grant No. 92-00226.

†Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. A substantial portion of this research done while at the Computer Science Department, UC Berkeley, under the National Science Foundation grant No. CCR-9505448.

‡Bell Labs, Murray Hill, NJ. Research done while at the International Computer Science Institute Berkeley. Research supported by a Habilitationsstipendium of the Deutsche Forschungsgemeinschaft, Grant Sh 57/1–1.

§Department of Mathematics, M.I.T. Supported by an NSF mathematical sciences postdoc. A substantial portion of this research done while visiting U.C. Berkeley.

¶Research done while at the International Computer Science Institute.

Abstract

We introduce a simple erasure recovery algorithm for codes derived from cascades of sparse bipartite graph and analyze the algorithm using the theory of discrete time random processes. As a result we obtain a simple criterion invovlving the fractions of nodes of different degrees on both sides of the graph which is necessary and sufficient for the decoding process to finish successfully with high probability. By carefully designing these degrees we construct for any given rate R and any given real number ϵ a sequence of linear codes of rate R which can be encoded in time proportional to $\ln(1/\epsilon)$ times their block length. Furthermore, a codeword can be recovered with high probability from an portion of its entries of length $(1+\epsilon)Rn$ or more. The recovery algorithm also runs in time proportional to $n\ln(1/\epsilon)$. Various implementation issues are discussed.

1 Introduction

A linear error-correcting code of block length n and dimension k over a finite field $\mathbb{F}_q$—an $[n,k]_q$-code for short—is a k-dimensional linear subspace of the standard vector space $\mathbb{F}_q^n$. The elements of the code are called codewords. To the code C there corresponds an *encoding map* Enc which is an isomorphism of the vector spaces $\mathbb{F}_q^k$ and C. A sender, who wishes to transmit a vector of k elements in $\mathbb{F}_q$ to a receiver uses the mapping Enc to encode that vector into a codeword. The *rate* k/n of the code is a measure for the amount of real information in each codeword. The minimum distance of the code is the minimum Hamming distance between two distinct codewords. A linear code of block length n, dimension k, and minimum distance d over $\mathbb{F}_q$ is called an $[n,k,d]_q$-code.

Linear codes can be used to reliably transmit information from a sender to a receiver: the sender first encodes the desired word into a codeword and transmits the codeword over the transmission channel. Depending on the nature of the errors imposed on the codeword through the channel, the receiver then applies appropriate algorithms to *decode* the received word. In this paper, we assume that the receiver knows the position of each received symbol within the stream of all encoding symbols. We adopt as our model of losses the *erasure channel*, introduced by Elias [3], in which each encoding symbol is lost with a fixed constant probability p in transit independent of all the other symbols. Elias [3] showed that the capacity of the erasure channel is $1-p$ and that a random linear code can be used to transmit over the erasure channel at any rate $R < 1-p$.

It is easy to see that a code of minimum distance d is capable of recovering $d-1$ or less erasures. Furthermore, a closer look reveals that this task can be done in time $O(n^3)$. The code is optimal with respect to recovering erasures if it can recover from any set of k coordinates of the encoding, i.e., if $d-1 = n-k$. Such codes are called MDS-codes. A standard class of MDS-codes is given by Reed-Solomon codes [9]. The connection of these codes with polynomial arithmetic allows for encoding and decoding in time $O(n\log^2 n\log\log n)$. (See, [2, Chapter 11.7] and [9, p. 369]). However, for small values of n, quadratic time algorithms are faster than the fast algorithms for the Reed-Solomon based codes, and for larger values of n the $O(\log^2 n\log\log n)$ multiplicative overhead in the running time of the fast algorithms (with a moderate sized constant hidden by the big-Oh notation) is large.

Elias's paper

Obviously, one cannot hope for better information recovery than those given by Reed-Solomon codes, but faster encoding and decoding times are desirable. We will design fast linear-time algorithms for transmitting just below channel capacity. For all $\epsilon > 0$ we produce rate $R = 1-p(1+\epsilon)$ codes along with decoding algorithms that recover from the random loss of a p fraction of the transmitted symbols in time proportional to $n\ln(1/\epsilon)$ with high probability, where n is the block length. They can also be encoded in time proportional to $n\ln(1/\epsilon)$. The fastest previously known encoding and decoding algorithms [1] with such a performance guarantee have run times proportional to $n \ln(1/\epsilon)/\epsilon$.

The overall structure of our codes are related to codes introduced in [15] for error-correction. We explain the general construction along with the encoding and decoding algorithms in Section 2.

Our encoding and decoding algorithms are almost symmetrical. Both are very simple, computing exactly one exclusive-or operation for each edge in a randomly chosen bipartite graph. As in many similar applications, the graph is chosen to be sparse, which immediately implies that the encoding and decoding algorithms are fast. Unlike many similar applications, the graph is not regular; instead it is quite irregular with a carefully chosen degree sequence. We describe the decoding algorithm as a process on the graph in Section 2.2. Our main tool is a model that characterizes almost exactly the performance of the decoding algorithm as a function of the degree sequence of the graph. In Section 3, we use this tool to model the progress of the decoding algorithm by a set of differential equations. The solution to these equations can then be expressed as polynomials in one variable with coefficients determined by the degree sequence. The positivity of one of these polynomials on the interval $(0, 1]$ with respect to a parameter δ guarantees that, with high probability, the decoding algorithm can recover almost all the message symbols from a loss of up to a δ fraction of the encoding symbols, see Proposition 1. The complete success of the decoding algorithm can then be demonstrated by combinatorial arguments such as Lemma 1.

Our analytical tools allow us to almost exactly characterize the performance of the decoding algorithm for any given degree sequence. Furthermore, they also help us to *design* good irregular degree sequences. In Section 4 we describe, given a parameter $\epsilon > 0$, a degree sequence for which the decoding is successful with high probability for a loss fraction δ that is within ϵ of $1 - R$. Although these graphs are irregular, with some nodes of degree $1/\epsilon$, the average node degree is only $\ln(1/\epsilon)$. This is one of the central results of the paper, i.e., a code with encoding and decoding times proportional to $n \ln(1/\epsilon)$ that can recover from a loss fraction that is within ϵ of optimal.

In Section 5 we discuss issues concerning practical implementations of our algorithms. This section includes methods for finding good degree sequences based on linear programming, and timings of the implementations. In the last section we summarize the main results of this paper and pose several open problems.

2 Graph Codes

In this section we will introduce a new class of codes. Special subclasses of these codes turn out to be almost MDS in the following sense: an $[n, k]_q$-code in this subclass is capable of recovering the message from any set of $k(1 + \epsilon)$ coordinate places, where ϵ is a small real number. The advantages of these codes are that they have linear time encoding and decoding algorithms, and that q can be arbitrary. However, for simplicity, we will in the following assume that the symbols are bits, i.e., that $q = 2$, Unfortunately, we need to pay a price for the high quality of these codes: the decoding algorithm is successful only with high probability. A precise statement of this will be provided later in Section 3.

We will explain the overall construction of the codes which is related to codes introduced in [15] for error-correction. We will also introduce simple and efficient encoding and recovery algorithms.

2.1 Erasure Codes via Bipartite Graphs

We begin by defining a code $\mathcal{C}(B)$ with k message bits and βk check bits, by associating these bits with a bipartite graph B, where $0 < \beta < 1$. The graph B has k left nodes and βk right nodes,

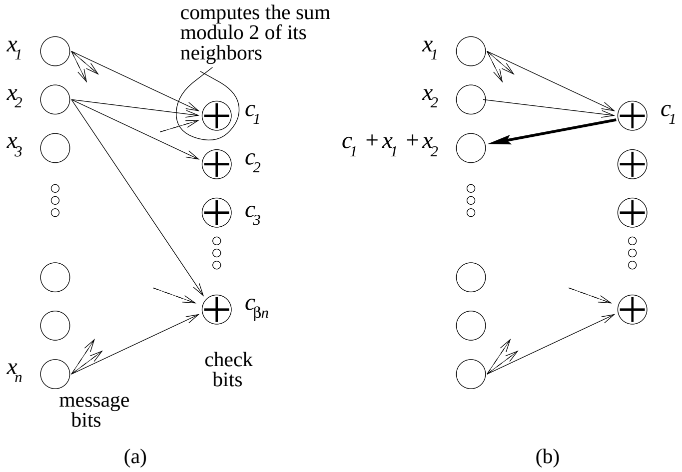

Figure 1: (a) A graph defines a mapping from message bits to check bits.
(b) Bits x_1, x_2, and c_1 are used to solve for x_3.

corresponding to the message bits and the check bits, respectively. In the following, we will refer to the left nodes of a bipartite graph as its *message bits* and to the right nodes as its *check bits.*

$\mathcal{C}(B)$ is encoded by setting each check bit to be the $\oplus$ (XOR) of its neighboring message bits in B (see Figure 1(a)). Thus, the encoding time is proportional to the number of edges in B.

Our main contribution is the design and analysis of the bipartite graph B so that the repetition of the following simplistic decoding operation recovers all the missing message bits.

Algorithm 1 (Loss recovery) *Given the value of a check bit and all but one of the message bits on which it depends, set the missing message bit to be the XOR of the check bit and its known message bits.*

See Figure 1(b) for an example of this algorithm, and Figure 2 for an example of full recovery. We will introduce methods for the design of sparse random graphs where repetition of this operation is guaranteed to usually recover all the message bits if at most $(1-\epsilon)\beta k$ of the message bits have been lost from $\mathcal{C}(B)$.

To produce codes that can recover from losses regardless of their location, we cascade codes of the form $\mathcal{C}(B)$: we first use $\mathcal{C}(B)$ to produce βk check bits for the original k message bits, we then use a similar code to produce $\beta^2 k$ check bits for the βk check bits of $\mathcal{C}(B)$, and so on (see Figure 3). At the last level, we use a more conventional loss resilient code, e.g., an RS-code.

