Fundamentals

(Chapter 2)

Chapter Overview

After this chapter you should:

- have understood fundamental limits to lossless compression.
- understood (in detail) what memoryless and Markov sources are, including their formal definition, and know how to compute the entropy of a given memoryless or Markov source.
- have understood the idea of unique decodability and understood how to use the Kraft-McMillan inequality.
- have understood how to encode integers using the Elias-gamma (Elias-γ) code and the Golomb code.

Lectures Overview

This chapter is scheduled to take about four lectures.

- Absolute limits on lossless compression.
- Memoryless sources.
- Markov sources.
- Coding integers.

Outline

- Absolute limits on compression.
  - There is no compression algorithm that losslessly compresses every file by even 1 byte.
  - There is no compression algorithm that losslessly compresses 1% of files by (exactly) 1 byte.
- The files we compress are very special!

Absolute Limits on Compression

Theorem

There is no compression algorithm that losslessly compresses every file by even 1 byte.

Proof. Assume an algorithm c exists that compresses every file by at least 1 byte. Let f be a file; let $c(f)$ be the compressed file output by c.

For all files f, $c(f)$ is at least 1 byte shorter than f.

$$f \to c(f) \to c(c(f)) \to \cdots \to \text{empty file}$$

Invert every step to recover f?

$\Rightarrow$ Contradiction!

Absolute Limits on Compression

Theorem

There is no compression algorithm that losslessly compresses 1% of files by (exactly) 1 byte.

Proof: Let c be the compression function. The first observation is:

- If f and g are two different files, then the compressed files $c(f)$ and $c(g)$ *must be* different.
- Proof by contradiction. If $f \neq g$ but $c(f) = c(g)$ then given the compressed file we cannot figure out which of f or g was the original file. So the compression is not lossless.

Absolute Limits on Compression

Theorem (cont'd)

There is no compression algorithm that losslessly compresses 1% of files by (exactly) 1 byte.

Some notation. For any $i \geq 0$:

- let S_i be the set of all files of size i bytes.
  - $S_0 = \{\text{empty file}\}$.
  - $S_1 = \{\ldots, a, \ldots, z, A, \ldots, Z, 0, \ldots, 9, \ldots\}$
  - $S_2 = \{.., aa, ab, .., zz, .., AA, .., ZZ, 0A, .., 9Z, ..\}$
  - and so on.
  - $|S_0| = 1$, $|S_1| = 256$, $|S_2| = 256^2$. In general, $|S_i| = 256^i$.
- let C_i be the set all files of size i bytes that c can compress by (exactly) 1 byte. Note: $C_i \subseteq S_i$.

Absolute Limits on Compression

Theorem (cont'd)

There is no compression algorithm that losslessly compresses 1% of files by (exactly) 1 byte.

- If f is a file with i bytes which can be compressed then $c(f)$ has $i-1$ bytes.
  - if $f \in C_i$ then $c(f) \in S_{i-1}$.
- Recall that two different i-byte files must be compressed to two *different* smaller files.
  - If $f \in C_i$ and $g \in C_i$, and $f \neq g$ then $c(f) \neq c(g)$.
- By the pigeon-hole principle, $|C_i| \leq |S_{i-1}|$. In words: the number of i-byte files compressible by (exactly) 1 byte is at most the number $i-1$ byte files.

