Optimizing the Degree Distribution of LT Codes with an Importance Sampling Approach

Esa Hyytiä[†]
Centre for Quantifiable Quality of Service in Communication Systems, Norwegian University of Science and Technology, Trondheim, Norway

Tuomas Tirronen[‡]
Networking Laboratory
Helsinki University of Technology
Finland

Jorma Virtamo[‡]
Networking Laboratory
Helsinki University of Technology
Finland

***Abstract*— Fountain coding principle introduced by Byers et al. in 1998 describes an efficient way to transfer information over erasure channels. In this paper, we focus on a particular class of fountain codes, the LT codes. The key component of LT codes is the so-called degree distribution used in the encoding procedure. The degree distribution is the sole component responsible for the efficiency of the LT codes. In general, the optimization of the degree distribution is not a trivial problem. This paper describes an algorithm for iterative optimization of parameterized degree distributions for LT codes. In particular, we use methods utilized in importance sampling theory to construct an objective function which eventually is optimized with non-linear optimization methods. The proposed method is studied for message lengths of a couple of hundred blocks and less. We present some examples of degree distributions optimized with the proposed algorithm with comparisons to the performance of previously proposed distributions.**

I. Introduction

Digital fountain coding is a relatively new concept for digital content distribution introduced in 1998 [1] and is based on rateless error correcting codes for erasure channels. The name originates from an analogy to a fountain spraying water drops, which then are collected into a bucket. This translates into servers spraying random pieces of data, e.g. packets, which receivers then collect. When a sufficient number of packets is collected the file can be decoded. With good fountain codes the total size of the packets needed for decoding (on average) is close to the original size of the file, although some overhead is necessary due to the nature of these codes. An important characteristic of a digital fountain is that it is irrelevant which particular packets are received. As soon as a certain amount of the packets are received the message can be decoded (with high probability).

It should be noted that codes enabling such fountain coding scenario have already existed for some time, namely the Reed-Solomon codes [2] and LDPC [3] codes to some extent. The key benefit of recently discovered codes is the efficiency: with good fountain codes the computational complexity of the encoding and decoding processes is low for even long message lengths n.

The Reed-Solomon codes can be used as very efficient fountain codes when the message length n is small, requiring no overhead packets at all. The original data is divided into n blocks and encoded into $n + k$ symbol blocks, which are then distributed. It is sufficient to collect any n encoded blocks to retrieve the original data. The downside is that for large n the Reed-Solomon codes are impractical due to computational complexity. However, the optimization of other fountain coding methods is still an interesting task for which we give some insight in this paper.

In particular, our contribution in this paper is an algorithm for iterative optimization of the degree distribution used in LT codes by using an approach based on importance sampling. We define two different objectives for the optimization. Minimizing the mean number of packets needed for decoding is perhaps the more natural goal but we also discuss the maximization of the probability of successful decoding in exactly n steps.

Samples of coding and decoding processes are generated by simulation using some degree distribution. The samples are then used to estimate the expectation of the objective function that would result if another degree distribution had been used by weighing the samples appropriately. This weighting is accomplished by calculating the likelihood ratios in similar fashion as in general importance sampling methods. Having an estimate for the objective function as a function of the degree distribution is the cornerstone for our optimization algorithm.

The results produced with the algorithm are approximative but still turn out to show good performance when compared, e.g., to previously published soliton distributions [4] in the range of message lengths of couple of hundred packets and less. A completely different study of the coding process based on an analytical combinatorial approach is presented in [5]. However, this method is viable only for very small message lengths n, while the method presented in this paper is suitable for optimization of degree distribution for larger values of n.

The proposed method works directly for small message lengths n. For larger n parameterized degree distributions are used to obtain reasonable results and running times. The use of parameterized distributions, however, turns out to be a

[†] "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU and UNINETT. http://www.ntnu.no/Q2S/

[‡] This work was partially funded by the Finnish Funding Agency for Technology and Innovation (projects PAN-NET and ABI).

Algorithm 1 A general LT encoding algorithm

1: **repeat**
2: choose a degree d from degree distribution $\rho(d)$.
3: choose uniformly at random d blocks $m(i_1), \ldots, m(i_d)$.
4: send $m(i_1) \oplus m(i_2) \oplus \cdots \oplus m(i_d)$.
5: **until** enough output symbols are sent.

reasonable method. Note that the previously proposed, well-working degree distributions called soliton distributions are also parameterized.

The rest of the paper is organized as follows. In Section I we give a brief introduction to LT codes, degree distributions and the notation used in this paper. Section II compactly reviews the principles of importance sampling and defines the algorithm which we use to optimize the degree distribution. In Section III we give numerical examples and results obtained using the proposed algorithm. Section IV concludes the paper.

A. LT codes

LT codes were published by Michael Luby in 2002 [4]. These codes are rateless, meaning that the rate does not need to be fixed beforehand, and encoding symbols can be generated on the fly [6], [7]. LT codes are also the first class of codes fully realizing the digital fountain concept presented in [1].

1) Encoding of LT code: The encoding process is extremely simple. The key parameter for encoding is the so-called degree distribution:

Definition 1 (Degree distribution): The degree distribution $\rho(i)$ of LT code is a probability distribution defining the number of blocks combined in a packet.

The degree distribution is sampled to obtain the degree d for the next output packet. The output packet is generated by choosing d blocks from the original file uniformly at random and calculating the sum of these blocks in $GF(2)$ arithmetic, i.e., by applying bitwise XOR operation. Algorithm 1 shows the encoding algorithm for the LT code [4]. Stopping condition for the encoding algorithm can be specified, e.g., by agreeing on the number of packets beforehand, or the recipient(s) can send an acknowledgement when enough packets have been received. Note that it does not matter what the block (symbol) length is. One input block could be just one bit or a larger chunk, the encoding and decoding processes are the same regardless. XOR operations are always done bitwise between the whole blocks.

2) Decoding of LT codes: Decoding is done iteratively by using information of which source blocks are added together in a received packet. This information needs to be included somehow in the encoding procedure; different implementation alternatives are available but are not discussed here. First the decoder removes the possible known blocks from each received packet by taking a XOR between the packet and the known block(s). If the degree of the packet is higher than 1 (after the removal of the known blocks) it consists of more than one original blocks and it is stored in the buffer. If a degree-1 packet has been recovered, it is identical to an original block, i.e., one new block has been discovered. Next the newly discovered block is removed from the other buffered packets including it. If this step reveals new degree-1 packets, the decoding continues iteratively until the original message is fully decoded. Otherwise the decoder moves to wait for a next packet. The decoding process is sketched in listing Algorithm 2.

