About Chapter 50

The following exercise provides a helpful background for digital fountain codes.

▷ Exercise 50.1.[3] An author proofreads his $K = 700$-page book by inspecting random pages. He makes N page-inspections, and does not take any precautions to avoid inspecting the same page twice.

(a) After $N = K$ page-inspections, what fraction of pages do you expect have never been inspected?

(b) After $N > K$ page-inspections, what is the probability that one or more pages have never been inspected?

(c) Show that in order for the probability that all K pages have been inspected to be $1 - \delta$, we require $N \simeq K \ln(K/\delta)$ page-inspections.

[This problem is commonly presented in terms of throwing N balls at random into K bins; what's the probability that every bin gets at least one ball?]

50

Digital Fountain Codes

Digital fountain codes are record-breaking sparse-graph codes for channels with erasures.

Channels with erasures are of great importance. For example, files sent over the internet are chopped into packets, and each packet is either received without error or not received. A simple channel model describing this situation is a q-ary erasure channel, which has (for all inputs in the input alphabet $\{0, 1, 2, \ldots, q{-}1\}$) a probability $1 - f$ of transmitting the input without error, and probability f of delivering the output '?'. The alphabet size q is 2^l, where l is the number of bits in a packet.

Common methods for communicating over such channels employ a feedback channel from receiver to sender that is used to control the retransmission of erased packets. For example, the receiver might send back messages that identify the *missing* packets, which are then retransmitted. Alternatively, the receiver might send back messages that acknowledge each *received* packet; the sender keeps track of which packets have been acknowledged and retransmits the others until all packets have been acknowledged.

These simple retransmission protocols have the advantage that they will work regardless of the erasure probability f, but purists who have learned their Shannon theory will feel that these retransmission protocols are wasteful. If the erasure probability f is large, the number of feedback messages sent by the first protocol will be large. Under the second protocol, it's likely that the receiver will end up receiving multiple redundant copies of some packets, and heavy use is made of the feedback channel. According to Shannon, there is no need for the feedback channel: the capacity of the forward channel is $(1 - f)l$ bits, whether or not we have feedback.

The wastefulness of the simple retransmission protocols is especially evident in the case of a broadcast channel with erasures – channels where one sender broadcasts to many receivers, and each receiver receives a random fraction $(1 - f)$ of the packets. If every packet that is missed by one or more receivers has to be retransmitted, those retransmissions will be terribly redundant. Every receiver will have already received most of the retransmitted packets.

So, we would like to make erasure-correcting codes that require no feedback or almost no feedback. The classic block codes for erasure correction are called Reed–Solomon codes. An (N, K) Reed–Solomon code (over an alphabet of size $q = 2^l$) has the ideal property that if any K of the N transmitted symbols are received then the original K source symbols can be recovered. [See Berlekamp (1968) or Lin and Costello (1983) for further information; Reed–Solomon codes exist for $N < q$.] But Reed–Solomon codes have the disadvantage that they are practical only for small K, N, and q: standard im-

plementations of encoding and decoding have a cost of order $K(N-K)\log_2 N$ packet operations. Furthermore, with a Reed–Solomon code, as with any block code, one must estimate the erasure probability f and choose the code rate $R = K/N$ *before* transmission. If we are unlucky and f is larger than expected and the receiver receives fewer than K symbols, what are we to do? We'd like a simple way to extend the code on the fly to create a lower-rate (N', K) code. For Reed–Solomon codes, no such on-the-fly method exists.

There is a better way, pioneered by Michael Luby (2002) at his company Digital Fountain, the first company whose business is based on sparse-graph codes.

The digital fountain codes I describe here, *LT codes*, were invented by Luby in 1998. The idea of a digital fountain code is as follows. The encoder is a fountain that produces an endless supply of water drops (encoded packets); let's say the original source file has a size of Kl bits, and each drop contains l encoded bits. Now, anyone who wishes to receive the encoded file holds a bucket under the fountain and collects drops until the number of drops in the bucket is a little larger than K. They can then recover the original file.

