COMP3491 Codes and Cryptography
Academic year 2020-21

Maximilien Gadouleau

Version 2021.04.09

Contents

Chapter 1

Data Compression

Data compression is the art and science of removing redundancy from data efficiently. There are two main kinds of data compression: lossless (the data can be fully recovered after compression/decompression) and lossy (the data cannot be fully recovered, but instead some loss of quality is tolerated).

The first five sections deal with lossless compression; the last five deal with lossy compression.

1 Statistical compression I: Huffman coding

1.1 Prefix codes

Memoryless sources We have some data that we wish to encode. It could be anything: Spoken English, Data from a digital camera sensor, DNA string, etc.

We model our data as coming from a memoryless source X. We imagine that symbols are emitted at random according to the probability distribution of X. In other words, we view our data as a random string $X_1, X_2, \ldots$ over some alphabet $\mathscr{X}$. Our memoryless assumption is that those form a sequence of independent identically distributed (i.i.d.) random variables: $X_i \sim X$ for all i.

More concretely, for any $x \in \mathscr{X}$ and any i, the probability

$$\mathbb{P}(X_i = x)$$

is independent of i, and of all previous or future emitted symbols.

Note that this is not always a valid assumption. We will look into source modelling into more detail in the next lectures.

The coding problem We have a source emitting symbols in $\mathscr{X} = \{x_1, \ldots, x_n\}$ with respective probabilities $\{p_1, \ldots, p_n\}$.

Question: If $\mathscr{D}$ is an alphabet of D code symbols, how can we encode the source symbols using code words (finite strings of code symbols) as economically as possible?

Formally: a source code is a map $C : \mathscr{X} \to \mathscr{D}^*$ where $\mathscr{D}^*$ is the set of all finite strings of symbols in $\mathscr{D}$.

The words $C(x)$ are called the codewords, and the integers $|C(x)|$ (the length of $C(x)$) are the word lengths.

We can extend the code to messages as follows. A message is any finite string of source symbols $m = m_1 \ldots m_k \in \mathscr{X}^*$ and its encoding is the obvious concatenation

$$C(m) = C(m_1)C(m_2)\ldots C(m_k).$$

Prefix codes A code C is uniquely decodable (a.k.a. uniquely decipherable) if every finite string in $\mathscr{D}^*$ is the image of at most one message.

A prefix of a word $w = w_1 \ldots w_k \in \mathscr{D}^*$ is any word of the form $w_1 \ldots w_l$ for some $0 \le l \le k$ (for $l = 0$, we obtain the empty word). A code is prefix (a.k.a. instantaneous or prefix-free) if there are no two distinct source symbols $x, y \in \mathscr{X}$ such that $C(x)$ is a prefix of $C(y)$.

Theorem 1.1. *A prefix code is uniquely decodable.*

Proof. Let C be a prefix code, and let $w = C(m)$ for some message $m = m_1 \dots m_k \in \mathscr{X}^*$. We give a decoding algorithm which, given w, determines m. Let $w = w_1 \dots w_l$.

Let i be the smallest integer such that $w_1 \dots w_i$ is a codeword, say $w_1 \dots w_i = C(x)$. Then the $m_1 = x$. Indeed, if $m_1 = y \neq x$, then $C(x)$ is a prefix of $C(y)$, which is a contradiction. Then repeat this step, beginning with w_{i+1} and hence determining m_2, and so on until w is empty. □

Example 1.2. *Let* $\mathscr{X} = \{a,b,c,d,e\}$, $\mathscr{D} = \{0,1\}$ *and*

$$\begin{aligned} C(a) &= 01 \\ C(b) &= 100 \\ C(c) &= 101 \\ C(d) &= 1101 \\ C(e) &= 1111. \end{aligned}$$

Suppose we need to decode the word $C(m) = w = 10010111011111100101$. *We proceed as follows. We read the word until we reach a codeword:*

$$\begin{aligned} w_1 &= 1 \\ w_1 w_2 &= 10 \\ w_1 w_2 w_3 &= 100 = C(b). \end{aligned}$$

Therefore $m_1 = b$. *We continue until we reach a codeword:*

$$\begin{aligned} w_4 &= 1 \\ w_4 w_5 &= 10 \\ w_4 w_5 w_6 &= 101 = C(c). \end{aligned}$$

Therefore $m_2 = c$. *And so on... Exercise 1.2 asks you to finish this simple example.*

1.2 Huffman codes

Compact codes Our main aim is to design codes where the typical length of messages is reduced dramatically. The basic idea is to assign short codewords to more frequent symbols and longer codewords to less frequent ones.

More formally, the average length (a.k.a. expected length) of the code is

$$L(C) = \mathbb{E}(|C(X)|) = \sum_{x \in \mathscr{X}} |C(x)| \mathbb{P}(X = x).$$

A code is compact (for a given source X) if it is uniquely decodable and it minimises the average length of codewords over all uniquely decodable codes.

Theorem 1.3. *A uniquely decodable code with prescribed word lengths exists if and only if a prefix code with the same word lengths exists.*

We shall prove this result in Lecture 35.

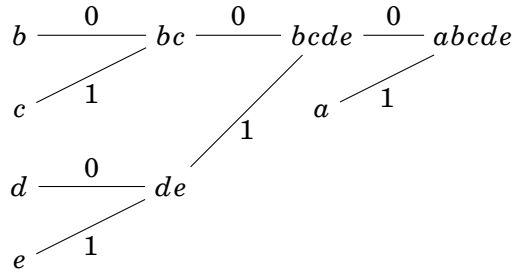
Corollary 1.4. *For any source* X, *there is a compact prefix code for* X.

Binary Huffman code The key is to construct a tree where the leaves correspond to the symbols in $\mathscr{X}$ and the paths from the root to the leaves give the codewords.

The tree is constructed iteratively. Suppose $\mathscr{X} = \{x_1, \dots, x_n\}$ with $p_1 \geq p_2 \geq \cdots \geq p_{n-1} \geq p_n$. Then merge x_{n-1} and x_n into a new symbol, say $x_{n-1,n}$ with probability $p_{n-1} + p_n$, and let x_{n-1} and x_n be the children of $x_{n-1,n}$ on the tree. Label the edges from $x_{n-1,n}$ to its children as 0 and 1, respectively. Repeat for the new source $X^{(1)} = \{x_1, \dots, x_{n-2}, x_{n-1,n}\}$ (making sure to order the symbols in non-decreasing probability). Repeat until the final source $X^{(n-1)}$ only has one symbol left with probability 1; that symbol is the root of the tree.

Once the tree is built, read off the labels on the path from the root to a leaf to get the corresponding codeword.

Example 1.5. *Let X with respective probabilities $a : 0.4, b : 0.2, c : 0.15, d : 0.15, e : 0.1$.*



The code is then

$$\begin{aligned} C(a) &= 1 \\ C(b) &= 000 \\ C(c) &= 001 \\ C(d) &= 010 \\ C(e) &= 011 \end{aligned}$$

The average length is then 2.2 *bits per symbol.*

Note that there is no need for general tie-breaking rules. Indeed, different merges may yield different codes, and maybe even different code lengths, but always the same expected length. Similarly, the assignment of 0 or 1 does not change the code lengths.

Huffman codes are compact Clearly, Huffman codes are prefix. The proof that they are compact is by induction on the number of symbols and omitted. It can be found in [5, Section 5.8].

Non-binary Huffman codes Huffman codes can be extended to non-binary alphabets: If we have an alphabet of D characters, we group the D least likely symbols at each stage of reducing the source. When expanding the code we append each of the D characters to one of the least likely symbols' codewords.

We must end up with exactly D symbols in the final source, so we may need to pad the original source up to $D + k(D-1)$ by adding symbols of probability 0.

1.3 See further

Codes and Automata The mathematical theory of uniquely decodable codes is reviewed in [2], where they are simply referred to as codes. The language generated by a prefix code can be recognised by very a simple deterministic finite automaton; in fact, the relation between codes and automata is very deep and explored throughout the book. Note that this book hardly talks about data compression!

Canonical Huffman codes As we shall see in Exercise 1.5, there can be several different Huffman trees for the same source. However, there is always a so-called canonical Huffman tree (and hence code) with a special shape that can be easily computed; see [14, 3.2.2]. We shall encounter a similar idea in Lecture 35.

Adaptive Huffman coding Huffman coding is based on a source X with given probabilities. In general, the probability of an element is computed by its relative frequency in the message; for instance, if the message has 100 characters, 34 of them are "e", then the probability of "e" is 34%. Computing those probabilities then requires scanning the whole document before building the tree. Adaptive Huffman coding, on the other hand, builds the Huffman tree as the document is scanned, making small updates (if any) each time a new character is scanned.

1.4 Exercises

Exercise 1.1. Let $\mathscr{X} = \{x_1, \dots, x_q\}$ for $q \geq 2$. Give a binary code $C : \mathscr{X} \to \{0,1\}^*$ that is uniquely decodable but neither prefix nor suffix.

Exercise 1.2. Finish the example of decoding a prefix code.

Exercise 1.3. How could you make the decoding algorithm of prefix codes more efficient? Would you use that modification for decoding Huffman codes? How would you include the decision problem: given $w \in \mathscr{D}^*$, determine whether w is a codeword.

Exercise 1.4. Construct a binary Huffman code for X with probabilities $0.5, 0.2, 0.15, 0.1, 0.05$. What is the average length, and how does it compare with the one in Example 1.5?

Exercise 1.5. Let X have probabilities $(1/3, 1/3, 1/4, 1/12)$. Show that, depending on how you merge, the binary Huffman coding procedure may lead to different code lengths, namely $(2,2,2,2)$ or $(1,2,3,3)$. Verify that the average length remains the same, though.

2 Statistical compression II: Arithmetic coding

2.1 Arithmetic coding

Limitation of Huffman coding Consider a source with a heavily imbalanced distribution: say $a : 0.99$ and $b : 0.01$. Suppose we want to encode the sequence

$$m = aaaaaaaaaa$$

(of length 10) using Huffman coding. Then we would require 10 bits (the length of the message).

However, if you compute the probability of that particular 10-character sequence, we get

$$p(m) = p(a)^{10} \approx 0.904.$$

So if we were to compute the Huffman code based on all 2^{10} possible sequences, m would be encoded as only one bit!

The main limitation of Huffman is then apparent: the codewords are only defined for symbols, not messages. Arithmetic coding offers a way of working at the sequence level, thereby assigning a particular tag to any sequence, without working out all the tags for all sequences of the same length.

Suppose $X = \{a_1, \dots, a_n\}$ with respective probabilities $p_1, \dots, p_n$. We want to encode the message $m = a_{i_1} \dots a_{i_k}$. The output of the arithmetic encoder will be a number in the range $[0,1)$ that uniquely describes m.

Example 2.1. *Let $X = \{a_1, a_2, a_3\}$ with probabilities $p_1 = 0.4, p_2 = 0.5, p_3 = 0.1$. The interval $[0,1)$ is subdivided among the three symbols as*

$$a_1 : [0, 0.4), \quad a_2 : [0.4, 0.9), \quad a_3 : [0.9, 1).$$

The interval is then recursively subdivided in the same fashion, e.g. $a_1 : [0, 0.4)$ is subdivided into

$$a_1a_1 : [0, 0.16), \quad a_1a_2 : [0.16, 0.36), \quad a_1a_3 : [0.36, 0.4).$$

The final code for a_1a_3 could be any number in the range $[0.36, 0.4)$. The decoding is performed by iteratively performing the splits and choosing the interval where the code belongs. For instance, say we send $c = 0.36$ (obviously, we only send 36), then the decoder first finds out that $c \in [0, 0.4)$ hence $m_1 = a_1$; then $c \in [0.36, 0.4)$ hence $m_2 = a_2$.

For each symbol processed, the current interval gets smaller and requires more bits to express it, but the final output is a single number for the whole sequence, which is not simply the concatenation of the codewords for its symbols. We illustrate the encoding and decoding processes in more detail by using a slightly more complex example.

We show the compression steps for the string "SWISS␣MISS". This time, the probabilities directly arise from the character frequencies, which are computed as a preliminary step to the encoding process.

Character x	Frequency	Probability	Range $[L(x),H(x))$
S	5	0.5	[0.5, 1.0)
W	1	0.1	[0.4, 0.5)
I	2	0.2	[0.2, 0.4)
M	1	0.1	[0.1, 0.2)
␣	1	0.1	[0.0, 0.1)

The encoding process begins by defining two variables Low and $High$ and setting them to 0 and 1, respectively.They define an interval $[Low, High)$. As symbols are being input and processed, the values of $High$ and Low are moved closer together. As the symbol x is being input and processed, Low and $High$ are updated according to

$$High \leftarrow Low + (High - Low)H(x),$$
$$Low \leftarrow Low + (High - Low)L(x).$$

x	$L(x)$	$H(x)$	Low	$High$
			0	1
S	0.5	1.0	0.5	1.0
W	0.4	0.5	0.70	0.75
I	0.2	0.4	0.71	0.72
S	0.5	1.0	0.715	0.72
S	0.5	1.0	0.7175	0.72
␣	0.0	0.1	0.7175	0.71775
M	0.1	0.2	0.717525	0.717550
I	0.2	0.4	0.717530	0.717535
S	0.5	1.0	0.7175325	0.717535
S	0.5	1.0	0.71753375	0.717535

The final code is the final value of Low, 0.71753375 of which only the eight digits 71753375 need to be written.

The decoder first inputs the symbols and their range, and reconstructs the table of frequencies and probabilities. It then inputs the rest of the code. The first digit is 7, so the number is $0.7\ldots \in [0.5,1)$: the first symbol is then S. It carries on, updating the code number to remove the effect of the character it just input. More explicitly, after the character x, it performs the update

$$C \leftarrow \frac{C - L(x)}{H(x) - L(x)}.$$

The decoder carries on until $C = 0$, in which case there should be a way to make it stop (either an end-of-file symbol is part of the input, or the length of the input was given in the header of the code).

x	$L(x)$	$H(x)$	C
			0.71753375
S	0.5	1.0	0.4350675
W	0.4	0.5	0.350675
I	0.2	0.4	0.753375
S	0.5	1.0	0.50675
S	0.5	1.0	0.0135
␣	0.0	0.1	0.135
M	0.1	0.2	0.35
I	0.2	0.4	0.75
S	0.5	1.0	0.5
S	0.5	1.0	0

2.2 Implementation details

Using integers The encoding as described before is not practical, since it uses numbers of unlimited precision for Low and $High$. The decoder process is also impractical: the number C can be a very long integer.

Any practical implementation of arithmetic coding should only use integers and should not be very long. Here is an implementation that uses integers with only four digits (We only give the encoder, but the decoder can be worked out as "doing the same in reverse," as we are getting used to seeing.)

The main idea is that once the leftmost digits of Low and $High$ are equal, then they remain equal henceforth. So we should "forget about" the leftmost digit once the encoder has output it. This is done by shifting the digits. Using four digits, we first initialise $L^* = 0000$ (corresponding to $Low = 0.0000\cdots = 0$) and $H^* = 9999$ (corresponding to $High = 0.9999\cdots = 1$), and we proceed as follows.

x	Low	$High$	Digit	L^*	H^*
	0	1		0000	9999
S	0.5	1		5000	9999
W	0.7	0.75	7	0000	4999
I	0.1	0.2	1	0000	9999
S	0.5	1.0		5000	9999
S	0.75	1.0		7500	9999
␣	0.75	0.775	7	5000	7499
M	0.525	0.55	5	2500	4999
I	0.3	0.35	3	0000	4999
S	0.25	0.5		2500	4999
S	0.375	0.5	3750		4999

In this toy example, we used four digits, but in practice we should be using enough to make sure that enough information is conveyed by H^* and L^* at all times. Another potential issue is that of underflow, when for instance $High$ decreases too fast and loses its significant digits. Scaling is then performed to avoid this situation.

Using binary strings Firstly, note that we can choose to output any number in the range $[Low, High)$, ans not necessarily Low per se. A certain choice of value may have fewer digits in its binary expansion, and hence require less space than Low. Moreover, obviously operations should be carried out in binary instead of decimal.

It can be shown that, if one uses the number $(Low + High)/2$, then one only needs to transmit the first

$$l = \left\lceil \log \frac{1}{p(m)} \right\rceil + 1$$

bits of that number, where $p(m)$ is the probability of the input sequence m. As we shall see in Lecture 35, this is very close to optimality indeed.

2.3 Exercises

Exercise 2.1. With the same source as in Example 2.1, encode the message $m = a_2a_3a_1a_1a_3$.

Exercise 2.2. Work out a decoder for the four-digit implementation of arithmetic coding, and decode the output of the SWISS MISS example.

3 Lempel-Ziv I: LZ77

3.1 Limitations of statistical compression

In Lectures 1 and 2 we looked at compact codes for data being emitted by a memoryless source - a random process.

This week we look at encoding a fixed file of data efficiently. Rather than having estimates for the probabilities of each symbol, we can look at the whole message and determine the frequency of each symbol. Compact codes for memoryless sources are guaranteed to be optimal on average, but we may not have an average message. If the encoding is not determined in advance (as can be done for known sources), but is message dependent, then we must transmit the code as well as the encoded message.

Recall, memoryless sources emit each symbol independently of any previous symbols. There is no 'pattern' to the data beyond the frequency of each symbol. For instance, consider the message

$$abbaeadcaadccbaabaaa$$

(20 characters). The letter frequencies are a:10/20, b:4/20, c:3/20, d:2/20, e:1/20. These agree exactly with the probabilities in Exercise 1.4. Using the Huffman code $abbaeadcaadccbaabaaa$ becomes

$$10000101111011001011011001001000110 0111$$

(39 characters, as expected - 1.95 bits on average).

It is not obvious that we can do better here, and in general for randomly chosen messages with these frequencies we simply can't! But what about:

$$aaaaaaaaaabbbbcccdde$$

or:

$$abababababacacacadadae?$$

Again, Huffman coding would yield 39 characters. Clearly, those messages have more than just statistical redundancy; they also have a form of structural redundancy to which statistical methods such as Huffman coding are oblivious.

Source modelling A lot of work was done in the early days of text compression to model natural languages and to understand their redundancy. The first work is Shannon's statistical analysis of English text [15], and it has been significantly refined over the years (see Cover and King [4] for a survey of techniques).

Using a completely different approach, Zipf [19] exhibited a remarkable variety of hyperbolic laws in social sciences; in particular the distribution of words in a natural language approximately satisfies the beautiful law described below. Suppose a natural language has N words, sorted in non-increasing frequency ($p(1) \geq p(2) \geq \ldots p(r) \cdots \geq p(N)$). Then the probability of the word at the r-th rank is

$$p(r) = \frac{\mu}{r},$$

with

$$\mu \approx \frac{1}{\log_e N + \gamma},$$

where $\gamma = 0.577\ldots$ is the Euler-Mascheroni constant. Finer models have been proposed, e.g. by Mandelbrot [11].

We have already looked at modelling English as a sequence of random letters with frequencies. This is called the first-order model of English. Random text from this model (plus space) would look like:

```
ocroh hli rgwr nmielwis eu ll nbnesebya th eei alhenhttpa oobttva nah brl
```

We could do better by regarding English not as 26 letters, but as 26^2 pairs of letters (digrams), E.g. AB QU ZA QZ. If we analyse the frequencies of digrams, we can choose the next letter based upon the previous letter and the digram frequencies. E.g. Q will almost certainly be followed by U, T is most likely to be followed by H. This is called the second-order model of English. Random text from this model (plus space) would look like:

```
on ie antsoutinys are t inctore st be s deamy achin d ilonasive tucoowe at teasonare
fuso tizin andy tobe seace ctisbe
```

Random text from the third-order model of English would look like:

```
in no ist lat whey cratict froure birs grocid pondenome of demonstures of the
reptagin is regoactiona of cre
```

There are finer and finer models of the English language, some based on n-gram frequencies, other (more accurate), based on frequencies of sequences of words. Examples of text generated from 12-gram model (for letters) and 6-gram model (for text) can be found in [1, Chapter 4].

One could then consider using Huffman coding (or any other statistical technique) with finer and finer models. There are two major issues with this approach.

1. The alphabet of the source X explodes! If we consider just the 4-gram model (for letters), then the alphabet is of size $26^4 = 456,976$. In general, the alphabet size grows exponentially with n for n-grams.

2. The model is only appropriate for a particular sort of text. The model for English is inappropriate for German or French, let alone Greek, Russian or Chinese. So that strategy is not easily portable.

3.2 Lempel-Ziv

The main idea of dictionary based compression is to construct a table (dictionary) of commonly used subsequences and refer to this to build the coded message. The main idea behind Lempel-Ziv (LZ77) is to use the message itself as a dictionary.

The LZ77 encoding algorithm works as follows. The encoding scans the message from first to last character. For implementation purposes, it uses a sliding window, of size W and a look-ahead buffer of size L. Consider the message $m_1 \dots m_n$. When encoding at character i, look for the largest l such that the first l characters of the look-ahead buffer match l consecutive characters in the sliding window, i.e.

$$m_i \dots m_{i+l-1} = m_{i-d} \dots m_{i-d+l-1}$$

where $d \leq W$ and $l \leq L$. Append the coded message with (d, l, m_{i+l}). Resume encoding at character $i+l+1$.

The L77 decoding algorithm reads a list of triplets (d, l, m_{i+l}), which it interprets as the instruction:

> Print out $m_{i-d} \dots m_{i-d+l-1} m_{i+l}$ (the l successive characters of m starting from d positions ago, and then m_{i+l}).

Example 3.1. *Encoding the sequence*

$$ABRACADABRA$$

using LZ77 (with say infinite W and L) yields

$$(0,0,A) \quad (0,0,B) \quad (0,0,R) \quad (3,1,C) \quad (2,1,D) \quad (7,4,-)$$

Example 3.2. *A longer example now:*

Peter Piper picked a peck of pickled peppers;
A peck of pickled peppers Peter Piper picked;
If Peter Piper picked a peck of pickled peppers,
Where's the peck of pickled peppers Peter Piper picked?

We obtain:

$$(0,0,P)(0,0,e)(0,0,t)(2,1,r)(0,0,)(6,1,i)(0,0,p)(6,3,p)(6,1,c)(0,0,k)(7,1,d)$$
$$(7,1,a)(9,2,e)(9,2,)(0,0,o)(0,0,f)(17,5,l)(18,3,p)(4,1,p)(32,3,s)(0,0,;)$$
$$(0,0,A)(26,24,)(71,18,;)(0,0,I)(38,2,P)(93,43,,)$$
$$(0,0,W)(0,0,h)(6,2,e)(0,0,')(75,2,t)(8,2,)(103,42,?)$$

193 characters encoded as 34 triples. If each triple is 3 bytes - that is 193 bytes reduced to 102 bytes.

How much space do we need? Each triple in the encoding includes $d \leq W$, $l \leq L$ and a character. For ASCII, the character takes 8 bits. In total, we need

$$\log_2(W+1) + \log_2(L+1) + 8$$

bits to encode a triple.

Typical values are $W = 2^{16} - 1 = 65535$, and $L = 2^8 - 1 = 255$, so we need $16 + 8 + 8$ bits per triple, i.e. 4 bytes. This much can be wasteful, especially if the value of l is very low ($l = 0$ means that this is a new character for instance).

3.3 Applications of LZ77

LZSS Lempel-Ziv-Storer-Szymanski (LZSS) is a popular variant of LZ77 introduced in 1982. The main improvement is that it includes a flag to distinguish between new characters and tokens. That way, a new character does not need to be encoded as a full token, and tokens only have two fields instead of three.

DEFLATE Deflate is a lossless compression technique that combines LZSS and Huffman coding. The key idea is that Lempel-Ziv removes structural redundancy from the data, but its output still has some statistical redundancy; the latter is then removed by Huffman coding.

Deflate is everywhere: in `gzip`, in the ZIP file format, in PNG, etc. (Technically, ZIP allows for many different compression techniques, but Deflate is the one that's used most of the time.)

LZMA The Lempel–Ziv–Markov chain algorithm (LZMA) was developed for 7z. Its description is out of the scope of these lectures.

3.4 See further

Variants There are many variants of LZ77, e.g. LZX, LZRW1, LZRW4. Have a look!

VCDIFF File differencing refers to any method that compresses the differences between two files (say the source and the target files). The term delta compression is also used. VCDIFF is a method for file differencing based on LZ77. The basic idea is very simple:

1. append the target file to the source file to make one massive file
2. use LZ77 to compress that massive file
3. only save the part relating to the target file of the output of LZ77.

The implementation is more involved; see [10]. In general, delta compression is an important problem that is still the subject of ongoing research.

3.5 Exercises

Exercise 3.1. Decode the following string encoded with LZ77.

$$(0,0,r)(0,0,i)(0,0,n)(0,0,g)(0,0,)$$
$$(0,0,a)(2,1,r)(7,4,o)(7,2,o)(0,0,s)(9,1,e)(3,1,)$$
$$(16,2,p)(9,1,c)(0,0,k)(9,1,t)(7,1,f)(0,0,u)(0,0,l)(1,1,)$$
$$(11,1,f)(15,3,s)(24,5,t)(6,1,s)(0,0,h)(11,1,o)(8,9,w)(20,1,)$$
$$(11,1,l)(33,2,f)(5,4,d)(15,1,w)(0,0,n)$$

Exercise 3.2. Watch this youtube video on the repetitiveness of pop music: Pop Music is Stuck on Repeat. Can you guess which is the most repetitive song in the history of the Billboard Hot 100?

Exercise 3.3. Write your own LZ77 encoder (in Python, Java, or any other language). Can you find famous pieces of fiction that compresses massively, or hardly at all?

4 Lempel-Ziv II: LZ78

4.1 LZ78

Basic idea The LZ78 method does not use any search buffer, look-ahead buffer, or sliding window. Instead, it simply keeps a dictionary of previously encountered strings. The dictionary starts with the empty string at position zero and its size is only limited by the memory size.

The encoder outputs two-field tokens (instead of three-field tokens in LZ77). Each token simply corresponds to a new string in the dictionary: it is of the form

$$(i,x),$$

where i is the position of the longest match in the dictionary and x is the final character of the string.

Nothing is ever deleted from the dictionary:

- Advantage over LZ77: future strings will be compressed even if they only match strings in the distant past;
- Drawback: the dictionary can become very large!

Example Once again, it is best explained via a simple example. Say we want to compress

sir␣sid␣eastman␣easily

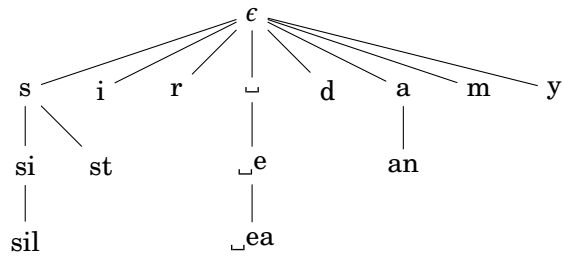
The tokens are then (in order!):

Dictionary position	String	Token
0	ϵ	
1	s	(0, s)
2	i	(0,i)
3	r	(0,r)
4	␣	(0,␣)
5	si	(1,i)
6	d	(0,d)
7	␣e	(4,e)
8	a	(0,a)
9	st	(1,t)
10	m	(0,m)
11	an	(8,n)
12	␣ea	(7,a)
13	sil	(5,l)
14	y	(0,y)

And the compressed output is the list of tokens

(0,s) (0,i) (0,r) (0,␣) (1,i) (0,d) (4,e) (0,a) (1,t) (0,m) (8,n) (7,a) (5,l) (0,y)

Once again, the decoder sees these tokens as "instructions." But following these instructions means searching in the dictionary. A useful data structure for the dictionary is a tree, where the root is the empty string and a new string is added to the tree as a child of the string it refers to on its token. Such a tree is called a trie.



4.2 LZW

Basic idea Lempel-Ziv-Welch (LZW) is a variant of LZ78, with two main differences.

1. The dictionary is initialised with all possible characters. If we are compressing an ASCII file, then positions 0 to 255 are filled at initialisation.

2. The tokens only have one field! Since we always work with at least one character (that can always be found in the dictionary), there is no need to output the next character.

Let us go back to our example:

sir␣sid␣eastman␣easily

The dictionary is initialised with all 256 ASCII characters in positions 0 to 255, e.g. a is in position 97, b in 98, s in 115, z in 122. The first character in the string is s (in the dictionary at position 115). Since si does not appear in the dictionary, we add si to the dictionary at 256, and we continue with the character i. Again, since ir is not in the dictionary, we add ir at 257 and continue with the character r.

The dictionary (omitting positions 0 to 255) and the tokens look like this:

Position	String	Token	What the token encodes
256	si	115	s
257	ir	105	i
258	r␣	114	r
259	␣a	32	␣
260	sid	256	si
261	d␣	100	d
262	␣e	32	␣
263	ea	101	e
264	as	97	a
265	st	115	s
266	tm	116	t
267	ma	109	m
268	an	97	a
269	n␣	110	n
270	␣ea	262	␣e
271	asi	264	as
272	il	105	i
273	ly	108	l
		121	y

The output is then

$$115, 105, 114, 32, 256, 100, 32, 101, 97, 115, 116, 109, 97, 110, 262, 264, 105, 108, 121$$

The dictionary can once again be stored as a tree, but the implementation is more complex than for LZ78. A thorough description is given in [13, 3.13.2].

4.3 Applications of LZW

GIF The ubiquitous Graphics Interchange Format (GIF) uses a variation of LZW. It uses a dynamic, growing dictionary. It starts with the number of bits per pixel b: $b = 2$ for monochromatic images, $b = 8$ for an image with 256 colours of shades of grey. The dictionary starts with 2^{b+1} entries and is doubled in size every time it fills up until it reaches $2^{12} = 4,096$ entries. At that point, the encoder may want to start a new dictionary!

GIF is not actually that good at image compression because it is unidimensional. It scans the image row after row, so it can detect similarities within a row but has trouble dealing with similarities across rows instead.

Limitations One major issue of using LZW (e.g. for GIF), is that LZW is patented. In response to that, the Portable Network Graphics format was created in the mid-90s (finalised in 96). It is based on DEFLATE (and hence LZSS) instead.

Another application of LZW was the Unix shell compression utility `compress`, that was used in the 80s. However, it was superseded by `gzip`, which typically outperforms it in terms of compression ratio.

4.4 See further

Variants LZ78 and LZW also have a few variants, notably LZMW, LZAP and LZY. Have another look!

Kolmogorov complexity The principle of Lempel-Ziv encoding is to construct a list of instructions to the decoder of the form "Copy that string (and add that character)." But what if we allowed any sort of instructions?

The Kolmogorov complexity is a concept that predates Lempel-Ziv. It aims at evaluating the "intrinsic" complexity of a binary string. Simply put, the Kolmogorov complexity of a string x w.r.t. a Turing machine U, denoted $K_U(x)$, is the shortest length of a program for U that prints out x and halts. Obviously, $K_U(x)$ is not computable. But still, we can say a lot about the Kolmogorov complexity of a random string: it's about the length of the string. Therefore, almost any string is incompressible! The study of Kolmogorov complexity and associated concepts (e.g. Solomonoff's universal probability or Chaitin's Omega number) is very intriguing but outside the scope of this course.

4.5 Exercises

Exercise 4.1. Encode the string

sir␣sid␣eastman␣easily␣teases␣sea␣sick␣seals

with LZ78: give the dictionary table, the trie, and the output.

Encode the same string with LZW.

Exercise 4.2. Select a few (small) images, and compare the file sizes for those when saved as .bmp, .gif and .png.

5 Context-based compression

5.1 Context

Context-based compression Statistical compression (mainly for text, but not only) can be based on two properties. The first property is the frequency of symbols: the model assigns probabilities to the symbol according to their frequency in the document.

The second one is the context. In practice, the context a symbol consists of the N symbols preceding it (note that we cannot use symbols succeeding it, as the decoder typically does not know them yet!). Context-based compression then uses the context of a symbol to predict it (i.e. to assign it a probability).

For instance, let's use a context of only one character. The letter h occurs in typical English text only about 5% of the time. However, if the current symbol is t, then there is a much higher probability (around 30%) that the next symbol will be h, since the digram th is very common in English. Note that the prediction is about assigning probabilities, not trying to figure out the next symbol exactly (which is impossible).

Static v Adaptive contexts A static context-based modeler always uses the same probabilities, which are stored in some large table. Those probabilities are usually obtained by crawling through many documents (say typical English texts). There are issues with that approach, notably the fact that this might assign zero probabilities to some strings.

An adaptive context-based modeler also maintains tables of probabilities of all the possible digrams (or trigrams, or even longer n-grams). But this time the tables are updated all the time as more data are input, which adapts the probabilities to the particular data being compressed. Adaptive context-based compression might be slower, but typically results in better compression.

Context length One may think at first that the larger the number N of symbols in the context, the better the compression. However, this might not be the case:

1. A large N requires to write the first N symbols in plain text, which might hurt the overall compression.

2. If N is too large, then there are simply too many contexts, which makes storing, reading off, and writing on the table of probabilities infeasible.

3. A very long context contains information about the nature of old data. It is not uncommon to have files where different parts have different symbol distributions.

Therefore, in practice relatively small contexts are used in practice (for text compression, traditional methods use no more than 10 characters).

5.2 PPM

Basic idea Prediction by Partial Matching (PPM) is based on an encoder that keeps a statistical model of the text. It starts with an order-N context. It searches its data structure for an occurrence of the current context C followed by the next symbol S. If it finds no such occurrence, if decreases the order of the context to $N-1$ and tries again (the new context C' is the final $N-1$ characters of C). It keeps shortening the context until it is successful.

The encoder reads the next symbol S from the input stream, looks at the current order-N context C (the last N symbols read), and based on the previous input data, computes the probability P that S will appear following C. The encoder then calls an adaptive arithmetic encoder to encode S with probability P. If the probability P is zero, the PPM encoder tries with a smaller context; it reduces the context until $P \neq 0$. What if the symbol S has not been seen yet (and hence, even with order-0 context, we still have $P = 0$)? Then the PPM encoder enters order-(-1) context, where the probability of S is simply 1/(size of alphabet).

Example 5.1. *Let us look at the contexts and frequency counts for the following string with* 11 *symbols:*

xyzzxyxyzzx

Order 4		*Order 3*		*Order 2*		*Order 1*		*Order 0*	
$xyzz \to x$	2	$xyz \to z$	2	$xy \to z$	2	$x \to y$	3	x	4
$yzzx \to y$	1	$yzz \to x$	2	$xy \to x$	1	$y \to z$	2	y	3
$zzxy \to x$	1	$zzx \to y$	1	$yz \to z$	2	$y \to x$	1	z	4
$zxyx \to y$	1	$zxy \to x$	1	$zz \to x$	2	$z \to z$	2		
$xyxy \to z$	1	$xyx \to y$	1	$zx \to y$	1	$z \to x$	2		
$yxyz \to z$	1	$yxy \to z$	1	$yx \to y$	1				

Now, how does the encoder tell the decoder which order context it is currently using (and hence what the decoder should be using too)? The answer is to have a dedicated escape symbol, which we'll denote esc, which should be output whenever the context size is decreased. Since this is a new character, we should also assign a probability for the escape symbol for every encountered context. There are various ways (heuristics) of assigning such probabilities. Here, we will use the so-called Method A, where the escape symbol is assigned a frequency of 1.

We are now in position to give a more explicit example. Encoding a full sequence is actually quite tedious to explain so we'll only encode a few characters. We use contexts of order at most 2. Let us consider

this␣is␣the␣tithe

The first few symbols are not very interesting, so let us skip forward. Let's assume we have already encoded "this␣is" and we wish to encode the next character ␣.

We assume the word length for arithmetic coding is six bits (we used four decimal digits in our example in Lecture 2). For the sake of simplicity, we have $Low = 0$ and hence $L^* = 000000$ and $High = 1$ hence $H^* = 111111$. (As we shall see, the low and high values may vary over time.)

Here is what the table of contexts looks like

Order 2		Order 1		Order 0	
th → i	1	t → h	1	t	1
th → esc	1	t → esc	1	h	1
hi → s	1	h → i	1	i	2
hi → esc	1	h → esc	1	s	2
is → ␣	1	i → s	2	␣	1
is → esc	1	i → esc	1	esc	1
s␣ → i	1	␣ → i	1		
s␣ → esc	1	␣ → esc	1		
␣i → s	1	s → ␣	1		
␣i → esc	1	s → esc	1		

The second-order context is "is". We use characters in the order of the table: the first row gives the first interval and so on. In this context, the probability of the space sign "␣" and the probability of the escape symbol esc are both equal to 1/2, and

$$L(␣) = 0, H(␣) = 1/2 = L(esc), H(esc) = 1.$$

The update equations for the new Low and $High$ are

$$\begin{aligned} Low &\leftarrow Low + (High - Low)L(x) = 0, \\ L^* &\leftarrow 000000, \\ High &\leftarrow Low + (High - Low)H(x) = 1/2, \\ H^* &\leftarrow 011111. \end{aligned}$$

Since the first (most significant) bit of L^* and H^* coincide, we shift that bit out and shift 0 into L^* and shift 1 into H^*. So we obtain:

1. Encoded sequence for "␣": 0,

2. Lower bound $L^* = 000000$,
3. Higher bound $H^* = 111111$.

The table of contexts now becomes:

Order 2		Order 1		Order 0	
th → i	1	t → h	1	t	1
th → esc	1	t → esc	1	h	1
hi → s	1	h → i	1	i	2
hi → esc	1	h → esc	1	s	2
is → ␣	2	i → s	2	␣	2
is → esc	1	i → esc	1	esc	1
s␣ → i	1	␣ → i	1		
s␣ → esc	1	␣ → esc	1		
␣i → s	1	s → ␣	2		
␣i → esc	1	s → esc	1		

The next symbol is "t". The second-order context is "s␣". Since "t" has zero frequency in this context, we need to encode the escape symbol. By a similar argument as above, we obtain

1. Encoded escape symbol sequence: 1,
2. Lower bound $L^* = 000000$,
3. Higher bound $H^* = 111111$.

We need to look at the first-order context, which is "␣". Again, "t" does not appear with this context, so we encode another escape symbol. We obtain

1. Encoded escape symbol sequence: 1,
2. Lower bound $L^* = 000000$,
3. Higher bound $H^* = 111111$.

We need to look at the zero-th order context. This time, "t" has already appeared, and is assigned the interval $[0, 1/9)$. We then have

$$\begin{aligned} Low &\leftarrow Low + (High - Low)L(x) = 0, \\ L^* &\leftarrow 000000, \\ High &\leftarrow Low + (High - Low)H(x) = 1/9, \\ H^* &\leftarrow 000111. \end{aligned}$$

Since the three leftmost bits are equal, we shift them out. We finally obtain

1. Encoded sequence: for "t": 000,
2. Lower bound $L^* = 000000$,
3. Higher bound $H^* = 111111$.

So, to encode "␣t", we have transmitted 011000.

Note that there would be a slight difference in practice: to keep everything integral, we would use $High = 63$, $Low = 0$ and perform an update of the form

$$\begin{aligned} Low &\leftarrow Low + \left\lfloor (High - Low + 1)\frac{1}{9} \right\rfloor = 0 = 000000 \\ High &\leftarrow High + \left\lfloor (High - Low + 1)\frac{1}{9} \right\rfloor - 1 = 6 = 000110 \end{aligned}$$

Then we would have: Higher bound $H^* = 110111$.

Methods B and C Two other main ways of assigning frequencies to the escape symbols aim to make the escape symbol more probable, which typically reduces the size of the resulting sequence for that symbol. The main idea is that if a context is followed by many different characters, then you are likely to encounter yet another character following that same context. For instance, think of the context "s" in English, which can be followed by virtually any other letter. Methods B and C give the escape symbol a count equal to the number of symbols following the context; Method B then subtracts the count of every other symbol by one, while Method C does not amend those.

5.3 See further

RAR The main application of PPM is in the Roshal Archive (RAR) file format.

Context mixing In context mixing, the next-symbol predictions of two or more statistical models are combined to yield a prediction that is often more accurate than any of the individual predictions. The PAQ series of data compression programs use context mixing; they are the cutting edge in lossless compression in terms of compression ratio (at the expense of speed and memory usage) [9].

Note that the problem of mixing different contexts is a very challenging issue in machine learning; that could be a very interesting topic for a project...

BWT The Burrows-Wheeler transform (BWT) is a very clever way of converting a list of symbols into one that is much more structured. You only need a little more information to make sure that the transform does not lose any information. By structured, we mean that it is "almost sorted." After the transform, one can use very simple techniques to efficiently encode the structured list. Unfortunately, BWT-based compression requires to scan and to manipulate the whole message, which is an important drawback compared to the adaptive PPM.

5.4 Exercises

Exercise 5.1. Update the table in Example 5.1 if the following character is x.

Exercise 5.2. Finish the encoding of the sequence "this␣is␣the␣tithe". To update $High$ and Low, you may use our simple technique based on rational numbers, or use the version with integers instead.

6 Transform coding I: Mathematical background

6.1 Fourier series and Fourier transform

Fourier series The representation of periodic functions in terms of a series of sines and cosines was first used by Fourier in 1812. This idea has spread like wildfire across mathematics and its applications and is used in a ridiculous range of areas under different guises.

In this section, we want to give an introduction to Fourier series. We shall be rather informal: we will not prove many of the claimed results, and we will not worry about the precise assumptions we make.

Let f be a periodic function of period T, i.e.

$$f(t) = f(t+nT) \quad \forall t \in \mathbb{R}, n \in \mathbb{Z}.$$

Then we can write $f(t)$ as

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{\mathrm{i}k\omega t},$$

where $\mathrm{i} = \sqrt{-1}$ and

$$\omega = \frac{2\pi}{T}.$$

Equivalently, we have

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\omega t) + \sum_{k=1}^{\infty} b_k \sin(k\omega t).$$

In other words, the periodic functions of period T form a vector space, and $\{e^{in\omega t}\}$ forms a basis. In fact, this basis is orthonormal w.r.t. the inner product

$$\langle f(t), g(t)\rangle = \frac{1}{T}\int_0^T f(t)g^*(t)\,dt,$$

where $g^*(t)$ denotes the complex conjugate of g. The coefficients can then be obtained as follows:

$$c_k = \langle f(t), e^{ik\omega t}\rangle = \frac{1}{T}\int_0^T f(t)e^{-ik\omega t}dt. \tag{6.1}$$

The intuition behind this representation is as follows. Suppose we have a periodic signal f. Then its c_k Fourier coefficients decompose the signal into basic fluctuating signals; each $e^{ik\omega t}$ fluctuates at a frequency of $k\omega/(2\pi)$. As such, these coefficients give us a measure of the different amounts of fluctuation in the signal.

Discrete Fourier transform We want to handle signals that are not periodic, but instead limited in time. Suppose we have a signal $f(t)$ that is limited in time (say from $t = 0$ to $t = t_1$). Then we can extend it to a periodic signal by doing

$$f_P(t) = \sum_{n\in\mathbb{Z}} f(t - nT),$$

where $T > t_1$. This is the so-called periodic extension of f.

Moreover, we are dealing with discrete signals: instead of $f(t)$, we are considering $\{f_0, f_1, \ldots, f_{N-1}\}$. We can discretise Equation (6.1) as follows:

$$F_k = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} f_n e^{-i\frac{2\pi kn}{N}} \quad k = 0, 1, \ldots, N-1.$$

The coefficients F_k are called the discrete Fourier transform (DFT) of f. We can recover the signal from its DFT by

$$f_n = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} F_k e^{i\frac{2\pi kn}{N}} \quad n = 0, 1, \ldots, N-1.$$

Matrix representation of DFT Let $f = (f_0, f_1, \ldots, f_{N-1})$ and let $F = (F_0, \ldots, F_{n-1})$ be its DFT. Then we have

$$F = \mathbf{A}f,$$

where $\mathbf{A}$ is an $N \times N$ matrix with coefficients

$$a_{i,j} = \frac{1}{\sqrt{N}} e^{-i\frac{2\pi}{N}ij}.$$

For example, with $N = 4$ we obtain

$$\mathbf{A} = \frac{1}{\sqrt{4}}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

Unitary and orthogonal transforms The matrix $\mathbf{A}$ from the DFT is unitary: it satisfies

$$\mathbf{A}^\dagger = \mathbf{A}^{-1},$$

where $\mathbf{A}^\dagger$ is its conjugate transpose: $a^\dagger_{i,j} = a^*_{j,i}$. It is easily shown that the following are equivalent for a matrix $\mathbf{U}$:

1. $\mathbf{U}$ is unitary;

2. $\mathbf{U}$ preserves the inner product, i.e.
$$\langle x, y\rangle = \langle \mathbf{U}x, \mathbf{U}y\rangle;$$

3. $\mathbf{U}$ preserves the norm, i.e.
$$\|x\|^2 = \sum_{i=0}^{N-1} |x_i|^2 = \langle x, x\rangle = \langle \mathbf{U}x, \mathbf{U}x\rangle = \sum_{i=0}^{N-1} |(\mathbf{U}x)_i|^2 = \|\mathbf{U}x\|^2.$$

We will not actually use the DFT and instead we will restrict ourselves to real matrices. A real unitary matrix is called orthogonal. So an orthogonal matrix is a real matrix $\mathbf{A}$ such that
$$\mathbf{A}^{-1} = \mathbf{A}^\top.$$

6.2 Two-dimensional transforms

Two-dimensional data In general, in one dimension, we could apply any orthogonal transform as such:
$$\theta = \mathbf{A}x, \quad x = \mathbf{A}^\top\theta.$$
We will use transforms for two-dimensional data (small blocks of pixels). How are we going to apply a one-dimensional transform to two-dimensional data? The answer is actually easy: we apply the transform column-wise and row-wise.

Matrix form More succinctly, let $\mathbf{X}$ be an $N \times N$ matrix with entries $\{x_{i,j} : i, j = 0, \dots, N-1\}$. We then perform
$$\Theta = \mathbf{A}\mathbf{X}\mathbf{A}^\top.$$
(Θ is another $N \times N$ matrix.) Multiplying on the left by $\mathbf{A}$ performs the transform column-wise, while multiplying on the right by $\mathbf{A}^\top$ performs the transform row-wise. By associativity,
$$\Theta = (\mathbf{A}\mathbf{X})\mathbf{A}^\top = \mathbf{A}(\mathbf{X}\mathbf{A}^\top),$$
and hence the order does not matter!

Note that the inverse is straightforward:
$$\mathbf{X} = \mathbf{A}^\top\Theta\mathbf{A}.$$

Basis matrices Let $\mathbf{X}$ be an $N \times N$ matrix. For all $i, j \in \{0, \dots, N-1\}$, let $\mathbf{E}_{i,j}$ be the matrix with a single 1 in position (i, j) and 0 everywhere else. Then those matrices form a basis for the vector space of all $N \times N$ matrices, and we have the decomposition
$$\mathbf{X} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x_{i,j}\mathbf{E}_{i,j}.$$
(Trivial, isn't it?)

Any two-dimensional orthogonal transform also yields a similar decomposition. We still denote $\Theta = \mathbf{A}\mathbf{X}\mathbf{A}^\top$, and we denote the entries of Θ as $\theta_{i,j}$. We then have
$$\begin{aligned}\mathbf{X} &= \mathbf{A}^\top\Theta\mathbf{A} \\ &= \mathbf{A}^\top\left(\sum_{i,j}\theta_{i,j}\mathbf{E}_{i,j}\right)\mathbf{A} \\ &= \sum_{i,j}\theta_{i,j}\mathbf{A}_{i,j},\end{aligned}$$
where the matrices $\mathbf{A}_{i,j} := \mathbf{A}^\top\mathbf{E}_{i,j}\mathbf{A}$ are the basis matrices of the transform. More concretely, denote the k-th row of $\mathbf{A}$ as a_k, then
$$\mathbf{A}_{i,j} = a_i^\top a_j.$$

For instance, let

$$\mathbf{A} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Then the four basis matrices are given by

$$\mathbf{A}_{0,0} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad \mathbf{A}_{0,1} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$$
$$\mathbf{A}_{1,0} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} \qquad \mathbf{A}_{1,1} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

6.3 Modus operandi of transform coding

Norm and energy The square of the norm of a vector can be viewed as its energy. Orthogonal transforms preserve the energy (as they are unitary matrices). The efficacy of a transform depends on how much energy compaction is provided by the transform. Intuitively, the transform helps to accumulate the energy on a few symbols; those symbols should be kept, while the other ones can be discarded at little loss.

One measure of the energy compaction offered by a transform is the transform coding gain, defined as follows (for simplicity, we define it for one-dimensional data). Let σ_i^2 be the variance of the transformed coefficient θ_i for $i = 0, \ldots, N-1$, then the transform coding gain is the ration of the arithmetic mean of variances over their geometric mean:

$$G_{\mathrm{TC}} := \frac{\frac{1}{N}\sum_{i=0}^{N-1}\sigma_i^2}{\left(\sum_{i=0}^{N-1}\sigma_i^2\right)^{\frac{1}{N}}}.$$

(The derivation of this gain is out of the scope of this course.)

Three steps of transform coding Transform coding consists of three steps.

Step 1: Transform First, the data is split into blocks of size N. Each block is mapped into a transform sequence using a reversible transform (usually orthogonal).

Step 2: Quantization Secondly, the transformed sequence is quantized. The quantization strategy depends on three main factors:

1. the desired average bit rate
2. the statistics of the various elements of the transformed sequence
3. the effect of distortion in the transformed coefficients on the reconstructed sequence.

Quantization is an important problem with some nice maths behind it, but we will skip the details here.

Step 3: Encoding Thirdly, the quantized value is encoded using some binary encoding technique, e.g. Huffman or arithmetic coding.

6.4 See further

FFT The Fast Fourier Transform (FFT) is, as its name suggests, a very efficient way of computing the DFT via divide and conquer. It was due to Cooley and Tuckey in 1965... even though Gauss had already discovered it in 1805!

Karhunen-Loève transform The KL transform is the one that maximises the transform coding gain. However, it is based on the actual data and is impractical for our purposes.

6.5 Exercises

Exercise 6.1. Walsh-Hadamard transform. For all N a power of 2, the discrete Walsh-Hadamard transform (DWHT) is defined as follows. For $N = 1$, let $\mathbf{H}_1 = (1)$. For $N \geq 1$, let

$$\mathbf{H}_{2N} = C \begin{pmatrix} \mathbf{H}_N & \mathbf{H}_N \\ \mathbf{H}_N & -\mathbf{H}_N \end{pmatrix}.$$

(This transform comes under different names; to make matters worse, those matrices are sometimes called Sylvester matrices.)

1. Verify that the Discrete Walsh-Hadamard Transform is indeed orthogonal for the right choice of constant C.
2. Give $\mathbf{H}_2$, $\mathbf{H}_4$ and $\mathbf{H}_8$.
3. Give all sixteen basis matrices for $N = 4$.
4. Find a closed form formula for the (i, j) entry of $\mathbf{H}_N$.

Exercise 6.2. Give the sixteen basis matrices for the DFT with $N = 4$.

Exercise 6.3. Consider the data

$$\mathbf{X} = \begin{pmatrix} 4 & 3 & 2 & 1 \\ 3 & 2 & 1 & 1 \\ 2 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}.$$

1. Compute its two-dimensional DFT.
2. Compute its two-dimensional DWHT.

7 Transform coding II: JPEG

7.1 Operations before transform coding

Color space transformation First, the image should be converted from RGB into a different color space called $Y'C_BC_R$ (a.k.a. YCbCr). It has three components Y', C_B and C_R: the Y' component represents the brightness of a pixel, and the C_B and C_R components represent the chrominance (split into blue and red components).

Mathematically, let R, B, G be the red, green and blue signals, which all take a value between 0 and 1. Then

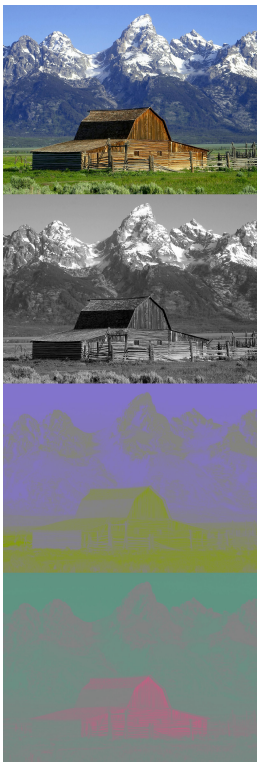
$$\begin{aligned} Y' &= K_R R + K_G G + K_B B, \\ C_B &= \frac{1}{2} \frac{B - Y'}{1 - K_B}, \\ C_R &= \frac{1}{2} \frac{R - Y'}{1 - K_R}, \end{aligned}$$

where K_R, K_G, and K_B are constants that satisfy $K_R + K_G + K_B = 1$. For instance,

$$\begin{aligned} K_R &= 0.299, \\ K_G &= 0.587, \\ K_B &= 0.114. \end{aligned}$$

The YCbCr colour space conversion allows greater compression without a significant effect on perceptual image quality (or greater perceptual image quality for the same compression). The compression

is more efficient because the brightness information, which is more important to the eventual perceptual quality of the image, is confined to a single channel. This more closely corresponds to the perception of color in the human visual system.



Downsampling Humans can see considerably more fine detail in the brightness of an image (the Y' component) than in the hue and color saturation of an image (the Cb and Cr components). As such, the next step is chroma downsampling, which reduces the spatial resolution of the Cb and Cr components.

Block splitting JPEG splits an image into 8×8 blocks of pixels and applies transform coding to each block. We will focus on one channel and see the data as an 8×8 matrix of integers $\mathbf{X}$. We will use a running example with

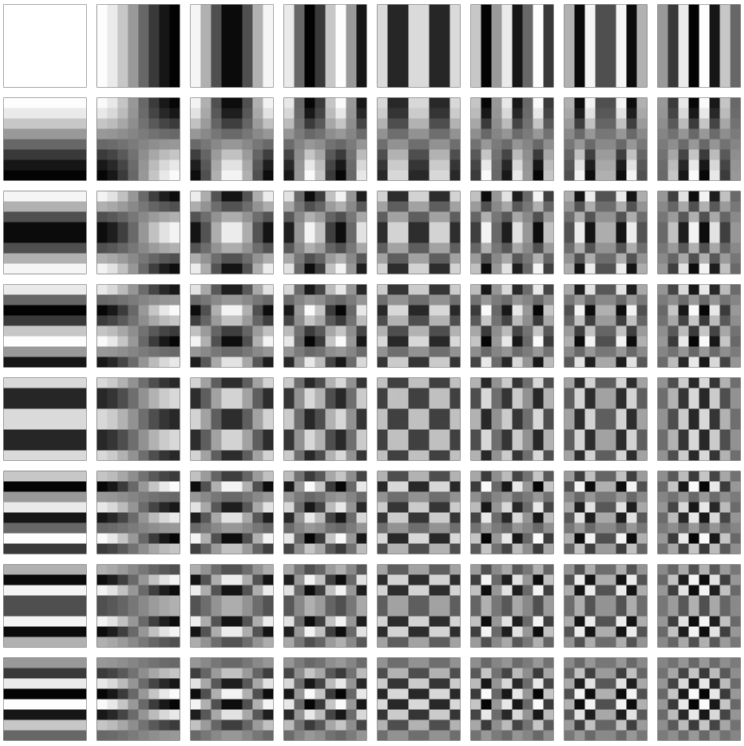
$$\mathbf{X} = \begin{pmatrix} 124 & 125 & 122 & 120 & 122 & 119 & 117 & 118 \\ 121 & 121 & 120 & 119 & 119 & 120 & 120 & 118 \\ 126 & 124 & 123 & 122 & 121 & 121 & 120 & 120 \\ 124 & 124 & 125 & 125 & 126 & 125 & 124 & 124 \\ 127 & 127 & 128 & 129 & 130 & 128 & 127 & 125 \\ 143 & 142 & 143 & 142 & 140 & 139 & 139 & 139 \\ 150 & 148 & 152 & 152 & 152 & 152 & 150 & 151 \\ 156 & 159 & 158 & 155 & 158 & 158 & 157 & 156 \end{pmatrix}.$$

7.2 Step 1: Transform

DCT The discrete cosine transform (DCT) is similar to the DFT, but only takes the cosines into consideration. More precisely, the transform matrix is $\mathbf{C}$, where

$$c_{i,j} = C_i \sqrt{\frac{1}{N}} \cos\left(\frac{(2j+1)i\pi}{2N}\right),$$

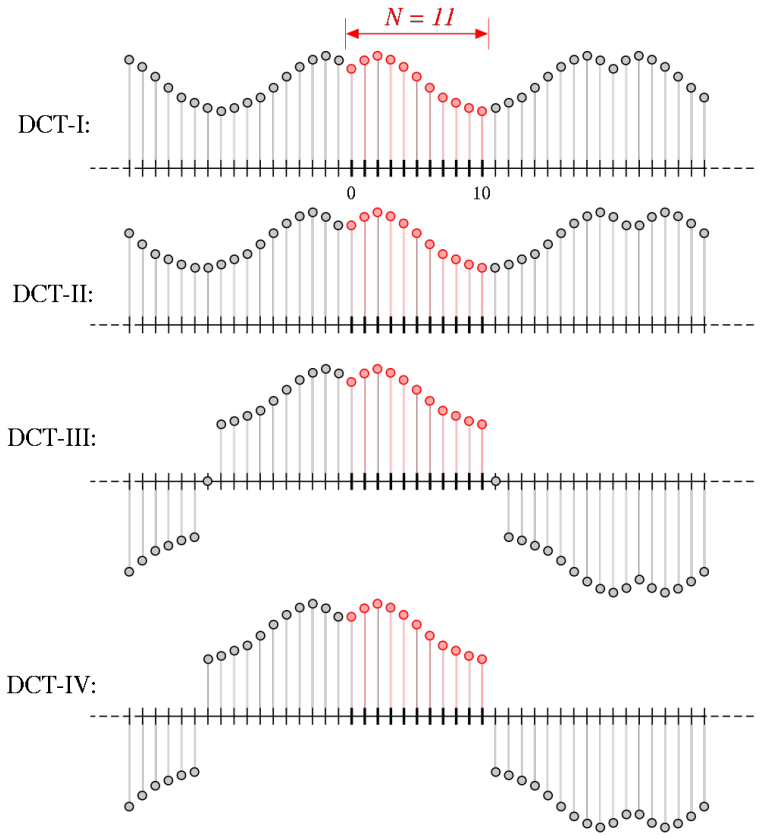
where $C_0 = 1$ and $C_i = 2$ for all $i \in \{1, \ldots, N-1\}$. The basis matrices for the DCT ($N = 8$) are given below.



