Information and Coding

Part I: Data Compression

Marek Rychlik

Department of Mathematics
University of Arizona

February 25, 2016

What is coding — informally?

Definition (Coding)

Coding is algorithmic rewriting of data, while achieving desirable objectives.

Three main types of objectives

1. Reducing the cost of storage and transmission time required (data compression);
2. Preventing data loss due to random causes, such as defects in physical media (error correcting codes);
3. Achieving privacy of comunications (data encryption).

Without data compression we would not have

1. high-definition digital streaming video;
2. cellular telephony;
3. high-definition digital cinema.

Coriolanus — the play by William Shakespeare

"Before we proceed any further, hear me speak. Speak, speak. You are all resolved rather to die than to famish? Resolved. resolved. First, you know Caius Marcius is chief enemy to the people. We know't, we know't. Let us kill him, and we'll have corn at our own price. Is't a verdict? No more talking on't; let it be done: away, away! One word, good citizens... [total of 141,830 characters]"

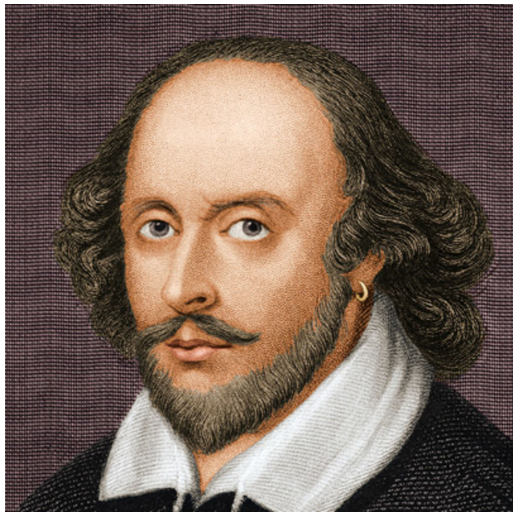
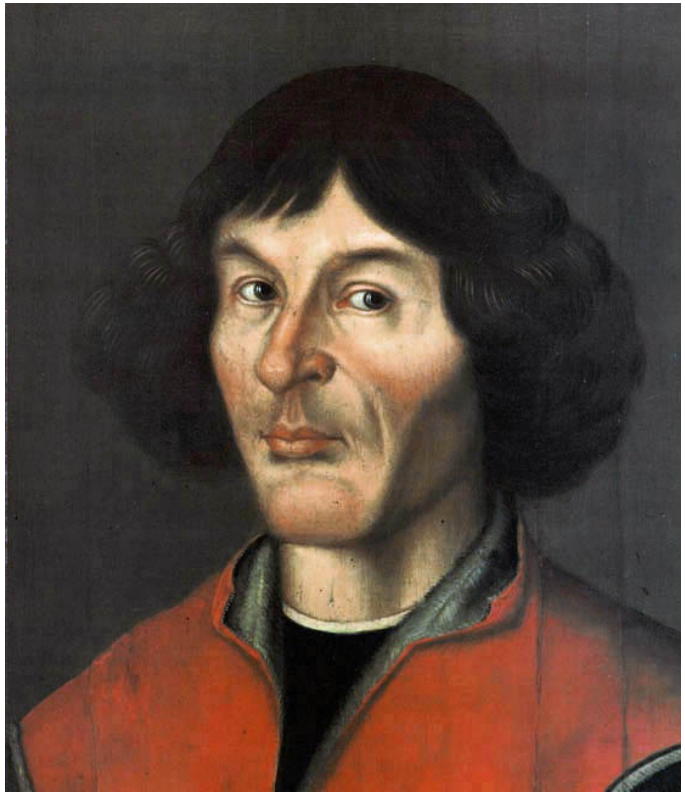
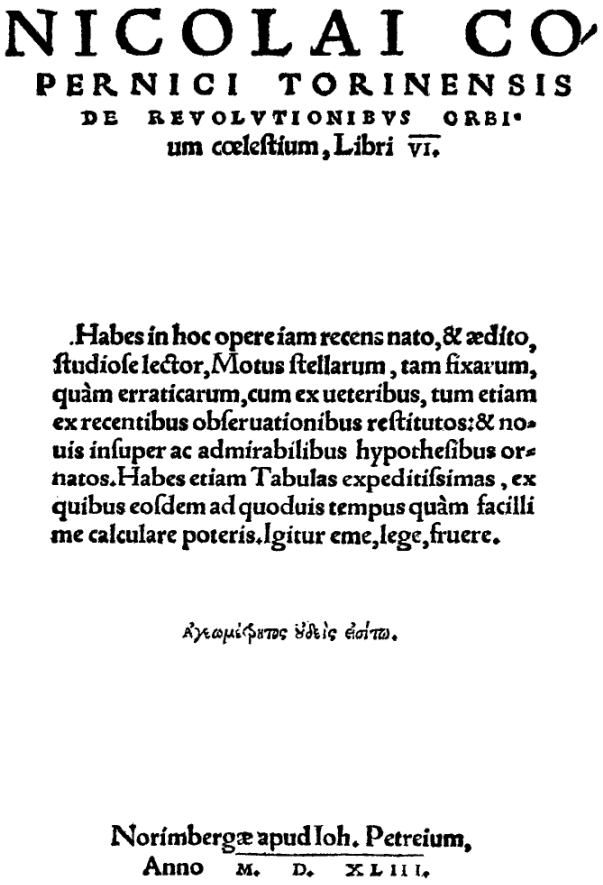