Formally, we construct a sequence of codes $\mathcal{C}(B_0), \ldots, \mathcal{C}(B_m)$ from a sequence of graphs $B_0, \ldots, B_m$, where B_i has $\beta^i k$ left nodes and $\beta^{i+1} k$ right nodes. We select m so that $\beta^{m+1} k$ is roughly $\sqrt{k}$ and we end the cascade with a loss resilient code C of rate $1-\beta$ with $\beta^{m+1} k$ message bits for which we know how to recover from the random loss of β fraction of its bits with high probability. We then define the code $\mathcal{C}(B_0, B_1, \ldots, B_m, C)$ to be a code with k message bits and

$$\sum_{i=1}^{m+1} \beta^i k + \beta^{m+2} k/(1-\beta) = k\beta/(1-\beta)$$

check bits formed by using $\mathcal{C}(B_0)$ to produce βk check bits for the k message bits, using $\mathcal{C}(B_i)$ to form $\beta^{i+1} k$ check bits for the $\beta^i k$ bits produced by $\mathcal{C}(B_{i-1})$, and finally using C to produce an additional $k\beta^{m+2}/(1-\beta)$ check bits for the $\beta^{m+1} k$ bits output by $\mathcal{C}(B_m)$. As $\mathcal{C}(B_0, B_1, \ldots, B_m, C)$ has k message bits and $k\beta/(1-\beta)$ check bits, it is a code of rate $1-\beta$.

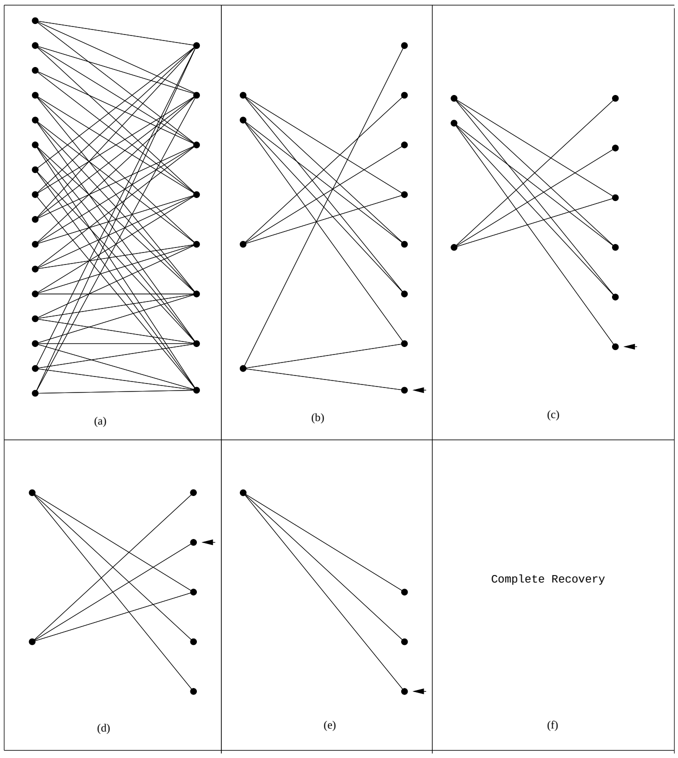

Figure 2: All stages of the recovery. (a) Original graph, (b) Graph induced by the set of lost nodes on the left, (c)–(f) Recovery process

Remark 1 *Assuming that the code C can be encoded and decoded in quadratic time (an assumption which is certainly true for RS-codes), the code $\mathcal{C}(B_0, \ldots, B_m, C)$ can be encoded and decoded in time proportional to the number of edges in all the $\mathcal{C}(B_i)$.*

We begin by using the decoding algorithm for C to recover losses that occur within its corresponding bits. If C recovers all the losses, then the algorithm now knows all the check bits produced by $\mathcal{C}(B_m)$, which it can then use to recover losses in the inputs to $\mathcal{C}(B_m)$. As the inputs to each $\mathcal{C}(B_i)$ were the check bits of $\mathcal{C}(B_{i-1})$, we can work our way back up the recursion until we use the check bits produced by $\mathcal{C}(B_0)$ to recover losses in the original k message bits. If we can show that C can recover from the random loss of a $\beta(1-\epsilon)$ fraction of its bits with high probability, and that each $\mathcal{C}(B_i)$ can recover from the random loss of a $\beta(1-\epsilon)$ fraction of its message bits with high probability, then we have shown that $\mathcal{C}(B_0, B_1, \ldots, B_m, C)$ is a rate $1-\beta$ code that can recover from the random loss of a $\beta(1-\epsilon')$ fraction of its bits with high probability, for some ϵ'. Details can be found in the proof of Theorem 3.

For the remainder of this section and much of the next section, we only concern ourselves with finding graphs B so that the decoding algorithm can recover from a $\beta(1-\epsilon)$ fraction of losses in the message bits of $\mathcal{C}(B)$, given all of its check bits.

2.2 The Graph Process and Degree Sequences

We now relate the decoding process of $\mathcal{C}(B)$ to a process on a subgraph of B, so that hereafter we can use this simpler terminology when describing the process. This subgraph consists of all nodes on the left that were lost but have not been decoded thus far, all the nodes on the right, and all

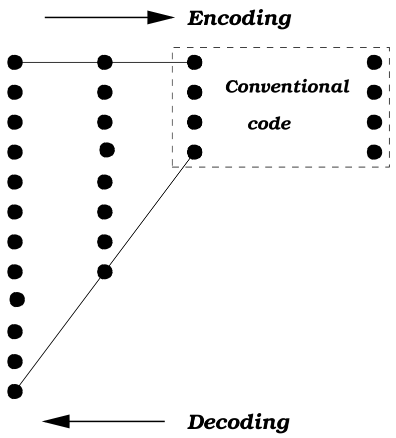

Figure 3: The code levels and directions of encoding and decoding process

the edges between these nodes. Recall that the decoding process requires finding a check bit on the right such that only one adjacent message bit is missing; this adjacent bit can then be recovered. In terms of the subgraph, this is equivalent to finding a node of degree one on the right, and removing it, its neighbor, and all edges adjacent to its neighbor from the subgraph. We refer to this entire sequence of events hereafter as one step of the decoding process. We repeat this step until there are no nodes of degree one available on the right. The entire process is successful if it does not halt until all nodes on the left are removed, or equivalently, until all edges are removed.

The graphs that we use are chosen at random from a set of sparse bipartite graphs with a carefully chosen degree sequence. In contrast with many applications of random graphs in computer science, our graphs are not regular.

We refer to edges that are adjacent to a node of degree i on the left (right) as *edges of degree* i on the left (right). Each of our degree sequences is specified by a pair of vectors $(\lambda_1, \ldots, \lambda_m)$ and $(\rho_1, \ldots, \rho_m)$, where λ_i is the initial fraction of edges on the left of degree i and ρ_j is the initial fraction of edges on the right of degree j. Note that our graphs are specified in terms of fractions of *edges*, and not *nodes*, of each degree, as this form turns out to be more convenient.

Note that, in the corresponding subgraph B' of B remaining after each step, the matching remaining on B' still corresponds to a random permutation. Hence, conditioned on the degree sequence of the remaining subgraph after each step, the subgraph that remains is uniform over all subgraphs with this degree sequence. The evolution of the degree sequence is therefore a Markov process, a fact we make use of below.

In the next two sections, we develop techniques for the analysis of the process for general degree sequences.

3 Sharp Concentration and Analysis of the Decoding

We will analyze the decoding algorithm in 1 by viewing it as a discrete random process. (For an alternative analysis, see [8].) We model the evolution of the main parameters of this system by a system of differential equations. These parameters include the number of edges of different right and left degrees, as well as the total number of edges and the average degrees of the bipartite graph on both sides. We will need a result which makes sure that these parameters are sharply concentrated around the solutions of the system of equations, in the sense that the variation in the parameters are small compared with the total number of steps. For the sake of keeping the technical discussion at an acceptable level, we will not aim at proving the best possible results on the quality of the sharpness of the concentration.

In the fist part of this section, we will state a general sharp concentration result. Such results have been obtained by Kurtz [7] who studied Markov jump processes, and have been used previously by many researchers, see [4, 6, 10, 11, 13, 16] and the references therein. We will, however, use a version due to Wormald [16] which has the advantage of being directly applicable to our situation.

Next we will set up the appropriate system of differential equations, and solve them explicitly. This will provide us with a concrete condition on the bipartite graph for successful decoding. However, we can only make limited use of the sharp concentration result, as this will only guarantee continuation of the recovery process as long as the number of edges in the induced subgraphs is a constant fraction of the original number of edges. To prove that the process ends successfully, we will need a combinatorial argument which proves that the random graph obtained at this stage of the decoding has reasonable expansion properties, with high probability.

3.1 Sharp Concentration

For analyzing the recovery algorithm in 1 we need to keep track of nodes of degree one on the right side of the bipartite graph. Doing this deterministically is impossible, as the erasures occur randomly on the left side. It is thus not surprising that the analysis requires tools from probability theory. We may regard the number of nodes of different degrees on the left and the right side of the graph as random variables that evolve over time. It is relatively easy to compute the conditional expectation of these random variables. This is done in the next subsection. What we need is a tool that asserts that these random variables do not deviate too much from their expected value. This is a typical example of a so-called sharp concentration result which we will derive in this subsection. We will assume that the reader is familiar with basic concepts such as (super- and sub-)martingales [12].

In the sequel we will follow [16] rather closely.

The evolution of the number of nodes of different degrees on the graphs considered is a typical example of a discrete time random process. Let Ω denote a probability space and S a measurable space. A *discrete time random process* over Ω with state space S is a sequence $Q = (Q_0, Q_1, \ldots)$ of random variables $Q_i : \Omega \to S$. To every $\omega \in \Omega$ corresponds a *realisation* $(Q_0(\omega), Q_1(\omega), \ldots)$ of the process. The *history* of the process up to time t is the sequence $H_t = (Q_0, Q_1, \ldots, Q_t)$. For a real-valued measurable function y defined on $S^+ := \cup_{i \geq 1} S^i$, the random variable $y(H_t)$ is denoted by Y_t.