Technically, JPEG does not apply the DCT to $\mathbf{X}$ directly. Instead, it subtracts 128 to each value (so that for instance, the top left value is now -4) before applying the DCT. Following our running example, we obtain

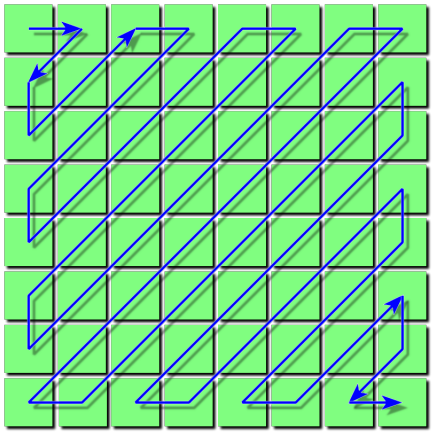
$$\Theta = \begin{pmatrix} 39.88 & 6.56 & -2.24 & 1.22 & -0.37 & -1.08 & 0.79 & 1.13 \\ -102.43 & 4.56 & 2.26 & 1.12 & 0.35 & -0.63 & -1.05 & -0.48 \\ 37.77 & 1.31 & 1.77 & 0.25 & -1.50 & -2.21 & -0.10 & 0.23 \\ -5.67 & 2.24 & -1.32 & -0.81 & 1.41 & 0.22 & -0.13 & 0.17 \\ -3.37 & -0.74 & -1.75 & 0.77 & -0.62 & -2.65 & -1.30 & 0.76 \\ 5.98 & -0.13 & -0.45 & -0.77 & 1.99 & -0.26 & 1.46 & 0.00 \\ 3.97 & 5.52 & 2.39 & -0.55 & -0.051 & -0.84 & -0.52 & -0.13 \\ -3.43 & 0.51 & -1.07 & 0.87 & 0.96 & 0.09 & 0.33 & 0.01 \end{pmatrix}.$$

DCT over DFT Why use the DCT instead of the DFT? Let's use the one-dimensional case to explain this. The DFT has a major problem: it "presumes" that the signal $(f_0, \dots, f_{N-1})$ has period N, since it uses a basis a functions that are periodic of period N. But there could be a large discontinuity in our signal: f_0 may differ from f_{N-1}. The DFT needs accounts for that discontinuity by adding a term of high frequency and in turn modifying all the lower frequency terms: that can ruin everything. On the other hand, the DCT "presumes" that the signal has period $2N$ by effectively working on a new signal $(f_0, \dots, f_{N-1}, f_{N-1}, f_{N-2}, \dots, f_0)$. That signal does not have such discontinuity anymore. See below for an example; note that there are four main kinds of DCT and that JPEG uses DCT-II.



7.3 Step 2: Quantization

Zigzag scan The basis matrices of the DCT represent different fluctuations of frequency increasing with both i and j. In particular, the $\theta_{0,0}$ coefficient is referred to as the DC coefficient as it corresponds to zero frequency and is proportional to the average value of $x_{i,j}$. The other coefficients are referred to as the AC coefficients. The coefficients are sorted according to a zigzag scan, displayed below.



Quantization The JPEG algorithm uses so-called "uniform midtread quantization." The quantization steps are organized in a quantization table; an example is given in the following matrix.

$$\mathbf{Q} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

These values are actually determined by a quality coefficient specified by the user.

The quantized value of $\theta_{i,j}$ is

$$l_{i.j} = \left\lfloor \frac{\theta_{i,j}}{Q_{i,j}} \right\rceil,$$

where $\lfloor x \rceil$ denotes the integer nearest to x: $\lfloor x \rceil = \lfloor x + 0.5 \rfloor$.

We can see that the step size increases as we move from the DC coefficient to higher frequency coefficients. Therefore, more quantization error will be introduced at higher-frequency levels. This is because quantization errors in the DC and lower AC coefficients are more easily detectable by the human visual system than quantization errors in the higher AC frequencies.

Following our example, we obtain the following quantized coefficients

$$\mathbf{L} = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

All coefficients with magnitude less than half the corresponding step size will be set to zero. Because the step sizes at the end of the zigzag scan are larger, we typically see a long run of zeroes toward the end of the scan. The entire run of zeroes at the end of the scan can be code by an end of block (EOB) code after the last nonzero label.

Reconstruction at the decoder's end is straightforward: the reconstructed value $\hat{\theta}_{i,j}$ is given by

$$\hat{\theta}_{i,j} = l_{i,j} Q_{i,j}.$$

In our example, we obtain

$$\hat{\Theta} = \begin{pmatrix} 32 & 11 & 0 & 0 & 0 & 0 & 0 & 0 \\ -108 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 42 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The reconstructed data is then

$$\hat{\mathbf{X}} = \begin{pmatrix} 123 & 122 & 122 & 121 & 120 & 120 & 119 & 119 \\ 121 & 121 & 121 & 120 & 119 & 118 & 118 & 118 \\ 121 & 121 & 120 & 119 & 119 & 118 & 117 & 117 \\ 124 & 124 & 123 & 122 & 122 & 121 & 120 & 120 \\ 130 & 130 & 129 & 129 & 128 & 128 & 128 & 127 \\ 141 & 141 & 140 & 140 & 139 & 138 & 138 & 137 \\ 152 & 152 & 151 & 151 & 150 & 149 & 149 & 148 \\ 159 & 159 & 158 & 157 & 157 & 156 & 155 & 155 \end{pmatrix}.$$

7.4 Step 3: Encoding

There are two distinct forms of encoding, one for the DC coefficient and one for the AC coefficients.

Encoding the DC coefficient The DC coefficient is directly related to the average value of the data in the block. As such, it typically does not change massively from one block to the next. We then use differential encoding, where we encode the difference between the current block and the previous one.

Values of the change are grouped in Categories. Intuitively, the n-th category contains all changes that need n bits to be written. Formally,

1. for $n = 0$, the category C_0 is $C_0 = \{0\}$;
2. then for all $1 \leq n \leq 15$, the category C_n is defined as

$$C_n = \{-(2^n - 1), \ldots, -2^{n-1}\} \cup \{2^{n-1}, \ldots, 2^n - 1\};$$

   (hence $C_1 = \{-1, 1\}$, $C_2 = \{-3, -2, 2, 3\}$, and so on)
3. finally, for $n = 16$, $C_{16} = 2^{15}$.

The category numbers are encoded using Huffman coding; we then use n bits to specify a particular value in category n. For instance, if the difference is 6, then we would send the Huffman codeword for category 3 and then 3 bits to specify that the value is 6.

Encoding the AC coefficients The AC coefficients also use the grouping into categories but they are encoded according to a different Huffman code. The key idea is that we are likely to encounter long runs of zeros in the zigzag scan (even before the EOB), so we want to encode a whole run of zeros at a time. Therefore, we not only encode the category number C, but also the number Z of zero-valued labels since the last nonzero label. The pair C/Z forms a pointer to a predetermined Huffman code.

For instance, suppose the value is -2 (Category $C = 2$) and there have been $Z = 5$ zero-valued labels prior to this label in the zigzag scan. Then we would send the Huffman codeword for 2/5 and then send 2 more bits to identify the value 2.

7.5 See further

JFIF JPEG is the algorithm to compress images; the JPEG File Interchange Format (JFIF) is the most common file format for storing JPEG encoded images. It contains all the necessary supplementary information, such as the image size.

Run Length Encoding The idea of encoding lengths of runs (used in the encoding of AC coefficients) is actually a very basic form of compression, called Run Length Encoding (RLE). It is the base of the Bitmap image format.

QM coder The QM coder is a binary adaptive arithmetic encoder. It can be used as an alternative to Huffman coding. Using QM may yield a higher compression rate but at a cost of higher complexity; as such it is seldom used. The QM coder has variants, notably the MQ coder and the M coder.

7.6 Exercises

Exercise 7.1. Write a program simulating an 8×8 transform coder and decoder without quantization. This should work for the DCT, DFT and DWHT. You may pre-compute and store the transform matrices.

Exercise 7.2. Using the program for Exercise 7.1, perform a very crude quantization: only keep the first M coefficients in the zigzag scan, where M is a parameter that can be set from 1 to N. Compare the quality of the reconstruction for DFT, DCT, and DWHT. Try to use continuous tone image blocks or random image blocks.

8 Wavelet coding I: Mathematical background

8.1 The Continuous Wavelet Transform

Mathematical definitions The continuous wavelet transform (CWT) of a function $f(t)$ involves a mother wavelet $\psi(t)$. The mother wavelet is scaled by a factor a and translated by b as such:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t-b}{a}\right).$$

We view the $\psi_{a,b}$ functions as a basis, and we naturally compute the inner product

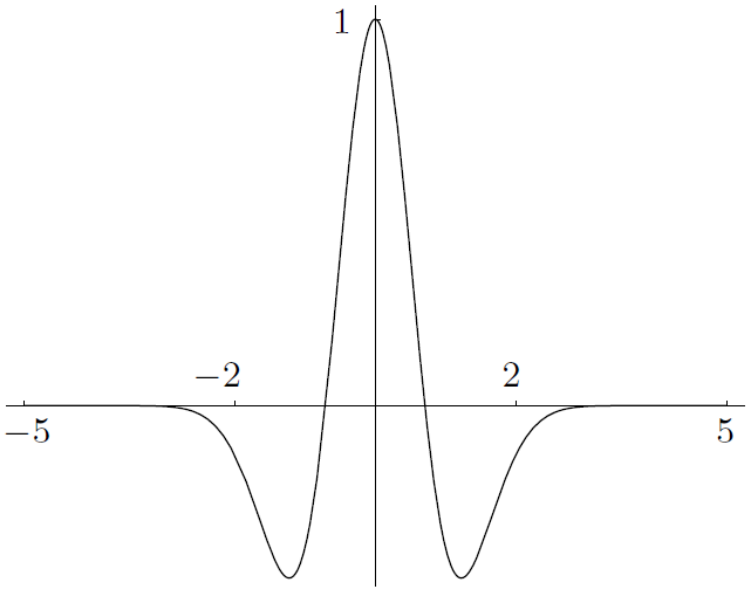
$$W(a,b) := \langle f(t), \psi_{a,b}(t)\rangle = \int_{-\infty}^{\infty} f(t)\psi^*_{a,b}(t)\,dt.$$

The mother wavelet needs to satisfy three properties.

1. It has zero average:
$$\int_{-\infty}^{\infty} \psi(t)\,dt = 0.$$
2. It has finite energy:
$$\int_{-\infty}^{\infty} |\psi(t)|^2\,dt < \infty.$$
3. The admissibility condition. Let
$$\Psi(\omega) := \int_{-\infty}^{\infty} \psi(t)\mathrm{e}^{-\mathrm{i}\omega t}\,dt,$$
$$C := \int_{-\infty}^{\infty} \frac{|\Psi(\omega)^2|}{|\omega|}\,d\omega.$$
Then the admissibility condition requires that $0 < C < \infty$. This technical condition ensures that the inverse CWT exists. You needn't remember the details.

Intuition The CWT provides a time-frequency representation of a signal. Let us consider the signal $f(t) = \sin(t)$ as a basic example.

Firstly, note that due to the finite energy condition, the mother wavelet amplitude decreases rapidly when t tends to plus or minus infinity. A good example is the Mexican hat wavelet:

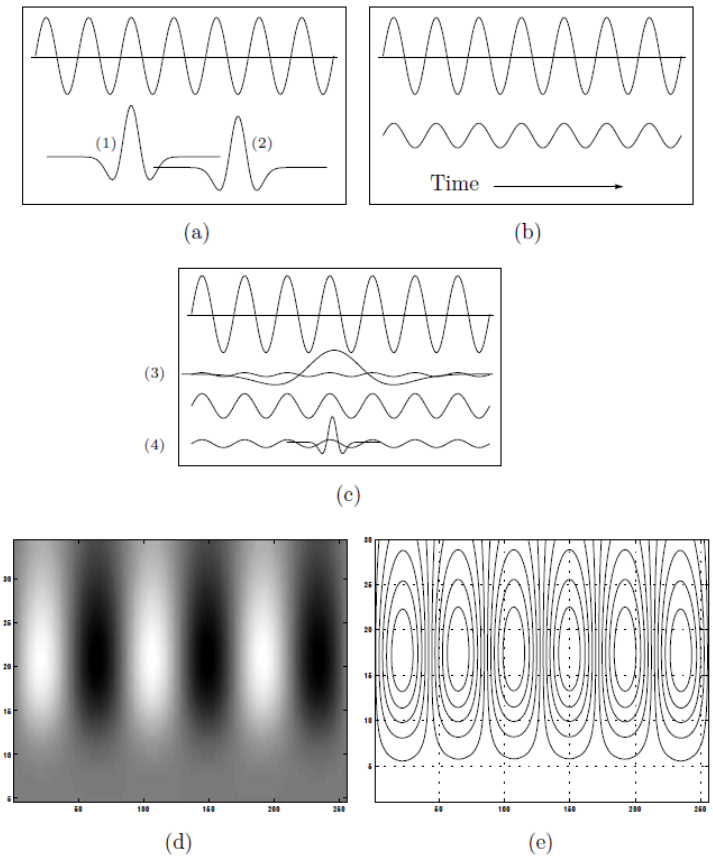


The Mexican hat wavelet is similar to having one oscillation.

For a fixed a, the set $\{\psi_{a,b} : b \in \mathbb{R}\}$ is a sequence of the same function translated over time. If the Mexican hat is placed in phase with the sinusoid, then the inner product will be a large positive number; if it is in opposite phase it will be a large negative number. So it will create oscillations.

On the other hand, for a fixed b, the set $\{\psi_{a,b} : a \in \mathbb{R}\}$ is a sequence of the same function stretched and squeezed. For the Mexican hat, it means changing the frequency of its one oscillation. If the frequency matches that of the sinusoid, it will oscillate a lot. If the frequencies do not match, the oscillations will have a much lower amplitude.

This yields the following time-frequency diagram.



8.2 The Haar transform

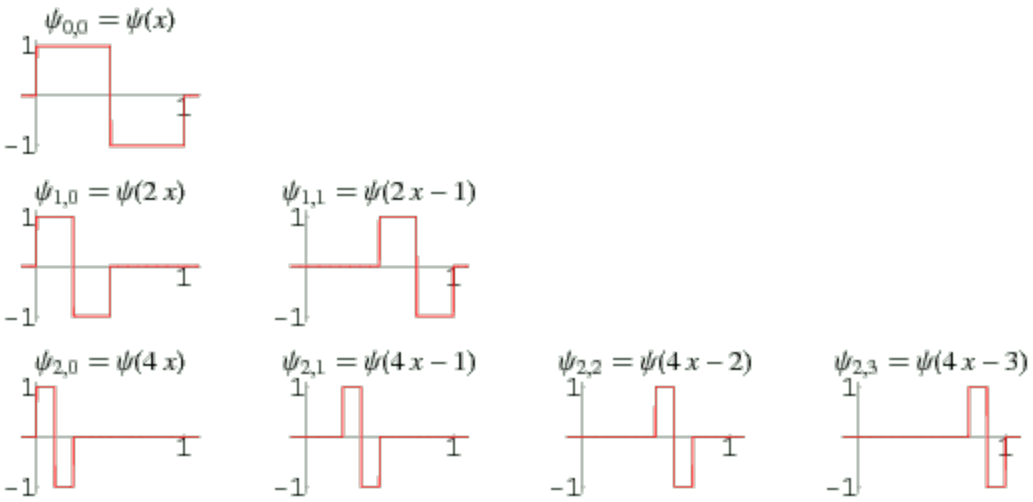
Once again, we will be dealing with two-dimensional, discrete time signals. Let us consider the simplest wavelet in discrete form: the Haar transform. It is based on the Haar wavelet

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \le t < 1/2, \\ -1 & \text{if } 1/2 \le t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

It also needs the scaling function

$$\phi(t) = \begin{cases} 1 & \text{if } 0 \le t < 1, \\ 0 & \text{otherwise} \end{cases}$$

to take the DC term into account.



Leaving out some details, here's how we can discretise this. Let $N = 2^n$, then the Haar matrix $\mathbf{H}_N$ is defined recursively as follows. Firstly,

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Then

$$\mathbf{H}_{2N} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{H}_N \otimes (1,1) \\ \mathbf{I}_N \otimes (1,-1) \end{pmatrix},$$

where $\mathbf{I}_N$ is the identity matrix and $\otimes$ denotes the Kronecker product.

Note: the Kronecker product of two matrices $\mathbf{A}$ and $\mathbf{B}$ is (informally) defined as follows. Say $\mathbf{A}$ is $m \times n$ and $\mathbf{B}$ is $r \times s$, then $\mathbf{K} = \mathbf{A} \otimes \mathbf{B}$ is an $mr \times ns$ matrix, where every entry $a_{i,j}$ of $\mathbf{A}$ has been replaced by the whole matrix $a_{i,j}\mathbf{B}$.

For instance, we have

$$\begin{aligned}
\mathbf{H}_4 &= \frac{1}{\sqrt{2}} \begin{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes (1,1) \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes (1,-1) \end{pmatrix} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} \cdot (1,1) & \frac{1}{\sqrt{2}} \cdot (1,1) \\ \frac{1}{\sqrt{2}} \cdot (1,1) & -\frac{1}{\sqrt{2}} \cdot (1,1) \\ 1 \cdot (1,-1) & 0 \cdot (1,-1) \\ 0 \cdot (1,-1) & 1 \cdot (1,-1) \end{pmatrix} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix}.
\end{aligned}$$

We then have

$$\mathbf{H}_4 = \frac{1}{\sqrt{4}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}$$

$$\mathbf{H}_8 = \frac{1}{\sqrt{8}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{pmatrix}.$$

(Yes, it is the same as for the DWHT for $N = 2$. But it's different for $N = 4$ and after that!)

The Haar matrix can be decomposed into "computing averages and differences" as follows. For all N powers of 2, let

$$\Delta_N = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \ddots & \dots & \dots & \dots & 1 & 1 \\ 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & \ddots & \dots & \dots & \dots & 1 & -1 \end{pmatrix},$$

so that

$$\Delta_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix},$$

$$\Delta_4 = \frac{1}{\sqrt{4}}\begin{pmatrix}1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 1 & -1 & 0 & 0\\ 0 & 0 & 1 & -1\end{pmatrix},$$

$$\Delta_8 = \frac{1}{\sqrt{8}}\begin{pmatrix}1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1\\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1\end{pmatrix}.$$

We can then decompose, e.g. $\mathbf{H}_8$ as follows:

$$\mathbf{H}_8 = \left(\begin{array}{c|c}\Delta_2 & \mathbf{0}\\ \hline \mathbf{0} & \mathbf{I}_6\end{array}\right)\left(\begin{array}{c|c}\Delta_4 & \mathbf{0}\\ \hline \mathbf{0} & \mathbf{I}_4\end{array}\right)\begin{pmatrix}\Delta_8\end{pmatrix}$$

$$\mathbf{H}_8 = \left(\begin{array}{cc|cccccc}c & c & 0 & 0 & 0 & 0 & 0 & 0\\ c & -c & 0 & 0 & 0 & 0 & 0 & 0\\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\end{array}\right)\left(\begin{array}{cccc|cccc}b & b & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & b & b & 0 & 0 & 0 & 0\\ b & -b & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & b & -b & 0 & 0 & 0 & 0\\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\end{array}\right)\begin{pmatrix}a & a & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & a & a & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & a & a & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & a & a\\ a & -a & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & a & -a & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & a & -a & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & a & -a\end{pmatrix},$$

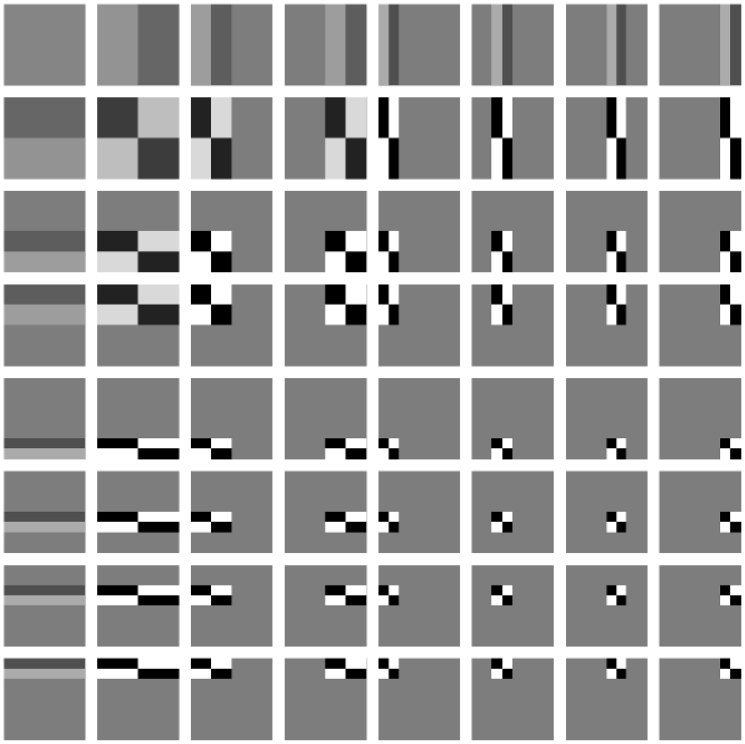
where $c = \frac{1}{\sqrt{2}}$, $b = \frac{1}{\sqrt{4}}$, and $a = \frac{1}{\sqrt{8}}$.

This product should be read from right to left: $\mathbf{H}_8 = \mathbf{CBA}$. First, $\mathbf{A}$ computes the four averages of adjacent of points, and keeps their differences in order to remain reversible. Second, $\mathbf{B}$ does the same as $\mathbf{A}$, but only for the four averages (and leaves the differences untouched), thus creating two more differences. Finally, $\mathbf{C}$ computes the average of the remaining two averages, to get one final average and seven other differences.

We associate with each iteration a quantity called resolution, which is defined as the number of remaining averages at the end of the iteration. The resolutions after each of the three iterations above are $4(=2^2)$, $2(=2^1)$, and $1(=2^0)$.

We can think of the averages as a coarse resolution representation of the original image, and of the details as the data needed to reconstruct the original image from this coarse resolution. If the pixels of the image are correlated, the coarse representation will resemble the original pixels, while the details will be small.

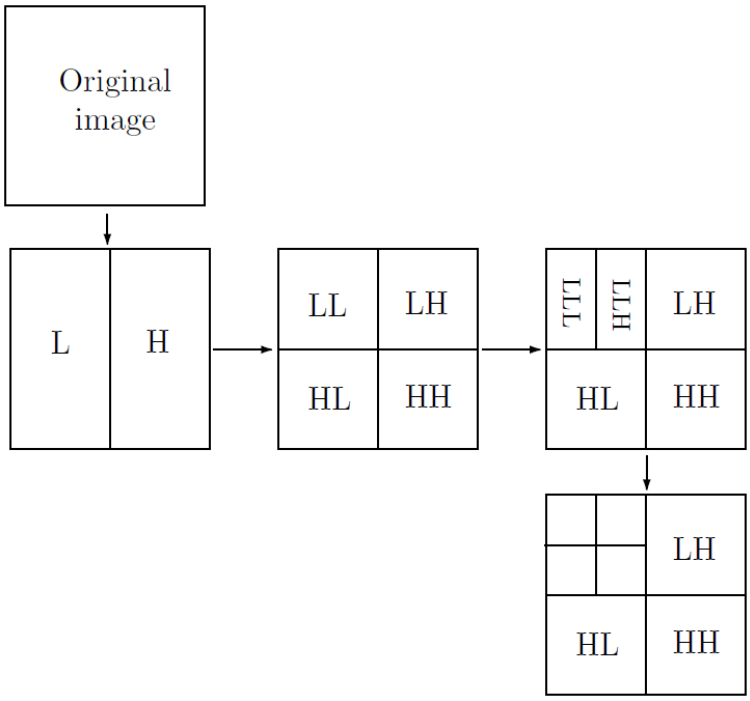
Here are the basis matrices of the Haar transform for $N = 8$.



The Haar transform in 2D The simplest way of applying the Haar transform in 2D is the standard image wavelet transform, where we simply apply the 1D-transform row-wise and then column-wise. That is, we do

$$\Theta = \mathbf{H}(\mathbf{X}\mathbf{H}^\top).$$

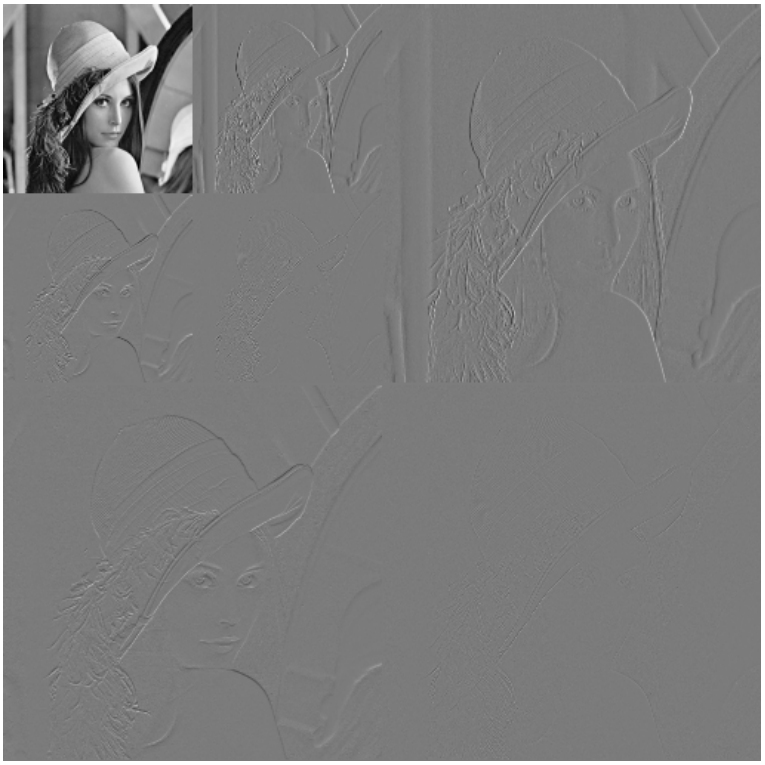
Another, much more common, technique is the pyramid image wavelet transform. The idea is to decompose the Haar matrix as a chain of "averages and differences" computations and alternate row and column operations.



Typically, the averages (that end up in the top left hand region) have large values, while the differences (in the three other regions) tend to have small values. Those regions are called subbands. Subbands actually reflect different geometrical artifacts of the image:

- the upper-right subband (usually referred to as LH) corresponds to vertical artifacts;
- the lower-left subband (usually referred to as HL) corresponds to horizontal artifacts;
- the lower-right subband (usually referred to as HH) corresponds to diagonal artifacts.

Below is a typical image decomposition using the pyramid decomposition.



8.3 See further

Filter banks The most common way of implementing the Discrete Wavelet Transform (not just for Haar, but for any wavelet) is via the use of filter banks. Here's a rapid explanation of the intuition. We can view $\mathbf{H}_2$ as using two filters:

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

the first row corresponds to a lowpass filter, that only takes the DC term (low frequency); the second row corresponds to a highpass filter, that only takes the high frequency term. In general, the "averages and differences" computation can be viewed as applying two simple filters on different parts of the data; those filters are then placed in series to compute the whole transform.

Various image decompositions Even though the pyramid image decomposition is by far the most common, many different image decompositions have been proposed, e.g. Laplacian pyramid, line, quincux, (adaptive) wavelet packet transform, full wavelet decomposition, etc.

Other wavelets We will see two other wavelets in the next lecture. But many other wavelets have been proposed: Meyer, Morlet, Shannon, the large family of Daubechies wavelets, that of Coifman, etc. Their definition can range from easy (e.g. Morlet) to highly involved (Daubechies).

8.4 Exercises

Exercise 8.1. What is the number of nonzero entries in $\mathbf{H}_N$?

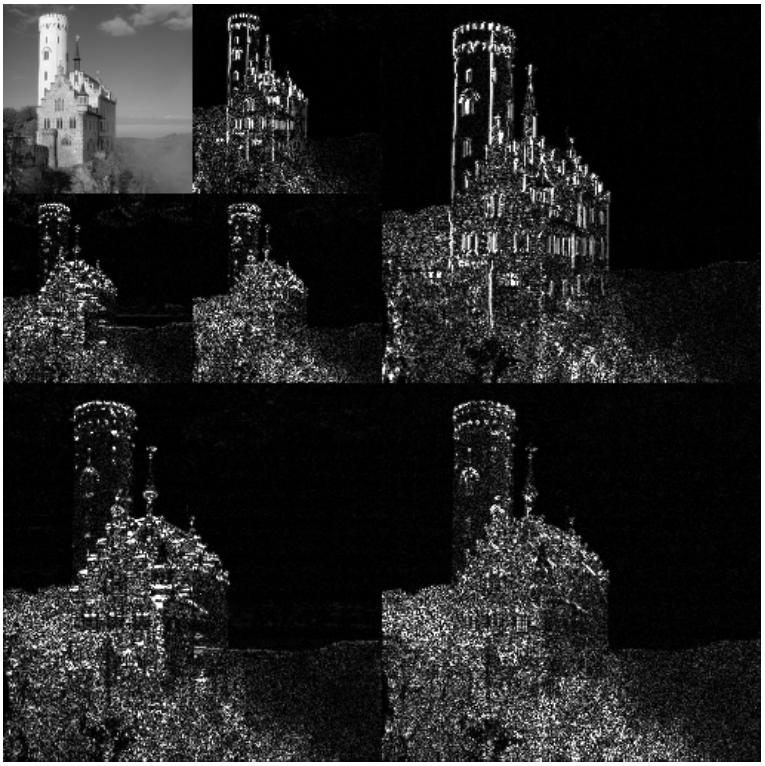
Exercise 8.2. What is the inverse of $\mathbf{H}_N$? Is that matrix orthogonal?

Exercise 8.3. Compute the two-dimensional Haar transform of the $\mathbf{X}$ data from Lecture 7 using the pyramid decomposition.

9 Wavelet coding II: JPEG 2000

9.1 Objectives of JPEG 2000

- High compression efficiency. Bitrates of less than 0.25 bpp are expected for highly detailed grayscale images.
- The ability to handle large images, up to $2^{32} \times 2^{32}$ pixels (the original JPEG can handle images of up to $2^{16} \times 2^{16}$).
- Progressive image transmission. The proposed standard can decompress an image progressively by SNR, resolution, color component, or region of interest.
- Easy, fast access to various points in the compressed stream. The decoder can pan/zoom the image while decompressing only parts of it. The decoder can rotate and crop the image while decompressing it.
- Error resilience. Error-correcting codes can be included in the compressed stream, to improve transmission reliability in noisy environments.



9.2 Operations before wavelet transform coding

A colour image consists of three colour components (typically RGB). First of all, we perform a DC level shifting on each component. If the pixel values range in the interval $[0, 2^k - 1]$, then we subtract $2^k - 1$ to each pixel value. Then we transform the components by means of either a reversible component transform (RCT) or an irreversible component transform (ICT). Both are very similar to the YCbCr conversion used in JPEG. The RCT is used for lossless compression; the ICT is used for lossy compression. Each transformed component is then compressed separately.

More precisely, let I_0, I_1, I_2 denote the three colour components of a certain pixel (after DC shifting). Then the RCT is the triple (Y_0, Y_1, Y_2) given by

$$
\begin{aligned}
Y_0 &= \left\lfloor \frac{I_0 + 2I_1 + I_2}{4} \right\rfloor, \\
Y_1 &= I_2 - I_1, \\
Y_2 &= I_0 - I_1.
\end{aligned}
$$

The ICT is given by

$$\begin{aligned} Y_0 &= 0.299I_0 + 0.587I_1 + 0.144I_2, \\ Y_1 &= -0.16875I_0 - 0.33126I_1 + 0.5I_2, \\ Y_2 &= 0.5I_0 - 0.41869I_1 - 0.08131I_2. \end{aligned}$$

Tiling Each transformed colour component of the image is partitioned into rectangular, nonoverlapping tiles. Since the colour components may have different resolutions, they may use different tile sizes. Tiles may have any size, up to the size of the entire image (i.e. one tile). All the tiles of a given colour component have the same size, except those at the edges. Each tile is compressed individually.

9.3 The wavelets used

JPEG 200 used two different wavelets, one for lossy compression and the other for lossless compression. The details are given below, but of course you needn't remember them.

We now denote a row of extended pixels in a tile by $P(k), P(k+1), \ldots, P(m)$. Since the pixels have been extended, index values below k and above m can be used. The LeGall-Tabatabai 5/3 (LGT 5/3) transform computes wavelet coefficients $C(i)$ by executing the following two steps successively:

$$\begin{aligned} C(2i+1) &= P(2i+1) - \left\lfloor \frac{P(2i)+P(2i+2)}{2} \right\rfloor && k-1 \le 2i+1 < m+1 && \text{step 1} \\ C(2i) &= P(2i) + \left\lfloor \frac{C(2i-1)+C(2i+1)+2}{4} \right\rfloor && k \le 2i < m+1 && \text{step 2.} \end{aligned}$$

The Cohen–Daubechies–Feauveau 9/7 (CDF 9/7) wavelet transform is computed by executing the following six steps successively:

$$\begin{aligned} C(2i+1) &= P(2i+1) + \alpha[P(2i)+P(2i+2)], && k-3 \le 2i+1 < m+3 && \text{step 1} \\ C(2i) &= P(2i) + \beta[C(2i-1)+C(2i+1)], && k-2 \le 2i < m+2 && \text{step 2} \\ C(2i+1) &= C(2i+1) + \gamma[C(2i)+C(2i+2)], && k-1 \le 2i+1 < m+1 && \text{step 3} \\ C(2i) &= C(2i) + \delta[C(2i-1)+C(2i+1)], && k \le 2i < m && \text{step 4} \\ C(2i+1) &= -KC(2i+1), && k \le 2i+1 < m && \text{step 5} \\ C(2i) &= \frac{1}{K}C(2i), && k \le 2i < m && \text{step 6} \end{aligned}$$

where the five constants (wavelet filter coefficients) used by JPEG 2000 are given by $\alpha = -1.586134342$, $\beta = -0.052980118$, $\gamma = 0.882911075$, $\delta = 0.443506852$, and $K = 1.230174105$.

9.4 JPEG 2000 encoding

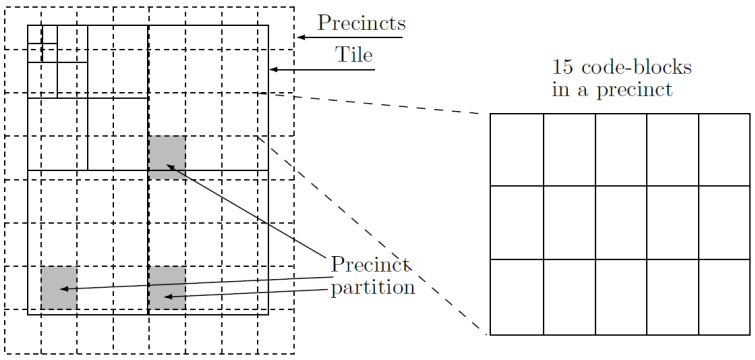
In this section, we give a brief overview of how JPEG 2000 encodes an image. We shall skip many details.

Basic idea The basic idea is similar to transform coding: transform, quantise, and encode. In JPEG, this idea was applied to every 8x8 pixel block. On the other hand, JPEG 2000 uses a sophisticated means of partitioning the image, using tiles, precincts and code-blocks. Also, the binary encoding in JPEG 2000 is much more sophisticated than the Huffman coding used in JPEG.

Wavelet transform The one-dimensional wavelet transforms described above are applied L times, where L is a parameter (either user-controlled or set by the encoder), and are interleaved on rows and columns to form L resolutions of subbands. Resolution $L-1$ is the original image, and resolution 0 is the lowest-frequency subband.

Quantization Each subband can have a different quantization step size. Each wavelet coefficient in the subband is divided by the quantization step size and the result is truncated. The quantization step size may be determined iteratively in order to achieve a target bitrate (i.e., the compression factor may be specified in advance by the user) or in order to achieve a predetermined level of image quality.

Precincts and Code-Blocks Consider a tile in a colour component. The original pixels are wavelet transformed, resulting in subbands of L resolution levels. A grid of rectangles known as precincts is now imposed on the entire image. The origin of the precinct grid is anchored at the top-left corner of the image and the dimensions of a precinct (its width and height) are powers of 2. Notice that subband boundaries are generally not identical to precinct boundaries.



We now examine the three subbands of a certain resolution and pick three precincts located in the same regions in the three subbands (the three gray rectangles in the figure above). These three precincts constitute a precinct partition. The grid of precincts is now divided into a finer grid of code-blocks, which are the basic units to be arithmetically coded.

In our example, each precinct is subdivided into 15 code-blocks; therefore, a precinct partition consists of 45 code-blocks. Think of the tiles, precincts, and code-blocks as coarse, medium, and fine partitions of the image, respectively. Partitioning the image into smaller and smaller units helps in creating memory-efficient implementations, streaming, and allowing easy access to many points in the bitstream. However, most implementations may ignore this partitioning and just use one tile, one precinct and one code-block for the image.

Entropy coding The wavelet coefficients of a given code-block are then encoded. The method used by JPEG 2000 is called Embedded Block Coding with Optimized Truncation (EBCOT). It is a context-based compression algorithm, based on the arithmetic encoder MQ (a variant of QM). We shall skip this part entirely.

Packets and layers After all the bits of all the coefficients of all the code-blocks of a precinct partition have been encoded into a short bitstream, a header is added to that bitstream, thereby turning it into a packet. A packet can be considered a quality increment for one level of resolution at a certain spatial location.

A layer is a set of packets, one from each precinct partition of each resolution level. Thus, a layer is a quality increment for the entire image at full resolution.

Progressive Transmission This is an important feature of JPEG 2000. The standard provides four ways of progressively transmitting and decoding an image: by resolution, quality, spatial location, and component. Progression is achieved simply by storing the packets in a specific order in the bitstream.

For example, quality progression can be achieved by arranging the packets in layers, within each layer by component, within each component by resolution level, and within each resolution level by precinct partition. Resolution progression is achieved when the packets are arranged by precinct partition (innermost nesting), layer, image component, and resolution level (outermost nesting).

When an image is encoded, the packets are placed in the bitstream in a certain order, corresponding to a certain progression. If a user or an application require a different progression (and thus a different

order of the packets), it should be easy to read the bitstream, identify the packets, and rearrange them. This process is known as parsing, and is an easy task because of the many markers embedded in the bitstream. Thus, the bitstream can be parsed without having to decode any of it.

9.5 See further

DjVu DjVu is a wavelet-based compression algorithm and file format used for digitized books. Using wavelets makes sense for such pictures, Indeed, JPEG is designed for continuous-tone images (little variations); other techniques were designed for discrete-tone images (e.g. FABD); but digitized books have both features.

Fingerprint images The Wavelet Scalar Quantization algorithm (WSQ) is a compression algorithm used for gray-scale fingerprint images. It has become a standard for the exchange and storage of fingerprint images. This method is preferred over JPEG because at the same compression ratios WSQ doesn't present the "blocking artifacts" and loss of fine-scale features that are not acceptable for identification in financial environments and criminal justice.

9.6 Exercises

Exercise 9.1. Write a program that visualises the Haar, LGT 5/3 and CDF 9/7 wavelet pyramid image decomposition.

10 Video compression

10.1 The main tenets of video compression

Video compression is based on two principles:

1. spatial redundancy: in each frame because of pixel correlation
2. temporal redundancy: a video frame is very similar to its immediate neighbours (predecessor and successor).

A typical technique for video compression starts by encoding the first frame using a still image compression method. It should then encode several successive frames by identifying the differences between a frame and its predecessor, and encoding these differences. Those are referred to as an inter frame (or non intra frame) If a frame is very different from its predecessor, it should be coded independently of any other frame. Such a frame is called an intra frame.

Encoding a frame F_i in terms of its predecessor F_{i-1} introduces some distortions. Using a long sequence of inter frames may lead to accumulated errors. This is why intra frames should be used from time to time inside a sequence, not just at its beginning. Therefore, a inter frame could be encoded based on both past and future frames: those are called bidirectional.

In short, we have three kinds of frames:

- Intra frames, usually labelled I. An I frame is decoded independently of all the other frames.
- Predictive frames, labelled P. A P frame is decoded using the preceding P frame.
- Bidirectional frames, labelled B. A B frame is decoded using the preceding and following I or P frames.

The encoding and decoding is then done in a different order to the display order.

10.2 Motion compensation

Motion compensation The difference between consecutive frames is usually small because it is the result of moving the scene, the camera, or both between frames. This feature can therefore be exploited to achieve better compression. If the encoder discovers that a part P of the preceding frame has been rigidly moved to a different location in the current frame, then P can be compressed by writing the following three items on the compressed stream: its previous location, its current location, and information identifying the boundaries of P.

Motion compensation is effective if objects are just translated, not scaled or rotated. Drastic changes in illumination from frame to frame also reduce the effectiveness of this method.

In principle, such a part can have any shape. In practice, we are limited to equal size square blocks. The encoder scans the current frame block by block. For each block B it searches the preceding frame for a similar block C. Finding such a block, the encoder writes the difference between its past and present locations on the output. This difference is of the form

$$(C_x - B_x, C_y - By) = (\Delta_x, \Delta_y),$$

so it is called a motion vector.

Block Search If B is the current block in the current frame, then the previous frame has to be searched for a block similar to B. The search is normally restricted to a small area (called the search area) around B, defined by the maximum displacement parameters dx and dy. These parameters specify the maximum horizontal and vertical distances, in pixels, between B and any matching block in the previous frame. If B is a square with side b, the search area will contain $(b+2dx)(b+2dy)$ pixels and will consist of $(2dx+1)(2dy+1)$ distinct, overlapping $b \times b$ squares. The number of candidate blocks in this area is therefore proportional to $dxdy$.

Distortion measure This is the most sensitive part of the encoder. The distortion measure selects the best match for block B. It has to be simple and fast, but also reliable. Here are a few choices.

The mean absolute error (or mean absolute difference) calculates the average of the absolute differences between a pixel $B_{i,j}$ in B and its counterpart $C_{i,j}$ in a candidate block C:

$$\frac{1}{b^2}\sum_{i=1}^{b}\sum_{j=1}^{b}|B_{i,j}-C_{i,j}|.$$

This measure is calculated for each of the $(2dx+1)(2dy+1)$ distinct, overlapping $b \times b$ candidate blocks, and the smallest distortion (say, for block $\hat{C}$) is examined. If it is smaller than the search threshold, then $\hat{C}$ is selected as the match for B. Otherwise, there is no match for B, and B has to be encoded without motion compensation. The mean square difference is similar:

$$\frac{1}{b^2}\sum_{i=1}^{b}\sum_{j=1}^{b}(B_{i,j}-C_{i,j})^2.$$

The pel difference classification (PDC) counts how many differences $|B_{i,j}-C_{i,j}|$ are below a given PDC parameter p.

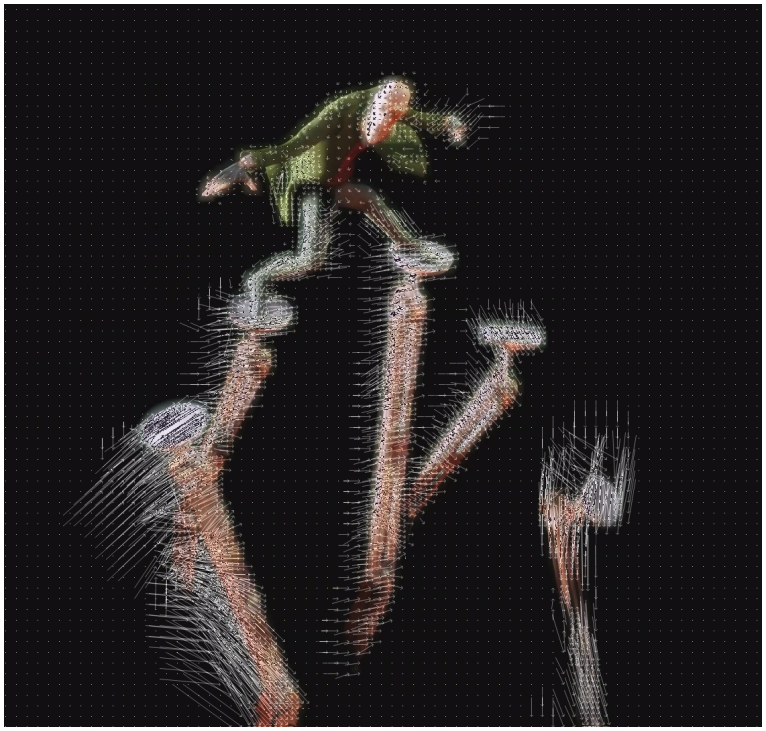
Motion Vector Correction Once a block C has been selected as the best match for B, a motion vector is computed as the difference between the upper-left corner of C and the upper-left corner of B.

Regardless of how the matching was determined, the motion vector may be wrong because of noise, local minima in the frame, or because the matching algorithm is not perfect. It is possible to apply smoothing techniques to the motion vectors after they have been calculated, in an attempt to improve the matching. Spatial correlations in the image suggest that the motion vectors should also be correlated. If certain vectors are found to violate this, they can be corrected.

Coding motion vectors A large part of the current frame (perhaps close to half of it) may be converted to motion vectors, which is why the way these vectors are encoded is crucial; it must also be lossless. Two properties of motion vectors help in encoding them:

1. they are correlated;
2. their distribution is nonuniform.

Therefore they can be efficiently coded using e.g. Huffman or arithmetic coding.



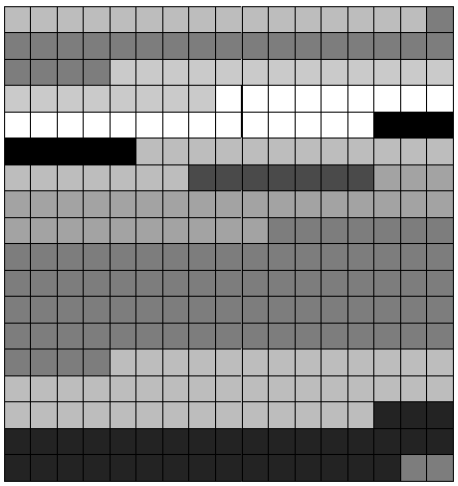
Coding the prediction error Motion compensation is lossy, since a block B is normally matched to a slightly different block C. Compression can be improved by coding the difference between the current uncompressed and compressed frames on a block by block basis and only for blocks that differ much. This is usually done by transform coding. The difference is then written on the output, following each frame.

10.3 MPEG-1

This section gives a brief overview of the MPEG-1 standard. We focus on the video compression part; MPEG also compresses audio. MPEG uses I, P, and B pictures (frames). The pictures are arranged in a certain order, called the coding order, but (after being decoded) they are output and displayed in a different order, called the display order.

Macroblocks and slices The basic building block of an MPEG picture is the macroblock. It consists of a 16×16 block of luminance (grayscale) samples and two 8×8 blocks of the matching chrominance samples. The MPEG compression of a macroblock consists mainly in passing each of the six blocks through a DCT, then quantizing and encoding the results. It is very similar to JPEG compression (but with different quantization and code tables).

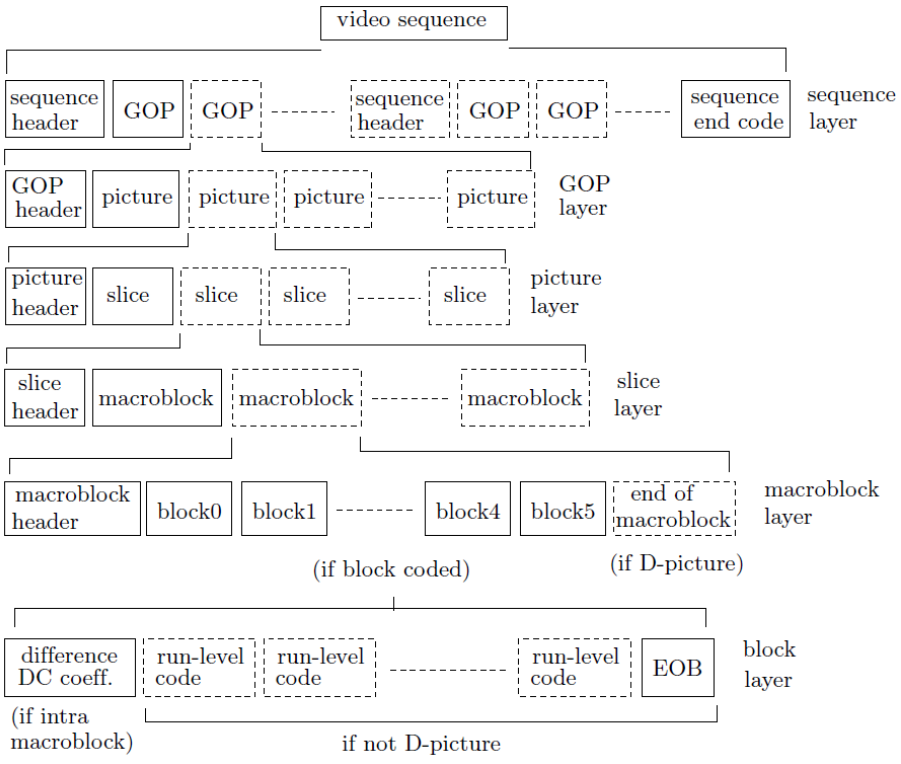
A picture in MPEG is organized in slices, where each slice is a contiguous set of macroblocks (in raster order) that have the same luminance component. The concept of slices makes sense because a picture may often contain large uniform areas, causing many contiguous macroblocks to have the same luminance. Each square in the picture below is a macroblock; those are ordered in slices. Notice that a slice can continue from scan line to scan line.



Transform coding When a picture is encoded in nonintra mode (i.e., it is encoded by means of another picture, normally its predecessor), the MPEG encoder generates the differences between the pictures, then applies the DCT to the differences. In such a case, the DCT does not contribute much to the compression, because the differences are already decorrelated. Nevertheless, the DCT is useful even in this case, since it is followed by quantization, and the quantization in nonintra coding can be quite deep. The precision of the numbers processed by the DCT in MPEG also depends on whether intra or nonintra coding is used.

The actual quantization step is more sophisticated than in JPEG, so we will not go through it here. The quantized numbers QDCT are Huffman coded, using the nonadaptive Huffman method and Huffman code tables that were computed by gathering statistics from many training image sequences. The particular code table being used depends on the type of picture being encoded. The encoding part is similar to that of JPEG, using a zigzag scan, the EOB symbol, etc.

Layer structure The output of the encoder is arranged into layers: sequence, group of pictures (GOP), picture, slice, macroblock, and block. (Note that, on top of I, P and B pictures, MPEG also allows for D pictures, where only the DC information is kept; those are very rare.)



10.4 See further

Suboptimal search methods In motion compensation, instead of searching all the candidate blocks in the search area, some algorithms only search some of them. These suboptimal search methods are heuristics that trade compression efficiency for speed.

MP3 MPEG-1 had to include a way of compressing audio. The MPEG-1 Audio Layer III (MP3) audio compression format then became massively popular, and has outlived the video part of MPEG-1!

MPEG-4 and H.264 The MPEG standard has evolved over time. After MPEG-1 came MPEG-2, used in DVDs. A leap forward was made in MPEG-4, with the ability to define objects and to code on the object level. This was further improved in H.264, also referred to as Advanced Video Coding (AVC), or MPEG-4 Part 10, or Advanced Video Coding (MPEG-4 AVC). H.264 is the standard now, used in e.g. BluRay, and is still under maintenance and improvement.

10.5 Exercises

Exercise 10.1. We could generalise the mean absolute and squared error as follows. Let $x = (x_0, \dots, x_{N-1})$ be a vector, and let $p \geq 1$, then its p-norm is defined as

$$\|x\|_p := \left(\sum_{i=0}^{N-1} |x_i|^p \right)^{\frac{1}{p}}.$$

(Forgetting the scaling factor $1/b^2$, the absolute error is given by $\|B - C\|_1$.)

1. Prove that the p-norm is indeed a norm, i.e. for all vectors x and y and any scalar a,

$$\begin{aligned} \|x\|_p &\geq 0 \\ \|x\|_p = 0 &\implies x = 0 \\ \|x + y\|_p &\leq \|x\|_p + \|y\|_p \\ \|ax\|_p &= |a| \|x\|_p. \end{aligned}$$

2. Comment on the "fairness" of the p-norm, as p varies from 1 towards infinity.
3. In particular, what is $\lim_{p \to \infty} \|x\|_p$?
4. What happens when $0 < p < 1$?
5. What do you get for $p = 0$?

Exercise 10.2. We mentioned that motion vectors were correlated and with a nonuniform distribution. How could one take advantage of these two properties when encoding? (You needn't come up with the solution that's actually used in MPEG, just think of a few different approaches and try to assess their benefits/drawbacks.)

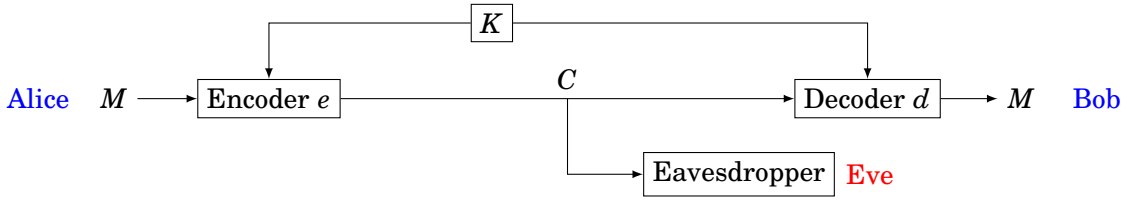
Chapter 2

Cryptography

11 Introduction to cryptography

11.1 Fundamentals of cryptography

Basic set-up All messages are sent through an external channel:



Now we are interested in protecting the contents of the message from eavesdroppers, known as Eve. The message sender, and encoder, is traditionally called Alice. The receiver, and decoder, is traditionally called Bob.

Notation The message $M \in \mathscr{M}$ is known as the plaintext. Alice and Bob have some secret information $K \in \mathscr{K}$, known as the key. Alice encrypts using an encryption function $e : \mathscr{M} \times \mathscr{K} \to \mathscr{C}$. The transmitted sequence

$$C = e(M,K) \in \mathscr{C}$$

is called the ciphertext. Bob receives C, and decrypts using the decryption function $d : \mathscr{C} \times \mathscr{K} \to \mathscr{M}$. We require that

$$d(e(M,K),K) = M$$

for all plaintexts M. This implies that for a given key K, the encryption function is injective:

$$e(M_1,K) = e(M_2,K) \implies M_1 = M_2.$$

Kerchoff's principles Clearly, if Eve knows both the decoding algorithm d and the key K, then she can decipher the ciphertext. So at least one of the two should be kept secret. Auguste Kerchoffs argued that only the key should be kept secret:

> The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.

This is known as Kerchoff's principle.

There are three main arguments supporting Kerchoff's principle:

1. It is much easier to maintain secrecy of a short key than to keep secret the (more complex) algorithm they are using. (Imagine using encryption to secure the communication between all pairs of employees in some organisation.)

2. If the shared, secret information is ever exposed, then it will be much easier to change a key than to replace an encryption scheme.

3. For large-scale deployment, it is much easier for users to all rely on the same encryption algorithm or software (with different keys) than for everyone to use their own custom algorithm. In fact, it is desirable for encryption schemes to be standardise so that compatibility is ensured by default and users will use a scheme that has undergone public scrutiny without weaknesses being found.

Kerchoff's principle leads us to use cryptosystems whose designs are completely public, instead of trying to rely on "security by obscurity."

Breaking a cryptosystem The enemy (Eve) is assumed to know the encryption function e, the decryption function d, and also to have various additional information, such as language statistics etc. She will certainly have some ciphertext C. All she lacks to decipher to message is the key K.

There are three standard forms of attack:

1. Ciphertext-only attack.

2. Known-plaintext attack. Eve is assumed to also have a sample of plaintext and its encryption.

3. Chosen-plaintext attack. Eve is able to acquire chosen samples of plaintext/ciphertext pairs.

Current cryptosystems are required to be secure against chosen-plaintext attacks.

11.2 Examples of cryptosystems

Early history Aeneas Tacticus has a chapter "On secret messages" in his classic military manual on the art of war (c. 360BC). His most secret method was to thread a string through holes in a piece of bone in an order corresponding to the letters of the message.

In the Caesar cipher, allegedly used by Julius Caesar, involves shifting all the letters of the alphabet a fixed number of letters later in the alphabet (wrapping from z round to a). For instance, shifting by four characters, "pinkfloyd" becomes "tmrojpsch".

Substitution cipher In the substitution cipher, the key is a permutation of the letters of the alphabet:

$$\begin{aligned} &\texttt{ABCDEFGHIJKLMNOPQRSTUVWXYZ} \\ K = &\texttt{QWERTYUIOPASDFGHJKLZXCVBNM} \end{aligned}$$

Encryption is achieved by applying the permutation letter-wise. For instance, using the key above,

$$\begin{aligned} M &= \texttt{SENDMORETROOPS} \\ C &= \texttt{LTFRDGKTZKGGHL} \end{aligned}$$

This scheme is highly vulnerable to frequency analysis for large texts. As we saw earlier, the letter frequencies are heavily biased in text. Since the substitution cipher preserves the character frequencies, the attacker can guess the key by analysing the frequencies in the text. For instance, the most frequent letter in the ciphertext is probably the image of E under the permutation, and so on.

Vigenère cipher The Vigenère cipher is a polyalphabetic substitution cipher, an extension of the Caesar and (simple) substitution ciphers. The key is a sequence of letters, e.g. $K = \texttt{BOTTLEOFRUM}$ The key is repeated until it is as long as the message, and corresponding message and key characters are added (mod 26).

$$\begin{aligned} M &= \texttt{THETREASUREISUNDERTHEROCKBYTHETHIRDPALMTREE} \\ K &= \texttt{BOTTLEOFRUMBOTTLEOFRUMBOTTLEOFRUMBOTTLEOFRU} \\ C &= \texttt{VWYNDJPYMMRKHOHPJGZZZEQREVKYWKLCVTSJUXRIXWZ} \end{aligned}$$

If the length of the key is guess right, then the attacker can still use frequency analysis.

Autokey Vigenère cipher The basic idea is to break the frequency analysis by having a non-repeating key. Again, the key is a sequence of letters e.g. $K = \texttt{BOTTLEOFRUM}$. The key is repeated only once, instead it is augmented to K^* with the start of the message.

$$M = \texttt{THETREASUREISUNDERTHEROCKBYTHETHIRDPALMTREE}$$
$$K = \texttt{BOTTLEOFRUM}$$
$$K^* = \texttt{BOTTLEOFRUMTHETREASUREISUNDERTHEROCKBYTHETH}$$
$$C = \texttt{VWYNDJPYMMRCAZHVJSMCWWXVFPCYZYBMAGGACKGBWYM}$$

This system was considered unbreakable for nearly 300 years. Then in 1863, a Prussian major discovered that the the length of the keyword can be accurately determined by looking at the separation of repeated digrams. Once the key length is known, frequency analysis can be used to determine the Caesar shift for that set of character positions.

For instance, let us guess a key length of 11. Frequency test each letter of alphabet as candidates for first letter of K. When we hit `B` we decode `T`, use `T` to decode `I`, `I` to get `O`, `O` to get `R`, etc.

11.3 Perfect cryptosystems

Perfect cryptosystems A cryptosystem is perfect if knowledge of the ciphertext C gives absolutely no information about the plaintext M.