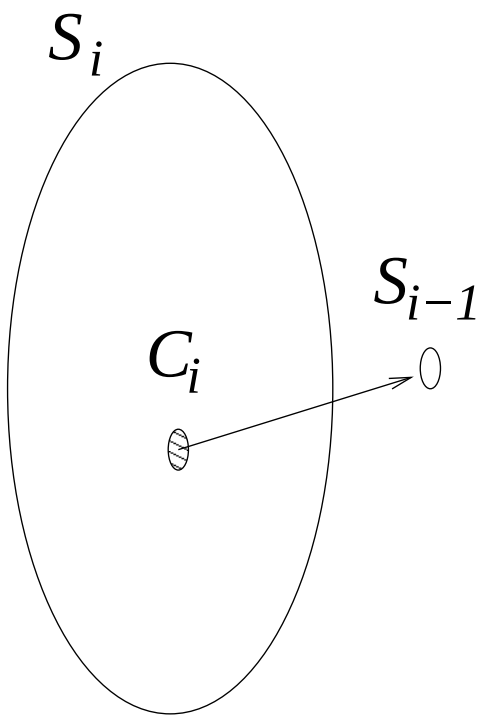
Absolute Limits on Compression

Theorem (cont'd)

There is no compression algorithm that losslessly compresses 1% of files by (exactly) 1 byte.

Recall that $|C_i| \leq |S_{i-1}|$. Now we compare the sizes of C_i and S_i.

- $|S_i| = 256^i$, $|S_{i-1}| = 256^{i-1}$.
- So, $|S_{i-1}| = |S_i|/256$.
- So, $|C_i| \leq |S_{i-1}| = |S_i|/256$.
- So, $|C_i| \leq |S_i|/256$ ($< 0.5\%$!).



Summary: number of files one can losslessly compress is at most the number of smaller files. The number of smaller files is small by comparison to the number of files we want to compress.

Q: Why is Compression Possible?

A: The files we encounter in real life are very special files. Consider just small files of 100 characters, and assume:

- 10 billion (10^{10}) computers printing 1 billion characters per second or 10 million (10^7) files per second, running for 100 years $\sim 3.15 \times 10^9 < 10^{10}$ seconds).
- files with 100 characters produced: $< 10^{10} \times 10^7 \times 10^{10} = 10^{27}$.
- files with 100 characters: $256^{100} \sim 10^{240}$.

Thus, all possible 100-character files that we could conceivably encounter are a very special tiny tiny tiny minority of all possible files.

Outline

- Modelling and coding (review).
- Memoryless models.
- Shannon's theorem and Entropy.

Modelling and Coding

Modelling: Why is a file "special"? Can we model its characteristics?

- ▷ Next: memoryless model.

Coding: Knowing the characteristics, can we find a way to compress the data?

Memoryless models (review)

- Such models assume that in most files, some symbols are more frequent than others.
- These models use *only* frequency information. They do not, for example, consider the context in which a symbol appears.
- Memoryless models are very simple, so the compression you get by modelling a file in a memoryless way is usually not very good.
- However, memoryless models apply to a wide range of files.

Memoryless Source

Definition (Memoryless Source)

- Outputs symbols from an *alphabet* A consisting of n symbols $\sigma_1, \ldots, \sigma_n$.
- Symbol σ_i is output with probability p_i, and $\sum_{i=1}^{n} p_i = 1$.
- The symbols are output in a 'memoryless' fashion: the probability that the next symbol output is σ_i is always p_i, regardless of what symbols were output previously.

- Models frequency information and nothing else.
- Example MLS: alphabet $\{a, b, c\}$. So, $\sigma_1 = a, \sigma_2 = b, \sigma_3 = c$. Probabilities $p_1 = 0.2 = \Pr(a)$, $p_2 = 0.3 = \Pr(b)$, $p_3 = 0.5 = \Pr(c)$.
- After outputting *aaaaaaaaaaa*, the probability that the next symbol output by the MLS is a is still only 0.2.

Shannon's Theorem

Theorem (Shannon's Theorem)

Given a MLS with alphabet $\{\sigma_1, \ldots, \sigma_n\}$ *and with* $\Pr[\sigma_i] = p_i$*, the optimal variable-length code (i.e. that has the lowest possible average code length, and gives lossless compression) uses* $\log_2(1/p_i)$ *bits to code* σ_i*, for all* $i = 1, \ldots, n$*.*

- Probably the most fundamental theorem in information theory.
- Example MLS: alphabet $\{a, b, c\}$, probabilities $0.2, 0.3, 0.5$.
  The optimal code is **the** code which uses:
  - $\log_2(1/0.2) = 2.322$ bits (3dp) for a;
  - $\log_2(1/0.3) = 1.737$ bits (3dp) for b;
  - $\log_2(1/0.5) = 1$ bit for c;

  Any code that tries something different is either not lossless or has higher average code length.