We say that a function $f : \mathbb{R}^j \to \mathbb{R}$ satisfies a Lipschitz condition on $D \subseteq \mathbb{R}^j$ if there exists a constant $L > 0$ such that

$$|f(u) - f(v)| \leq L \cdot \sum_{i=1}^{j} |u_i - v_i|,$$

for all $u, v \in D$.

For a sequence of real-valued random variables X_m, we say that $X_m = O(f(m))$ *always*, if $\sup\{x \mid \mathbf{Pr}(X_m = x) \neq 0\} = O(f(m))$.

For the proof of our concentration result we need the following version of Azuma's Inequality [16, Lemma 1].

Theorem 1 (Azuma's Inequality) *Let $X_0, X_1, \ldots$ be a supermartingale with respect to a sequence of σ-algebras $\mathcal{F}_i$ with $\mathcal{F}_0 = \{\emptyset, \Omega\}$, and suppose that $X_0 = 0$ and $|X_{k+1} - X_k| \leq c$ for some constant c and for $i \geq 0$. Then for all $\alpha > 0$ we have*

$$\mathbf{Pr}(X_k \geq \alpha c) \leq \exp(-\alpha^2/2k).$$

Theorem 2 *Let $(Q^{(m)})_{m\geq 1}$ be a sequence of discrete time random processes $Q^{(m)} = (Q_0^{(m)}, Q_1^{(m)}, \ldots)$ over a probability space Ω with state space S_m and $H_t^{(m)} := (Q_0^{(m)}, \ldots, Q_t^{(m)})$ be the m-th history up to time t. Let d be a positive integer. For $1 \leq i \leq d$ and all positive integers m let $y^{(i,m)}: S_m^+ \to \mathbb{R}$ be a measurable function such that $|y^{(i,m)}(h)| < Cm$ for all $h \in S_m^+$ and for some constant C (independent of i, m, h). Furthermore let $f_1, \ldots, f_d$ be functions from $\mathbb{R}^{d+1}$ to $\mathbb{R}$. Suppose also that for some function $M: \mathbb{N} \to \mathbb{N}$ the following is satisfied:*

(i) *There exists a constant C' such that for all m, for all $t < M(m)$, and for all $i \leq d$:*

$$|Y_{t+1}^{(i,m)} - Y_t^{(i,m)}| < C',$$

where $Y_t^{(i,m)} := y^{(i,m)}(H_t^{(i,m)})$;

(ii) *for all i and uniformly over all (m,t) with $t < M(m)$ we have*

$$\mathbf{E}(Y_{t+1}^{(i,m)} - Y_t^{(i,m)} \mid H_t) = f_i(t/m, Y_t^{(1,m)}/m, \ldots, Y_t^{(d,m)}/m);$$

(iii) *for each $i \leq d$ the function f_i is continuous and satisfies a Lipschitz condition on D, where D is some bounded connected open set containing the intersection of $\{(t, z_1, \ldots, z_d) \mid t \geq 0\}$ with some open neighborhood $\{(0, z_1, \ldots, z_d) \mid \mathbf{Pr}(Y_0^{(i,m)} = z_i m \mid 1 \leq i \leq d) \neq 0$ for some $m\}$.*

Then the following holds.

(1) *For $(0, \zeta_1, \ldots, \zeta_d) \in D$ the system of differential equations*

$$\frac{\mathrm{d}z_i}{\mathrm{d}\tau} = f_i(\tau, z_1, \ldots, z_d), \quad i = 1, \ldots, d,$$

has a unique solution in D for $z_i: \mathbb{R} \to \mathbb{R}$ passing through $z_i(0) = \zeta_i$, $1 \leq i \leq d$.

(2) *There is a constant c such that*

$$\mathbf{Pr}(Y_t^{(i)} > m z_i(t/m) + cm^{5/6}) < m^{2/3} \exp(-\sqrt[3]{m}/2),$$

for $0 \leq t \leq \min\{\sigma m, M(m)\}$ and for each i, where $z_i(t)$ is the solution in (1) with $\zeta_i = \mathbf{E}(Y_0^{(i)})/m$, and $\sigma = \sigma(m)$ is the supremum of those τ to which the solution can be extended.

Proof. We will modify the proof in [16] slightly to obtain the error bounds asserted in the theorem. First, note that by a standard result in the theory of first order differential equations, there is a unique solution in (1).

As in [16] we will simplify the notation by considering $d = 1$ and referring to $y^{(1,m)}$, z_1, and f_1 as y, z, and f, and so on. The proof for general d is similar.

Let $w := \lceil m^{2/3} \rceil$, and assume that $0 \leq t \leq M - w$. We will first demonstrate concentration of $Y_{t+w} - Y_t$. Notice that the Lipschitz condition on f and Condition (ii) imply that for all $0 \leq k < w$

$$\mathbf{E}(Y_{t+k+1} - Y_{t+k} \mid H_{t+k}) = f\left(\frac{t+k}{m}, \frac{Y_{t+k}}{m}\right) \leq f\left(\frac{t}{m}, \frac{Y_t}{m}\right) + \gamma \frac{2k+1}{m}$$

for some constant γ.

For fixed t, define the random variable $X_k := Y_{t+k} - Y_t - kf(t/m, Y_t/m) - \gamma k^2/m$. Note that

$$X_{k+1} - X_k \quad = \quad Y_{t+k+1} - Y_{t+k} - f\left(\frac{t}{m}, \frac{Y_t}{m}\right) - \gamma \frac{2k+1}{m}.$$

This shows that the X_k form a supermartingale with respect to $H_t, \ldots, H_{t+w}$, as $0 \geq \mathbf{E}(X_{k+1} - X_k | H_{t+k}) = \mathbf{E}(X_{k+1}|H_{t+k}) - X_k$. Furthermore, the above equality shows that $|X_{k+1} - X_k| \leq C_2$ for some constant C_2. We can now apply Azuma's Inequality 1. As $X_0 = 0$, we obtain $\mathbf{Pr}(X_w \geq \alpha C_2) \leq \exp(-\frac{\alpha^2}{2w})$, for any $0 < \alpha$. (The parameter α will be chosen later.) The lower tail can be bounded in exactly the same way, using a submartingale. This gives for any constant B (to be chosen later)

$$\mathbf{Pr}\left(\left| Y_{t+w} - Y_t - wf\left(\frac{t}{m}, \frac{Y_t}{m}\right) \right| \geq (\gamma + B)\frac{w^2}{m} + \alpha C_2\right) \leq \exp\left(-\frac{\alpha^2}{2w}\right). \qquad (1)$$

Now let $k_\ell := \ell w$, where $\ell = 0, 1, \ldots, \ell_0$ and $\ell_0 = \lfloor \min\{m/w, \sigma m/w\} \rfloor$. Let

$$T_\ell := \frac{(\gamma w^2 + \alpha m C_2)((1 + Bw/m)^\ell - 1)}{Bw}.$$

We will prove by induction on ℓ that

$$\pi_\ell := \mathbf{Pr}(|Y_{k_\ell} - z(k_\ell/m)m| \geq T_\ell) \leq \ell \exp\left(-\frac{\alpha^2}{2w}\right).$$

The assertion is obvious for the induction start $\ell = 0$, as $z(0) = \mathbf{E}(Y_0)/m$. Define

$$\begin{aligned} A_1 &:= Y_{k_{\ell+1}} - Y_{k_\ell} - wf(k_\ell/m, Y_{k_\ell}/m), \\ A_2 &:= Y_{k_\ell} - mz(k_\ell/m), \\ A_3 &:= mz(k_{\ell+1}/m) - mz(k_\ell/m) - wf(k_\ell/m, Y_{k_\ell}/m). \end{aligned}$$

Note that

$$|Y_{k_{\ell+1}} - mz(k_{\ell+1}/m)| = |A_1 + A_2 - A_3| \leq |A_1| + |A_2| + |A_3|.$$

The inductive hypothesis gives that $|A_2| < T_\ell$ with probability at least $1 - \ell \exp(-\alpha^2/2w)$. Further, by 1 we have $|A_1| < \gamma w^2/m + \alpha C_2$ with probability at least $1 - \exp(-\alpha^2/2w)$. To bound A_3 we proceed as follows. By the mean value theorem we have that $z(k_{\ell+1}/m) - z(k_\ell/m) = wz'(\xi)/m$, where z' is the derivative of z and ξ is some real number with $k_\ell/m \leq \xi \leq k_{\ell+1}/m$. Note that z satisfies the differential equation in (1), hence $z'(\xi) = f(\xi, z(\xi))$, and by the Lipschitz condition on f we obtain $|A_3| \leq Lw(|k_\ell/m - \xi| + |Y_{k_\ell}/m - z(\xi)|)$. By the continuity of z and the inductive hypothesis, we see that for suitable choice of the constant B we have

$$|A_3| \leq B\left(\frac{w^2}{m} + \frac{wT_\ell}{m}\right),$$

for large enough m. Altogether we obtain

$$|A_1| + |A_2| + |A_3| \leq T_{\ell+1}$$

with probability at least $1 - \ell \exp(-\alpha^2/2w) - \exp(-\alpha^2/2w) = 1 - (\ell+1)\exp(-\alpha^2/2w)$. Now we choose $\alpha = \sqrt{m}$. Then $T_\ell \leq T_{m/w} \leq (\exp(B) - 1)((\gamma + B)m^{4/3} + m^{3/2}C_2)/(Bm^{2/3}) = O(m^{5/6})$ for all ℓ. Hence, we see that (2) is satisfied at $t = k_\ell$ with probability at least $1 - m^{2/3}\exp(-\sqrt[3]{m}/2)$. Furthermore, as $|Y_t - Y_{k_\ell}| \leq C' m^{2/3}$ for all $k_\ell \leq t \leq k_{\ell+1}$, we contend that $Y_t = mz(t/m) + O(m^{5/6})$ for all t in the specified range, with probability at least $1 - m^{2/3}\exp(-\sqrt[3]{m}/2)$. □

We remark that one can have better choices for α and w in the above proof which make the error terms smaller, at the expense of making the error probability slightly larger.

