Improving Run Length Encoding by Preprocessing

Sven Fiergolla[*], Petra Wolf[*]

DCC 2021

[*]Universität Trier - Germany

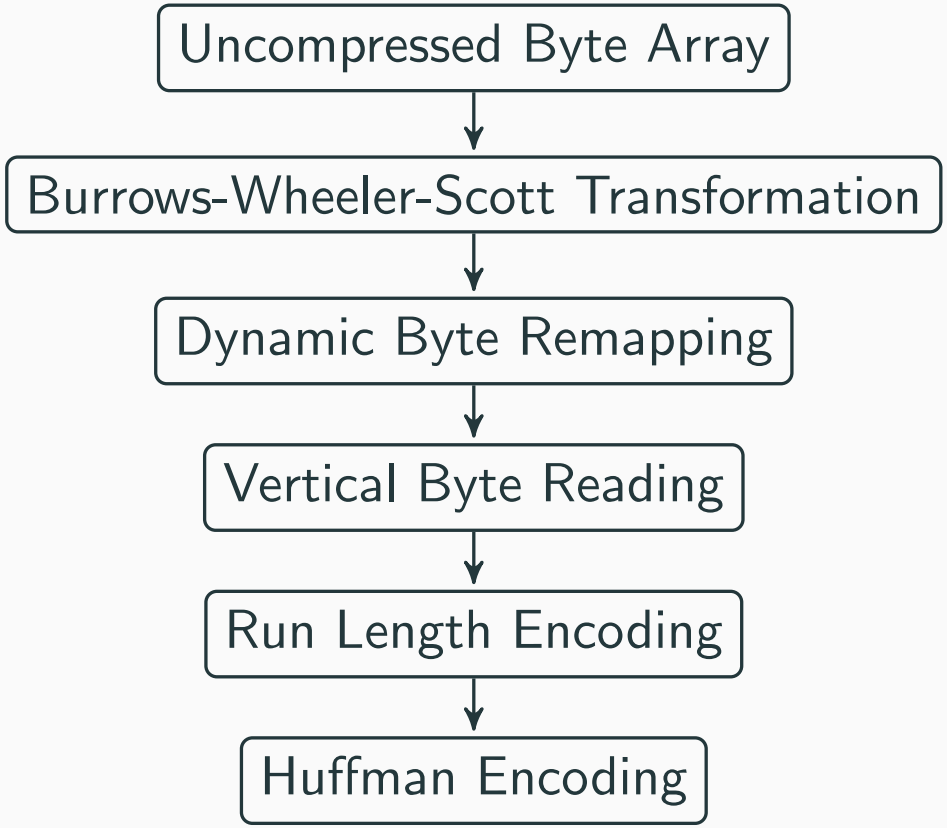
Main Idea

Problem: RLE on binary strings $\Rightarrow$ no long runs

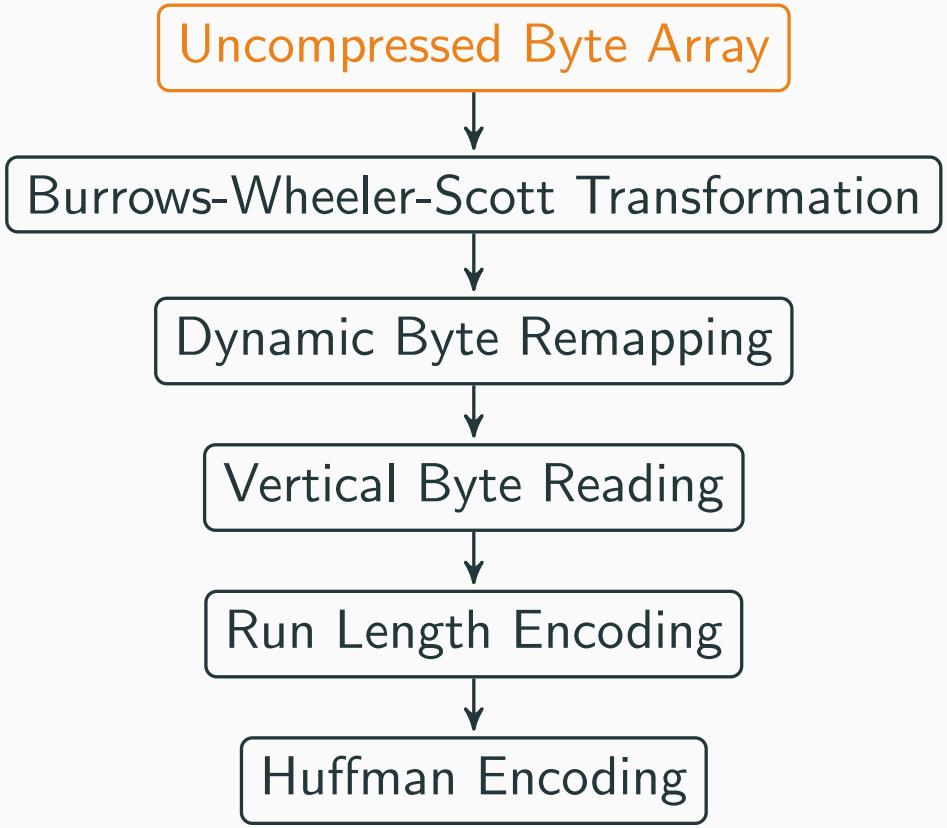
Idea: Read string vertically: First, all most significant bits, then second most significant etc.

Proposed Technique: Vertical byte reading combined with several preprocessing steps encoded via RLE.