Figure: W. Shakespeare

An early printed scientific paper





NICOLAI CO-
PERNICI TORINENSIS
DE REVOLVTIONIBVS ORBI-
um cœleſtium, Libri VI.

Habes in hoc opere iam recens nato, & ædito, ſtudioſe lector, Motus ſtellarum, tam fixarum, quàm erraticarum, cum ex ueteribus, tum etiam ex recentibus obſeruationibus reſtitutos: & nouis inſuper ac admirabilibus hypotheſibus ornatos. Habes etiam Tabulas expeditiſsimas, ex quibus eoſdem ad quoduis tempus quàm facilli me calculare poteris. Igitur eme, lege, fruere.

Ἀγεωμέτρητος μηδεὶς εἰσίτω.

Norimbergæ apud Ioh. Petreium,
Anno M. D. XLIII.

Figure: Nikolaus Kopernikus, 1543, “Polish astronomer” (Encyclopedia Britannica)

Character encoding

- Characters are numbered.
- The oldest standard is ASCII (American Standard Code for Information Interchange).
- ASCII is pronounced “as key”.
- Encodes characters as numbers 0–255.
- Successor to ASCII is Unicode. It aims at encoding all characters in all human alphabets.
- Only ASCII is used in the current talk.

Bits and Bytes

- The term *bit* stands for a *binary digit* and it is either 0 or 1.
- Bits are digits of base-2 (binary) representation of numbers, e.g. 6 in decimal is 101 in binary.

$$(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 1 = (6)_{10}$$

- It is a *normalized unit of information*; we may think that information comes as a sequence of answers to Yes/No questions. A fraction of a bit makes sense, statistically speaking (e.g. tossing a biased coin).
- A byte is a sequence of 8 bits. A byte can also be thought of as an unsigned integer in the range 0–255.

$$(10000100)_2 = 1 \cdot 2^7 + 1 \cdot 2^2 = 128 + 4 = (123)_{10}$$

Hexadecimal and Octal notation for integers

- 1 byte = 8 bits = 2 groups of 4 bits; $2^4 = 16$ "hexadecimal" = base 16
- 16 digits required for hexadecimal: 0–9, A,B,C,D,E,F (letters used as digits)

$$(1010\ 0100)_2 = (8+2)\cdot 16 + 4 = (A4)_{16} = \backslash xA4$$

- Octal notation uses groups of 3 bits. 8 digits required: 0–7.

$$(10\ 100\ 100)_2 = (244)_8 = \backslash o244$$

- NOTE: 8 is not divisible by 3, the highest octal digit of a byte is 3.