Shannon's Theorem

Theorem (Shannon's Theorem)

Given a MLS with alphabet $\{\sigma_1, \ldots, \sigma_n\}$ *and with* $\Pr[\sigma_i] = p_i$*, the optimal variable-length code (i.e. that has the lowest possible average code length, and gives lossless compression) uses* $\log_2(1/p_i)$ *bits to code* σ_i*, for all* $i = 1, \ldots, n$*.*

- Low (high) probability symbol: longer (shorter) code length.
- Theorem gives information content of symbol. Low probability symbols have higher information content.

> *Inspector Gregory: "Is there any point to which you would wish to draw my attention?"*
> *Sherlock Holmes: "To the curious incident of the dog in the night-time."*
> *Inspector: "The dog did nothing in the night-time."*
> *Holmes: "That was the curious incident."*

Entropy

Definition (Entropy)

Given a MLS with alphabet $\{\sigma_1, \ldots, \sigma_n\}$ and with $\Pr[\sigma_i] = p_i$, the *entropy* of source S, denoted by $H(S)$, is given by the formula:

$$H(S) = p_1 \log_2(1/p_1) + p_2 \log_2(1/p_2) + \ldots + p_n \log_2(1/p_n)$$

Example MLS S: alphabet $\{a, b, c\}$, probabilities $0.2, 0.3, 0.5$.

$$H(S) = 0.2 \log_2(\frac{1}{0.2}) + 0.3 \log_2(\frac{1}{0.3}) + 0.5 \log_2(\frac{1}{0.5}) = 1.49 \text{ (2dp) bits/symb.}$$

- By Shannon's theorem, $H(S)$ is the optimal average number of bits to code the output of an MLS S.

Entropy

- Entropy: measure of information content per symbol of MLS.
- Can't losslessly compress output of MLS beyond its entropy.
- If probabilities of symbols are very skewed (some symbols much more likely than others), the entropy will be low. The file is *compressible*.
- If probabilities of symbols are all equal (a uniform distribution) then the entropy is maximum. The file is *not compressible*.
  - If $p_1 = \ldots = p_n = 1/n$, $H = \log_2 n$ (maximum).

Entropy (Examples)

Let the alphabet be $A = \{a, b, c, d\}$. The raw data should be encoded using a code like $a = \mathbf{00}$, $b = \mathbf{01}$, $c = \mathbf{10}$ and $d = \mathbf{11}$. This takes 2 bits/symbol.

Now take two MLS:

1. $\Pr[a] = 0.6$, $\Pr[b] = 0.2$, $\Pr[c] = \Pr[d] = 0.1$.
   - $H = -(0.6 \times \log_2 0.6 + 0.2 \times \log_2 0.2 + 0.1 \times \log_2 0.1 + 0.1 \times \log_2 0.1 = 1.57$ bits/symbol (2dp).
   - This MLS is compressible.
2. $\Pr[a] = \Pr[b] = \Pr[c] = \Pr[d] = 0.25$.
   - $H = 0.25 \times 2 + \ldots + 0.25 \times 2 = 2$ bits/symbol.
   - This MLS is not compressible.

Outline

- Markov models
- Stationary probability
- Entropy of Markov sources
- Differences between Markov sources and MLS

Markov models (review)

Probability of symbol is fixed in memoryless model. Is this realistic?

`stati`?

`C 0.025775 E 0.112578 O 0.058215 N 0.056035 S 0.060289`

- Probability depends on context. Most of the time, given some context, the probability distribution becomes more skewed — better compression.
- Entropy of the frequency distribution of English alphabet (i.e. memoryless modelling) uses over 3.5 bits/symbol.
- Human studies show that English text can be coded in very few bits, between 0.6 and 1.3 bits/symbol.
- We should try to model context information.