Algorithm 2 A general LT decoding algorithm

1: **repeat**
2: **while** no degree-1 packets in buffer $\mathcal{B}$ **do**
3: $\mathcal{B} \leftarrow$ received packet $-$ known blocks.
4: **end while**
5: $m(j) \leftarrow$ degree-1 packet from $\mathcal{B}$. $\{j$ discovered$\}$
6: **for all** $c \in \mathcal{B} : c$ includes $m(j)$ **do**
7: $c \leftarrow c \oplus m(j)$
8: **end for**
9: **until** original message is recovered.

Note that the decoding process is suboptimal in the sense that not all the information in the received packets is used. For example, if message consists of $n = 3$ blocks, the recipient could decode the original message if he had three different packets which each consists of two symbols. This, however, would make the decoding algorithm computationally more demanding, as this method equates to solving a linear system of equations, which is a laborious operation in general case for large message lengths.

B. Notation

As mentioned already, the number of blocks in the message is denoted by n, i.e., n is the number of input symbols (or blocks) in the message. The degree distribution is denoted by $\rho(d)$. When convenient, we also refer to the point probabilities by p_j, i.e., $p_j = \rho(j)$.

C. Degree distributions

Later in this paper we will consider several degree distributions and make some comparisons of their performance against the optimized distributions obtained by the IS approach. The reference distributions are the following:

Definition 2 (Uniform distribution):

$$p_i = 1/n \quad \forall\, i = 1, \ldots, n.$$

Definition 3 (Degree-1 distribution):

$$p_1 = 1, \text{ and } p_i = 0 \quad \forall\, i = 2, \ldots, n.$$

The degree-1 distribution is sometimes referred to as all-at-once distribution in the literature.

Definition 4 (Soliton distribution):

$$p_1 = 1/n,$$
$$p_i = \frac{1}{i(i-1)}, \qquad \text{for } i = 2, 3, \ldots, n.$$

This is sometimes also referred to as the ideal soliton distribution [6]. This is the distribution which yields optimal

performance in expectation. However, it is not practical to use the soliton distribution in the LT coding process. Its improved counterpart, the so-called robust soliton distribution has two additional parameters and is defined as follows:

Definition 5 (Robust soliton distribution [6]): First, define

$$S = c \ln(n/\delta) \sqrt{n},$$

where c and δ are the extra parameters. Then, define

$$\tau_i = \begin{cases} \frac{S}{i \cdot n}, & \text{for } i = 1, 2, \ldots, (n/S) - 1, \\ \frac{n}{S} \ln(S/\delta), & \text{for } i = (n/S), \\ 0, & \text{otherwise.} \end{cases}$$

Then, the robust soliton distribution is obtained by a linear combination of the (ideal) soliton distribution and τ_i,

$$p_i = \frac{p_i^* + \tau_i}{C},$$

where p_i^* corresponds to ideal soliton distribution (Def. 4) and C is the normalization constant. Further discussion and reasoning for this distribution can be found in the references.

II. Importance Sampling Approach

Importance sampling (IS) belongs to the family of Monte Carlo methods, which are simulation methods used to either generate samples from a (usually complicated) probability distribution or to efficiently estimate the expectations of functions of a random variable $\mathbf{X}$. IS is used for the latter task as a variance reduction technique to decrease the number of samples needed for a successful estimation of the desired expectation as is explained in [8, Chapter 4]. However, this property is not interesting in our scenario. What we want to borrow from the IS theory is the general concept of importance sampling: samples generated with one probability distribution can be used to estimate some expectation with a different probability distribution. It should be noted, however, that the use of a distribution completely different from the original one results in a poor estimate, i.e., the used distribution should have some resemblance to the original distribution.

Let us consider the situation where we want to calculate the expectation of a function $h(\mathbf{X})$ of the random variable $\mathbf{X}$, with probability density function $p(\mathbf{x})$:

$$E[h(\mathbf{X})] = \int h(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x}. \tag{1}$$

By drawing samples $\{\mathbf{X}^{(i)}\}_{i=1}^m$ from $p(\mathbf{x})$, we could calculate an estimate for the expectation:

$$\hat{h} = \frac{1}{m} \sum_{i=1}^{m} h(\mathbf{X}^{(i)}). \tag{2}$$

In IS, $p(\mathbf{x})$ is replaced by another probability distribution $g(\mathbf{x})$. Let the random variable obeying this distribution be $\tilde{\mathbf{X}}$. We can write (1) equally as

$$E[h(\mathbf{X})] = \int h(\mathbf{x}) \frac{p(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) \, d\mathbf{x}. \tag{3}$$

This shows that we can generate samples $\{\tilde{\mathbf{X}}^{(i)}\}_{i=1}^m$ from $g(\mathbf{x})$ and use these to calculate the estimate (2)

$$\hat{h} = \frac{1}{m} \sum_{i=1}^{m} w(\tilde{\mathbf{X}}^{(i)}) h(\tilde{\mathbf{X}}^{(i)}), \tag{4}$$

where $w(\tilde{\mathbf{X}}^{(i)})$ denotes the *importance ratio* (also *likelihood ratio* in some references)

$$w(\mathbf{x}) = \frac{p(\mathbf{x})}{g(\mathbf{x})}, \tag{5}$$

where we assume that $g(\mathbf{x}) > 0 \; \forall \; \mathbf{x} : p(\mathbf{x}) h(\mathbf{x}) \neq 0$. The variance of estimate (4) can be lower than that of in (2) with an appropriate choice of the sampling function $g(\mathbf{x})$.

The concept of importance ratio and the general idea behind imporance sampling is exploited in the following sections to generate an optimization strategy.

A. Objectives of Optimization

Let random variable T denote the number of received packets required to decode a message. The natural goal in optimizing the degree distribution of LT code is to make the mean number of overhead packets,

$$\text{overhead} = \frac{\mathrm{E}[T] - n}{n}, \tag{6}$$

needed as small as possible. On the other hand, in some scenarios we might also be interested in the probability for decoding in exactly n sent packets. This reasoning leads to two mutually exclusive optimization goals pursued in this work.