ASCII codes

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Extended ASCII codes

128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Reading some characters of “Coriolanus” in MATLAB

```
%% File: p1.m
% Read first 10 characters from Coriolanus
f=fopen('TextCompression/coriolan.txt','r');
A=fread(f,10,'uint8');
fclose(f);
A'                          % Print in decimal
dec2hex(A)                  % Print in hexadecimal
dec2bin(A)                  % Print in binary
```

Output I

```
>> p1

ans =

    66   101   102   111   114   101    32   119


ans =

42
65
66
6F
```

Output II

```
72
65
20
77
65
20


ans =

1000010
1100101
1100110
1101111
```

Output III

```
1110010
1100101
0100000
1110111
1100101
0100000
```

An important paradigm of data compression

$$\boxed{\text{Data Compression} = \text{Data Modeling} + \text{Source Coding}}$$

A data model of "Coriolanus"

Example (Data Modeling)

Out model of "Coriolanus" is a random data model: Every character is drawn at random from the alphabet, according to the probability distribution

$$P = (p_1, p_2, \ldots, p_n)$$

Numbers p_j are considered parameters, to be estimated.

George Box, a statistician, *circa* 1979

All models are wrong but some are useful.

Arithmetic coding as a source coding approach

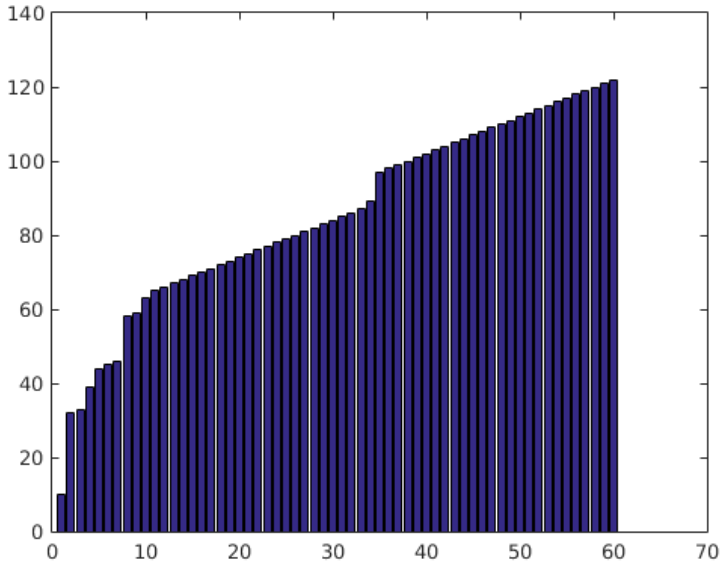
Example (Source Coding)

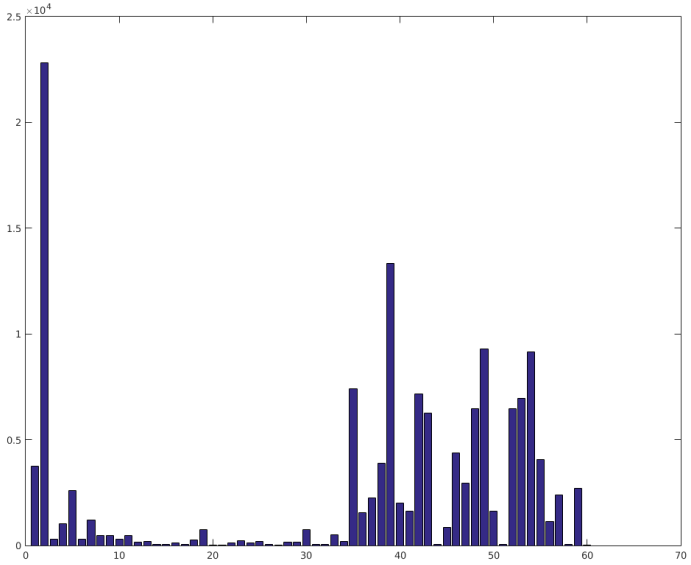
Our source coding technique is arithmetic coding, to be explained later. “Coriolanus” will be encoded as a single binary fraction with approximately $640,000$ binary digits (bits) after the binary point!