Theorem 11.1. *In a perfect cryptosystem there must be at least as many possible keys as possible plaintexts.*

Proof. Suppose $|\mathcal{K}| < |\mathcal{M}|$ and let C be a ciphertext. Let $d(C)$ denote the set of plaintexts that be decrypted from C:

$$d(C) = \{M \in \mathcal{M} : \exists K \in \mathcal{K}\, e(M,K) = C\}.$$

We then have $d(C) \subseteq \mathcal{M}$. Also, since for every key K encryption is injective, we have $|d(C)| \leq |\mathcal{K}| < |\mathcal{M}|$. Thus, $d(C) \subset \mathcal{M}$ and there exists $M^* \in \mathcal{M} \setminus d(C)$: the ciphertext C gives away the information that the plaintext is not M^*. □

One-time pad In the one-time pad, the key length of the Vigenère cipher is extended to be as long as the plaintext. The key characters are generated uniformly at random.

$$M = \texttt{THETREASUREISUNDERTHEROCKBYTHETHIRDPALMTREE}$$
$$K = \texttt{BHMNHRPBHQOILQWDBWPQCJNMYPKIPBZPWFZMDFNRBUC}$$
$$C = \texttt{VPRHZWQUCITRELKHGOJYHBCPJRJCXGTXFXDCERALTZH}$$

Each character of C is now a uniformly random letter, independently of all others. Thus each ciphertext C is equally likely, no matter what the plaintext M is. The One-time pad is a perfect cryptosystem.

There are two main problems with the one-time pad:

1. Key generation. In theory, one would need to generate a very large amount of random characters. Alternatively, one could use a cryptographically secure pseudo-random number generator.

2. Key distribution. The keys are usually not shared amongst only two people, but a whole book of keys is deployed to many agents in the same organisation. This deployment of a large amount of data to many people is not secure in itself.

11.4 See further

Enigma Mechanical cipher machines came into wide use by World War II. The most famous is the German Enigma machine.

Enigma was a polyalphabetic substitution cipher based on electromechanical rotors that changed the substitution cipher after each letter. The substitution cipher would not repeat for 16900 characters. Probably would have been unbreakable if better procedures had been followed during use.

Work on breaking the Enigma cipher led to major advances in computing, in large part by Alan Turing and Gordon Welchman.

11.5 Exercises

Exercise 11.1. I have used a simple substitution cipher to encode some text; the ciphertext is the file `ciphertext1.txt`. Compute the letter frequencies in the paragraph. Look up the letter frequencies for common English on the web. Use this knowledge to decipher the paragraph. If it is too much work to compute letter frequencies for the whole paragraph, you could use a short part of the paragraph. Think about how this would affect the results.

Exercise 11.2. I did the same with another text to get `ciphertext2.txt`. This one may be a bit more difficult - the letter frequencies are a bit odd.

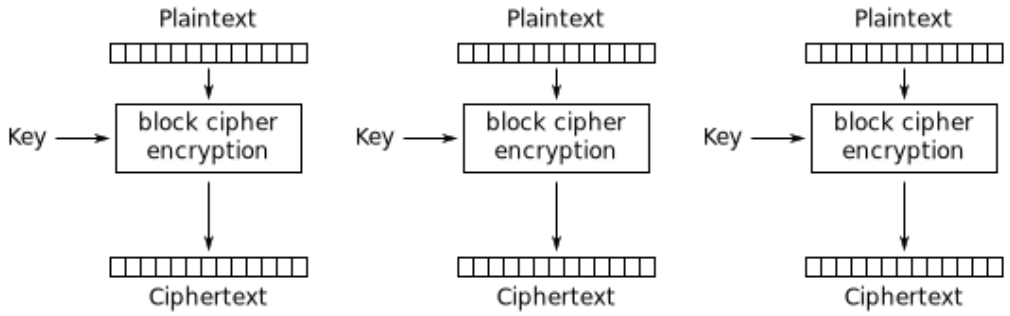
12 Symmetric cryptography I: Block ciphers

An n-bit block cipher is a function $e : \{0,1\}^n \times \mathcal{K} \to \{0,1\}^n$, such that for each key $K \in \mathcal{K}$, $e(M,K)$ is an invertible permutation of $\{0,1\}^n$ (the encryption function for K).

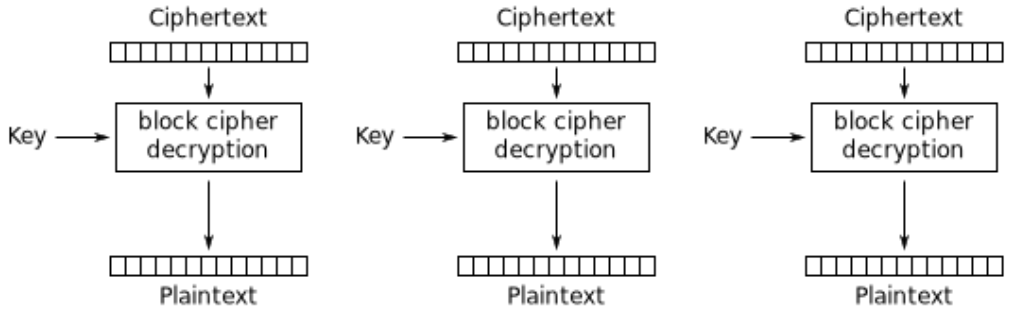
12.1 Modes of operations

A block cipher is not a cryptosystem per se. It is only used to encrypt a particular block of data. It forms a cryptosystem when the mode of operation is specified.

Electronic Codebook The simplest of the encryption modes is the electronic codebook (ECB) mode (named after conventional physical codebooks). The message is divided into blocks, and each block is encrypted separately.

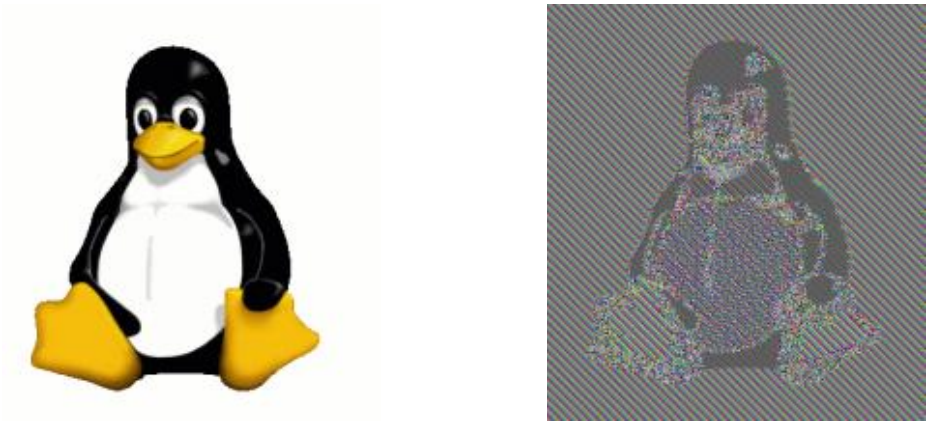


Electronic Codebook (ECB) mode encryption

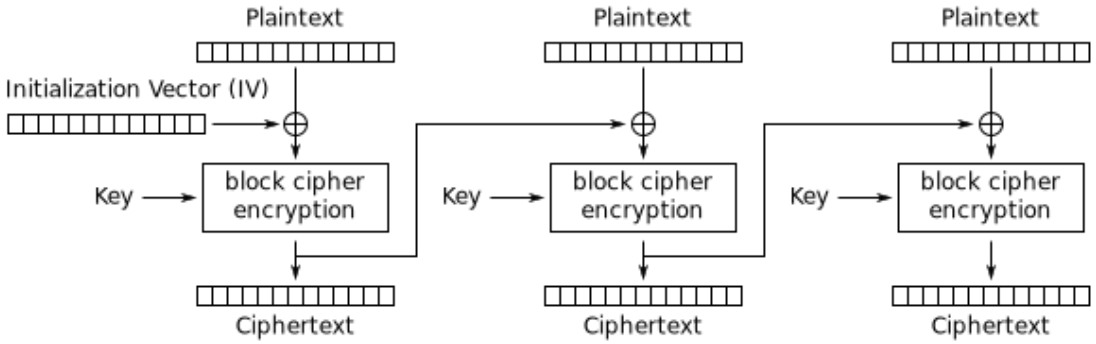


Electronic Codebook (ECB) mode decryption

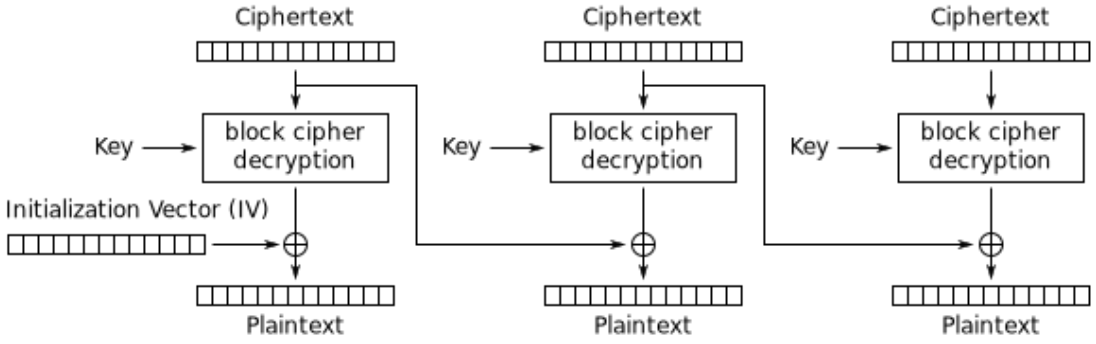
The disadvantage of this method is a lack of diffusion. Because ECB encrypts identical plaintext blocks into identical ciphertext blocks, it does not hide data patterns well Therefore, global patterns across the plaintext are still kept in the ciphertext. As such, ECB is not recommended for use in cryptographic protocols.



Cipher Block Chaining In the cipher block chaining (CBC) mode of operation, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector (IV) must be used in the first block.

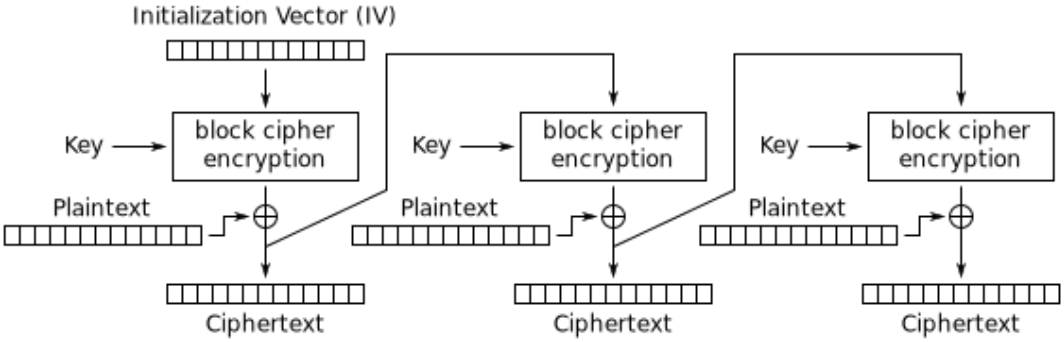


Cipher Block Chaining (CBC) mode encryption

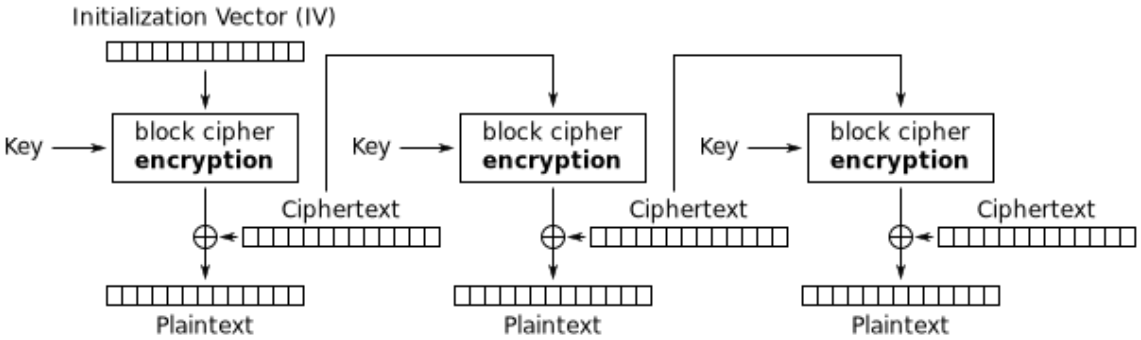


Cipher Block Chaining (CBC) mode decryption

Cipher Feedback The cipher feedback (CFB) mode, in its simplest variation is using the entire output of the block cipher. In this variation, it is very similar to CBC, makes a block cipher into a self-synchronizing stream cipher.



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

12.2 Data Encryption Standard

Feistel cipher A Feistel cipher is an iterated cipher mapping a $2t$-bit plaintext (L_0, R_0), for t-bit blocks L_0 and R_0, to a ciphertext (R_r, L_r), through an r-round process where $r \geq 1$.

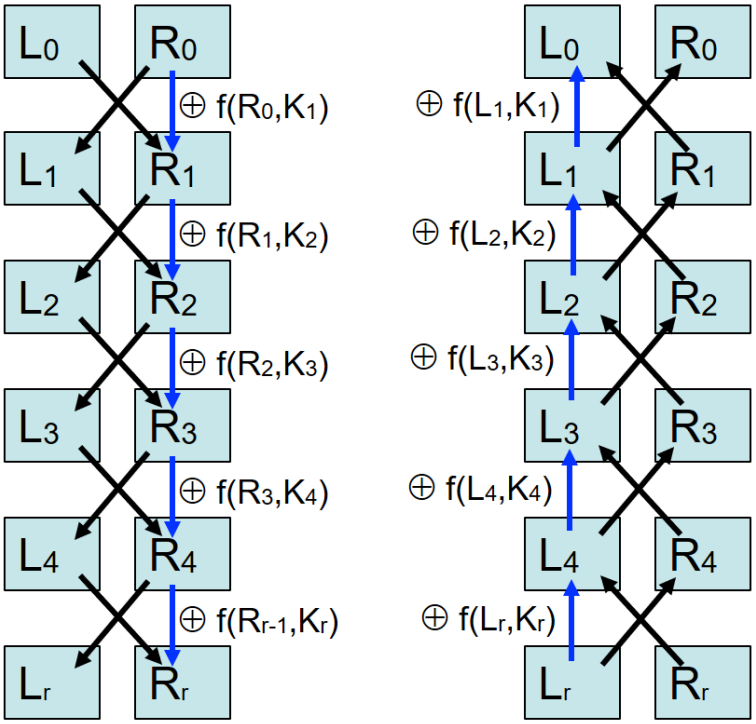
For $1 \leq i \leq r$, round i maps $(L_{i-1}, R_{i-1}), K_i \rightarrow (L_i, R_i)$ as follows:

$$L_i = R_{i-1},$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

where each subkey K_i is derived from the cipher key K.

Typically in a Feistel cipher, $r \geq 3$ and often is even. The Feistel structure specifically orders the ciphertext output as (R_r, L_r) rather than (L_r, R_r); the blocks are exchanged from their usual order after the last round. Decryption is thereby achieved using the same r-round process but with subkeys used in reverse order, K_r through K_1.

The f function of the Feistel cipher need not be invertible to allow inversion of the Feistel cipher. We illustrate that successive rounds of a Feistel cipher operate on alternating halves of the ciphertext, while the other remains constant.



The Data Encryption Standard (DES) is a Feistel cipher which processes plaintext blocks of $n = 64$ bits, producing 64-bit ciphertext blocks.

DES key schedule The effective size of the secret key K is $k = 56$ bits. More precisely, the input key K is specified as a 64-bit key, 8 bits of which (bits $8, 16, \ldots, 64$) are used as odd-parity bits:

$$k_8 = k_1 \oplus k_2 \oplus k_3 \oplus k_4 \oplus k_5 \oplus k_6 \oplus k_7 \oplus 1$$

(and so on for $k_{16}, k_{24}, \ldots, k_{64}$). The 2^{56} keys implement (at most) 2^{56} of the $2^{64}!$ possible bijections on 64-bit blocks. A widely held belief is that the parity bits were introduced to reduce the effective key size from 64 to 56 bits, to intentionally reduce the cost of exhaustive key search by a factor of 256.

Algorithm DES key schedule
INPUT: 64-bit key $K = k_1 \ldots k_{64}$.
OUTPUT: sixteen 48-bit keys K_i, $1 \le i \le 16$.
1. Define v_i, $1 \le i \le 16$ as follows: $v_i = 1$ for $i \in \{1, 2, 9, 16\}$; $v_i = 2$ otherwise.
2. $(C_0, D_0) \leftarrow PC1(K)$.
3. For i from 1 to 16, let $X^{\leftarrow v}$ denote X shifted cyclically to the left v times, then do

$$C_i \leftarrow C_{i-1}^{\leftarrow v_i},$$
$$D_i \leftarrow D_{i-1}^{\leftarrow v_i},$$
$$K_i \leftarrow PC2(C_i, D_i).$$

$PC1$

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

$PC2$

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Feistel function for DES Encryption proceeds in 16 stages or rounds. From the input key K, sixteen 48-bit subkeys K_i are generated, one for each round. Within each round, 8 fixed, 6-to-4 bit substitution mappings (S-boxes) S_i, collectively denoted S, are used. The 64-bit plaintext is divided into 32-bit halves L_0 and R_0. Each round is functionally equivalent, taking 32-bit inputs L_{i-1} and R_{i-1} from the previous round and producing 32-bit outputs L_i and R_i for $1 \le i \le 16$, as follows:

$$\begin{aligned} L_i &= R_{i-1}, \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i), \end{aligned}$$

where

$$f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)).$$

Here E is a fixed expansion permutation mapping R_{i-1} from 32 to 48 bits. P is another fixed permutation on 32 bits. The tables below give E and P, that is

$$\begin{aligned} E(r_1, r_2, r_3, \ldots, r_{31}, r_{32}) &= (r_{32}, r_1, r_2, \ldots, r_{32}, r_1) \\ P(t_1, t_2, t_3, \ldots, t_{31}, t_{32}) &= (t_{16}, t_7, t_{20}, \ldots, t_4, t_{25}). \end{aligned}$$

E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

The S_i S-boxes are given in the table below. The S-boxes are the only nonlinear components of the encryption, and as such are the crucial component of its security. In DES, the S-boxes were introduced as lookup tables, with no indication of why they were chosen like that in the first place.

They are applied as follows. First, let $B_i = (b_1, b_2, b_3, b_4, b_5, b_6)$, then row $= 2b_1 + b_6$, column $= 8b_2 + 4b_3 + 2b_4 + b_5$. Then look-up the table and write the entry in binary (on four bits). Four instance, here is how to compute $S_1(011011)$:

$$\begin{aligned} B &= 011011 \\ \text{row} &= 1 \\ \text{column} &= 13 \\ S_1[1,13] &= 5 = 0101. \end{aligned}$$

row	column number															
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
S_1																
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2																
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
[1]	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
[2]	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
[3]	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

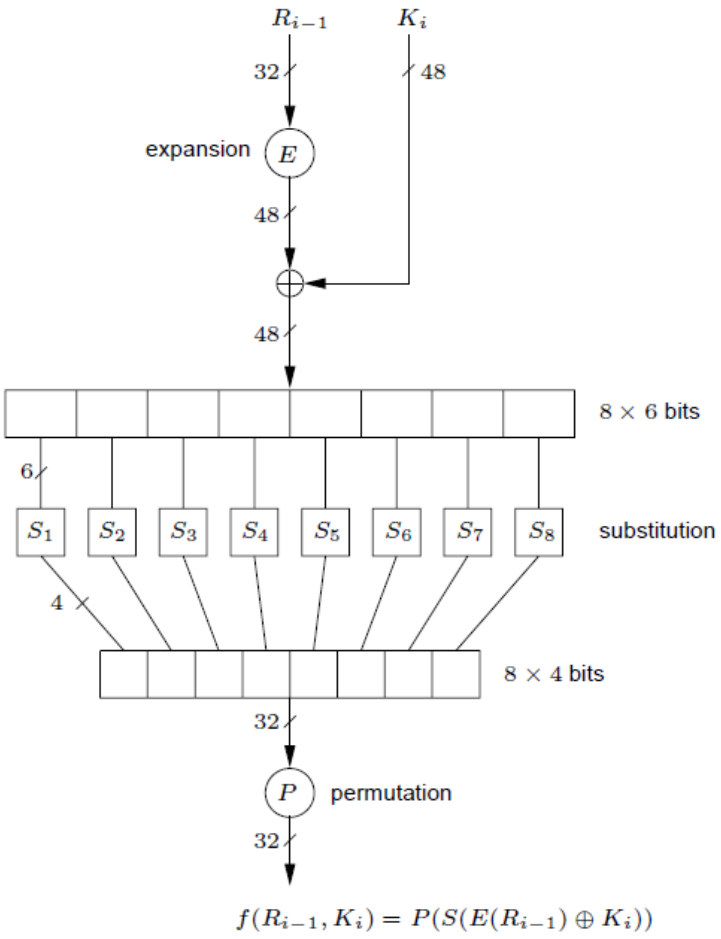
Here is how to compute the Feistel function f:

1. Expand $R_{i-1} = r_1 r_2 \ldots r_{32}$ from 32 to 48 bits using E: $T \leftarrow E(R_{i-1})$. (Thus $T = r_{32} r_1 r_2 \ldots r_{32} r_1$.)

2. $T \leftarrow T \oplus K_i$. Represent T as eight 6-bit character strings: $(B_1, \ldots, B_8) = T$.

3. $T \leftarrow (S_1(B_1), S_2(B_2), \ldots, S_8(B_8))$.

4. $T \leftarrow P(T)$.



The whole DES algorithm An initial bit permutation (IP) precedes the first round; following the last round, the left and right halves are exchanged and, finally, the resulting string is bit-permuted by the inverse of IP, sometimes referred to as final bit permutation $FP = IP^{-1}$. Similarly to the tables above, we have

$$IP(m_1, m_2, \ldots, m_{64}) = (m_{58}, m_{50}, \ldots, m_7)$$
$$FP(b_1, b_2, \ldots, b_{64}) = (b_{40}, b_8, \ldots, b_{25}).$$

IP

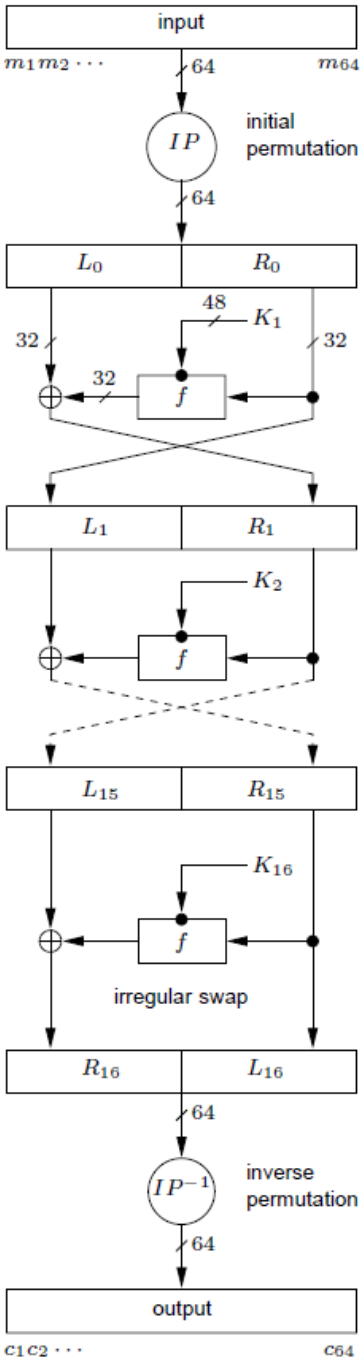
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

FP

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Algorithm Data Encryption Standard (DES)
INPUT: plaintext $M = (m_1, \ldots, m_{64})$; 64-bit key $K = (k_1, \ldots, k_{64})$.
OUTPUT: 64-bit ciphertext block $C = (c_1, \ldots, c_{64})$.
1. Compute sixteen 48-bit round keys K_i from K.
2. $(L_0, R_0) \leftarrow IP(M)$.
3. for i from 1 to 16, compute L_i and R_i as above.
4. $B = (b_1, b_2, \ldots, b_{64}) \leftarrow (R_{16}, L_{16})$.
5. $C \leftarrow FP(B)$.

Decryption involves the same key and algorithm, but with subkeys applied to the internal rounds in the reverse order.



12.3 See further

Other modes of operation A few other modes of operation have been proposed. We can mention Output feedback (OFB), similar to CFB, but where the output of the encryption block function (rather than the ciphertext) serves as the feedback. Another related mode of operation is Counter (CTR), which encrypts successive values of a counter.

AES The Advanced Encryption Standard (AES) was introduced in 2002 in order to replace DES as the new standard for block ciphers. Unlike DES, where some details were kept secret, AES was selected using a competition where all the algorithms had to explain all the details and design choices.

Overview of AES: 128 bit message block represented as a 4 by 4 array of bytes. It applies 10 rounds of:

- S-box transformation

- ShiftRows
- MixColumns
- Add sub-key.

The first three operations are all fixed. The fourth one is the only use of the key. Again, the S-box substitutions are the key non-linear element. The rest helps diffuse uncertainty.

12.4 Exercises

Exercise 12.1. By hand, encrypt the following plaintext $M = 01110001$ (8 bits) using a Feistel cipher with 4 rounds and function

$$f(x_1,x_2,x_3,x_4) = x_1x_2 + x_2x_3 + x_3x_4 + x_4x_1 \mod 2.$$

For simplicity, we apply f to $R_{i-1} \oplus K_i$, so that the actual Feistel function is

$$f(r_1,r_2,r_3,r_4,k_1,k_2,k_3,k_4) = (r_1+k_1)(r_2+k_2)+(r_2+k_2)(r_3+k_3)+(r_3+k_3)(r_4+k_4)+(r_4+k_4)(r_1+k_1) \mod 2.$$

You can choose anything you want for the four 4-bit subkeys K_1,K_2,K_3,K_4.

Decrypt the ciphertext and verify that you get the right plaintext.

Exercise 12.2. A DES weak key is a key K such that $E_K(E_K(x)) = x$ for all x, i.e., defining an involution. Verify that the four keys in the table below are indeed weak keys. I give K in hexadecimal and the corresponding C_0 and D_0; note that the subkeys are easy to determine in those cases!

weak key (hexadecimal)	C_0	D_0
0101 0101 0101 0101	0…0	0…0
FEFE FEFE FEFE FEFE	1…1	1…1
1F1F 1F1F 0E0E 0E0E	0…0	1…1
E0E0 E0E0 F1F1 F1F1	1…1	0…0

Those actually are the only weak keys of DES.

13 Symmetric cryptography II: Cryptanalysis

13.1 Weaknesses of DES

Attacking DES Even in 1977 the key size (56 bits) was considered 'low'. The NSA 'encouraged' the dropping of key size from 128 to 56 bits at the time - could they mount a brute force attack even then? Proposed machines to break DES by brute force are:

- 1977. Diffie and Hellman outline a $20million machine to crack DES in 1 day.
- 1993. Wiener proposed a $1million machine to crack DES in 7 hours.
- 1997. RSA corp. sponsored a competition to crack DES. It was won by a program using downtime of home PCs connected to the internet. 1 PC would have taken 2285 years to search the key space. They had 78,000 PCs signed up by the time they found the key, after searching 1/4 of the key space.
- 1998. The EFF built a $250,000 machine that cracks DES in 2 days.
- 2006. A $10,000 machine (COPACOBANA) is built from off-the-shelf kit that cracks DES in less than a week.

Key size If our key size is 56 bits, there are 2^{56} possible keys and testing randomly we would expect to test about 2^{55} before we cracked it. Suppose we can test a billion (2^{30}) possible keys per second. There are 2^{25} seconds per year, so we test about 2^{55} keys per year.

The amount of time required to break a 128-bit key is very much greater. About

$$2^{127} = 170,141,183,460,469,231,731,687,303,715,884,105,728$$

possibilities must be checked. A device that checks a billion billion keys (2^{60}) per second would require about 10^{13} years to exhaust the key space. This is a thousand times longer than the age of the universe.

AES permits the use of 256-bit keys. Breaking a symmetric 256-bit key by brute force requires 2^{128} times more computational power than a 128-bit key. A device that could check a billion billion (10^{18}) AES keys per second would require about 10^{51} years to exhaust the 256-bit key space.

TripleDES and DES-X Two main variants of DES were introduced to increase the key space/size.

TripleDES: Take three DES keys K_1, K_2, K_3. Encrypt a block M as

$$C = e(d(e(M,K_1),K_2),K_3),$$

where e is the standard DES encryption, and d is the standard DES decryption. The key size is now $3 \times 56 = 168$ bits, but effective security is only ca. 112 bits due to meet-in-the-middle attacks. TripleDES is backwards compatible taking $K_1 = K_2 = K_3$.

DES-X: Take a DES key K, and two supplementary 64 bit keys K_1, K_2. Encrypt a block M as

$$C = K_2 + e(M + K_1, K).$$

The Key size is now 184 bits, but effectively ca. 119 bits. Since only one DES encryption is performed, this is much faster than TripleDES.

13.2 Cryptanalysis of block ciphers

Meet-in-the-middle Suppose we are using a k-bit cryptosystem with encryption function e and decryption function d. We decide to increase security by encrypting twice as follows.: choose two keys K_1 and K_2 each of k bits, and then do

$$C = e(e(M,K_1),K_2).$$

The effective security is not $2k$ bits. Suppose a plaintext/ciphertext pair M, C is publicly known. The attacker could compute a table of $e(M,K)$ for all 2^k possible keys K. Then she can compute $d(C,K')$ for all possible keys K', checking for a match:

$$e(M,K) = d(C,K').$$

This gives a pair of keys such that

$$e(e(M,K),K') = C.$$

The total encryptions/decryptions performed is 2^{k+1}. The space required is only enough to store 2^k keys and text snippets.

Linear cryptanalysis Linear cryptanalysis is quite involved, so we only give the basic ideas.

Suppose that it is possible to find a probabilistic linear relationship between a subset of plaintext bits and a subset of state bits immediately after the substitutions performed in the last round, In other words, there exists a subset of bits whose exclusive-or behaves in a non-random fashion (it takes the value 0 with probability bounded away from 1/2).

Now assume that an attacker has a large number of plaintext-ciphertext pairs for the same unknown key K (i.e. this is a known-plaintext attack). For each of the plaintext-ciphertext pairs, we begin to decrypt the ciphertext, using all possible candidate keys for the last round of the cipher. We keep track of when the linear relationship does hold for each key. At the end of this process, we hope that the

candidate key with the most biased count (furthest away from 1/2 of the number of pairs) contains the correct value for these bits.

Performing a linear attack involves computing a linear approximation of the S-boxes. Therefore, a secure S-box must have a high nonlinearity, defined as the distance to any linear or affine function.

Linear cryptanalysis was introduced by Matsui in 1994. He actually performed a linear attack on DES the same year. The linear cryptanalysis of DES is a known-plaintext attack using 2^{43} plaintext-ciphertext pairs, all of which are encrypted using the same unknown key. This actual attack did not have any impact on the security of DES due to the large number of pairs required. Nonetheless, AES was designed to resit linear cryptanalysis.

Differential cryptanalysis Differential cryptanalysis is similar to linear cryptanalysis in many respects. The main difference is that it involves comparing the xor of two inputs to the xor of the corresponding two outputs.

Differential cryptanalysis is a chosen-plaintext attack. The attacker has a large number of tuples (x, x^*, y, y^*) (input, input, output, output), where the value $x' = x \oplus x^*$ is fixed and x and x^* are encrypted using the same key.

Differential cryptanalysis was discovered in the late 1980s by Biham and Shamir. They noted that DES was resistant to differential cryptanalysis. In fact, IBM and the NSA knew about differential cryptanalysis since the mid-70s and hence designed DES to be robust against this sort of attack.

13.3 See further

Design of S-boxes Mathematically, an S-box is a function $\{0,1\}^n \to \{0,1\}^m$ for some m and n. It can then be viewed as a tuple of m Boolean functions $(s_1, \ldots, s_m)$, where $s_i : \{0,1\}^n \to \{0,1\}$ is a Boolean function.

The security of the block cipher depends on the S-box, and hence on its component Boolean functions. As such, different parameters of Boolean functions have been introduced: nonlinearity, balancedness, resiliency, correlation immunity, algebraic degree, etc.

13.4 Exercises

Exercise 13.1. Perform a meet-in-the-middle attack on the simple Feistel cipher of Exercise 12.1. (You can choose the solution of that exercise as your publicly known plaintext-ciphertext pair.)

14 RSA I: The system

14.1 Public-key cryptography

One-way functions A function f is said to be a one-way function if:

1. For all inputs x it is easy to compute $f(x)$.
2. Given some (random) y, it is hard to find an x such that $y = f(x)$.

By easy, we usually mean that it may be computed in time polynomial in $\log x$. Hardness is not so easy to define. At first, you may want inverting f to be an NP-hard problem. But note that NP-hardness is a worst case criterion; random cases, or the majority of cases may still be easy (Knapsack encryption suffers from that drawback). Also, hardness is time dependent: what was in feasible in 1977 is relatively easy today. The minimum requirement for hardness is that there is no known efficient algorithm that works on random instances.

One-way functions We will look at two candidate one-way functions: multiplication and exponentiation modulo a prime. In this lecture, we focus on multiplication.

Multiplication Given two integers a, b, it is easy (polynomial time) to compute the product ab.

Inverting (factoring) Given a large integer c, is it difficult to find integers a, b such that $ab = c$?

Not in general. If you pick a, b randomly then each is even with probability 1/2. So $c = a \cdot b$ is even with probability 3/4. You can often return $c = 2 \cdot (c/2)$. Better to pick two k bit primes, a, b. Now it is usually hard to invert. The decision problem (Does c have a factor less than m?) is not thought to be in P. It is in both NP and co-NP (given a prime factorisation of c you can check both membership and non-membership). The best known algorithm (general number field sieve), for an n bit number runs in time

$$\exp\{(1.92 + o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}\}.$$

Public-key cryptography A trapdoor function is a one-way function that can be easily inverted with a secret key. A public-key cryptosystem uses a trapdoor function as follows:

1. Alice encrypts her message M, using a one-way function f: $C = f(M)$.
2. Bob has a secret method of opening the trapdoor and inverting f.
3. Even if Eve intercepts $f(M)$, she cannot invert without knowing Bob's secret trapdoor.

Trapdoor functions were first proposed by James Ellis at GCHQ in 1969. The first public key cryptosystem was developed by Clifford Cocks, also at GCHQ, in 1973. Both developments remained secret until 1997.

RSA cryptosystem Rivest, Shamir and Adleman rediscovered, and published, Cocks' scheme in 1977. It relies on the difficulty of factoring for security.

Key generation

1. Bob chooses two large primes p and q.
2. Bob computes the public modulus $n = pq$, and the public exponent e co-prime to $\phi(n) = (p-1)(q-1)$. ($e < \phi(n)$.)
3. Bob publishes his public key (n, e).
4. His private key is the unique integer d such that $ed = 1 \mod \phi(n)$.

Encryption

1. Alice breaks her message into blocks so that each block M satisfies $M < n$.
2. Alice encrypts M as $C = M^e \mod n$.

Decryption

1. Bob can decrypt using $C^d = M \mod n$.

Proof that RSA works The correctness of decryption is nontrivial. It follows from this simple number theoretic result, which can be viewed as a simple case of Lagrange's theorem on the order of subgroups of a finite group.

Theorem 14.1. *For any integer n and any $a \in \mathbb{Z}_n^*$,*

$$a^{\phi(n)} = 1 \mod n.$$

Therefore, the decryption is correct:

$$C^d = M^{ed} = M^{ed + k\phi(n)} = M \mod n.$$

Computing time of RSA All the operations (key generation, encryption and decryption) are done in expected polynomial time.

Primality testing is very fast using randomised algorithms, and possible using a deterministic polynomial time algorithm. If we select random odd k-bit numbers, we expect to test polynomially many before we find a prime.

Computing $n = pq$ and $\phi(n) = (p-1)(q-1)$ is simple multiplication.

Finding e by random sampling and testing (Euclid's Algorithm), we expect to take polynomially many tests. We can also pick e to be a prime number, then we just need to check it does not divide $\phi(n)$. In practice, we often use the prime number

$$e = 65537 = 2^{16} + 1.$$

Finding d is again Euclid's Algorithm. Encryption and decryption can be achieved by repeated squaring.

Generating random prime numbers In practice, one uses the Miller-Rabin primality test. It is a yes-biased Monte Carlo algorithm, that is it always give an answer, and never returns "n is composite" when n is actually prime. The algorithm runs very fast. The probability of getting it wrong is exponentially small.

TLS Amongst the many applications of RSA, let us mention Transport Layer Security (TLS). This is the newer version of SSL, used as the protocol for secure websites (https). Here is a massively reduced summary:

1. Client contacts server and they agree to use the highest level cryptography suite they both know.
2. The client generates a random number and communicates it to the server using public key cryptography (RSA) and the server's public key.
3. The server and client both compute a secret master key from the random number.
4. The master key is used to communicate using symmetric cryptography (AES).

Why? Because symmetric cryptography is about 1000 times faster than RSA.

14.2 See further

The Rabin cryptosystem RSA is not the only publick-key cryptosystem based on the complexity of factoring. Another example is the Rabin cryptosystem, described below.

Key generation Let $n = pq$, where p and q are primes and $p, q = 3 \mod 4$. Then n is the public key, while (p, q) is the private key.

Encryption Let $x \in \mathbb{Z}_n^*$, then

$$e(x) = x^2 \mod n.$$

Decryption Let $y \in \mathbb{Z}_n^*$, then

$$d(x) = \sqrt{y} \mod n.$$

(The restriction $p, q = 3 \mod n$ can actually be omitted.)

The one main drawback of the Rabin cryptosystem is that encryption is not injective. Indeed, any ciphertext y has four possible square roots, and hence corresponds to four possible plaintexts! For instance, let $p = 7$, $q = 11$, $y = 23$. Then a bit of algebra shows that

$$10^2, 67^2, 32^2, 45^2 = 23 \mod 77.$$

14.3 Exercises

Exercise 14.1. Alice and Bob decide to communicate using RSA. Bob picks primes $p = 5$ and $q = 13$, and public exponent $e = 19$.

1. What information does Bob publish as his public key? Give the specific values from this example.
2. What information comprises Bob's private key? Compute it in this example.
3. What is the process for Alice to send a message to Bob? Alice first coverts each character into a number (as given in the character - number conversion table attached). She then encrypts each number using Bob's public information, and converts it back to a character to send to Bob. Illustrate how she would encrypt the message "Hell".
4. Explain the process for Bob to decrypt messages he receives and decrypt the message ':Uo'.
5. What would the encryption of the numbers 0, 1 and 64 be? Why aren't they used in the character conversion table?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
-	-	␣	⮐	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Q	R	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f	g	h	i	j
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	.	,	?	!
60	61	62	63	64															
:	;	’	⭲	-															

15 RSA II: Cryptanalysis

15.1 Vulnerabilities of RSA

General vulnerabilities Due to its popularity, RSA has been the subject of much research trying to break the cryptosystem.

Several schemes are known for attacking the system, which perform well if special conditions are met, for instance:

- Primes p and q are 'too close' (searching from $\sqrt{n}$ up to $(p+q)/2$ reveals a factor).
- If d is too small ($< n^{0.25}$), then d can be recovered from (n, e).
- Sending the same message to different people with the same e is insecure.

No scheme for cracking RSA in general, except by factoring n, is known.

Other issues Problem of non-random messages: If you only send a limited range of messages, Eve can try encrypting each to see which gives the intercepted cryptogram. For instance, if you always send 'Transfer X pounds into my account' or 'My 3-digit card security code is X'.

This is why Alice should always pad messages with random bits.

15.2 Factorisation algorithms

Trial division The naive factoring algorithm, called trial division: "check all numbers p up to $\sqrt{n}$ to see if $p \mid n-1$" runs in exponential time. Indeed, the size of the input is $\log n$, while the algorithm would run in $\Omega(\sqrt{n}) = \Omega(c^{\log n})$ for some constant c. There are no polynomial time algorithms known for factoring, but below we present three algorithms that can be much more efficient than the naive factoring algorithm.

Pollard's $p-1$ algorithm Pollard's $p-1$ algorithm has two inputs: the odd integer n to be factored, and a prescribed bound B.

Algorithm 1 Pollard $p-1$ factoring algorithm(n,B)

```
a ← 2
for j ← 2 to B do
    a ← a^j mod n
d ← gcd(a − 1, n)
if 1 < d < n then
    return d
else
    return "failure"
```

Here's a simple explanation of why this works. Suppose p is a prime divisor of n, and suppose that $p-1$ is B-smooth: $r \le B$ for every prime power $r \mid (p-1)$. That is, we assume that $p-1$ only has small factors in its factorisation. For example, the number $2,000$ is B-smooth for any $B \ge 125$, since it factors as

$$2000 = 2^4 \times 5^3 = 16 \times 125.$$

Theorem 15.1. *If p is a prime divisor of n, and $r \le B$ for every prime power $r \mid (p-1)$, then Pollard's $p-1$ algorithm(n,B) returns d, a nontrivial factor of n.*

Proof. The key idea in the proof is that $p \mid a-1$. Let us prove this now.

At the end of the for loop, we have

$$a = 2^{B!} \mod n.$$

Since $p \mid n$, we have

$$a = 2^{B!} \mod p.$$

Since all the terms in the factorisation of $p-1$ belong to $\{1,2,3,\dots,B\}$, we must have

$$p-1 \mid 1 \times 2 \times 3 \times \cdots \times B = B!.$$

Say $B! = \gamma(p-1)$. By Fermat's theorem, $a^{p-1} = 1 \mod p$, hence

$$a = 2^{B!} \mod p = \left(2^{p-1}\right)^{\gamma} \mod p = 1^{\gamma} \mod p = 1 \mod p.$$

Thus $p \mid a-1$.

We can now finish the proof. Since $p \mid n$ and $p \mid a-1$, we have $p \mid d$, where $d = \gcd(a-1,n)$. By definition, $d \mid n$ and, unless $a = 1$, $1 < d < n$; that is, d is a nontrivial divisor of n. □

The running time is clearly of the form $O(B\,\mathrm{poly}(\log n))$. So if B itself is $O(\mathrm{poly}(\log n))$, the $p-1$ algorithm runs in polynomial time. However, it may not work: if $p-1$ has a factor greater than B, then the algorithm may return a failure. (Note that if $B = \sqrt{n}$, then the $p-1$ algorithm is sure to work, but that's no improvement compared to the naive algorithm.)

Pollard's $p-1$ algorithm can then be thwarted by constructed $n = pq$ without small factors as follows. First, choose a large prime p_1 such that $p = 2p_1 + 1$ is also prime, and similarly a large prime q_1 such that $q = 2q_1 + 1$ is also prime. (That task is not actually onerous.) Then the RSA modulus $n = pq$ will be resistant to factorisation using Pollard's $p-1$ algorithm.

Pollard's rho algorithm Let p be the smallest prime divisor of n. Suppose there exist two integers $x, x' \in \mathbb{Z}_n$ such that $x \ne x'$ and $x = x' \mod p$. Then $p \le \gcd(x-x', n) < n$, hence the gcd is a nontrivial factor of n. To find such a collision, one can take a random set X of size $|X| \approx 1.17\sqrt{p}$: by the birthday paradox, we would find a collision with probability 50%. However, we would need to compute $\Theta(|X|^2) = \Theta(p) = \Theta(\sqrt{n})$ gcds.

The Pollard rho algorithm is a clever heuristic to find collisions without that many gcds.Let $f(x)$ be a polynomial with integer coefficients. A popular choice is

$$f(x) = x^2 + 1.$$

We will presume that the mapping $x \mapsto f(x) \mod p$ behaves like a random mapping. That way we can start with a particular value of x, say x_0 and search for a collision on its orbit $X = \{x_0, x_1, \ldots, x_k\}$, where

$$x_1 = f(x_0) \mod n, x_2 = f(x_1) = f^2(x_0) \mod n, \ldots, x_k = f^k(x_0) \mod n.$$

There must be a collision somewhere on the orbit. Indeed, the mapping $x \mapsto f(x) \mod n$ is a self-map of $\mathbb{Z}_n$, so there are i and j such that $f^i(x_0) \mod n = f^j(x_0) \mod n$. Therefore, there exist λ and π such that

$$x_\lambda = x_{\lambda+\pi} \mod p.$$

We say λ is the pre-period of x_0 and π is its period. If we draw the directed graph $D = (X \mod p, A)$ with arcs $A = \{(x_i \mod p, x_{i+1} \mod p)\}$, we get a rho-like shape, hence the name of the algorithm.

Let i' satisfy $i' \geq i$ and $i' = 0 \mod \pi$, then

$$x_{2i'} = x_{i'} \mod p.$$

Consider the sequences $B = (x_0, x_1, x_2, \ldots)$ and $C = (x_0, x_2, x_4, \ldots)$. We then have $b_{i'} = x_{i'}$ and $c_{i'} = x_{2i'} = b_{i'} \mod p$: we can find a collision simply by following those two sequences. We expect i' to be at most $\sqrt{p}$.

Algorithm 2 Pollard ρ factoring algorithm(n, x_1)

```
x ← x_0
x' ← f(x) mod n
p ← gcd(x − x', n)
while p = 1 do
    x ← f(x) mod n
    x' ← f^2(x') mod n
    p ← gcd(x − x', n)
if p < n then
    return p
else
    return "failure"
```

Note that there is a small chance of failure if the collision $x = x' \mod p$ actually satisfies $x = x'$. This occurs with probability roughly p/n, which is very small in practice. And in any case, if this occurs, then one can simply re-run the algorithm with a different initial point x_0.

Quadratic sieve algorithm Given x, y with $x^2 = y^2 \mod n$ and $x \neq \pm y \mod n$, we obtain

$$0 = x^2 - y^2 = (x - y)(x + y) \mod n,$$

and hence $n|(x - y)(x + y)$. But n neither divides $x + y$ nor $x - y$, thus $\gcd(x - y, n)$ is one of the prime factors of n.

The quadratic sieve algorithm tries to generate such a pair x, y. The key idea is to select a sequence of elements $x_i \in \mathbb{Z}_n$ and generate the corresponding sequence of values

$$q_i = x_i^2 \mod n.$$

We choose $x_i = \lceil \sqrt{n} \rceil + i$ (or some choose a double-indexed sequence $x_{i,j} = \left\lceil \sqrt{jn} \right\rceil + i$) because then $x_i^2 \mod n$ is a small number. In fact, we select a bound B and we only keep the q_i's that are B-smooth, so we can factor them efficiently.

Let $p_1, \ldots, p_k$ denote the k prime numbers less than or equal to B. We then obtain

$$\begin{aligned} q_1 = x_1^2 \mod n &= \prod_{j=1}^{k} p_j^{e_{1,j}} \\ q_2 = x_2^2 \mod n &= \prod_{j=1}^{k} p_1^{e_{2,j}} \\ &\vdots \\ q_l = x_l^2 \mod n &= \prod_{j=1}^{k} p_j^{e_{l,j}}. \end{aligned}$$

The next step is to find a subset of the $\{q_i\}$ whose product is a square (in $\mathbb{N}$). If we take the subset S, the product is

$$z = \prod_{i \in S} q_i = \prod_{j=1}^{k} p_j^{\sum_{i \in S} e_{i,j}}.$$

The number z is a square if and only if $\sum_{i \in S} e_{i,j}$ is an even number for all j. Finding an S that satisfies this property is actually easy! (This is based on linear algebra over the binary field GF(2), we'll look at that stuff in Lecture 23.) Suppose we find the right S, then we obtain

$$z = \left(\prod_{i \in S} x_i \right)^2 \mod n.$$

This gives us two square roots modulo n of z: $x^2 = y^2 = z \mod n$. This pair of square roots may not be valid (we may have $x = \pm y \mod n$), but if we use $l > k + 1$, we ensure to have multiple sets S and hence multiple pairs of square roots; heuristically this will yield at least one valid pair.

The running time of the quadratic sieve algorithm depends on B. A careful analysis of the number of B-smooth numbers yields a subexponential $B = O(c^{\sqrt{\log n \log\log n}})$ and an overall subexponential running time of

$$O\left(\exp((1 + o(1))\sqrt{\ln n \ln\ln n})\right).$$

State-of-the-art The current best algorithm for factoring is the so-called Galois Field Number Sieve. It recently (February 2020) managed to factor the number RSA-250: a 250-digit (829 bits) number $n = pq$. In fact, RSA has a challenge with a list of semiprimes ($n = pq$) of increasing length that people try to factor. The list goes up to a number that's 2048 bits long.

15.3 See further

Side-channel attacks Another type of attack on RSA and similar systems was discovered by Paul Kocher in 1995. He showed that it is possible to discover the decryption exponent by carefully timing the computation times for a series of decryptions (see [17, Section 6.2] for a review). Even though Kocher's timing attack can be thwarted, it was nonetheless a game changer as it opened the way for a new range of so-called side-channel attacks, whereby the attacker has access to other kinds of data such as timing, power consumption, memory usage, heat diffusion, etc.

Schor's algorithm Integer factorisation can be done in polynomial time... on a quantum computer. Schor's algorithm is beyond the scope of this course. However, the threat of quantum computers is taken very seriously. Indeed, NIST are holding a competition to standardise Post-Quantum Cryptography (PQC), i.e. public-key cryptosystems and signature schemes based on problems which are believed to be secure against quantum attacks. There are two main kinds of post-quantum cryptosystems left: those based on lattice problems (Lectures 19 and 20) and those based on error-correcting codes (Lectures 29 and 30).

15.4 Exercises

Exercise 15.1. Apply Pollard's $p-1$ algorithm for $n = 15770708441$ and $B = 175$ to obtain

$$n = 135979 \times 115979.$$

(Yes, you need a computer for this one.)

Exercise 15.2. Let $n = 7171$. With $x_0 = 1$, use Pollard's rho algorithm to find the factor 71 of n. (You should get $\gcd(x_{11} - x_{22}, n) = 71$.)

16 Discrete logarithm cryptography

16.1 Diffie-Hellman key exchange

The Diffie-Hellman key exchange (1976) is a mechanism to agree on a secret key by exchanging messages without revealing the actual key value. It is based on the difficulty of discrete logarithms.

If p is a prime, then we say $g \in \mathbb{Z}_p^*$ is primitive mod p (or a generator) if all the elements of $\mathbb{Z}_p^*$ are powers of g:

$$\{g, g^2 \mod p, g^3 \mod p, \ldots, g^{p-1} \mod p = 1\} = \{1, 2, \ldots, p-1\}.$$

It is easily shown that there are $\phi(p-1)$ primitive elements mod p, where $\phi(n)$ is Euler's totient function.

Here is the protocol.

1. Alice and Bob publicly agree a large prime p and a primitive element g.

2. Alice and Bob choose private keys a and b (between 2 and $p-2$).

3. Alice computes

$$A = g^a \mod p,$$

and sends A to Bob.

4. Bob computes

$$B = g^b \mod p,$$

and sends B to Alice.

5. Alice computes the shared secret key

$$K = B^a \mod p.$$

6. Bob computes K as

$$K = A^b \mod p.$$

Alice and Bob know the same K, since

$$A^b = (g^a)^b = g^{ab} = (g^b)^a = B^a.$$

Eve only knows A, B, p and g. If Eve could compute discrete logarithms easily, then she could obtain (for instance): $a = \log_g A$ and then $K = B^a \mod p$.

16.2 The ElGamal cryptosystem

Let p be a large prime number, and let $g \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathscr{M} = \mathbb{Z}_p^*$, $\mathscr{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$.

Key generation
1. Let $A = g^a \mod p$ for some a
2. Public key: (p, g, A)
3. Private key: a

Encryption Choose a secret random number $k \in \mathbb{Z}_{p-1}$, then

$$\begin{aligned} e(x,k) &= (y_1, y_2), \\ y_1 &= g^k \mod p \\ y_2 &= xA^k \mod p. \end{aligned}$$

Decryption

$$d(y_1, y_2) = y_2 (y_1^a)^{-1} \mod p.$$

Again, decryption is correct since:

$$d(y_1, y_2) = y_2 (y_1^a)^{-1} = xA^k (g^{ak})^{-1} = x \mod p.$$

Here is an example. Let $p = 2579$ and $g = 2$; g is a primitive element mod p. Let $a = 765$, then

$$A = g^a \mod p = 2^{765} \mod 2579 = 949.$$

Now suppose Alice wishes to send the message $x = 1299$. She randomly chooses $k = 853$, then

$$\begin{aligned} y_1 &= g^k \mod p = 2^{853} \mod 2579 = 435, \\ y_2 &= xA^k \mod p = 1299 \times 949^{853} \mod 2579 = 2396. \end{aligned}$$

When Bob receives the ciphertext $(435, 2396)$, he computes

$$x = y_2 (y_1^a)^{-1} \mod 2579 = 2396 \times (435^{765})^{-1} \mod 2579 = 1299.$$

16.3 Some algorithms for discrete logarithm

Naive discrete log algorithms The problem is: given a prime p, a primitive element g mod p, and $A \in \mathbb{Z}_p^*$, find a such that $A = g^a$. The first naive algorithm is exhaustive search: compute the powers of g in sequence $(1, g, g^2, g^3, \ldots)$ until we hit $g^a = A$. This will take $\Omega(p)$ time (for simplicity we assume operations mod p are done in constant time), which is exponential in the size of p. Another approach is to precompute all the possible values g^i and then sort the list (i, g^i) according to their second coordinates. The precomputation still takes $O(p)$, but once that is done, finding A on the list only takes $O(\log p)$ thanks to search by dichotomy.

Shank's algorithm Shank's algorithm is a time-memory tradeoff between the two naive approaches we've mentioned above.

Algorithm 3 Shank's algorithm$(p-1, g, A)$

$m \leftarrow \lceil \sqrt{p-1} \rceil$
for $j \leftarrow 0$ to $m-1$ **do**
    Compute g^{mj}
$L_1 \leftarrow$ the list of pairs (j, g^{mj}) sorted according to their second coordinates
for $i \leftarrow 0$ to $m-1$ **do**
    Compute Ag^{-i}
$L_2 \leftarrow$ the list of pairs (i, Ag^{-i}) sorted according to their second coordinates
Find two pairs $(j, B) \in L_1$ and $(i, B) \in L_2$ with the same second coordinate
return $mj + i \mod p - 1$

Indeed it works:

$$Ag^{-i} = g^{mj} \iff A = g^{mj+i}.$$

Shank's algorithm is also called Giant step-baby step algorithm. Indeed, the first few lines of the algorithm that determine L_1 are precomputations. They are the giant steps: we take steps of size $\approx \sqrt{p}$. After that, computing L_2 is the baby steps (of size step 1).

Pollard's rho algorithm We have already seen Pollard's rho algorithm for factoring. The same basic idea can be adapted to compute the discrete logarithm as well. Once again, we form a sequence $x_0, x_1, \ldots$ by iterating a random-looking function f. We still aim to find a collision of the form $x_i = x_{2i}$.

First, partition $\mathbb{Z}_p^*$ into three sets S_1, S_2, S_3 of roughly equal size. We work on triples $(x,s,t) \in \mathbb{Z}_p^* \times \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ and we define

$$f(x,s,t) = \begin{cases} (Ax, s, t+1) & \text{if } x \in S_1 \\ (x^2, 2s, 2t) & \text{if } x \in S_2 \\ (gx, s+1, t) & \text{if } x \in S_3. \end{cases}$$

We furthermore enforce that our triples have the property

$$x = g^s A^t.$$

Note that f preserves that property. So we only need to start with a triple satisfying that property, say $(1,0,0)$. So we define

$$(x_i, s_i, t_i) = \begin{cases} (1,0,0) & \text{if } i = 0 \\ f(x_{i-1}, s_{i-1}, t_{i-1}) & \text{otherwise.} \end{cases}$$

(Note: we need to make sure $1 \notin S_2$, otherwise we would get stuck at $(1,0,0)$.)

The following lemma justifies looking for a collision.

Lemma 16.1. *Suppose $x_i = x_{2i}$ and $\gcd(t_{2i} - t_i, p-1) = 1$, then*

$$a = \log_g A = (s_i - s_{2i})(t_{2i} - t_i)^{-1} \mod p-1.$$

Proof. Under the hypothesis, we have

$$\begin{aligned} g^{s_{2i}} A^{t_{2i}} &= g^{s_i} A^{t_i} \\ g^{s_{2i} + a t_{2i}} &= g^{s_i + a t_i} \\ s_{2i} + a t_{2i} &= s_i + a t_i \mod p-1 \\ a &= (s_{2i} - s_i)(t_i - t_{2i})^{-1} \mod p-1. \end{aligned}$$

□

We thus obtain the algorithm.

Algorithm 4 Pollard ρ discrete log algorithm$(p-1, g, A)$

```
Partition Z_p^* = S_1 ∪ S_2 ∪ S_3
(x, s, t) ← (1, 0, 0)
(x', s', t') ← f(x, s, t)
while x ≠ x' do
    (x, s, t) ← f(x, s, t)
    (x', s', t') ← f^2(x', s', t')
if gcd(t' − t, p − 1) = 1 then
    return (s − s')(t' − t)^{-1} mod p − 1
else
    return "failure"
```

If the collision is not valid, i.e. $\gcd(t'-t,n)=d>1$, then the situation is not too bad. In fact, there are d possible solutions for a, so if d is not too large one can determine the d possible solutions and try them all out to find the correct one.

Again, under reasonable assumptions about the randomness of the function f, we expect to compute discrete logarithms in $O(\sqrt{p})$ iterations.

16.4 See further

More algorithms for discrete logarithm Again, there are many more algorithms for discrete logarithms. We briefly mention Pohlig-Hellman and the Index Calculus algorithm; both are reviewed in [16]. The latter runs in subexponential running time. The same team that holds the record for factoring the largest semiprime also recently broke the record for discrete log. In fact, the best algorithms for factoring and discrete log run in similar times.

Elliptic curves In abstract terms, the discrete logarithm problem can be applied to any finite cyclic group. A finite cyclic group is a pair $(G,\cdot)$ where G is a finite set (say of order $|G|=n$) and $\cdot$ is a binary (product) operation on G such that $G=\{g^0=1,g^1=g,\ldots,g^{n-2}\}$ for some $g\in G$. (The element g is then called the generator.) So far, we have focused on the cyclic group $(\mathbb{Z}_p^*,\times)$. But there are many other cyclic groups out there.

Even though two cyclic groups of the same order are algebraically equivalent (isomorphic), the way they are described makes the discrete logarithm problem more or less difficult. Think of the cyclic group $(\mathbb{Z}_{p-1},+)$ with generator 1. Even though this group is equivalent to $(\mathbb{Z}_p^*,\times)$, its discrete log problem is trivial!

Elliptic-curve cryptography (ECC) is based on the discrete logarithm for cyclic groups defined on points of elliptic curves, which is harder than the discrete log for $(\mathbb{Z}_p^*,\times)$. The algebra used there is very involved so we will not have time to talk about it. Note that ECC uses significantly smaller keys than RSA for the same level of security (e.g. 108 bits for ECC against 512 bits for RSA).

16.5 Exercises

Exercise 16.1. Go through the following example of Diffie-Hellman key exchange:

$$\begin{aligned} p&=17,\\ g&=3,\\ a&=5,\\ b&=11. \end{aligned}$$

Compute all the messages and the agreed key.

Exercise 16.2. Here is a naive attempt at key exchange, where Alice wants to transmit an n-bit key $K=(k_1,\ldots,k_n)$ to Bob.

1. Alice generates a random string $a=(a_1,\ldots,a_n)$ and sends
$$A=K\oplus a.$$
2. Bob receives A, generates another random string $b=(b_1,\ldots b_n)$, and sends back
$$B=A\oplus b.$$
3. Alice then "removes" the string a and sends back
$$A'=B\oplus a=K\oplus b.$$
4. Bob obtains the key by computing
$$K=A'\oplus b.$$

As you can see, the key K is never sent in plaintext. However, this scheme is insecure! Why?