LT stands for 'Luby transform'.

Digital fountain codes are *rateless* in the sense that the number of encoded packets that can be generated from the source message is potentially limitless; and the number of encoded packets generated can be determined on the fly. Regardless of the statistics of the erasure events on the channel, we can send as many encoded packets as are needed in order for the decoder to recover the source data. The source data can be decoded from any set of K' encoded packets, for K' slightly larger than K (in practice, about 5% larger).

Digital fountain codes also have fantastically small encoding and decoding complexities. With probability $1 - \delta$, K packets can be communicated with average encoding and decoding costs both of order $K \ln(K/\delta)$ packet operations.

Luby calls these codes *universal* because they are simultaneously near-optimal for every erasure channel, and they are very efficient as the file length K grows. The overhead $K' - K$ is of order $\sqrt{K}(\ln(K/\delta))^2$.

▶ 50.1 A digital fountain's encoder

Each encoded packet t_n is produced from the source file $s_1 s_2 s_3 \ldots s_K$ as follows:

1. Randomly choose the degree d_n of the packet from a degree distribution $\rho(d)$; the appropriate choice of ρ depends on the source file size K, as we'll discuss later.

2. Choose, uniformly at random, d_n distinct input packets, and set t_n equal to the bitwise sum, modulo 2 of those d_n packets. This sum can be done by successively exclusive-or-ing the packets together.

This encoding operation defines a graph connecting encoded packets to source packets. If the mean degree $\bar{d}$ is significantly smaller than K then the graph is sparse. We can think of the resulting code as an irregular low-density generator-matrix code.

The decoder needs to know the degree of each packet that is received, and which source packets it is connected to in the graph. This information can be communicated to the decoder in various ways. For example, if the sender and receiver have synchronized clocks, they could use identical pseudo-random

number generators, seeded by the clock, to choose each random degree and each set of connections. Alternatively, the sender could pick a random key, κ_n, given which the degree and the connections are determined by a pseudo-random process, and send that key in the header of the packet. As long as the packet size l is much bigger than the key size (which need only be 32 bits or so), this key introduces only a small overhead cost.

▶ 50.2 The decoder

Decoding a sparse-graph code is especially easy in the case of an erasure channel. The decoder's task is to recover $\mathbf{s}$ from $\mathbf{t} = \mathbf{G}\mathbf{s}$, where $\mathbf{G}$ is the matrix associated with the graph. The simple way to attempt to solve this problem is by message-passing. We can think of the decoding algorithm as the sum–product algorithm if we wish, but all messages are either *completely uncertain* messages or *completely certain* messages. Uncertain messages assert that a message packet s_k could have any value, with equal probability; certain messages assert that s_k has a particular value, with probability one.

This simplicity of the messages allows a simple description of the decoding process. We'll call the encoded packets $\{t_n\}$ check nodes.

1. Find a check node t_n that is connected to *only one* source packet s_k. (If there is no such check node, this decoding algorithm halts at this point, and fails to recover all the source packets.)
   (a) Set $s_k = t_n$.
   (b) Add s_k to all checks $t_{n'}$ that are connected to s_k:
   $$t_{n'} := t_{n'} + s_k \quad \text{for all } n' \text{ such that } G_{n'k} = 1. \tag{50.1}$$
   (c) Remove all the edges connected to the source packet s_k.
2. Repeat (1) until all $\{s_k\}$ are determined.

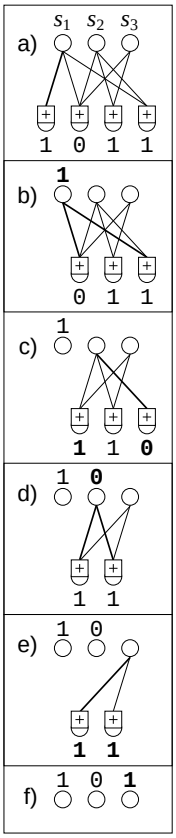

Figure 50.1. Example decoding for a digital fountain code with $K = 3$ source bits and $N = 4$ encoded bits.

