Improving Run Length Encoding through preprocessing

Sven Fiergolla

14. Januar 2020

Introduction

Basics

Design

Implementation

Evaluation and Discussion

Introduction - A Bit of History

- rise of multimedia
- rise of the World Wide Web
- ever increasing data transfer

Introduction - A Bit of History

- rise of multimedia
- rise of the World Wide Web
- ever increasing data transfer

- compress to save storage space & to handle new types and volumes of data

Introduction - The Situation Today

- burst of sensors and IoT
- massive and rapid increasing data transfer

Introduction - The Situation Today

- burst of sensors and IoT
- massive and rapid increasing data transfer

- compress to lower transmission cost / time
- compress to handle increasing resolution, fidelity, dynamic range
- compression for cold archiving

Basics of Compression

- Non random data contains redundant information
- Compression is about pattern or structure identification and exploitation

Basics of Compression

- Non random data contains redundant information
- Compression is about pattern or structure identification and exploitation
- No algorithm can compress all possible data of a given length, even by one byte (Kolmogorov Complexity)

Basics of Compression - Entropy Encoding

- generating a probability model for the data
- compute variable length codes

Basics of Compression - Entropy Encoding

- generating a probability model for the data
- compute variable length codes
- low speed, high compression strength
- recommended for poorly structured data

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities
- Run Length Encoding (1967)
  - computes runs of identical symbols

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities
- Run Length Encoding (1967)
  - computes runs of identical symbols
- Arithmetic Encoding (1979)
  - encodes a message of symbols in a single rational number in [0,1]

Basics of Compression - Entropy Encoding

- Huffman Encoding (1952)
  - computes optimal length prefix-free codes for symbols acc. to their probabilities
- Run Length Encoding (1967)
  - computes runs of identical symbols
- Arithmetic Encoding (1979)
  - encodes a message of symbols in a single rational number in [0,1]
- Asymmetric Numeral Systems (ANS) Encoding (2014)
  - encodes a message of symbols in a single natural number

Run Length Encoding (RLE)

- employed in the transmission of analog television signals as far back as 1967
- particularly well suited to palette-based bitmap images such as computer icons

Run Length Encoding

$$aaaaabbbbbbaaaaaabb$$

Run Length Encoding

$$aaaaabbbbbbaaaaaabb$$

$$a^5b^6a^6b^2$$

Run Length Encoding

$$aabaabbabbbababaabb$$

Run Length Encoding

$$aabaabbabbbababaabb$$

$$a^2b^1a^2b^2a^1b^3a^1b^1a^1b^1a^2b^2$$

Huffman Encoding

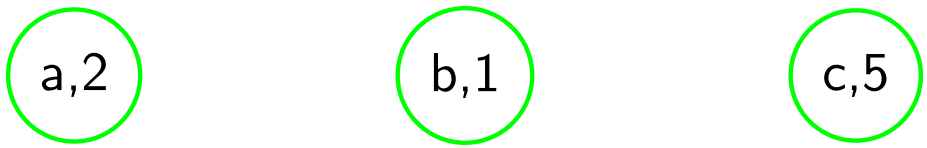


Figure: Example Huffman tree with 3 leaf nodes.

Huffman Encoding

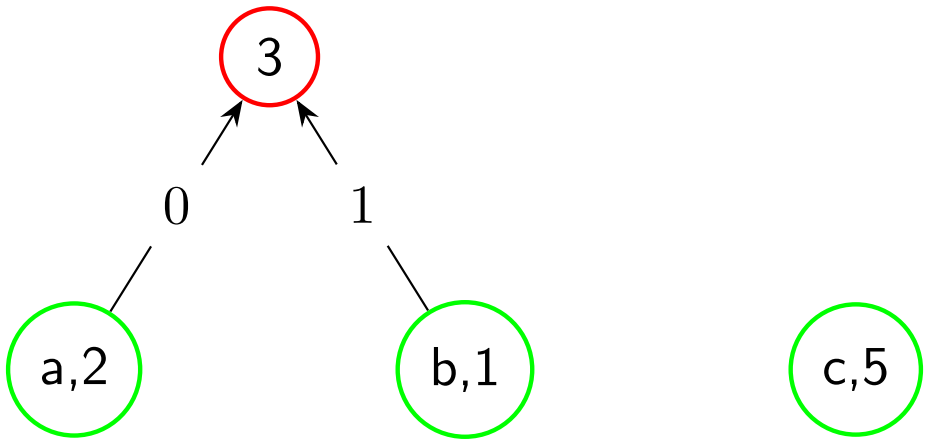


Figure: Example Huffman tree with 3 leaf nodes.