Definition 6 (MinAvg, MaxPr): Optimization objectives:

1) Objective *MinAvg* corresponds to finding a degree distribution which minimizes the average number of packets $\mathrm{E}[T]$ needed for a successful decoding.
2) Objective *MaxPr* corresponds to finding a degree distribution which maximizes the probability of decoding a message with exactly n sent packets, $\mathrm{P}\{T = n\}$.

The second objective above can also be generalized to consider the probability of decoding with $n + k$ packets, where $k > 0$.

B. Simulation of the LT process

As already mentioned, our goal is to optimize a given quantity by choosing the degree distribution $\rho(d)$ appropriately. Let random variable R denote the outcome of a single transmission of a message. With *MinAvg* the objective was to minimize the mean number of packets needed for a successful decoding a message. Thus, in this case R corresponds to the number of packets required to decode the message, $R = T$, and the objective can be written as

$$\textit{MinAvg}: \quad \min \; \mathrm{E}[R].$$

Similarly, when the objective is *MaxPr*, i.e., the maximization of the decoding probability after receiving n packets, we can assign random variable R value 1 if the decoding is succesful after receiving n packets, and 0 otherwise, i.e., $R = 1(T = n)$. Then the objective can be written as

$$\textit{MaxPr}: \quad \max \; \mathrm{E}[R].$$

In particular, we use m simulated samples $S_k, k = 1, \ldots, m$, of the LT process to construct an estimator for a chosen objective. Each of the samples S_k describes one simulated transmission of a message of length n. Thus, S_k consists of the outcome of the transmission, R_k, and the numbers of how many packets of a particular degree were sent, i.e., $S_k = \{R_k, \mathbf{n}^{(k)}\}$, where R_k depends on the objective and $\mathbf{n}^{(k)}$ denotes a vector with component i describing the number of degree i packets sent during the transmission.

C. Estimator for the Objective Function

The first step is to construct an estimator $\widehat{R}$ for the chosen objective function using the available information of m simulated samples $S_k, k = 1, \ldots, m$. The goal is to construct an algorithm, which takes some degree distribution vector $\mathbf{p}$, defined by the point probabilities $p_1, \ldots, p_n$, as input and outputs a better one. To this end we use the idea of importance sampling in a sense in the opposite direction. We generate samples with a given distribution vector $\mathbf{p}$ and use (4) to estimate what the expectation $\widehat{R}(\mathbf{q})$ of the objective function would be if another degree distribution $\mathbf{q}$ had been used. Thus we obtain an estimate for the interesting expectation as a function of the degree distribution, allowing us to use this for optimization studies. In particular, we have

$$\widehat{R}\,(\mathbf{q}) = \frac{1}{m}\sum_{k=1}^{m} R_k \prod_i \left(\frac{q_i}{p_i}\right)^{n_i^{(k)}}, \tag{7}$$

where m is the number of samples generated, and R_k and $n_i^{(k)}$ specify the observed objective function and number of degree-i packets generated in the kth sample. An important point to note here is that, given the degree i of an encoded packet, the i source blocks are chosen at random, each combination being equally probable. Thus, when the degree distribution is changed only the numbers of packets of different degrees matter in defining the probability of a sample with respect to the new measure.

Note that, as explained in Section II-B, both objectives in Definition 6 can be taken into account; for objective *MinAvg* R_k represents the number of packet needed for successful decoding, for objective *MaxPr* we define R_k to be one when the decoding succeeds in the defined number of steps and zero otherwise.

While the estimate $\widehat{R}(\mathbf{q})$ could be directly optimized, the problem is that with a finite number of samples the estimate is not exact, especially when $\mathbf{q}$ is far from $\mathbf{p}$, i.e., the estimate has large variance. The problem of the number of samples is the downside of this optimization strategy, especially when the number of optimized parameters is large. First of all the generation of samples is not immediate, and secondly for larger values of n the computation and memory requirements become an obstacle.

Alternatively, by derivation of (7) it is possible to get an estimate for the gradient of the objective function with respect to $\mathbf{q}$ at $\mathbf{q} = \mathbf{p}$ allowing us to use, for example, the method of steepest descent (gradient method) [9] for optimization. The estimate for the gradient has roughly the same statistical accuracy as the estimate for the expecation itself because it is evaluated at $\mathbf{q} = \mathbf{p}$ where all the likelihood ratios are equal to one. In the steepest descent method the gradient defines the direction where the next candidate for a degree distribution would lie. For instance line search can be used to find the optimal point in this direction. When we have taken the step towards the optimal point (according to the estimate), new set of samples can be generated using the simulator and thus the algorithm proceeds iteratively by calculating new points (degree distributions) until some convergence criterion is met. For brevity, we call the developed algorithm ISG-algorithm, the acronym ISG standing for the initials of the words importance sampling and gradient.

1) Gradient and Projection Vectors: The component i of the gradient of the estimate $\widehat{R}(\mathbf{q})$ is:

$$\widehat{g}_i = \frac{\partial \widehat{R}}{\partial q_i} = \frac{1}{m}\sum_{k=1}^{m} R_k n_i^{(k)} \frac{1}{p_i}\left(\frac{q_i}{p_i}\right)^{n_i^{(k)}-1}. \tag{8}$$

When this is evaluated at point $\mathbf{q} = \mathbf{p} \;\Leftrightarrow\; q_i = p_i \;\forall\; i$ we have:

$$\left(\frac{\partial \widehat{R}}{\partial q_i}\right)_{\mathbf{q}=\mathbf{p}} = \frac{1}{m}\sum_{k=1}^{m} R_k \frac{n_i^{(k)}}{p_i}. \tag{9}$$

Next we have to ensure that if we actually take the step suggested by the gradient (9), i.e., we calculate $\mathbf{p}_{\text{new}} = \mathbf{p} + \lambda\mathbf{g}$ for some parameter λ, the resulting point $\mathbf{p}_{\text{new}}$ is a proper probability distribution. This means that all components $(p_{\text{new}})_i \in [0, 1]$ and the sum of the components is one.

Projecting gradient (9) to hyperplane $\mathbf{g} \cdot \mathbf{e} = 0$, where $\mathbf{e}$ is a vector of ones guarantees that the sum of the components is equal to one. To take care of the other requirement, we limit the change of each component relatively so that the value does not decrease below zero. This combined with the hyperplane projection guarantees that every component $(p_{\text{new}})_i \in [0, 1]$. By using these restrictions, we ensure that the gradient points in the right direction in space, where each point corresponds to a probability distribution.