Markov source (definition)

Definition (Markov Source)

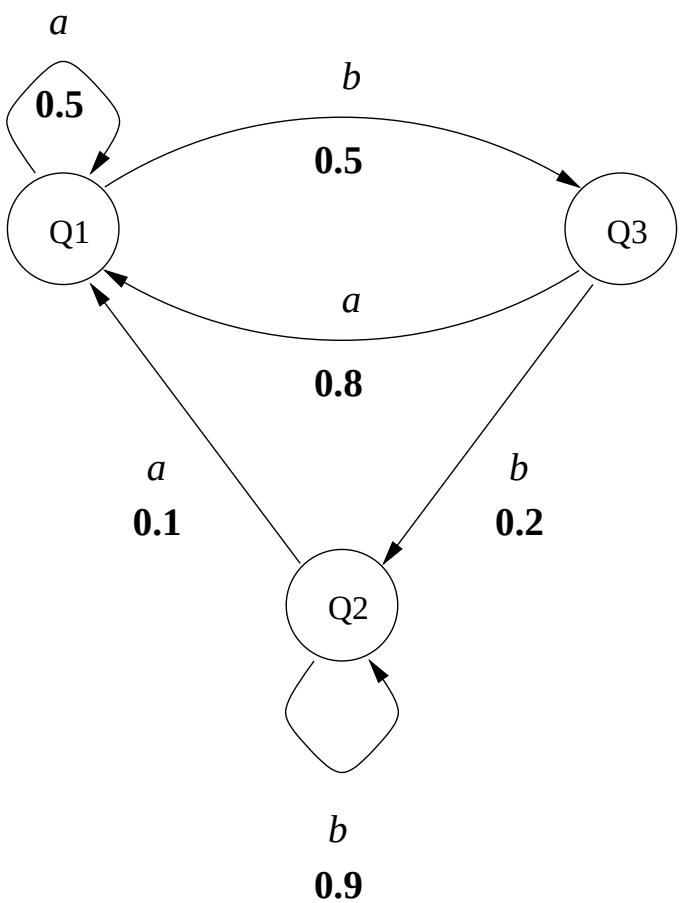
A Markov source S over an alphabet $A = \{\sigma_1, \ldots, \sigma_n\}$ comprises

- A set of *states* $Q = \{q_1, \ldots, q_m\}$, and,
- a set of *transitions* $T \subseteq Q \times Q \times A$. Each transition is labelled with a probability.

In addition, for any state q:

- all transitions of the form (q, q', σ) have probabilities adding to 1. In words, all transitions going out of a state have probabilities adding up to 1.
- all transitions of the form (q, q', σ) have different symbols σ associated with them. In words, all transitions going out of a state have different symbols associated with them.

Markov source (example)



- States Q_1, Q_2, Q_3.
- Transitions are:
  - (Q_1, Q_1, a), with prob 0.5,
  - (Q_1, Q_3, b), with prob 0.5,
  - (Q_2, Q_1, a), with prob 0.1,
  - $\vdots$