This decoding process is illustrated in figure 50.1 for a toy case where each packet is just one bit. There are three source packets (shown by the upper circles) and four received packets (shown by the lower check symbols), which have the values $t_1t_2t_3t_4 = \texttt{1011}$ at the start of the algorithm.

At the first iteration, the only check node that is connected to a sole source bit is the first check node (panel a). We set that source bit s_1 accordingly (panel b), discard the check node, then add the value of s_1 (1) to the checks to which it is connected (panel c), disconnecting s_1 from the graph. At the start of the second iteration (panel c), the fourth check node is connected to a sole source bit, s_2. We set s_2 to t_4 (0, in panel d), and add s_2 to the two checks it is connected to (panel e). Finally, we find that two check nodes are both connected to s_3, and they agree about the value of s_3 (as we would hope!), which is restored in panel f.

▶ 50.3 Designing the degree distribution

The probability distribution $\rho(d)$ of the degree is a critical part of the design: occasional encoded packets must have high degree (i.e., d similar to K) in order to ensure that there are not some source packets that are connected to no-one. Many packets must have low degree, so that the decoding process

can get started, and keep going, and so that the total number of addition operations involved in the encoding and decoding is kept small. For a given degree distribution $\rho(d)$, the statistics of the decoding process can be predicted by an appropriate version of density evolution.

Ideally, to avoid redundancy, we'd like the received graph to have the property that just one check node has degree one at each iteration. At each iteration, when this check node is processed, the degrees in the graph are reduced in such a way that one new degree-one check node appears. *In expectation*, this ideal behaviour is achieved by the *ideal soliton distribution*,

$$\begin{aligned} \rho(1) &= 1/K \\ \rho(d) &= \frac{1}{d(d-1)} \quad \text{for } d = 2, 3, \ldots, K. \end{aligned} \tag{50.2}$$

The expected degree under this distribution is roughly $\ln K$.

▷ Exercise 50.2.[2] Derive the ideal soliton distribution. At the first iteration ($t = 0$) let the number of packets of degree d be $h_0(d)$; show that (for $d > 1$) the expected number of packets of degree d that have their degree reduced to $d - 1$ is $h_0(d)d/K$; and at the tth iteration, when t of the K packets have been recovered and the number of packets of degree d is $h_t(d)$, the expected number of packets of degree d that have their degree reduced to $d - 1$ is $h_t(d)d/(K - t)$. Hence show that in order to have the expected number of packets of degree 1 satisfy $h_t(1) = 1$ for all $t \in \{0, \ldots K - 1\}$, we must to start with have $h_0(1) = 1$ and $h_0(2) = K/2$; and more generally, $h_t(2) = (K - t)/2$; then by recursion solve for $h_0(d)$ for $d = 3$ upwards.

This degree distribution works poorly in practice, because fluctuations around the expected behaviour make it very likely that at some point in the decoding process there will be no degree-one check nodes; and, furthermore, a few source nodes will receive no connections at all. A small modification fixes these problems.

The *robust soliton distribution* has two extra parameters, c and δ; it is designed to ensure that the expected number of degree-one checks is about

$$S \equiv c \ln(K/\delta)\sqrt{K}, \tag{50.3}$$

rather than 1, throughout the decoding process. The parameter δ is a bound on the probability that the decoding fails to run to completion after a certain number K' of packets have been received. The parameter c is a constant of order 1, if our aim is to prove Luby's main theorem about LT codes; in practice however it can be viewed as a free parameter, with a value somewhat smaller than 1 giving good results. We define a positive function

$$\tau(d) = \begin{cases} \frac{S}{K}\frac{1}{d} & \text{for } d = 1, 2, \ldots (K/S)-1 \\ \frac{S}{K}\ln(S/\delta) & \text{for } d = K/S \\ 0 & \text{for } d > K/S \end{cases} \tag{50.4}$$

(see figure 50.2 and exercise 50.4 (p.594)) then add the ideal soliton distribution ρ to τ and normalize to obtain the robust soliton distribution, μ:

$$\mu(d) = \frac{\rho(d) + \tau(d)}{Z}, \tag{50.5}$$

where $Z = \sum_d \rho(d) + \tau(d)$. The number of encoded packets required at the receiving end to ensure that the decoding can run to completion, with probability at least $1 - \delta$, is $K' = KZ$.