The projection of the gradient vector is

$$\mathbf{g}_{\text{proj}} = \mathbf{g} - \frac{1}{n}(\mathbf{g} \cdot \mathbf{e})\mathbf{e}. \tag{10}$$

For component i, $i = 1, \ldots, n$:

$$\begin{aligned}(g_{\text{proj}})_i &= g_i - \frac{1}{n}\sum_{i=1}^{n} g_i \\ &= \frac{1}{m}\sum_{k=1}^{m} \underbrace{R_k\left(\frac{n_i^{(k)}}{p_i} - \frac{1}{n}\sum_{i=1}^{n}\frac{n_i^{(k)}}{p_i}\right)}_{s_i^{(k)}}.\end{aligned} \tag{11}$$

The expression $s_i^{(k)}$ can be considered as one sample of the projected gradient. The estimated value of the gradient projection is then the calculated sample mean as given in (11). As the estimate (7) is calculated from simulation samples, we need a criterion for the number of samples we want to use for

calculating the actual estimate. In our optimization algorithm we use the standard deviation of the projected gradient vector as a measure for the number of samples to be generated. We need to calculate the sample standard deviation for the projected gradient, and use this value to control the accuracy. By modifying the criterion for the accuracy we can strike at the balance between the accuracy of the calculated distributions and practical running times of the algorithm.

We generate the samples $s_i^{(k)}$ as one long simulation run, where the sample variance is given by:

$$\sigma_i^2 = E\left[(\mathbf{X}-\mu)^2\right] = \frac{1}{m}\sum_{k=1}^{m}\left(s_i^{(k)} - (g_{\text{proj}})_i\right)^2, \quad (12)$$

and the sample standard deviation:

$$\sigma_i = \sqrt{\frac{1}{m-1}\sum_{k=1}^{m}\left(s_i^{(k)} - (g_{\text{proj}})_i\right)^2}. \quad (13)$$

To calculate the standard error of the mean (11), we use the following result:

$$\text{Var}\left[\bar{\mathbf{X}}\right] = \frac{\sigma_i^2}{m}, \quad (14)$$

where m is the number of samples and $\bar{\mathbf{X}}$ is the sample mean of the random variable $\mathbf{X}$. Hence, the standard error of the mean value of the samples of the projected gradient is:

$$\sigma_{g_i} = \sqrt{\frac{1}{m(m-1)}\sum_{k=1}^{m}\left[R_k\left(\frac{n_i^{(k)}}{p_i} - \frac{1}{n}\sum_{i=1}^{n}\frac{n_i^{(k)}}{p_i}\right) - (g_{\text{proj}})_i\right]}.$$

On the other hand this estimate can be represented in a way more convenient for this algorithm:

$$\sigma_{g_i} = \sqrt{\frac{1}{m(m-1)}\sum_{k=1}^{m}\left(s_i^{(k)}\right)^2 - \frac{1}{m}\left(\sum_{k=1}^{m}s_i^{(k)}\right)^2}. \quad (15)$$

This latter form is used when generating new samples. The form (15) is faster to use because both sums are easy to calculate as running sums during the execution of the algorithm and thus is more practical.

2) Line Search for Step Length Calculation: When we have the projected gradient we still need to do a line search to find the optimum in the direction of the gradient. This means that we want to find a λ such that

$$f(\lambda) = \widehat{R}\left(\mathbf{p} + \lambda\mathbf{g}\right), \quad (16)$$

is optimized, where $\mathbf{p}$ is the starting point and $\mathbf{g}$ the calculated (projected) gradient. Thus f represents the one-dimensional function in the direction of the gradient.

We have chosen a simple bisection search to do the line search as presented in [9]. Bisection search works on a specified interval for convex functions by calculating the value of the derivate of the function to be optimized in the middle point of the target interval by halving the target interval based on the sign of the derivative.

As an example case of how the bisection search works, we look into the maximization of a differentiable function f on interval $[a, b]$. Let $f'(\lambda)$ denote the derivative at (middle) point λ. Then,

i) if $f'(\lambda) = 0$, then maximum (or minimum) is at λ,
ii) if $f'(\lambda) < 0$, then the maximum is left of λ,
iii) if $f'(\lambda) > 0$, then the maximum is right of λ.

These cover all possible cases and lead to Algorithm 3.

Algorithm 3 A general bisection search algorithm

Require: $a_0 < b_0$, $k = 0$
repeat
  $c_k \leftarrow \frac{1}{2}(a_k + b_k)$
  if $f'(c_k) > 0$ **then**
    $a_{k+1} \leftarrow c_k$, $b_{k+1} \leftarrow b_k$ {update left bound}
  else
    $a_{k+1} \leftarrow a_k$, $b_{k+1} \leftarrow c_k$ {update right bound}
  end if
  $k \leftarrow k + 1$
until $k = N$

The execution of the algorithm is controlled through parameter N, which describes the maximum number of iterations executed in the algorithms. Usually we want to define a threshold value l so that the algorithm will stop when the length of the interval $[a_{k+1}, b_{k+1}]$ is less than l. The number of steps needed to achieve this can be easily calculated:

$$\begin{aligned}\left(\frac{1}{2}\right)^N &\leq \frac{l}{b-a}\\ N\log\frac{1}{2} &\leq \log l - \log(b-a)\\ N &\leq \frac{\log(b-a) - \log l}{\log 2}\\ \Rightarrow\ N &= \left\lceil\frac{\log(b-a) - \log l}{\log 2}\right\rceil. \end{aligned} \quad (17)$$

3) General Parameterized Distributions: The idea in Section II-B can be extended to include parameterized distributions instead of a general one where point probabilities are the parameters.