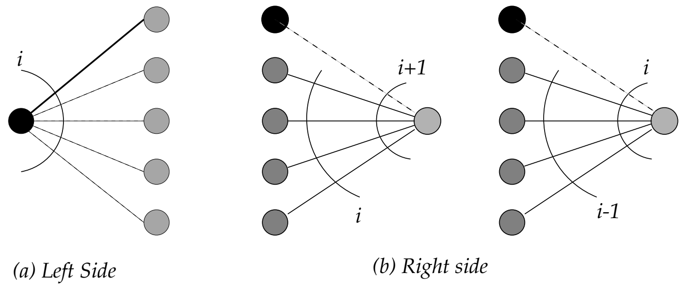

Figure 4: Description of differential equations

3.2 The Differential Equations

We begin with the initial random graph B, with k left nodes and βk right nodes. Consider the two vectors (λ_i) and (ρ_i), where λ_i and ρ_i are the fractions of edges of degree i on the left and right, with respect to the total number E of edges in the original graph. The average node degree on the left a_ℓ initially satisfies $a_\ell^{-1} = \sum_i \lambda_i/i$, and similarly the average node degree on the right a_r initially satisfies $a_r^{-1} = \sum_i \rho_i/i$.

In the application of Theorem 2 each time step corresponds to recovering one node on the left hand side. Furthermore, the parameter m corresponds to the total number E of edges. Let δ be the fraction of losses in the message. Initially, just prior to time 0, each node on the left is removed with probability $1-\delta$ (because the corresponding message bit is successfully received), and thus the initial subgraph of B contains δk nodes on the left. If the process terminates successfully, it runs until time $\delta k = E\delta/a_\ell$. As in the last subsection, we denote by τ the scaled time t/E. Hence, τ runs from 0 to δ/a_ℓ.

Let G be the graph obtained after a random deletion of $(1-\delta)k$ nodes on the left. We let Q_t be the t-th edge removed in the graph G, and by G_t the graph obtained after removing $Q_1,\ldots,Q_t$, all left nodes they are connected to, and all edges coming out of these nodes. If the process has already stopped at time $t-1$, we set $G_t = G_{t-1}$ for convenience.

We denote by $\mathcal{L}_t^{(i)}$ the number of edges of left degree i at time t, and by $\mathcal{R}_t^{(i)}$ the number of edges of right degree i at time t. Let E be the total number of edges in the original graph. We let $\ell_t^{(i)} := \mathcal{L}_t^{(i)}/E$ and $r_t^{(i)} := \mathcal{R}_t^{(i)}/E$ represent the fraction of edges (in terms of E) of degree i on the left and right, respectively, at time t. We denote by e_t the fraction of the edges remaining at time t, that is, $e_t = \sum_i \ell_t^{(i)} = \sum_i r_t^{(i)}$.

First we note that $|\mathcal{L}_{t+1}^{(i)} - \mathcal{L}_t^{(i)}| \le 1$ for all i and t, so Condition (i) in Theorem 2 is satisfied. Recall that at each step, a random node of degree one on the right is chosen, and the corresponding node on the left and all of its adjacent edges are deleted. (If there is no such node, the process necessarily stops.) The probability that the edge adjacent to the node of degree one on the right has degree i on the left is $\ell_t^{(i)}/e_t$, and in this case we lose i edges of degree i, see Figure 4(a). Hence, we have

$$\mathbf{E}(\mathcal{L}_{t+1}^{(i)} - \mathcal{L}_t^{(i)} \mid H_t) = -\frac{i\ell_t^{(i)}}{e_t},$$

for $i = 1,\ldots,d$, where d is the maximum degree on the left hand side.

Abusing the notation slightly, we set

$$f_i(s,\ell_1,\ldots,\ell_d) := -\frac{i\ell_i}{e},$$

where $e = \sum_j \ell_j$, and see that Condition (ii) of Theorem 2 is satisfied for these functions. Furthermore, fixing $\eta > 0$, we further see that these functions satisfy Condition (iii) in the domain D defined by the inequalities $0 < s < \delta + \eta$, $0 < \ell_i < 1 + \eta$, for all $i = 1, \ldots, d$, and $\eta < e < \delta + \eta$. Now Theorem 2 implies that with high probability we have $\mathcal{L}_t^{(i)} = E\ell_i(t/E) + O(E^{5/6})$ uniformly for all $E\eta \le t \le (\delta + \eta)E$, where $\ell_i(\tau)$ form the solution to

$$\frac{\mathrm{d}\ell_i(\tau)}{\mathrm{d}\tau} = -\frac{i\ell_i(\tau)}{e} \tag{2}$$

for $i = 1, \ldots, d$.

To discuss the evolution of the right hand side, first note that $|\mathcal{R}_{t+1}^{(j)} - \mathcal{R}_t^{(j)}| < d$, where d is the maximum degree on the left. This is because a left node is connected to at most d right nodes and one of the right neighbors has been used to recover the left node. Hence, Condition (i) of the theorem is satisfied. Note that when we remove a node of degree i on the left, we remove the one edge of degree one from the right, along with the $i - 1$ other edges adjacent to this node. Hence the expected number of other edges deleted is $a_t - 1$, where $a_t = \sum i\ell_t^{(i)}/e_t$. The right endpoints of these $i - 1$ other edges on the right hand side are randomly distributed. If one of these edges is of degree j on the right, we lose j edges of degree j, and gain $j - 1$ edges of degree $j - 1$, see Figure 4(b). The probability that an edge has degree j on the right is just $r_t^{(j)}/e_t$. Then, for $i > 1$, we have

$$\mathbf{E}(\mathcal{R}_{t+1}^{(i)} - \mathcal{R}_t^{(i)} \mid H_t) = (r_t^{(i+1)} - r_t^{(i)})\frac{i(a_t - 1)}{e_t}$$

(We assume that $r_t(i)$ is defined for all positive i, and is 0 for sufficiently large i.) The case $i = 1$ plays a special role, as we must take into account that at each step an edge of degree one on the right is removed. This gives

$$\mathbf{E}(\mathcal{R}_{t+1}^{(1)} - \mathcal{R}_t^{(1)} \mid H_t) = (r_t^{(2)} - r_t^{(1)})\frac{(a_t - 1)}{e_t} - 1.$$

Let μ be the maximum degree of a node on the right. Abusing the notation slightly, we set

$$\begin{aligned}
g_\mu(\tau, r_1, \ldots, r_\mu) &:= -r_\mu \frac{\mu(a-1)}{e} \\
g_i(\tau, r_1, \ldots, r_\mu) &:= (r_{i+1} - r_i)\frac{i(a-1)}{e} \quad \text{for } 1 < i < \mu, \\
g_1(\tau, r_1, \ldots, r_\mu) &:= (r_2 - r_1)\frac{(a-1)}{e} - 1,
\end{aligned}$$

where $e = \sum_j r_j$ and $a := \sum_j j\ell_j/e$. Fixing any $\eta > 0$, we see as in the case of the left hand side evolution, that these functions satisfy a Lipschitz condition, as long as $\tau = t/E > \eta$. Application of Theorem 2 thus yields that almost surely, we have $\mathcal{R}_t^{(i)} = Er_i(t/E) + O(E^{5/6})$ uniformly for all $\eta E \le t \le (\delta + \eta)E$, where $r_i(\tau)$ form the solution to

$$\frac{\mathrm{d}r_i(\tau)}{\mathrm{d}t} = (r_{i+1}(\tau) - r_i(\tau))\frac{i(a(\tau) - 1)}{e(\tau)} \qquad \text{for } i > 1, \tag{3}$$

and

$$\frac{\mathrm{d}r_1(\tau)}{\mathrm{d}\tau} = (r_2(\tau) - r_1(\tau))\frac{(a(\tau) - 1)}{e(\tau)} - 1. \tag{4}$$

Our key interest is in the progression of $r_1(\tau)$ as a function of τ. As long as $r_1(\tau) > 0$, so that we have a node of degree one on the right, the process continues; when $r_1(\tau) = 0$ the process stops.

Hence we would like $r_1(\tau) > 0$ until all nodes on the left are deleted and the process terminates successfully.

These differential equations are more easily solved by defining x so that $\mathrm{d}x/\mathrm{d}\tau = -x/e(\tau)$. The value of x in terms of τ is then $x := \exp(-\int_0^\tau \mathrm{d}s/e(s))$. By substituting $\mathrm{d}x/x$ for $\mathrm{d}\tau/e(\tau)$, Equation 2 becomes $\mathrm{d}\ell_i(x)/\mathrm{d}x = i\ell_i(x)/x$, and integrating yields $\ell_i(x) = c_i x^i$. Note that $x = 1$ for $t = 0$, and $\ell_i(\tau = 0) = \delta\lambda_i$. Hence, $c_i = \delta\lambda_i$ and

$$\ell_i(x) = \delta\lambda_i x^i.$$

Since $\ell_i(x)$ goes to zero as τ goes to δ/a_ℓ, x runs over the interval $(0, 1]$.

Solving for the $r_i(\tau)$ is more involved.

We will express the result using the degree sequence functions

$$\lambda(x) := \sum_{i\geq 1} \lambda_i x^{i-1}, \tag{5}$$

$$\rho(x) := \sum_{i\geq 1} \rho_i x^{i-1}. \tag{6}$$

Again, we start with the substitution $x := \exp(-\int_0^\tau \mathrm{d}s/e(s))$. This transforms for $i > 1$ Equation 3 into

$$r_i'(x) = i(r_{i+1}(x) - r_i(x))\frac{a(x) - 1}{x},$$

where prime stands for derivative with respect to the variable x. Note that the average degree $a(x)$ equals $\sum i\ell_i(x)/e(x)$, which in terms of the function $\lambda(x)$ can be written as $1 + x\lambda'(x)/\lambda(x)$. Hence, we obtain for $i > 1$

$$r_i'(x) = i(r_{i+1}(x) - r_i(x))\frac{\lambda'(x)}{\lambda(x)}.$$

These equations can be solved recursively, starting with the highest nonzero r_i.