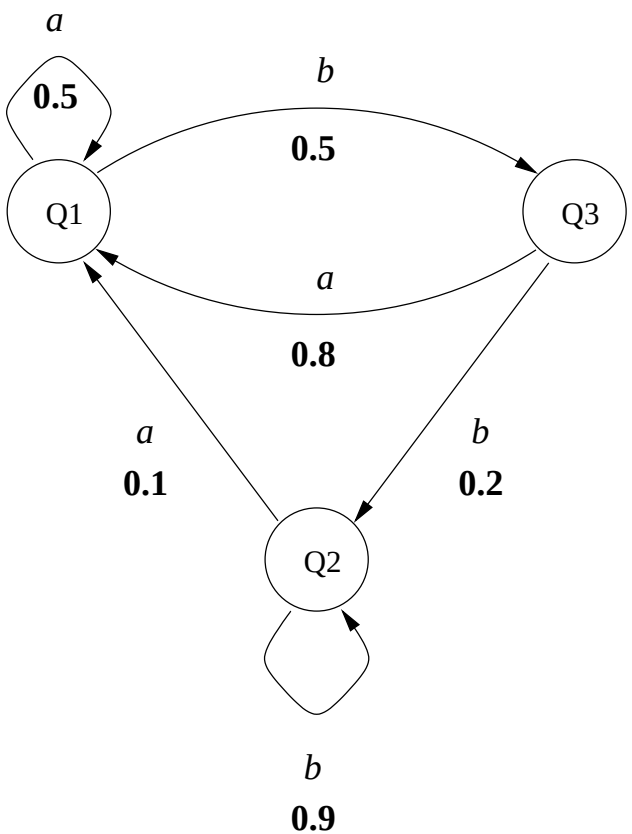
Follow transitions according to probabilities, output symbols accordingly.

▷ Source on left generates *aba* with probability 0.2 starting in Q_1 ($0.5 \times 0.8 \times 0.5$).

▷ The state of the Markov source is a way to remember some context information.

Stationary Probability

The probability that a Markov source is in a particular state *appears* to depend on the start state and how many steps it runs.



Step	q_1	q_2	q_3
0	1.000	0.000	0.000
1	0.500	0.000	0.500
2	0.650	0.100	0.250
3	0.535	0.140	0.325
4	0.542	0.191	0.268
...	...	...	...
10	0.441	0.334	0.225
...	...	...	...
20	0.406	0.390	0.204
...	...	...	...
39	0.400	0.400	0.200
40	0.400	0.400	0.200
41	0.400	0.400	0.200
...	...	...	...

Stationary Probability

Definition (Stationary probability)

In the long run, for "most" Markov sources, the probability of being in state q_i converges to a value called the *stationary probability* of that state. The value of the stationary probability is independent of the start state. The stationary probability of state i is denoted by $\Pr(q_i)$.

(Note: Google's PageRank is basically a stationary probability.)

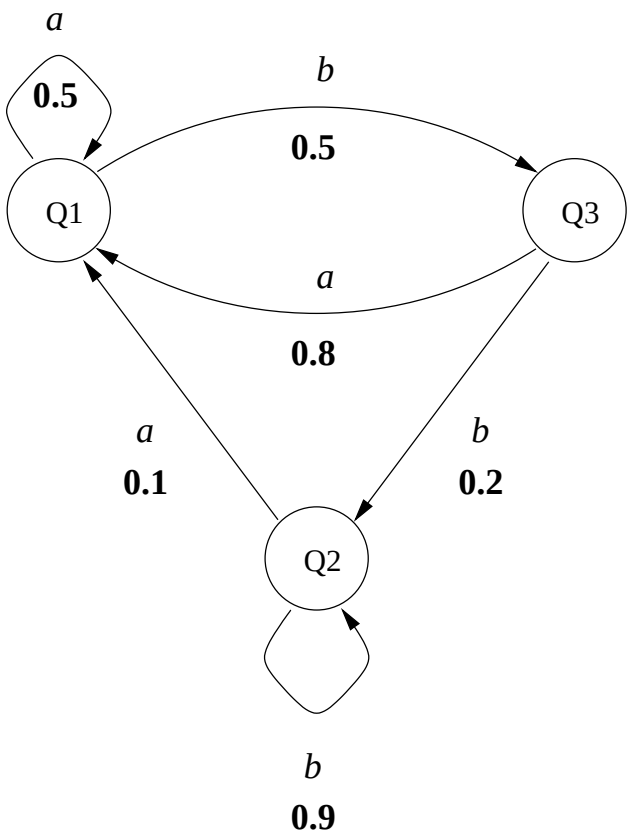
Entropy of Markov Source

Definition (Entropy of Markov Source)