Let θ and η be vectors of n parameters, which define a degree distribution. We define the estimate in (7) again using parameterized probability distributions:

$$\widehat{R}\left(\eta\right) = \frac{1}{m}\sum_{k=1}^{m}R_k\prod_i\left(\frac{p_i(\eta)}{p_i(\theta)}\right)^{n_i^{(k)}}, \quad (18)$$

where η denotes the starting point in parameter space, serving similar function as $\mathbf{q}$, in Sections II-B and II-C.1. The gradient can be computed by differentiating with respect to parameters

η_i, using the chain rule for product differentiation:

$$\widehat{g}_j = \frac{\partial \widehat{R}}{\partial \eta_j} = \frac{1}{m}\sum_{k=1}^{m} R_k \sum_i \frac{n_i^{(k)}}{p_i(\theta)} \left(\frac{p_i(\eta)}{p_i(\theta)}\right)^{n_i^{(k)}-1} \frac{\partial p_i(\eta)}{\partial \eta_j} \prod_{l\neq i} \left(\frac{p_l(\eta)}{p_l(\theta)}\right)^{n_l^{(k)}}. \quad (19)$$

When this is evaluated at $\eta = \theta$ we arrive at a simpler form:

$$\left(\frac{\partial \widehat{R}}{\partial \eta_j}\right)_{\eta=\theta} = \frac{1}{m}\sum_{k=1}^{m} R_k \sum_i \frac{n_i^{(k)}}{p_i(\eta)} \frac{\partial p_i(\eta)}{\partial \eta_j}. \quad (20)$$

This closely resembles the form in (9), but in addition the information of the derivative of the parameterized distribution is included.

4) Stopping Condition: The estimate $\widehat{R}$ is calculated from simulation results as shown in (7). When considering the convergence of the algorithm, we take into account the "noise" in the simulation, that is, the fact that this is only an estimate based on some finite number of simulation results. We can approximate this noise as the standard deviation of the estimate of the average number of the packets needed for decoding $\widehat{R}$. The calculation is performed in the same way as presented in Section II-C.1 with the projected gradient vector:

$$\sigma_{\widehat{R}} = \sqrt{\frac{1}{m(m-1)}\sum_{i=1}^{m}\left(R_k \prod_i \left(\frac{q_i}{p_i}\right)^{n_i^{(k)}} - \widehat{R}\right)^2}. \quad (21)$$

Now, after every iteration of the ISG-algorithm, we calculate the estimate for the average number of packets $\widehat{R}$ needed for a successful decoding. We compare the last two values of $\widehat{R}$ and if their absolute difference is smaller than the standard deviation $\sigma_{\widehat{R}}$, we stop the algorithm.

This means that we stop the algorithm when the approximated noise in the simulation is larger than the difference between the last two calculated estimates. The noise overwhelms the difference, accordingly we can conclude that the last two estimates are calculated using degree distributions as near the optimal ones as we can get to with the given number of samples.

D. ISG-Algorithm

1) Overview of the Algorithm: This section will present a general framework of the ISG-algorithm. The actual algorithm consists of several sub-algorithms, most of which are well-known (bisection search, gradient based optimization etc.). The actual idea of the algorithm is simple as discussed earlier, some complexity occurs from the many parameters which needs to be set to control the convergence of the algorithm and the generation of the samples.

In the case of point distribution optimization, the dimension of the problem at hand is the number n of the blocks in the message to be transferred. The vectors manipulated during the execution are thus of length n in the case where point distributions are optimized directly. The optimization then takes place in space where points are degree distributions. Criteria for a point to belong to this space is that all components are positive and all components sum up to one.

With parameterized degree distributions, the number of vector components is the number of parameters used. The space is now the parameter space, each point corresponding to different values of parameters. It might be reasonable to restrict this space somewhat in certain scenarios, for example not allowing negative values for parameters.

The following description summarizes both point and parameterized distribution optimization. The algorithm takes as an input some degree distribution, defined by point probabilities $p_1, \ldots, p_n$, threshold ε for sample generation, and interval and threshold for bisection search.

1) Use the given probability distribution (either point or parameterized form) as a starting distribution $\mathbf{p}$.
2) Generate samples S using the degree distribution $\mathbf{p}$. Generate samples until accuracy is less than ε as described in Section II-C.1.
3) Use the samples S to calculate the gradient $\mathbf{g}$. This is, in the case of point distribution, the projection of the gradient vector (11) or, when optimizing parameterized distributions, the gradient (20).
4) Optional: divide the gradient $\mathbf{g}$ by its length, $\mathbf{g} \leftarrow \frac{\mathbf{g}}{\|\mathbf{g}\|}$.
5) Do a bisection search in the direction of the gradient, as described is Section II-C.2. This means that we optimize $\widehat{R}(\mathbf{p} + \lambda\mathbf{g})$, either finding a minimum or maximum depending on the goal (Definition 6). As a result we have the step length λ.
6) The step towards a better distribution is: $\text{step} \leftarrow \lambda \cdot \mathbf{g}$.
7) Limit the change of each component to 90% of the previous value. This percentage can also be varied. This ensures that the conditions $\sum p_i = 1$ and $p_i \in [0, 1]$ are met for point distributions and for parameterized distributions retains the parameters on positive side.
8) Move in the direction of the gradient: $\mathbf{p} \leftarrow \mathbf{p} + \text{step}$.
9) Calculate the value of $\widehat{R}$ using (7) and the standard deviation $\sigma_{\widehat{R}}$ using (21). If the absolute difference between the last two estimates is less than the standard deviation, then stop. Otherwise continue and go back to step 2.

2) Implementation Issues of the Algorithm: For the goal *MaxPr* an automatic implementation is easy to make. The line search (maximization) behaves well and has a clear maximum to which the algorithm converges.

This is, however, not the case with the goal *MinAvg*. The interval for the line search has to be chosen carefully, otherwise the minimization can converge into a non-feasible region. The limited amount of samples causes the form of the estimate $\widehat{R}$ to have some peculiarities. For example, when optimizing the point distribution, there always exists a minimum of zero at point $\mathbf{q} = 0$, as can easily be seen from (7). This means that in order to generate the results, visual inspection of the line search and proper convergence is advisable. Proper interval $[a_0, b_0]$ for starting point of Algorithm 3 can be selected either by visual inspection or by trial and error. In any case it

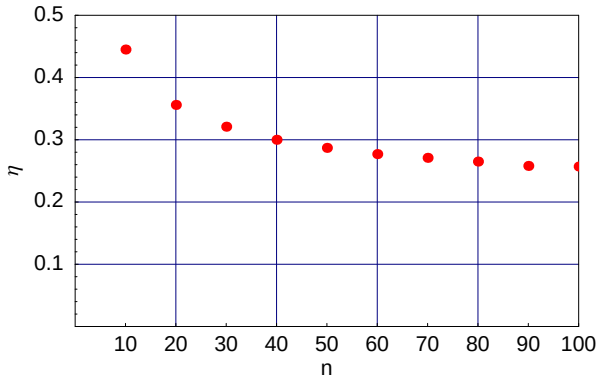

Fig. 1. Optimized parameter values for form (22).

is easy to check the results of the algorithm (i.e., proposed distributions) to discard the poor intermediate results.