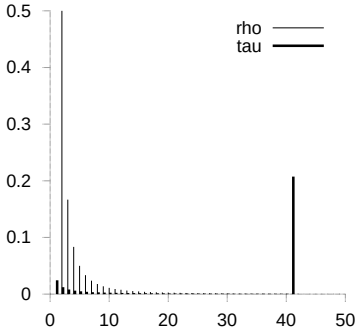

Figure 50.2. The distributions $\rho(d)$ and $\tau(d)$ for the case $K = 10\,000$, $c = 0.2$, $\delta = 0.05$, which gives $S = 244$, $K/S = 41$, and $Z \simeq 1.3$. The distribution τ is largest at $d = 1$ and $d = K/S$.

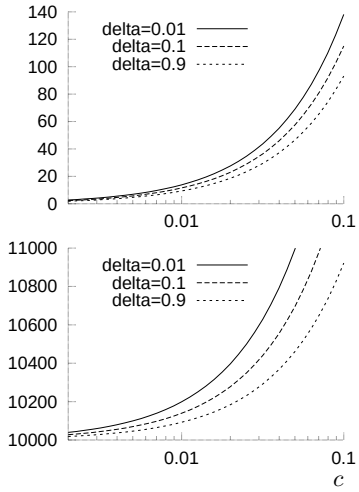

Figure 50.3. The number of degree-one checks S (upper figure) and the quantity K' (lower figure) as a function of the two parameters c and δ, for $K = 10\,000$. Luby's main theorem proves that there exists a value of c such that, given K' received packets, the decoding algorithm will recover the K source packets with probability $1 - \delta$.

Luby's (2002) analysis explains how the small-d end of τ has the role of ensuring that the decoding process gets started, and the spike in τ at $d = K/S$ is included to ensure that every source packet is likely to be connected to a check at least once. Luby's key result is that (for an appropriate value of the constant c) receiving $K' = K + 2\ln(S/\delta)S$ checks ensures that all packets can be recovered with probability at least $1 - \delta$. In the illustrative figures I have set the allowable decoder failure probability δ quite large, because the actual failure probability is much smaller than is suggested by Luby's conservative analysis.

In practice, LT codes can be tuned so that a file of original size $K \simeq 10\,000$ packets is recovered with an overhead of about 5%. Figure 50.4 shows histograms of the actual number of packets required for a couple of settings of the parameters, achieving mean overheads smaller than 5% and 10% respectively.

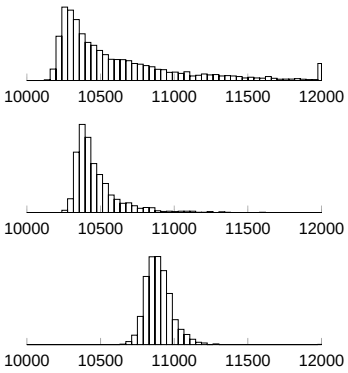

Figure 50.4. Histograms of the actual number of packets N required in order to recover a file of size $K = 10\,000$ packets. The parameters were as follows: top histogram: $c = 0.01$, $\delta = 0.5$ ($S = 10$, $K/S = 1010$, and $Z \simeq 1.01$); middle: $c = 0.03$, $\delta = 0.5$ ($S = 30$, $K/S = 337$, and $Z \simeq 1.03$); bottom: $c = 0.1$, $\delta = 0.5$ ($S = 99$, $K/S = 101$, and $Z \simeq 1.1$).