Proposed Algorithm



Proposed Algorithm

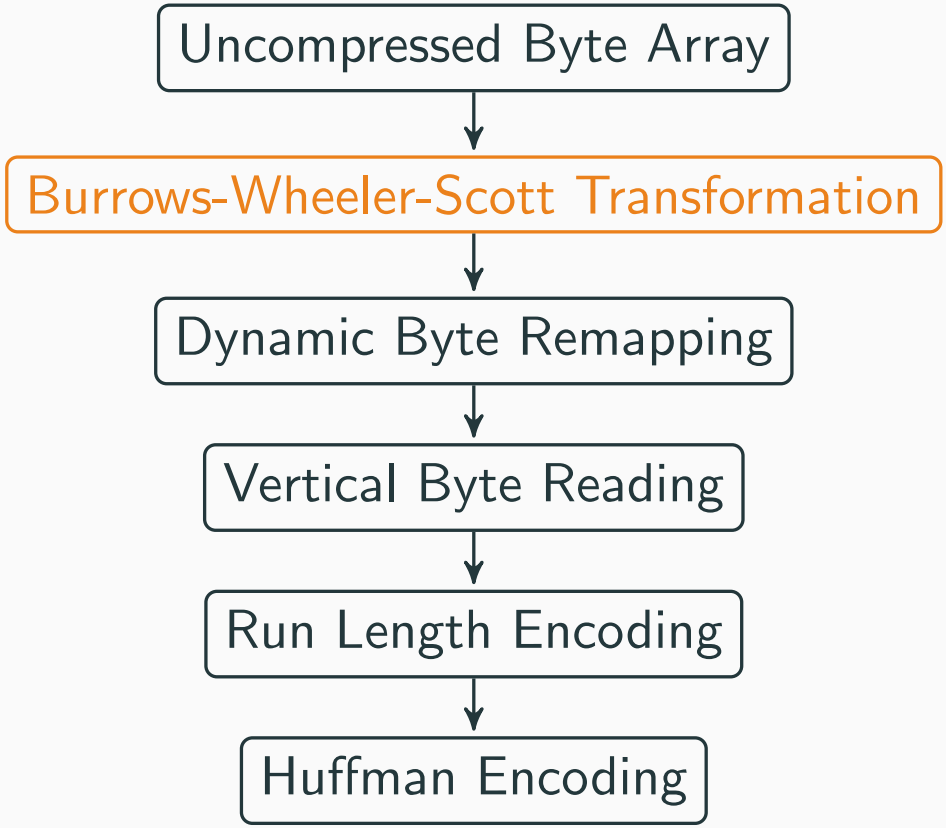


Example

Input string *abraca*

01100001 01100010 01110010 01100001 01100011 01100001

Proposed Algorithm

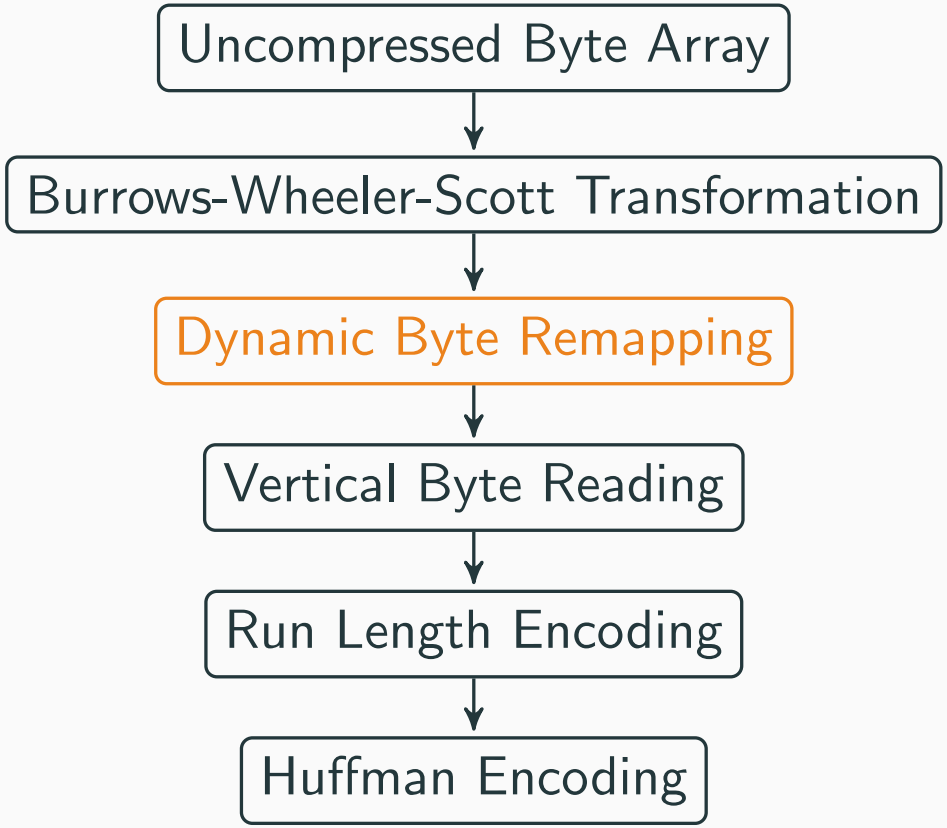


Example

caraab