III. Numerical Results

A. Parameterized Form $e^{-\eta i}$

Let us consider the next parameterized distribution,

$$p_i = \frac{e^{-\eta i}}{\sum_i e^{-\eta i}} \tag{22}$$

which is actually a geometric distribution for which the optimal parameters can be computed efficiently. We consider both of the objectives given in Definition 6.

1) Maximizing the Decoding Probability: There are some issues when optimizing for maximum decoding probability. The probability for the decoding to succeed in exactly n steps approaches zero as n grows. In point distribution optimization, this is not a major problem, although in the cases where n is near 10, major part of the simulation results produce coefficient $R_k = 0$ in (7), thus rendering a large part of the simulation results useless.

With the parameterized form (22) results were calculated for cases $n = 10, 20, 30$ and 40, maximizing the probability of decoding in exactly n steps. The algorithm was ran with a hard limit of 10^6 samples per round, 10 rounds at maximum. Generation threshold ε was set at $\varepsilon = 0.1$. Line search interval was $[0, 0.05]$ with stopping threshold of 0.0001.

The optimization results for these cases are presented in Table I. There is no point to continue this for larger values of n, as even for $n = 40$ only approximately 0.01% of the simulation results generate successful decoding in 40 steps, when the simulations are started from parameter value $\eta = 0.33$. This would mean that even with large amount of samples, say 10^7, only 1000 samples would give successful decoding, resulting in high inefficiency of the ISG-algorithm.

For maximization of large values of n we can relax the requirement of decoding in exactly n steps to $n + k$ steps, where k can be anything $k \geq 1$. Of course, the larger the value of k, the more samples will take part in forming the estimate of $\widehat{R}$, thus giving more accurate optimization results. The situation however is not analogous to maximizing the probability of decoding in at most n steps, as the algorithm maximizes for the conditions asked for, that is, decoding in at most $n + k$ steps.

Because of this inefficiency when maximizing the decoding probability, we will focus on the minimization of the average number of packets needed for decoding for the rest of this paper.

2) Minimizing the Average Number of Needed Packets: The optimization was performed using ISG-algorithm with generation threshold $\varepsilon = 0.1$, with a hard limit of 10^6 samples. The bisection search was performed on interval $[0, 0.1]$ with threshold 0.001, which translates to 10 iterations in (II-C.2). The resulting parameters for $n = 10, 20, \ldots, 100$ are presented in Fig. 1. The values which produced the lowest overheads were chosen.

Using the parameters plotted in Fig. 1, we did 10000 simulations of the LT process in order to get a final estimate for the performance of the optimized degree distributions. That is, for each message length n we have first optimized the parameter ν. Then, using this optimized degree distribution we have ran another set of simulations from which we have estimated both the sample mean $\hat{T}$ and standard deviation $\hat{\sigma}(T)$ of random variable T (for each n). The resulting sample means for the number of packets needed for decoding are presented in Fig. 2 . In Fig. 3 we have plotted values of $\hat{T}$ and $\hat{\sigma}(T)$. The sample standard deviations are illustrated by "error bars" corresponding to values $\hat{T} \pm \sigma(\hat{T})$. The dotted line represents reference performance of $T = n$.

Thus, Fig. 3 corresponds to the same situation as Fig. 2 but the representation is in absolute values of $\hat{T}$ with added bars to illustrate how the standard deviation behaves. Fig. 2 shows that as n increases the overhead decreases a little. Still the overhead of nearly 40% is probably not satisfactory for applications.

B. Parameterized Form $e^{-\eta_1 i} + \eta_2 e^{-\eta_3 i}$

The parameterized form

$$p_i = e^{-\eta_1 i} + \eta_2 e^{-\eta_3 i}, \tag{23}$$

with normalization, is an enhanced form of (22). The added term should allow finetuning the form of the distribution, and if possible, generate better results especially when $n \approx 100$.

However, while the ISG-algorithm works nicely with the added number of parameters, we did not find the results any better than with the optimized case with one parameter. One problem might be existing local minima, which were found by using different sets of starting parameters. Table II lists some

TABLE I

Fraction of useful samples with *MaxPr* objective

n	η	useful samples
10	0.476	4.6%
20	0.387	0.44%
30	0.350	0.06%
40	0.33	0.01%

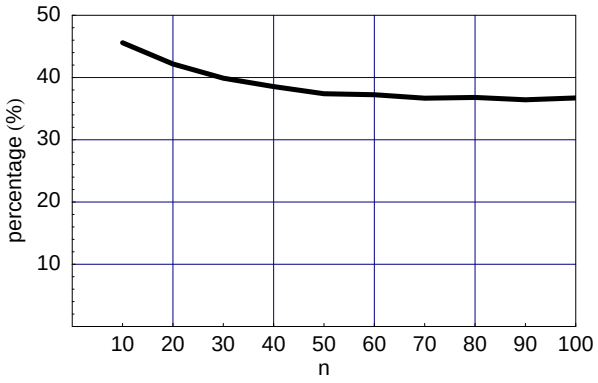

Fig. 2. Percentages of overhead packets when optimization is done to minimize the average number of packets needed for decoding for form (22).

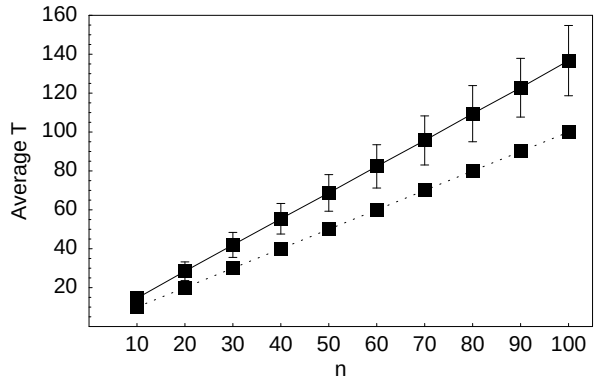

Fig. 3. Standard deviations of simulations shown with error bars for form (22). The dotted line below simulation results represents the reference performace $T = n$.

results for the case $n = 100$. We see that many of the tested cases produce similar results. Also, it seems that parameter η_2 does not change much, if at all, during the optimization. The best results achieved, with overhead percent around 37% match the results generated with only one parameter. This means that in order to produce better distributions, the form of the parameterized distribution should be changed.