Let S be a Markov source with states $q_1, \ldots, q_m$, and let $H(q_i)$ and $\Pr(q_i)$ be the entropy and stationary probability, respectively, of state q_i. Then the entropy of S, denoted by $H(S)$, is given by:

$$H(S) = H(q_1) \times \Pr(q_1) + \ldots + H(q_m) \times \Pr(q_m).$$

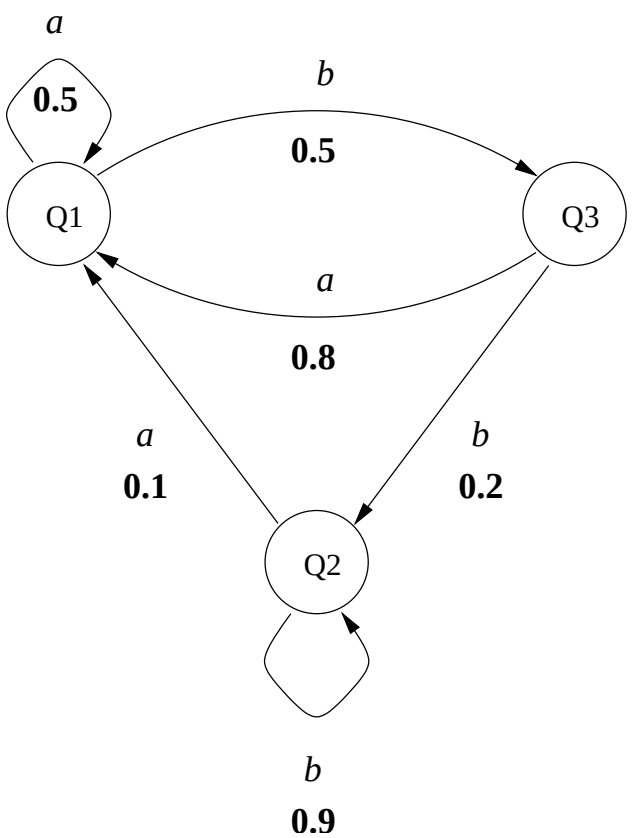
Example: Entropy of Markov Source



State	Q_1	Q_2	Q_3
Stat Prob ($\Pr(Q_i)$)	0.40	0.40	0.20
Entropy ($H(Q_i)$)	1.00	0.46	0.72

$$
\begin{aligned}
H(S) &= \Pr(Q_1) \times H(Q_1) + \Pr(Q_2) \times H(Q_2) + \\
&\quad \Pr(Q_3) \times H(Q_3) \\
&= 0.40 \times 1.00 + 0.40 \times 0.46 + \\
&\quad 0.20 \times 0.72 \\
&= 0.732 \text{ bits/symbol.}
\end{aligned}
$$

Symbol Frequencies



- Frequency of a = $\Pr(q_1) \times \Pr[\text{outputting } a \text{ in } q_1] + \Pr(q_2) \times \Pr[\text{outputting } a \text{ in } q_2] + \Pr(q_3) \times \Pr[\text{outputting } a \text{ in } q_3] = 0.4 \times 0.5 + 0.4 \times 0.1 + 0.2 \times 0.8 = 0.4$
- Frequency of $b = 1 - 0.4 = 0.6$.
- The entropy is ~ 0.732 bits/symbol.
- If we had an MLS with $\Pr(a) = 0.4$ and $\Pr(b) = 0.6$ then the entropy would be ~ 0.972 bits/symbol.
- Careful modelling gives better compression.

