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

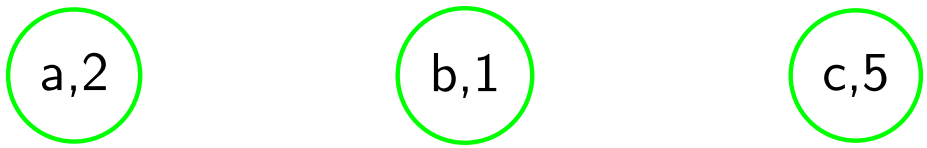


Figure: Example Huffman tree with 3 leaf nodes.

Huffman Encoding

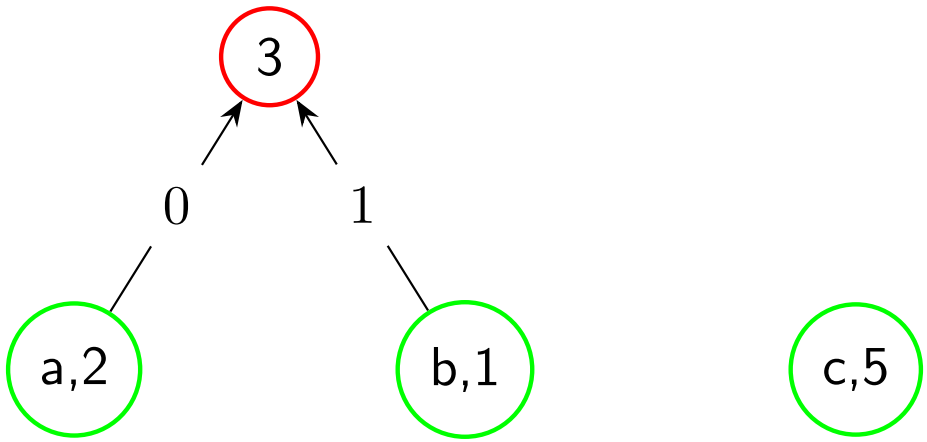


Figure: Example Huffman tree with 3 leaf nodes.

Huffman Encoding

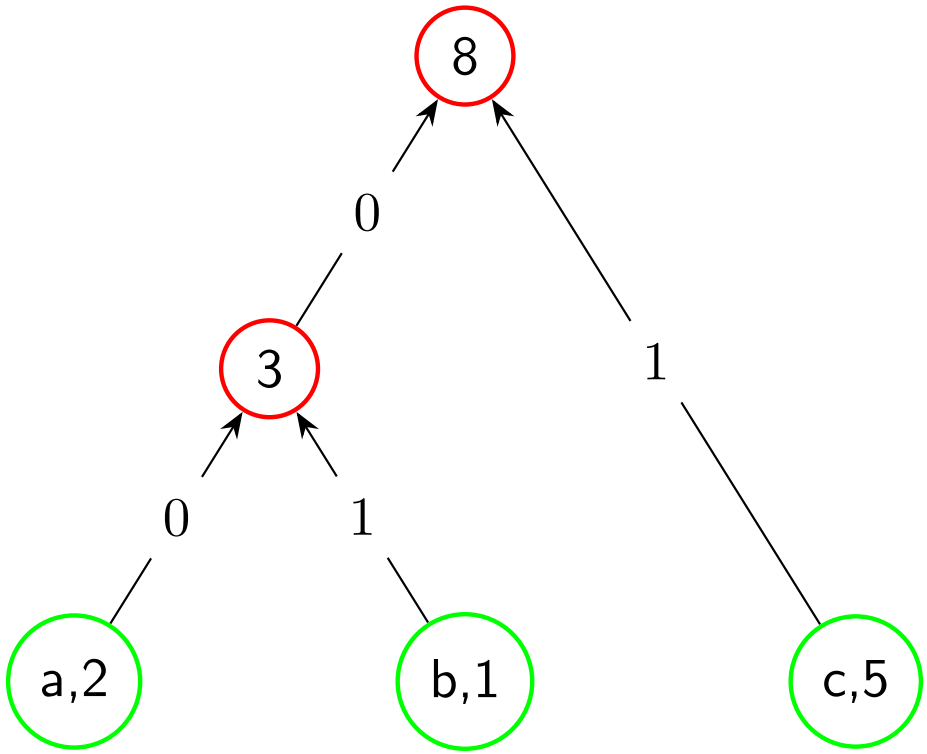


Figure: Example Huffman tree with 3 leaf nodes.