The explicit solution is given by

$$r_i(x) = \lambda(x)^i\Big(-i\int_1^x r_{i+1}(y)\lambda(y)^{-i}\frac{\lambda'(y)}{\lambda(y)}\mathrm{d}y + c_i\Big) \tag{7}$$

for some constants c_i to be determined from the initial conditions for r_i. It is straightforward to check that

$$r_i(x) = \sum_{j\geq i}(-1)^{i+j}\binom{j-1}{i-1}c_j\lambda(x)^j. \tag{8}$$

Further, one verifies directly that

$$c_i = \sum_{j\geq i}\binom{j-1}{i-1}r_j(1).$$

(Note that $x = 1$ corresponds to the beginning of the process.) We proceed with the determination of $r_j(1)$, the expected fraction of edges of right degree one at time 0: because each node on the left is deleted randomly just prior to time 0 with probability $1 - \delta$, and the graph is a random graph over those with the given degree sequence, to the nodes on the right it is as though each edge is deleted with probability $1 - \delta$. Hence, an edge whose right incident node had degree j before the deletion stage remains in the graph and has degree i afterwards with probability $\binom{j-1}{i-1}\delta^i(1-\delta)^{j-i}$. Thus,

$$r_j(1) = \sum_{m\geq j}\rho_m\binom{m-1}{j-1}\delta^j(1-\delta)^{m-j}. \tag{9}$$

Plugging in the last formula in that of c_i we see that

$$c_i = \sum_{m \geq i} \binom{m-1}{i-1} \rho_m \delta^i.$$

(Use $\binom{m-1}{j-1}\binom{j-1}{i-1} = \binom{m-1}{i-1}\binom{m-j}{j-i}$.) Hence, we obtain for $i > 1$ from 8

$$r_i(x) = \sum_{m \geq j \geq i} (-1)^{i+j} \binom{j-1}{i-1} \binom{m-1}{j-1} \rho_m (\delta\lambda(x))^j. \tag{10}$$

To obtain the formula for $r_1(x)$, we note that $r_1(x) = e(x) - \sum_{i>1} r_i(x)$. The sum of the right hand side of 10 over all $i \geq 1$ equals

$$\sum_{m \geq j} (-1)^{j-1} \binom{m-1}{j-1} \rho_m (\delta\lambda(x))^j \sum_{i \leq j} (-1)^{i-1} \binom{j-1}{i-1}$$
$$= \delta\lambda(x) \quad .$$

(The inner sum equals 1 if $j = 1$, and is zero otherwise.) Hence, we have

$$\begin{aligned} r_1(x) &= e(x) - \delta\lambda(x) \\ &\quad + \delta\lambda(x) \sum_m \rho_m \sum_{j \leq m} (-1)^{j-1} \binom{m-1}{j-1} (\delta\lambda(x))^{j-1} \\ &= x\delta\lambda(x) - \delta\lambda(x) + \delta\lambda(x) \sum_m \rho_m (1 - \delta\lambda(x))^{m-1} \\ &= \delta\lambda(x) \Big[x - 1 + \rho(1 - \delta\lambda(x)) \Big]. \ \square \end{aligned}$$

We thus see that the solution $r_1(x)$ of the differential equation 4 satisfying the initial condition 9 is given by

$$r_1(x) \quad = \quad \delta\lambda(x) \left[x - 1 + \rho(1 - \delta\lambda(x)) \right]. \tag{11}$$

This immediately gives rise to the following result.

Proposition 1 *Let B be a bipartite graph with k message bits, and chosen at random with edge-degrees specified by $\lambda(x)$ and $\rho(x)$. Let δ be fixed so that*

$$\rho(1 - \delta\lambda(x)) > 1 - x, \qquad \text{for } x \in (0, 1].$$

For all $\eta > 0$ there is some k_0 such that for all $k \geq k_0$, if the message bits of $\mathcal{C}(B)$ are lost independently with probability δ, then with probability at least $1 - k^{2/3} \exp(-\sqrt[3]{k}/2)$ the recovery algorithm terminates with at most ηk message bits not recovered.

Proof. Let E be the number of edges in the graph. Then $E = ka_\ell$, where a_ℓ is the average degree of the nodes in the left side of the graph, which is a constant for fixed λ and ρ. (Note that $a_\ell = \sum \lambda_i / i$.) Let $\mu := \eta / a_\ell$. By 11 and the preceding discussions, with probability at least $1 - k^{2/3} \exp(-\sqrt[3]{k}/2)$ the number of nodes of degree one on the right is,

$$\delta\lambda(x) \left[x - 1 + \rho(1 - \delta\lambda(x)) \right] + O(k^{5/6}),$$

for $x \in (\eta', 1]$, where $\eta' = \exp(-\int_0^\mu \mathrm{d}s/e(s))$. By our assumption, this number is positive (for large enough k), which proves the assertion. □

The foregoing proposition does not prove that the decoding process terminates successfully recovering all the missing nodes on the left hand side. To do this, we need a combinatorial argument which deals with expansion properties of random graphs.

Lemma 1 *Let B be a bipartite graph with k left nodes chosen at random with edge-degrees specified by $\lambda(x)$ and $\rho(x)$, such that $\lambda(x)$ has $\lambda_1 = \lambda_2 = 0$. Then there is some $\eta > 0$, such that, with probability $1 - O(k^{-3/2})$, the recovery process restricted to the subgraph induced by any η-fraction of the left nodes terminates successfully.*

PROOF. Let S be any set of nodes on the left of size at most ηk, where η will be chosen later. Let a be the average degree of these nodes. If the number of nodes on the right that are neighbors of S is greater than $a|S|/2$, then one of these nodes has only one neighbor in $|S|$, and so the process can continue. Thus, we only need to show that the initial graph is a good expander on small sets.

Let $\mathcal{E}_s$ denote the event that a subset of size s of the nodes on the left has at most $as/2$ neighbors. We will first bound $\mathbf{Pr}(\mathcal{E}_s)$, and then sum $\mathbf{Pr}(\mathcal{E}_s)$ over all values of s no larger than ηk. Fix any subset S of the left nodes of size s, and any subset T of the right nodes of size $as/2$. There are $\binom{k}{s}$ ways of choosing S, and $\binom{k}{as/2}$ ways of choosing T. The probability that T contains all the as neighbors of the vertices in S is $(as/2\beta k)^{as}$. Hence, we have

$$\mathbf{Pr}(\mathcal{E}_s) \leq \binom{k}{s}\binom{\beta k}{as/2}\left(\frac{as}{2\beta k}\right)^{as}.$$

Note that $\binom{n}{k} \leq (ne/k)^k$, hence we have

$$\mathbf{Pr}(\mathcal{E}_s) \leq \left(\frac{s}{k}\right)^{(a/2-1)s} c^s \leq \left(\frac{sc^2}{k}\right)^{s/2},$$

where c is a constant (depending on β and a). Since the graph does not have nodes of degree one or two, we have that $\mathbf{Pr}(\mathcal{E}_1) = \mathbf{Pr}(\mathcal{E}_2) = 0$. Choosing $\eta \leq 1/(2c^2)$ yields

$$\sum_{s=1}^{\eta k}\left(\frac{sc^2}{k}\right)^{s/2} = \sum_{s=3}^{\eta k}\left(\frac{sc^2}{k}\right)^{s/2} \leq \frac{3c^2}{k\sqrt{k}} + \sum_{s=4}^{\eta k}\frac{1}{2^s} = O\left(\frac{1}{k\sqrt{k}}\right),$$

which shows that, with high probability, the original graph is an expander on small subsets. □

The above proof shows that the main contribution for the error-probability comes from nodes of degree three on the left. For the same reason, it is easy to see that nodes of degree 2 will lead to a constant error probability. We leave the details of this argument to the reader.

Altogether we obtain the main theorem of this section.

Theorem 3 *Let k be an integer, and suppose that $\mathcal{C} = \mathcal{C}(B_1, \ldots, B_m, C)$ is a cascade of bipartite graphs as explained in Section 2, where B_1 has k left nodes. Suppose that each B_i is chosen at random with edge-degrees specified by $\lambda(x)$ and $\rho(x)$, such that $\lambda(x)$ has $\lambda_1 = \lambda_2 = 0$, and suppose that δ is such that*

$$\rho(1 - \delta\lambda(x)) > 1 - x, \tag{12}$$

for all $0 < x \leq 1$. Then, if at most a δ-fraction of the coordinates of an encoded word in $\mathcal{C}$ are lost independently at random, then the loss recovery process given in 1 terminates successfully, with probability $1 - O(k^{-3/4})$, and does so in $O(k)$ steps.

PROOF. At each level of the cascade the number of edges equals the average degree of the nodes on the left times the number of the nodes. The average degree is always $1/\int_0^1 \lambda(t)\mathrm{d}t$, which is a constant. Hence, the total number of edges in the the cascade (up to the last layer) is $O(k)$, which shows that the recovery process needs $O(k)$ steps. (See Remark 1.)

Next we bound the probability that there is some j such that the fraction of left nodes lost on the left side of the graph B_j is larger than $\delta' := \delta + 1/\sqrt[8]{k}$. We use a version of the Chernoff bounds given in [12, Prob. 4.7(c), pp. 98]. According to that, for any j the probability that there are more losses than $\delta(k/2^j) + (k/2^j)^{3/4}$ is upper bounded by $\exp(-2\sqrt{k/2^j})$, which is smaller than $\exp(-2\sqrt[4]{k})$. The required probability is certainly at most equal to the sum of these probabilities (union bound), which is $\log(k)\exp(-\sqrt[4]{k})/2$. (Note that there are $\log(k)/2$ such j's).