Example Outputs

Markov source

```
bbbbbbbbbabbbbbbbbbaabaab
abaabbbbabbbabaaaabbbbbbb    [Number a's = 38
bbbbababaababababaaaaaaaa     Number b's = 62]
bbbbaaabbbbaabababbbbbabbb
```

Runs of $b = 9, 9, 1, 1, 1, 4, 3, 1, 11, 1, 1, 1, 1, 1, 1, 4, 4, 1, 6, 3.$

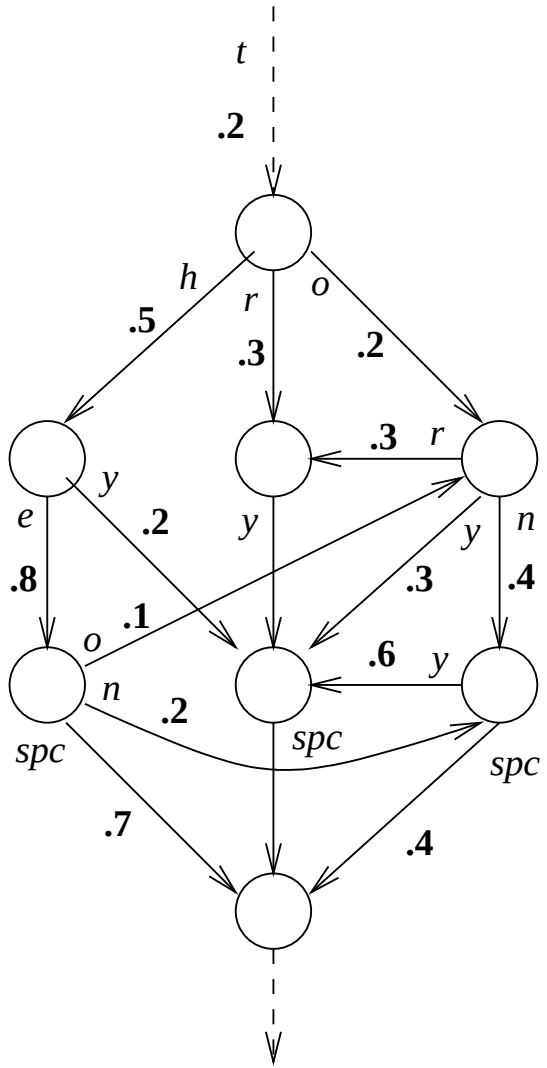
MLS

```
babbabbbbabababaabbabaabbb
bbabaaabaabababababaabbbbb    Number a's = 41
bbaabaaabbbbaabababababaabb   Number b's = 59
aaabbabbbbabbabbbbbabbbbbb
```

Runs of $b = 1, 2, 4, 1, 1, 2, 1, 5, 1, 1, 1, 1, 1, 1, 1, 6, 1, 4, 1, 1, 1, 1, 2, 2, 4, 2, 5, 5.$

Summary

- Files with similar frequency counts may come from very different sources.
- For lossless compression of most kinds of raw data, Markov models are likely to give better compression. (Fake Markov model of English text on the right.)



Outline

- Unique decodability.
- Kraft-McMillan inequality.
- Prefix codes.

Unique Decodability

Alphabet $A = \{a, b, c, d\}$, using variable-length encoding of symbols.

▷ decoding is not obvious.

	Prob.	Code A	Code B
a	0.5	0	0
b	0.25	10	10
c	0.125	110	110
d	0.125	111	101

Avg bits/symbol: 1.75.

Code A: 010111110 = ?

Code B: 0101010 = abda = adab = abbb.

Unique Decodability

Kraft-McMillan inequality

Definition

Suppose that we have n symbols $\sigma_1, \sigma_2, \ldots, \sigma_n$, and n lengths $l_1, l_2, \ldots, l_n$. A code which assigns binary codes to these symbols such that σ_1 is coded using l_1 bits, σ_2 using l_2 bits, ..., σ_n using l_n bits, and is uniquely decodable, must satisfy this equation:

$$2^{-l_1} + 2^{-l_2} + \ldots + 2^{-l_n} \leq 1. \quad (1)$$

- If (1) is true, then we *can* construct uniquely decodable codes, with a little care (feasible).
- If (1) is not true, then we *cannot* construct uniquely decodable codes, no matter what (not feasible).