17 Hash functions

17.1 Security of hash functions

Hash function A hash function is simply a function that takes inputs of some length and compresses them into short, fixed-length outputs. The classic use of hash functions is in data structures, where they can be used to build hash tables that enable constant-time lookup when storing a set of elements. A good hash function for this purpose is one that yields few collisions, where a collision is a pair x, x' such that $H(x) = H(x')$.

Collision resistance Informally, a function H is collision resistant if is intractable (say infeasible with a probabilistic polynomial time algorithm) to find a collision in H. We will only be interested in hash functions whose domain is larger than their range. Therefore, collisions must exist, but such collisions are hard to find.

Cryptographic hash functions used in practice generally have a fixed output length and are unkeyed, meaning that the hash function is just a function

$$H : \{0,1\}^* \to \{0,1\}^l.$$

This is problematic in practice, as there is always a constant-time algorithm to find a collision (e.g. list all possible inputs of length $l+1$, then you must find a collision there). Nonetheless, the unkeyed hash functions used in day-to-day life are collision resistant for all practical purposes.

Weaker notions of security In some applications, we only need to rely on weaker security requirements, for instance:

Second-preimage resistance Given x, it is hard to find an $x' \neq x$ such that $H(x') = H(x)$.

Preimage resistance Given y, it is hard to find an x such that $H(x) = y$. (This is equivalent to: H is a one-way function.)

17.2 Constructions

Hash functions H are often constructed by first designing a collision resistant compression function h handling fixed-length inputs, and then using domain extension to handle arbitrary-length inputs. We now look at a way of constructing the compression function using block ciphers, and a way of extending the domain by applying the compression function recursively.

Hash functions from block ciphers Let $e(k,x)$ be a block cipher with n-bit key length and l-bit block length: for a plaintext block $x \in \{0,1\}^l$ and a key $k \in \{0,1\}^n$, it returns $e(x,k)$. We can then define the compression function $h : \{0,1\}^{n+l} \to \{0,1\}^l$ as

$$h(x,k) = e(x,k) \oplus x.$$

This is called the Davies-Meyer construction.

Collision resistance of the compression function can be proved for so-called ideal ciphers, which are a much stronger notion of security than what we needed for the design of block ciphers. In particular, in an ideal cipher, the permutations $e(\cdot,k)$ and $e(\cdot,k')$ must behave independently even if k and k' only differ by one bit. Also, in a ideal-cipher there can be no keys k where $e(\cdot,k)$ should behave randomly even with the knowledge of k. As such, block ciphers used for encryption, e.g. DES and AES are not suitable for the Davies-Meyer construction.

The Merkle-Damgård transform The Merkle-Damgård transform is a common approach for domain extension while maintaining collision resistance. It is used in practice in MD5 and in the family of hash functions SHA.

For simplicity, suppose $h : \{0,1\}^{2n} \to \{0,1\}^n$ is a collision resistant compression function (in fact, we can work with $\{0,1\}^{n+m} \to \{0,1\}^n$ for any $m \geq 1$). We will now construct a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^n$. Again for simplicity we will assume that the input x has length $L = |x| < 2^n$.

The transform goes as follows.

1. Split x into $B = \lceil L/n \rceil$ blocks of length n, padding x with zeros if necessary, to obtain $x_1, \ldots, x_B$.
2. Set $x_{B+1} = L$, encoded in binary as an n-bit string.
3. Set $z_0 = 0^n$ (the initialisation vector IV).
4. For $i = 1, \ldots, B+1$, compute
$$z_i = h(z_{i-1}, x_i).$$
5. Return $H(x) = z_{B+1}$.

17.3 Applications

Fingerprinting and deduplication When using a hash function H, the hash (or digest) of a file serves as a unique identifier for that file. The hash function $H(x)$ of the file x is like a fingerprint, and one can check whether two files are equal by comparing their digests. This idea can be used in different ways.

Virus fingerprinting Virus scanners identify viruses and block or quarantine them. One of the most basic steps for this is to store a database containing the hashes of known viruses and then to look up the hash of a downloaded file in this database.

Deduplication Data deduplication is used to eliminate duplicate copies of data, especially in the context of cloud storage where multiple users rely on a single cloud service to store their data. The key observation here is that if multiple users wish to store the same file, then the file only needs to be stored once and need not be uploaded separately by each user.

Password hashing One of the most common and important uses of hash functions is for password protection. Instead of storing the passwords in the clear, only the hash of the password is stored instead. That is, the hard drive stores the value $hpw = H(pw)$ in a password file; later, when the user enters their password pw, the system checks whether $H(pw) = hpw$ before granting access.

If the password is chosen from a relatively small space of possibilities (e.g. a dictionary of English words, with only less than 100,000 words), then an attacker could enumerate all possible passwords and check their hash. We would like to claim that an attacker can do no better than this. However, preimage resistance only guarantees that $H(x)$ is hard to invert when x is chosen uniformly from a large domain like $\{0,1\}^n$. It says nothing about the hardness of inverting H when x is drawn according to another distribution (or from a smaller domain altogether). Moreover, preimage resistance says nothing about the concrete time it takes to find a preimage: if there is an algorithm that finds a collision in time $2^{n/2}$ (still exponential!), then a 30-bit password would be cracked in 2^{15} time.

Mitigation of attacks against password hashing One technique to mitigate the threat of password cracking is the use of "slow" has functions, or to slow down existing hash functions by using multiple operations (i.e. computing $H^I(pw)$ for some $I \gg 1$). this slows down legitimate users by a factor of I, which is OK if I is not too high (say, $I = 10^3$). On the other hand, it has a significant impact on an adversary attempting to crack thousands of passwords at once.

A second method is to introduce a salt. When a user registers their passwords, the laptop/server will generate a long random value s (a "salt") unique to that user, and store $(s, hpww = H(s, pw))$ instead of merely storing $H(pw)$ as before. Since s is unknown to the adversary in advance, preprocessing is

ineffective and the best they can do is wait to obtain the password file and perform the exhaustive search as before.

17.4 See further

Attacks against hash functions The main point of an attack is to find a collision. If $H : \{0,1\}^* \to \{0,1\}^l$, then the naive algorithm of trying $H(x)$ for all $x \in \{0,1\}^{l+1}$ is sure to find a collision. But one can do better by using the birthday paradox. Indeed, it can be shown that for $X = \{x_1, \ldots, x_q\} \subseteq \{0,1\}^l$ chosen uniformly at random, the probability of finding a collision is roughly 1/2 for $q = \Theta(2^{l/2})$. That simple attack uses a lot of memory but can be adapted to use much less space (the so-called Small-space birthday attacks).

There are other attacks aimed at inverting one-way function (to attack against preimage resistance), offering clever tradeoffs between time and memory.

Secure Hash Algorithm The Secure Hash Algorithm (SHA) refers to a series of cryptographic hash functions standardised by NIST. Perhaps the most well-known is SHA-1, introduced in 1995, and replacing the insecure SHA-0. It was then superseded by SHA-2 (which includes SHA-256 and SHA-512, referring to the respective output lengths). All those hash functions use the Davies-Meyer construction and the Merkle-Damgård transform. Note that the block ciphers were designed ad hoc for the SHA hash functions.

The new standard is now SHA-3 (Keccak), which was standardised in 2015. Its design is significantly different from previous SHA functions. Indeed, it does not use Davies-Meyer or Merkle-Damgård, but instead uses an unkeyed permutation to build the compression function and then uses the so-called sponge construction for domain extension.

17.5 Exercises

Exercise 17.1. We have looked at weak keys of DES in Exercise 12.2. It can be shown that if the key K is weak, then we can easily find fixed points, i.e. x such that $e(x,K) = x$. Given that fact, show that it is easy to find collisions for the Davies-Meyer construction using DES.

Exercise 17.2. Let us evaluate the performance of a birthday attack for collision. Suppose the compression function h behaves randomly for inputs of size l (equal to the output).

1. Choose q elements $x_1, \ldots, x_q \in \{0,1\}^l$ uniformly at random. Show that the probability of finding a collision is then equal to
$$P = 1 - \left(1 - \frac{1}{2^l}\right)\left(1 - \frac{2}{2^l}\right)\ldots\left(1 - \frac{q-1}{2^l}\right).$$

2. Obviously, we are looking to have $q \ll 2^l$. By using the approximation $1 - \delta \approx e^{-\delta}$ for small $\delta > 0$, approximate P to
$$P \approx e^{-\frac{q^2}{2^{l+1}}}.$$

3. Say we are aiming at a probability of success of $P = 1/2$, then show that we only need
$$q \approx \sqrt{2\ln 2} \cdot 2^{l/2}.$$

Exercise 17.3. MD5 is an old hash function that was proven insecure in 2004. Nowadays, any laptop can implement an attack finding a collision for MD5 in a few seconds (we can even ask for meaningful collisions) However, MD5 is still used by some legacy code to this day. Have a look around and see if you can spot usage of MD5.

Similarly, see where SHA-256 is being used instead of SHA-3.

18 Signature schemes

18.1 Introduction to digital signatures

Man-in-the-middle Suppose Alice and Bob are using Diffie-Hellman to exchange a key. Suppose Eve acts as the "man-in-the-middle", i.e. she intercepts the messages between Alice and Bob (unbeknownst to either of them). Then Eve can fool them as follows.

1. Eve forms her own secret key e, and public key $E = g^e \mod p$.
2. Eve intercepts $A = g^a \mod p$, and sends E on to Bob.
3. Bob responds with $B = g^b \mod p$, which Eve intercepts and sends E back to Alice.
4. Eve computes $K_A = A^e = g^{ae} \mod p$ to communicate with Alice and $K_B = B^e = g^{eb} \mod p$ to communicate with Bob.
5. Alice computes $K_A = E^a = g^{ae} \mod p$.
6. Bob computes $K_B = g^{be} \mod p$.

That way, Eve can continue to intercept messages, decrypt, re-encrypt and forward without Alice or Bob realising it. Eve can even alter messages without Alice or Bob being aware.

Digital signatures A digital signature is an addendum to a digital document or message to verify who wrote or sent it. If M is a message Alice wants to send to Bob, she creates a signature S, and sends the pair (M,S) to Bob. Bob must have some way of verifying that Alice created the signature and that the message has not been altered since she sent it.

A digital signature should provide the same guarantees as a traditional hand-written signature. Namely:

Unforgeability Only Alice should be able to sign her name to a message.

Undeniability Alice should not be able to later deny that she signed a signed message.

Authentication The signature should confirm that the contents of the message are as intended.

Clearly S must be a function of M and of some private key d_A known only to Alice. $S = f(M, d_A)$

18.2 Some digital signature schemes

Public key based schemes Any public key cryptosystem in which encryption and decryption are commutative, i.e. if

$$e(d(M)) = d(e(M)) = M,$$

can be used as a signature scheme.

RSA signature: Recall Alice has a public key (n,e) and private key d, such that

$$C^d = M^{ed} = M \mod n.$$

1. Alice creates the signature by 'decrypting' her message
$$S = M^d \mod n.$$
2. Alice send the pair (M,S) to Bob.
3. Bob 'encrypts' S, and verifies that
$$S^e = M^{de} = M \mod n.$$

Being able to sign general messages requires knowledge of the secret key d, so Bob trusts the message is from Alice.

Example Alice's public key is $(n = 65, e = 19)$. Her private key is $d = 43$. Alice sends the message "Hi" using conversion from Exercise 14.1:

$$\text{Hi} \mapsto M = (11, 38)$$
$$S = (11^{43} \mod 65, 38^{43} \mod 65) = (41, 12).$$

Bob receives the message/signature pair $((11,38),(41,12))$. He converts $(11,38)$ into "Hi". He checks S by checking $(41^{19} \mod 65, 12^{19} \mod 65) = (11,38)$. It does - he knows only someone with knowledge of Alice's private key could have created the message! I.e. it is from Alice.

Problems Firstly, Eve can sign random messages:

1. Eve chooses any $R < n$.
2. Eve computes $M = R^e \mod n$.
3. Eve can send (M,R) as a valid message/signature pair from Alice.

Secondly, Eve may be able to dupe Alice into signing unseen messages:

1. Eve chooses $R < n$ and computes $R^{-1} \mod n$.
2. Eve asks Alice to sign $M_1 = MR \mod n$ and $M_2 = R^{-1} \mod n$.
3. Eve can use
$$S_1 S_2 = M_1^d M_2^d = M^d R^d R^{-d} = M^d \mod n,$$
to sign M, which Alice has never seen.

18.3 Hashing and signing

'Hash then sign' schemes

Setup Alice and Bob need to setup the following:

- Alice has a public key (e.g. (n,e)), and private key (e.g. d);
- Alice and Bob agree on some public hash function h to use.

Signing To sign a message M, Alice:

1. computes the hash of the message $H = h(M)$;
2. uses her private key to sign the hashed message, e.g. $S = H^d$;
3. sends the pair (M,S) to Bob.

Verification Bob verifies that the signature is correct as follows:

1. Bob computes the hash of the message $H = h(M)$;
2. Bob checks that the signature matches the hash, i.e. $S^e = H$.

Much quicker than signing, transmitting and checking the full message.

Signed, encrypted messages So far all signed messages have been sent in the clear: $(M, S(h(M))$. To send a secure message Alice may:

1. Compute the hash H of the message M.
2. Compute the signature S for H using her private key.
3. Encrypt the full message (M, S) using Bob's public key, obtaining C.
4. Send C to Bob.

To verify, Bob then:

1. Decrypts C to obtain (M, S).
2. Computes the hash H of M.
3. Encrypts the signature S using Alice's public key, obtaining H'.
4. Checks that $H' = H$.

18.4 Certificate Authorities and chains of trust

Trusted Authorities All this can still be foiled if Eve intercepts every message from the beginning, and forges replies. We need a method of knowing we really have Bob's public key, and not one Eve has advertised falsely as Bob's.

This is overcome by have a trusted authority, or certificate authority (CA), who verifies people's identities and public keys, and signs a certificate to confirm that they are who they say they are.

For example, imagine Bob wishes to set up an internet shop. He generates private and public keys, and approaches a CA (e.g. Verisign). The CA checks him out and signs a certificate saying he is Bob and his public key is correct. If Alice finds Bob's shop and wishes to send a secure message, she can check that the public key advertised does belong to Bob, because it is signed by the CA, and Alice already knows the CAs public key.

Browsers come pre-configured with the public keys of major certificate authorities. You have to trust the CA to do its homework and check the identity. In 2001 someone convinced Verisign that they were a representative of Microsoft, and got a certificate signed by Verisign in the name of Microsoft Corporation. It later transpired that they were unconnected, and Microsoft and Verisign acted quickly to try and stop them.

Chains of trust How do I know to trust the CA? There are a few major CAs whose details are built into modern web browsers. You can accept others, if you trust them. Other CAs have their details certified by one of the major ones. Corporations are certified by a CA. You have to trust the chain of certifications, up to the major CA, in order to trust the website.

18.5 See further

Discrete logarithm based signatures The first signature scheme based on discrete logarithm is the ElGamal scheme. This was modifiesd by Schnorr, and in turn this became the Digital Signature Algorithm (DSA), developed by the NSA behind closed doors and proposed in 1991. Similarly to DES, it was based on a relatively small modulus p of 512 bits, leading to criticism. NIST then allowed for variable modulus sizes. The DSA was then superseded in 2000 by the Elliptic Curve Discrete Signature Algorithm (ECDSA), which is similar to DSA but as its name suggests, is based on the discrete logarithm problem on elliptic curves.

Lamport's scheme The Lamport signature scheme is based on hash functions. It is one-time-secure, that is it is secure as long as the private key is used to sign only a single message. Here is how it works with a cryptographic hash function H.

Suppose Alice wants to sign the l-bit long message $m = m_1 \dots m_l$. She first chooses $2l$ words $x_{0,1}, \dots, x_{0,l}; x_{1,1}, \dots, x_{1,l} \in \{0,1\}^n$ at random. She then computes their images $y_{i,j} = H(x_{i,j})$. She keeps the $x_{i,j}$ secret and releases the $y_{i,j}$ in public.

Then the signature of $m = m_1 \dots m_l$ is $\sigma = (\sigma_1, \dots, \sigma_l) = (x_{m_1,1}, \dots, x_{m_l,l})$. Bob verifies that it is genuine by verifying that $H(\sigma_j) = y_{j,m_j}$ for all j.

18.6 Exercises

Exercise 18.1. Bob has RSA public key $(n,e) = (65,19)$. 'Bob' sends you the message/signature pair $(M,S) = (Yes, hoS)$. Verify whether this message is correctly signed by Bob or not. Note - you should treat each character as a separate message, i.e. $(M,S) = (M_1M_2M_3, S_1S_2S_3)$, $M_1 = Y$, $S_1 = h$, etc.

Exercise 18.2. Generate a valid message/signature pair that appears to be signed by Bob without using his private key. (The exact message may be out of your control.)

Exercise 18.3. Generate a valid signature for the messages 'hAw' and 'H' using Bob's private key $(d = 43)$.

Exercise 18.4. Assume you have tricked Bob into providing the signatures in Exercise 18.3. Use these two signatures to generate a valid signature for the message 'Now', and verify that it is correct.

19 Lattice-based cryptography I

Lattice-based cryptography is one of the two main branches of post-quantum cryptography, the other being code-based cryptography (which will be covered in Lectures 29 and 30). Lattice-based cryptography covers all cryptosystems that rely on the hardness of lattice problems for their security.

19.1 Post-Quantum Cryptography

Quantum algorithms A quantum algorithm is any algorithm that utilises quantum computation in solving a problem, and requires a quantum computer to be run.
There are two quantum algorithms that are important to cryptography, Grover's algorithm and Shor's algorithm. Grover's algorithm is used as a search algorithm, and can find an item in a list in $O(\sqrt{N})$ queries, compared to standard classical search algorithms that take $O(N)$ queries. Shor's algorithm is a polynomial time algorithm for solving integer factorisation, compared to the current classical algorithm which has an exponential run time.

What does this mean for cryptography? A brute force search for an AES(-256) key currently takes roughly 2^{256} time, as there are 2^{256} keys. Using Grover's algorithm for the brute force search has a run time of $\sqrt{2^{256}} = 2^{128}$. This can be easily remedied by doubling the size of the key. For virtually all public key cryptosystems (RSA, Diffie-Hellman, Elliptic Curve), Shor's algorithm can break the cryptosystem in polynomial time, the only solution to this is to create new cryptosystems that are secure against quantum attacks.
A Post-Quantum cryptosystem is any cryptosystem that runs on a classical computer and is secure against an attacker who has access to a quantum computer. Quantum cryptosystems, where the cryptosystem requires a quantum computer to run and is secure against an attacker who has access to a quantum computer, do exist and are in use today, albeit in a limited capacity.

19.2 Learning With Errors (LWE)

LWE Learning with Errors is parametrised by two integers, n and q, and an arbitrary error distribution over $\mathbb{Z}_q$, χ. Quite often χ is based on the Gaussian distribution or the Binomial distribution. Given

a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ we create a single Learning With Errors instance by choosing a random vector $\mathbf{a}$ from $\mathbb{Z}_q^n$, sampling a random integer from our error distribution χ and calculating $b = \mathbf{a}^\top \cdot \mathbf{s} + e$, we then output the pair $(\mathbf{a}, b)$.

For example with $n = 4$, $q = 17$, a secret vector of $\mathbf{s} = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix}$, a random vector $\mathbf{a}$ of $\begin{pmatrix} 0 \\ 12 \\ 16 \\ 13 \end{pmatrix}$ and a random error e of -1, we calculate

$$b = \mathbf{a}^\top \cdot \mathbf{s} + e = (10 \cdot 0 + 12 \cdot 1 + 16 \cdot 2 + 13 \cdot 3) - 1 = 82 \bmod 17 = 14,$$

giving us a pair $([10, 12, 16, 13], 14)$. As we often want to calculate multiple pairs using the same secret we often write this as $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, where

$$\mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{Z}_q^m, \ \mathbf{A} = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_m \end{pmatrix} \in \mathbb{Z}_q^{m \times n}, \ \mathbf{s} \in \mathbb{Z}_q^n \text{ and } \mathbf{e} = \begin{pmatrix} e_1 \\ \vdots \\ e_m \end{pmatrix} \in \mathbb{Z}_q^m$$

with each e_i being drawn independently at random from χ. This gives us m equations, where each one satisfies $b_i = \mathbf{a}_i^\top \cdot \mathbf{s} + e_i$.

LWE problems There are two LWE problems, Decision-LWE and Search-LWE.
Decision-LWE.
Instance: A set of m pairs $(\mathbf{a}_i, b_i)$, where $\mathbf{a}_i \in \mathbb{Z}_q^n, b_i \in \mathbb{Z}_q$.
Question: Are the pairs LWE instances or are they random?

Search-LWE.
Instance: A set of m pairs $(\mathbf{a}_i, b_i)$, where $\mathbf{a}_i \in \mathbb{Z}_q^n, b_i \in \mathbb{Z}_q$.
Question: Find the secret vector s that was used to create the pairs.

Solving either of these problems (either classically or using a quantum computer) is at least as hard as solving several NP-hard lattice problems using a quantum computer.

19.3 The Learning With Errors Cryptosystem

Key Generation Key Generation is done by creating a set of LWE equations, with $(\mathbf{A}, \mathbf{b})$ being the public key and $\mathbf{s}$ being the private key.

1. Choose $\mathbf{A}$ uniformly at random from $\mathbb{Z}_q^{m \times n}$.
2. Choose $\mathbf{s}$ uniformly at random from $\mathbb{Z}_q^n$.
3. Sample $\mathbf{e}$ from χ^m.
4. Calculate $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$
5. Return $(\mathbf{A}, \mathbf{b})$ as the public key.
6. Return $\mathbf{s}$ as the private key.

Encryption Encryption is done by turning the public key into two more sets of equations using a new secret vector, we can only encrypt one bit, pt, at a time.

1. Choose $\mathbf{r}$ uniformly at random from $\mathbb{Z}_2^m$.
2. Sample $\mathbf{e}'$ from χ^n.

3. Sample e'' from χ.
4. Calculate $\mathbf{a}'^{\top} = \mathbf{r}^{\top} \cdot \mathbf{A} + \mathbf{e}'^{\top}$
5. Calculate $b' = \mathbf{r}^{\top} \cdot \mathbf{b} + e'' + pt \cdot \frac{q}{2}$
6. Return $(\mathbf{a}', b')$ as the ciphertext.

Decryption Decryption is done by using the private key to calculate $\mathbf{a}'^{\top} \cdot \mathbf{s} \approx b'$

1. $v = \mathbf{a}'^{\top} \cdot \mathbf{s}$
2. $v' = b'$
3. $m' = v' - v$
4. If m' is closer to 0 than $\frac{q}{2} \pmod q$ then $pt = 0$ else $pt = 1$.

Correctness In order to prove that the cryptosystem is correct, we need to show that $m' \approx pt \cdot \frac{q}{2}$.

$$\begin{aligned} m' - m\frac{q}{2} &= v' - v - m\frac{q}{2} \\ &= b' - \mathbf{a}'^{\top} \cdot \mathbf{s} \\ &= \mathbf{r}'^{\top} \cdot \mathbf{b} + e'' - \mathbf{a}'^{\top} \cdot \mathbf{s} \\ &= \mathbf{r}'^{\top} \cdot (\mathbf{A} \cdot \mathbf{s} + \mathbf{e}) + e'' - (\mathbf{r}^{\top} \cdot \mathbf{A} + \mathbf{e}'^{\top}) \cdot \mathbf{s} \\ &= \mathbf{r}'^{\top} \cdot \mathbf{A} \cdot \mathbf{s} + \mathbf{r}'^{\top} \mathbf{e} + e'' - \mathbf{r}^{\top} \cdot \mathbf{A}\mathbf{s} - \mathbf{e}'^{\top} \cdot \mathbf{s} \\ &= \mathbf{r}'^{\top} \cdot \mathbf{e} + e'' - \mathbf{e}'^{\top} \cdot \mathbf{s} \end{aligned}$$

The cryptosystem is correct when $|\mathbf{r}'^{\top} \cdot \mathbf{e} + e'' - \mathbf{e}'^{\top} \cdot \mathbf{s}| < \frac{q}{4}$.
In order to ensure that the system is 'hard' to break, $q \approx n^2$ and $m \approx n\log(q)$. Typically we use parameters $n \approx 8$, $q \approx 2^{13}$, $m \approx 512$, with $\mathbf{s}, \mathbf{r}$ being taken from the range $[-5, \ldots, 5]$ rather than from $[0, \ldots, q]$.

Security Like the LWE problem, the LWE cryptosystem is as hard to break as solving some NP-hard lattice problems. In the next section we discuss these lattice problems in more detail.

19.4 Example

For the following examples, we're going to be using the following parameters: $q = 13,\ n = 4,\ m = 14$.

Key Generation Our randomly chosen $\mathbf{A}, \mathbf{s}$ and $\mathbf{e}$ are:

$$\mathbf{A} = \begin{pmatrix} 4 & 11 & 5 & 7 \\ 7 & 3 & 3 & 10 \\ 9 & 8 & 9 & 3 \\ 12 & 7 & 1 & 9 \\ 0 & 8 & 2 & 9 \\ 7 & 11 & 4 & 5 \\ 11 & 0 & 11 & 7 \\ 0 & 3 & 5 & 1 \\ 6 & 5 & 4 & 1 \\ 3 & 8 & 12 & 9 \\ 6 & 0 & 12 & 1 \\ 7 & 3 & 8 & 9 \\ 5 & 0 & 1 & 1 \\ 11 & 5 & 0 & 8 \end{pmatrix}, \mathbf{s} = \begin{pmatrix} 7 \\ 3 \\ 11 \\ 12 \end{pmatrix}, \mathbf{e} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Giving:

$$\mathbf{b} = \mathbf{A}\cdot\mathbf{s}+\mathbf{e} = \begin{pmatrix} 4 & 11 & 5 & 7 \\ 7 & 3 & 3 & 10 \\ 9 & 8 & 9 & 3 \\ 12 & 7 & 1 & 9 \\ 0 & 8 & 2 & 9 \\ 7 & 11 & 4 & 5 \\ 11 & 0 & 11 & 7 \\ 0 & 3 & 5 & 1 \\ 6 & 5 & 4 & 1 \\ 3 & 8 & 12 & 9 \\ 6 & 0 & 12 & 1 \\ 7 & 3 & 8 & 9 \\ 5 & 0 & 1 & 1 \\ 11 & 5 & 0 & 8 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ 3 \\ 11 \\ 12 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \bmod 13 = \begin{pmatrix} 5 \\ 3 \\ 1 \\ 2 \\ 11 \\ 4 \\ 10 \\ 11 \\ 9 \\ 12 \\ 4 \\ 7 \\ 6 \\ 6 \end{pmatrix}$$

$(\mathbf{A}, \mathbf{b})$ is returned as the public key and $\mathbf{s}$ is returned as the private key.

Encryption We're going to encrypt a message of '1'. First we generate our random values:

$$\mathbf{r}^\top = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix},\ \mathbf{e}'^\top = \begin{pmatrix} 0 & -1 & 0 & 0 \end{pmatrix},\ e'' = -1$$

This gives us:

$$\mathbf{a}'^\top = \mathbf{r}'^\top\cdot\mathbf{A}+\mathbf{e}'^\top = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 4 & 11 & 5 & 7 \\ 7 & 3 & 3 & 10 \\ 9 & 8 & 9 & 3 \\ 12 & 7 & 1 & 9 \\ 0 & 8 & 2 & 9 \\ 7 & 11 & 4 & 5 \\ 11 & 0 & 11 & 7 \\ 0 & 3 & 5 & 1 \\ 6 & 5 & 4 & 1 \\ 3 & 8 & 12 & 9 \\ 6 & 0 & 12 & 1 \\ 7 & 3 & 8 & 9 \\ 5 & 0 & 1 & 1 \\ 11 & 5 & 0 & 8 \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 6 & 10 & 2 & 2 \end{pmatrix}$$

$$b' = \mathbf{r}^\top\cdot\mathbf{b}+e''+\mathrm{pt}\cdot\frac{q}{2} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 3 \\ 1 \\ 2 \\ 11 \\ 4 \\ 10 \\ 11 \\ 9 \\ 12 \\ 4 \\ 7 \\ 6 \\ 6 \end{pmatrix} - 1 + 6 = 9.$$

Giving us a ciphertext of $(\mathbf{a}', b')$

Decryption Given the ciphertext and the private key, find the corresponding plaintext.

$$v = \mathbf{a}'^{\top} \cdot \mathbf{s} = \begin{pmatrix} 6 & 10 & 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ 3 \\ 11 \\ 12 \end{pmatrix} \bmod 13 = 1.$$

$$v' - v = 9 - 1 = 8$$

As 8 is closer to 6 than 13, we return 1. We've correctly decrypted the ciphertext.

Non-deterministic In all the cryptosystems you've seen so far, if you encrypt the same message with the same key, you will always get the same result. We call cryptosystems where this happens deterministic. The LWE cryptosystem is non-deterministic, and so every time you encrypt the same message with the same key the ciphertext will be different (with high probability). For example another valid encryption of the message '1' using the above public key is the ciphertext $\mathbf{a}'^{\top} = \begin{pmatrix} 0 & 6 & 12 & 2 \end{pmatrix}$, $b' = 9$, or even $\mathbf{a}'^{\top} = \begin{pmatrix} 3 & 10 & 6 & 2 \end{pmatrix}$, $b' = 10$

A second decryption Since there are multiple valid encryptions of the message '1', we now try to decrypt one of the other ciphertexts.

$$\mathbf{a}'^{\top} = \begin{pmatrix} 3 & 10 & 6 & 2 \end{pmatrix},\ b' = 10$$

$$v = \mathbf{a}'^{\top} \cdot \mathbf{s} = \begin{pmatrix} 3 & 10 & 6 & 2 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ 3 \\ 11 \\ 12 \end{pmatrix} \bmod 13 = 11.$$

$$v' - v = 10 - 11 = 12$$

As 12 is closer to 13 than 6, we return 0. We've got a decryption failure!

19.5 See further

Learning with Rounding The LWE problem has a large number of variants, a common one is Learning with Rounding (LWR) where instead of adding errors, we round from q to p (for $p \ll q$). The equation changes from $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ to $\mathbf{b} = \lfloor \mathbf{A} \cdot \mathbf{s} \rceil_{q \to p}$.

$\lfloor x \rceil_{q \to p} = \left\lfloor \frac{p}{q} x \right\rceil \bmod p$ e.g. rounding $\begin{pmatrix} 5 \\ 7 \\ 15 \\ 13 \end{pmatrix}$ from $\mathbb{Z}_{16}$ to $\mathbb{Z}_4$ would give $\begin{pmatrix} 1 \\ 2 \\ 0 \\ 3 \end{pmatrix}$.

NIST-PQC standardisation After DES was broken, NIST (The US National Institute of Standards and Technology) launched a competition to find an algorithm to replace it in 1997, and the process ended in 2001 with Rijndael selected as the 'winner'. We are currently going through a similar standardisation process for post-quantum cryptography, although we expect multiple winners. Leading candidates include Kyber (based on a LWE variant), Saber (based on a LWR variant), NTRU (another hard lattice

problem) and Classic McEliece (based on code based crypto, covered in Lecture 29); alternate candidates include Frodo (based on LWE), NTRU-Prime, BIKE (code-based), HQC (code-based) and SIKE (based on Supersingular isogenies).

19.6 Exercises

Exercise 19.1. In the paragraph on non-determinism, two alternate encryptions of the message were given, we decrypted one of them and had a decryption failure. Decrypt the other and check if it decrypts correctly.

Exercise 19.2. A realistic parameter set for a 128-bit secure LWE cryptosystem, is $q = 2^{15}, n = 8, m = 640$. Assuming we're sending a 128-bit message, what is the size of the message after it has been encrypted? How does this compare to other public key encryption schemes you've met so far?

20 Lattice-based cryptography II

20.1 Lattices

The basis of a lattice is a set of n linearly independent vectors in $\mathbb{R}^d$, and is denoted $\mathbf{B} = (\mathbf{b}_1, ..., \mathbf{b}_n) \in \mathbb{R}^{d \times n}$.
A lattice is set of points generated by taking all integer linear combinations of a basis. The lattice generated by the basis $\mathbf{B}$ is:

$$\mathscr{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{n} x_i \cdot \mathbf{b}_i : \forall x_i \in \mathbb{Z} \right\}.$$

A lattice is said to be full rank if $d = n$. We will only be looking at full rank lattices, and so can treat the basis $\mathbf{B}$ as being a square matrix of dimension $n \times n$. Treating the basis as a matrix, we can also write the definition of a lattice as

$$\mathscr{L}(\mathbf{B}) = \{\mathbf{B} \cdot \mathbf{x} : \forall \mathbf{x} \in \mathbb{Z}^n\}.$$

Some examples:
The basis $\mathbf{B} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ gives the integer lattice $\mathbb{Z}^2$, since every vector can be created by some integer combination of $\mathbf{B}$, a visualisation of this can be seen in Figure 2.1.
A visualisation of the lattice generated from the basis $\mathbf{B} = \begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix}$ is given in Figure 2.2.

Fundamental parallelepiped The fundamental parallelepiped is the area bounded by all possible 0,1 combinations of the basis vectors, and is defined as the set of points

$$\mathscr{P}(\mathbf{B}) = \mathbf{B} \cdot [0,1)^n = \left\{ \sum_{i=0}^{n} x_i \cdot \mathbf{b}_i : \forall 0 \le x_i < 1 \right\}.$$

In Figure 2.3 you can see $\mathscr{P}(\mathbf{B})$ highlighted. If you think of the lattice as a wall, you can consider the fundamental parallelepiped as being a brick. The entire lattice is made up of an infinite amount of shifted copies of the fundamental parallelepiped, also known as a tiling. A visualisation of this can be seen in Figure 2.4.
The volume of a lattice, denoted $\det(\mathscr{L})$ is the defined as the volume of the fundamental parallelepiped, $\text{vol}(\mathscr{P}(\mathbf{B}))$. The volume of the fundamental parallelepiped is equal to the determinant of the basis:

$$\det(\mathscr{L}(\mathbf{B})) = \text{vol}(\mathscr{P}(\mathbf{B})) = |\det(\mathbf{B})|.$$

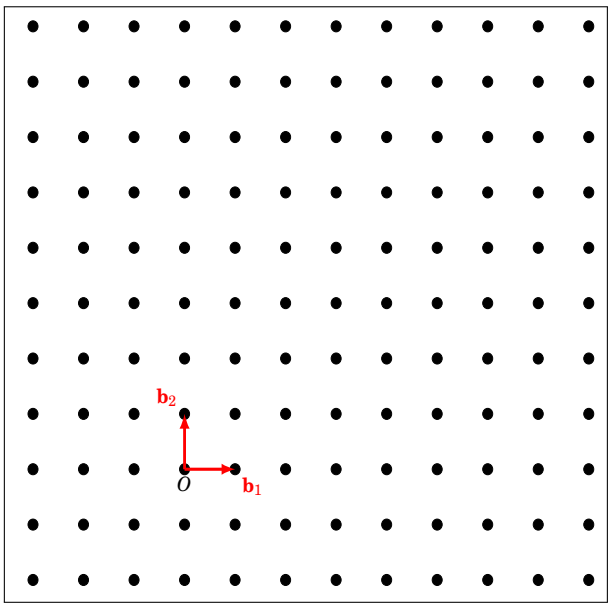


Figure 2.1: The lattice formed from $\mathbf{B} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

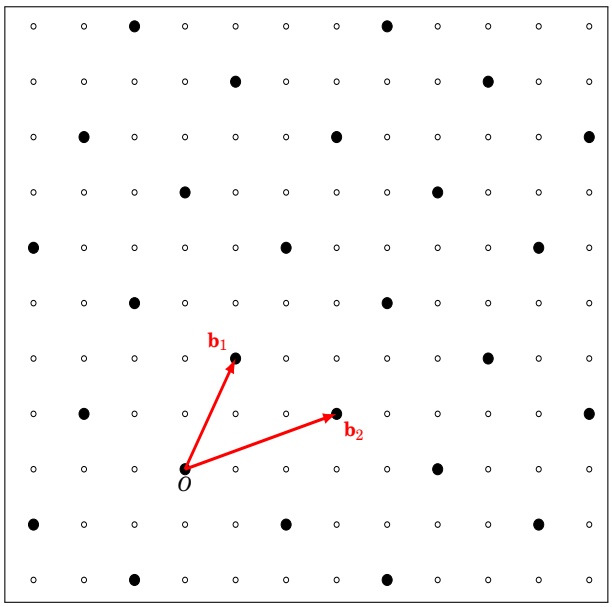


Figure 2.2: The lattice formed from $\mathbf{B} = \begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix}$

Minimum distance The minimum distance (λ_1) on a lattice is the smallest distance between any two lattice points:

$$\lambda_1 = \min\{||\mathbf{x}-\mathbf{y}|| \forall \mathbf{x}, \mathbf{y} \in \mathscr{L}, \mathbf{x} \neq \mathbf{y}\}.$$

The minimum distance of the lattice is also the length of the shortest vector in the lattice.
Minkowski's theorem states that for any lattice,

$$\lambda_1(\mathscr{L}) \leq \sqrt{n} \cdot \det(\mathscr{L})^{1/n}.$$

This gives us an upper bound for the length of the shortest vector.

20.2 Shortest Vector Problem

Shortest Vector Problem.
Instance: A lattice $\mathscr{L}(\mathbf{B})$.
Question: What is the shortest non-zero vector in the lattice? (i.e. find $\mathbf{v} \in \mathscr{L}$ s.t. $||\mathbf{v}|| = \lambda_1(\mathscr{L})$).

There also exists an approximate version of this problem, where we are asked to find a vector that is at most γ times the size of the longest vector. Where $\gamma = \gamma(n) \geq 1$.
Approximate-Shortest Vector Problem.
Instance: A lattice $\mathscr{L}(\mathbf{B})$.
Question: Find a non-zero vector in the lattice that is at most γ times longer than the shortest vector. (i.e. find $\mathbf{v} \in \mathscr{L}$ s.t. $||\mathbf{v}|| \leq \gamma(n) \cdot \lambda_1(\mathscr{L})$).

SVP has been shown to be NP-Hard, for approximate-SVP the hardness depends on the approximation factor (γ). For $\gamma = O(1)$, approximate-SVP is "hard", for $\gamma = O(\sqrt{n})$ it's in NP $\cap$ co-NP, for $\gamma = 2^{\tilde{O}(n)}$ it's in P. For crypto we tend to rely on the hardness of approximate-SVP for $\gamma = \text{poly}(n)$.
There are a number of approaches for solving SVP, we'll discuss two of them.

Enumeration The 'brute force' method of solving SVP is enumeration, given a lattice we enumerate through every lattice point in some bounded region, typically the fundamental parallelepiped. Whilst very slow, this method is still often used as it has very low memory requirements.

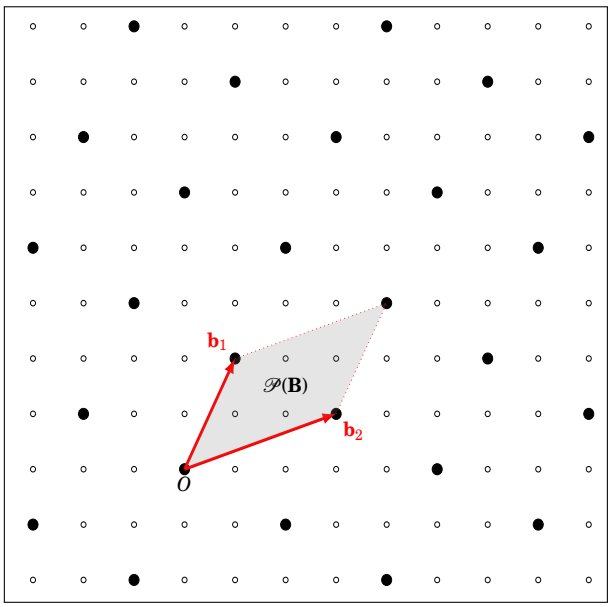


Figure 2.3: The fundamental parallelepiped $\mathscr{P}(\mathbf{B})$

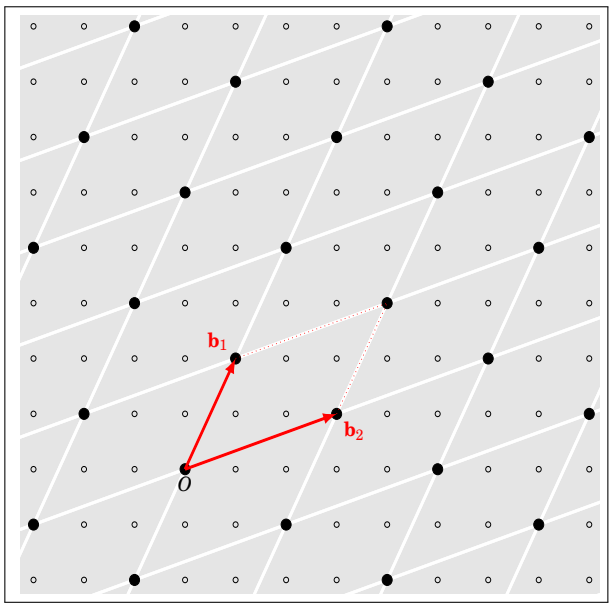


Figure 2.4: Tiling the lattice with $\mathscr{P}(\mathbf{B})$

Sieving Sieving algorithms are randomised algorithms that, in some cases, run very efficiently but use an exponential amount of space. They work by generating some random lattice points and then taking the difference between these points (Figure 2.5). The difference between any two lattice points is also a valid lattice point, and so this is then added to the list (Figure 2.6). Hopefully as this is repeated a short vector is found.

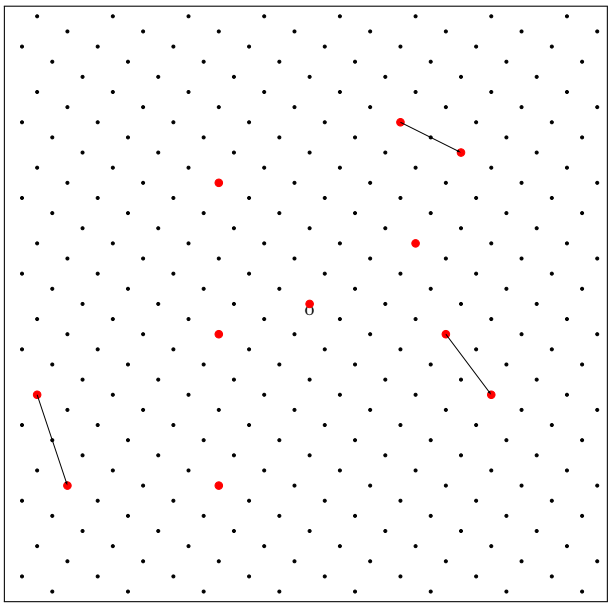


Figure 2.5: Choosing some random points and finding the differences.

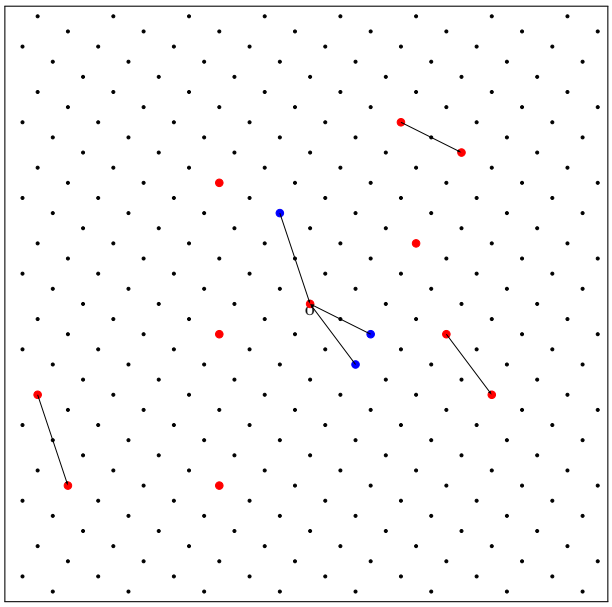


Figure 2.6: Adding the new vectors to the list, ready to repeat.

20.3 Breaking LWE

The main method for breaking LWE involves turning it into a SVP instance. Given an LWE instance $(\mathbf{A}, \mathbf{b})$, $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, we can construct an SVP instance by creating a lattice basis

$$\mathbf{B} = \begin{pmatrix} \mathbf{A}^\top & 0 \\ \mathbf{b}^\top & 1 \end{pmatrix} \in \mathbb{Z}_q^{(n+1)\times(m+1)}.$$

Every point on this lattice can be defined by $(\mathbf{x}^\top | y) \cdot \mathbf{B} = \begin{pmatrix} \mathbf{x}^\top \cdot \mathbf{A}^\top + y \cdot \mathbf{b}^\top \\ y \end{pmatrix}$, where $\mathbf{x} \in \mathbb{Z}^n$ and $y \in \mathbb{Z}$. The length of the vector represented by this point is minimised when $\mathbf{x}^\top \cdot \mathbf{A}^\top + y \cdot \mathbf{b}^\top$ is at its minimum, in fact the length of this vector is at its shortest when $\mathbf{x} = \mathbf{s}$ and $y = -1$, giving us $(\mathbf{s}^\top \cdot \mathbf{A}^\top - \mathbf{b}^\top | -1) = ((\mathbf{A} \cdot \mathbf{s} - \mathbf{b})^\top | -1) = (\mathbf{e}^\top | -1)$.
If we can find the shortest vector in this lattice $(\mathbf{e}^\top | -1)$, then we can calculate $(\mathbf{e}^\top | -1) \cdot \mathbf{B}^{-1} = (\mathbf{s}^\top | -1)$ and therefore have solved the LWE instance.

20.4 See further

Homomorphic encryption LWE is very versatile, and has opened up new branches of cryptography. It has allowed cryptographers to create the first homomorphic encryption system, where the exists some addition (or multiplication) such that applying that function to two ciphertexts and decrypting would give the same result as just adding (or multiplying) the initial plaintexts, i.e. $d(Add(e(p1), e(p2))) = p1 + p2$, where $e()$, $d()$ are the encryption and decryption functions.

What other crypto is there? Cryptography is an ever expanding field of research, other interesting areas include:

- Multi-party computation - multiple users want to perform a computation on all their inputs, without anyone revealing their inputs.
- Threshold cryptography - a group of n users want to encrypt some data, such that for some $t < n$, any subgroup of these users of size at least t can decrypt the data while any subgroup of these users of size less than t are unable to.
- Zero knowledge proofs - prove to someone that you know some value without giving away the value.
- Random number generation.

Chapter 3

Error-correcting codes

21 Linear codes I: Introduction to error-correcting codes

21.1 Error control

The tenets of error control Messages are subject to errors when transmitted through a channel. The channel can either be spatial (data transmission) or temporal (storage). The main advantage of digital data over analog data is that we can perform error correction. We use two main assumptions:

- We simplify the channel.
- We suppose errors occur infrequently.

The main idea is to add redundancy to the data.

The codebook idea The language itself is an error correcting code:

> "I am going to the conkert tonight."

In this case, it is easy to detect and correct the error:

> "I am going to the concert tonight."

Some sequences of letters are correct, others are incorrect. The correct ones form a code.

21.2 Binary repetition and parity-check codes

Basic error detection Binary parity-check code: add one more bit to the sequence of bits so that the overall number of 1's is even. For instance,

$$1100011 \xrightarrow{\text{encoding}} 11000110$$

The codebook then consists of all sequences of a given length with an even number of bits equal to 1. This was used in the first version of ASCII (7 bits for the symbol + 1 parity-check bit).

The parity-check code can detect one error (simply check the number of ones!) e.g.

$$1100011 \xrightarrow{\text{encoding}} 11000110 \xrightarrow{\text{channel}} 11010110$$

It cannot detect two errors:

$$1100011 \xrightarrow{\text{encoding}} 11000110 \xrightarrow{\text{channel}} 11011110$$

It cannot correct any error: e.g. if we receive 11010110, what happened?

$$1100011 \xrightarrow{\text{encoding}} 11000110 \xrightarrow{\text{channel}} 11010110 \quad \textit{or}$$

$$1101011 \xrightarrow{\text{encoding}} 11010111 \xrightarrow{\text{channel}} 11010110 \quad ?$$

In this case, there are eight possible scenarios, one for each error location.

Basic error correction Binary repetition code: send the same bit n times (e.g. $n = 3$)

$$0 \to 000$$
$$1 \to 111.$$

The codebook consists of two codewords: $\{000, 111\}$.

The repetition code can correct $\lfloor \frac{n-1}{2} \rfloor$ errors e.g.

$$0 \xrightarrow{\text{encoder}} 000 \xrightarrow{\text{channel}} 010 \xrightarrow{\text{decoder}} 0$$
$$1 \xrightarrow{\text{encoder}} 111 \xrightarrow{\text{channel}} 110 \xrightarrow{\text{decoder}} 1$$

It can detect $n-1$ errors e.g. $0 \to 000 \to 110$. However, the repetition code has a very low rate: $\frac{1}{n}$! We need more rate-efficient techniques.

Objectives We want to design a "good" error correcting code, i.e.

1. Detects and corrects many errors
2. High rate
3. Easy to encode and decode.

The first two are conflicting: compromise depending on the channel quality. We will mainly focus on error correction over detection.

21.3 Minimum distance

Hamming distance The Hamming distance between two vectors is the number of times they disagree. E.g. $d_H(100, 101) = 1$. It is a metric:

1. $d_H(\mathbf{x}, \mathbf{y}) \geq 0$,
2. $d_H(\mathbf{x}, \mathbf{y}) = d_H(\mathbf{y}, \mathbf{x})$,
3. $d_H(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$,
4. $d_H(\mathbf{x}, \mathbf{y}) \leq d_H(\mathbf{x}, \mathbf{z}) + d_H(\mathbf{z}, \mathbf{y})$ (triangular inequality).

Thus, it has a geometric meaning.

The Hamming weight of a vector is simply the number of nonzero coefficients:

$$w_H(\mathbf{x}) = d_H(\mathbf{x}, \mathbf{0}).$$

Hamming distance and error correction The decoding problem is as follows. If the destination receives the vector $\mathbf{v}$, the decoder returns the unique nearest (in terms of Hamming distance) codeword to $\mathbf{v}$ if it exists.

Problem: what if that codeword is not unique? We thus need a sufficient condition to make sure the decoding is non-ambiguous.

Minimum distance A code C is a subset of $[q]^n$, where $[q]$ represents any set of size q. The minimum distance of C, denoted $d_{\min}(C)$, is the minimum Hamming distance between two distinct codewords in C:

$$d_{\min}(C) := \min\{d_H(\mathbf{c}, \mathbf{c}') : \mathbf{c}, \mathbf{c}' \in C, \mathbf{c} \neq \mathbf{c}'\}.$$

Theorem 21.1. *A code can correct t errors if and only if it has minimum distance $d_{\min} \geq 2t + 1$.*

Proof. For any $\mathbf{c} \in \mathrm{GF}(q)^n$, let $S_r(\mathbf{c})$ denote the sphere of Hamming radius r centred around $\mathbf{c}$:

$$S_r(\mathbf{c}) = \left\{\mathbf{v} \in \mathrm{GF}(q)^n : d_{\mathrm{H}}(\mathbf{v},\mathbf{c}) \le r\right\}.$$

Then it is clear that C corrects t errors if and only if $S_t(\mathbf{c}) \cap S_t(\mathbf{c}') = \emptyset$ for distinct codewords $\mathbf{c}$ and $\mathbf{c}'$.

Suppose $S_t(\mathbf{c}) \cap S_t(\mathbf{c}') \neq \emptyset$ for distinct codewords $\mathbf{c}$ and $\mathbf{c}'$, say $d_{\mathrm{H}}(\mathbf{v},\mathbf{c}) \le t$ and $d_{\mathrm{H}}(\mathbf{v},\mathbf{c}') \le t$. Then the triangular inequality yields

$$d_{\mathrm{H}}(\mathbf{c},\mathbf{c}') \le d_{\mathrm{H}}(\mathbf{v},\mathbf{c}) + d_{\mathrm{H}}(\mathbf{v},\mathbf{c}') \le 2t,$$

and hence $d_{\min} \le 2t$.

Conversely, let C have minimum distance $d_{\min} \le 2t$. Let $\mathbf{c},\mathbf{c}' \in \mathsf{C}$ at Hamming distance $2t$; we now construct $\mathbf{v} \in S_t(\mathbf{c}) \cap S_t(\mathbf{c}')$. This is clear if $d_{\min} \le t$, so we assume $d_{\min} > t$. Let Δ be the set of coordinates where $\mathbf{c}$ and $\mathbf{c}'$ disagree, i.e. $|\Delta| \le 2t$ and $\mathbf{c}_j \neq \mathbf{c}_j$ for all $j \in \Delta$. Select $\Gamma \subseteq \Delta$ with $|\Gamma| = t$, and define

$$\mathbf{v}_\Gamma = \mathbf{c}_\Gamma, \qquad \mathbf{v}_{\Delta\setminus\Gamma} = \mathbf{c}'_{\Delta\setminus\Gamma}, \qquad \mathbf{v}_{\{1,\dots,n\}\setminus\Delta} = \mathbf{c}_{\{1,\dots,n\}\setminus\Delta} = \mathbf{c}'_{\{1,\dots,n\}\setminus\Delta}.$$

Then indeed $d_{\mathrm{H}}(\mathbf{v},\mathbf{c}) = d_{\min} - t \le t$ and $d_{\mathrm{H}}(\mathbf{v},\mathbf{c}') = t$. □

21.4 Bounds on codes

Let $A(n,q,d)$ be the maximum cardinality of a q-ary code of length n and minimum distance (at least) d. The volume $|S_r(\mathbf{c})|$ of the sphere does not depend on its centre and is denoted by $V(n,q,r)$.

Sphere-packing bound The sphere-packing bound (a.k.a. Hamming bound) is an upper bound on $A(q,n,d)$.

Theorem 21.2 (Sphere-packing bound.)**.**

$$A(n,q,d) \le \frac{q^n}{V\left(n,q,\left\lfloor\frac{d-1}{2}\right\rfloor\right)}.$$

Proof. Let C be an optimal code with cardinality $A(n,q,d)$. Let $t = \left\lfloor\frac{d-1}{2}\right\rfloor$. The spheres of radius t centred around the codewords do not overlap (they are the decoding regions!) and hence can be packed. Thus there are $|\mathsf{C}|V(n,q,t) \le q^n$ vectors in those spheres. □

Gilbert bound The Gilbert bound (a.k.a. Gilbert-Varshamov bound) is a lower bound on $A(q,n,d)$. From the proof one can easily derive a greedy algorithm to produce a code whose cardinality reaches or surpasses that bound.

Theorem 21.3 (Gilbert bound)**.**

$$A(n,q,d) \ge \frac{q^n}{V(n,q,d-1)}.$$

Proof. Let C be an optimal code with cardinality $A(n,q,d)$. For every vector $\mathbf{v} \in [q]^n$, there exists $\mathbf{c} \in \mathsf{C}$ at Hamming distance at most $d-1$, for otherwise $\mathsf{C} \cup \{\mathbf{v}\}$ would have minimum distance $\ge d$ and a larger cardinality. In other words, the spheres of radius $d-1$ centred around the codewords cover the whole space. There are $q^n \le |\mathsf{C}|V(n,q,d-1)$ vectors. □

21.5 See further

Other metrics In this course, we will focus on traditional error-correcting codes, where the codewords are vectors and the distance between two codewords is the Hamming distance. However, other metrics have been proposed for different situations. We mention two below.

Firstly, the Hamming metric does not take into account the amplitude of an error: if a coordinate is changed during transmission, then it yields one Hamming error, whatever it is. On the other hand, the

Lee metric has been proposed for situations where the amplitude matters, such as phase modulated signals (and again, errors with low amplitude are less likely than errors with high magnitude). Formally, let $[q] = \{0, 1, \ldots, q-1\}$, and for any $a, b \in [q]$ let

$$l(a,b) = \min\{|a-b|, q-|a-b|\}$$

(imagine the symbols being placed on a cycle you can turn clockwise or anticlockwise). Then the Lee distance between $\mathbf{x}$ and $\mathbf{y}$ is simply $\sum_i l(x_i, y_i)$.

Secondly, consider data that is stored in a two-dimensional array (like your campus card). Typical errors may corrupt a whole (or at least a significant part of a) column or row of that array. The rank metric was proposed for that scenario. The codewords are $m \times n$ matrices, and the distance between two matrices $\mathbf{M}$ and $\mathbf{N}$ is the rank distance $\operatorname{rank}(\mathbf{M}-\mathbf{N})$. Note that corrupting a whole row corresponds to adding a matrix $\mathbf{E}$ where the nonzero entries are confined to that row (and hence $\operatorname{rank}(\mathbf{E}) = 1$). The same holds for columns. Therefore, corrupting t rows or columns amounts to adding a matrix $\mathbf{E}_t$ with $\operatorname{rank}(\mathbf{E}_t) \leq t$.

Other bounds on codes Many more bounds on codes have been derived. We can briefly mention the Singleton bound, the Plotkin bound, the Johnson bound, the code-anticode bound, linear programming bounds, etc.

The EAN The International Article Number (underneath the barcode) uses a variant of the parity-check code: $\mathbf{c} = (c_1, \ldots, c_{13})$, where

$$c_{13} = -\sum_{i=0}^{5} (c_{2i+1} + 3c_{2i+2}) \mod 10.$$

For instance, let us determine the last symbol of this barcode: $5-045092-36551$?

$$\begin{aligned} c_{13} &= -[5 + (3\times 0) + 4 + (3\times 5) + 0 + (3\times 9) + 2 + (3\times 3) + 6 + (3\times 5) + 5 + (3\times 1)] \mod 10 \\ &= -1 \mod 10 \\ &= 9. \end{aligned}$$

21.6 Exercises

Exercise 21.1. Prove that the Hamming distance is a metric on $[q]^n$.

Exercise 21.2. Prove that if $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ have the same Hamming weight, then their Hamming distance is even.

Exercise 21.3. Prove that

$$V(q,n,r) = \sum_{d=0}^{r} \binom{n}{d} (q-1)^d.$$

Exercise 21.4. Prove that for any $\mathbf{x}, \mathbf{y} \in [q]^n$ at distance d, and any $0 \leq t \leq d$, there are exactly $\binom{d}{t}$ vectors $\mathbf{v} \in [q]^n$ such that $d_{\mathrm{H}}(\mathbf{v}, \mathbf{x}) = t$ and $d_{\mathrm{H}}(\mathbf{v}, \mathbf{y}) = d - t$.

Exercise 21.5. Find out how the CIS id's (e.g. "bcdf12") are generated. I suspect at least one of the two digits is some form of parity-check, but I don't know!

22 Linear codes II: Finite fields

22.1 Construction of finite fields

Finite fields A field is an algebraic structure in which we can add, subtract, multiply and divide (and multiplication distributes over addition), just like over the real numbers.