Huffman Encoding

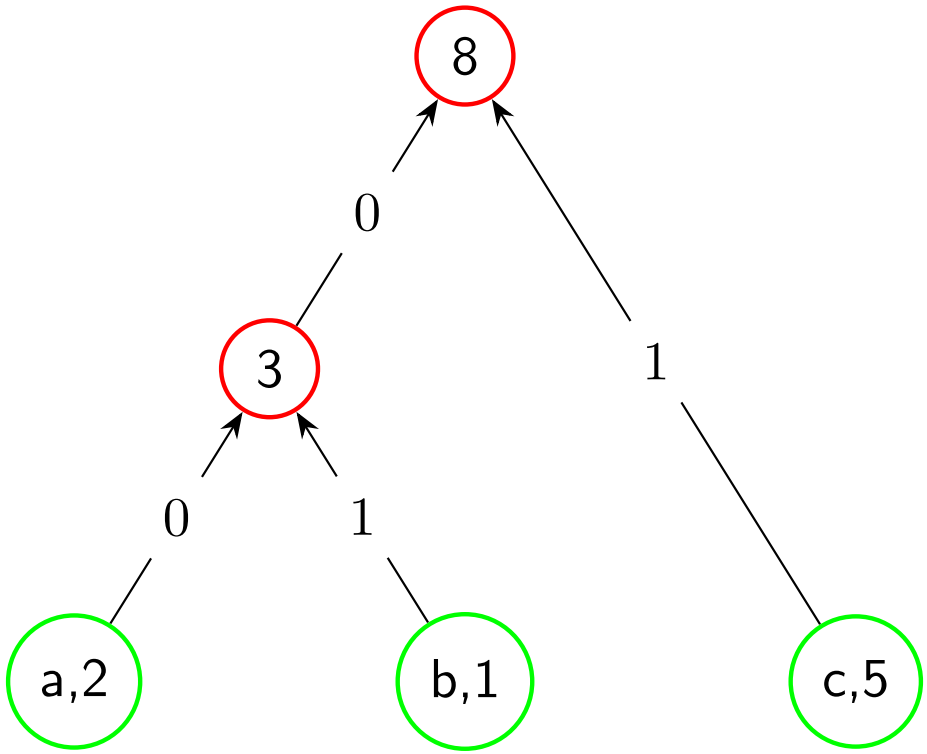


Figure: Example Huffman tree with 3 leaf nodes.

Basics of Compression - Dictionary Encoding

- maintain a dictionary of strings for either a *sliding window* or the whole data
  - replace later occurrence with reference position an length

Basics of Compression - Dictionary Encoding

- maintain a dictionary of strings for either a *sliding window* or the whole data
  - replace later occurrence with reference position an length
- High speed, moderate compression strength

Basics of Compression - Dictionary Encoding

- maintain a dictionary of strings for either a *sliding window* or the whole data
  - replace later occurrence with reference position an length
- High speed, moderate compression strength
- Famous Lempel-Ziv methods LZ77 and LZ78 (1977/78)
  - many derivatives, some still used today

State of the art

method	options	size in bytes	compression	*bps*
uncompressed		3,145,718	100.0%	8.00
compress 4.2.4		1,250,382	40.4%	3.24
gzip v1.10	-9	1,021,720	32.4%	2.60
ZIP v3.0	-9	1,019,783	32.4%	2.59
zstandard 1.4.2	–ultra-23 -long=30	887,004	28.1%	2.25
bzip2 v1.0.8	–best	832,443	26.4%	2.11
brotli 1.0.7	-q 11 -w 24	826,638	26.3%	2.10
p7zip 16.02 (deflate)	a -mx10	821,873	26.1%	2.08
p7zip 16.02 (PPMd)	a -mm=ppmd o=32	763,067	24.2%	1.93
ZPAQ v7.15	-m5	659,700	20.9%	1.67
paq8hp*	-	-	-	-
cmix v18	-c -d	554,983	17.6%	1.41

Table: State of the art compression ratios on the Calgary Corpus.