The alphabet and "table" of *Coriolanus* in MATLAB

```
%% File: p2.m
%% Read all characters of Coriolanus as ASCII codes
f=fopen('TextCompression/coriolan.txt','r');
A=fread(f,inf,'uint8');
fclose(f);
%% Conduct frequency analysis (build "table")
% C is the alphabet: as per documentation
% of unique, IA and IC are index arrays s.t.
%       C = A(IA) and A = C(IC)
[C,IA,IC]=unique(A);
table=histcounts(IC,1:(length(C)+1));
%% Bar plots of alphabet and table
bar(C);bar(table)
```

The alphabet and “table” bar plots





Entropy of a probability distribution

Definition (Shannon Entropy)

Let $P = (p_1, p_2, \ldots, p_n)$ be
a probability distribution on numbers 1–n.
The number $H(P)$ defined by the formula:

$$H(P) = \sum_{j=1}^{n} (-\log_2 p_j) \cdot p_j$$

is called
the Shannon entropy of the distribution P.

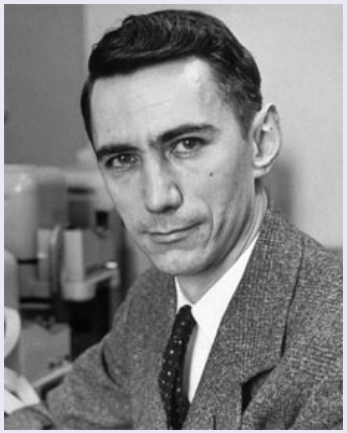


Figure: C. Shannon

Entropy of the “Coriolanus” distribution

```
%% File: p3.m
%% Read data, find table
p2;
%% Find entropy
p=table/sum(table);
entropy=sum(-p.*log2(p))
```

```
>> p3

entropy =

    4.5502

>>
```

Meaning of entropy in data compression

Entropy as mean number of bits per symbol

If entropy is $h = 4.5502$ bits, we expect h bits to be issued in the compressed message, per every symbol of the original message. Since ASCII encoding uses 1 byte (=8 bits), per symbol, se expect compression ratio to be:

$$\frac{8}{4.5502} = 1.7582$$

Calculation of arithmetic code in MATLAB

```
%% File: p4.m
%% Read text file into array A, compute table, alphabet, index array IC
p2;
%% Use arithmetic coding to arithmetically encode the sequence of indices
code=arithenco(IC,table);
compression_ratio=(8*length(A))/length(code)
```

```
>> p4

compression_ratio =

    1.7581
```

What is arithmetic code?

Arithmetic code

Arithmetic code is a binary code which represents digits of a real number $x \in [0, 1]$ uniquely assigned to the original message (Shakespeare's play "Coriolanus"). The number has approximately

$$H(P) \cdot N = 645354$$

binary digits after the binary point:

$$x = 0.d_1 d_2 \ldots$$

where $d_j \in \{0, 1\}$.

Commentary on arithmetic coding

- The idea of arithmetic coding comes from Shannon-Fano-Elias coding.
- The algorithm was implemented in a practical manner by IBM around 1980.
- IBM obtained a patend for arithmetic coding and defended it vigorously for approximately 20 years. The value of the algorithm as intellectual property was estimated at tens of billions of dollars.
- Now there are many freely available implementations of arithmetic coding, including the MATLAB implementation.

The principle of arithmetic coding I

1. Let $m = (a_0, a_1, \ldots, a_N)$ be a message with

$$a_j \in \{1, 2, \ldots, n\}$$

2. Let $P = (p_1, p_2, \ldots, p_n)$ be a relative frequency table of the symbols in a message.
3. Every partial message $m_K = (a_0, a_1, \ldots, a_K)$ is assigned a semi-open subinterval of $I = [0, 1)$.
4. m_0 (the empty message) is assigned $I_0 = I$.
5. If m_{K-1} is assigned interval I_{K-1}, interval $I_K \subseteq I_{K-1}$ and the length of I_K is proportional to p_{a_K} (the probability of the K-th symbol).

The principle of arithmetic coding II

6. As the length of the interval

$$|I_N| = \prod_{j=0}^{N} p_{a_j}$$

$$-\log_2 |I_N| = \sum_{j=0}^{N} (-\log p_{a_j}) = N \cdot \sum_{l=1}^{n} (-\log p_l) \frac{N_l}{N}$$

$$= N \cdot \sum_{l=1}^{n} (-\log p_l) p_l = N \cdot H(P).$$

where N_l is the number of times l appears in the message, and N is the length of the message.