More formally, a group is a pair $(G, \cdot)$ where G is a set and $\cdot$ is binary operation on G that satisfies the three following properties (for all $a, b, c \in G$):

1. Associativity: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
2. Identity element: there exists $e \in G$ such that $a \cdot e = e \cdot a = a$.
3. Inverses: there exists a^{-1} such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

If furthermore, the operation is commutative, i.e. $a \cdot b = b \cdot a$, then the group $(G, \cdot)$ is called abelian. A field is a 5-tuple $(F, +, \times, 0, 1)$ such that

1. $(F, +)$ is an abelian group with identity 0.
2. $(F^*, \times)$ is an abelian group with identity 1.
3. Multiplication distributes over addition: $a \times (b + c) = (a \times b) + (a \times c)$.

For instance, $(\mathbb{R}, +, \times, 0, 1)$ is a field. We use similar notation for any other field F e.g. writing $-a$ for the additive inverse of a.

Theorem 22.1 (Galois)**.** *Any finite field is of order (size) $q = p^m$, where p is a prime number and m is any integer ≥ 1. Moreover, all fields of size q are the same (up to renaming elements).*

We can then talk about the field of order q, commonly denoted as $\mathrm{GF}(q)$ or $\mathbb{F}_q$.

Prime fields $\mathrm{GF}(p)$ Finite fields of prime order $\mathrm{GF}(p)$ are simply modular arithmetic $\mod p$. E.g. $p = 5$

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

We can indeed add, and subtract: $-1 = 4$, $-2 = 3$, $-3 = 2$, $-4 = 1$.
We can multiply and divide (not by zero, of course): $1^{-1} = 1$, $2^{-1} = 3$, $3^{-1} = 2$, $4^{-1} = 4$.

This holds for all prime p. The only nontrivial thing to prove is that every nonzero integer $\beta \in \mathbb{Z}_p^*$ has an inverse. This follows from a fundamental result about the gcd: if $a, b \in \mathbb{Z}$ have $\gcd(a, b) = d$, then there exist $s, t \in \mathbb{Z}$ such that $as + bt = d$. Since $\gcd(\beta, p) = 1$, there exists γ such that $\beta\gamma = 1 \mod p$.

Extension fields $\mathrm{GF}(p^m)$ The extension field $\mathrm{GF}(p^m)$ is constructed as follows. Let $P(x)$ be a polynomial of degree m over $\mathrm{GF}(p)$. We say $P(x)$ is irreducible if it cannot be factored as $P(x) = Q(x)R(x)$ with $\deg Q(x), \deg R(x) < m$. We say $P(x)$ is primitive if it is irreducible and the smallest positive integer n such that $P(x) \mid x^n - 1$ is $n = p^m - 1$.

Now, choose a primitive polynomial $P(x)$ of degree m over $\mathrm{GF}(p)$, and let α be a root. Then

$$\mathrm{GF}(p^m) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{p^m - 2}\}$$

For example, let us construct GF(4). Let α be a root of $x^2 + x + 1$, i.e.

$$\alpha^2 + \alpha + 1 = 0, \quad \text{or equivalently,} \quad \alpha^2 = \alpha + 1.$$

+	0	1	α	$\alpha+1$
0	0	1	α	$\alpha+1$
1	1	0	$\alpha+1$	α
α	α	$\alpha+1$	0	1
$\alpha+1$	$\alpha+1$	α	1	0

×	0	1	α	α^2
0	0	0	0	0
1	0	1	α	α^2
α	0	α	α^2	1
α^2	0	α^2	1	α

The field GF(8) can be constructed similarly. Let α be a root of $x^3 + x + 1$, i.e.

$$\alpha^3 + \alpha + 1 = 0.$$

Then

$$\mathrm{GF}(8) = \{0, 1, \alpha, \alpha^2, \alpha^3 = \alpha + 1, \alpha^4 = \alpha^2 + \alpha, \alpha^5 = \alpha^2 + \alpha + 1, \alpha^6 = \alpha^2 + 1\}.$$

22.2 Properties

Multiplicative structure The multiplicative structure of $\mathrm{GF}(q)^*$ is very simple: it is that of a cyclic group of order $q-1$. More concretely,

$$\mathrm{GF}(q)^* = \{\alpha, \alpha^2, \ldots, \alpha^{q-2}, \alpha^{q-1} = 1\},$$

where α is the root of $P(x)$. This is the exponential notation. Let $\beta \in \mathrm{GF}(q)^*$, then $\beta = \alpha^k$ for some $0 \leq k \leq q-2$, thus

$$\beta^{q-1} = \left(\alpha^k\right)^{q-1} = \left(\alpha^{q-1}\right)^k = 1.$$

For any $\beta \in GF(q)^*$, the order of β, denoted as $o(\beta)$ is the smallest positive integer t such that $\beta^t = 1$. If $o(\beta) = q-1$, then β is said to be primitive (and hence any element in $\mathrm{GF}(q)^*$ can be expressed as a power of β).

Additive structure For any $\beta \in \mathrm{GF}(p^m)$, we have

$$p\beta = \underbrace{\beta + \beta + \cdots + \beta}_{p \text{ times}} = 0.$$

We say that p is the characteristic of $\mathrm{GF}(p^m)$. Consequently, for any $\beta, \gamma \in \mathrm{GF}(p^m)$,

$$(\beta + \gamma)^p = \beta^p + \gamma^p.$$

$\mathrm{GF}(p^m)$ forms a vector space of dimension m over $\mathrm{GF}(p)$, with the polynomial basis $(1, \alpha, \ldots, \alpha^{m-1})$. This yields the polynomial notation:

$$\beta = \sum_{i=0}^{m-1} \beta_i \alpha^i \quad \beta_i \in \mathrm{GF}(p).$$

Subfields In general, we can construct $\mathrm{GF}(q^m)$ from a primitive polynomial over $\mathrm{GF}(q)$. Then $\mathrm{GF}(q^m)$ contains $\mathrm{GF}(q^n)$ as a subfield if and only if $n \mid m$. For instance, $\mathrm{GF}(64) = \mathrm{GF}(2^6)$ contains $\mathrm{GF}(2^6)$, $\mathrm{GF}(2^3)$, $\mathrm{GF}(2^2)$ and $\mathrm{GF}(2^1)$ as subfields.

Theorem 22.2. *$\beta \in \mathrm{GF}(q^m)$ lies in the subfield $\mathrm{GF}(q)$ if and only if $\beta^{q-1} = 1$.*

22.3 See further

Évariste Galois Finite fields were discovered by the French mathematician Évariste Galois (1811-1832). He made massive contributions to algebra (founding group theory, so-called Galois theory and finite fields) by the age of 21, before dying in a duel against an officer. His fascinating story combines genius, political activism, romanticism, and incomprehension.

22.4 Exercises

Exercise 22.1. Let F be a field. Prove the following statements.

1. For any element $a \in F$, $a0 = 0a = 0$.
2. The field F does not have any divisor of zero, i.e. if $ab = 0$, then either $a = 0$ or $b = 0$.

Exercise 22.2. Give the addition and multiplication tables for GF(7).

Exercise 22.3. GF(16) is generated by $x^4 + x + 1$. Give the conversion exponential-polynomial (like what we did for GF(8) in Lecture 1).

Exercise 22.4. GF(9) is generated by $x^2 + x + 2$. Give the conversion exponential-polynomial (like what we did for GF(8) in Lecture 1).

Exercise 22.5. Prove the following. Let $\mathrm{GF}(q)$ have characteristic p and $\beta, \gamma \in \mathrm{GF}(q)$. Then

$$(\beta+\gamma)^p = \beta^p + \gamma^p.$$

Exercise 22.6. Verify that for any $\beta \in \mathrm{GF}(q)$, $\beta^q = \beta$. Conclude that the mapping $\sigma : \mathrm{GF}(q) \to \mathrm{GF}(q)$, $\sigma(\beta) = \beta^p$ (called the Frobenius automorphism) satisfies the following properties:

1. it is a field automorphism, i.e. it preserves addition and multiplication,
2. it fixes $\mathrm{GF}(p)$, i.e. $\sigma(b) = b$ for all $b \in \mathrm{GF}(p)$,
3. it is a linear mapping when we view $\mathrm{GF}(q)$ as a vector space over $\mathrm{GF}(p)$.

23 Linear codes III: Linear codes

23.1 Linear codes

$\mathrm{GF}(q)^n$ is a vector space, hence we can do linear algebra. Following the convention of coding theory, we use row vectors.

Generator matrix Let C be a subspace of dimension k of $\mathrm{GF}(q)^n$ generated by the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ (its basis). We can represent C by a generator matrix $\mathbf{G}$, a $k \times n$ matrix over $\mathrm{GF}(q)$ with $\mathbf{v}_1, \ldots, \mathbf{v}_k$ as rows:

$$\mathbf{G} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \cdots \\ \mathbf{v}_k \end{pmatrix} \in \mathrm{GF}(q)^{k \times n}.$$

We then have

$$\mathsf{C} = \{\mathbf{mG} : \mathbf{m} \in \mathrm{GF}(q)^k\}.$$

The generator matrix is an efficient way to encode the data. The original message $\mathbf{m}$ of length k is encoded into the corresponding codeword $\mathbf{c}$ by multiplying by the generator matrix:

$$\mathbf{c} = \mathbf{mG}.$$

Repetition and parity-check codes The repetition code

$$\mathsf{R}(q,n) := \{(a, \ldots, a) : a \in \mathrm{GF}(q)\}$$

is the line (subspace of dimension 1) spanned by $\mathbf{v}_1 = (1, \ldots, 1)$. Then

$$\mathbf{G}_{\mathsf{R}} = \begin{pmatrix} 1 & \ldots & 1 \end{pmatrix}.$$

The parity-check code

$$\mathsf{PC}(q,n) := \left\{ \mathbf{v} \in \mathrm{GF}(q)^n : \sum_{i=1}^{n} v_i = 0 \right\}$$

is the hyperplane (subspace of dimension $n-1$) with generator matrix

$$\mathbf{G}_{\mathsf{PC}(q,n)} = \left(\begin{array}{c|c} \mathbf{I}_{n-1} & (-1)^\top \end{array} \right).$$

For instance, for $n = 5$ we have

$$\mathbf{G}_{\mathsf{PC}} = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}.$$

Generator matrices In general, the generator matrix is not unique. For instance, $\mathsf{PC}(2,3)$, the length-3 parity-check code over GF(2) has six distinct generator matrices:

$$\begin{pmatrix}1&1&0\\1&0&1\end{pmatrix} \quad \begin{pmatrix}1&1&0\\0&1&1\end{pmatrix}$$
$$\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix} \quad \begin{pmatrix}1&0&1\\0&1&1\end{pmatrix}$$
$$\begin{pmatrix}0&1&1\\1&1&0\end{pmatrix} \quad \begin{pmatrix}0&1&1\\1&0&1\end{pmatrix}$$

Dual subspace The inner product of two vectors $\mathbf{u}$ and $\mathbf{v}$ is given by:

$$\mathbf{u}\cdot\mathbf{v} := \sum_{i=1}^{n} u_i v_i = \mathbf{u}\mathbf{v}^\top.$$

Be careful: unlike in the real case, we can have $\mathbf{v}\cdot\mathbf{v} = 0$, although $\mathbf{v} \neq 0$. For any subspace C, the set

$$\mathsf{C}^\perp := \{\mathbf{v} \in \mathrm{GF}(q)^n : \mathbf{c}\cdot\mathbf{v} = 0 \; \forall \, \mathbf{c} \in \mathsf{C}\}$$

is a subspace, called the dual subspace of C. For any C, we have $(\mathsf{C}^\perp)^\perp = \mathsf{C}$ and

$$\dim(\mathsf{C}) + \dim(\mathsf{C}^\perp) = n.$$

Theorem 23.1. *Parity-check and repetition codes are dual.*

Proof. For any $\mathbf{v} \in \mathrm{GF}(q)^n$, $\mathbf{v} \in \mathsf{PC}(q,n)$ if and only if $\sum_{i=1}^n v_i = 0$. This is equivalent to

$$\mathbf{v}\cdot\mathbf{0} = \mathbf{v}\cdot\mathbf{1} = 0,$$

which means $\mathbf{v} \in \mathsf{R}(q,n)^\perp$. □

Parity-check matrix The parity-check matrix $\mathbf{H}$ of C is the generator matrix of $\mathsf{C}^\perp$. $\mathbf{H}$ is an $(n-k \times n)$ matrix over $\mathrm{GF}(q)$. E.g.

$$\mathbf{H}_{\mathsf{PC}} = \mathbf{G}_{\mathsf{R}},$$
$$\mathbf{H}_{\mathsf{R}} = \mathbf{G}_{\mathsf{PC}}.$$

We can then express C as

$$\mathsf{C} = \left\{\mathbf{c} \in \mathrm{GF}(q)^n : \mathbf{c}\mathbf{H}^\top = \mathbf{0}\right\}.$$

Therefore, $\mathbf{H}$ gives an efficient way of detecting errors. It will also be useful in correcting them.

Relation between generator and parity-check matrices

Theorem 23.2. *For any* C*, let* $\mathbf{G}$ *and* $\mathbf{H}$ *be a generator matrix and a parity-check matrix of* C*, respectively. Then*

$$\mathbf{G}\mathbf{H}^\top = \mathbf{0},$$

where $\mathbf{0}$ *is the all-zero matrix of size* $(k \times n-k)$*.*

Equivalent codes Two linear codes are said to be equivalent if one can be obtained from the other by permuting coordinates. In other words, C and D are equivalent if there exists a permutation $\pi \in S_n$ such that $\mathsf{D} = \pi(\mathsf{C})$. More explicitly, for any $\pi \in S_n$, consider the corresponding permutation matrix $\mathbf{P}$, which is an $n \times n$ matrix with

$$\mathbf{P}_{ij} = \begin{cases} 1 & \text{if } \pi(i) = j, \\ 0 & \text{otherwise.} \end{cases}$$

We then have

$$\mathbf{G}_{\mathsf{D}} = \mathbf{G}_{\mathsf{C}}\mathbf{P}.$$

It is clear that two equivalent codes have the same length, dimension, and minimum distance.

Systematic form Since $\mathbf{G}$ is a $k \times n$ matrix with full row rank, it contains a nonsingular $k \times k$ matrix $\mathbf{M}$ (i.e. k linearly independent columns). Up to equivalence, we can assume those are the first k columns:

$$\mathbf{G} = \left(\begin{array}{c|c} \mathbf{M} & \mathbf{P} \end{array} \right),$$

for some $k \times (n-k)$ matrix $\mathbf{P}$. For any nonsingular $k \times k$ matrix $\mathbf{R}$, the matrix $\mathbf{RG}$ is also a generator matrix of $\mathbf{G}$. Choosing $\mathbf{R} = \mathbf{M}^{-1}$, we obtain

$$\mathbf{G}' = \left(\begin{array}{c|c} \mathbf{I}_k & \mathbf{Q} \end{array} \right),$$

for some $k \times (n-k)$ matrix $\mathbf{Q}$. A generator matrix containing $\mathbf{I}_k$ is said to be in systematic form.

23.2 Parameters of a linear code

A linear code is a subspace of $\mathrm{GF}(q)^n$.

Parameters of a linear code The main parameters of a linear code are its:

- Length n
- Dimension k
- Minimum distance $d_{\min}$
- Field order q
- Redundancy $r = n - k$
- Rate $R = k/n$
- Error-correction capability $t = \lfloor (d_{\min} - 1)/2 \rfloor$

We usually write $[n, k, d_{\min}]_q$-"name of code," but we can omit $d_{\min}$ or q.

Parameter	Notation	$\mathsf{PC}(n,q)$	$\mathsf{R}(n,q)$
Length	n	n	n
Dimension	k	$n-1$	1
Minimum distance	$d_{\min}$	2	n
Field order	q	q	q
Redundancy	r	1	$n-1$
Rate	R	$1-1/n$	$1/n$
Error-correction capability	t	0	$\lfloor (n-1)/2 \rfloor$

Minimum distance of a linear code

Theorem 23.3. *Viewing the columns of* $\mathbf{H}$ *as vectors in* $\mathrm{GF}(q)^{n-k}$, $d_{\min}$ *is the minimum number of linearly dependent columns of* $\mathbf{H}$.

The minimum weight of a linear code C is defined as

$$w_{\min}(\mathsf{C}) = \min\{w_{\mathrm{H}}(\mathbf{c}) : \mathbf{c} \in \mathsf{C} \setminus \{\mathbf{0}\}\}.$$

Lemma 23.4. *For any linear code* C, *the minimum distance is equal to the minimum weight:*

$$d_{\min}(\mathsf{C}) = w_{\min}(\mathsf{C}).$$

Proof of Theorem 23.3. For any set of columns $\mathbf{h}_{i_1}, \ldots, \mathbf{h}_{i_m}$ of $\mathbf{H}$, those are linearly dependent if and only if there exist $c_{i_1}, \ldots, c_{i_m} \in \mathrm{GF}(q)^*$ such that

$$c_{i_1}\mathbf{h}_{i_1} + \cdots + c_{i_m}\mathbf{h}_{i_m} = \mathbf{0}.$$

Denoting $\mathbf{c} \in \mathrm{GF}(q)^n$ with $c_j = 0$ outside of $\{i_1, \ldots, i_m\}$, this is equivalent to $\mathbf{c}\mathbf{H}^\top = \mathbf{0}$, and in turn $\mathbf{c} \in \mathsf{C}$. □

E.g. for the $[5,1,5]$-repetition code,

$$\mathbf{H}_{\mathsf{R}} = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}.$$

Any set of four columns is linearly independent, but the set of all five columns is linearly dependent. Therefore, $d_{\min} = 5$.

23.3 See further

Tables of linear codes We do not know (and probably will never know) the largest minimum distance of a linear code for a given length and dimension. Tables of the best linear codes known so far are kept at `codetables` (see more links there).

MacWilliams identity One of the most remarkable results for linear codes is the MacWilliams identity, which relates the distance distribution of a linear code C to that of its dual $\mathsf{C}^{\perp}$. Let $A_0, A_1, \ldots, A_n$ denote the weight (or distance) distribution of C:

$$A_i = |\{\mathbf{c} \in \mathsf{C} : w_{\mathrm{H}}(\mathbf{c}) = i\}|.$$

(So in particular $A_0 = 1$, $A_1 = \cdots = A_{d_{\min}-1} = 0$.) Then construct the weight enumerator of C (a bivariate polynomial):

$$W_{\mathsf{C}}(x,y) = \sum_{i=0}^{n} A_i x^{n-i} y^i.$$

Theorem 23.5. *Let C be an $[n,k]_q$ linear code, then*

$$W_{\mathsf{C}^{\perp}}(x,y) = \frac{1}{|\mathscr{C}|} W_{\mathsf{C}}(x + (q-1)y, x - y)$$

Check it works for the repetition code/parity-check code pair!

23.4 Exercises

Exercise 23.1. Prove that two binary vectors $\mathbf{x}, \mathbf{y} \in \mathrm{GF}(2)^r$ are linearly independent if and only if they are nonzero and distinct.

Exercise 23.2 (Relating $\mathbf{G}$ and $\mathbf{H}$.)**.** Any linear code D is equivalent to a code C which has a generator matrix of the form $\mathbf{G}_{\mathsf{C}} = (\mathbf{I}_k | \mathbf{M})$, where $\mathbf{M} \in \mathrm{GF}(q)^{k \times n-k}$. Give a parity-check matrix for C.

Say that $\mathsf{D} = \mathsf{C}\mathbf{P}$, for some permutation matrix $\mathbf{P}$. Give a generator matrix and a parity-check matrix for D.

Exercise 23.3. Prove Lemma 23.4.

Exercise 23.4. Give the parameters (length, dimension, redundancy, minimum distance) of the code with the following parity-check matrix over GF(3).

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 1 & 2 & 0 & 1 \\ 0 & 0 & 2 & 1 & 1 & 2 & 0 \\ 1 & 1 & 1 & 1 & 1 & 2 & 2 \end{pmatrix}.$$

Is $\mathbf{v} = (0,1,1,1,1,1,2)$ a codeword?

24 Binary codes I: Hamming codes

24.1 Definition and properties

Definition For any $r \geq 2$, the Hamming code $\mathsf{H}(r)$ is a binary linear code with a parity-check matrix whose columns are all the integers from 1 to $2^r - 1$ in binary. Any two columns are linearly independent; the first three are linearly dependent. Therefore $d_{\min} = 3$.

Here are the first few Hamming codes.

- For $r = 2$, the $[3,1,3]_2$-repetition code:

$$\mathbf{H} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

- For $r = 3$, the $[7,4,3]_2$-Hamming code:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

- For $r = 4$, the $[15,11,3]_2$-Hamming code:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

In general, a Hamming code has parameters

- Length $n = 2^r - 1$
- Dimension $k = 2^r - r - 1$
- Minimum distance $d_{\min} = 3$
- Field order 2
- Redundancy $r = n - k$
- Rate $R = 1 - r/(2^r - 1)$
- Error correction capability $t = 1$

Generator matrix for $r = 3$

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Optimality of Hamming codes According to the sphere-packing bound, if C is a binary code of length n and error-correction capability $t \geq 1$, then

$$|\mathsf{C}| \leq \frac{2^n}{n+1}.$$

Theorem 24.1. *Hamming codes reach the sphere-packing bound.*

24.2 Encoding and decoding

Decoding Encoding is easy: use the generator matrix $\mathbf{G}$.

Decoder problem for any code:

- Input: a vector $\mathbf{v} \in \mathrm{GF}(q)^n$.
- Output: the unique codeword $\mathbf{c}$ at Hamming distance $\leq t$ from $\mathbf{v}$.

Remarkable property of the Hamming code: a vector $\mathbf{v} \in \mathrm{GF}(2)^n$ either is a codeword, or is at Hamming distance 1 from a unique codeword!

Example The source and destination use the $[7,4,3]_2$-Hamming code. The source wants to transmit the four-bit message

$$\mathbf{m} = (0,0,1,1).$$

The source encodes the message

$$\mathbf{c} = \mathbf{mG} = (1,0,0,0,0,1,1).$$

During transmission on the channel, the sixth bit is flipped. The receiver then obtains

$$\mathbf{v} = \mathbf{c} + \mathbf{e}_6 = (1,0,0,0,0,0,1).$$

Syndrome decoding The received vector $\mathbf{v}$ is either a codeword $\mathbf{c}$ or of the form $\mathbf{c} + \mathbf{e}_i$ for some i. If $\mathbf{v}$ is a codeword, we have $\mathbf{vH}^\top = \mathbf{0}$. Otherwise,

$$\begin{aligned}\mathbf{vH}^\top &= (\mathbf{c} + \mathbf{e}_i)\mathbf{H}^\top \\ &= \mathbf{cH}^\top + \mathbf{e}_i\mathbf{H}^\top \\ &= \mathbf{e}_i\mathbf{H}^\top \\ &= i^{th} \text{ column of } \mathbf{H}.\end{aligned}$$

Decoding: Compute the syndrome $\mathbf{vH}^\top$ to obtain i, and hence the correct codeword $\mathbf{v} + \mathbf{e}_i$.

Example For the $[7,4,3]_2$-Hamming code. Receive $\mathbf{v} = (1,0,0,0,0,0,1)$. Compute

$$\mathbf{vH}^\top = (1,1,0).$$

Then

$$i = 1 \times 4 + 1 \times 2 + 1 \times 0 = 6$$

and the codeword is $\mathbf{c} = \mathbf{v} + \mathbf{e}_6 = (1,0,0,0,0,1,1)$.

Recovering the original message Once we get the codeword $\mathbf{c}$, we still need to get the original message $\mathbf{m}$! This is easy, because our choice of generator matrix is systematic: remove the positions $1,2,4,\ldots,2^{r-1}$ from $\mathbf{c}$.

E.g.: $\mathbf{c} = (1,0,0,0,0,1,1)$ means that the original message is $\mathbf{m} = (0,0,1,1)$.

24.3 Perfect codes

A code that reaches the sphere-packing bound is called perfect. We have already encountered some perfect linear codes:

- The trivial code $\mathsf{C} = \mathrm{GF}(q)^n$, with error-correction capability $t = 0$;
- The repetition code $\mathsf{R}(n,q)$ for some values of n and q;
- One can argue that the trivial code $\mathsf{C} = \{\mathbf{0}\}$ (the zero-dimensional subspace) is also perfect, as there is only one decoding sphere encompassing the whole space;

- The binary Hamming codes.

In a theoretical *tour de force*, all perfect linear codes have been classified by Van Lint and Tietäväinen in 1973. Firstly, one can define Hamming codes over all finite fields, and those are perfect. Secondly, there are the two Golay codes:

1. the binary $[23,12,7]_2$ Golay code G_{23} with generator matrix

$$\mathbf{G}_{\mathsf{G}_{23}} = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&1&1&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&1&1&1&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&1&1&1&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&1&1&1&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&0&1&1&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&1&1&0&0&1&0&0&1&1&1\\
0&0&0&0&0&0&1&0&0&0&0&0&1&1&1&1&0&0&1&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&0&1&1&1&1&1&0&0&1&0&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1&1&1&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1&1&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&0&1&0&1&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&1&0&1&0&1&0&1
\end{pmatrix},$$

2. and the ternary $[11,6,5]_3$ Golay code G_{11} with generator matrix

$$\mathbf{G}_{\mathsf{G}_{11}} = \begin{pmatrix}
1&0&0&0&0&1&1&1&2&2&0\\
0&1&0&0&0&1&1&2&1&0&2\\
0&0&1&0&0&1&2&1&0&1&2\\
0&0&0&1&0&1&2&0&1&2&1\\
0&0&0&0&1&1&0&2&2&1&1
\end{pmatrix}.$$

And that's it!

24.4 See further

Weight distribution of Hamming codes The dual of the Hamming code $\mathsf{H}(r)$ is called the simplex code. The simplex code has a remarkable property: all codewords are equidistant! More precisely, the weight distribution of the simplex code $\mathsf{C} = \mathsf{H}(r)^\perp$ is

$$A_0 = 1, A_{2^{r-1}} = 2^r - 1.$$

So its weight enumerator is

$$W_{\mathsf{C}}(x,y) = x^{2^r-1} + (2^r-1)x^{2^{r-1}-1}y^{2^r}.$$

The MacWilliams identity then gives the weight enumerator of the Hamming code:

$$W_{\mathsf{H}(r)}(x,y) = W_{\mathsf{C}^\perp}(x,y) = 2^{-r}W_{\mathsf{C}}(x+y,x-y) = 2^{-r}\left\{(x+y)^{2^r-1} + (2^r-1)(x+y)^{2^{r-1}-1}(x-y)^{2^r}\right\},$$

whence one can readily calculate the weight distribution of $\mathsf{H}(r)$.

24.5 Exercises

Exercise 24.1. Give the whole codebook of the $[7,4,3]_2$-Hamming code.

Exercise 24.2. Give a generator matrix of the $[15,11,3]_2$-Hamming code.

Exercise 24.3. Decode the vector

$$\mathbf{v} = (1,1,1,1,1,1,1,0,0,1,0,0,0,0,0)$$

for the relevant Hamming code.

Exercise 24.4. Verify that all Hamming codes and both Golay codes are perfect.

Exercise 24.5. For which values of n and q is the repetition code $\mathsf{R}(n,q)$ a perfect code?

25 Binary codes II: Reed-Muller codes

25.1 Definition

Boolean functions We can express logical functions in GF(2): let $x, y \in \mathrm{GF}(2)$

$$\begin{aligned} x \wedge y &= xy \\ x \vee y &= x + y + xy \\ \neg x &= x + 1 \\ x \oplus y &= x + y. \end{aligned}$$

A Boolean function is any function $f : \mathrm{GF}(2)^m \to \mathrm{GF}(2)$.

Polynomial expansion of Boolean functions

Theorem 25.1. *Any Boolean function $f(x_1, \ldots, x_m)$ is a polynomial of the variables $x_1, \ldots, x_m$.*

We can then talk about the degree of any Boolean function f. Explicitly, a Boolean function f is a linear combination of monomials, i.e. polynomials of the form

$$x_I = x_{i_1} x_{i_2} \ldots x_{i_d}$$

for any subset $I = \{i_1, \ldots, i_d\} \subseteq \{1, \ldots m\}$. The quantity $d = |I|$ is the degree of the monomial and the degree of f is the maximum degree of its monomials. (The monomial $x_\emptyset = 1$, corresponding to the empty set, has degree zero.)

We shall list monomials according to the so-called standard order, as follows. It's sort of a "reverse lexicographic order," which is best explained by an example, say with four variables:

$$\begin{aligned} &1; \\ &x_4, x_3, x_2, x_1; \\ &x_3x_4, x_2x_4, x_1x_4, x_2x_3, x_1x_3, x_1x_2; \\ &x_2x_3x_4, x_1x_3x_4, x_1x_2x_4, x_1x_2x_3; \\ &x_1x_2x_3x_4. \end{aligned}$$

We can represent any f by the vector of all the values it takes. For any $a = (a_1, \ldots, a_m) \in \mathrm{GF}(2)^m$, its lexicographic index is

$$l(a) := \sum_{i=1}^{m} a_i 2^{i-1} \quad (\text{in } \mathbb{N}).$$

We associate a and $l(a)$. We then define the evaluation vector

$$\mathbf{f} = (f(0), f(1), \ldots, f(2^m - 1)) \in \mathrm{GF}(2)^{2^m}$$

Example 25.2. *Let $m = 4$ and*

$$f(x_1, x_2, x_3, x_4) = x_1x_2 + x_2x_4 + x_1.$$

Its evaluation vector is

$$\begin{aligned} \mathbf{f} = \big(&f(0000), f(1000), f(0100), f(1100), f(0010), f(1010), f(0110), f(1110); \\ &f(0001), f(1001), f(0101), f(1101), f(0011), f(1011), f(0111), f(1111)\big) \\ = (&0,1,0,0,0,1,0,0;0,1,1,1,0,1,1,1). \end{aligned}$$

Definition of Reed-Muller codes The r-th order binary Reed-Muller (or RM) code $\mathsf{RM}(r,m)$ of length $n = 2^m$, for $0 \le r \le m$, is the set of all vectors $\mathbf{f}$, where $f(x_1,\ldots,x_m)$ is a Boolean function of degree at most r.

Lemma 25.3. $\mathsf{RM}(r,m)$ *is a linear code.*

Proof. If $\mathbf{f},\mathbf{g} \in \mathsf{RM}(r,m)$, then $\deg f, \deg g \le r$ and hence $\deg(f+g) \le r$ and $\mathbf{f}+\mathbf{g} \in \mathsf{RM}(r,m)$. □

Alternatively, the set of polynomials of degree at most r form a vector space. Its basis is given by all the monomials:

$$1, \quad x_m,\ldots,x_1, \quad x_m x_{m-1},\ldots,x_1x_2, \quad \ldots, \quad x_1\cdots x_m.$$

This shows that the dimension of $\mathsf{RM}(r,m)$ is

$$k = \sum_{d=0}^{r} \binom{m}{d}.$$

The family of Reed-Muller codes contains some of the codes we have previously encountered:

- $\mathsf{RM}(0,m)$ is the $[2^m,1,2^m]$-repetition code.
- $\mathsf{RM}(m,m) = \mathrm{GF}(2)^{2^m}$ is the $[2^m,2^m,1]$-trivial code.
- $\mathsf{RM}(m-1,m)$ is the $[2^m,2^m-1,2]$-parity check code.

Example Let $\mathbf{G}$ be a generator matrix of $\mathsf{RM}(1,3)$. The space of polynomials in 3 variables of degree at most 1 has the following basis (in standard order):

$$1, \quad x_3, \quad x_2, \quad x_1.$$

The corresponding vectors are the rows of $\mathbf{G}$:

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

This is the standard generator matrix for $\mathsf{RM}(1,3)$.

Parameters of RM codes Parameters of $\mathsf{RM}(r,m)$:

- Length $n = 2^m$
- Dimension $k = \sum_{d=0}^{r} \binom{m}{d}$
- Minimum distance $d_{\min} = 2^{m-r}$
- Field order 2
- Redundancy $n-k = \sum_{d=0}^{m-r} \binom{m}{d}$
- Rate $R = k/n$
- Error-correction capability $t = 2^{m-r-1}-1$ if $r < m$ and $t = 0$ if $r = m$

Note: r is not the redundancy of $\mathsf{RM}(r,m)$ (this is an unfortunate clash of notation conventions).

Parity-check matrix Easily enough, the dual of a Reed-Muller code is another Red-Muller code!

Theorem 25.4. $\mathsf{RM}(r,m)^\perp = \mathsf{RM}(m-r-1,m)$ *for* $0 \le r \le m-1$.

The proof of Theorem 25.4 is non-examinable. Let $\mathbf{a} \in \mathsf{RM}(m-r-1,m)$, $\mathbf{b} \in \mathsf{RM}(r,m)$ and $a(v_1,\ldots,v_m)$ and $b(v_1,\ldots,v_m)$ be the corresponding polynomials. We have $\deg a \le m-r-1$ and $\deg b \le r$, hence $\deg(ab) \le m-1$ and $\mathbf{ab} \in \mathsf{RM}(m-1,m)$ has even weight. Therefore, $\mathbf{a}\cdot\mathbf{b} = 0$ and $\mathsf{RM}(m-r-1,m) \subseteq \mathsf{RM}(r,m)^\perp$. Since

$$\dim \mathsf{RM}(m-r-1,m) = n - \dim \mathsf{RM}(r,m),$$

we obtain equality. □

(Note that for $r=m$, $\mathsf{RM}(m,m)$ is the whole space $\mathrm{GF}(2)^{2^m}$, whose dual is the trivial space of dimension zero.) Therefore,

$$\mathbf{H}_{\mathsf{RM}(r,m)} = \mathbf{G}_{\mathsf{RM}(m-r-1,m)}.$$

25.2 Reed decoding

Standard generator matrix The Reed decoding algorithm only works for the standard generator matrix. It is recursive, in the sense that the algorithm for $\mathsf{RM}(r,m)$ first determines the coordinates of the plaintext in the subspace generated by the monomial of degree r, and then apply the Reed decoding algorithm for $\mathsf{RM}(r-1,m)$. Decoding $\mathsf{RM}(0,m)$ is straightforward!

We illustrate it by studying $\mathsf{RM}(2,4)$: the $[16,11,4]$ second order RM code of length 16.

$\mathbf{1}$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\mathbf{x}_4$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$\mathbf{x}_3$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
$\mathbf{x}_2$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
$\mathbf{x}_1$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\mathbf{x}_3\mathbf{x}_4$	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
$\mathbf{x}_2\mathbf{x}_4$	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
$\mathbf{x}_1\mathbf{x}_4$	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
$\mathbf{x}_2\mathbf{x}_3$	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
$\mathbf{x}_1\mathbf{x}_3$	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1
$\mathbf{x}_1\mathbf{x}_2$	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
$\mathbf{x}_2\mathbf{x}_3\mathbf{x}_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$\mathbf{x}_1\mathbf{x}_3\mathbf{x}_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
$\mathbf{x}_1\mathbf{x}_2\mathbf{x}_4$	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
$\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
$\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3\mathbf{x}_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Reed decoding The message vector

$$\mathbf{m} = (m_\emptyset, m_{\{4\}}, m_{\{3\}}, m_{\{2\}}, m_{\{1\}}, m_{\{3,4\}}, m_{\{2,4\}}, m_{\{1,4\}}, m_{\{2,3\}}, m_{\{1,3\}}, m_{\{1,2\}})$$

is encoded to

$$\begin{aligned}\mathbf{c} &= \mathbf{mG}\\ &= m_\emptyset\mathbf{1} + m_{\{4\}}\mathbf{x}_4 + \cdots + m_{\{1\}}\mathbf{x}_1 + \cdots + m_{\{1,2\}}\mathbf{x}_1\mathbf{x}_2\\ &= (c_0, c_1, \ldots c_{15}).\end{aligned}$$

This is a single-error-correcting code ($t=1$). We will decode the error by majority logic decoding.

Majority logic decoding The first step is to recover the terms of highest degree $m_{\{1,2\}},\dots,m_{\{3,4\}}$. If there are no errors, then

$$\begin{aligned}
m_{\{1,2\}} &= c_0 + c_1 + c_2 + c_3 \\
&= c_4 + c_5 + c_6 + c_7 \\
&= c_8 + c_9 + c_{10} + c_{11} \\
&= c_{12} + c_{13} + c_{14} + c_{15}, \\
m_{\{1,3\}} &= c_0 + c_1 + c_4 + c_5 \\
&= c_2 + c_3 + c_6 + c_7 \\
&= c_8 + c_9 + c_{12} + c_{13} \\
&= c_{10} + c_{11} + c_{14} + c_{15}. \\
&\vdots \\
m_{\{3,4\}} &= c_0 + c_4 + c_8 + c_{12} \\
&= c_1 + c_5 + c_9 + c_{13} \\
&= c_2 + c_6 + c_{10} + c_{14} \\
&= c_3 + c_7 + c_{11} + c_{15}.
\end{aligned}$$

We obtain 4 votes for $m_{\{1,2\}}$, 4 votes for $m_{\{1,3\}}$, etc. So if one error occurs, the majority vote is still correct, and thus each $m_{\{i,j\}}$ is obtained correctly.

To find the symbols $m_{\{1\}},\dots,m_{\{4\}}$, let

$$\begin{aligned}
\mathbf{v}' &= \mathbf{v} - (m_{\{3,4\}}\mathbf{x}_3\mathbf{x}_4 + \cdots + m_{\{1,2\}}\mathbf{x}_1\mathbf{x}_2) \\
&= \mathbf{v}'_0 \dots \mathbf{v}'_{15}.
\end{aligned}$$

Again, if there is no error, we have

$$\begin{aligned}
m_{\{1\}} &= c'_0 + c'_1 \\
&= c'_2 + c'_3 \\
&\vdots \\
&= c'_{14} + c'_{15},
\end{aligned}$$

and so on for every $1 \le i \le 4$. We have 8 votes for each $m_{\{i\}}$, and hence majority still works.

Finally, determining $m_\emptyset$ is easy:

$$\mathbf{v}'' = \mathbf{v}' - (m_{\{4\}}\mathbf{x}_4 + \cdots + m_{\{1\}}\mathbf{x}_1) = m_\emptyset \mathbf{1} + \text{error},$$

and $m_\emptyset = 0$ or 1 depending on the number of ones in x".

A little more formally; non-examinable To find the terms of highest degree, we have to perform the following summations. We denote any vector $\mathbf{v} \in \mathrm{GF}(2)^{2^m}$ as

$$\mathbf{v} = \left(v_0 = v_{(0,\dots,0)},\, v_1 = v_{(1,0,\dots,0)},\, \dots,\, v_{2^m-1} = v_{(1,\dots,1)}\right).$$

For any subset $\sigma = \{\sigma_1,\dots,\sigma_r\} \subseteq \{1,\dots,m\}$, consider the subspace

$$S(\sigma) = \left\{\mathbf{s} \in \mathrm{GF}(2)^m : s_{\sigma_i} = 0 \,\forall 1 \le i \le r\right\}.$$

Let $\tau = \{1,\dots,m\} \setminus \sigma$.

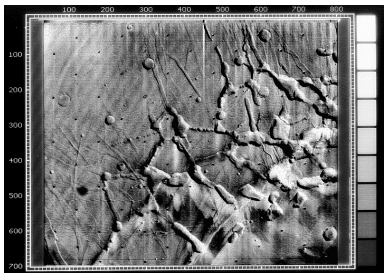
Theorem 25.5. *If there are no errors, m_σ is given by*

$$m_\sigma = \sum_{\mathbf{t} \in S(\tau)} c_{\mathbf{t}+\mathbf{s}} \quad \forall \mathbf{s} \in S(\sigma).$$

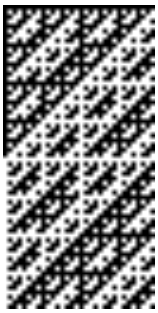
This equation gives 2^{m-r} votes for m_σ, one for each value of $\mathbf{s}$.

25.3 Applications

Mariner 9 was a space probe, launched in 1971 to photograph Mars from 900 miles altitude. The resolution was 1 pixel per 100m. A typical picture looked like this.



Data words (pixels) were 6 bits long (64 shades of grey). Limits of directional antenna meant transmissions were limited to about 30 bits. The $[5,1,5]_2$-repetition code was considered as it is easy to implement and corrects 2 errors. Instead a $[32,6,16]$-code was used, namely the Reed-Muller code $\mathsf{RM}(1,5)$. This can correct 7 errors for about the same data rate! The codebook is displayed below, where black is 1 and white is 0 or vice versa.



25.4 See further

Hadamard codes Here is a construction of the code $\mathsf{RM}(1,m)$, by listing all codewords. Recall the Walsh-Hadamard transform from Exercise 6.1 defined for all $n = 2^m$. From that matrix $\mathbf{H}_n \in \mathbb{R}^{n\times n}$, obtain a new matrix $\mathbf{H}'_n \in \mathrm{GF}(2)^{2n\times n}$ as follows. First, remove the normalisation constant, so we are left with a $\{+1,-1\}$ matrix; second, replace $+1 \mapsto 0$ and $-1 \mapsto 1$; third, extend the matrix by adding n rows, where row $N+i$ is the complement of row i. For instance, for $m = 2$, we obtain:

$$\mathbf{H}_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, \qquad \mathbf{H}'_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

Note that the orthogonality property of $\mathbf{H}_n$ implies that

$$d_{\mathrm{H}}(\mathbf{c},\mathbf{c}') = 2^{m-1}$$

for all distinct codewords of $\mathsf{RM}(1,m)$. In turn, this means that all codewords, apart from the all-zero and the all-one codewords, of the Hadamard code have weight $n/2$!

Nonlinearity and Hadamard codes We encountered the nonlinearity of a Boolean function as a measure of its security when used in an S-box. The nonlinearity of $f : \mathrm{GF}(2)^m \to \mathrm{GF}(2)$ can be defined as its distance to the Hadamard code:

$$N(f) = \min\{d_{\mathrm{H}}(\mathbf{f}, \mathbf{c}) : \mathbf{c} \in \mathsf{RM}(1, m)\}.$$

A function with highest nonlinearity (for m even, as it's given by $2^{m-1} - 2^{m/2-1}$) is called bent. An early example of bent function is

$$f(x_1, \ldots, x_m) = x_1 x_2 + \ldots x_{m-1} x_m.$$

25.5 Exercises

Exercise 25.1. Prove that any Boolean function is a polynomial. What is the polynomial form of

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge \neg x_4) \vee (x_3 \wedge x_4)?$$

Exercise 25.2. Prove directly (i.e. without using the dual) that $\mathsf{RM}(m-1, m)$ is a parity-check code.

Exercise 25.3. Use Reed decoding for $\mathsf{RM}(2, 4)$ to decode

$$\mathbf{v} = (0,1,1,0,0,0,1,0,0,1,1,0,1,0,0,1).$$

Comment on the number of votes you need for the decoding.

26 Reed-Solomon codes I: Cyclic codes

26.1 Cyclotomic cosets and minimal polynomials

Conjugacy class Let $\beta \in \mathrm{GF}(q^m)$. The conjugates of β w.r.t. $\mathrm{GF}(q)$ are the elements $\beta, \beta^q, \beta^{q^2}, \ldots$. The set is called the conjugacy class of β. The conjugacy class of β contains d elements, where $\beta^{q^d} = \beta$ and $d \mid m$. It is easy to see that for any polynomial $p(x)$ in $\mathrm{GF}(q)[x]$, if β is a root of $p(x)$, then all its conjugates are roots also; indeed, if $p(\beta) = 0$ then

$$p(\beta^q) = \sum_i p_i \beta^{i \cdot q} = \sum_i p_i^q \beta^q = \left(\sum_i p_i \beta^i \right)^q = p(\beta)^q = 0.$$

Minimal polynomial The minimal polynomial of β w.r.t. $\mathrm{GF}(q)$ is the smallest-degree monic nonzero polynomial $p(x)$ in $\mathrm{GF}(q)[x]$ such that $p(\beta) = 0$. (Monic means that the leading coefficient is equal to 1.) We denote it $M_\beta(x)$ or $M_t(x)$ where $\beta = \alpha^t$. The roots of the minimal polynomial of β w.r.t. $\mathrm{GF}(q)$ are then exactly the conjugates of β w.r.t. $\mathrm{GF}(q)$. That is,

$$M_t(x) = (x - \beta)(x - \beta^q) \cdots (x - \beta^{q^{d-1}}).$$

Example for GF(8) We are using the primitive polynomial $x^3 + x + 1$ over $\mathrm{GF}(2)$ (so here $q = 2$ and $q^m = 2^3 = 8$).

Conjugacy class	Minimal polynomial
$\{0\}$	$M_*(x) = (x - 0) = x$
$\{\alpha^0 = 1\}$	$M_0(x) = (x - 1) = x + 1$
$\{\alpha, \alpha^2, \alpha^4\}$	$M_1(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4) = x^3 + x + 1$
$\{\alpha^3, \alpha^6, \alpha^5\}$	$M_3(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^5) = x^3 + x^2 + 1$

Factoring $x^n - 1$ The set of nonzero elements of in $\mathrm{GF}(q^m)$ form the complete set of roots of the polynomial $x^{q^m - 1} - 1$. The order of q modulo n is the smallest integer m such that n divides $q^m - 1$.If m is the order of q modulo n, then $\mathrm{GF}(q^m)$ is the smallest extension field of $\mathrm{GF}(q)$ which contains the primitive n-th roots of unity.

We will focus on $n = q^m - 1$. Then factoring $x^{q^m - 1} - 1$ is done by computing the minimal polynomials in $\mathrm{GF}(q^m)$, e.g.

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1) \quad \text{in } \mathrm{GF}(2)[x].$$

26.2 Definition

Definition of cyclic codes An $[n,k]$ linear code C is cyclic if for every codeword $\mathbf{c}=(c_0,c_1,\ldots,c_{n-1})\in\mathsf{C}$, its cyclic shift $\mathbf{c}'=(c_{n-1},c_0,\ldots,c_{n-2})$ is also in C.

The key to understand cyclic codes is by viewing each codeword $\mathbf{c}$ as a code polynomial:

$$c(x)=c_0+c_1x+\cdots+c_{n-1}x^{n-1}.$$

Then the cyclic shift corresponds to multiplying by $x \mod (x^n-1)$:

$$xc(x)=c'(x) \mod (x^n-1).$$

Algebraic definition We place ourselves in $\mathrm{GF}(q)[x]/(x^n-1)$. Let $c(x)$ be a codeword in C, then

$$xc(x),\quad x^2c(x),\quad \ldots,\quad x^{n-1}c(x),\quad x^nc(x)=c(x)\in\mathsf{C}.$$

Since the code is linear, we obtain that for any $a(x)$, $a(x)c(x)\in\mathsf{C}$. Thus C is an ideal of $\mathrm{GF}(q)[x]/(x^n-1)$.

Theorem 26.1. *C is a q-ary cyclic code of length n if and only if the code polynomials form an ideal in $\mathrm{GF}(q)[x]/(x^n-1)$.*

Classification of cyclic codes

Theorem 26.2. *Let C be an $[n,k]_q$ cyclic code.*

1. *There exists a unique monic polynomial $g(x)$ with minimal degree $r=n-k<n$. $g(x)$ is the generator polynomial of C.*
2. *Every code polynomial $c(x)\in\mathsf{C}$ can be expressed uniquely as*
$$c(x)=m(x)g(x),$$
where $m(x)$ is the message polynomial, of degree $\deg m(x)<k$.
3. *$g(x)$ is a factor of x^n-1 in $\mathrm{GF}(q)[x]$.*

Generator matrix The generator matrix of C with generator polynomial $g(x)=\sum_{i=0}^{r}g_ix^i$ is easy to determine:

$$\mathbf{G}=\begin{pmatrix} g_0 & g_1 & \cdots & g_r & & & \mathbf{0} \\ & g_0 & g_1 & \cdots & g_r & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & g_0 & g_1 & \cdots & g_r \\ \mathbf{0} & & & & g_0 & g_1 & \cdots & g_r \end{pmatrix}.$$

Indeed, $c(x)=m(x)g(x)$ corresponds to $\mathbf{c}=\mathbf{mG}$.

Parity-check matrix There exists $h(x)\in\mathrm{GF}(q)[x]$ such that

$$g(x)h(x)=x^n-1 \quad \text{in } \mathrm{GF}(q)[x].$$

This is the parity polynomial of C. Then $c(x)\in\mathsf{C}$ if and only if

$$c(x)h(x)=0 \quad \text{in } \mathrm{GF}(q)[x]/(x^n-1).$$

In matrix form, this is equivalent to $\mathbf{cH}^\top=\mathbf{0}$, where

$$\mathbf{H}=\begin{pmatrix} h_k & \ldots & h_1 & h_0 & & & \mathbf{0} \\ & h_k & \ldots & h_1 & h_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & h_k & \ldots & h_1 & h_0 \\ \mathbf{0} & & & & h_k & \ldots & h_1 & h_0 \end{pmatrix}.$$

In particular, this shows that the dual of a cyclic code is another cyclic code. More precisely, for a polynomial $f(x) = \sum_{i=0}^{d} f_i x^i$ of degree d, its reciprocal is

$$f^*(x) := x^d f(x^{-1}) = \sum_{i=0}^{d} f_{d-i} x^i.$$

Then if C is the cyclic code generated by $g(x)$, then $C^\perp$ is the cyclic code generated by $h^*(x)$, the reciprocal of the parity polynomial of C.

26.3 Examples

Trivial examples Firstly, what happens when we choose $g(x) = 1$? $g(x) = x^n - 1$?

Secondly, we can always factor $x^n - 1$ as

$$x^n - 1 = (x-1)(1 + x + \cdots + x^{n-1}).$$

The $[n, 1, n]$-repetition code is cyclic, with generator polynomial

$$g_{\text{repetition}}(x) = 1 + x + \cdots + x^{n-1}.$$

The $[n, n-1, 2]$-parity-check code is cyclic, with generator polynomial

$$g_{\text{parity}}(x) = x - 1.$$

Binary cyclic codes of length 7 We need to factor $x^7 - 1$ in GF(2)[x]. Let α be a root of of the primitive polynomial of $x^3 + x + 1$ (this generates GF(8)). The conjugacy classes and respective minimal polynomials are as follows (recall that $\alpha^7 = 1$)

$$\{1\} : x + 1$$
$$\{\alpha, \alpha^2, \alpha^4\} : x^3 + x + 1$$
$$\{\alpha^3, \alpha^6, \alpha^5\} : x^3 + x^2 + 1.$$

Then any generator polynomial $g(x)$ is the product of some of these three polynomials. There are eight possibilities.

26.4 Applications

Burst error detection Cyclic codes are very useful to detect burst-errors. A burst-error of length b is a polynomial of the form

$$e(x) = \sum_{i=j}^{j+b-1} e_i x^i, \quad e_j \neq 0, \quad e_{j+b-1} \neq 0$$

(where indices are computed cyclically).

A cyclic code can detect

- all burst-errors of length r or less,
- the fraction $1 - \frac{q^{1-r}}{q-1}$ of all burst-errors of length $r+1$,
- the fraction $1 - q^{-r}$ of all burst-errors of length $b > r+1$.

CRC Cyclic Redundancy Check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. This is a binary linear code, with encoding in systematic form. It is technically not a cyclic code, but nearly!

The encoding is straightforward and based on long division of polynomials. Let r be the redundancy of the code (typically 32) and let $g(x)$ be a monic polynomial of degree r (typically, we want $g(x)$ to be a multiple of $x+1$). Let $m(x)$ be the message polynomial, and let $R(x) = x^r m(x) \mod g(x)$ be the remainder of the long division of $x^r m(x)$ by $g(x)$ (so that $\deg R(x) < r$). Then the codeword is

$$c(x) = x^r m(x) + R(x);$$

which basically means appending $R(x)$ to $m(x)$.

Some of the key properties of cyclic codes, notably burst-error detection, are kept by CRC. Moreover, the encoding and decoding are extremely fast, which makes CRC highly efficient. Places where CRC is used include data compression files such as Zip, PNG, MPEG-2, but also network protocols such as IEEE 802.3 (Ethernet).

26.5 Exercises

Exercise 26.1. List the conjugacy classes and the according minimal polynomials of GF(16) with respect to GF(2).

Exercise 26.2. List the conjugacy classes and the according minimal polynomials of GF(16) with respect to GF(4).

Exercise 26.3. Prove that the $[7,4,3]$-Hamming code is equivalent to a cyclic code. What is the generator polynomial?

Exercise 26.4. For all eight cyclic codes of length 7 over GF(2), give their dimension and minimum distance.

27 Reed-Solomon codes II: BCH and RS codes

27.1 BCH codes

The BCH bound The Bose–Ray-Chaudhuri–Hocquenghem (BCH) bound is a lower bound on the minimum distance of cyclic codes where the generator polynomial has consecutive roots.

Theorem 27.1 (BCH bound)**.** *Let* C *be a cyclic code with generator polynomial* $g(x)$ *such that for some* $b \geq 0$ *and* $\delta \geq 2$*,*

$$g(\alpha^b) = g(\alpha^{b+1}) = \cdots = g(\alpha^{b+\delta-2}) = 0.$$

Then $d_{\min}(\mathsf{C}) \geq \delta$*.*

The parameter δ is called the design distance of C.

Proof. If $c(x) \in \mathsf{C}$, then $c(\alpha^b) = \cdots = c(\alpha^{b+\delta-2}) = 0$. Hence the parity-check matrix of C contains the following $\delta - 1$ rows:

$$\mathbf{H}' = \begin{pmatrix} 1 & \alpha^b & \dots & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & \dots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \alpha^{b+\delta-2} & \dots & \alpha^{(n-1)(b+\delta-2)} \end{pmatrix}.$$

Consider any set $\{a_1, \dots, a_{\delta-1}\}$ of $\delta - 1$ columns of $\mathbf{H}'$, we obtain a square matrix with

$$\det \begin{pmatrix} \alpha^{a_1 b} & \dots & \alpha^{a_{\delta-1} b} \\ \alpha^{a_1 (b+1)} & \dots & \alpha^{a_{\delta-1}(b+1)} \\ \vdots & \dots & \vdots \\ \alpha^{a_1(b+\delta-2)} & \dots & \alpha^{a_{\delta-1}(b+\delta-2)} \end{pmatrix} = \alpha^{(a_1+\cdots+a_{\delta-1})b} \det \begin{pmatrix} 1 & \dots & 1 \\ \alpha^{a_1} & \dots & \alpha^{a_{\delta-1}} \\ \vdots & \dots & \vdots \\ \alpha^{(\delta-2)a_1} & \dots & \alpha^{(\delta-2)a_{\delta-1}} \end{pmatrix}.$$

The last matrix is (the transpose of) a Vandermonde matrix; its determinant is

$$\prod_{i \neq j} (\alpha^{a_i} - \alpha^{a_j}) \neq 0.$$

Therefore any set of $\delta - 1$ columns of $\mathbf{H}'$ are linearly independent, and by extension any set of $\delta - 1$ columns of $\mathbf{H}$ are linearly independent. □

BCH codes A cyclic code of length $n = q^m - 1$ over GF(q) is a BCH code of design distance δ if

$$g(x) = \operatorname{lcm}\{M_b(x), \ldots, M_{b+\delta-2}(x)\}.$$

Notes: Technically, these are called primitive BCH codes, because in general we can construct BCH codes for any $n \mid q^m - 1$. If $b = 1$, the BCH code is called narrow-sense.

27.2 Reed-Solomon codes

Reed-Solomon codes as BCH codes A Reed-Solomon code is a q^m-ary BCH code of length $q^m - 1$.

RS(n,k) is the narrow-sense Reed-Solomon code of length n and dimension k (this is a code over GF(q^m), where $n = q^m - 1$). In GF(q^m), the cyclotomic cosets modulo $q^m - 1$ are singletons, so

$$M_i(x) = x - \alpha^i \quad 0 \leq i \leq q^m - 2.$$

Thus, the generator polynomial of RS(n,k) is

$$g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{n-k}).$$

In general, the dual of a Reed-Solomon code is also a Reed-Solomon code.

Parity-check matrix of RS(n,k) For Reed-Solomon codes, the $\mathbf{H}'$ in the proof of the BCH bound is the whole parity-check matrix:

$$\mathbf{H}_{\mathrm{RS}(n,k)} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \ldots & \alpha^{(n-1)(n-k)} \end{pmatrix}.$$

E.g. RS(7,3) is a code over GF(8) with parity-check matrix

$$\mathbf{H}_{\mathrm{RS}(7,3)} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \\ 1 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha & \alpha^4 \\ 1 & \alpha^4 & \alpha & \alpha^5 & \alpha^2 & \alpha^6 & \alpha^3 \end{pmatrix}.$$

Minimum distance of Reed-Solomon codes

Theorem 27.2 (Singleton bound)**.** *For any* $[n,k]$ *linear code* C*,*

$$d_{\min}(\mathsf{C}) \leq n - k + 1.$$

Proof. Any set of $n - k + 1$ columns of $\mathbf{H}$ must be linearly dependent. □

A code reaching the Singleton bound is called Maximum Distance Separable (MDS). The BCH bound proves that Reed-Solomon codes are MDS.

Theorem 27.3. RS(n,k) *is an MDS code, i.e.*

$$d_{\min} = \delta = n - k + 1.$$

Parameters of Reed-Solomon codes Parameters of $\mathsf{RS}(n,k)$:

- Length n
- Dimension k
- Minimum distance $d_{\min} = n-k+1$
- Field order $q^m = n+1$
- Redundancy $n-k = d_{\min}-1$
- Rate $R = k/n$
- Error-correction capability $t = \left\lfloor \frac{n-k+1}{2} \right\rfloor$

Reed-Solomon codes as polynomial evaluations For any polynomial $f(x)$ with coefficients in $\mathrm{GF}(q^m)$, let

$$\mathbf{f} = (f(1), f(\alpha), \ldots, f(\alpha^{n-1})) \in \mathrm{GF}(q^m)^n$$

be its evaluation vector.

Theorem 27.4. *The narrow-sense code $\mathsf{RS}(n,k)$ is the set of all evaluations of polynomials of degree at most $k-1$ over all nonzero elements of $\mathrm{GF}(q^m)$, where $n = q^m - 1$:*

$$\mathsf{RS}(n,k) = \{\mathbf{f} : \deg f(x) \le k-1\}.$$

This yields the following generator matrix:

$$\mathbf{G}_{\mathsf{RS}(n,k)} = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ 1 & \alpha & \ldots & \alpha^{n-1} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \alpha^{k-1} & \ldots & \alpha^{(k-1)(n-1)} \end{pmatrix}.$$

E.g. for $\mathsf{RS}(7,3)$,

$$\mathbf{G}_{\mathsf{RS}(7,3)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \end{pmatrix}.$$

E.g. encoding $(0, \alpha, \alpha^2)$, i.e. $f(x) = \alpha^2 + \alpha x$:

$$\begin{aligned} \mathbf{f} &= (f(1), f(\alpha), f(\alpha^2), f(\alpha^3), f(\alpha^4), f(\alpha^5), f(\alpha^6)) \\ &= (\alpha^4, 0, \alpha^5, \alpha, \alpha^3, 1, \alpha^6). \end{aligned}$$

27.3 Applications

Reed-Solomon codes are arguably the most used family of codes in the recent history of error-correcting codes. Even though they are being superseded in some applications, they are still going strong to this day. We mention a few applications below.