For large enough k Condition 12 is satisfied for δ' instead of δ (by continuity). Hence, invoking Proposition 1, for any $\eta > 0$ and any of the graphs B_j in the cascade our recovery algorithm 1 stops with less than $\eta k/2^j$ nodes unrecovered, with probability $1 - O(\exp(-k^\gamma))$ for some positive γ. Now Lemma 1 applies and shows that, for small enough η, the recovery process ends successfully with probability $1 - O((2^j/k)^{3/2})$. The probability that our algorithm fails on at least one of the graphs is thus at most $\sum_j (2^j/k)^{3/2}$, where j runs from 0 to $\log(k)/2$. This is equal to $O(k^{-3/4})$, which shows the assertion. $\square$

For designing graphs that lead to good erasure codes, it is thus necessary to fulfill Condition 12. It is sometimes desirable to use the "dual condition"

$$\delta\lambda(1 - \rho(y)) < 1 - y, \tag{13}$$

for $y \in [0, 1)$, which is obtained from 12 by substituting $y := \rho^{-1}(1 - x)$. Note that ρ has an inverse on $(0, 1]$, as it is monotonically increasing.

In the next section we will use this theorem to analyze decoding properties of codes obtained from regular graphs.

4 Asymptotically Good Codes

In this section, we construct codes that are close to optimal in terms of their erasure recovery rate, and have linear time encoding and loss recovery algorithms. We do this by finding an infinite sequence of solutions to the differential equations of Section 3 in which δ is close to $1 - R$, where R is the rate.

Let B be a bipartite graph with k left nodes and βk right nodes. We describe our choice for the left and right degree sequences of B that satisfy Condition 12. Let D be a positive integer that is used to trade off the average degree with how well the decoding process works, i.e., how close we can make δ to $\beta = 1 - R$ and still have the process finish successfully most of the time.

The left degree sequence is described by the following truncated heavy tail distribution. Let $H(D) = \sum_{i=1}^{D} 1/i$ be the harmonic sum truncated at D, and thus $H(D) \sim \ln(D)$. Then, for all $i = 2, \ldots, D+1$, the fraction of edges of degree i on the left is given by $\lambda_i := 1/(H(D)(i-1))$. The average left degree a_ℓ equals $H(D)(D+1)/D$. Recall that we require the average right degree, a_r, to satisfy $a_r = a_\ell/\beta$. The right degree sequence is defined by the Poisson distribution with mean a_r: for all $i \geq 1$ the fraction of edges of degree i on the right equals

$$\rho_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!},$$

where α is chosen to guarantee that the average degree on the right is a_r. In other words, α satisfies $\alpha e^\alpha/(e^\alpha - 1) = a_r$.

Note that we allow $\rho_i > 0$ for all $i \geq 1$, and hence $\rho(x)$ is not truly a polynomial, but a power series. However, truncating the power series $\rho(x)$ at a sufficiently high term gives a finite distribution of the edge degrees for which the next lemma is still valid.

We show that when $\delta = \beta(1 - 1/D)$, then the condition is satisfied, i.e., $\rho(1 - \delta\lambda(x)) > 1 - x$ on $(0, 1]$, where $\lambda(x) = \sum_i \lambda_i x^{i-1}$ and $\rho(x) = \sum_i \rho_i x^{i-1}$. Note that $\lambda(x)$ is the expansion of $-\ln(1-x)$ truncated at the Dth term, and scaled so that $\lambda(1) = 1$. Further, $\rho(x) = e^{\alpha(x-1)}$.

Lemma 2 *With the above choices for $\rho(x)$ and $\lambda(x)$ we have $\rho(1 - \delta\lambda(x)) > 1 - x$ on $(0, 1]$ if $\delta \leq \beta/(1 + 1/D)$.*

Proof. Since $\rho(x)$ increases monotonically in x, we have

$$\rho(1 - \delta\lambda(x)) > \rho(1 + \delta \ln(1 - x)/H(D)) = (1 - x)^{\alpha\delta/H(D)}.$$

As $a_\ell = H(D)(1 + 1/D)$ and $a_r = a_\ell/\beta$, we obtain $\alpha\delta/H(D) = (1 - e^{-\alpha})(1 + 1/D)\delta/\beta < \delta(1 + 1/D)/\beta \leq 1$, which shows that the right hand side of the above inequality is larger than $1 - x$ on $(0, 1]$. □

A problem is that Lemma 1 does not apply to this system because there are nodes of degree two on the left. Indeed, simulations demonstrate that for these choices of $\lambda(x)$ and $\rho(x)$ a small number of nodes often do remain. To overcome this problem, we make a small change in the structure of the graph B. Let $\gamma := \beta/D^2$. We split the βk right nodes of B into two distinct sets, the first set consisting of $(\beta - \gamma)k$ nodes and the second set consisting of γk nodes. The graph B is then formed by taking the union of two graphs, B_1 and B_2. B_1 is formed as described up to this point between the k left nodes and the first set of $(\beta - \gamma)k$ right nodes. B_2 is formed between the k left nodes and the second set of γk right nodes, where each of the k left nodes has degree three and the $3k$ edges are connected randomly to the γk right nodes.

Lemma 3 *Let B be the bipartite graph just described. Then, with probability $1 - O(k^{-3/2})$, the decoding process terminates successfully when started on a subgraph of B induced by δk of the left nodes and all βk of the right nodes, where $\delta = \beta(1 - 1/D)$.*

Proof. In the analysis of the process, we may think of B_2 as being held in reserve to handle nodes not already dealt with using B_1. First, using the same method as in Lemma 1 we can prove that there is some η such that an set S of $s \leq \eta k$ left nodes in the graph B_2 expands to a set of at least $3s/2$ nodes on the right, with probability $1 - O(1/k^{3/2})$. (Note that all nodes on the left have degree three in this graph.) Combining Proposition 1 and Lemma 2, we see that the recovery process started on B_1 terminates with less than ηk nodes on the left unrecovered, with probability $1 - O(\exp(-k^a))$ for some positive a: note that the ratio of the number of left nodes to the number of right nodes in the graph B_2 equals $\beta(1 - 1/D^2)$, hence the condition in Lemma 2 translates to $\delta \leq \beta(1 - 1/D^2)/(1 + 1/D) = \beta(1 - 1/D)$, which is obviously true. By the aforementioned expansion property of the subgraph of B_2 induced by the set of left nodes unrecovered, we see that the process terminates successfully. □

Note that the degree of each left node in this modified construction of B is at most three bigger than the average degree of each left node in the construction of B described at the beginning of this section. We can use this observation and the lemma above to immediately prove the following.

Theorem 4 *For any R with $0 < R < 1$, any ϵ with $0 < \epsilon < 1$, and sufficiently large block length n, there is a loss resilient code that, with probability $1 - O(n^{-3/4})$, is able to recover from the random loss of a $(1 - R)(1 - \epsilon)$-fraction of its bits in time proportional to $n \ln(1/\epsilon)$.*

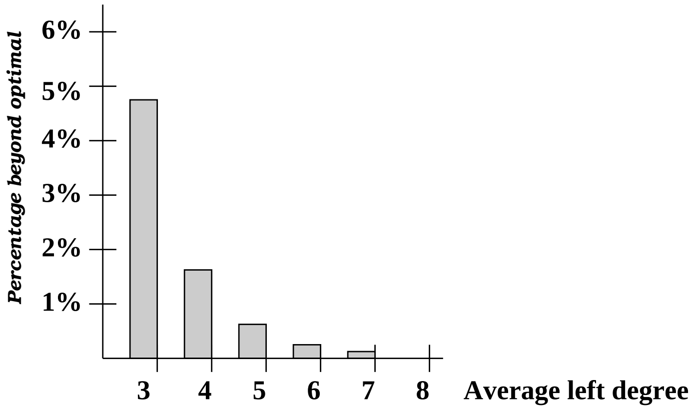

Figure 5: Loss recovery of irregular graphs: left degree versus the overhead beyond optimal value needed to recover (rate = 1/2)

PROOF. Set $D = \lceil 1/\epsilon \rceil$ to get a one level code with the properties described in Lemma 3. Cascade versions of these codes as described in Section 2 to get the entire code. As was pointed out above, the average degree a_ℓ of the left nodes in each of the layers is upper bounded by $3 + \sum_{i=1}^{D} 1/i < 4 + \ln(1/\epsilon)$, hence is proportional to $\ln(1/\epsilon)$ for fixed ϵ. Hence, the total number of edges in the bipartite layers of the graph is proportional to $n \ln(1/\epsilon)$, which proves the assertion on the decoding time.

Using Lemma 3 and the same analysis as in the proof of Theorem 3, we can show that the code constructed above can recover, with probability $1 - O(n^{-3/4})$, all the message bits, if a random δ-fraction of the encoding is missing, where $\delta = \beta(1 - 1/D)$. Noting that $\beta = 1 - R$, we obtain the result. □

Figure 5 shows the running time of the encoding/recovery algorithms (as multiples of the block length) versus the overhead needed to recover. For instance, for large n, one can construct in this way $[n,k]_q$-codes that have encoding/recovery algorithms running in time $\sim 7n$, which can recover a codeword from a random set of $1.002k$ of its coordinates.

5 Practical Considerations

The discussions in the preceding sections have been of a more theoretical, rather than a practical nature. However, as the graph codes designed via the above mentioned theorems can be used in practical situations, it is important to describe possible implementations. That's what we are going to do in this section. We will start with modifying our construction by allowing losses to also occur on the right hand side. An analysis of this type provides us with some insight in how to design cascaded versions of our codes with much fewer levels, and faster decoding algorithms for the end level. Next, we will show how to use a linear programming approach to design bipartite graphs which give rise to very good codes. Finally, we will briefly discuss some of our implementations.

5.1 Fraction of Left Nodes Unrecovered

So far we have assumed in our analysis that in each layer of the cascade all the check bits are received when trying to recover the message bits. The reason we made this assumption is that in the original construction the cascading sequence of bipartite graphs is completed by adding a standard loss resilient code at the last level.