K-M inequality $\Rightarrow$ Shannon's Theorem

Example: Kraft-McMillan inequality

Symbol	a	b	c	d
Code-length X	1	2	2	3
Code-length Y	1	2	3	3

$$(\mathrm{X}) \sum_{i=1}^{4} 2^{-l_i} = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^2} + \frac{1}{2^3} = \frac{9}{8} > 1.$$

Hence not feasible (can't construct codes).

$$(\mathrm{Y}) \sum_{i=1}^{4} 2^{-l_i} = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^3} = 1.$$

Hence feasible (can construct codes).

Unique Decodability

- *Prefix* codes (aka *prefix-free* codes) are those where no code is a prefix of another code.
  ▷ 0, 10, 110, 1111 $\surd$.
  ▷ 0, 10, 110, 1011 $\times$.
- Prefix codes are uniquely decodable.
  - All prefix codes satisfy KM inequality (obviously).
- There are uniquely decodable codes that are *not* prefix codes: $a = 0$, $b = 01$, c=11.

Unique Decodability

To check if a code is UD:

1. Does the code satisfy KMI? If no, NOT UD. If yes, go to (2).
2. Is the code a prefix code? If yes, UD. If no, go to (3) or (4).
3. Is there a counter-example? If yes, NOT UD. If no, go to (4).
4. Describe algorithm (flowchart/automaton) for decoding.

For example, take the code $a = 0$, $b = 01$, c=11. Answer to (1) is yes. Answer to (2) is no. So we go to (4), and note that if we reverse the codes, we get a prefix code. So the algorithm is: reverse the file, and decode using prefix codes.

Outline

- Problem definition, motivation.
- Unary code.
- Elias-gamma (Elias-γ) code.
- Golomb code.

Coding Integers

Often, we need to code a *sequence* of *non-negative* integers (i.e. each integer is ≥ 0). There is no (useful) upper bound on how big the integers can get.

- Such codes are used in many compression algorithms.
- Good exercise on creating UD codes.

Example sequence: 5, 1, 3, 8. First idea: binary encoding.

- $5 = \mathbf{101}$, $1 = \mathbf{1}$, $3 = \mathbf{11}$, $8 = \mathbf{1000}$.
- Encoded sequence: $\mathbf{1011111000}$
- Decoding? $\mathbf{10}$, $\mathbf{11}$, $\mathbf{1}$, $\mathbf{111000} = 2, 3, 1, 56$.

▷ Unique decodability means each integer must be "self-delimiting".

▷ You can use KMI to check that this was doomed from the start.

Coding Integers: Unary Code

Encode integer $i \geq 0$ as i **1**s followed by a **0**.

- $5 = \mathbf{111110}$, $1 = \mathbf{10}$, $3 = \mathbf{1110}$, $8 = \mathbf{111111110}$.
- Encoded sequence: $\mathbf{11111010111011111110}$.
- Decodable?

▷ Problem: size of integers.

Coding Integers: Golomb Code

- Integer value $M > 1$ known to coder and decoder.
- A given integer $i \geq 0$ written as follows (example: $i = 23$, $M = 10$):

1.	$q = \lfloor i/M \rfloor$ (in unary)	**110**
2.	$r = i \bmod M$ $(\triangleright\ r < M)$	
	Encode r in binary using $\lceil \log_2 M \rceil$ bits.	**0011**
3.	Concatenate (1) and (2)	**1100011**

Decodable: $M = 10$, input $=$ **1110100001011** $= 38, 9$.

Golomb coding is particularly good for run-length encoding.

Coding Integers: Gamma Code

Given an integer $i \geq 1$ (example: $i = 5$).

1.	Write i in binary.	**101**
2.	Remove leading **1**.	**01**
3.	Encode length of (2) in unary.	**110**
	Concatenate (3) and (2).	**11001**

Decodable (why?).