Design

Design - Calgary Corpus

file	size	description
bib	111261	ASCII text - 725 bibliographic references
book1	768771	unformatted ASCII text
book2	610856	ASCII text in UNIX “troff” format
geo	102400	32 bit numbers in IBM floating point format
news	377109	ASCII text - USENET batch file on a variety of topics
obj1	21504	VAX executable program
obj2	246814	Macintosh executable program
paper1	53161	UNIX “troff” format
paper2	82199	UNIX “troff” format
pic	513216	1728 x 2376 bitmap image
progc	39611	Source code in C
progl	71646	Source code in Lisp
progp	49379	Source code in Pascal
trans	93695	ASCII and control characters

Table: The Calgary Corpus.

Design - Unmodified compression

bits per rle number	byte-wise RLE		binary RLE	
	ratio in %	*bps*	ratio in %	*bps*
8	165	13.20	329	26.38
7	154	12.38	288	23.11
6	144	11.57	248	19.87
5	134	10.77	208	16.66
4	125	10.00	168	13.51
3	116	9.29	131	10.50
2	109	8.74	104	8.36

Table: Byte-wise RLE on the Calgary Corpus.

Design - Unmodified compression

file	size original	$\frac{bits}{\text{RLE number}}$	size encoded	ratio in %	*bps*
pic	513216	2	350292	68.25	5.46
		3	235067	45.80	3.66
		4	165745	32.29	2.58
		5	126349	24.61	1.96
		6	106773	20.80	1.66
		7	100098	19.50	1.56
		8	101014	19.68	1.57

Table: The file *pic* with increasing bits per binary RLE encoded number.

Design - Unmodified compression

file	size original	$\frac{bits}{\text{RLE number}}$	size encoded	ratio in %	*bps*
pic	513216	2	350292	68.25	5.46
		3	235067	45.80	3.66
		4	165745	32.29	2.58
		5	126349	24.61	1.96
		6	106773	20.80	1.66
		7	100098	19.50	1.56
		8	101014	19.68	1.57

Table: The file *pic* with increasing bits per binary RLE encoded number.

- Byte-wise RLE achieves 27.2% of its original size using 2.17 *bps*.

Design - Preprocessing

- Most files other than pallet based images do not contain long runs of identical bit values.

Design - Preprocessing

- Most files other than pallet based images do not contain long runs of identical bit values.
- Files with long runs work really well with RLE.

Design - Preprocessing

- Most files other than pallet based images do not contain long runs of identical bit values.
- Files with long runs work really well with RLE.
- Artificially creating runs on arbitrary data will improve the performance of RLE.