7. We select a number in I_N with the smallest number of binary digits, approximately $N \cdot H(P)$.

Useful definitions I

- An *alphabet* is any finite set A.
- A *message* $m = a_0 a_1 \ldots a_n$ (or $m = (a_0, a_1, \ldots, a_n)$) over the alphabet A is any finite sequence of elements of A. The set of all messages is denoted A^+. We have this disjoint union of Cartesian powers of the alphabet:

$$A^+ = \dot{\bigcup_{n=0}^{\infty}} \bigtimes_{j=1}^{n} A$$

- A *code* is a map $\mathcal{C} : A^+ \to B^+$.

Useful definitions II

- A *symbol code* is a map $\mathcal{C} : A \to B^+$, i.e. a map that maps symbols to one alphabet A to messages (or blocks of symbols) in another alphabet B. The symbol code extends to $\mathcal{C} : A^+ \to B^+$ of all messages over alphabet A to all messages over alphabet B, by concatenation:

$$\mathcal{C}(a_0 a_1 \dots a_n) = \mathcal{C}(a_0)\mathcal{C}(a_1)\dots\mathcal{C}(a_n)$$

- The *length* of a message is its number of symbols; The function $\ell : A^+ \to \{0, 1, \dots\}$ maps messages to their lengths.
- A *binary (symbol) code* is a symbols code $\mathcal{C} : A \to \{0, 1\}$.
- A code $\mathcal{C} : A^+ \to B^+$ is *lossless*, if it is 1:1 as a map.
- A symbol code $\mathcal{C} : A \to B^+$ is called *uniform* iff $\ell \circ \mathcal{C} = \mathrm{const.}$.

Morse code — an old, binary, non-uniform code with some compression

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

An Excercise — "Coriolanus" in Morse Code

Problem (Coriolanus to Morse Code)

Write a program which will encode Shakespeare's "Coriolanus" in Morse code. Comparse the output length to that of the original message (141,830 characters). Also, estimate the duration in "units" of the transmission of "Coriolanus", assuming that the "unit" is $1/4$ *second.*

Speed of Morse Code "Benchmark"

"High-speed telegraphy contests are held; according to the Guinness Book of Records in June 2005 at the International Amateur Radio Union's 6th World Championship in High Speed Telegraphy in Primorsko, Bulgaria, Andrei Bindasov of Belarus transmitted 230 morse code marks of mixed text in one minute."

The expected length of the code

Definition (Expected length of a binary symbol code)

Given a probability distribution $P : A \to [0,1]$ on the alphabet A, the *expected length of a binary symbol code* $\mathcal{C} : A \to \{0,1\}^+$ is defined as:

$$\mathbb{E}(\ell \circ \mathcal{C}) = \sum_{a \in A} \ell(\mathcal{C}(a)) \cdot P(a).$$

Shannon source coding theorem

Theorem (Shannon source coding theorem — Shannon, 1948)

If a binary symbol code $\mathcal{C} : A \to \{0,1\}^+$ *is lossless then*

$$\mathbb{E}(\ell \circ \mathcal{C}) \geq H(P)$$

where $H(P)$ *is the Shannon entropy of the distribution* P*:*

$$H(P) = \sum_{a \in A} (-\log_2 P(a)) \cdot P(a)$$

Kraft inequality (extended variant)

Theorem (Kraft Inequality)

If A is countable and $\mathcal{C} : A \to \{0,1\}^+$ is a lossless symbol code then

$$\sum_{a \in A} 2^{-D(a)} \leq 1.$$

where $D(a) = \ell(\mathcal{C}(a))$.