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 -m×10	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

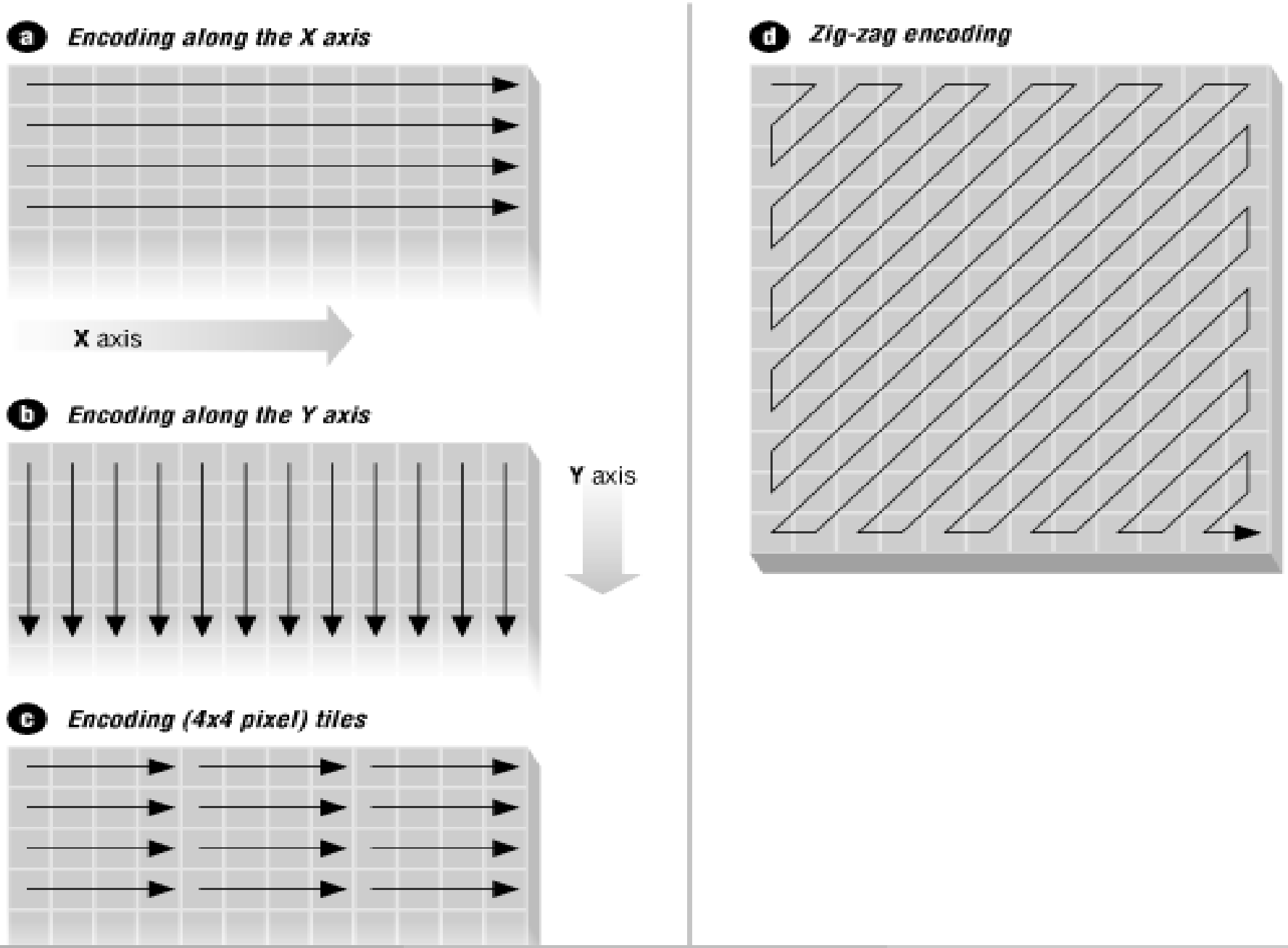
Preprocessing

- 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

TODO: show byte remapping

Preprocessing - Burrows-Wheeler-Transformation

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

Preprocessing - Burrows-Wheeler-Transformation

Implementation

Evaluation and Discussion