Data storage Reed–Solomon codes are very widely used in mass storage systems to correct the burst errors associated with media defects. In particular, Reed–Solomon codes are a key component of the compact disc. It was the first use of strong error correction coding in a mass-produced consumer product, and DVD uses a similar scheme. In the CD, two layers of Reed–Solomon coding separated by a 28-way convolutional interleaver yields a scheme called Cross-Interleaved Reed–Solomon Coding (CIRC).

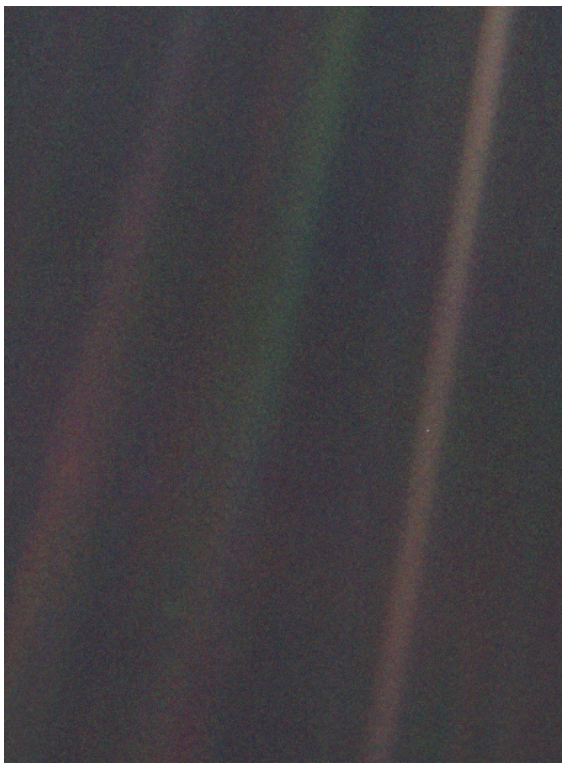
The first element of a CIRC decoder is a relatively weak inner [32,28]-Reed–Solomon code, shortened from a [255,251]-code with 8-bit symbols (i.e. over $\mathrm{GF}(2^8)$). This code can correct up to 2 byte errors per 32-byte block. More importantly, it flags as erasures any uncorrectable blocks, i.e., blocks with

more than 2 byte errors. The decoded 28-byte blocks, with erasure indications, are then spread by the deinterleaver to different blocks of the [28,24] outer code. Thanks to the deinterleaving, an erased 28-byte block from the inner code becomes a single erased byte in each of 28 outer code blocks. The outer code easily corrects this, since it can handle up to 4 such erasures per block.

The result is a CIRC that can completely correct error bursts up to 4,000 bits, or about 2.5 mm on the disc surface. This code is so strong that most CD playback errors are almost certainly caused by tracking errors that cause the laser to jump track, not by uncorrectable error bursts.

Bar code Almost all two-dimensional bar codes such as PDF-417, MaxiCode, Datamatrix, QR Code, and Aztec Code use Reed–Solomon error correction to allow correct reading even if a portion of the bar code is damaged. When the bar code scanner cannot recognize a bar code symbol, it will treat it as an erasure.

Space transmission Voyager introduced Reed–Solomon coding concatenated with convolutional codes, a practice that has since become very widespread in deep space and satellite (e.g., direct digital broadcasting) communications. In particular, the famous Pale Blue Dot picture was brought to you by a Reed-Solomon code.



Modern versions of concatenated Reed–Solomon/Viterbi-decoded convolutional codes were and are used on the Mars Pathfinder, Galileo, Mars Exploration Rover and Cassini missions, where they perform within about 1–1.5 dB of the theoretical limit called the Shannon capacity. Nonetheless, these concatenated codes are now being replaced by more powerful turbo codes.

27.4 See further

Refinements of the BCH bound Many lower bounds on the minimum distance of cyclic codes, that refine the BCH bound, have been found over the years: by Hartmann-Tzeng, Roos, etc.

MDS codes Other constructions of MDS codes have been found, including extended Reed-Solomon codes, Generalized Reed-Solomon codes, Gabidulin codes, etc. The main limitation of MDS codes is that they are short. For instance, $\mathrm{RS}(n,k)$ has $n \leq q-1$. In fact, the MDS conjecture, given below, is widely considered to be true.

MDS conjecture. If an $(n,k,d=n-k+1)_q$-MDS code exists, then one of the following holds:

1. $k \le 1$ or $k \ge n-1$;
2. q is even, $n = q+2$ and $k \in \{3, q-1\}$;
3. $n \le q+1$.

Gabidulin codes Recall the rank metric from Lecture 21. The whole theory of cyclic codes, BCH codes and Reed-Solomon codes (both viewed as BCH codes and evaluations) can be adapted to the case of so-called linearized polynomials. These basically correspond to linear functions $f : \mathrm{GF}(q^m) \to \mathrm{GF}(q^m)$ of the form $f(x) = \mathbf{M}x$ where x is viewed as a vector in $\mathrm{GF}(q)^m$. In particular, Gabidulin gave an EEA decoding algorithm for these codes.

27.5 Exercises

Exercise 27.1. Give the generator matrix and the parity-check matrix of $\mathsf{RS}(7,5)$.

Exercise 27.2. Prove that Reed-Solomon codes are MDS, by viewing them as polynomial evaluations.

Exercise 27.3. Prove that if C is MDS, then so is $\mathsf{C}^\perp$.

28 Reed-Solomon codes III: Decoding

28.1 The Extended Euclidean Algorithm

Polynomials and the gcd's $\mathrm{GF}(q)[x]$: Set of polynomials over the field $\mathrm{GF}(q)$. Just like integers, polynomials over $\mathrm{GF}(q)$ have a long division.

If $a(x), b(x) \in \mathrm{GF}(q)[x]$ with $\deg a(x) > \deg b(x)$, then

$$a(x) = q(x) \cdot b(x) + r(x),$$

where $\deg r(x) < \deg b(x)$. $q(x)$ is called the quotient and $r(x)$ the remainder. We can then use the Euclidean algorithm to determine the gcd between two polynomials.

Euclidean algorithm Input: $a(x), b(x) \in \mathrm{GF}(q)[x]$ with $\deg(a) > \deg(b)$
Output: $\gcd(a,b)$

Algorithm 5 EUCLID(a,b)

```
if b == 0 then
    return a
else
    return EUCLID(b, a mod b)
```

Extended Euclidean algorithm (EEA) The EEA finds s and t such that

$$sa + tb = \gcd(a,b).$$

It uses four sets of variables $\{q_i, r_i, s_i, t_i\}$.

1. Initial conditions:

$$r_{-1} = a,\ r_0 = b, \quad s_{-1} = 1,\ s_0 = 0, \quad t_{-1} = 0,\ t_0 = 1, \quad i = 1.$$

2. If $r_{i-1} \neq 0$, perform the long division $r_{i-2} = q_i r_{i-1} + r_i$.
3. $s_i = s_{i-2} - q_i s_{i-1}$ and $t_i = t_{i-2} - q_i t_{i-1}$.
4. Repeat steps 2 and 3 until $r_i = 0$. Then $r_{i-1} = \gcd(a,b)$ and $s_{i-1}a + t_{i-1}b = r_{i-1}$.

Over GF(2), the EEA for $a = x^4 + x^3 + x$ and $b = x^3 + x$ runs as follows.

i	q_i	r_i	s_i	t_i
-1	$--$	$x^4 + x^3 + x$	1	0
0	$--$	$x^3 + x$	0	1
1	$x+1$	x^2	1	$x+1$
2	x	x	x	$x^2 + x + 1$
3	x	0	$x^2 + 1$	$x^3 + x^2 + 1$

And indeed, $\gcd(a,b) = x$ and $xa + (x^2 + x + 1)b = x$.

28.2 EEA decoding

In this section, we go through the EEA decoding algorithm for Reed-Solomon codes. We illustrate it with the following running example. We are using RS(7,5), over GF(8) generated by $x^3 + x + 1$; this is a code of minimum distance $d_{\min} = 3$ and error-correction capability $t = 1$. The received vector is

$$\mathbf{v} = (1, \alpha, 1, 1, 1, 1, 1).$$

We then have $v(x) = 1 + \alpha x + x^2 + x^3 + x^4 + x^5 + x^6$.

The syndrome polynomial Consider the received polynomial $v(x) = c(x) + e(x)$, where

$$e(x) = \sum_{l=1}^{\nu} e_{i_l} x^{i_l}, \quad \nu \le t.$$

The syndromes are

$$S_j = v(\alpha^j) = e(\alpha^j) = \sum_{l=1}^{\nu} e_{i_l} X_l^j, \quad j = 1, \ldots, 2t.$$

where $\{X_l = \alpha^{i_l}\}$ are the error locators. Then let $S(x) = 1 + \sum_{j=1}^{\infty} S_j x^j$ be the syndrome polynomial; we have

$$S(x) = 1 + \sum_{l=1}^{\nu} e_{i_l} \left(\frac{X_l x}{1 - X_l x} \right).$$

In our example,

$$\begin{aligned} S_1 &= v(\alpha^1) = \alpha^4, \\ S_2 &= v(\alpha^2) = \alpha^5, \\ S(x) &= \alpha^5 x^2 + \alpha^4 x + 1. \end{aligned}$$

The key equation Let $\Lambda(x)$ be the error-locator polynomial, whose roots are the inverses of the error locators:

$$\Lambda(x) := \prod_{l=1}^{\nu} (1 - X_l x) = \sum_{m=0}^{\nu} \Lambda_m x^m$$

Then define the error magnitude polynomial

$$\begin{aligned} \Omega(x) &= \Lambda(x) S(x) \\ &= \left[\prod_{l=1}^{\nu} (1 - X_l x) \right] \left[1 + \sum_{l=1}^{\nu} e_{i_l} \left(\frac{X_l x}{1 - X_l x} \right) \right] \\ &= \Lambda(x) + \sum_{l=1}^{\nu} \left[e_{i_l} X_l x \prod_{j \ne l} (1 - X_l x) \right]. \end{aligned}$$

We obtain the key equation

$$\Lambda(x) S(x) = \Omega(x) \mod x^{2t+1}.$$

Using the EEA The key equation is equivalent to

$$\Theta(x)x^{2t+1} + \Lambda(x)S(x) = \Omega(x)$$

for some $\Theta(x)$. We can then use the EEA to determine the gcd of x^{2t+1} and $S(x)$ and to generate sets of solutions $(t_i = \Lambda^{(i)}(x), r_i = \Omega^{(i)}(x))$ that satisfy

$$\begin{aligned} s_i x^{2t+1} + t_i S(x) &= \Theta^{(i)}(x)x^{2t+1} + \Lambda^{(i)}(x)S(x) \\ &= \Omega^{(i)}(x). \end{aligned}$$

The following theorem gives an efficient way of telling when the EEA returns the right solution.

Theorem 28.1. *If* $\deg\Omega^{(i)}(x) \le t$, *then* $\Lambda^{(i)}(x) = \Lambda(x)$ *and* $\Omega^{(i)}(x) = \Omega(x)$ *(both up to the same multiplicative constant).*

The multiplicative constant comes from the fact that $\Omega(0) = \Lambda(0) = 1$ (both constant terms are normalised), while the output of the EEA will give something like $\Omega^{(i)}(0) = \Lambda^{(i)}(0) = \beta$. We just then only need to divide by β to recover the error-locator and error magnitude polynomials.

In our example, the EEA runs as follows (we do not actually need s_i but we include it for the sake of completeness).

i	q_i	$r_i = \Omega^{(i)}$	$s_i = \Theta^{(i)}$	$t_i = \Lambda^{(i)}$
-1	–	x^3	1	0
0	–	$\alpha^5x^2 + \alpha^4x + 1$	0	1
1	$\alpha^2x + \alpha$	$\alpha^3x + \alpha$	1	$\alpha^2x + \alpha$

Since $\deg r_1 = 1 \le t$, we stop here and obtain

$$\begin{aligned} \Omega(x) &= (\alpha^3x + \alpha)\cdot\alpha^{-1} = \alpha^2x + 1, \\ \Lambda(x) &= (\alpha^2x + \alpha)\cdot\alpha^{-1} = \alpha x + 1. \end{aligned}$$

Determining the error locations Chien search is an efficient way to find the roots of a polynomial; we use it on the error-locator polynomial.

Denote $\gamma_{j,i} = \Lambda_j\alpha^{ij}$, then $\gamma_{j,i+1} = \gamma_{j,i}\alpha^j$.

Initial values: $\gamma_{j,0} = \Lambda_j$ for $j = 0,\dots,s = \deg\Lambda(x)$ and $l = s$.

For $i = 0$ up to $q-2$, if

$$\Lambda(\alpha^i) = \sum_{j=0}^{\nu} \gamma_{j,i} = 0,$$

then α^i is a root of Λ. Then $X_l = \alpha^{-i}$ and decrement l.

In our example, we have $s = 1$, $\Lambda_0 = 1$, $\Lambda_1 = \alpha$. Chien search runs as follows.

$$\begin{array}{llll} i = 0 & \gamma_{0,0} = 1 & \gamma_{1,0} = \alpha & \Lambda(\alpha^0) \neq 0 \\ i = 1 & \gamma_{0,1} = 1 & \gamma_{1,1} = \alpha^2 & \Lambda(\alpha^1) \neq 0 \\ i = 2 & \gamma_{0,2} = 1 & \gamma_{1,2} = \alpha^3 & \Lambda(\alpha^2) \neq 0 \\ i = 3 & \gamma_{0,3} = 1 & \gamma_{1,3} = \alpha^4 & \Lambda(\alpha^3) \neq 0 \\ i = 4 & \gamma_{0,4} = 1 & \gamma_{1,4} = \alpha^5 & \Lambda(\alpha^4) \neq 0 \\ i = 5 & \gamma_{0,5} = 1 & \gamma_{1,5} = \alpha^6 & \Lambda(\alpha^5) \neq 0 \\ i = 6 & \gamma_{0,6} = 1 & \gamma_{1,6} = \alpha^0 & \Lambda(\alpha^6) = 0 \end{array}$$

And indeed α^6 is the only root of $\Lambda(x)$, so $X_1^{-1} = \alpha^6$. Therefore, $X_1 = \alpha^1$ and the error is located on coordinate $i_1 = 1$.

Determining the error magnitudes Let $f(x) = \sum_{i=0}^{m} f_i x^i \in \mathrm{GF}(q)[x]$. The formal derivative $f'(x)$ is defined by

$$f'(x) = \sum_{i=1}^{m} i f_i x^{i-1}.$$

Theorem 28.2 (The Forney algorithm.)**.** *The error magnitudes are given by*

$$e_{i_l} = \frac{-X_l \Omega(X_l^{-1})}{\Lambda'(X_l^{-1})}.$$

In our example, we have $\Lambda'(x) = \alpha$, and hence the error magnitude is

$$e_{i_1} = \frac{-\alpha^1 \cdot \Omega(\alpha^6)}{\Lambda'(\alpha^6)} = \alpha^3 = \alpha + 1.$$

Therefore, the error polynomial is

$$e(x) = e_{i_1} x^{i_1} = \alpha^3 x.$$

The codeword polynomial and the actual codewords are then

$$c(x) = v(x) - e(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6$$
$$\mathbf{c} = (1,1,1,1,1,1,1).$$

The whole algorithm

1. Compute the syndromes $S_1, \ldots, S_{n-k}$.
2. Set the following initial conditions:
$$r_{-1}(x) = x^{2t+1}, \quad r_0(x) = S(x), \quad t_{-1}(x) = 0, \quad t_0(x) = 1.$$
3. Using the EEA, compute the successive remainders $r_i(x)$ and the corresponding $t_i(x)$ until $\deg r_i(x) \leq t$.
4. Find the roots of $t_i(x) = \Lambda(x)$ by Chien search, thus determining the error locations.
5. Determine the magnitude of the errors by the Forney algorithm.
6. Construct the error polynomial and determine the codeword polynomial.

28.3 See further

Other decoding algorithms Due to their structure, Reed-Solomon codes have many decoding algorithms:

- Extended Euclidean Algorithm
- Berlekamp-Massey
- Peterson-Gorenstein-Zierler
- Welch-Berlekamp
- Sudan
- Guruswami-Sudan
- Kötter-Vardy
- Blahut's frequency-domain decoding...

But that's another story!

28.4 Exercises

Exercise 28.1. Run the EEA to find the gcd of $x^{10}+x^6+1$ and x^7+1 in GF(2)[x].

Exercise 28.2. Decode $\mathbf{v}=(0,0,\alpha^5,1,\alpha^2,0,\alpha^2)$ using EEA decoding for RS(7,3). (Use GF(8) generated by x^3+x+1.)

29 Code-based cryptography I: The McEliece cryptosystem

29.1 Generalised Reed-Solomon codes and Goppa codes

Generalised Reed-Solomon codes Recall that $\mathsf{RS}(n,k')$ is the set of evaluations of polynomials of degree $<k'$ over all nonzero elements of GF(q^m).

Let $\alpha=(\alpha_1,\ldots,\alpha_n)$ where the α_i are distinct elements of GF(q^m), and let $\mathbf{v}=(v_1,\ldots,v_n)$ where the v_i are nonzero elements of GF(q^m). Then

$$\mathsf{GRS}_{k'}(\alpha,\mathbf{v}):=\left\{(v_1f(\alpha_1),\ldots,v_nf(\alpha_n)):f(x)\in \mathrm{GF}(q^m)[x],\deg(f)<k'\right\}.$$

Clearly, $\mathsf{RS}(n,k')=\mathsf{GRS}_{k'}(\alpha,\mathbf{1})$ for $\alpha_i=\alpha^{i-1}$.

Some properties of GRS Generator matrix

$$\mathbf{G}=\begin{pmatrix} v_1 & v_2 & \ldots & v_n \\ \alpha_1v_1 & \alpha_2v_2 & \ldots & \alpha_nv_n \\ \alpha_1^2v_1 & \alpha_2^2v_2 & \ldots & \alpha_n^2v_n \\ \ldots & \ldots & \ldots & \ldots \\ \alpha_1^{k'-1}v_1 & \alpha_2^{k'-1}v_2 & \ldots & \alpha_n^{k'-1}v_n \end{pmatrix}$$

where $r=n-k'$.

GRS codes are MDS: $d_{\min}=r+1$.

Properties of GRS The dual of a GRS code is another GRS code:

$$\mathsf{GRS}_{k'}(\alpha,\mathbf{v})^{\perp}=\mathsf{GRS}_r(\alpha,\mathbf{y})$$

for some $\mathbf{y}$, we obtain the parity-check matrix of $\mathsf{GRS}_{k'}(\alpha,\mathbf{v})$.

$$\begin{aligned}\mathbf{H}&=\begin{pmatrix} y_1 & y_2 & \ldots & y_n \\ \alpha_1y_1 & \alpha_2y_2 & \ldots & \alpha_ny_n \\ \alpha_1^2y_1 & \alpha_2^2y_2 & \ldots & \alpha_n^2y_n \\ \ldots & \ldots & \ldots & \ldots \\ \alpha_1^{r-1}y_1 & \alpha_2^{r-1}y_2 & \ldots & \alpha_n^{r-1}y_n \end{pmatrix} \\ &=\begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \ldots & \ldots & \ldots & \ldots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \ldots & \alpha_n^{r-1} \end{pmatrix}\begin{pmatrix} y_1 & & & 0 \\ & y_2 & & \\ & & \ddots & \\ 0 & & & y_n \end{pmatrix}.\end{aligned}$$

Alternant codes The alternant code $\mathsf{A}(\alpha,\mathbf{y})$ consists of all the codewords of $\mathsf{GRS}_{k'}(\alpha,\mathbf{v})$ with coefficients in GF(q):

$$\mathsf{A}(\alpha,\mathbf{y})=\mathsf{GRS}_{k'}(\alpha,\mathbf{v})\cap \mathrm{GF}(q)^n.$$

Parameters of the alternant code:

1. Length n
2. Dimension $k \geq n - mr$
3. Minimum distance $d \geq r + 1$.
4. A parity-check $\bar{\mathbf{H}}$ of $\mathsf{A}(\alpha, \mathbf{y})$ can be obtained by replacing each element $a \in \mathrm{GF}(q^m)$ in $\mathbf{H}$ for the GRS code by its expansion as a vector in $\mathrm{GF}(q)^m$.

Hamming codes are alternant codes Let $q = 2$, $m \geq 2$, $n = 2^m - 1$, $r = 1$, and consider $\mathsf{RS}(n, n-1)$. For instance, if $m = 3$ we obtain $\mathsf{RS}(7,6)$ with parity-check matrix

$$\mathbf{H}_{\mathsf{RS}(7,6)} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \end{pmatrix}.$$

So for the alternant code,

$$\bar{\mathbf{H}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

That is the Hamming code (up to equivalence)! Note that in this case, $d = 3 > r + 1$.

Goppa codes Let $G(x)$ be a polynomial in $\mathrm{GF}(q^m)[x]$ of degree r, called the Goppa polynomial. Let $L = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathrm{GF}(q^m)$ such that $G(\alpha_i) \neq 0$ for all $\alpha_i \in L$. (Usually, L is chosen to be all the non-roots of $G(x)$.)

Then the Goppa code $\Gamma(L, G)$ is the alternant code:

$$\Gamma(L, G) = \mathsf{A}(\alpha, \mathbf{y})$$

for $y_i = G(\alpha_i)^{-1}$. If $G(x)$ is irreducible, $\Gamma(L, G)$ is called an irreducible Goppa code.

Parameters of $\Gamma(L, G)$:

1. Length $n = |L|$,
2. Dimension $k \geq n - mr$, $r = \deg G(x)$,
3. Minimum distance $d \geq r + 1$.

Parity-check matrix

$$\mathbf{H} = \begin{pmatrix} G(\alpha_1)^{-1} & G(\alpha_2)^{-1} & \ldots & G(\alpha_n)^{-1} \\ \alpha_1 G(\alpha_1)^{-1} & \alpha_2 G(\alpha_2)^{-1} & \ldots & \alpha_n G(\alpha_n)^{-1} \\ \alpha_1^2 G(\alpha_1)^{-1} & \alpha_2^2 G(\alpha_2)^{-1} & \ldots & \alpha_n^2 G(\alpha_n)^{-1} \\ \ldots & \ldots & \ldots & \ldots \\ \alpha_1^{r-1} G(\alpha_1)^{-1} & \alpha_2^{r-1} G(\alpha_2)^{-1} & \ldots & \alpha_n^{r-1} G(\alpha_n)^{-1} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & \ldots 1 & \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \ldots & \ldots & \ldots & \ldots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \ldots & \alpha_n^{r-1} \end{pmatrix} \begin{pmatrix} G(\alpha_1)^{-1} & & & 0 \\ & G(\alpha_2)^{-1} & & \\ & & \ddots & \\ 0 & & & G(\alpha_2)^{-1} \end{pmatrix}.$$

Binary Goppa codes For binary Goppa codes, we can significantly improve the lower bound on the minimum distance.

Theorem 29.1. *Suppose that $G(x)$ has no multiple roots, then*

$$d_{\min}(\Gamma(L, G)) \geq 2 \deg G(x) + 1.$$

Patterson gave an efficient decoding algorithm (based on the EEA) to correct r errors.

29.2 The McEliece public-key cryptosystem

McEliece cryptosystem (1978) Let F be a family of t-error correcting q-ary linear $[n,k]$ codes.

Key generation

1. select $\mathsf{C} \in \mathsf{F}$.
2. Public key: $\mathbf{G} \in \mathrm{GF}(q)^{k\times n}$, a generator matrix of C.
3. Secret key: $\Phi : \mathrm{GF}(q)^n \to \mathsf{C}$, a t-bounded decoder.

Encryption $E_{\mathbf{G}} : \mathrm{GF}(q)^k \to \mathrm{GF}(q)^n$,

$$E_{\mathbf{G}}(\mathbf{m}) = \mathbf{mG} + \mathbf{e},$$

where $\mathbf{e}$ is a random error vector of weight t.

Decryption $D_\Phi : \mathrm{GF}(q)^n \to \mathrm{GF}(q)^k$,

$$D_\Phi(\mathbf{v}) = \Phi(\mathbf{v})\mathbf{G}^*,$$

with $\mathbf{GG}^* = \mathbf{I}$.

Indeed it works:

$$D_\Phi(E_{\mathbf{G}}(\mathbf{m})) = \Phi(\mathbf{mG}+\mathbf{e})\mathbf{G}^* = \mathbf{mGG}^* = \mathbf{m}.$$

McEliece's original system (1978)

Secret key

- $\bar{\mathbf{G}}$: generator matrix of a binary irreducible Goppa code.
- $\mathbf{S}$: $k \times k$ nonsingular matrix.
- $\mathbf{P}$: $n \times n$ permutation matrix.

Public key

$$\mathbf{G} = \mathbf{S}\bar{\mathbf{G}}\mathbf{P}.$$

Encryption

$$\mathbf{v} = \mathbf{mG} + \mathbf{e}.$$

Decryption

1. Compute $\mathbf{z} = \mathbf{vP}^{-1}$.
2. Decode $\mathbf{z}$ to obtain a codeword $\mathbf{c}$ and the corresponding original message $\bar{\mathbf{m}}$ (for $\bar{\mathbf{G}}$).
3. Compute $\mathbf{m} = \bar{\mathbf{m}}\mathbf{S}^{-1}$.

Original parameters:

- plaintext size: $k = 524$ bits
- ciphertext size: $n = 1024$ bits
- error weight: $t = 50$ bits

Toy example Original McEliece system with the $[7,4,3]_2$-Hamming code.
Secret key:

$$\bar{\mathbf{G}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathbf{S} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Public key:

$$\mathbf{G} = \mathbf{S}\bar{\mathbf{G}}\mathbf{P} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Plaintext: $\mathbf{m} = (1,0,1,0)$.

Ciphertext:

$$\begin{aligned} \mathbf{c} &= \mathbf{m}\mathbf{G} = (1,0,1,0,1,0,0), \\ \mathbf{e} &= (1,0,0,0,0,0,0) \\ \mathbf{y} &= (0,0,1,0,1,0,0). \end{aligned}$$

Decryption:

$$\begin{aligned} \mathbf{z} &= \mathbf{y}\mathbf{P}^{-1} &&= (0,0,0,1,0,1,0) \\ \bar{\mathbf{c}} &= \mathbf{z} + \mathbf{e}_2 &&= (0,1,0,1,0,1,0) \\ \bar{\mathbf{m}} & &&= (0,0,1,0) \\ \mathbf{m} &= \bar{\mathbf{m}}\mathbf{S}^{-1} &&= (1,0,1,0). \end{aligned}$$

Advantages/Disadvantages of the McEliece System
Advantages

1. Both encryption and decryption have quadratic complexity in block length. That compares very well to RSA.
2. Encryption and decryption can also be very efficiently implemented in hardware.
3. No polynomial time quantum algorithm is known to decode a general linear block code. Even better, it is known that decoding a general linear code is an NP-hard problem.

Disadvantages

1. The public key is fairly large. About 0.5 Mbits compared to 0.1 Mbits for RSA and 0.02 Mbits for elliptic curves.
2. The data rate is only about one half.

29.3 See further

Niederreiter cryptosystem (1986) Let F be a family of t-error correcting q-ary linear $[n,k]$ codes, $r = n-k$. Let $S_n(t)$ denote the set of error vectors of length n and Hamming weight t:

$$S_n(t) = \left\{\mathbf{e} \in \mathrm{GF}(q)^n : w_{\mathrm{H}}(\mathbf{e}) = t\right\}.$$

Key generation

1. Choose $\mathsf{C} \in \mathsf{F}$.
2. Public key: $\mathbf{H} \in \mathrm{GF}(q)^{r\times n}$, a parity-check matrix of C.
3. Secret key: $\Psi : \mathrm{GF}(q)^r \to \mathrm{GF}(q)^n$, a t-bounded $\mathbf{H}$-syndrome decoder

Encryption $E_{\mathbf{H}} : S_n(t) \to \mathrm{GF}(q)^r$,

$$E_{\mathbf{H}}(\mathbf{e}) = \mathbf{e}\mathbf{H}^\top.$$

Decryption $D_\Psi : \mathrm{GF}(q)^r \to S_n(t)$,

$$D_\Psi(\mathbf{s}) = \Psi(\mathbf{s}).$$

Indeed it works:

$$D_\Psi(E_{\mathbf{H}}(\mathbf{e})) = \Psi(\mathbf{e}\mathbf{H}^\top) = \mathbf{e}.$$

29.4 Exercises

Exercise 29.1. Prove that GRS codes are MDS.

Exercise 29.2. Why would it be impractical to use long Hamming codes for the McEliece system?

Exercise 29.3. Implement the McEliece cryptosystem with original n and k, but using a Reed-Muller code with similar error-correction capability. You need to implement the key generation, encryption and decryption algorithms.

30 Code-based cryptography II: Cryptanalysis

30.1 Security of McEliece cryptosystem

Hard Decoding Problems In [Berlekamp, McEliece, van Tilborg, 78]

SYNDROME DECODING is NP-complete.

Instance: $\mathbf{H} \in \mathrm{GF}(2)^{r\times n}$, $\mathbf{s} \in \mathrm{GF}(2)^r$, t integer.
Question: Is there $\mathbf{e} \in \mathrm{GF}(2)^n$ such that $w_{\mathrm{H}}(\mathbf{e}) \le t$ and $\mathbf{e}\mathbf{H}^\top = \mathbf{s}$?

COMPUTATIONAL SYNDROME DECODING is NP-hard.

Instance: $\mathbf{H} \in \mathrm{GF}(2)^{r\times n}$, $\mathbf{s} \in \mathrm{GF}(2)^r$, t integer.
Output: $\mathbf{e} \in \mathrm{GF}(2)^n$ such that $w_{\mathrm{H}}(\mathbf{e}) \le t$ and $\mathbf{e}\mathbf{H}^\top = \mathbf{s}$.

(Probably) Hard Structural Problems GOPPA CODE DISTINGUISHING is in NP.

Instance: $\mathbf{G} \in \mathrm{GF}(2)^{k\times n}$.
Question: Does $\mathbf{G}$ span a binary Goppa code?

NP: the property is easy to check given (L,G).
Completeness status is unknown.

GOPPA CODE RECONSTRUCTION.

Instance: $\mathbf{G} \in \mathrm{GF}(2)^{k\times n}$.
Output: (L,G) such that $\mathbf{G}$ is a generator matrix of $\Gamma(L,G)$.

Complexity is unknown.

Security reduction for McEliece A (T,ϵ)-adversary for McEliece (with Goppa codes) is an algorithm Λ that runs in time T such that,

$$\mathbb{P}(\Lambda(\mathbf{xG}+\mathbf{e})=\mathbf{e}|\mathbf{G}\in \text{Goppa}) \geq \epsilon.$$

A (T,ϵ)-distinguisher for Goppa codes is an algorithm $\Delta : \mathrm{GF}(2)^{k\times n} \to \{True, False\}$ that runs in time T such that

$$\text{Advantage}(\Delta) := |\mathbb{P}(\Delta(\mathbf{G})) - \mathbb{P}(\Delta(\mathbf{G})|\mathbf{G}\in \text{Goppa})| \geq \epsilon.$$

A (T,ϵ)-decoder is a decoding algorithm that runs in time T and which decodes successfully with probability at least ϵ.

Theorem 30.1. *If there exists a (T,ϵ)-adversary against McEliece then there exists either a $(T,\epsilon/2)$-decoder for t errors in a random $[n,k]$ code, or a $(T+O(n^2),\epsilon/2)$-distinguisher for Goppa codes.*

30.2 Attacks against McEliece

Best Known Attacks Decoding attacks. For the public-key encryption schemes the best attack is always Information Set Decoding (ISD).

Key attacks. Most proposals using families other than binary Goppa codes have been broken. For binary Goppa codes there are only exhaustive attacks enumerating either generator polynomials or supports.

Prange Algorithm (Information Set Decoding) 1962 An information set for a linear code C is a set $I = \{i_1,\ldots,i_k\}$ of columns of $\mathbf{G}$ such that the corresponding $k\times k$ matrix $\mathbf{G}_I$ is invertible.

For any codeword $\mathbf{c}$, we then have

$$\mathbf{c} = \mathbf{c}_I \mathbf{G}_I^{-1}\mathbf{G}.$$

Suppose we received $\mathbf{y} = \mathbf{c}+\mathbf{e}$ with $\mathbf{e}$ of weight $\leq t$.

1. Choose an information set I and suppose that none of these positions have errors ($\mathbf{e}_I = \mathbf{0}$ and hence $\mathbf{y}_I = \mathbf{c}_I$).

2. This yields an estimate for $\mathbf{c}$:
$$\hat{\mathbf{c}} = \mathbf{y}_I \mathbf{G}_I^{-1}\mathbf{G}.$$

3. If $d_{\mathrm{H}}(\hat{\mathbf{c}},\mathbf{y}) \leq t$, then return $\hat{\mathbf{c}}$; else, go to Step 1.

Complexity of ISD The probability of finding k coordinates I such that $\mathbf{e}_I = \mathbf{0}$ is

$$P = \frac{\binom{n-t}{k}}{\binom{n}{k}} \approx \left(\frac{n-t}{n}\right)^k.$$

In general, not all sets of k coordinates form an information set, but we can view this upper bound on the probability of success for the adversary as a "security guarantee." This estimate yields an expected running time of $O(n^3\cdot P^{-1})$, which is exponential in k.

Lee and Brickell's attack The first nontrivial improvement on Prange's attack was given by Lee and Brickell. The main idea is that, once an information set I is selected, to consider more than one potential codeword. More precisely, it is based on a fixed parameter p and for every information set I it runs the following steps until it finds an error vector $\hat{\mathbf{e}}'$ with weight at most t.

1. Compute $\hat{\mathbf{c}} = \mathbf{y}_I \mathbf{G}_I^{-1} \mathbf{G}$ as before.
2. For every $\hat{\mathbf{e}}_I \in \mathrm{GF}(2)^k$ with Hamming weight at most p, compute

$$\hat{\mathbf{e}}' = \mathbf{y} - \hat{\mathbf{c}} - \hat{\mathbf{e}}_I \mathbf{G}_I^{-1} \mathbf{G}.$$

In other words, now we only need to look for an information set I such that

$$w_{\mathrm{H}}(\mathbf{e}_I) \leq p.$$

(Prange's attack is a special case, for $p = 0$.) The success probability is about

$$P \approx \frac{\binom{t}{p}\binom{n-t}{k-p}}{\binom{n}{k}} \approx \left(\frac{n-t}{n}\right)^{k-p} \left(\frac{t}{p}\right)^{p}.$$

Improvements (non-exhaustive list)

- Prange 1962;
- Lee, Brickel 1988;
- Stern 1989 and Dumer 1991;
- Canteaut, Chabaud 1998;
- May, Meurer, Thomae 2011;
- Becker, Joux, May, Meurer, 2012;
- May, Ozerov, 2015.

For instance, for a binary code of length $0.5n$, correcting an amount of 10% of errors costs $O(2^{0.09n})$ using BJMM. Note that most of the improvements concern binary codes.

Key attacks How to propose a family of codes? The family should contain "large" enough codes to resist to message recovery attacks and be large enough (to avoid brute force search). Moreover, the structure of the code should be easily hidden (which is the hardest task).

So far, Goppa codes have been resistant to key recovery attacks because they meet both criteria. Other classes of codes are insecure, e.g. GRS, Reed-Muller, LDPC, etc.

A distinguisher for GRS codes We now give a key property of GRS codes, which can then be used to produce a distinguisher and hence break McEliece using GRS codes.

For any $[n,k]_q$-linear code C, let

$$\mathsf{C}^2 := \{\mathbf{c} * \mathbf{d} := (c_1 d_1, \ldots, c_n d_n) : \mathbf{c}, \mathbf{d} \in \mathsf{C}\}.$$

Clearly, C^2 is a linear code too. Its dimension is at most $\binom{k+1}{2}$.

Theorem 30.2. *For a random linear code with $\binom{k+1}{2} < n$, C^2 has dimension $\binom{k+1}{2}$ with high probability.*

On the other hand,

$$\mathsf{GRS}_k(\alpha, \mathbf{v})^2 = \mathsf{GRS}_{2k-1}(\alpha, \mathbf{v} * \mathbf{v}).$$

30.3 See further

Other attacks Resend-message Attack. [Berson, 97] The same message is sent twice with the same public key G. Then the message can be recovered.

Reaction Attack. [Kobara, Imai, 00] We assume the decryption system can be used as an oracle and behaves differently when its input is at distance $> t$ from the code or when its input is at distance $\leq t$ from the code. Then the oracle can be transformed into a decoder.

The GPT cryptosystem Rank metric codes have been proposed for code-based cryptography by Gabidulin, Paramonov and Tretjakov in 1991. Unfortunately, their scheme was base on Gabidulin codes, which have too much structure (since they are analogues of Reed-Solomon codes) and hence it got broken. Nonetheless, some people think that rank metric based cryptography has a lot of potential, so there is ongoing work looking for suitable codes for that application.

30.4 Exercises

Exercise 30.1. Prove that a linear code is MDS if and only if I is an information set for any $I \subseteq [n]$ of cardinality k.

Exercise 30.2. The aim of this question is to count the information sets of the Hamming code. Prove the following facts.

1. $I \subseteq [n]$ is an information set of C if and only if $\bar{I} = [n] \setminus I$ is an information set of $\mathsf{C}^{\perp}$.
2. There are exactly
$$(2^r - 1) \cdot (2^r - 2) \cdots (2^r - 2^{r-1})$$
nonsingular matrices in $\mathrm{GF}(2)^{r \times r}$.
3. The Hamming code of length $n = 2^r - 1$ has exactly
$$\frac{(2^r - 1) \cdot (2^r - 2) \cdots (2^r - 2^{r-1})}{r!}$$
information sets.
4. The probability that a random set of k columns is an information set tends to
$$\phi\left(\frac{1}{2}\right) \approx 0.2888,$$
where $\phi(x)$ is Euler's function.

Exercise 30.3. Use information set decoding to decrypt the ciphertext in the toy example of Lecture 29. Choose at least one "wrong" information set, that does contain an error.

Chapter 4

Information Theory

Information Theory is the mathematical theory behind quantifying and transmitting information. As such, it determines the fundamental limitations of data compression, cryptosystems, and error-correcting codes.

This chapter is rather light, omitting many technical details and focusing on the ideas and intuitions instead. Those details can be found in some of the different textbooks on Information Theory available out there. I personally recommend the book by Cover and Thomas [5], which is the most accessible while being thorough. The book by Yeung [18] is also very good, though drier; Csiszár and Körner's book [6] is a useful reference but very heavy; finally, ElGamal and Kim's [8] is a valuable resource on Network Information Theory.

31 Entropy I: The basics

31.1 Discrete random variables

Definitions A discrete random variable X is a mapping p_X from a countable set $\mathscr{X}$ to $\mathbb{R}^+$ such that

$$\sum_{x\in\mathscr{X}} p_X(x) = 1.$$

We usually just write $p(x)$ instead of $p_X(x)$. Usually, $\mathscr{X}$ will be a finite subset of the real numbers.

We can then form functions of the random variable X: if $f:\mathbb{R}\to\mathbb{R}$, then $f(X)$ is the random variable such that

$$p_{f(X)}(y) = \sum_{x:f(x)=y} p(x).$$

Expectation and standard deviation The expectation of a discrete random variable X is

$$\mathbb{E}(X) := \sum_{x\in\mathscr{X}} p(x)x.$$

The variance of X is

$$\begin{aligned}\mathrm{Var}X &:= \mathbb{E}((X-\mathbb{E}(X))^2)\\ &= \mathbb{E}(X^2)-\mathbb{E}(X)^2.\end{aligned}$$

The standard deviation of X is

$$\sigma = \sqrt{\mathrm{Var}X}.$$

Classical examples The B(p) random variable is

$$X = \begin{cases} 1 & \text{with probability } p\\ 0 & \text{with probability } 1-p.\end{cases}$$

More generally, the binomial distribution $\mathrm{B}(n,p)$ is defined by

$$p(w) = \binom{n}{w} p^w (1-p)^{n-w} \quad (w \in \{0,1,\dots,n\}).$$

(So that $\mathrm{B}(p) = \mathrm{B}(1,p)$.) The uniform random variable $\mathrm{U}(\mathscr{X})$ on $\mathscr{X}$ is simply defined as

$$p_{\mathrm{U}}(x) = \frac{1}{|\mathscr{X}|}.$$

31.2 Entropy

Entropy Let X be a discrete random variable with alphabet $\mathscr{X}$ and probability mass function

$$p(x) = \mathbb{P}(X = x), \forall x \in \mathscr{X}.$$

Entropy: amount of information contained in random variable X/ amount of uncertainty about X

The entropy of a discrete random variable X is defined as

$$\begin{aligned} H(X) &:= -\sum_{x \in \mathscr{X}} p(x) \log p(x) \\ &= -\mathbb{E}_p(\log p(X)). \end{aligned}$$

We use the convention $0 \log 0 = 0$ (i.e. events with zero probability do not contribute to the entropy). Unless specified otherwise, the logarithm is in base 2 and the entropy is expressed in bits.

The most basic example is that of a Bernoulli random variable:

$$X = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1-p. \end{cases}$$

$$H(p) := H(X) = -p \log p - (1-p) \log(1-p)$$

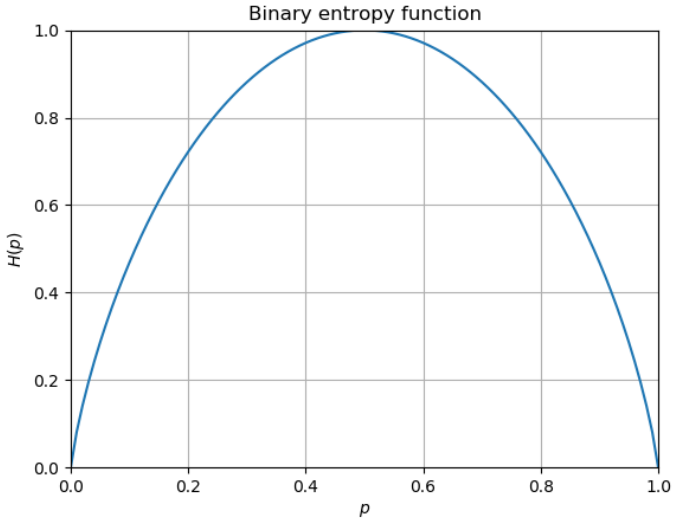
H(p) or H(alpha) refers to bernoulli variable

$H(p)$ is plotted below. We have the following intuitive properties:

1. $H(p) = H(1-p)$, since switching p for $1-p$ only flips the roles of 0 and 1;
2. $H(0) = H(1) = 0$, since in that case, the Bernoulli random variable X is actually deterministic.
3. $H(1/2) = 1$, since in that case, X is a "pure" random bit.

Entropy = uncertainty and uncertainty = 0 here

100% uncertainty here



Intuition: Information ≈ Uncertainty Intuitively, the entropy of X is:

- the amount of uncertainty we have about X;
- the amount of information contained in X.

In more detail, the information contained in the event x is

$$-\log p(x) = \log \frac{1}{p(x)}.$$

The more infrequent an event, the more informative it is! Thus, the entropy is the average amount of information given by the random variable X.

Entropy examples

- The entropy of a random bit is 1 bit.
- The entropy of a random digit (0-9) is

$$\log 10 = 3.32 \text{ bits.}$$ Use the formula

- What has greater entropy, a race X with seven runners, with probabilities

$$\left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\right)$$

of winning or a race Y with eight runners with winning probabilities

$$\left(\frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{4}, \frac{1}{4}\right)?$$

Answer: the entropies are actually equal:

$$\begin{aligned} H(X) &= -\left(\frac{3}{6}\log\frac{1}{6}\right) - \left(\frac{4}{8}\log\frac{1}{8}\right) \\ &= 2 + \frac{1}{2}\log 3 \\ &= 2.79. \\ H(Y) &= -\left(\frac{6}{12}\log\frac{1}{12}\right) - \left(\frac{2}{4}\log\frac{1}{4}\right) \\ &= 2 + \frac{1}{2}\log 3 \\ &= 2.79. \end{aligned}$$

Exercise 31.3 asks you interpret this result.

31.3 Properties of the entropy function

Basic properties

1. $H(X) \geq 0$, with equality if and only if X is deterministic.
2. $H(X)$ only depends on the distribution $p(x)$, not on the values taken by X.
3. The events with zero probability do not contribute to the entropy.
4. The entropy is a continuous function of the distribution.

Jensen's inequality A function $f(x)$ is convex over an interval (a,b) if for every $x_1, x_2 \in (a,b)$ and $0 \le \lambda \le 1$,

$$\lambda f(x_1) + (1-\lambda) f(x_2) \ge f(\lambda x_1 + (1-\lambda) x_2).$$

A function is strictly convex if equality holds if and only if $\lambda \in \{0,1\}$. A function f is concave if $-f$ is convex.

If f is twice differentiable, then f is convex if and only if $f'' \ge 0$. i.e. dy/dx twice and if +Ve then convex

It is easy to show that $\log x$ is strictly concave and hence $-\log x = \log \frac{1}{x}$ is strictly convex.

Theorem 31.1 (Jensen's inequality)**.** *If f is a convex function and Y is a discrete random variable, then*

$$\mathbb{E}(f(Y)) \ge f(\mathbb{E}(Y)).$$

Moreover, if f is strictly convex, then equality implies that Y is a constant.

More explicitly, if Y satisfies $\mathbb{P}(Y = y) = p_y$, then

Same as equation in green when y only has two values lambda and 1-lambda. this is a generalisation

$$\sum_y p_y f(y) \ge f\left(\sum_y p_y y\right).$$

When y has 3 values (x1, x2, x3) the line shown in diagram below becomes a triangle above the curve. E(f(Y) is any point inside triangle including edges, then f(E(Y) is corresponding point on curve

The definition of convexity is then a special case for $\mathbb{P}(Y = x_1) = \lambda$ and $\mathbb{P}(Y = x_2) = 1 - \lambda$.

Entropy of the uniform distribution The entropy of the uniform distribution on $\mathcal{X}$ is

$$H(\mathrm{U}) = \sum_{x \in \mathcal{X}} \frac{1}{|\mathcal{X}|} \log |\mathcal{X}| = \log |\mathcal{X}|.$$

Log of number of events

We now prove that the uniform distribution maximises the entropy. Because each piece of information is non significant

Theorem 31.2. *For any X on $\mathcal{X}$,*

$$H(X) \le \log |\mathcal{X}| = H(\mathrm{U}).$$

Proof. The proof will use Jensen's inequality for the convex function $f(x) = x \log x$. We have

$$\begin{aligned} \log |\mathcal{X}| - H(X) &= \log |\mathcal{X}| + \sum_x p(x) \log p(x) \\ &= \sum_x p(x) \log (|\mathcal{X}| p(x)). \end{aligned}$$

Let Y be the random variable such that

$$\mathbb{P}(Y = p(x)|\mathcal{X}|) = \frac{1}{|\mathcal{X}|},$$

with expectation

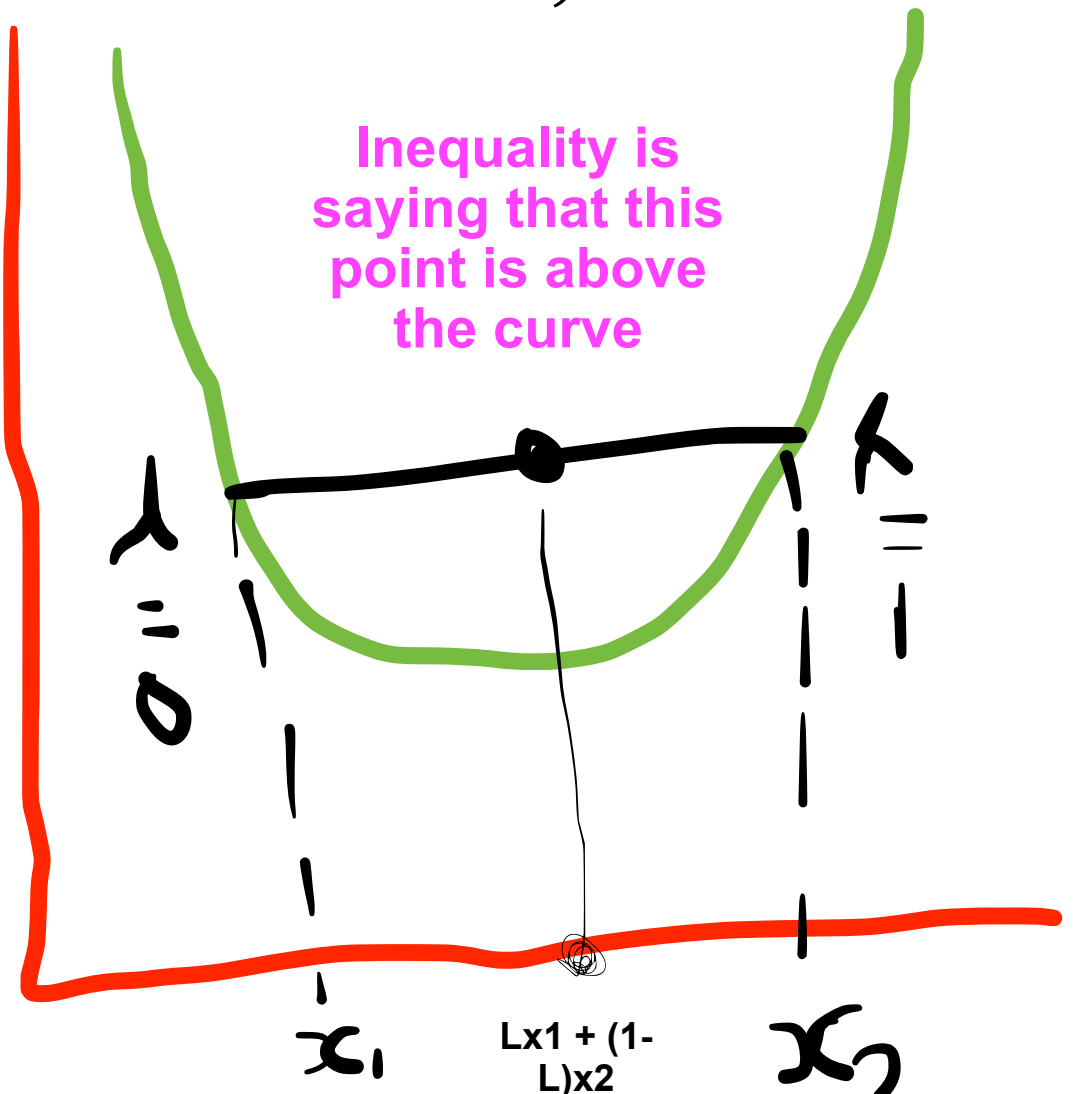
$$\mathbb{E}(Y) = \sum_x \frac{1}{|\mathcal{X}|} (p(x)|\mathcal{X}|) = 1.$$

We can now apply Jensen's inequality:

$$\begin{aligned} \log |\mathcal{X}| - H(X) &= \mathbb{E}(f(Y)) \\ &\ge f(\mathbb{E}(Y)) \\ &= 1 \log 1 \\ &= 0 \end{aligned}$$

□

crete random variable, then



An axiomatic view of entropy Surprisingly enough, the entropy can be uniquely characterised based on a set of "intuitive" properties that any measure of information should satisfy.

1. For any permutation π, $H(p_1,\dots,p_n) = H(p_{\pi(1)},\dots,p_{\pi(n)})$.
2. $H(p_1,\dots,p_n)$ is maximised for the uniform dsitribution $p_1 = p_2 = \cdots = p_n = 1/n$.
3. $H(p_1,\dots,p_n) \geq 0$, with equality only if $p_i = 1$ for some i. i.e. if probability of one of the events = 1, then no uncertainty and entropy = 0
4. $H(p_1,\dots,p_n,0) = H(p_1,\dots,p_n)$.
5. $H(1/n,\dots,1/n) \leq H(1/(n+1),\dots,1/(n+1))$. Uniform distribution of n events gives you less information (less uncertainty) than uniform distribution of n+1 events.
6. H is continuous.
7. $H(1/(mn),\dots,1/(mn)) = H(1/n,\dots,1/n) + H(1/m,\dots,1/m)$. M by N grid- sum of uncertainty of knowing what row youre at and knowing what column youre at
8. $H(p_1,\dots,p_m,q_1,\dots,q_n) = H(p,q) + pH(p_1/p,\dots,p_m/p) + qH(q_1/q,\dots,q_n/q)$, where $p = p_1+\cdots+p_m$ and $q = q_1+\cdots+q_n$. Say m women and n men: split uncertainty to uncertainty given culprit was a man and vice versa

31.4 See further

Rényi entropy The Shannon entropy $H(X) = \sum_x \log(1/p(x))$ was generalised by Rényi as follows. For any $\alpha \in [0,\infty) \setminus \{1\}$, let

$$H_\alpha(X) := \frac{1}{1-\alpha} \log \sum_{x\in\mathcal{X}} p(x)^\alpha.$$

For simplicity, let us assume that $p(x) > 0$ for all $x \in \mathcal{X}$. The Rényi entropy has three important special cases:

- For $\alpha = 0$, we obtain
$$H_0(X) = \log|\mathcal{X}|.$$
This is sometimes referred to as the Hartley entropy of X.
- For $\alpha \to \infty$, we obtain
$$H_\infty = \min_{x\in\mathcal{X}} \log \frac{1}{p(x)}.$$
This is commonly referred to as the min-entropy of X.
- Both limits for α approaching 1 are equal to the Shannon entropy:
$$\lim_{\alpha\to 1^-} H_\alpha(X) = \lim_{\alpha\to 1^+} H_\alpha(X) = H(X).$$
Therefore, this is sometimes denoted $H_1(X)$.

A fourth special case, which finds applications in statistics, is for $\alpha = 2$. Suppose X and Y are i.i.d. Then

$$\log \mathbb{P}(X = Y) = \log \sum_{x\in\mathcal{X}} p(x)^2 = \frac{1}{2} H_2(X).$$

As such, $H_2(X)$ is referred to as the collision entropy of X.

31.5 Exercises

Exercise 31.1 (Entropy with different bases)**.** For any $b > 1$, Let $H_b(X) = -\sum_x p(x)\log_b(p(x))$. Show that

$$H_b(X) = (\log_b a)H_a(X)$$

for all $a, b > 0$.

Exercise 31.2. What is the entropy of a (uniform) random letter (A-Z)? What if you include upper and lower case letters, and common punctuation?

Exercise 31.3. Why did we get the same entropy for the two races X and Y?

Exercise 31.4. Prove Jensen's inequality when Y has a finite number of events.

Exercise 31.5. Plot the Rényi entropy of a Bernoulli variable for varying values of α. Is it always concave?

Exercise 31.6. Keeping p constant, plot $\frac{1}{n}H(\mathrm{B}(n,p))$ for increasing values of n. What do you observe?

32 Entropy II: Mutual information

Want to know how random variables relate to each other: how much info they contain about each other

32.1 Joint entropy and conditional entropy

Joint entropy and conditional entropy The joint entropy $H(X,Y)$ of a pair of discrete random variables (X,Y) with a joint distribution $p(x,y)$ is defined as Joint distribution X and Y

$$\begin{aligned} H(X,Y) &:= -\sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p(x,y)\log p(x,y) \\ &= -\mathbb{E}(\log p(X,Y)). \end{aligned}$$

(Basically, we view the pair (X,Y) as a new random variable and compute its entropy.)

The conditional entropy of Y given X is defined as Once we know X, how much uncertainty is left about Y on average

$$\begin{aligned} H(Y|X) &:= \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \\ &= -\mathbb{E}_{p(x,y)}(\log p(Y|X)). \end{aligned}$$

It is the average uncertainty about Y once we know X.

Chain rule

Theorem 32.1 (Chain rule)**.** Amount of information contained in X and Y (or uncertainty)

$$H(X,Y) = H(X) + H(Y|X).$$ uncertainty of X add the uncertainty of Y once we know X

Proof. We could derive the result from the summation formulas. But we could be very quick and elegant by working with expectations:

$$\begin{aligned} p(X,Y) &= p(X)p(Y|X) \\ \log p(X,Y) &= \log p(X) + \log p(Y|X) \\ \mathbb{E}(\log p(X,Y)) &= \mathbb{E}(\log p(X)) + \mathbb{E}(\log p(Y|X)). \end{aligned}$$

□

Conditioning by a random variable is "transparent," as seen below.

Corollary 32.2. $H(X,Y|Z) = H(X|Z) + H(Y|X,Z)$.

32.2 Mutual information

Relative entropy The relative entropy (or Kullback Leibler divergence) between two probability mass functions $p(x)$ and $q(x)$ is defined as

Relative entropy: how one probability distribution differs from a second

$$D(p||q) := \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)} = \mathbb{E}_p \log\frac{p(X)}{q(X)}.$$

Not symmetric

We use the convention $0\log\frac{0}{q}=0$ and $p\log\frac{p}{0}=\infty$. Remark: $D(p||q)\neq D(q||p)$ in general!

Mutual information The mutual information $I(X;Y)$ is defined as

$$\begin{aligned} I(X;Y) &:= \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \\ &= D(p(x,y)||p(x)p(y)) \\ &= \mathbb{E}_{p(x,y)}\left(\log\frac{p(X,Y)}{p(X)p(Y)}\right). \end{aligned}$$

symmetric

Divergence between the joint distribution and the product

Intuitively, $I(X;Y)$ is

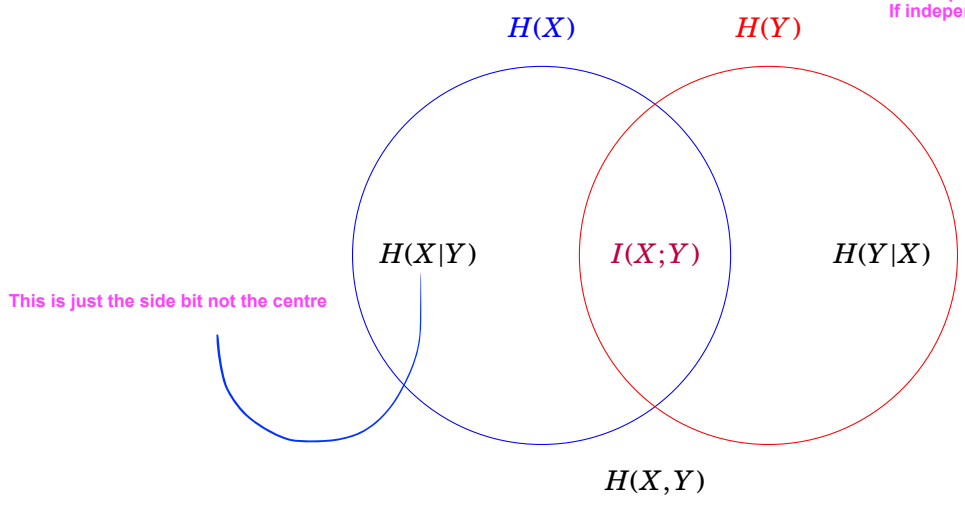
1. the amount of information about X contained in Y;
2. the amount of information about Y contained in X;
3. the amount of information in common between X and Y.

Venn diagram of entropy We have the following relations:

$$\begin{aligned} I(X;Y) &= I(Y;X), \\ I(X;X) &= H(X), \\ I(X;Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= H(X) + H(Y) - H(X,Y). \end{aligned}$$

These can be neatly viewed as a Venn diagram.