C. Sparse Degree Distributions

In [5] it was found out that there are only few conditions to be satisfied in order to get close to optimal degree distribution and thus we will next consider heuristically chosen sparse distributions. In particular, we consider distributions where positive probabilities are assigned to components whose indices are powers of two and less than n, i.e., for $i = 2^j, j = 0, \ldots, j_{\max}$, with $i_{\max} = 2^{j_{\max}}$ being the highest indice and less than the number of blocks, $i_{\max} < n$.

TABLE II

RESULTS FOR THREE PARAMETER EXAMPLE WITH $n = 100$.

Start parameters	Optimized parameters	Average T	Std $\hat{\sigma}(T)$
$\{0.4, 0.1, 0.4\}$	$\{0.26, 0.1, 0.38\}$	136.8	19.4
$\{0.4, -0.3, 0.4\}$	$\{0.27, -0.26, 0.42\}$	137.5	17.6
$\{0.4, -1.5, 0.4\}$	$\{0.44, -1.52, 0.24\}$	158.0	17.9
$\{0.9, -2.0, 0.4\}$	$\{0.90, -2.0, 0.29\}$	137.5	19.0
$\{0.4, 1.5, 0.23\}$	$\{0.34, 1.5, 0.23\}$	136.6	17.9

TABLE III

OPTIMIZED SPARSE DEGREE DISTRIBUTIONS

n	16	32	64	128
p_1	0.21	0.12	0.09	0.18
p_2	0.47	0.51	0.49	0.33
p_4	0.16	0.28	0.2	0.26
p_8	0.16	0	0.13	0.14
p_{16}	-	0.09	0.02	0.05
p_{32}	-	-	0.07	0.01
p_{64}	-	-	-	0.03
Avegage T	22.5	43.6	81.9	159.8
Std $\hat{\sigma}(T)$	4.2	6.8	7.7	12.1
Ovhd-%	0.41	0.36	0.28	0.25
Ovhd-% for (22)	0.43	0.40	0.37	-

TABLE IV

SPARSE DISTRIBUTIONS FOR $n = 100$

Max i	32	64
p_1	0.18	0.19
p_2	0.34	0.34
p_4	0.27	0.27
p_8	0.12	0.13
p_{16}	0.01	0.03
p_{32}	0.08	0.01
p_{64}	-	0.03
Average T	126.6	126.5
Std $\hat{\sigma}(T)$	10.9	11.2

First we test the algorithm for four different sparse distributions with $n = 16, 32, 64$ and 128. Numerical results for objective *MinAvg* are presented in Table III. The number of simulation samples was limited to one million and the other simulation parameters were similar as before. Initially each non-zero probability was given the same value. For comparison purposes, in the last two rows of Table III we have given the estimated overhead percentage for the sparse degree distributions and geometrical form (22). We see that the performance of the sparse distributions is better. There is great insensitivity in the optimized distributions, i.e., slight perturbation of the probabilities results in similar performance. This is typical for most forms of different degree distributions, which is also easy to confirm by manual tests. For further discussion on this kind of insensitivity see [5].

Next we optimize distributions for $n = 100$, where the last non-zero components are $i_{\max} = 32$ and $i_{\max} = 64$. The results are presented in Table IV. We see that both of the optimized distributions have similar performance. Again, both cases give a better performance than the geometric forms considered in Section III-A, where the overhead for $n = 100$ is roughly 37%.

D. Forms Based on the Soliton Distributions

Even more effective forms are achieved by considering the soliton distributions defined in Section I-A. One characteristic of both soliton distributions is that the probability for degree-one symbols is less than the probability for degree-two sym-

TABLE V

RESULTS FROM 10000 RUNS OF LT FOR $n = 100$

Distribution	Average T	Std $\hat{\sigma}(T)$
(24)	125.0	13.1
(24) with spike at $i = 50$	123.9	9.9
ideal soliton	169.5	72
robust soliton, $\sigma = 0.5, c = 0.01$	148.5	44.8
robust soliton, $\sigma = 0.5, c = 0.03$	134.9	23.9
robust soliton, $\sigma = 0.5, c = 0.1$	132.9	13.3

TABLE VI

RESULTS FROM 1000 RUNS OF LT PROCESS FOR $n = 1000$.

Distribution	Average T	Std $\hat{\sigma}(T)$
(24)	1130	84
(24) with spike at $i = 50$	1122	60.8
(24) with spike at $i = 100$	1121	37
robust soliton, $\sigma = 0.5, c = 0.01$	1185	150
robust soliton, $\sigma = 0.5, c = 0.03$	1128	65
robust soliton, $\sigma = 0.5, c = 0.1$	1177	36
robust soliton, $\sigma = 0.9, c = 0.04$	1124	57

bols. This should ensure that not too many redundant degree-one packets are sent, resulting in more efficient transmission. However, the ideal soliton distribution itself performs rather poorly, as is well-known. To improve it, we take the first two degree probabilities as parameters and define the rest of our distribution to be the ideal soliton distribution, i.e.,

$$p_i = \begin{cases} \eta_1, & \text{for} \quad i = 1, \\ \eta_2, & \text{for} \quad i = 2, \\ \frac{1}{i(i-1)}, & \text{for} \quad i = 3, \ldots, n. \end{cases} \tag{24}$$

This needs to be normalized to get proper probabilities for each component.

We ran the ISG-algorithm with a similar setup as before for $n = 100$, allowing the generation of 10^6 samples if unless the threshold of 0.1 is met before. Bisection search was performed in the interval $[0, 0.05]$ with threshold of 0.0001. The maximum number of iterations was again set to 15. The starting point was set at $\eta = (0.2\ 0.2)$.

The optimized values after 15 iterations were $\eta_{\text{opt}} = (0.09\ 0.36)$. 10000 runs of the LT process defined by corresponding degree distribution shows the overhead of around 25%, clearly a better result than with the previous geometric forms and similar in performance to distributions presented in Table IV.

The question of whether the spike present in robust soliton distribution is really necessary leads us to consider a slightly modified form of (24). We ran the ISG-algorithm with the same setup as above, but with an extra parameter for the probability of degree 50 packets. We started from $\eta = (0.2\ 0.2\ 0.2)$ and arrived at the optimized parameters $\eta_{\text{opt}} = (0.083\ 0.487\ 0.032)$. With this form the overhead is around 24%, again a slightly better result than with (24) and similar to the sparse distributions. Exact statistics are presented in Table V, where we have also included statistics of ideal and robust soliton distributions with different parameters.