There are some practical problems with this. One annoyance is that it is inconvenient to combine two different types of codes. A more serious problem is that standard loss resilient codes take quadratic time to encode and decode. Suppose the message is stretched to an encoding twice its length. In order to have the combined code run in linear time, this implies that the last graph in the cascading sequence has $\sqrt{n}$ left nodes, where n is the number of nodes associated with the original message, i.e., there are $\log(n)/2$ graphs in the sequence. In the analysis, we assume that an equal fraction of the nodes in each level of the graph are received. However, there is variance in this fraction at each level, with the worst expected fractional variance at the last level of $1/\sqrt[4]{n}$. Thus, if a message of length $65,536$ is stretched to an encoding of length $131,072$, then just because of the variance of $1/\sqrt[4]{n} = 0.063$, we expect to have to receive 1.063 times the message length of the encoding in order to recover the message.

A solution to this problem is to use many fewer levels of graphs in the cascade, and at the last level also use a random graph in place of a standard loss resilient code. We have tried this idea, with the last graph chosen from an appropriate distribution, and it works quite well. For example, using only three levels of graphs we can reliably recover a message of length $65,536$ from a random portion of length $67,700$ (i.e., 1.033 times the optimal of $65,536$) of an encoding of length $131,072$.

To design the graph for this solution, we need to analyze the decoding process when a random portion of both the message bits and the check bits are missing.

Again, we begin with the initial random graph B, with k left nodes and βk right nodes, and continue to work with the generating functions $\lambda(x)$ and $\rho(x)$ from Section 3. Suppose that each node on the right is removed with probability δ', while nodes on the left are removed with probability δ. The new process can now be studied as a process with losses on the left only, which runs on the subgraph $\tilde{B}$ of the initial consisting of the $(1-\delta')\beta n$ undeleted nodes on the right. Let $\tilde{\lambda}_i$ be the fraction of edges of degree i in $\tilde{B}$ with respect to the total number of edges in $\tilde{B}$. Define $\tilde{\rho}$ similarly. Obviously, $\tilde{\rho}_i = \rho_i$, as the number of edges of degree i *and* the number of total edges in $\tilde{B}$ are a $(1-\delta')$-factor of those of B. As for $\tilde{\lambda}_i$, it is easily seen that

$$\tilde{\lambda}_i = \sum_{j\geq i} \lambda_j \binom{j-1}{i-1}(1-\delta')^i(\delta')^{j-i}.$$

This is done as follows: an edge of degree j is with probability $1-\delta'$ connected to an undeleted node on the right. The probability that $j-i$ of the remaining $j-1$ edges is connected to one of the deleted nodes on the right is exactly a $(1-\delta')$-fraction of the above sum.

The above formula shows that $\tilde{\lambda}(x) = \lambda(\delta' + (1-\delta')x)$. Invoking Theorem 2 we see that the expected number of edges of right degree one at time x (with respect to the total number of edges in $\tilde{B}$) equals

$$\delta\lambda(\delta' + (1-\delta')x)\left[x - 1 + \rho(1 - \delta\lambda(\delta' + (1-\delta')x))\right].$$

Since the number of edges in $\tilde{B}$ is $(1-\delta')$ times the number of edges in B, we obtain the following result.

Lemma 4 *The fraction $r_1(x)$ of edges of right degree one at x with respect to the number of edges in the original graph B equals*

$$r_1(x) = \delta(1-\delta')\lambda(\delta' + (1-\delta')x)\left[x - 1 + \rho(1 - \delta\lambda(\delta' + (1-\delta')x))\right].$$

This result implies the condition

$$\rho(1 - \delta\lambda(\delta' + (1-\delta')x)) > 1 - x \qquad x \in (0,1] \tag{14}$$

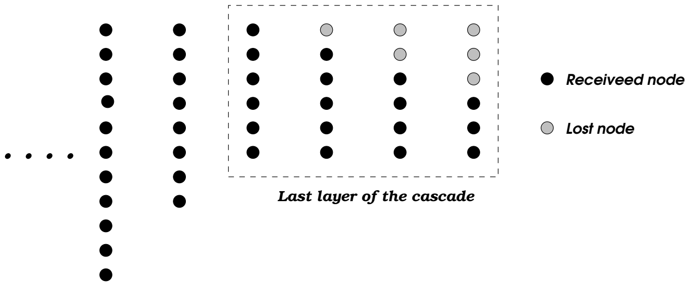

Figure 6: Bootstrapping strategy

for successful decoding.

The above inequality is not possible to satisfy for all $x \in (0,1]$ if $\delta' > 0$, for any value of δ: for $x = 0$ the left hand side equals $\rho(1 - \delta\lambda(\delta'))$ which is strictly less than 1. There is an intuitive reason for this: the subgraph $\tilde{B}$ on which the process starts will have edges of degree one on the left; these edges can only correct the left nodes they are connected to, and cannot help any other node on the left.

However, it turns out to be an interesting question to see what fraction of the left nodes can be recovered when a fraction δ' of the right nodes is missing. The answer to this question can be used to design cascading codes where the decoding process moves from right to left bootstrapping up to recover a higher and higher fraction of nodes at each successive decoded layer of the graph until it is able to recover all of the first (message) layer. (See Figure 6.)

Given the (λ) and (ρ) vectors, Condition 14 can be used to compute the smallest value of x for which the condition is still valid. In this section, we show how to compute the fraction of unrecovered nodes on the left at this value of x.

We retain the notation established earlier, e.g., $e(x)$ is the fraction of the original edges remaining at x. Let E be the number of edges in the original graph, N be the number of left nodes in the original graph, and thus the average left node degree is $a_\ell = E/N$. We define $b(x)$ to be the average node degree among nodes on the left that have at least one edge at x.

We define f_i to be the fraction of left nodes of degree i in the original graph, and thus $f_i = a_\ell \cdot \lambda_i / i$. We define $f(x)$ to be the expected fraction of original left nodes still not recovered at x. We define $\tilde{f}$ to be the fraction of left nodes that have all their neighbors among the original δ' fraction of missing right nodes. We define $\hat{f}(x)$ to be the expected fraction of left nodes that have at least one neighbor *not* among the original δ' fraction of missing right nodes and that are still not recovered at x.

One can verify that $f(x) = \delta\tilde{f} + \hat{f}(x)$, and that $\tilde{f} = \sum_i f_i(\delta')^i$. Thus, our goal is to deduce a closed form expression for $\hat{f}(x)$. The number of unrecovered left nodes with at least one neighbor at x is equal to the number of edges remaining at x divided by $b(x)$. The number of edges at x is $e(x)E$, and thus,

$$\hat{f}(x) = \frac{e(x)E}{b(x)N} = a_\ell \cdot e(x)/b(x).$$

We now turn to $b(x)$. It can be verified that

$$b(x) = \frac{e(x)}{\int_0^x e(y)/y \mathrm{d}y}.$$

From this it follows that $\hat{f}(x) = a_\ell \cdot \int_0^x e(y)/y \mathrm{d}y$. Recall that $e(y) = \delta(1-\delta')y\lambda(\delta' + (1-\delta')y)$, and thus $e(y)/y = \delta(1-\delta')\lambda(\delta' + (1-\delta')y)$. Further, recall that $\lambda(z) = \sum_i \lambda_i z^{i-1}$. Thus,

$$\int_0^x \lambda(\delta' + (1-\delta')y)\mathrm{d}y = \sum_i \frac{\lambda_i}{i} \cdot \frac{(\delta' + (1-\delta')y)^i}{1-\delta'} \Bigg|_0^x .$$

Thus,

$$\begin{aligned}\int_y^x e(y)/y\mathrm{d}y &= \delta \sum_i \frac{\lambda_i}{i} \cdot (\delta' + (1-\delta')y)^i \Bigg|_0^x \\ &= \frac{\delta}{a_\ell} \sum_i f_i \cdot (\delta' + (1-\delta')y)^i \Bigg|_0^x .\end{aligned}$$

This implies

$$\hat{f}(x) = \delta \Big[\sum_i f_i \cdot (\delta' + (1-\delta')x)^i - \tilde{f} \Big].$$

Finally, $f(x) = \hat{f}(x) + \delta \tilde{f} = \delta \sum_i f_i \cdot (\delta' + (1-\delta')x)^i$. This shows that the fraction of nodes unrecovered at time x is, up to small terms, equal to

$$\delta a_\ell \cdot \int_0^{(\delta' + (1-\delta')x)} \lambda(y)\mathrm{d}y.$$

5.2 Finding Degree Sequences Using Linear Programming

In this section we describe a heuristic approach that has proven effective in practice to find a good right degree sequence given a specific left degree sequence. The method uses linear programming and the differential equation analysis of Section 3.

Recall that, from the differential equations, a necessary condition for the process to complete is that $\rho(1 - \delta\lambda(x)) > 1 - x$ on $(0, 1]$. We first describe a heuristic for determining for a given (finite) vector (λ_i) representing the left degree sequence and a value for δ whether there is an appropriate (finite) vector (ρ_i) representing the right degree sequence satisfying this condition. We begin by choosing a set M of positive integers which we want to contain the degrees on the right hand side. To find appropriate ρ_m, $m \in M$, we use the condition given by Theorem 3 to generate linear constraints that the ρ_i must satisfy by considering different values of x. For example, by examining the condition at $x = 0.5$, we obtain the constraint $\rho(1 - \delta\lambda(0.5)) > 0.5$, which is linear in the ρ_i.

We generate constraints by choosing for x multiples of $1/N$ for some integer N. We also include the constraints $\rho_m \geq 0$ for all $m \in M$. We then use linear programming to determine if suitable ρ_m exist that satisfy our derived constraints. Note that we have a choice for the function we wish to optimize; one choice that works well is to minimize the sum of $\rho(1 - \delta\lambda(x)) + x - 1$ on the values of x chosen to generate the constraints. The best value for δ for given N is found by binary search.