Preprocessing

- Vertical interpretation of the input

Preprocessing

- Vertical interpretation of the input
- Dynamic byte remapping

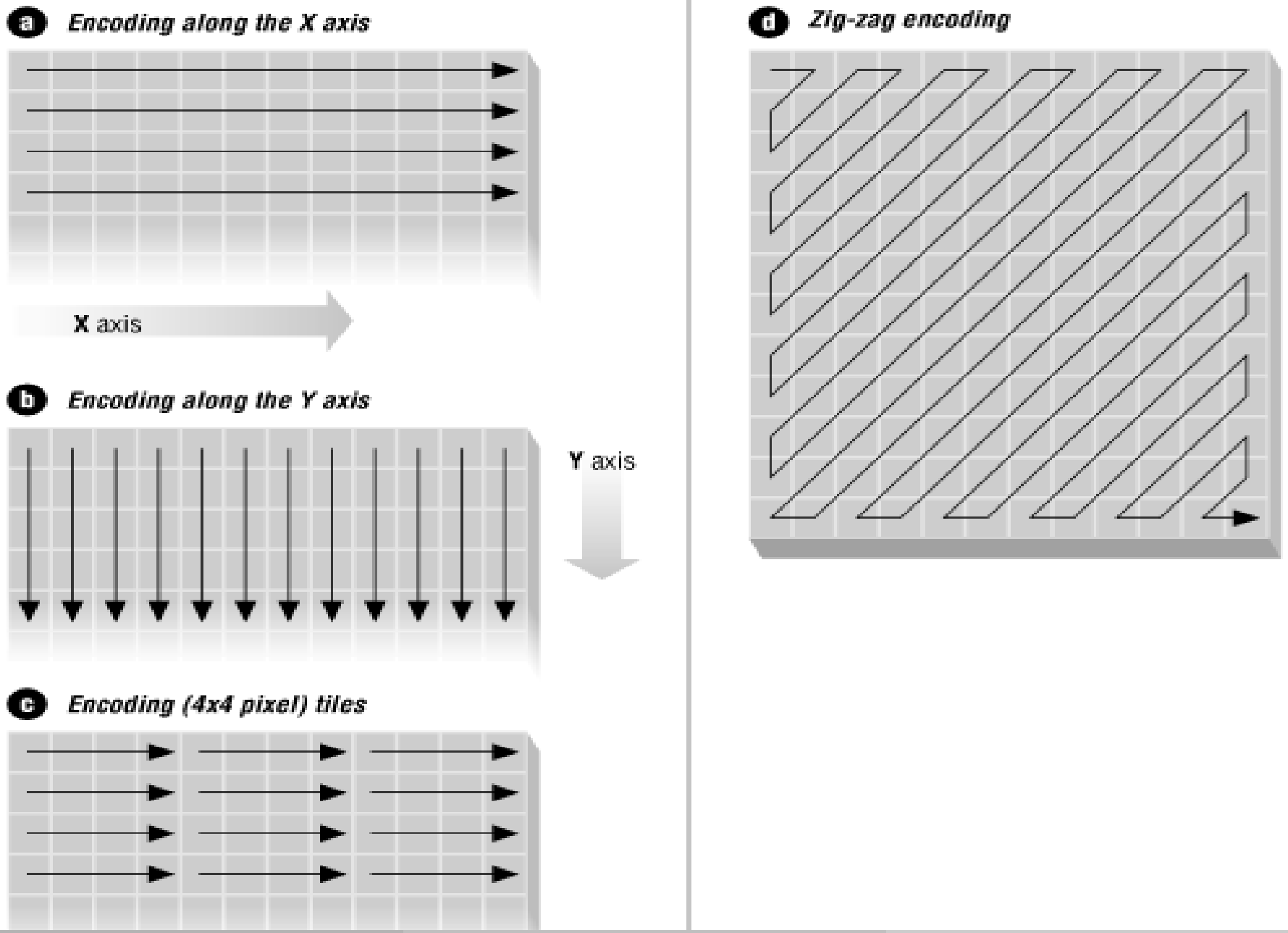
Preprocessing

- Vertical interpretation of the input
- Dynamic byte remapping
- Burrows-Wheeler-Transformation

Preprocessing

- Vertical interpretation of the input
- Dynamic byte remapping
- Burrows-Wheeler-Transformation
- Huffman Encoding of runs

Preprocessing - Vertical interpretation



Preprocessing - Vertical interpretation

TODO: show 1d and 2d data

Preprocessing - Vertical interpretation

bits per rle number	ratio in %	*bps*
8	255.22	20.41
7	224.45	17.95
6	194.74	15.57
5	167.04	13.36
4	142.58	11.40
3	127.80	10.22
2	139.79	11.18

Table: Binary RLE on vertical interpreted data, fixed run lengths.