Table V shows that while the robust soliton distribution performs much better than the ideal soliton distribution, our algorithm was able to find even better distributions first of all by simply using the first two probabilities of ideal soliton distribution as free parameters, and even better by introducing a spike. The behavior of ideal soliton distribution, as explained in Section I-A, is clearly very poor in real situations. With $n = 100$ the first two probabilities of ideal soliton distribution are 0.01 and 0.5, respectively, but our results show that the optimized (and normalized) values of 0.1 and 0.38 give a much better performance.

As (5) cannot be properly differentiated our algorithm cannot be used directly to optimize these parameters. Thus, it is unclear if there is a robust soliton distribution that gives better results for $n = 100$. A form where the tail distribution ensures that there is enough packets of high degrees could probably eliminate the need for the spike.

Fig. 4 shows the histograms of 10000 simulations of the LT processes run with our parameterized distribution resembling the robust soliton distribution and with the real robust soliton distribution. With our distribution the worst case requires nearly always less than 150 packets, with robust soliton distribution the tail goes much further, with several hundred cases with over 160 packets. We also note that with our optimized distribution, the average number of packet degrees was 7.7, and with the robust soliton distribution this was 8.6. This means that less operations are needed for decoding when using the form (24) with spike at $i = 50$ and parameter values $\eta_{\text{opt}} = (0.083\ 0.487\ 0.032)$, resulting in a little better decoding performance.

E. Tests With Larger n

As stated before, our implementation of the ISG-algorithm requires some optimization in order to work in reasonable time for n larger than 100. Nevertheless, we run some LT process simulations with $n = 1000$ to see how our optimized distribution for $n = 100$ performs when compared to the robust soliton distribution. To our surprise, (24) outperforms the robust soliton distribution with $n = 1000$, at least with a wide range of tested parameters.

The results presented in Table VI show that out of the tested distributions, the best performance was provided by our optimized form with the spike moved to $i = 100$. While the form with spike at $i = 50$ provides the same overhead as the form with $i = 100$, the standard deviation is much larger. The

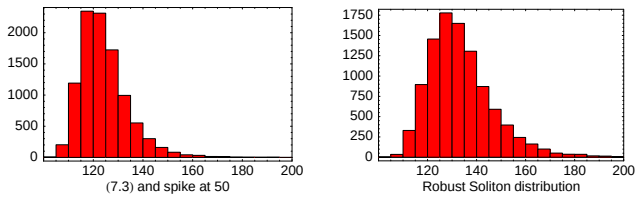

Fig. 4. Histograms of number of packets needed for successful decoding with our best parameterized form and with the robust soliton distribution for $n = 100$ with 10000 simulation runs.

values of the parameters used in (24) were the same as before, $\eta_{\text{opt}} = (0.083\ 0.487\ 0.032)$, regardless of the location of the spike. Without the spike, we see that the standard deviation is quite high. This could imply that the spike (or at least more probability mass at the tail distribution) is needed for a reasonable standard deviation.

The choice of the right parameter values with robust soliton distribution seems to be very important and a bad choice leads to poor performance. We did not find parameter combinations which would outperform the form (24) with optimized parameter values $\eta_{\text{opt}} = (0.083\ 0.487\ 0.032)$ neither for $n = 100$ nor for $n = 1000$. If still better distributions exist, that remains an open question and area for further work.

IV. Conclusions

We have proposed an iterative algorithm for optimization of the degree distribution used in LT codes. The basic idea of the algorithm is based on optimization of an objective function which is constructed using importance ratios, an idea borrowed from importance sampling theory. We have validated the correctness of this algorithm by means of numerical examples where the results have been compared to other numerical results calculated with the previously published degree distributions.

In agreement with [5], it seems that a suitable degree distribution needs to satisfy only few essential conditions. The soliton distribution is a good starting point, where most of the probability mass is situated on the low degrees. One characteristic is the proportion of probabilities of degree-one and two. The best distributions have in common that degree-two probability is the largest of all probabilities and only a fraction of the probability mass is needed for degree-one packets. Degree one packets are naturally vital for the correct function of the decoding algorithm, but too many of them makes the coding somewhat inefficient due to the redundancy of information. We also found out that the use of geometric forms result in worse performance compared to other tested forms. With geometric forms, the probability for degree-one packets was higher than degree-two packets, which was not the case with other, and better, distributions.

In addition, some probability assigned to higher components, i.e., a spike at some relatively high degree makes the performance considerably better. The use of the robust soliton distribution can be justified by the same argument. The spike provides sufficient amount of packets which have high enough degree to keep the decoding process alive at the end of the decoding process.

The optimized distributions are not very sensitive to perturbation of the different parameters as slightly modified distributions provide similar results. This is also a downside of the developed optimization algorithm; the objective function is insensitive to small changes in degree distributions near the optimum, and thus, with the simulation noise included, the degree distribution can be optimized only to certain accuracy.

We also conclude that both of the considered optimization objectives lead to a distribution with similar performance and that objective *MaxPr* is not feasible for large n due to the fact that the probability of recovering a message after receiving the first n packets diminishes and thus most simulation samples have no contribution to the estimate, and consequently are useless for the optimization.

The optimal forms of degree distributions for different message lengths continue to provide an interesting optimization problem. For large n (thousands), the robust soliton distributions have shown good performance [6], [7], [4]. For smaller n we have presented some alternatives in this paper but still the question of what really is the optimal form remains open.

References

[1] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, "A digital fountain approach to reliable distribution of bulk data," in *SIGCOMM*, 1998, pp. 56–67.

[2] Irvin Reed and Gustave Solomon, "Polynomial codes over certain finite fields," *SIAM Journal of Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[3] Robert G. Gallager, "Low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[4] Michael Luby, "LT Codes," in *Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 271–282.

[5] Esa Hyytiä, Tuomas Tirronen, and Jorma Virtamo, "Optimal degree distribution for LT codes with small message length," submitted, June 2006.

[6] David J.C. MacKay, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, 2004.

[7] Patrick Farrell and Jorge Castinera Moreira, *Essentials of Error–Control Coding*, John Wiley and Sons, 2006.

[8] Reuven Y. Rubinstein and Benjamin Melamed, *Modern Simulation and Modeling*, Wiley Series in Probability and Statistics. John Wiley & Sons Inc., 1998.

[9] Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, John Wiley and Sons, Inc., 2nd edition, 1993.