Given the solution from the linear programming problem, we can check whether the ρ_i computed satisfy the condition $\rho(1 - \delta\lambda(x)) > 1 - x$ on $(0, 1]$.

Due to our discretization, there are usually some *conflict subintervals* in which the solution does not satisfy this inequality. Choosing large values for the tradeoff parameter N results in smaller conflict intervals, although it requires more time to solve the linear program. For this reason we use small values of N during the binary search phase. Once a value for δ is found, we use larger values of N for that specific δ to obtain small conflict intervals. In the last step we get rid of the

Average Degree	Rate				
	1/2	2/3	3/4	4/5	9/10
5.70	1.036	1.023	1.016	1.013	1.006
6.82	1.024	1.013	1.010	1.007	1.004
8.01	1.014	1.008	1.007	1.005	1.002

Table 1: Close to optimal codes for different rates and average left degrees.

conflict intervals by appropriately decreasing the value of δ. This always works since $\rho(1-\delta\lambda(x))$ is a decreasing function of δ.

We ran the linear programming approach on left degree sequences of the form $3, 5, 9, \ldots, 2^i+1$ for codes with rates $1/2, 2/3, 3/4, 4/5, 9/10$ and average left degrees $5.70, 6.82, 8.01$. These results are gathered in Figure 1 which shows how much of the encoding is sufficient to recover the entire message as a fraction of the message length as the message length goes to infinity. Since these graphs do not have nodes of degree two on the left, Theorem 3 imply that with high probability the corresponding codes recover the entire message from the portion of the encoding indicated in the table, provided the message length is large enough. However, as the maximum degrees in the examples we have found are rather large (about 30000), these codes are rather impractical.

One major disadvantage of the approach given above is that we need to fix the left hand side of the graph. To overcome this difficulty, we use the dual condition 13. We can now use this condition and the linear programming approach to solve for the best λ given ρ, then use the original condition to solve for the best ρ given this λ, and so on. We have tried this strategy and it gives good results, although at this point we have not proved anything about its convergence to a (possibly optimal) pair of probability distributions.

For example, we found that the following pair of degree sequence functions yield $[2k,k]_q$-codes which are able to recover from any set of $1.01k$ coordinates, with high probability; the corresponding average degrees is 12:

$$\begin{aligned}
\lambda(x) &= 0.430034x^2 + 0.237331x^{12} + 0.007979x^{13} + 0.119493x^{47} + 0.052153x^{48} + \\
&\quad 0.079630x^{161} + 0.073380x^{162} \\
\rho(x) &= 0.713788x^9 + 0.122494x^{10} + 0.163718x^{199}.
\end{aligned}$$

Note that, on contrary to the examples above, the maximum node degrees in these graphs are much smaller. This makes them more practical for smaller values of k, than the codes giving rise to Table 1.

5.3 Implementations and Timings

In this subsection we report on some of the implementations of our codes. In all these examples a message consisting of 640000 packets was encoded into a vector of 128000 packets, and each packet consisted of 256 bytes. The cascade consisted of three layers: a first layer consisting of 640K nodes on the left, and 320 K nodes on the right, a second layer consisting of 302K nodes on the left and 160K nodes on the right, and a third layer consisting of 160K nodes on the left and on the right. The edge distributions of the graphs used in the first and the second layer were the heavy tail/Poisson distribution discussed in Section 4. The edge distribution in the third layer was different, and used some of the analysis of Section 5.1: the edge distribution on the left was a "double heavy tail" distribution, given by $\lambda(x) := \tilde{\lambda}(x^2)$, where $\tilde{\lambda}$ is the edge distribution function of the heavy tail distribution.

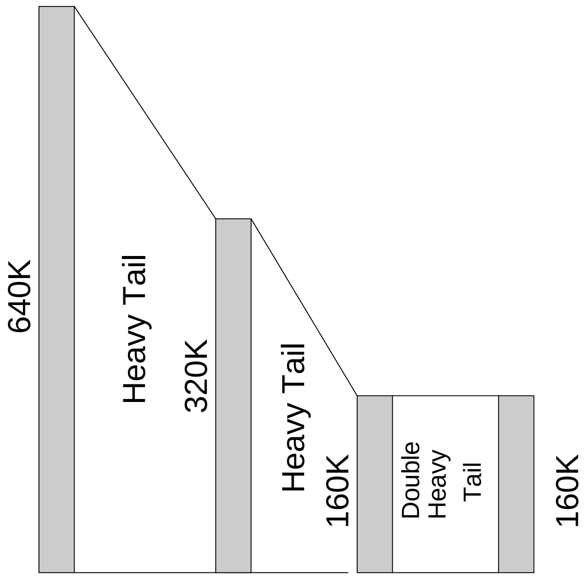

Figure 7: Cascade for example

To choose an appropriate random bipartite graph B with E edges, k nodes on the left, and βk nodes on the right, the program started with a bipartite graph B' with E nodes on both the left and right hand sides, with each node of B' representing an edge slot. Each node on the left hand side of B' was associated with a node on the left side of B, so that the distribution of degrees is given by $(\lambda_1, \ldots, \lambda_m)$, and similarly for the right. The program then choose a random matching (i.e., a random permutation) between the two sets of E nodes on B'. This induced a random bipartite graph on B (perhaps with multi-edges) in the obvious manner with the desired degree structure. In experiments it turned out that the existence of multi-edges is not a serious problem. This can be explained by the observation that one can analyze the process for random multigraphs instead of random graphs and that this analysis turns out to yield exactly the same results as the one carried out in Section 3.

The right endpoints of the edges in that graph were chosen randomly according to a Poisson distribution with mean equal to the average left degree of the edges. A schematic description is given in Figure 7. The average degree of the nodes in this graph was 8. The loss recovery algorithm was executed 1000 times, each time with a different random loss pattern. Figure 8 shows a "length overhead statistics:" the horizontal axis represents μ and the vertical axis represents the percentage of times where $(1+\mu)$ times the length of the message was needed to completely recover the message, based on the 1000 trials. In compliance with the results of Section 3 we see that the parameters are sharply concentrated around their mean value.

On a DEC-alpha machine with 300MHz and 64MB RAM the encoding took 0.58 CPU-seconds, and the loss recovery took 0.94 seconds, on average. This corresponds to a throughput of roughly 280 Mbit/sec.

On a Pentium Pro at 200 MHz and 64MB RAM, the encoding took 0.58 seconds, while the loss recovery took 1.73 seconds, on average. This corresponds to a throughput of roughly 150 Mbit/sec.

It should be noted that most of the time in our algorithm is spent in pointer chasing. The code used was a straight forward C-implementation. Use of more sophisticated data-types, and more intelligent pre-fetching strategies would probably speed up the code considerably.

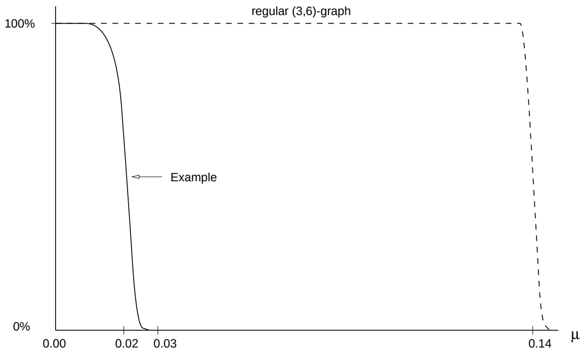

Figure 8: Length overhead statistics

6 Conclusion

We have introduced in this paper a class of error-correcting codes, based on a cascade of bipartite graphs. Although the idea of using sparse bipartite graphs for constructing codes is not new [5, 15], the construction of the graphs in each of the layers is novel. We obtained the construction by analyzing a simple loss recovery algorithm. The analysis used results asserting the sharp concentration of parameters in a discrete random process around their means. Using this, we could set up a simple condition that the degree sequence distributions of the left and right hand sides of the bipartite graphs had to satisfy in order for the process to finish successfully. We designed a sequence of asymptotically good erasure codes with linear time encoding and loss recovery algorithms. We should point out that our model of computation, as it stands, is that of a random access machine with unit cost. However, our construction can be modified using pre-fetching strategies to yield linear time algorithms for random access machines with logarithmic cost. The modification is quite similar to that given in [14].

References

[1] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Trans. Inform. Theory*, 42:1732–1736, 1996.

[2] R.E. Blahut. *Theory and Practice of Error Control Codes.* Addison Wesley, Reading, MA, 1983.

[3] P. Elias. Coding for two noisy channels. In *Information Theory, Third London Symposium*, pages 61–76, 1955.

[4] A. Frieze and S. Suen. Analysis of two simple heuristics on a random instance of k-SAT. *J. of Alg.*, 20:312–355, 1996.

[5] R. G. Gallager. *Low Density Parity-Check Codes.* MIT Press, Cambridge, MA, 1963.

[6] R. Karp and M. Sipser. Maximum matchings in sparse random graphs. In *Proc. 22nd FOCS*, pages 364–375, 1981.

[7] T.G. Kurtz. *Approximation of Population Processes.* CBMS-NSF Regional Conf. Series in Applied Math. SIAM, 1981.

[8] M. Luby, M. Mitzenmacher, and M.A. Shokrollahi. Analysis of random processes via and-or tree evaluation. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 364–373, 1998.

[9] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, 1988.

[10] M. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing.* PhD thesis, UC Berkeley, 1996.

[11] M. Mitzenmacher. Tight thresholds for the pure literal rule. Technical Report 1997-011, DEC/SRC, 1997.

[12] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[13] B. Pittel, J. Spencer, and N. Wormald. Sudden emergence of a giant k-core in a random graph. *J. Comb. Theory, Series B*, 67:111–151, 1996.

[14] M. Sipser and D. Spielman. Expander codes. *IEEE Trans. Inform. Theory*, 42:1710–1722, 1996.

[15] D. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Inform. Theory*, 42:1723–1731, 1996.

[16] N.C. Wormald. Differential equations for random processes and random graphs. *Ann. Appl. Prob.*, 5:1217–1235, 1995.