▶ 50.4 Applications

Digital fountain codes are an excellent solution in a wide variety of situations. Let's mention two.

Storage

You wish to make a backup of a large file, but you are aware that your magnetic tapes and hard drives are all unreliable in the sense that catastrophic failures, in which some stored packets are permanently lost within one device, occur at a rate of something like 10^{-3} per day. How should you store your file?

A digital fountain can be used to spray encoded packets all over the place, on every storage device available. Then to recover the backup file, whose size was K packets, one simply needs to find $K' \simeq K$ packets from anywhere. Corrupted packets do not matter; we simply skip over them and find more packets elsewhere.

This method of storage also has advantages in terms of *speed* of file recovery. In a hard drive, it is standard practice to store a file in successive sectors of a hard drive, to allow rapid reading of the file; but if, as occasionally happens, a packet is lost (owing to the reading head being off track for a moment, giving a burst of errors that cannot be corrected by the packet's error-correcting code), a whole revolution of the drive must be performed to bring back the packet to the head for a second read. The time taken for one revolution produces an undesirable delay in the file system.

If files were instead stored using the digital fountain principle, with the digital drops stored in one or more consecutive sectors on the drive, then one would never need to endure the delay of re-reading a packet; packet loss would become less important, and the hard drive could consequently be operated faster, with higher noise level, and with fewer resources devoted to noisy-channel coding.

▷ Exercise 50.3.[2] Compare the digital fountain method of robust storage on multiple hard drives with RAID (the redundant array of independent disks).

Broadcast

Imagine that ten thousand subscribers in an area wish to receive a digital movie from a broadcaster. The broadcaster can send the movie in packets

over a broadcast network – for example, by a wide-bandwidth phone line, or by satellite.

Imagine that not all packets are received at all the houses. Let's say $f = 0.1\%$ of them are lost at each house. In a standard approach in which the file is transmitted as a plain sequence of packets with no encoding, each house would have to notify the broadcaster of the fK missing packets, and request that they be retransmitted. And with ten thousand subscribers all requesting such retransmissions, there would be a retransmission request for almost every packet. Thus the broadcaster would have to repeat the entire broadcast twice in order to ensure that most subscribers have received the whole movie, and most users would have to wait roughly twice as long as the ideal time before the download was complete.

If the broadcaster uses a digital fountain to encode the movie, each subscriber can recover the movie from *any* $K' \simeq K$ packets. So the broadcast needs to last for only, say, $1.1K$ packets, and every house is very likely to have successfully recovered the whole file.

Another application is broadcasting data to cars. Imagine that we want to send updates to in-car navigation databases by satellite. There are hundreds of thousands of vehicles, and they can receive data only when they are out on the open road; there are no feedback channels. A standard method for sending the data is to put it in a *carousel*, broadcasting the packets in a fixed periodic sequence. 'Yes, a car may go through a tunnel, and miss out on a few hundred packets, but it will be able to collect those missed packets an hour later when the carousel has gone through a full revolution (we hope); or maybe the following day...'

If instead the satellite uses a digital fountain, each car needs to receive only an amount of data equal to the original file size (plus 5%).

Further reading

The encoders and decoders sold by Digital Fountain have even higher efficiency than the LT codes described here, and they work well for all blocklengths, not only large lengths such as $K \gtrsim 10\,000$. Shokrollahi (2003) presents *raptor codes*, which are an extension of LT codes with linear-time encoding and decoding.

▶ 50.5 Further exercises

▷ Exercise 50.4.[2] Understanding the robust soliton distribution.

Repeat the analysis of exercise 50.2 (p.592) but now aim to have the expected number of packets of degree 1 be $h_t(1) = 1 + S$ for all t, instead of 1. Show that the initial required number of packets is

$$h_0(d) = \frac{K}{d(d-1)} + \frac{S}{d} \quad \text{for } d > 1. \tag{50.6}$$

The reason for truncating the second term beyond $d = K/S$ and replacing it by the spike at $d = K/S$ (see equation (50.4)) is to ensure that the decoding complexity does not grow larger than $O(K \ln K)$.