01100011 01100010 01110010 01100001 01100001 01100001

Proposed Algorithm

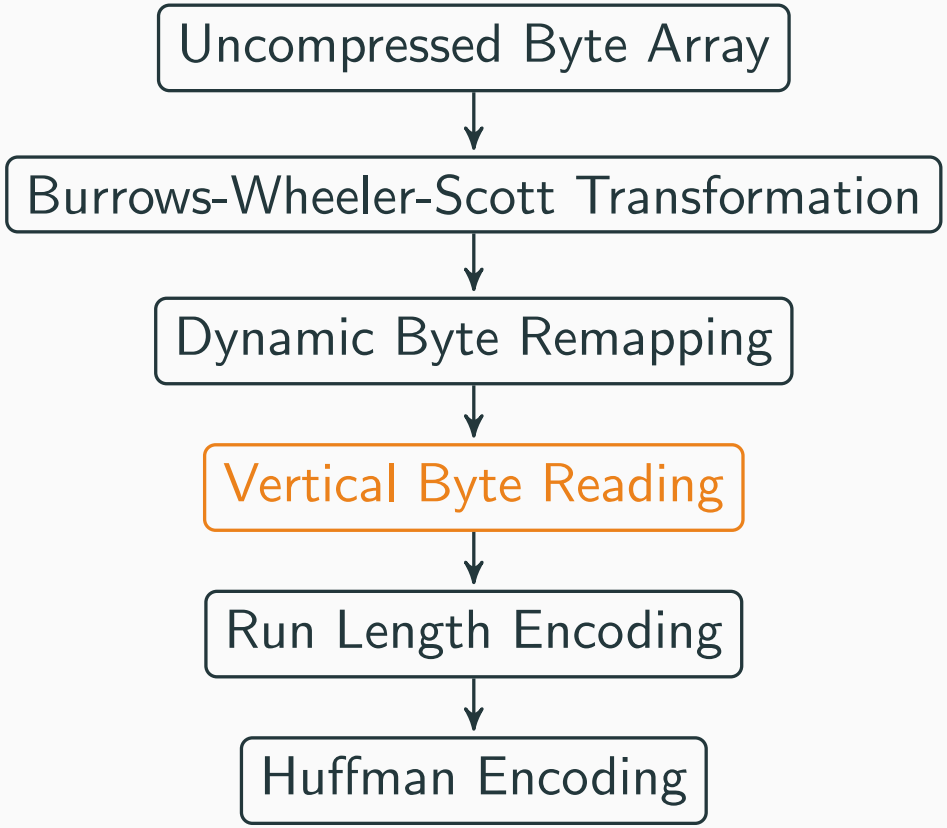


Example

$a \mapsto 00000000, b \mapsto 0000001, c \mapsto 00000010, r \mapsto 0000011$

00000010 00000000 00000011 00000000 00000000 00000001