Preprocessing - Vertical interpretation

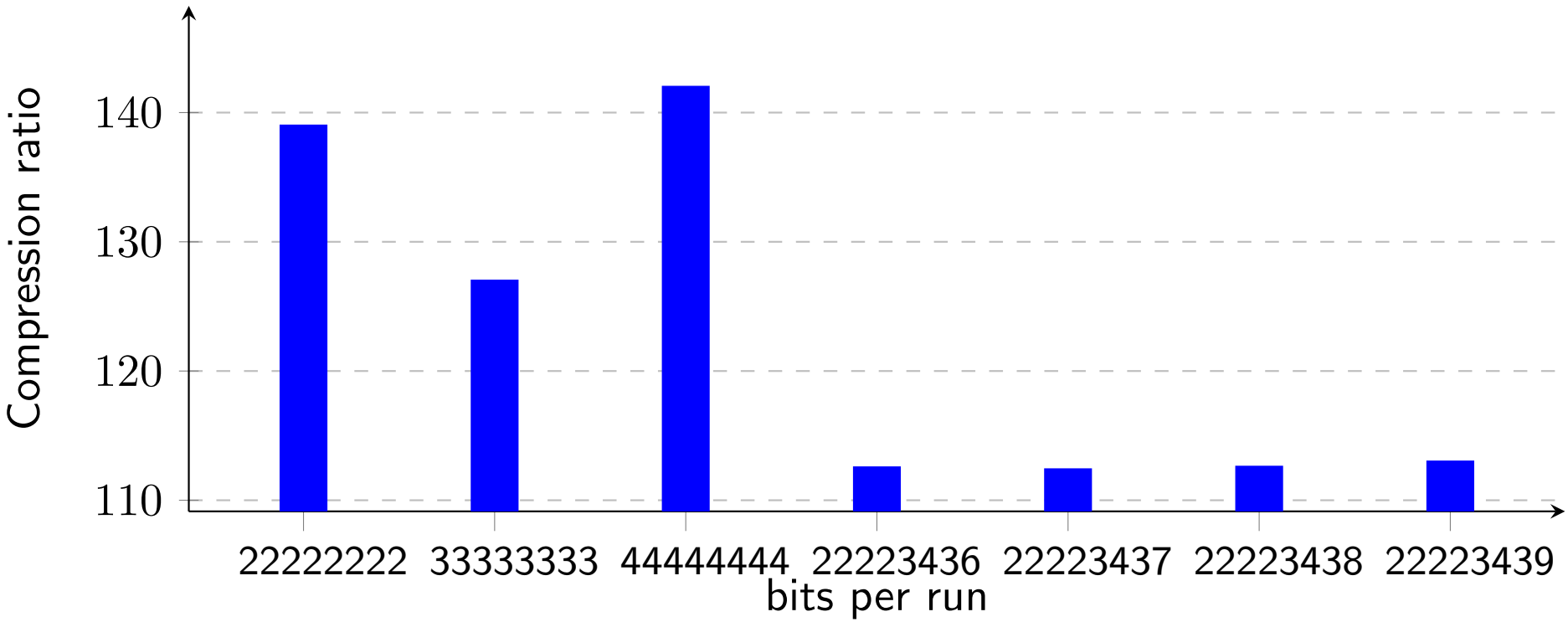


Figure: Byte mapping and varying maximum run lengths.

Preprocessing - Vertical interpretation

file	size original	size encoded	ratio in %	*bps*
bib	111261	129424	116.32	9.31
book1	768771	820463	106.72	8.54
book2	610856	659811	108.01	8.64
geo	102400	162274	158.47	12.68
news	377109	400810	106.28	8.50
obj1	21504	31592	146.91	11.75
obj2	246814	379591	153.80	12.30
paper1	53161	57654	108.45	8.68
paper2	82199	88121	107.20	8.58
pic	513216	533254	103.90	8.31
progc	39611	41360	104.42	8.35
progl	71646	74554	104.06	8.32
progp	49379	53403	108.15	8.65
trans	93695	99818	106.54	8.52
all files	3145718	3536225	112.41	8.99

Table: Calgary Corpus encoded, vertical encoding, using bits per run (2, 2, 2, 2, 3, 4, 3, 7).