The existence of nearly optimal codes

Theorem (Shannon-Fano, 1948)

For every alphabet A and a distribution function $P : A \to (0, 1]$ there exists a binary code $\mathcal{C} : A \to \{0, 1\}^+$ such that:

$$H(P) \leq \mathbb{E}(\ell \circ \mathcal{C}) \leq H(P) + 1.$$

Full encoder in MATLAB I

```
%% Specify input and output filenames
infile='coriolan.txt';
outfile='coriolan.zzz';
%% Read uncompressed text file
fileID=fopen(infile);
A=fread(fileID,inf,'uint8');
fclose(fileID);
disp('Entering RTG DEMO text encoder...');
%% Conduct frequency analysis (build "table")
% C is the alphabet: as per documentation of unique,
% IA and IC are index arrays C = A(IA) and A = C(IC)
[C,IA,IC]=unique(A);
table=histcounts(IC,1:(length(C)+1));
%% Find entropy
p=table/sum(table);
entropy=sum(-p.*log2(p));
disp(['Expecting ',num2str(entropy),' bits per symbol']);
%% Use arithmetic coding to encode the sequence
code=arithenco(IC,table);
%% Calculate bits per symbol
bitspersymbol=length(code)/length(A);
disp(['Achieved ',num2str(bitspersymbol),' bits per symbol']);
%% Save compressed data
f=fopen(outfile,'wb+');
fwrite(f,'_RTG');
%% Write data to file
```

Full encoder in MATLAB II

```
% ---------------- Write metadata ----------------
% Write unencoded data length
fwrite(f,length(A),'uint32');
% Write alphabet length
fwrite(f,length(C),'uint8');
% Write alphabet
fwrite(f,C,'uint8');
% Write table
bits_per_table_entry=floor(log2(max(table)))+1;
bytes_per_table_entry=ceil(bits_per_table_entry/8);
if bytes_per_table_entry > 2
    count_class='uint32';
elseif bytes_per_table_entry > 1
    count_class='uint16';
else
    count_class='uint8';
end
fh=str2func(count_class);
counts=fh(table);
fwrite(f,bytes_per_table_entry,'uint8');
fwrite(f,counts,count_class);
% ---------------- Write binary code ----------------
% Padd code to a multiple of 8 bits
padd=ceil(length(code)/8)*8-length(code);
code=[code;zeros(padd,1)];
% Pack code into bytes
packed_code=zeros(length(code)/8,1,'uint8');
```

Full encoder in MATLAB III

```
for i=1:length(packed_code)
    packed_code(i,1)=bi2de(code((1+8*(i-1)):(8+8*(i-1)))');
end
fwrite(f,length(packed_code),'uint32');
fwrite(f,packed_code,'uint8');
%-----------------------------------------------
fclose(f);
disp('Exiting RTG text encoder.');
```

Full decoder in MATLAB I

```
disp('Entering RTG text decoder...');
infile='coriolan.zzz';
outfile='coriolan_d.txt';
%% Decoder
f=fopen(infile,'r');
%% Read and check magic number
magic=fread(f,4,'*char')';
if all(magic=='_RTG')
    disp('Great! Magic number matches.');
else
    disp('Magic number does not match');
    return;
end
%% Read unencoded data length
len=fread(f,1,'uint32');
%% Read alphabet length
alphabet_len=fread(f,1,'uint8');
%% Read alphabet
C=fread(f,alphabet_len,'uint8');
%% Read table
bytes_per_table_entry=fread(f,1,'uint8');
if bytes_per_table_entry > 2
    count_class='uint32';
elseif bytes_per_table_entry > 1
    count_class='uint16';
else
```

Full decoder in MATLAB II

```
        count_class='uint8';
    end
    recovered_table=fread(f,alphabet_len,count_class)';
    %% Read packed code
    packed_code_len=fread(f,1,'uint32');
    packed_code=fread(f,packed_code_len,'uint8');
    %% Unpack the code
    code_len=packed_code_len*8;
    recovered_code=zeros(code_len,1,'double');
    recovered_code = de2bi(packed_code,8)';
    recovered_code = recovered_code(:);
    %% Recover index array
    IC=arithdeco(recovered_code,recovered_table,len);
    %% Apply alphabet
    recovered=C(IC);
    %% Write the file
    f=fopen(outfile,'w');
    fwrite(f,recovered);
    fclose(f);
    %----------------------------------------
    disp('Exiting RTG text decoder...');
```