X and Y completely correlated: merge into one circle
If independent: then separate, no overlap



32.3 Further properties

Conditional mutual information The conditional relative entropy $D(p(y|x)||q(y|x))$ is defined as

$$\begin{aligned} D(p(y|x)||q(y|x)) &:= \sum_x p(x) \sum_y p(y|x) \log \frac{p(y|x)}{q(y|x)} \\ &= \mathbb{E}_{p(x,y)} \left(\log \frac{p(Y|X)}{q(y|X)} \right). \end{aligned}$$

This bit not really detailed much in the lectures

The conditional mutual information of X and Y given Z is defined as

$$\begin{aligned} I(X;Y|Z) &:= H(X|Z) - H(X|Y,Z) \\ &= \mathbb{E}_{p(x,y,z)} \left(\log \frac{p(X,Y|Z)}{p(X|Z)p(Y|Z)} \right). \end{aligned}$$

Longer chain rules

Theorem 32.3. *Let $X_1, \ldots, X_n$ and Y be random variables. Then*

$$\begin{aligned} H(X_1, \ldots, X_n) &= \sum_{i=1}^{n} H(X_i|X_{i-1}, \ldots, X_1) \\ I(X_1, \ldots, X_n; Y) &= \sum_{i=1}^{n} I(X_i; Y|X_{i-1}, \ldots, X_1). \end{aligned}$$

For example, for $n = 3$ we obtain

$$\begin{aligned} H(X_1, x_2, X_3) &= H(X_1) + H(X_2|X_1) + H(X_3|X_2, X_1) \\ I(X_1, x_2, X_3; Y) &= I(X_1; Y) + I(X_2; Y|X_1) + I(X_3; Y|X_2, X_1). \end{aligned}$$

This example is more intuitive than the letters theorem

Information inequality

Theorem 32.4. *Let $p(x)$, $q(x)$, $x \in \mathcal{X}$ be two probability mass functions. Then*

$$D(p||q) \geq 0$$

Relative entropy is always non negative

with equality if and only if $p(x) = q(x)$ for all x.

Proof. The proof is based on Jensen's inequality:

$$\begin{aligned} -D(p||q) &= \mathbb{E}_p \left(\log \frac{q(x)}{p(x)} \right) \\ &\leq \log \mathbb{E}_p \left(\frac{q(x)}{p(x)} \right) \\ &= \log \sum_x q(x) \\ &\leq 0. \end{aligned}$$

□

Shannon's inequality

Corollary 32.5 (Non-negativity of mutual information)**.** *For any two random variables X and Y,*

$$I(X;Y) \geq 0,$$

X will never tell you a negative amount about Y and vice versa. At worst, they will be independent (=0)

with equality if and only if X and Y are independent.

Using a similar proof, we obtain results for the conditional mutual information.

Corollary 32.6 (Shannon's inequality)**.** *$D(p(y|x)||q(y|x)) \geq 0$ with equality if and only if $p(y|x) = q(y|x)$ for all y and x with $p(x) > 0$. Thus,*

$$I(X;Y|Z) \geq 0,$$

with equality if and only if X and Y are conditionally independent given Z.

More inequalities First, the proof that the uniform distribution maximises the entropy was using the non-negativity of the relative entropy in disguise. Let's do it explicitly this time.

Theorem 32.7. *$H(X) \leq \log|\mathcal{X}|$, with equality if and only if X has a uniform distribution over $\mathcal{X}$.*

Proof. Let $u(x) = \frac{1}{|\mathcal{X}|}$ be the uniform distribution on $\mathcal{X}$, then

$$H(X) = \log|\mathcal{X}| - D(p||u) \leq \log|\mathcal{X}|.$$

□

Secondly, knowing the value of a random variable Y can only give out some information about another random variable X on average. This is usually stated as "conditioning reduces entropy." Note that the average here is important: certain values y of Y can be "red herrings." Indeed, suppose that $p(x) = p(x') = 0.5$, but $p(x|y) = 0.99$ and $p(x'|y) = 0.01$. Suppose that after a certain experiment, I obtain $X = x'$ and $Y = y$; then if I showed you the value $Y = y$, you would bet that $X = x$!

Theorem 32.8 (Conditioning reduces entropy)**.**

Remember that this is the EXPECTED (avg) value of X given Y

$$H(X|Y) \leq H(X)$$

Once you know Y, the uncertainty about X is less

with equality if and only if X and Y are independent.

ie Venn circles are separate

32.4 Exercises

Exercise 32.1 (Example of joint entropy and mutual information)**.** Let X and Y be random variables on $\{x_0, x_1\}$ and $\{y_0, y_1\}$, respectively. They have the following joint distribution:

$$\begin{aligned} p(x_0, y_0) &= 0 \\ p(x_0, y_1) &= \frac{1}{8} \\ p(x_1, y_0) &= \frac{3}{4} \\ p(x_1, y_1) &= \frac{1}{8}. \end{aligned}$$

Calculate $H(X)$, $H(Y)$, $H(X,Y)$ and $I(X;Y)$.

Exercise 32.2. Prove that for any random variables $X_1, \ldots, X_n$,

$$H(X_1, \ldots, X_n) \leq \sum_{i=1}^{n} H(X_i),$$

with equality if and only if the X_i are independent.

Exercise 32.3 (Entropy of functions of a random variable)**.** Let X be a discrete random variable and $g(X)$ be any function of that variable. Show that

$$H(X) \geq H(g(X)).$$

In other words, processing data cannot increase the amount of information. Hint: compute $H(X, g(X))$ in two different ways.

Random variables X, Y, and Z are said to form a Markov chain in that order (denoted as $X \to Y \to Z$) if the joint probability mas function can be written as

$$p(x, y, z) = p(x)p(y|x)p(z|y).$$

In other words, each variable only depends on the preceding variable in the chain: Z depends on Y, Y depends on X, and X is independent from Y and Z.

Exercise 32.4. Prove that if $Z = f(Y)$, then $X \to Y \to Z$.

Exercise 32.5. Prove that X and Z are conditionally independent given Y, i.e.

$$p(x,z|y) = p(x|y)p(z|y).$$

Exercise 32.6. Prove the following theorem and corollary, called the data processing inequality.

If $X \to Y \to Z$, then

$$I(X;Y) \geq I(X;Z).$$

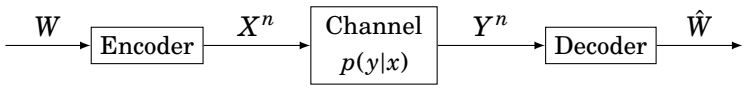
In particular, if $Z = g(Y)$, then $I(X;Y) \geq I(X;g(Y))$.

Why is this called the data processing inequality?

33 Capacity I: Definition and examples

33.1 Definition

A communication system A source produces a message W, encodes it into a sequence X^n and sends it through a channel. The receiver obtains a corrupted version Y^n and decodes it to some estimate $\hat{W}$ of W.



Discrete channel A discrete channel consists of an input alphabet $\mathscr{X}$, and output alphabet $\mathscr{Y}$, and a probability transition matrix $p(y|x)$ that expresses the probability of observing the output symbol y given that we send the symbol x.

The channel is called memoryless if the probability distribution of the output only depends on the input at that time and is conditionally independent of previous inputs or outputs.

The channel capacity of a discrete memoryless channel is defined as

$$C = \max_{p(x)} I(X;Y).$$

p(y)= \sum_x p(x) p(y|x)
Once p(x) is decided then p(y) is fixed/determined
p(x|y) = p(x) p(y|x) is fixed

Choosing p(x) gives us different ways of encoding - finding best encoding to maximise info being transmitted

Intuitively, the mutual information $I(X;Y)$ is the amount of information the output Y has about the input X: that's the amount of information that has been successfully transmitted through the channel. Changing the input distribution $p(x)$ amounts to different sorts of "encoding" the data: finding the best input that allows for the most information transmitted through the channel.

The mathematical definition of the capacity makes sense: for a given channel, the output probability distribution $p(y)$ is a function of the input probability distribution $p(x)$:

$$p(y) = \sum_x p(x)p(y|x),$$

and hence the mutual information $I(X;Y)$ is a function of $p(x)$.

33.2 Examples

Noiseless binary channel Suppose that the channel can carry one bit and it is perfect: $p(y|x) = 1$ if $y = x$. Then $I(X;Y) = H(X)$, which is maximised for $p(x) = (1/2, 1/2)$. Thus,

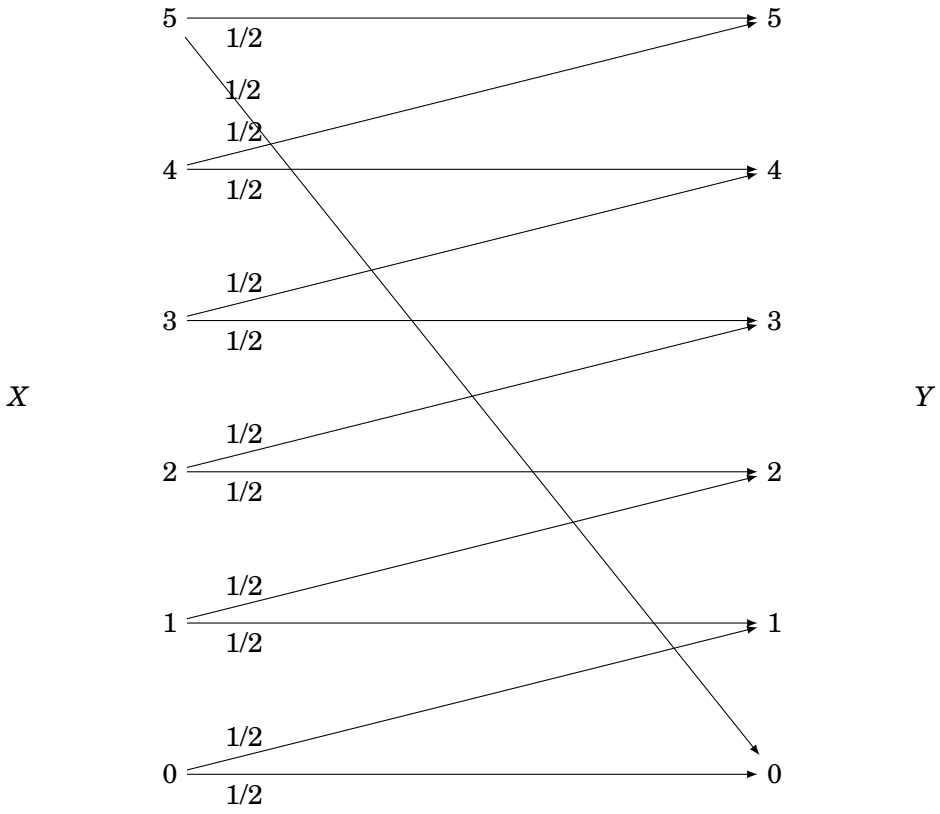
$$C = 1.$$

Channel capacity i.e. maximum mutual information

Noisy typewriter The alphabet is $\mathbb{Z}_{2q}$ and the input is either unchanged with probability 1/2 or turned to the next symbol with probability 1/2: alphabet is 0-> 2q-1 i.e. 2q symbols

$$p(x|x) = p(x+1 \mod 2q|x) = \frac{1}{2}.$$

The typewriter for $2q = 6$ is given below.



Theorem 33.1. *The capacity of the noisy typewriter is* $\log q$.

Proof. The proof contains two parts.

Firstly, we prove that the mutual information is always upper bounded by $\log q$. We have

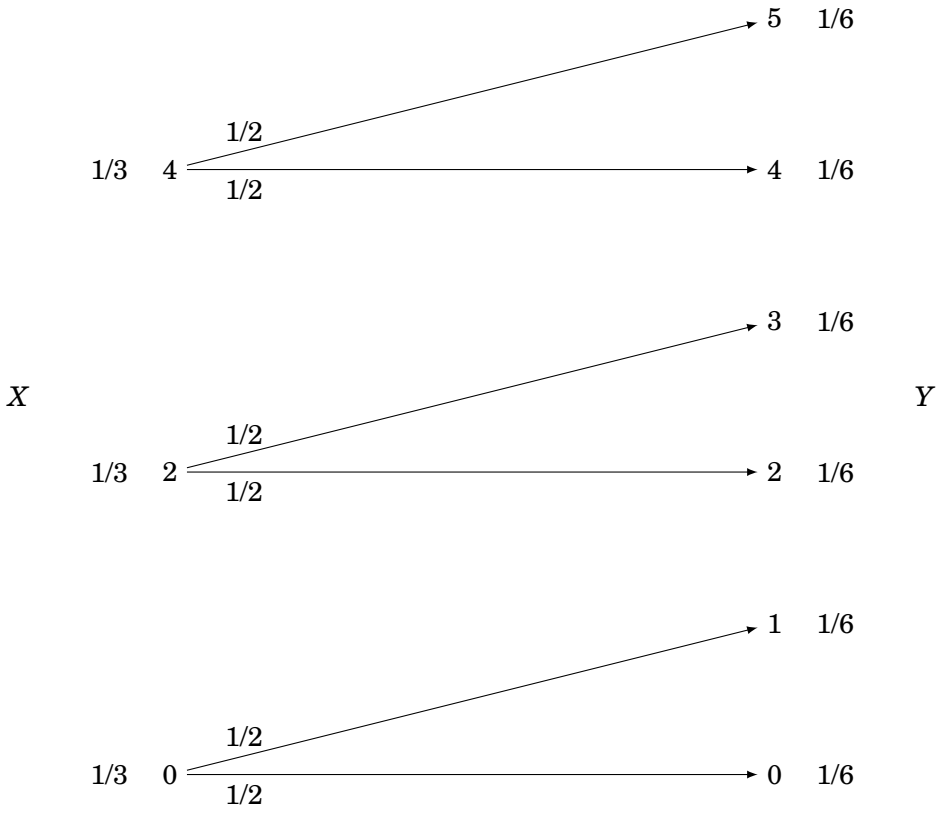
because upper bounded when a uniform distribution:
recall: H(X)<= log|X|, with equality if and only if X has a uniform distribution over X.

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H(Y) - 1 \le \log(2q) - 1 = \log q. \end{aligned}$$

The second part is to exhibit an input distribution that matches our upper bound. Here, we reach capacity if we send every other symbol equiprobably, i.e.

$$p(x) = \begin{cases} \frac{1}{q} & \text{if } x \equiv 0 \mod 2 \\ 0 & \text{if } x \equiv 1 \mod 2, \end{cases}$$

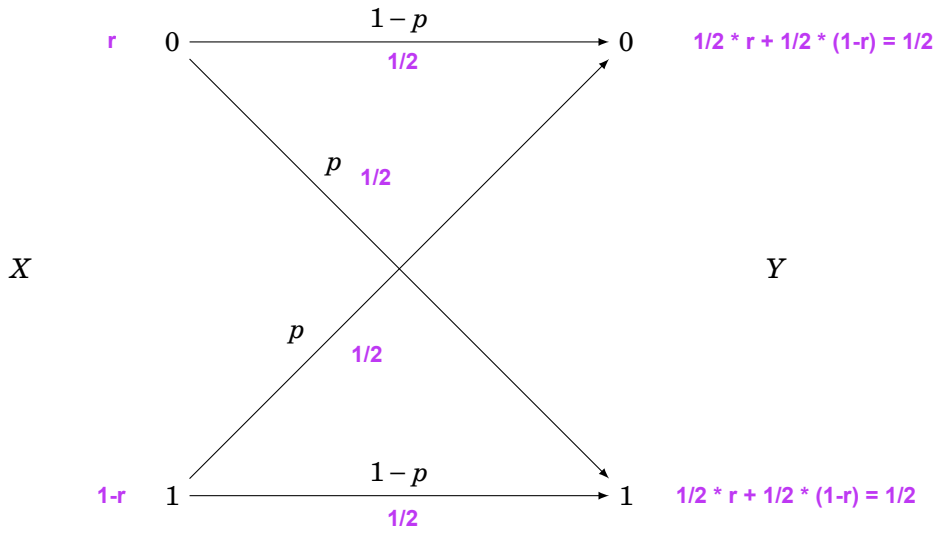
then, as we see below, Y is uniform over $\mathbb{Z}_q$. Thus $H(Y) = \log(2q)$ and $I(X;Y) = \log q$.



□

Binary symmetric channel Binary Symmetric Channel: $x, y \in \{0,1\}$,

$$p(x|x) = 1-p, \quad p(1-x|x) = p.$$



The capacity $C(p)$ of the BSC has a few intuitive properties.

1. $C(0) = 1$: in this case the channel is perfect. Crossover probability is 0
2. $C(p) = C(1-p)$: making that switch amounts to changing the roles of 0 and 1 at the destination.
3. $C(1/2) = 0$: in this case, Y receives random noise. If the number on each channel = 1/2: example in purple
Capacity = 0 - can't transmit any info down channel

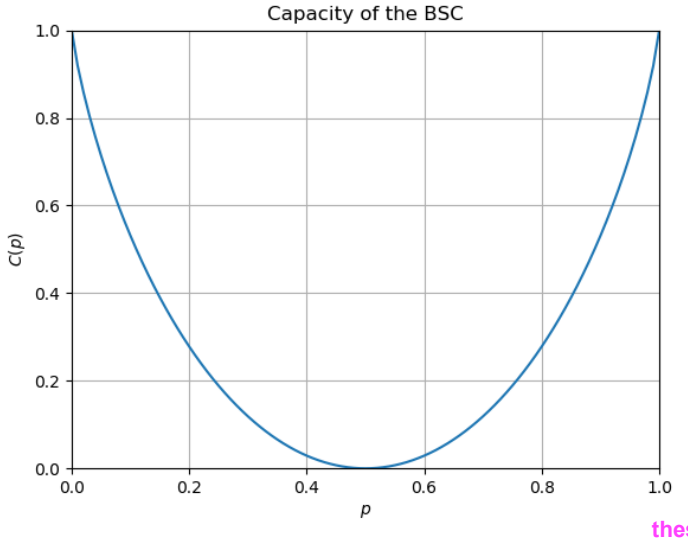
Theorem 33.2. *The capacity of a BSC with crossover probability p is*

$$C(p) = 1 - H(p).$$

Proof.

$$I(X;Y) = H(Y) - H(p) \le 1 - H(p),$$

with equality if and only if $H(Y) = 1$, which is achieved when the input has a uniform distribution. □



these 'powers' of n mean sequence length of n

33.3 The channel coding theorem

The n-th extension of the discrete memoryless channel (DMC) is the channel $(\mathcal{X}^n, p(y^n|x^n), \mathcal{Y}^n)$, where

$$p(y_k|x^k, y^{k-1}) = p(y_k|x_k), \quad k = 1, \dots, n.$$

Because memoryless, previous sequences (i.e. y^(k-1)) makes no difference to probability of output (still just probability of output given input)

We shall use the channel without feedback, i.e. $p(x_k|x^{k-1}, y^{k-1}) = p(x_k|x^{k-1})$ hence

$$p(y^n|x^n) = \prod_{i=1}^{n} p(y_i|x_i).$$

without feedback: x(i)s dont know anything about output y(i)s

input doesn't depend on output

Definition 33.3. *An (M,n) code for the channel $(\mathcal{X}, p(y|x), \mathcal{Y})$ consists of the following:*

1. *An index set $\{1,\dots,M\}$.* Set of messages source wishes to encode
2. *An encoding function $X^n : \{1,\dots,M\} \to \mathcal{X}^n$, yielding codewords $X^n(1),\dots,X^n(M)$. The set of codewords is called the codebook.* Codeword for each message. Assign each message with sequence of symbols that can be transmitted to channel
3. *A decoding function $g : \mathcal{Y}^n \to \{1,\dots,M\}$.* From output of channel, guess message

The rate of an (M,n) code is

$$R := \frac{\log M}{n} \text{ bits per transmission.}$$

Amount of useful info transmitting dividing by amount of stuff sending to channel (M = messages, n = number of bits)

Definition 33.4. *The conditional probability of error is*

$$\begin{aligned}\lambda_w &:= \mathbb{P}(g(Y^n) \neq w | X^n = X^n(w)) \\ &= \sum_{y^n : g(y^n) \neq w} p(y^n|x^n(w)).\end{aligned}$$

Probability of error given that input message is w

The maximal probability of error $\lambda^{(n)}$ for an (M,n) code is

$$\lambda^{(n)} := \max\{\lambda_w : w \in \{1,\dots,M\}\}.$$

As we use longer codes we want error to tend to 0

We say a rate R is achievable if there exists a sequence of $(2^{nR}, n)$ codes s.t. $\lambda^{(n)} \to 0$.

The channel coding theorem states that the channel capacity is equal to the maximum rate with which we can transmit data reliably (i.e., with negligible probability of error).

Theorem 33.5 (The channel coding theorem)**.** *All rates below capacity C are achievable. Specifically, for every rate $R < C$, there exists a sequence of $(2^{nR}, n)$ codes with maximum probability of error $\lambda^{(n)} \to 0$.*

Conversely, any sequence of $(2^{nR}, n)$ codes with $\lambda^{(n)} \to 0$ must have $R \le C$.

IT	ECC
Full knowledge of the channel: p(y|x)	"At most t errors will occur"
n -> infty	Finite block length n
Probability of error Pe -> 0	Pe = 0
No consideration of complexity	Efficient encoding and decoding
No construction of the actual code!	Lots of explicit constructions

33.4 See further

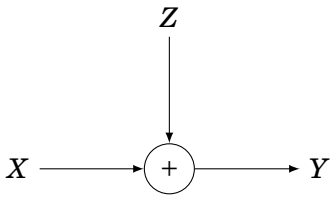
Proof of the theorem A sketch of the proof of the channel coding theorem can be found in Appendix 41. That is not examinable.

33.5 Exercises

Exercise 33.1. Consider the BSC, with input probability $X \sim \mathrm{B}(q)$. Write explicitly the distributions of Y and (X,Y). Give the explicit formulas for

$$H(X), \quad H(Y), \quad H(X|Y), \quad H(Y|X), \quad I(X;Y), \quad H(X,Y).$$

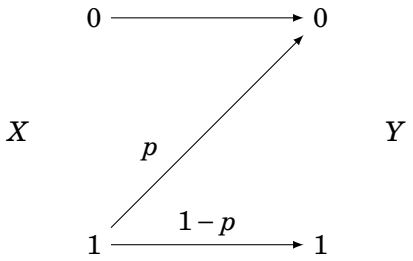
Exercise 33.2. Determine the capacity of the following additive channel, where $X \in \{0,1\}$ and $\mathbb{P}(Z = 0) = \mathbb{P}(Z = a) = \frac{1}{2}$ for some real number $a > 0$.



Note: we also use the notation

$$Z = \begin{pmatrix} 0, & a \\ \frac{1}{2}, & \frac{1}{2} \end{pmatrix}.$$

Exercise 33.3. The Z-channel with error probability p, also called binary asymmetric channel, is a channel where the transition $0 \to 1$ never occurs.



What is the capacity of this channel?

34 Capacity II: Multiple access and broadcast channels

Network information theory A system with many senders and many receivers contains many new elements in the communication problem: interference, cooperation and feedback. These issues are the domain of Network Information Theory.

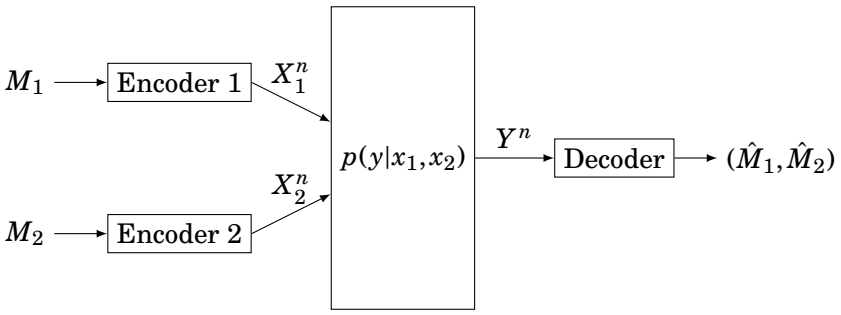
Problem: given many senders and receivers and a channel transition matrix which describes the effects of the interference and the noise in the network, determine whether or not the sources can be transmitted over the channel. This requires determining the capacity region of the network.

Determining the capacity region of systems with multiple senders/receivers is the main challenge of research in information theory nowadays. We will only look at some very special cases.

34.1 Multiple access channel

Definition A Multiple Access Channel consists of three alphabets $\mathcal{X}_1, \mathcal{X}_2, \mathcal{Y}$ and a probability transition matrix $p(y|x_1,x_2)$. Thus, there are two senders and one receiver on this channel.

Codewords of length n



Codes A $((2^{nR_1},2^{nR_2}),n)$ code for the MAC consists of two sets of integers $\mathcal{W}_1 = \{1,2,\dots,2^{nR_1}\}$ and $\mathcal{W}_2 = \{1,2,\dots,2^{nR_2}\}$ called the messages sets and two encoding functions

$$X_1 : \mathcal{W}_1 \to \mathcal{X}_1^n,$$
$$X_2 : \mathcal{W}_2 \to \mathcal{X}_2^n,$$

and a decoding function

$$g : \mathcal{Y}^n \to \mathcal{W}_1 \times \mathcal{W}_2.$$

The average probability of error is

$$P_e^{(n)} = \frac{1}{2^{nR_1+nR_2}} \sum_{(w_1,w_2)\in \mathcal{W}_1\times\mathcal{W}_2} \mathbb{P}(g(Y^n) \neq (w_1,w_2) | (w_1,w_2) \text{ was sent}).$$

Average probability of error tends to 0

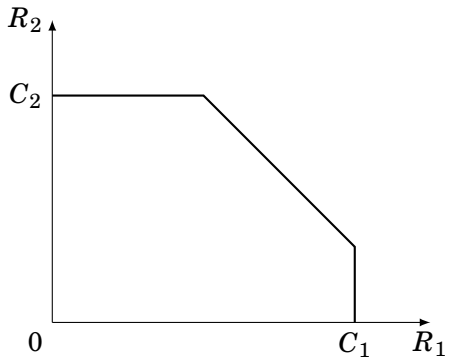
A rate pair (R_1,R_2) is achievable if there exists a sequence of $((2^{nR_1},2^{R_2}),n)$ codes with $P_e^{(n)} \to 0$.

Capacity region The capacity region of the MAC is the closure of the set of achievable (R_1,R_2) rate pairs.

Theorem 34.1. *The capacity region of a MAC $(\mathcal{X}_1 \times \mathcal{X}_2, p(y|x_1,x_2), \mathcal{Y})$ is the closure of the convex hull of all (R_1,R_2) satisfying*

$$R_1 < I(X_1;Y|X_2),$$
$$R_2 < I(X_2;Y|X_1),$$
$$R_1 + R_2 < I(X_1,X_2;Y)$$

for some product distribution $p_1(x_1)p_2(x_2)$ on $\mathcal{X}_1 \times \mathcal{X}_2$.



The inequality

$$R_1 + R_2 < I(X_1,X_2;Y)$$

corresponds to viewing the channel as a single-user channel with source (X_1,X_2). This inequality is simply the channel coding theorem for that channel.

The inequality

$$R_1 < I(X_1;Y|X_2)$$

means that once we know X_2, the channel then reduces to a single-user channel with source X_1. The channel coding theorem then yields $R_1 < I(X_1;Y|X_2)$. The inequality $R_2 < I(X_2;Y|X_1)$ is similar.

Example: independent channels Assume that we have two independent BSC, one from sender 1 (with crossover probability p_1), the other from sender 2 (with p_2). Since there is no interference between the senders, we can achieve any pair (R_1, R_2) as long as each rate corresponds to each BSC. Thus, the capacity region is then defined by the inequalities

Capacity region: rectangle

$$R_1 < 1 - H(p_1)$$
$$R_2 < 1 - H(p_2).$$

Binary multiplier channel In this case, $\mathcal{X}_1 = \mathcal{X}_2 = \mathcal{Y} = \{0,1\}$. The output is given by

Opposite of independent channels

$$Y = X_1 X_2.$$

Like a traffic light system

If $X_2 = 1$, then $Y = X_1$ and hence X_1 can transmit at a rate of 1 bit per transmission. Thus $R_1 \le 1$ (and similarly, $R_2 \le 1$.) Since Y is binary, $R_1 + R_2 \le 1$. By timesharing, we can achieve any rate pair $(R, 1-R)$.

Timesharing: for any n and $k \le n$, let

Rates sum to 1

Capacity region: perfect right angled triangle

$$X_{1,1} \dots X_{1,k} \text{ i.i.d} \sim \text{B}(1/2) \quad \text{2- Sends k messages}$$
$$X_{1,k+1} \dots X_{1,n} = 1, \quad \text{3 - Green light}$$
$$X_{2,1} \dots X_{2,k} = 1 \quad \text{1- green light means start sending}$$
$$X_{2,k+1} \dots X_{2,n} \text{ i.i.d} \sim \text{B}(1/2), \quad \text{4- send k messages}$$

This achieves a rate of $(k/n, 1 - k/n)$.

Generalisation to m users The situation is similar for $m \ge 2$ users. Let $[m] = \{1, 2, \dots, m\}$. For any $S \subseteq [m]$, denote its complement as $\bar{S}$, denote $R_S = \sum_{i \in S} R_i$ and $X_S = \{X_i : i \in S\}$.

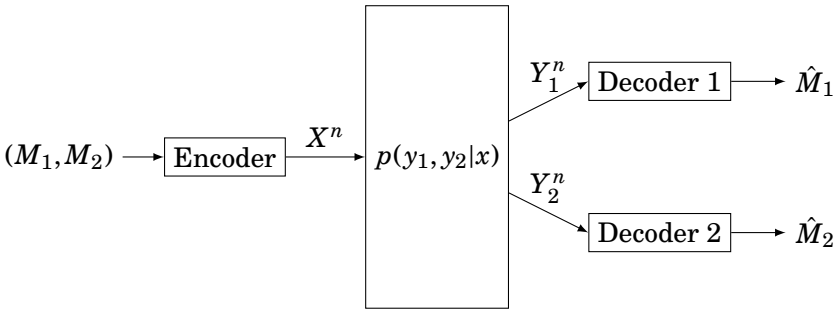
Theorem 34.2. *The capacity region of the m-user MAC is the closure of the convex hull of the rate vectors satisfying*

$$R_S < I\left(X_S; Y \mid X_{\bar{S}}\right)$$

for all $S \subseteq [m]$, for some product distribution $p_1(x_1) \dots p_m(x_m)$.

34.2 Broadcast channel

The broadcast channel



Some definitions for the broadcast channel A broadcast channel consists of three alphabets $\mathcal{X}, \mathcal{Y}_1, \mathcal{Y}_2$ and a probability transition matrix $p(y_1, y_2|x)$. The broadcast channel is memoryless if

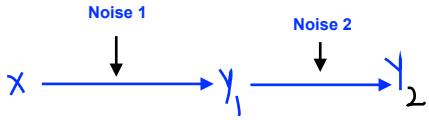
$$p(y_1^n, y_2^n|x^n) = \prod_{i=1}^{n} p(y_{1i}, y_{2i}|x_i).$$

$((2^{nR_1}, 2^{nR_2}), n)$ codes are defined as you might expect (with one encoder and two decoders this time). The probability of error is "disjunctive":

$$P_e^{(n)} = \mathbb{P}(g_1(Y_1^n) \neq M_1 \text{ or } g_2(Y_2^n) \neq M_2).$$

A rate pair (R_1, R_2) is achievable if there exists a sequence of codes with that rate pair and $P_e^{(n)} \to 0$.

Physically degraded broadcast channels A broadcast channel is called physically degraded if one of the following equivalent conditions is satisfied:



1. Y_2 is a random degradation of Y_1;
2. $X \to Y_1 \to Y_2$ (Markov chain); Y2 only depends on Y1 and Y1 only depends on X
3. $p(y_1, y_2|x) = p(y_1|x)p(y_2|y_1)$. Description of condition 2 mathematically

Theorem 34.3. *The capacity region for sending independent information over the degraded broadcast channel $X \to Y_1 \to Y_2$ is the convex hull of the closure of all (R_1, R_2) satisfying*

$$R_2 < I(U;Y_2)$$
$$R_1 < I(X;Y_1|U),$$

for some joint distribution $p(u)p(x|u)p(y_1,y_2|x)$, where the auxiliary random variable U is defined over $\mathcal{U}$ with $|\mathcal{U}| \le \min\{|\mathcal{X}|, |\mathcal{Y}_1|, |\mathcal{Y}_2|\}$.

Don't worry about this inequality

A bit of explanation The main idea is superposition coding, explained below.

The auxiliary variable U serves as a cloud centre that can be distinguished by both Y_1 and Y_2. Each cloud consists of 2^{nR_1} codewords X^n distinguishable by the receiver Y_1. Y_2 can only see the clouds, while Y_1 can see the individual codewords within the clouds.

Using superposition coding, one can achieve any rate pair s.t.

$$R_2 < I(U;Y_2), \quad \text{U is what was sent to Y2}$$
$$R_1 \le I(X;Y_1|U), \quad \text{Y1 decodes U}$$
$$R_1 + R_2 < I(X;Y_1) \quad \text{Overall amount of info can transmit to Y1}$$

for some $p(u,x)$.

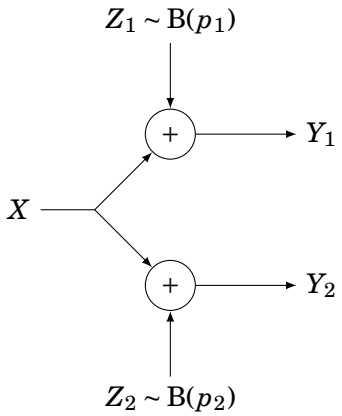
(The bound on $|\mathcal{U}|$ follows from convex set theory; this is outside the scope of this lecture.)

34.3 Exercises

Exercise 34.1. Suppose the rate pairs (R_1, R_2) and (R_1', R_2') are achievable for a DM-MAC. Show that $(\lambda R_1 + (1-\lambda)R_1', \lambda R_2 + (1-\lambda)R_2')$ is also achievable for any $\lambda \in [0,1]$.

Exercise 34.2. Prove that the capacity region of the DM-BC depends on $p(y_1, y_2|x)$ only through the conditional marginal distributions $p(y_1|x)$ and $p(y_2|x)$.

Exercise 34.3. The Binary Symmetric Broadcast Channel (BS-BC) is a Broadcast channel where each receiver sees a BSC. It is illustrated below.



The noises Z_1 and Z_2 are independent; without loss, you may assume $p_1, p_2 \le 1/2$.

Prove that the BSC-BC is degraded.

Exercise 34.4. Prove that the whole capacity region of the degraded broadcast channel is indeed given by superposition coding. You may use the results from the lecture.

35 Compression I: Entropy coding

35.1 Source codes

Source code We wish to encode the values of a random variable X into string of symbols over a finite alphabet $\mathscr{D} = \{0,1,\ldots,D-1\}$. In this lecture, we will use $\mathscr{D} = \{0,1\}$, but the analysis is the same for any D.

A binary source code for a random variable X is a mapping $C : \mathscr{X} \to \{0,1\}^*$. We denote the codeword corresponding to x as $C(x)$ and its length as $l(x)$. The expected length of the code is

Assigns every input symbol to a sequence

$$L(C) := \mathbb{E}(l(x)) = \sum_{x \in \mathscr{X}} p(x) l(x).$$

Uniquely decodable codes Since we are going to encode sequences of symbols $x_1 \ldots x_n$ from X, we need to make sure that no two sequences will be mapped to the same string of bits. We denote $x^n = x_1 x_2 \ldots x_n$ and

$$C(x^n) := C(x_1) C(x_2) \ldots C(x_n)$$

as the concatenation of these n codewords. The code is uniquely decodable if

$$x^n \neq y^m \Rightarrow C(x^n) \neq C(y^m).$$

For any two input sequences, codewords are different - non ambiguous

Prefix codes The i-th prefix of $x^n = x_1 \ldots x_n$ is simply $x^i := x_1 \ldots x_i$. A code is a prefix code (a.k.a. instantaneous code) if no codeword is a prefix of any other codeword. Clearly, a prefix code is uniquely decodable.

prefix codes are a pecial case: If no codeword is a prefix of another codeword

35.2 Kraft inequality

Binary trees A binary tree is a rooted tree where each parent has at most two children.

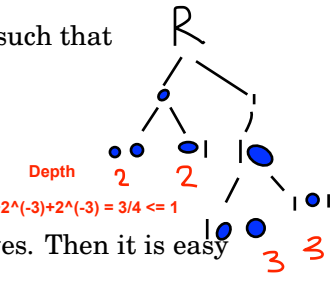
Prefix codes and binary trees are equivalent.

1. From a binary tree T, label the edges from a parent to each child as 0 and 1. Let the leaves be $\mathscr{X} = \{x_1, \ldots, x_k\}$, then the code $C_T : \mathscr{X} \to \{0,1\}^*$ defined by the labels from the root to each leaf is a prefix code.

2. Conversely, from a prefix code C, it is easy to reconstruct a binary tree T such that $C = C_T$.

Kraft sequences A (binary) Kraft sequence is a sequence of integers $L = l_1, \ldots, l_k$ such that

$$\sum_{i=1}^{k} 2^{-l_i} \leq 1.$$



Binary trees and Kraft sequences are equivalent in the following sense.

1. From a binary tree T, let $L_T = l_1, \ldots, l_k$ denote the respective depths of its leaves. Then it is easy to show that L_T is a Kraft sequence by induction.

2. Conversely, given any Kraft sequence $L = l_1, \ldots, l_k$, we can construct a binary tree T such that $L_T = L$ as follows. Call the first node of depth l_1 as codeword 1, and remove its descendants from the tree. Then call the first remaining node of level l_2 as codeword 2, etc.

The triple equivalence We have a triple equivalence

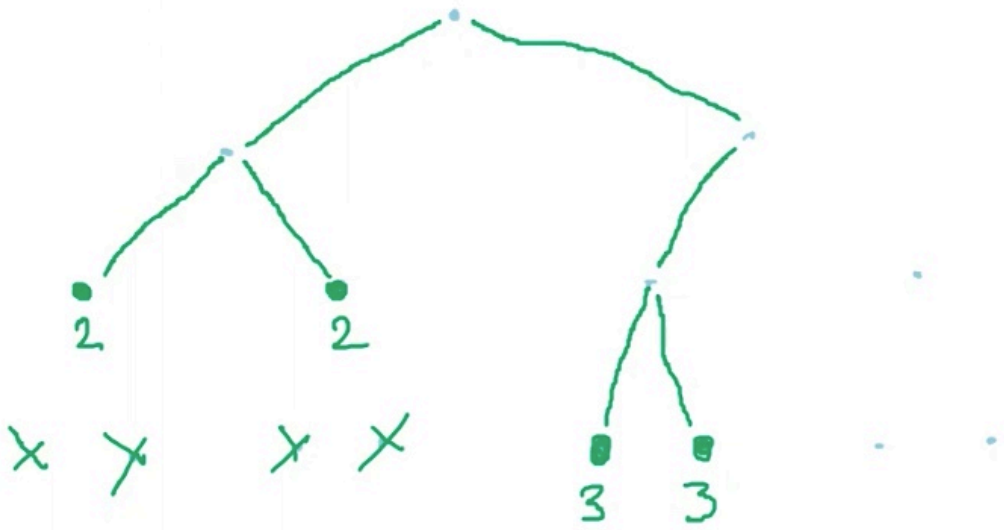
$$\text{Prefix code } C_T \equiv \text{Binary tree } T \equiv \text{Kraft sequence } L_T.$$

As a result, the codeword lengths of any prefix code form a Kraft sequence, i.e.

$$\sum_{x \in \mathscr{X}} 2^{-l(x)} \leq 1.$$

This is the Kraft inequality for prefix codes.

Kraft sequence (2, 2, 3, 3)



Kraft inequality for uniquely decodable codes

Theorem 35.1. *The codeword lengths of any uniquely decodable code form a Kraft sequence, i.e.*

Generalisation of previous part: this isn't just prefix codes

$$\sum_{x\in\mathscr{X}} 2^{-l(x)} \le 1.$$

Can find a prefix code with the exact same lengths as uniquely decodable code so will form a Kraft sequence

Note:

1. The inequality was first proved by Kraft for prefix codes; it was then generalised to uniquely decodable codes by McMillan.
2. We have proved the so-called converse of the Kraft inequality: any Kraft sequence is the sequence of lengths of codewords of a prefix code.
3. This shows that prefix codes are the best kind of uniquely decodable codes.

Proof of the Kraft inequality for uniquely decodable codes; non-examinable. We prove $\sum_x 2^{-l(x)} \le 1+\epsilon$ for any $\epsilon > 0$. Denote $l_{\max} := \max_{x\in\mathscr{X}} l(x)$ and let k such that $(kl_{\max})^{1/k} \le 1+\epsilon$. The number of sequences x^k mapped to codewords of length m is then at most 2^m. We have

Can't use trees to prove

$$\begin{aligned}\left(\sum_{x\in\mathscr{X}} 2^{-l(x)}\right)^k &= \sum_{x^k\in\mathscr{X}^k} 2^{-l(x^k)} \\ &= \sum_{m=1}^{kl_{\max}} |\{x^k : l(x^k) = m\}| \cdot 2^{-m} \\ &\le kl_{\max}.\end{aligned}$$

Thus $\sum_{x\in\mathscr{X}} 2^{-l(x)} \le (kl_{\max})^{1/k} \le 1+\epsilon$. □

35.3 Compact codes

The source coding theorem

Theorem 35.2. *Let $L^*(X)$ be the minimum expected length of a binary uniquely decodable code for X. Then*

$$H(X) \le L^*(X) < H(X)+1.$$

Proof. We first prove the lower bound. Let C^* be a uniquely decodable code with codeword lengths l_i^* and expected length L^*. We have

$$\begin{aligned}H(X) - L^*(X) &= \sum_i p_i \log\frac{1}{p_i} - \sum_i p_i l_i^* \\ &= \sum_i p_i \log\frac{2^{-l_i^*}}{p_i} \\ &\le \log\sum_i p_i \frac{2^{-l_i^*}}{p_i} \\ &\le \log 1 = 0.\end{aligned}$$

By Jensens inequality: expected value of log is no more than log of expected value

By craft inequality - no more than 1

For the upper bound, we use the converse of the Kraft inequality: if $l = l_1,\dots,l_k$ is a Kraft sequence, then there exists a prefix code with sequence of codeword lengths equal to l. The Shannon code is a (suboptimal) prefix code where

$$l(x) = \left\lceil \log\frac{1}{p(x)} \right\rceil.$$

This is a Kraft sequence:

$$\sum_{x\in\mathscr{X}} 2^{-\left\lceil \log\frac{1}{p(x)} \right\rceil} \le \sum_{x\in\mathscr{X}} p(x) = 1.$$

As such, the Shannon code always exists! We have

$$\begin{aligned} L^*(X) &\leq \sum_{x\in\mathcal{X}} p(x)\left\lceil \log\frac{1}{p(x)}\right\rceil \\ &< \sum_{x\in\mathcal{X}} p(x)\left(\log\frac{1}{p(x)}+1\right) \\ &= H(X)+1. \end{aligned}$$

□

Exercise 35.3 will show how to construct the Shannon code.

Triple equivalence: optimal case Say a prefix code C is optimal if it is compact for some probability distribution. Say a binary tree is optimal (sometimes referred to as full) if every parent has exactly two children. Say a Kraft sequence is optimal if $\sum_{i=1}^{k} 2^{-l_i} = 1$.

Theorem 35.3. *For any binary tree T,*

$$C_T \text{ is optimal} \iff T \text{ is optimal} \iff L_T \text{ is optimal}.$$

Proof. 1. C_T is optimal $\implies$ T is optimal . If T is not optimal, then let $v \in T$ have exactly one child u. Without loss, the edge uv is labelled 0 in C_T. Removing that 0 in C_T (which corresponds to contracting the edge in T yields a prefix code with a smaller expected length (whatever the distribution), thus C_T is not optimal.

2. T is optimal $\implies$ L_T is optimal. Easy proof by induction.

3. L_T is optimal $\implies$ C_T is optimal. If L_T is an optimal Kraft sequence, then the sequence $\{p_i = 2^{-l_i}\}$ is a probability distribution. If $p(X=i) = p_i$, we obtain

$$L(C_T) = \sum_i p_i l_i = -\sum_i p_i \log p_i = H(X),$$

and hence C_T is compact for X.

□

35.4 See further

Expected length of Huffman code As we saw in Lecture 1, Huffman codes are compact, i.e. they reach $L^*(X)$ for any X. Note that $L^*(X)$ is not necessarily equal to the entropy of X (see Exercise 35.1). However, some bounds can be obtained: we have seen that $L^*(X) - H(X) < 1$, but for instance Gallager showed that [7]

$$L^*(X) - H(X) \leq p_1 + 0.086.$$

Bounds on the redundancy $L^*(X) - H(X)$ of Huffman codes are reviewed in [12].

35.5 Exercises

Exercise 35.1. We have seen that the optimal expected length of a code is upper bounded by $H(X)+1$. Show that this is tight, i.e. for any $\epsilon > 0$, give an example of a source X for which the optimal length is at least $H(X)+1-\epsilon$.

Exercise 35.2. Let $X_1, \ldots, X_n$ be i.i.d. $\sim p(x)$ and denote the minimum expected codeword length per input symbol

$$L_n := \mathbb{E}\left(\frac{1}{n} l(X_1, \ldots, X_n)\right).$$

Then prove that

$$L_n \to H(X).$$

Exercise 35.3. Here is a construction of the Shannon code, sometimes referred to as the Shannon-Fano or Shannon-Fano-Elias code. Let $\mathcal{X} = \{1,\dots,m\}$ and X be a random variable on $\mathcal{X}$ with $p_i := \mathbb{P}(X = i)$ sorted such that $p_1 \geq \cdots \geq p_m$. Define

$$F_i := \sum_{k=1}^{i-1} p_i.$$

Then the codeword for i is the number $F_i \in [0,1]$ rounded down to $l_i := \lceil \log \frac{1}{p_i} \rceil$ bits.

1. Show that the Shannon code is a prefix code.
2. Give the Shannon code for the distribution $(1/2, 1/4, 1/8, 1/8)$. What is the expected length? How does it compare with the entropy?
3. Give the Shannon code for the distribution $(7/16, 3/16, 3/16, 2/16, 1/16)$. What is the expected length? How does it compare with the entropy?
4. The Shannon code can be viewed as an ancestor of arithmetic coding: why? What is the major difference between Shannon coding and arithmetic coding?

36 Compression II: Rate distortion theory

36.1 Introduction

We are now interested in the fundamental limits of lossy compression. First, we need a notion of how much is lost by compression.

Distortion

Definition 36.1. *A distortion measure is a mapping* Different models for different types of errors

$$d : \mathcal{X} \times \hat{\mathcal{X}} \to \mathbb{R}^+.$$

Original data x end data

A distortion measure is bounded if $d(x,\hat{x}) < \infty$ for all $x, \hat{x}$. In most cases, $\mathcal{X} = \hat{\mathcal{X}}$ and $d(x,x) = 0$ for all $x \in \mathcal{X}$. The two main distortion measures are: Bounded if measure always an actual number. 0 often means no errors

1. Hamming distortion (usually for discrete-valued data)

$$d(x,\hat{x}) = \begin{cases} 0 & \text{if } x = \hat{x}, \\ 1 & \text{if } x \neq \hat{x}. \end{cases}$$

2. Squared error distortion (usually for continuous-valued data)

$$d(x,\hat{x}) = (x - \hat{x})^2.$$ for continuous and unbounded data

Rate distortion function We now need to define the rate at which we can compress or quantize while keeping a reasonable distortion. In general we are interested in compressing a whole sequence of messages from the source X. How much can we destroy x by compressing it

Definition 36.2. *The rate distortion function for a source X and distortion measure d is*

$$R(D) := \min\{I(X;\hat{X}) : \mathbb{E}(d(x,\hat{x})) \leq D\}.$$ D is how much distortion we allow
So, minimum mutual info between X and X_hat so that expected distortion is less than variable D

This is the lowest rate to which we can compress (or quantize) the source X if we allow an average distortion of D.

Example: Bernoulli source with Hamming distortion

Theorem 36.3. *The rate distortion function for a* B(p) *source with Hamming distortion is given by*

Entropy of output - entropy of crossover probability

X is bernoulli variable

Avg distortion is between 0 and 1 cos hamming only outputs 0 or 1

$$R(D) = \begin{cases} H(p) - H(D), & 0 \le D \le \min\{p, 1-p\} \\ 0 & D > \min\{p, 1-p\}. \end{cases}$$

If lossless compression, so hamming distortion D = 0, then R(0) = entropy H(p) (amount of info contained)

If D = 1, R(1) = 0 i.e. if min distortion is 100% then, don't need to have anything in common.

Proof. W.l.o.g., $p < 1/2$. We wish to minimise $I(X;\hat{X})$; we use exclusive or $X \oplus \hat{X}$. The distortion condition is equivalent to $\mathbb{P}(X \neq \hat{X}) \le D$, hence

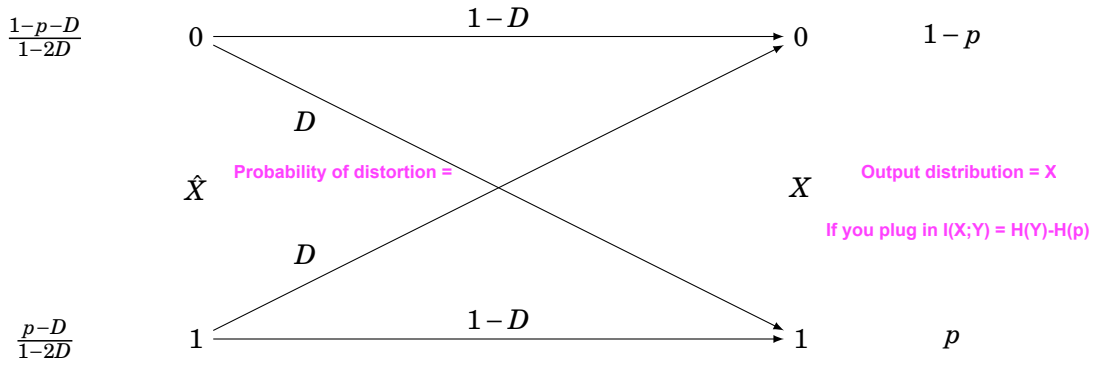
If we know that X and X_hat are different: If you know that X_hat is 0, then you know X = 1 and vv

Conditioning reduces entropy so this term is more than or equal to the one directly above

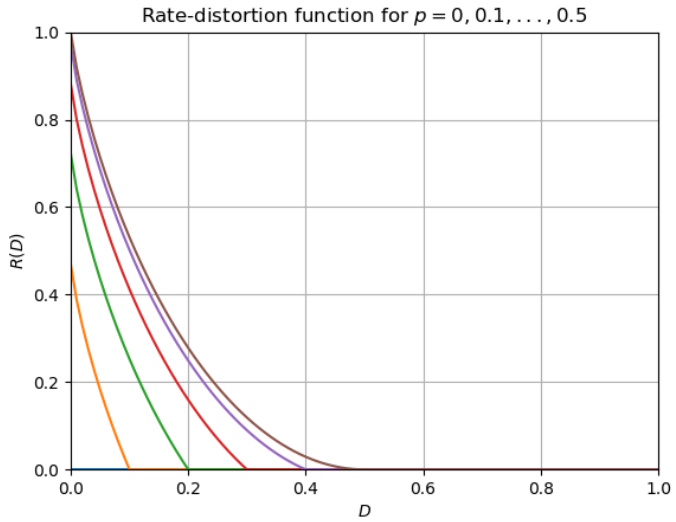
$$\begin{aligned} I(X;\hat{X}) &= H(X) - H(X|\hat{X}) \\ &= H(p) - H(X \oplus \hat{X}|\hat{X}) \\ &\ge H(p) - H(X \oplus \hat{X}) \\ &\ge H(p) - H(D). \end{aligned}$$

Thus $R(D) \ge H(p) - H(D)$.

Conversely, we achieve $R(D)$ by choosing $(X, \hat{X})$ to have the joint distribution given by the BSC below. This channel represents the reconstruction (decompression) operation. (For $D \ge p$, simply let $\hat{X} = 0$ with probability 1.)



□

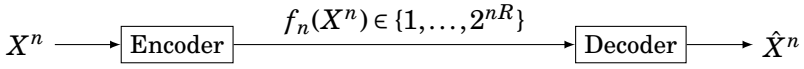


36.2 The rate distortion theorem

in distribution p(x)

Definitions Assume that we have a source that produces a sequence $X_1, \ldots, X_n$ i.i.d. $\sim p(x)$. The encoder describes the sequence X^n by an index $f_n(X^n) \in \{1, \ldots, 2^{nR}\}$. The decoder represents X^n by an estimate $\hat{X}^n$.

Encoder, f, that works on a length of n
Works at rate R



The distortion between x^n and $\hat{x}^n$ is the average

$$d(x^n,\hat{x}^n) = \frac{1}{n}\sum_{i=1}^{n} d(x_i,\hat{x}_i).$$

A $(2^{nR},n)$ rate distortion code consists of

1. An encoding function $f_n : \mathcal{X}^n \to \{1,\dots,2^{nR}\}$
2. A decoding (a.k.a. reproduction) function $g_n : \{1,\dots,2^{nR}\} \to \hat{\mathcal{X}}^n$.

The distortion of the code is

$$\mathbb{E}\left(d(X^n, g_n(f_n(X^n)))\right).$$

Expected value of distortion between Xn and X_hat_n

The set of n-tuples $\hat{X}^n(1) = g_n(1),\dots,\hat{X}^n(2^{nR}) = g_n(2^{nR})$ constitute the codebook, and $f_n^{-1}(1),\dots,f_n^{-1}(2^{nR})$ are the assignment regions.

The decoded codewords

Pre coded words - original data

The rate distortion theorem A rate distortion pair (R,D) is achievable if there exists a sequence of $(2^{nR},n)$ rate distortion codes (f_n,g_n) such that

If you can find a sequence of rate distortion codes R

$$\lim_{n\to\infty} \mathbb{E}\left(d(X^n, g_n(f_n(X^n)))\right) \le D.$$

Limit of expected distortion is less than D

Theorem 36.4 (Rate distortion theorem)**.** *The rate distortion function $R(D)$ is the infimum of rates R such that (R,D) is achievable.*

36.3 Covering codes

There is a connection between channel capacity and error-correcting codes. Similarly, there is a connection between rate distortion function and covering codes.

Sequences of length n over alphabet of size q

Covering radius A covering code is... a code: $\mathsf{C} \subseteq [q]^n$. The covering radius of C is defined as:

Covering code: codes are a fixed distance away from some codeword

$$\rho(\mathsf{C}) := \max_{\mathbf{x}\in[q]^n} \min_{\mathbf{c}\in\mathsf{C}} d_{\mathrm{H}}(\mathbf{x},\mathbf{c}).$$

It is the minimum radius ρ such that the spheres of radius ρ cover the whole space $[q]^n$.

Covering codes and lossy compression Let us see how this relates to lossy compression (and hence to rate distortion theory). For simplicity, we assume $|\mathsf{C}| = q^k$ for some k. Suppose we know an efficient representation of the codewords, i.e. a mapping $\phi : \mathsf{C} \to [q]^k$. Let ψ be the "reconstruction" function, defined by $\psi(\phi(\mathbf{c})) = \mathbf{c}$. Then we can compress any vector $\mathbf{x} \in [q]^n$ as a sequence in $[q]^k$ with Hamming distortion at most ρ as follows:

Compression

Hamming distance between original and decoded will be at maximum p/rho

1. Compression: Let $\mathbf{c} \in \mathsf{C}$ such that $d_{\mathrm{H}}(\mathbf{x},\mathbf{c}) \le \rho$, then compress $\mathbf{x}$ as $f(\mathbf{x}) = \phi(\mathbf{c}) = \mathbf{m} \in [q]^k$.
2. Decompression: $\hat{\mathbf{x}} = \psi(\mathbf{m}) = \mathbf{c}$.

Then the distortion satisfies $d_{\mathrm{H}}(\mathbf{x},\hat{\mathbf{x}}) \le \rho$.

For linear codes, we know how to go from a codeword $\mathbf{c} \in \mathsf{C}$ to its corresponding original message $\mathbf{m} \in \mathrm{GF}(q)^k$: if $\mathbf{G} = (\mathbf{I}_k|\mathbf{P})$, then $\phi(\mathbf{c}) = \mathbf{m} = (c_1,\dots,c_k)$. The reconstruction function ψ is simply the encoding function: $\psi(\mathbf{m}) = \mathbf{c} = \mathbf{m}\mathbf{G}$.

Covering radius and error-correction capability Since the spheres of radius $t = \lfloor (d_{\min} - 1)/2 \rfloor$ are packed in the space (they do not intersect), the covering radius is at least equal to the error-correction capability (also referred to as the packing radius) of C:

$$\rho \geq t.$$

Therefore, we have a beautiful three-way characterisation of perfect codes: a code is perfect if it satisfies one of the three equivalent properties below.

1. The spheres of radius t centred around the codewords cover the whole space.
2. The spheres of radius ρ centred around the codewords are packed, i.e. they do not intersect.
3. $\rho = t$.

In particular, the covering radius of any Hamming code is equal to 1. Reed-Solomon codes, on the other hand, are not good covering codes: we have $\rho = d_{\min} - 1 = n - k \geq 2t$.

Bounds on the cardinality of covering codes We are now interested in the minimum cardinality $K(n,q,\rho)$ of a code $C \subseteq [q]^n$ and covering radius (at most) ρ.

Theorem 36.5 (Sphere-covering bound)**.** *We have*

$$K(n,q,\rho) \geq \frac{q^n}{V(n,q,\rho)}.$$

Minimum cardinality = shortest representation of code

The proof is asked from you in Exercise 36.3.

Surprisingly, optimal covering codes are actually not very far away from sphere-covering bound (see [3, Theorem 12.1.2]):

$$K(n,q,\rho) \leq (\ln V(n,q,\rho) + 1)\frac{q^n}{V(n,q,\rho)}.$$

36.4 Exercises

Exercise 36.1. Consider a source X uniformly distributed in the set $\{1,2,\ldots,m\}$. Find the rate distortion function for this source with Hamming distortion.

Exercise 36.2 (Rate distortion function with infinite distortion)**.** Find the rate distortion function for $X \sim$ Bernoulli(1/2) and distortion

$$d(x,\hat{x}) = \begin{cases} 0 & \text{if } x = \hat{x}, \\ 1 & \text{if } x = 1, \hat{x} = 0, \\ \infty & \text{if } x = 0, \hat{x} = 1. \end{cases}$$

Exercise 36.3. Prove the sphere-covering bound.

Exercise 36.4. In this exercise, we determine the covering radius of Reed-Solomon codes: $\rho(\mathsf{RS}(n,k)) = n - k = d_{\min} - 1$ for all valid n and k.

1. As a warm-up, verify that the result holds for $k \in \{0, 1, n-1, n\}$.
2. Say a code C is maximal if there is no C' with $\mathsf{C} \subset \mathsf{C}'$ and $d_{\min}(\mathsf{C}') = d_{\min}(\mathsf{C})$. Verify that Reed-Solomon codes are maximal.
3. Let C be maximal. Prove that $\rho(\mathsf{C}) \leq d_{\min}(\mathsf{C}) - 1$.
4. Identify a vector $\mathbf{x}$ at distance $n - k$ from $\mathsf{RS}(n,k)$. Hint: I can think of $q^{k+1} - q^k$ such vectors for $k < n$.