Preprocessing - Byte Remapping

TODO: show byte remapping

Preprocessing - Byte Remapping

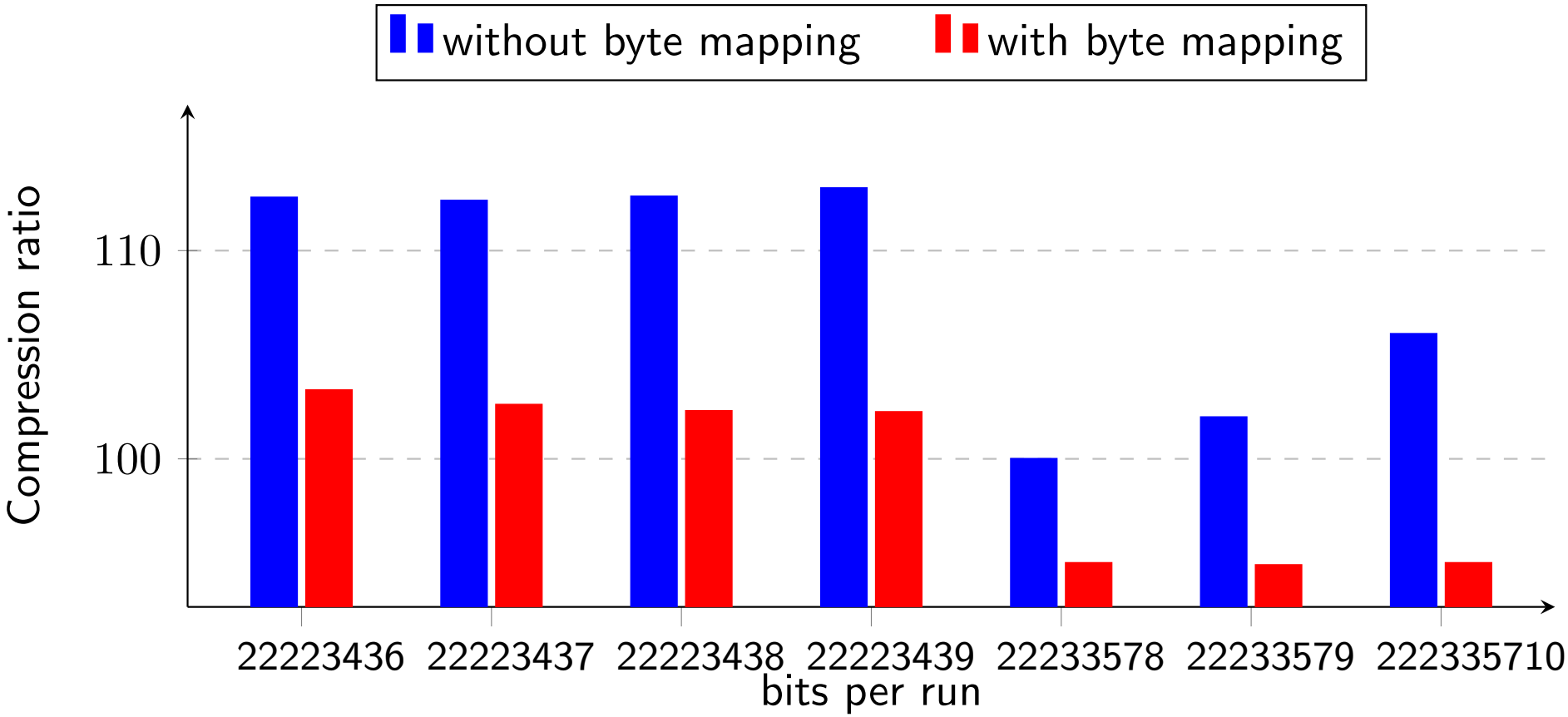


Figure: Byte mapping and varying maximum run lengths.

Preprocessing - Byte Remapping

file	size original	size encoded	ratio in %	*bps*
bib	111261	111579	100.29	8.02
book1	768771	669578	87.10	6.97
book2	610856	551757	90.33	7.23
geo	102400	144974	141.58	11.33
news	377109	363010	96.26	7.70
obj1	21504	30166	140.28	11.22
obj2	246814	340165	137.82	11.03
paper1	53161	50074	94.19	7.54
paper2	82199	71747	87.28	6.98
pic	513216	408136	79.53	6.36
progc	39611	38490	97.17	7.77
progl	71646	63765	89.00	7.12
progp	49379	46093	93.35	7.47
trans	93695	94729	101.10	8.09
all files	3145718	2988359	94.99	7.59

Table: Calgary Corpus encoded with vertical reading, byte remapping, using bits per run (2, 2, 3, 3, 3, 4, 5, 8).

Preprocessing - Burrows-Wheeler-Transformation

TODO

Preprocessing - Burrows-Wheeler-Transformation

$$S = abcabr$$

row 1	a	b	c	a	b	r
row 2	r	a	b	c	a	b
row 3	b	r	a	b	c	a
row 4	a	b	r	a	b	c
row 5	c	a	b	r	a	b
row 6	b	c	a	b	r	a

Table: Burrows Wheeler Transformation Matrix (all cyclic rotations).

Preprocessing - Burrows-Wheeler-Transformation

$$S = abcabr$$

row 1	a	b	c	a	b	r
row 2	a	b	r	a	b	c
row 3	b	c	a	b	r	a
row 4	b	r	a	b	c	a
row 5	c	a	b	r	a	b
row 6	r	a	b	c	a	b

Table: Burrows Wheeler Transformation Matrix (all cyclic rotations, sorted).