Estimate the expected number of packets $\sum_d h_0(d)$ and the expected number of edges in the sparse graph $\sum_d h_0(d)d$ (which determines the decoding complexity) if the histogram of packets is as given in (50.6). Compare with the expected numbers of packets and edges when the robust soliton distribution (50.4) is used.

Exercise 50.5.[4] Show that the spike at $d = K/S$ (equation (50.4)) is an adequate replacement for the tail of high-weight packets in (50.6).

Exercise 50.6.[3C] Investigate experimentally how necessary the spike at $d = K/S$ (equation (50.4)) is for successful decoding. Investigate also whether the tail of $\rho(d)$ beyond $d = K/S$ is necessary. What happens if all high-weight degrees are removed, both the spike at $d = K/S$ and the tail of $\rho(d)$ beyond $d = K/S$?

Exercise 50.7.[4] Fill in the details in the proof of Luby's main theorem, that receiving $K' = K + 2\ln(S/\delta)S$ checks ensures that all the source packets can be recovered with probability at least $1 - \delta$.

Exercise 50.8.[4C] Optimize the degree distribution of a digital fountain code for a file of $K = 10\,000$ packets. Pick a sensible objective function for your optimization, such as minimizing the mean of N, the number of packets required for complete decoding, or the 95th percentile of the histogram of N (figure 50.4).

▷ Exercise 50.9.[3] Make a model of the situation where a data stream is broadcast to cars, and quantify the advantage that the digital fountain has over the carousel method.

Exercise 50.10.[2] Construct a simple example to illustrate the fact that the digital fountain decoder of section 50.2 is suboptimal – it sometimes gives up even though the information available is sufficient to decode the whole file. How does the cost of the optimal decoder compare?

▷ Exercise 50.11.[2] If every transmitted packet were created by adding together source packets at random with probability $1/2$ of each source packet's being included, show that the probability that $K' = K$ received packets suffice for the optimal decoder to be able to recover the K source packets is just a little below $1/2$. [To put it another way, what is the probability that a random $K \times K$ matrix has full rank?]

Show that if $K' = K + \Delta$ packets are received, the probability that they will not suffice for the optimal decoder is roughly $2^{-\Delta}$.

▷ Exercise 50.12.[4C] Implement an optimal digital fountain decoder that uses the method of Richardson and Urbanke (2001b) derived for fast *encoding* of sparse-graph codes (section 47.7) to handle the matrix inversion required for optimal decoding. Now that you have changed the decoder, you can reoptimize the degree distribution, using higher-weight packets. By how much can you reduce the overhead? Confirm the assertion that this approach makes digital fountain codes viable as erasure-correcting codes for all blocklengths, not just the large blocklengths for which LT codes are excellent.

▷ Exercise 50.13.[5] Digital fountain codes are excellent rateless codes for erasure channels. Make a rateless code for a channel that has both erasures and *noise*.

▶ 50.6 Summary of sparse-graph codes

A simple method for designing error-correcting codes for noisy channels, first pioneered by Gallager (1962), has recently been rediscovered and generalized, and communication theory has been transformed. The practical performance of Gallager's low-density parity-check codes and their modern cousins is vastly better than the performance of the codes with which textbooks have been filled in the intervening years.

Which sparse-graph code is 'best' for a noisy channel depends on the chosen rate and blocklength, the permitted encoding and decoding complexity, and the question of whether occasional undetected errors are acceptable. Low-density parity-check codes are the most versatile; it's easy to make a competitive low-density parity-check code with almost any rate and blocklength, and low-density parity-check codes virtually never make undetected errors.

For the special case of the erasure channel, the sparse-graph codes that are best are digital fountain codes.

▶ 50.7 Conclusion

The best solution to the communication problem is:

> Combine a simple, pseudo-random code
> with a message-passing decoder.