37 Capacity and rate distortion I: Blahut-Arimoto algorithms

In this lecture, we describe two algorithms to compute the channel capacity and rate distortion function, respectively. Some technical details will be omitted; they can be found in [18, Chapter 9].

37.1 Alternating optimisation

We first describe an iterative algorithm to solve a rather generic optimisation problem. We shall then specify it for the capacity and the rate distortion function in the next subsections.

Setup A set $A \subseteq \mathbb{R}^n$ is convex if whenever $x, y \in A$, then the line segment between x and y also belongs to A. Let $A_1 \in \mathbb{R}^{n_1}$, $A_2 \in \mathbb{R}^{n_2}$ be convex sets. Consider the double supremum

$$\sup_{u_1 \in A_1} \sup_{u_2 \in A_2} f(u_1, u_2),$$

where f is bounded from above ($f(u_1, u_2) \le M$), is continuous and has continuous partial derivatives on $A_1 \times A_2$. Further assume that for all $u_2 \in A_2$, there exists a unique $c_1(u_2) \in A_1$ such that

$$f(c_1(u_2), u_2) = \max\{f(u_1, u_2) : u_1 \in A_1\},$$

When you fix u2, there is a unique point (u1) that maximises f + vv

and similarly, for all $u_1 \in A_1$, there exists a unique $c_2(u_1) \in A_2$ such that

$$f(u_1, c_2(u_1)) = \max\{f(u_1, u_2) : u_2 \in A_2\}.$$

Max is a supremum but supremum isn't necessarily a max

The alternating maximisation algorithm The algorithm is iterative, and alternatively updates u_1 and u_2 as

$$u_1^0, u_2^0, u_1^1, u_2^1, u_1^2, u_2^2 \dots$$

Will approach and converge to overall supreme

as follows. First of all, let $u_1^0 \in A_1$ be arbitrary and $u_2^0 = c_2(u_1^0)$. Next, for all $k \ge 1$, let

$$u_1^k = c_1(u_2^{k-1})$$
$$u_2^k = c_2(u_1^k).$$

Fixes u2 and maximises u1 accordingly

We fix one and find the optimum of the other. Then fix the second and find the optimum of the other. and repeat with the same two points

Intuitively, suppose you want to climb a mountain all the way to the top. Let u_1 denote your longitude, while u_2 denotes your latitude. You can only move along the E-W axis (changing u_1) or along the N-S axis (changing u_2). First of all, you look at the longitude u_1^0 of your starting point, then get to the highest point along the N-S axis to get to the point (u_1^0, u_2^0). You then make a 90° turn and move as high you can along the E-W axis to get to (u_1^1, u_2^0). And you keep going that way, moving N-S, then E-W, and so on until you reach the top.

Convergence Recall that an infinite sequence $(f^k : k \in \mathbb{N})$ that is non-decreasing and bounded from above converges to its supremum:

$$\lim_{k \to \infty} f^k = \sup_{k \in \mathbb{N}} f^k.$$

For all $k \in \mathbb{N}$, let $f^k := f(u_1^k, u_2^k)$, then this sequence is bounded from above (by our assumption on the function f), and is non-decreasing since:

Sequence is bounded from above because f^k is by our assumption

$$\begin{aligned} f^k &= f(u_1^k, u_2^k) \\ &= f(u_1^k, c_2(u_1^k)) \\ &\ge f(u_1^k, u_2^{k-1}) \\ &= f(c_2(u_2^{k-1}), u_2^{k-1}) \\ &\ge f(u_1^{k-1}, u_2^{k-1}) \\ &= f^{k-1}. \end{aligned}$$

In general, however, the limit is not necessary the global maximum f^*. Indeed, imagine there are two mountains of different altitude near each other, and that you start climbing the shorter one. The algorithm will take you to the summit of the short mountain (a local optimum) but it cannot send you to the summit of the other mountain (the global optimum).

This happens if too close to local optimum and too far from global optimum

Problem happens if function tryping to optimise is not concave

Concavity/convexity Nonetheless, if the function f is concave, then it can be shown that the algorithm will converge towards the global optimum.

Since maximising f is equivalent to minimising $\tilde{f} = -f$, we see that the alternating minimisation algorithm will converge the global minimum for

i.e. previous part works vice versa

$$\inf_{u_1 \in A_1} \inf_{u_2 \in A_2} \tilde{f}(u_1, u_2),$$

provided that $\tilde{f}$ should be convex.

infimum - Greatest lower bound
Min is when infimum is reached

37.2 BA algorithm for capacity

Expressing the capacity as an alternating maximisation problem Consider the channel $(\mathcal{X}, p(y|x), \mathcal{Y})$. For the sake of clarity, let us different letters for the input, output, joint and conditional probability distributions, i.e.

$$\begin{aligned}
Y|X &\sim p(y|x), && \text{Channel = p} \\
X &\sim r(x), && \text{Source X = r i.e. encoding, so maximising r finds the best possible, most efficient encoding} \\
X|Y &\sim q(x|y), && \text{Decoder = q. looking to do the best decoding possible} \\
\text{Overall prob dist of output} \quad Y &\sim s(y), && s(y) = \sum_{x'} r(x')p(y|x') \\
\text{Joint dist = t} \quad (X,Y) &\sim t(x,y), && t(x,y) = q(x|y)s(y) = r(x)p(y|x).
\end{aligned}$$

Then the mutual information can be expressed as

Trying to max mutual info

$$\begin{aligned}
I(X;Y) &= \sum_{x,y} t(x,y) \log \frac{t(x,y)}{r(x)s(y)} && \text{by definition} \\
&= \sum_{x,y} r(x)p(y|x) \log \frac{q(x|y)}{r(x)}.
\end{aligned}$$

The capacity is then the maximum over all input probabilities $r(x)$ of the quantity above. The key idea now is that if we view $q(x|y)$ as a second variable, then the mutual information is maximised when

Look at r as u1 and q as u2

Mutual info is maximised when q is what it should be

$$q(x|y) = \frac{r(x)p(y|x)}{s(y)} = \frac{r(x)p(y|x)}{\sum_{x'} r(x')p(y|x')}.$$

Once you find r, maximising q is easy.
Other way round is harder.

Omitting some technical details, we obtain

Capacity

$$\begin{aligned}
C &= \sup_{r \in A_1} \sup_{q \in A_2} \sum_{x,y} r(x)p(y|x) \log \frac{q(x|y)}{r(x)}, \\
A_1 &= \left\{ (r(x), x \in \mathcal{X}) : r(x) > 0 \ \forall x \in \mathcal{X}, \sum_x r(x) = 1 \right\} \\
A_2 &= \left\{ (q(x|y), (x,y) \in \mathcal{X} \times \mathcal{Y}) : q(x|y) = 0 \iff p(y|x) = 0, \sum_x q(x|y) = 1 \ \forall y \in \mathcal{Y} \right\}.
\end{aligned}$$

Finding the optima We can now use the alternating maximisation algorithm for

Using that encoding, whats the best possible decoding we can do, then working from that whats the best possible encoding etc

$$f(r,q) = \sum_{x,y} r(x)p(y|x) \log \frac{q(x|y)}{r(x)}.$$

For a fixed $r(x)$, finding the $q(x|y)$ that maximises $f(r,q)$ is straightforward. However, for a fixed $q(x|y)$, finding the $r(x)$ that maximises $f(r,q)$ requires the machinery of Lagrange multipliers, which we shall leave out, but the result is

$$r(x) = \frac{\prod_y q(x|y)^{p(y|x)}}{\sum_{x'} \prod_y q(x'|y)^{p(y|x')}}.$$

The algorithm Thus, the BA algorithm for channel capacity runs as follows. Start with a strictly positive input distribution r^0. Compute $q^0, r^1, q^1, r^2, \ldots$ alternately by

No 0 probabilities for input dist

Max q for that r, then max r for that q ...

$$q^k(x|y) = \frac{r^k(x)p(y|x)}{\sum_{x'} r^k(x')p(y|x')},$$
$$r^k(x) = \frac{\prod_y q^{k-1}(x|y)^{p(y|x)}}{\sum_{x'} \prod_y q^{k-1}(x'|y)^{p(y|x')}}.$$

Then as $k \to \infty$, r^k tends to the capacity achieving input distribution and $f(r^k, q^k) \to C$.

k is number in sequence so as sequence continues, r goes towards capacity/input distribution and f tend to capacity

37.3 BA algorithm for the rate distortion function. Non-examinable

Subsection 37.3 is not examinable.

What should we optimise? Here we assume $R(0) > 0$ (otherwise the problem is trivial), and we consider a distortion measure with maximum value $D_{\max}$. It can then be shown that $R(D)$ is a strictly decreasing convex function of D for $0 \le D \le D_{\max}$.

We determine a quantity that is minimised by a point $(D, R(D))$ on the rate-distortion function curve. Let s be the slope of the tangent at a point on the curve, and to emphasize that s will be an important parameter, we denote that point by $(D_s, R(D_s))$ (so that $s = \frac{dR}{dD}(D_s)$). That tangent hits the vertical axis at the point $(0, R(D_s) - sD_s)$. By convexity, the tangent at the point $(D_s, R(D_s))$ remains below the curve, and in fact for any achievable point (D', R') with $R' \ge R(D')$, we have

$$R' - sD' \ge R(D_s) - sD_s.$$

Therefore, to find the point $(D_s, R(D_s))$, we need to solve the minimisation:

$$\min_{\hat{X}} (I(X;\hat{X}) - sD(X,\hat{X})).$$

Expressing the rate distortion function as an alternating minimisation problem Again, let us use different names for the distributions:

$$\begin{aligned} X &\sim p(x) \\ \hat{X}|X &\sim Q(\hat{x}|x) \\ \hat{X} &\sim t(\hat{x}) \qquad & t(\hat{x}) = \sum_x p(x)Q(\hat{x}|x) \\ (X,\hat{X}) &\sim r(x,\hat{x}) \qquad & r(x,\hat{x}) = p(x)Q(\hat{x}|x). \end{aligned}$$

This time we need to express the mutual information and the distortion:

$$I(X;\hat{X}) = \sum_{x,\hat{x}} p(x)Q(\hat{x}|x) \log \frac{Q(\hat{x}|x)}{t(\hat{x})}$$
$$D(X,\hat{X}) = \sum_{x,\hat{x}} p(x)Q(\hat{x}|x)d(x,\hat{x}).$$

Our two parameters will be $Q(\hat{x}|x)$ and $t(\hat{x})$. Omitting technical details, we obtain

$$R(D_s) - sD_s = \inf_{Q \in A_1} \inf_{t \in A_2} \left\{ \sum_{x,\hat{x}} p(x)Q(\hat{x}|x) \log \frac{Q(\hat{x}|x)}{t(\hat{x})} - s \sum_{x,\hat{x}} p(x)Q(\hat{x}|x)d(x,\hat{x}) \right\},$$
$$A_1 = \left\{ (Q(\hat{x}|x), (x,\hat{x}) \in \mathcal{X} \times \hat{\mathcal{X}}) : Q(\hat{x}|x) > 0, \sum_{\hat{x}} Q(\hat{x}|x) = 1 \ \forall x \in \mathcal{X} \right\}$$
$$A_2 = \left\{ (t(\hat{x}), \hat{x} \in \hat{\mathcal{X}}) : t(\hat{x}) > 0 \ \forall \hat{x} \in \hat{\mathcal{X}}, \sum_{\hat{x}} t(\hat{x}) = 1 \right\}.$$

Finding the optima Again, when we fix $Q(\hat{x}|x)$, the optimal value of $t(\hat{x})$ is what we expect:

$$t(\hat{x}) = \sum_x p(x)Q(\hat{x}|x).$$

Lagrange multipliers give the converse optimum, for a given $t(\hat{x})$ it is given by

$$Q(\hat{x}|x) = \frac{t(\hat{x})e^{sd(x,\hat{x})}}{\sum_{\hat{x}'} t(\hat{x}')e^{sd(x,\hat{x}')}}$$

The algorithm Start with any strictly positive transition matrix Q^0. Compute $t^0, Q^1, t^1, Q^2, \ldots$ alternatively by

$$t^k = \sum_x p(x)Q^k(\hat{x}|x)$$
$$Q^k(\hat{x}|x) = \frac{t^{k-1}(\hat{x})e^{sd(x,\hat{x})}}{\sum_{\hat{x}'} t^{k-1}(\hat{x}')e^{sd(x,\hat{x}')}}$$

Then as $k \to \infty$, $D(p, Q^k)$ tends to D_s while $I(X;\hat{X}) = I(p, Q^k, t^k)$ tends to $R(D_s)$.

37.4 See further

Generalisations of Blahut-Arimoto The BA algorithms are special cases of a general iterative algorithm by Csiszár and Tusnády, which also includes the Expectation-Maximization (EM) algorithm for fitting models from incomplete data.

37.5 Exercises

Exercise 37.1. Prove that the function f used in the BA algorithm for capacity is concave.

Exercise 37.2. Prove that if g is convex and twice differentiable, then for any x, the tangent at x of the curve $(x, g(x))$ remains below the curve.

Exercise 37.3. Implement both BA algorithms. Verify that they converge towards the right value for the capacity of the BSC channel and for the rate-distortion function of a Bernoulli variable, respectively.

Exercise 37.4. In the BA algorithm for capacity, what would be the main problem in starting with r^0 which is not positive, i.e. $r^0(x) = 0$ for some $x \in \mathcal{X}$?

38 Capacity and rate distortion II: Gaussian variables

38.1 Continuous random variables

Continuous random variables A continuous random variable X is a mapping p from $\mathbb{R}$ to $\mathbb{R}^+$ such that

$$\int_{-\infty}^{\infty} p(x)dx = 1.$$

Any element $x \in \mathcal{X}$ is called an event; $p(x)$ is called the probability density function of X. We also use the interpretation

$$\mathbb{P}(X \leq x) = \int_{-\infty}^{x} p(t)dt.$$

Again, we can create functions of the random variable X. We denote the support of X as

$$S := \{x \in \mathbb{R} : p(x) > 0\}.$$

Expectation and standard deviation The expectation of a continuous random variable X is

$$\mathbb{E}(X) := \int_S x p(x) dx.$$

The variance of X is

$$\begin{aligned} \mathrm{Var}X &:= \mathbb{E}((X - \mathbb{E}(X))^2) \\ &= \mathbb{E}(X^2) - \mathbb{E}(X)^2. \end{aligned}$$

The standard deviation of X is

$$\sigma = \sqrt{\mathrm{Var}X}.$$

Simple examples The uniform distribution $U(a,b)$ over an interval $[a,b]$ $(b > a)$ is defined as

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \le x \le b \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to show that

$$\begin{aligned} \mathbb{E}(U(a,b)) &= \frac{1}{2}(a+b), \\ \mathrm{Var}U(a,b) &= \frac{1}{12}(b-a)^2. \end{aligned}$$

The most important continuous random variable is the normal (or Gaussian) distribution $\mathcal{N}(\mu, \sigma^2)$, defined as

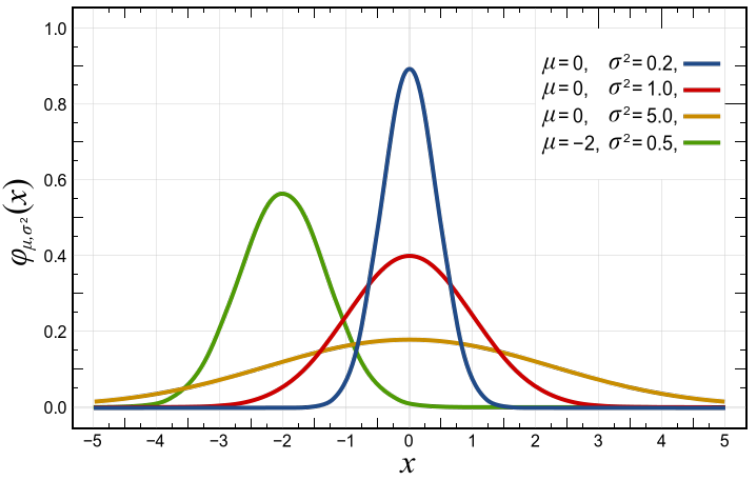
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

FOr all x in R (real)

It can be proved that

take for granted - integral of everything = 1

$$\begin{aligned} \mathbb{E}(\mathcal{N}(\mu, \sigma^2)) &= \mu, \\ \mathrm{Var}\mathcal{N}(\mu, \sigma^2) &= \sigma^2. \end{aligned}$$



38.2 Differential entropy

lower case h for differential(continuous)

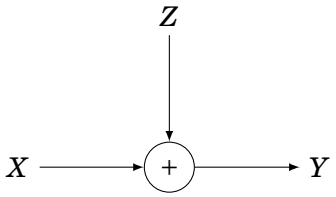
Differential entropy The differential entropy $h(X)$ of a continuous random variable X with density $f(x)$ is defined as

$$h(X) = -\int_S f(x) \log f(x) dx,$$

replace sum with integral

where S is the support of the random variable. We remark that $h(X)$ could be negative, or simply undefined!

it can be negative and undefined like normal discrete entropy



Differential entropy of a uniform distribution

Example 38.1. *Let X have density*

$$f(x) = \begin{cases} 1/a & \text{if } x \in [0,a] \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$h(X) = \log a.$$

See img 1

This example shows that the differential entropy does take values into account!

While the uniform distribution maximises the entropy in the discrete case, Gaussian variables maximise the differential entropy in the continuous case: for any Y with standard deviation σ, we have

$$h(Y) \le h(\mathcal{N}(0,\sigma^2)) = \frac{1}{2}\log 2\pi e\sigma^2 \text{ bits.}$$

(Exercise 38.2 asks for the proof.)

Related concepts Joint entropy, conditional entropy are defined similarly. So is the divergence

$$D(f||g) := \int f \log \frac{f}{g}$$

and the mutual information

$$\begin{aligned} I(X;Y) &= D(f(x,y)||f(x)f(y)) \\ &= \int f(x,y) \log \frac{f(x,y)}{f(x)f(y)} dxdy. \end{aligned}$$

We still have $D(f||g) \ge 0$ with equality iff $f = g$ almost everywhere.

A quick note on integration by parts You have seen integration by parts on closed intervals $[a,b]$:

$$\int_a^b u'(x)v(x)dx = [u(x)v(x)]_a^b - \int_a^b u(x)v'(x)dx.$$

Normally can only do if between a and b but assume can always use by parts

We will have to use integration by parts over (half)-open intervals, such as $(-\infty,\infty)$ or $[0,\infty)$. This is not allowed in general, but we will never meet any bad example, so you can safely assume that integration by parts is always allowed.

38.3 The Gaussian channel and its capacity

The Gaussian channel This is the most important continuous alphabet channel.

The noise Z is drawn from a Gaussian distribution with variance N. Thus

$$Y = X + Z, \quad Z \sim \mathcal{N}(0,N).$$

Mean 0 variance N

It is also called Additive White Gaussian Noise (AWGN) channel.

Capacity: naive approach Clearly, if the noise variance is zero, then the receiver receives the transmitted symbol perfectly. Since X can take any real value, the channel can transmit an arbitrary real number with no error.

If the noise variance is non-zero and there is no constraint on the input, we can choose an infinite subset of inputs arbitrarily far apart, so that they are distinguishable at the output with arbitrarily small probability of error. Again, this has infinite capacity. img2

Power constraint The most common limitation on the input is a power constraint. We assume an average power constraint: for any codeword x^n transmitted over the channel, we require

$$\frac{1}{n}\sum_{i=1}^{n} x_i^2 \le P.$$

This channel models many practical situations, such as radio and satellite links. (For wireless channels, we use so-called Rayleigh fading with Gaussian noise.)

The capacity of the Gaussian channel with power constraint P is defined as

$$C := \max\{I(X;Y) : \mathbb{E}(X^2) \le P\}.$$

Theorem 38.2. *The capacity of a Gaussian channel with power constraint P and noise variance N is*

$$C = \frac{1}{2}\log\left(1+\frac{P}{N}\right).$$

Non-examinable proof. The proof is actually similar to that of the BSC. We have

$$\begin{aligned} I(X;Y) &= h(Y) - h(Y|X) \\ &= h(Y) - h(Z). \end{aligned}$$

We have $h(Z) = \frac{1}{2}\log 2\pi eN$. Also,

$$\mathbb{E}(Y^2) = P + N,$$

whence

$$\begin{aligned} h(Y) &\le \frac{1}{2}\log 2\pi e(P+N), \\ I(X;Y) &\le \frac{1}{2}\log\left(1+\frac{P}{N}\right). \end{aligned}$$

The maximum is achieved when $X \sim \mathcal{N}(0,P)$.

□

Channel coding theorem for the Gaussian channel The capacity is still the supremum of the achievable rates for the Gaussian channel.

Definition 38.3. *(M,n) code for the Gaussian channel with power constraint P:*

1. *An index set $\{1,\dots,M\}$.*
2. *An encoding function $x^n : \{1,\dots,M\} \to \mathcal{X}^n$ satisfying the power constraint P, i.e. for every codeword*
$$\frac{1}{n}\sum_{i=1}^{n} x_i^2(w) \le P.$$
3. *A decoding function $g : \mathcal{Y}^n \to \{1,\dots,M\}$.*

The rate and the probability of error are defined as before. A rate is achievable if the maximum probability of error tends to zero.

Theorem 38.4. *The capacity of the Gaussian channel with power constraint P is the supremum of all achievable rates.*

38.4 Rate distortion for a Gaussian source

Let us use a Gaussian source with squared error distortion. Therefore, the distortion criterion is

$$\mathbb{E}(X-\hat{X})^2 \le D.$$

Theorem 38.5. *For an $\mathcal{N}(0,\sigma^2)$ source with squared error distortion,*

$$R(D) = \begin{cases} \frac{1}{2}\log\frac{\sigma^2}{D}, & 0 \le D \le \sigma^2 \\ 0, & D > \sigma^2. \end{cases}$$

Exercise 38.5 asks for the proof.

38.5 Exercises

Exercise 38.1. Verify the expectation and the variance of the uniform distribution $U(a,b)$.

Exercise 38.2. In this exercise, we compute the entropy of Gaussian variables and show that these variables do maximise the entropy.

Let $X \sim \mathcal{N}(0,\sigma^2)$.

1. Prove that
$$h(X) = \frac{1}{2}\log 2\pi e\sigma^2 \quad \text{bits.}$$
2. Prove that for any Y with standard deviation σ, $h(Y) \le h(X)$ for $X \sim \mathcal{N}(0,\sigma^2)$.

Hint: You may find it easier to compute the entropy in nats:

$$h_e(X) = -\int \phi \ln \phi \quad \text{nats,}$$

and then $h(X) = \frac{1}{\ln 2} h_e(X)$.

Note: Since $h(Z+\mu) = h(Z)$ for any random variable Z and any $\mu \in \mathbb{R}$, any normal distribution $\mathcal{N}(\mu,\sigma^2)$ will have the same differential entropy as above.

Exercise 38.3. Consider k parallel Gaussian channels

$$Y_j = X_j + Z_j, \quad j = 1,2,\dots,k$$

where $Z_j \sim \mathcal{N}(0,N_j)$, and the noise is assumed to be independent from channel to channel. Consider a common power constraint on the total power of the inputs, i.e.

$$\mathbb{E}\left(\sum_{j=1}^{k} X_j^2\right) \le P.$$

In order to simplify notation, we denote $X = (X_1,\dots,X_k)$ and $Y = (Y_1,\dots,Y_k)$.

Show that

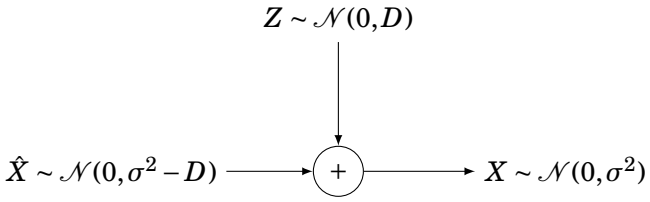
$$I(X;Y) \le \sum_{j=1}^{k} \frac{1}{2}\log\left(1+\frac{P_j}{N_j}\right),$$

where $P_j = \mathbb{E}(X_j^2)$ and $\sum_j P_j = P$. Give an input distribution which reaches that value.

Exercise 38.4.

1. Let Y_1 and Y_2 be conditionally independent given X. Prove that
$$I(X;Y_1,Y_2) = I(X;Y_1) + I(X;Y_2) - I(Y_1;Y_2).$$
2. Let K be a Gaussian channel, with input X and output $Y = X+Z$ and $Z \sim \mathcal{N}(0,N)$. Consider the "two-step" Gaussian channel K', where the input X is sent through K twice, hence the output is $(Y_1,Y_2) = (X+Z_1, X+Z_2)$ and Z_1, Z_2 are i.i.d. $\sim \mathcal{N}(0,N)$. Based on the previous item, show that the capacity of the "two-step" Gaussian channel is no more than twice than the capacity of the original Gaussian channel.

Exercise 38.5. Prove Theorem 38.5. Hint: the proof is similar to the Hamming rate distortion function of a Bernoulli variable. For instance, for the second part of the proof you want to consider this channel:



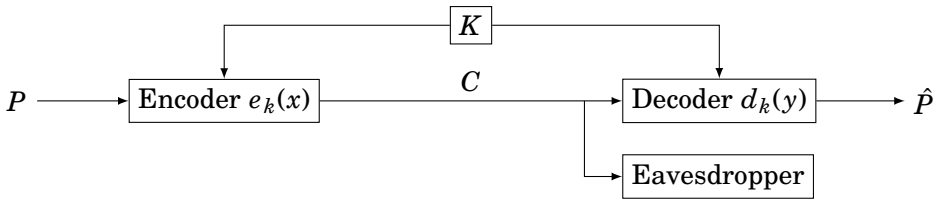
39 Secrecy I: Cryptosystems

39.1 Cryptosystems and perfect secrecy

Definition A cryptosystem is a five-tuple $(\mathcal{P},\mathcal{C},\mathcal{K},\mathcal{E},\mathcal{D})$, where

1. $\mathcal{P}$ is a finite set of possible plaintexts x.
2. $\mathcal{C}$ is a finite set of possible ciphertexts y.
3. $\mathcal{K}$, the keyspace, is a finite set of possible keys k.
4. $\mathcal{E}$ is a set of encryption rules $e_k : \mathcal{P} \to \mathcal{C}$ for every $k \in \mathcal{K}$.
5. $\mathcal{D}$ is a set of decryption rules $d_k : \mathcal{C} \to \mathcal{P}$ for every $k \in \mathcal{K}$.
6. For each $k \in \mathcal{K}$, we have
$$d_k(e_k(x)) = x$$
for every plaintext $x \in \mathcal{P}$.

Note: since e_k is injective, we have $|\mathcal{C}| \geq |\mathcal{P}|$. Also, we have $|\mathcal{E}| = |\mathcal{D}| = |\mathcal{K}|$.



Perfect secrecy Assumptions:

1. The key is chosen (by Alice and Bob) using some fixed probability distribution K.
2. The plaintext is chosen at random according to a random variable P.
3. Since the key is chosen before Alice knows what the plaintext will be, K and P are independent random variables.
4. The ciphertext is also a random variable C.

Perfect secrecy We say that a cryptosystem has perfect secrecy if one of the following equivalent properties is satisfied.

1. $p(x|y) = p(x)$ for all $x \in \mathcal{P}$, $y \in \mathcal{C}$.
2. P and C are independent.
3. $I(P;C) = 0$.

In other words, the ciphertext gives nothing away about the plaintext.

The shift cipher has perfect secrecy

Theorem 39.1. *Suppose the shift cipher is used, where each key is used with equal probability (i.e. K is uniform). Then for any plaintext probability P, the shift cipher has perfect secrecy.*

Proof. We have

$$I(P;C) = H(C) - H(C|P) = H(C) - H(K) \leq 0.$$

Thus $I(P;C) = 0$ and we have perfect secrecy. □

As an aside, we have also proved that $H(C) = H(K) = \log q$ in that case: the ciphertexts are also equiprobable.

39.2 Spurious keys and unicity distance

Key equivocation The conditional entropy $H(K|C)$ is called the key equivocation: that's how much uncertainty is left about the key when the ciphertext is known.

Theorem 39.2. *For any cryptosystem, the key equivocation is given by*

$$H(K|C) = H(K) + H(P) - H(C).$$

Proof. We have

$$H(K,P,C) = H(C|P,K) + H(K,P) = H(K) + H(P),$$

since the encoding is deterministic ($H(C|K,P) = 0$) and the key and the plaintext are independent ($I(K;P) = 0$).

Similarly, we also have

$$H(K,P,C) = H(P|C,K) + H(K,C) = H(C) + H(K|C).$$

Combining, we obtain the result. □

Spurious keys Suppose a string of plaintext $x^n = x_1x_2\dots x_n$ is encrypted using one key, producing $y^n = y_1y_2\dots y_n$.

Eve wants to determine the key. We are looking at ciphertext-only attacks, and we assume that Eve has infinite computational power. We also assume that Eve knows the plaintext is in a "natural" language, such as English. In general, Eve will be able to rule out certain keys, but many "possible" keys remain, only one of which is the correct key. The remaining possible, but incorrect, keys are the spurious keys.

For example, say the shift cipher is used and Eve receives `WNAJW`, then we can check that there are only two meaningful plaintexts, namely `RIVER` and `ARENA`, for encryption keys `F` and `W`. One of these is correct, the other one is spurious.

Language redundancy Let P^n be the random distribution on all n-grams of plaintext. The entropy (per letter) of a natural language L is

$$H_L = \lim_{n\to\infty} \frac{H(P^n)}{n}.$$

For instance, for the English language, we have (roughly)

$$H(P) \approx 4.219, \quad H(P^2) \approx 2 \times 3.90, \quad \dots \quad H_L \leq 1.5.$$

The redundancy of the language L is defined as

$$R_L = 1 - \frac{H_L}{\log|\mathscr{P}|}.$$

Bound on the number of spurious keys

Theorem 39.3. *Consider a cryptosystem with $|\mathscr{C}| = |\mathscr{P}|$ and uniform key. Then given a string of ciphertext of length n (n large enough), the expected number of spurious keys s_n satisfies*

$$s_n \geq \frac{|\mathscr{K}|}{|\mathscr{P}|^{nR_L}} - 1.$$

Note: the RHS tends to zero exponentially quickly as $n \to \infty$.

Proof. The key equivocation is

$$H(K|C^n) = H(K) + H(P^n) - H(C^n),$$

where

$$\begin{aligned} H(K) &= \log|\mathscr{K}|, \\ H(P^n) &\geq nH_L = n(1-R_L)\log|\mathscr{P}|, \\ H(C^n) &\leq n\log|\mathscr{C}| = n\log|\mathscr{P}|. \end{aligned}$$

Thus,

$$H(K|C^n) \geq \log|\mathscr{K}| - nR_L\log|\mathscr{P}|.$$

Given $y^n \in \mathscr{C}^n$, let $K(y^n)$ denote the set of possible keys that were used to encrypt a message into y^n:

$$K(y^n) = \{K \in \mathscr{K} : \exists x^n \in \mathscr{P}^n, p(x^n) > 0, e_K(x^n) = y^n\},$$

then $s_n + 1 = \sum_{y^n} p(y^n)|K(y^n)|$.

We can relate it to the key evocation as follows.

$$\begin{aligned} H(K|C^n) &= \sum_{y^n \in \mathscr{C}^n} p(y^n)H(K|y^n) \\ &\leq \sum_{y^n \in \mathscr{C}^n} p(y^n)\log|K(y^n)| \\ &\leq \log \sum_{y^n \in \mathscr{C}^n} p(y^n)|K(y^n)| \\ &= \log(s_n + 1). \end{aligned}$$

Combining, we obtain the result.

□

Unicity distance The unicity distance of a cryptosystem is defined to be the value of n at which the expected number of spurious keys becomes zero. In other words, it is the average amount of ciphertext required for an opponent to be able to uniquely compute the key (given enough computing time).

We have

$$s_n \geq \frac{|\mathscr{K}|}{|\mathscr{P}|^{nR_L}} - 1.$$

If we view the RHS as an actual estimate for s_n and solve for n, we get that the unicity distance is about

$$\frac{\log|\mathscr{K}|}{R_L\log|\mathscr{P}|}.$$

Some people then use this estimate as their *de facto* definition.

39.3 Exercises

Exercise 39.1. let n be a positive integer. A Latin square of order n is an $n \times n$ array L of the integers $[n] = \{1, \ldots, n\}$ such that every one of the n integers occurs exactly once in every row and every column of L. An example of a Latin square of order 4 is given as follows.

$$L = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}.$$

1. For any n, give a construction of a Latin square of order n. (Actually, there are incredibly many of those!)

2. Given a Latin square L of order n, define a cryptosystem with $\mathscr{K} = \mathscr{P} = \mathscr{C} = [n]$ which has perfect secrecy if $\mathscr{K}$ is uniform.

Exercise 39.2. Suppose a cryptosystem achieves perfect secrecy for a particular plaintext probability distribution. Prove that perfect secrecy is maintained for any plaintext distribution.

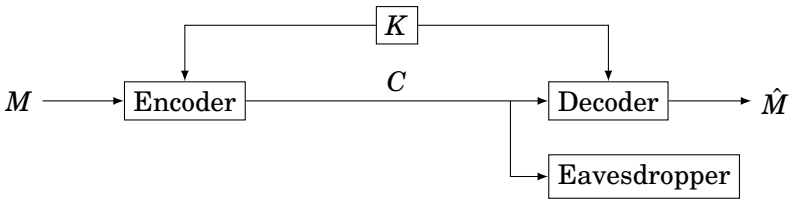
Exercise 39.3. Prove that if a cryptosystem has perfect secrecy and $|\mathscr{K}| = |\mathscr{C}| = |\mathscr{P}|$, then every ciperext is equiprobable.

40 Secrecy II: Secrecy capacity

40.1 Perfect secrecy

Perfect secrecy revisited A sender (Alice) wishes to communicate a message $M \in [2^{nR}]$ to a receiver (Bob) over a public channel in the presence of an eavesdropper (Eve) who observes the channel output. Alice and Bob share a key $K \in \mathbb{N}$, which is unknown to Eve.

How many key bits ($H(K)$) are needed so that Eve cannot obtain any information about the message?



Perfect secrecy The message (plaintext) is uniform: $M \sim U(2^{nR})$. The encoder (encryption function) assigns a ciphertext $c(m,k)$ to each message $m \in [2^{nR}]$ and key k, and the decoder (decryption function) assigns a message $\hat{m}(c,k)$ to each ciphertext c and key k. The communication system is said to have perfect secrecy if

- $\mathbb{P}(M \neq \hat{M}(C(M,K),K)) = 0$, i.e. the message is decoded correctly, and
- the information leakage $I(C;M) = 0$, i.e. the ciphertext reveals no information about the message.

Perfect secrecy theorem The necessary and sufficient condition to attain perfect secrecy is that the key entropy should be no less than the plaintext entropy. Compare it with Theorem 11.1.

Theorem 40.1 (Shannon's perfect secrecy theorem)**.** *To achieve perfect secrecy, it is necessary to have*

$$H(K) \geq H(M).$$

Proof. For perfect secrecy, we must have

$$\begin{aligned} H(M) &= H(M|C) \\ &\le H(M,K|C) \\ &= H(K|C) \\ &\le H(K). \end{aligned}$$

□

Conversely, for any $h_k \ge h_m \ge 0$, there exists a system with perfect secrecy with $H(K) = h_k$ and $H(M) = h_m$. Indeed, let the key K be uniformly distributed over $[2^{nR}]$ and take the bitwise XOR of the binary expansions of M and K (the one-time pad!). It can be readily checked that M and C are independent.

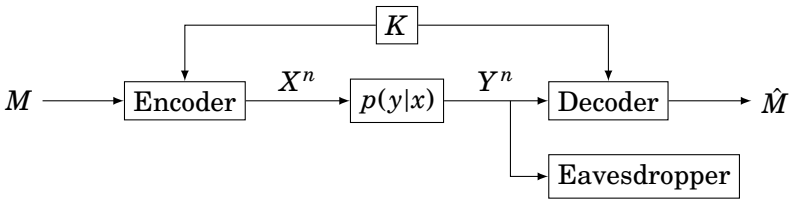
Consequences of perfect secrecy So this is bad news! Alice and Bob need to share a key as long as the message itself. There are several directions to avoid this limitation of Shannon's secrecy system

- Computational security: how to exploit the computational hardness of recovering the message from the ciphertext without knowing the key?
- Wiretap channel: how to use the better quality of Alice-Bob channel to outdo Eve

We shall take a look at the wiretap channel.

40.2 Secrecy capacity

Secure Communication over a DMC Consider the following extension of the Shannon secrecy system to DMC.



Secrecy codes Formally, a $(2^{nR}, 2^{nR_K}, n)$ secrecy code for the system consists of:

1. A message set $[2^{nR}]$ and a key set $[2^{nR_K}]$
2. A randomized encoder that generates codeword $X^n(m,k) \in \mathcal{X}^n$ according to the conditional pmf $p(x^n|m,k)$ for each message-key pair $(m,k) \in [2^{nR}] \times [2^{nR_K}]$
3. A decoder that assigns a message $\hat{m}(y^n,k)$ or an error message e to each received sequence $y^n \in \mathcal{Y}^n$ and key $k \in [2^{nR_K}]$

Information leakage We assume that the message-key pair (M,K) is uniformly distributed over $[2^{nR}] \times [2^{nR_K}]$. The information leakage rate associated with the $(2^{nR}, 2^{nR_K}, n)$ secrecy code is defined as

$$R_l^{(n)} = \frac{1}{n} I(M;Y^n).$$

The average probability of error for the secrecy code is defined as

$$P_e^{(n)} = \mathbb{P}(M \neq \hat{M}(Y^n,K)).$$

A rate R is achievable at key rate R_K if there exists a sequence of $(2^{nR}, 2^{nR_K}, n)$ codes with $R_l^{(n)} \to 0$ and $P_e^{(n)} \to 0$ as $n \to \infty$

Secrecy capacity The secrecy capacity $C(R_K)$ of the DMC as the supremum of all achievable rates at key rate R_K.

This asymptotic notion of secrecy is weaker than Shannon's original notion of perfect secrecy. Can we send at a higher rate?

Theorem 40.2. *The secrecy capacity is*

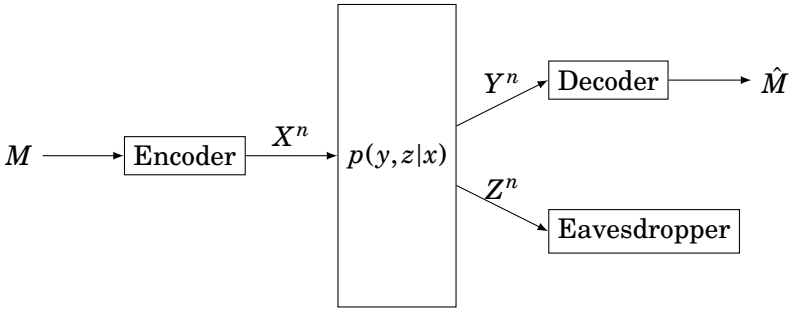
$$C(R_K) = \min\left\{R_K, \max_{p(x)} I(X;Y)\right\}.$$

Thus secure communication is still limited by the key rate until saturated by the channel capacity.

40.3 The wiretap channel

The Wiretap Channel A discrete-memoryless wiretap channel (DM-WTC) $(\mathcal{X}, p(y,z|x), \mathcal{Y} \times \mathcal{Z})$ is a DM-Broadcast Channel with sender X, legitimate receiver Y and eavesdropper Z.

The sender X wishes to send a message $M \in [2^{nR}]$ reliably to the receiver Y while keeping it secret from the eavesdropper Z.



Codes for the wiretap channel A $(2^{nR}, n)$ secrecy code for the DM-WTC consists of:

1. A message set $[2^{nR}]$
2. A randomized encoder that generates a codeword $X^n(m)$, $m \in [2^{nR}]$, according to $p(x^n|m)$
3. A decoder that assigns a message $\hat{m}(y^n)$ or an error message e to each received sequence $y^n \in \mathcal{Y}^n$

The message is drawn uniformly at random: $M \sim U[2^{nR}]$. The information leakage rate is defined as

$$R_l^{(n)} := \frac{1}{n} I(M;Z^n)$$

Secrecy capacity of the wiretap channel The probability of error, achievability, and the secrecy capacity C_S are defined as for the problem of secure communication over DMC.

Theorem 40.3. *The secrecy capacity of the DM-WTC $(\mathcal{X}, p(y,z|x), \mathcal{Y} \times \mathcal{Z})$ is given by*

$$C_S = \max_{p(u,x)} (I(U;Y) - I(U;Z))$$

for $|\mathcal{U}| \leq |\mathcal{X}|$.

Physically degraded wiretap channel The wiretap channel was first introduced by Wyner, who assumed that the channel to the eavesdropper is a physically degraded version of the channel to the receiver, i.e. $p(y,z|x) = p(y|x)p(z|y)$. Under this assumption,

$$\begin{aligned} I(U;Y) - I(U;Z) &= I(U;Y|Z) \\ &\leq I(X;Y|Z) \\ &= I(X;Y) - I(X;Z) \end{aligned}$$

Hence, the secrecy capacity in this case is

$$C_S = \max_{p(x)} (I(X;Y) - I(X;Z)).$$

Binary Symmetric Wiretap Channel Consider the Binary Symmetric Wiretap Channel BS-WTC:

$$Y = X \oplus Z_1,$$
$$Z = X \oplus Z_2,$$

where Z_1 and Z_2 are independent $\mathrm{B}(p_1)$ and $\mathrm{B}(p_2)$ variables, respectively. Recall that Bob receives Y, while Eve receives Z. This is a physically degrade channel.

We can already see three special cases of the secrecy capacity.

1. If $p_2 < p_1$, then $C_S = 0$. Indeed, Eve receives a message of higher quality than Bob's, and hence there is no hope of communicating anything securely to Bob.

2. If $p_2 = 1/2$, then $C_S = 1 - H(p_1)$. Indeed, Eve's channel is totally unreliable: $I(Z;X) = 0$. So the only limitation is that given by the BSC towards Bob, with capacity $1 - H(p_1)$.

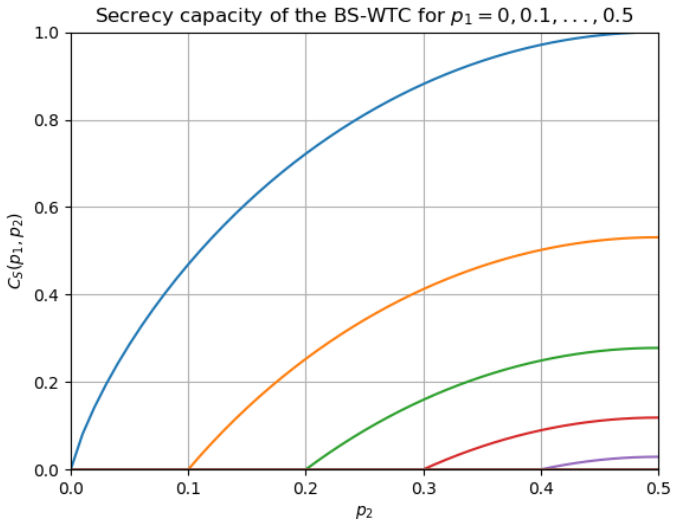
3. If $p_1 = 0$, then $C_S = H(p_2)$. Indeed, Bob's channel is perfect, so the only limitation is given by the BSC towards Eve. We have $I(Z;X) = H(X) - H(p_2)$, therefore we can transmit at a rate of $H(X) = H(p_2)$.

In general, the intuition is as follows. The BSC towards Bob "removes" $H(p_1)$ of information about the message, while the BSC towards Eve "removes" $H(p_2)$ of information. Therefore, if we make sure to send zero information towards Eve, we can only send with rate $H(p_2) - H(p_1)$ towards Bob and of 0 towards Eve.

Theorem 40.4. *The secrecy capacity for this channel is*

$$C_S = \begin{cases} H(p_2) - H(p_1) & \text{for } p_2 \geq p_1, \\ 0 & \text{otherwise.} \end{cases}$$

Part of the proof is asked from you in Exercise 40.1.



40.4 Exercises

Exercise 40.1. Prove that the secrecy capacity of the BS-WTC is $C_S = H(p_2) - H(p_1)$ for $p_2 \geq p_1$.

Exercise 40.2. The Gaussian wiretap channel is defined as the analogue of the BS-WTC for Gaussian noise. What do you think the secrecy capacity of that channel is? (Hard: prove the upper bound again!)

Appendix A

Additional material for Information Theory

41 The channel coding theorem: sketch of proof

Here we give a summary of the proof of the channel coding theorem.

Theorem 41.1 (The channel coding theorem)**.** *All rates below capacity C are achievable. Specifically, for every rate $R < C$, there exists a sequence of $(2^{nR}, n)$ codes with maximum probability of error $\lambda^{(n)} \to 0$.*
Conversely, any sequence of $(2^{nR}, n)$ codes with $\lambda^{(n)} \to 0$ must have $R \leq C$.

41.1 The Asymptotic Equipartition Property (AEP)

The Asymptotic Equipartition Property Analogue of the law of large numbers for information theory.

Context: an information source with probability distribution $p(x)$ keeps producing random variables $X_1, \ldots, X_n$ i.i.d with distribution $p(x)$.

Claim: For n large, almost all sequences $X_1, \ldots, X_n$ are very similar, and their behaviour is dictated by the entropy.

Theorem 41.2 (AEP)**.** *If $X_1, X_2, \ldots$ are i.i.d $\sim p(x)$, then*

$$-\frac{1}{n}\log p(X_1, \ldots, X_n) \to H(X) \quad \textit{in probability.}$$

Proof. Since the X_i are i.i.d, so are $\log p(X_i)$. The weak law of large numbers gives

$$\begin{aligned} -\frac{1}{n}\log p(X_1, \ldots, X_n) &= -\frac{1}{n}\sum_i \log p(X_i) \\ &\to -\mathbb{E}(\log p(X)) \quad \text{in probability} \\ &= H(X). \end{aligned}$$

□

Typical sets

Definition 41.3. *The typical set $A_\epsilon^{(n)}$ w.r.t. $p(x)$ is the set of sequences $x^n = (x_1, \ldots, x_n) \in \mathcal{X}^n$ such that*

$$H(X) - \epsilon \leq -\frac{1}{n}\log p(x^n) \leq H(X) + \epsilon.$$

Theorem 41.4 (The AEP for Typical Sequences)**.** *1. $\mathbb{P}(A_\epsilon^{(n)}) \geq 1 - \epsilon$ for n large enough.*

2. $|A_\epsilon^{(n)}| \le 2^{n(H(X)+\epsilon)}$.
3. $|A_\epsilon^{(n)}| \ge (1-\epsilon)2^{n(H(X)-\epsilon)}$ *for n large enough.*

The typical set has probability nearly 1, all elements of the typical set are nearly equiprobable, and the number of elements in the typical set is nearly $2^{nH(X)}$.

Example of the AEP Consider n coin flips $X_1,\dots,X_n$, where heads (1) has probability p and tails (0) probability $1-p$. We expect to see around np heads on average, and all runs with roughly np heads are nearly equiprobable. For any sequence $x^n \in \{0,1\}^n$ with w heads,

$$p(x^n) = p^w(1-p)^{n-w}.$$

The AEP indeed yields

$$-\frac{1}{n}\log p(X_1,\dots,X_n) \to H(p)$$

and more precisely, for typical sets and n large enough:

$$\begin{aligned} A_\epsilon^{(n)} &= \{x^n : |w-np| \le \epsilon n/|\log p - \log(1-p)|\} \\ \mathbb{P}(A_\epsilon^{(n)}) &\ge 1-\epsilon \\ (1-\epsilon)2^{n(H(p)-\epsilon)} \le |A_\epsilon^{(n)}| &\le 2^{n(H(p)+\epsilon)}. \end{aligned}$$

AEP and data compression As an illustration, we show how the AEP yields nearly optimal data compression. Let $X_1,\dots,X_n$ be i.i.d. We want a short description of this sequence.

Theorem 41.5. *Let X^n be i.i.d $\sim p(x)$ and $\epsilon > 0$. Then there exists a code which maps sequences x^n of length n into binary strings of length $l(x^n)$ such that the encoding is one-to-one and*

$$\mathbb{E}\left(\frac{1}{n}l(X^n)\right) \le H(X)+\epsilon.$$

Idea of proof:

1. Split $\mathcal{X}^n$ into typical sequences (in $A_\epsilon^{(n)}$) and atypical ones. Every codeword will start with either 1 (typical sequence) or 0 (atypical sequence).
2. In $A_\epsilon^{(n)}$, encode using at most $\log|A_\epsilon^{(n)}|+1 \le n(H(X)+\epsilon)+1$ bits.
3. Use brute force enumeration of atypical ones (unimportant: they are unlikely to occur).

The joint AEP

Definition 41.6. $A_\epsilon^{(n)}$*: set of jointly typical sequences* (x^n,y^n) *w.r.t.* $p(x,y)$*,*

$$\begin{aligned} A_\epsilon^{(n)} := \Big\{ &(x^n,y^n) \in \mathcal{X}^n \times \mathcal{Y}^n : \\ &\left|-\frac{1}{n}\log p(x^n) - H(X)\right| < \epsilon, \\ &\left|-\frac{1}{n}\log p(y^n) - H(Y)\right| < \epsilon, \\ &\left|-\frac{1}{n}\log p(x^n,y^n) - H(X,Y)\right| < \epsilon \Big\}, \end{aligned}$$

where

$$p(x^n,y^n) = \prod_{i=1}^n p(x_i,y_i).$$

Theorem 41.7 (Joint AEP). *Let (X^n, Y^n) be n-sequences drawn i.i.d. according to $p(x^n, y^n)$. Then*

1. $\Pr\{(X^n, Y^n) \in A_\epsilon^{(n)}\} \to 1$ *as* $n \to \infty$.
2. $|A_\epsilon^{(n)}| \leq 2^{nH(X,Y)+\epsilon}$.
3. *If* $(\tilde{X}^n, \tilde{Y}^n) \sim p(x^n)p(y^n)$, *then*

$$\Pr\{(\tilde{X}^n, \tilde{Y}^n) \in A_\epsilon^{(n)}\} \leq 2^{-n(I(X;Y)-3\epsilon)}.$$

Also, for n large enough,

$$\Pr\{(\tilde{X}^n, \tilde{Y}^n) \in A_\epsilon^{(n)}\} \geq (1-\epsilon)2^{-n(I(X;Y)+3\epsilon)}.$$

41.2 Achievability

Intuition For large block lengths, any channel looks like a noisy typewriter and the channel has a subset of inputs that produce essentially disjoint sequences at the output. For each typical input n-sequence there are roughly $2^{nH(Y|X)}$ possible Y sequences, all of them equally likely. We wish to ensure that two different X sequences produce different Y sequences. There are roughly $2^{nH(Y)}$ typical Y sequences, hence roughly $2^{nH(Y)-nH(Y|X)} = 2^{nI(X;Y)}$ different sets: code rate of $I(X;Y)$.

Random coding We first prove that rates $R < C$ are achievable. Fix $p(x)$. Generate independently 2^{nR} codewords according to the distribution

$$p(x^n) = \prod_{i=1}^{n} p(x_i).$$

The codewords are represented as the rows of a matrix

$$\mathsf{C} = \begin{pmatrix} x_1(1) & \dots & x_n(1) \\ \vdots & \vdots & \vdots \\ x_1(2^{nR}) & \dots & x_n(2^{nR}) \end{pmatrix}$$

The probability of generating a particular code $\mathscr{C}$ is

$$\mathbb{P}(\mathsf{C}) = \prod_{w=1}^{2^{nR}} \prod_{i=1}^{n} p(x_i(w)).$$

Consider the following sequence of events:

1. A random code C is generated.
2. Sender and receiver know C and the channel $p(y|x)$.
3. A message W is chosen according to a uniform distribution

$$\mathbb{P}(W = w) = 2^{-nR}, \quad w = 1, \dots, 2^{nR}.$$

4. The w-th codeword $X^n(w)$ is sent over the channel.
5. The receiver receives Y^n according to

$$p(y^n|x^n(w)) = \prod_{i=1}^{n} p(y_i|x_i(w)).$$

6. The receiver guesses which message was sent using typical set decoding: he declares that $\hat{W}$ was sent if
   - $(X^n(\hat{W}), Y^n)$ is joint typical and
   - $(X^n(w), Y^n)$ is not jointly typical for any other w.
7. Decoding error if $\hat{W} \neq W$. Let E be the error event.

Analysis of Probability of error Idea: consider the average over all codes.

By symmetry of the code construction, the average probability of error averaged over all codes does not depend on the particular index that was sent. So, let us assume $W = 1$ was transmitted. We obtain

$$\begin{aligned}\mathbb{P}(E) &= \sum_{\mathscr{C}} P(\mathscr{C}) P_e^{(n)}(\mathscr{C}) \\ &= \mathbb{P}(E|W=1).\end{aligned}$$

Denoting the event $E_w := \{(X^n(w), Y^n) \in A_\epsilon^{(n)}\}$, we have

$$\begin{aligned}\mathbb{P}(E|W=1) &= P(E_1^c \cup E_2 \cup \ldots \cup E_{2^{nR}}) \\ &\leq P(E_1^c) + \sum_{w=2}^{2^{nR}} P(E_w).\end{aligned}$$

By the joint AEP, $P(E_1^c) \leq \epsilon$ for n large enough. By the code generation, $X^n(1)$ and $X^n(i)$ are independent, so are Y^n and $X^n(w)$, $w \neq 1$. Hence by the joint AEP,

$$P(E_w) \leq 2^{-n(I(X;Y)-3\epsilon)},$$

whence

$$\mathbb{P}(E) \leq 2\epsilon$$

for $R < I(X;Y) - 3\epsilon$ and n large enough.

Finishing up the proof of achievability We finish the proof as such.

1. Choose $p(x)$ which achieves capacity: this replaces $I(X;Y)$ with C.
2. Get rid of average over codebooks. There exists a codebook C with average error $\leq 2\epsilon$.
3. Throw away half the codewords in C (the ones with highest λ_w) to obtain C^*. We have $\lambda^{(n)}(\mathsf{C}^*) \leq 4\epsilon$ and the rate of C^* is $R - \frac{1}{n} \to R$.

41.3 Converse

Fano's inequality The decoder wishes to estimate X with distribution $p(x)$. They observe Y, related to X by the conditional distribution $p(y|x)$. From Y, they calculate an estimate $g(Y) = \hat{X}$ of X.

Let the probability of error be

$$P_e := \mathbb{P}(\hat{X} \neq X).$$

Fano's inequality relates P_e to $H(X|Y)$.

Theorem 41.8 (Fano's inequality)**.**

$$H(P_e) + P_e \log(|\mathscr{X}| - 1) \geq H(X|Y).$$

Proof. Because this inequality is based on simple entropy calculations, we give its proof in full. Define an error random variable

$$E = \begin{cases} 1 & \text{if } \hat{X} \neq X \\ 0 & \text{if } \hat{X} = X. \end{cases}$$

By the Venn diagram of entropy,

$$H(X|Y) = H(E|Y) + H(X|E,Y) - H(E|X,Y).$$

However,

$$\begin{aligned} H(E|X,Y) &= 0, \\ H(E|Y) &\leq H(E) = H(P_e), \\ H(X|E,Y) &= (1-P_e)H(X|Y,E=0) + P_e H(X|Y,E=1) \leq P_e \log(|\mathscr{X}|-1). \end{aligned}$$

Thus $H(X|Y) \leq H(P_e) + P_e \log(|\mathscr{X}| - 1)$. □

Converse to the coding theorem We will show that if the probability of error tends to zero, then the code rate must be at most C.

Lemma 41.9 (Fano's inequality)**.** *For a DMC with a codebook $\mathscr{C}$ and the input messages uniformly distributed, let $P_e^{(n)} = \mathbb{P}(W \neq g(Y^n))$. Then*

$$H(W|Y^n) \leq 1 + P_e^{(n)} nR.$$

We can easily show that using the channel multiple times does not increase the capacity.

Lemma 41.10. *Let Y^n be the result of passing X^n through a DMC. Then*

$$I(X^n; Y^n) \leq nC.$$

Proof of converse If $\lambda^{(n)} \to 0$, then $P_e^{(n)} \to 0$. For each n, let W be drawn according to a uniform distribution, then $P_e^{(n)} = \mathbb{P}(\hat{W} \neq W)$ and

$$\begin{aligned} nR = H(W) &= H(W|Y^n) + I(W; Y^n) \\ &\leq 1 + P_e^{(n)} nR + nC. \end{aligned}$$

Divide by n, take the limit, we obtain $R \leq C$.

Strong converse We showed that

$$P_e^{(n)} \geq 1 - \frac{C}{R} - \frac{1}{nR} \to 1 - \frac{C}{R}.$$

Therefore, if the rate is higher than the capacity, the probability of error is bounded away from zero. In fact, there is a strong converse: if $R > C$, then $P_e^{(n)} \to 1$ exponentially. The capacity is a clear dividing point!

Bibliography

[1] T.C. Bell, J. G. Cleary, and I.H. Witten. *Text Compression*. Prentice Hall, 1990.

[2] Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*, volume 129 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, second edition, 2010.

[3] Gérard Cohen, Iiro Honkala, Simon Litsyn, and Antoine Lobstein. *Covering Codes*. North-Holland, 1997.

[4] T. Cover and R. King. A convergent gambling estimate of the entropy of english. *IEEE Transactions on Information Theory*, 24(4):413–421, July 1978.

[5] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, second edition, 2006.

[6] Imre Csiszár and János Körner. *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Cambridge University Press, second edition, 2011.

[7] Robert G. Gallager. Variations on a theme by huffman. *IEEE Transactions on Information Theory*, 24:668–674, November 1978.

[8] Abbas El Gamal and Young-Han Kim. *Network Information Theory*. Cambridge University Press, 2011.

[9] Byron Knoll and Nando de Freitas. A machine learning perspective on predictive coding with PAQ. *Arxiv*, pages 1–30, 2011.

[10] D. Korn, J. MacDonald, J. Mogul, and K. Vo. The vcdiff generic differencing and compression data format. *Network Working Group, Request for Comments*, 3284, 2002.

[11] B. Mandelbrot. An information theory of the statistical structure of language. *Proc. Symposium on Applications of Communication Theory*, pages 486–500, September 1952.

[12] Soheil Mohajer, Payam Pakzad, and Ali Kakhbod. Tight bounds on the redundancy of huffman codes. *IEEE Transactions on Information Theory*, 58:6737–6746, November 2012.

[13] David Salomon. *Data Compression: The Complete Reference*. Springer, 3rd edition, 2004.

[14] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, fourth edition, 2012.

[15] Claude Shannon. Prediction and entropy of printed english. *Bell System Technical J.*, pages 50–64, January 1951.

[16] Douglas R. Stinson. *Cryptography Theory and Practice*. Chapman & Hall/CRC, second edition, 2002.

[17] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice Hall, second edition, 2002.

[18] Raymond W. Yeung. *Information Theory and Network Coding*. Springer, 2008.

[19] G. Zipf. *Human behavior and the principle of least effort*. Addison-Wesley, 1949.