Preprocessing - Burrows-Wheeler-Transformation

$$S = abcabr$$

row 1	a	b	c	a	b	r
row 2	a	b	r	a	b	c
row 3	b	c	a	b	r	a
row 4	b	r	a	b	c	a
row 5	c	a	b	r	a	b
row 6	r	a	b	c	a	b

Table: Burrows Wheeler Transformation Matrix (all cyclic rotations, sorted).

$$L = rcaabb$$

$$i = 1$$

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

Preprocessing - inverse Burrows-Wheeler-Transformation

$$L = rcaabb$$

$$i = 1$$

word	word with position	sorted
r	(r,1)	(a,3)
c	(c,2)	(a,4)
a	(a,3)	(b,5)
a	(a,4)	(b,6)
b	(b,5)	(c,2)
b	(b,6)	(r,1)

Table: Standard permutation generation of the word L.

Preprocessing - inverse Burrows-Wheeler-Transformation

This yields a standard permutation of:

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 2 & 1 \end{pmatrix} \tag{1}$$

Preprocessing - inverse Burrows-Wheeler-Transformation

This yields a standard permutation of:

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 2 & 1 \end{pmatrix} \tag{1}$$

Following $\pi_L^1(1)$ to $\pi_L^6(1)$:

$$3 \xrightarrow{\pi_L} 5 \xrightarrow{\pi_L} 2 \xrightarrow{\pi_L} 4 \xrightarrow{\pi_L} 6 \xrightarrow{\pi_L} 1 \tag{2}$$

Preprocessing - inverse Burrows-Wheeler-Transformation

This yields a standard permutation of:

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 2 & 1 \end{pmatrix} \tag{1}$$

Following $\pi_L^1(1)$ to $\pi_L^6(1)$:

$$3 \xrightarrow{\pi_L} 5 \xrightarrow{\pi_L} 2 \xrightarrow{\pi_L} 4 \xrightarrow{\pi_L} 6 \xrightarrow{\pi_L} 1 \tag{2}$$

Applying the sequence to the labeling function of the word L:

$$\lambda_L(3)\,\lambda_L(5)\,\lambda_L(2)\,\lambda_L(4)\,\lambda_L(6)\,\lambda_L(1) = abcabr = w \tag{3}$$

Preprocessing - Burrows-Wheeler-Transformation

bits per rle number	ratio in %	*bps*
3	95.41	7.63
2	91.39	7.31

Table: Initial BWT implementation on byte wise RLE.

Preprocessing - Burrows-Wheeler-Transformation

bits per rle number	ratio in %	*bps*
3	95.41	7.63
2	91.39	7.31

Table: Initial BWT implementation on byte wise RLE.

bits per rle number	ratio in %	*bps*
3	91.62	7.33
2	89.46	7.15

Table: Burrows Wheeler Transformation on byte wise RLE.

Preprocessing - Burrows-Wheeler-Transformation

bits per rle number	ratio in %	*bps*
8	74.42	5.95
7	69.90	5.59
6	65.58	5.24
5	61.71	4.93
4	58.98	4.71
3	59.18	4.73
2	67.69	5.41

Table: Modified Burrows Wheeler Transformation on byte wise RLE.

Preprocessing - Burrows-Wheeler-Transformation

file	size original	size encoded	compression	*bps*
bib	111261	59285	53.28	4.26
book1	768771	590879	76.86	6.15
book2	610856	374742	61.35	4.91
geo	102400	101192	98.82	7.91
news	377109	246047	65.25	5.22
obj1	21504	16467	76.58	6.13
obj2	246814	126626	51.30	4.10
paper1	53161	34130	64.20	5.14
paper2	82199	56507	68.74	5.50
pic	513216	136074	26.51	2.12
progc	39611	24312	61.38	4.91
progl	71646	31466	43.92	3.51
progp	49379	20862	42.25	3.38
trans	93695	32835	35.04	2.80
all files	3145718	1855520	58.98	4.71

Table: Calgary Corpus encoded with byte wise RLE after a Burrows-Wheeler-Transformation with 4 bit per run.

Preprocessing - Burrows-Wheeler-Transformation

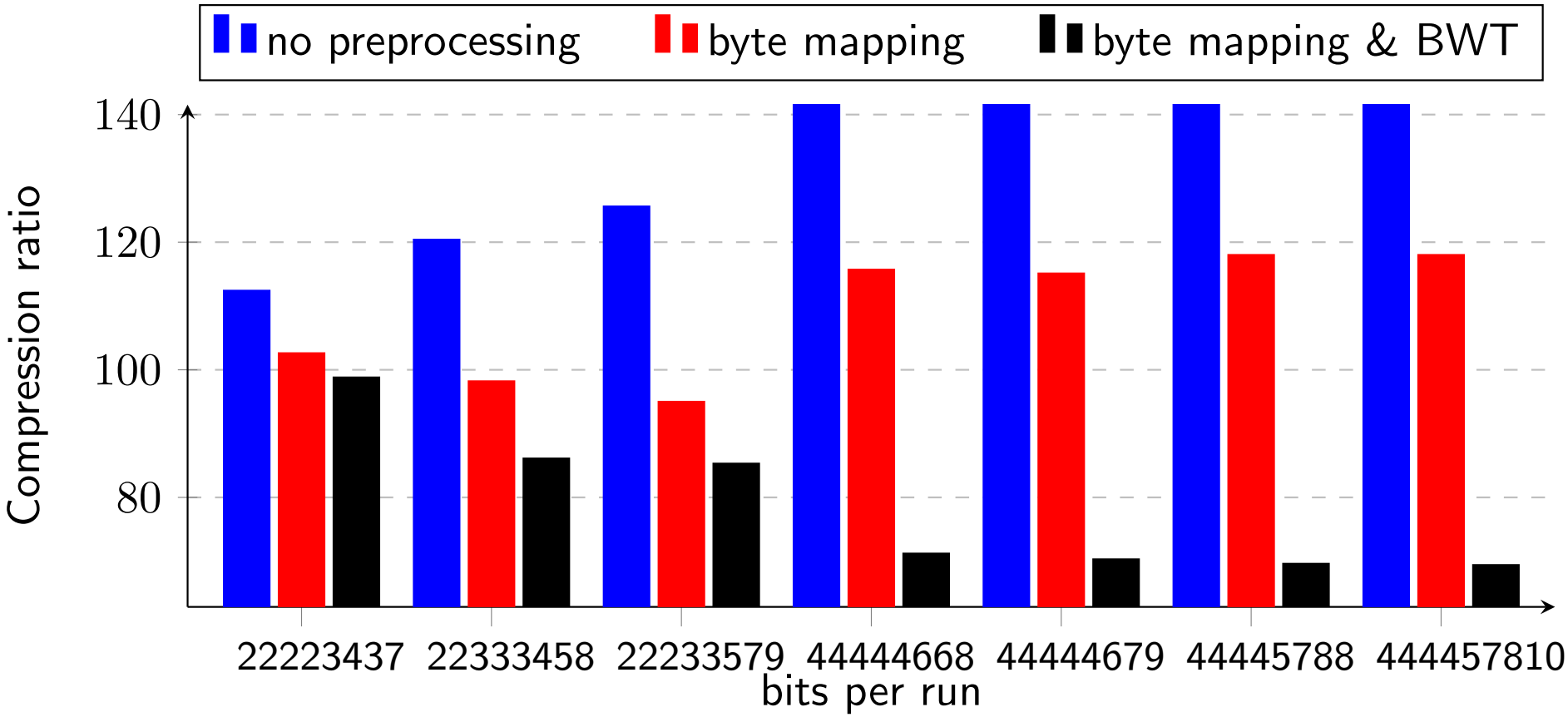


Figure: Byte mapping and varying maximum run lengths, all preprocessing steps.

Preprocessing - Burrows-Wheeler-Transformation

file	size original	size encoded	ratio in %	*bps*
bib	111261	73843	66.37	5.31
book1	768771	570348	74.19	5.94
book2	610856	409639	67.06	5.36
geo	102400	145950	142.53	11.40
news	377109	275396	73.03	5.84
obj1	21504	27023	125.66	10.05
obj2	246814	213392	86.46	6.92
paper1	53161	37344	70.25	5.62
paper2	82199	56490	68.72	5.50
pic	513216	227914	44.41	3.55
progc	39611	28275	71.38	5.71
progl	71646	38144	53.24	4.26
progp	49379	27029	54.74	4.38
trans	93695	49314	52.63	4.21
all files	3145718	2184197	69.43	5.55

Table: Calgary Corpus encoded, byte mapping and a BWTS as preprocessing, using bits per run (4, 4, 4, 4, 5, 7, 8, 10).

Preprocessing - Huffman Encoding RLE runs

Implementation

Evaluation and Discussion