Proposed Algorithm



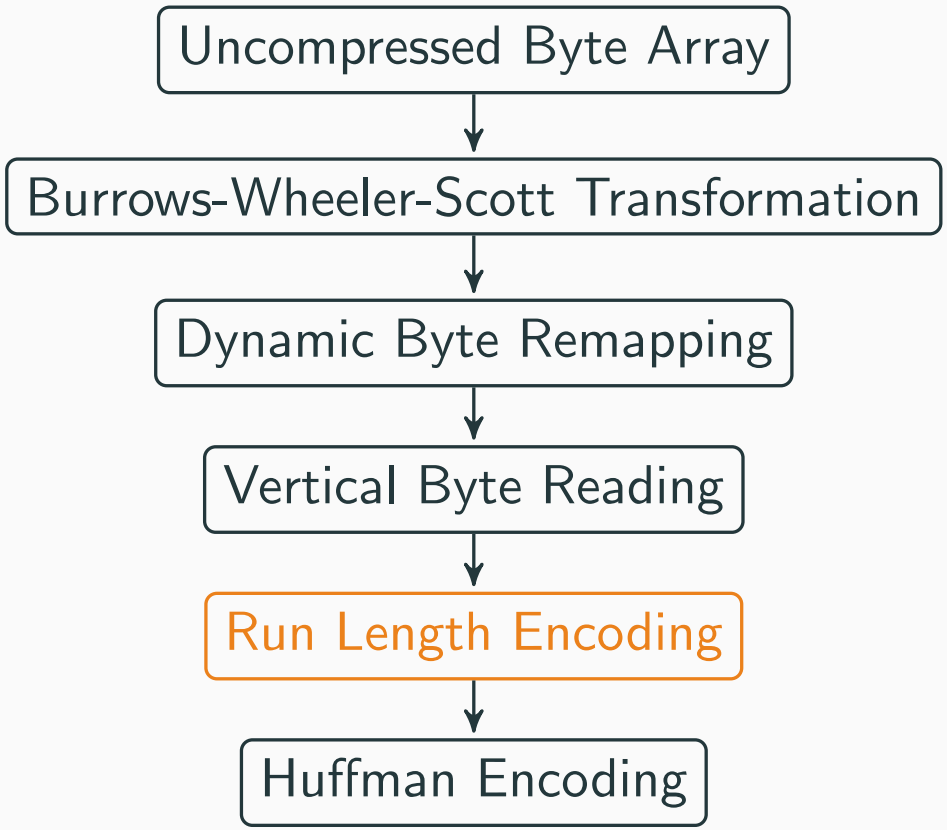
00000010
00000000
00000011
00000000
00000000
00000001

Example

00000010 00000000 00000011 00000000 00000000 00000001

000000 000000 000000 000000 000000 000000 101000 001001

Proposed Algorithm

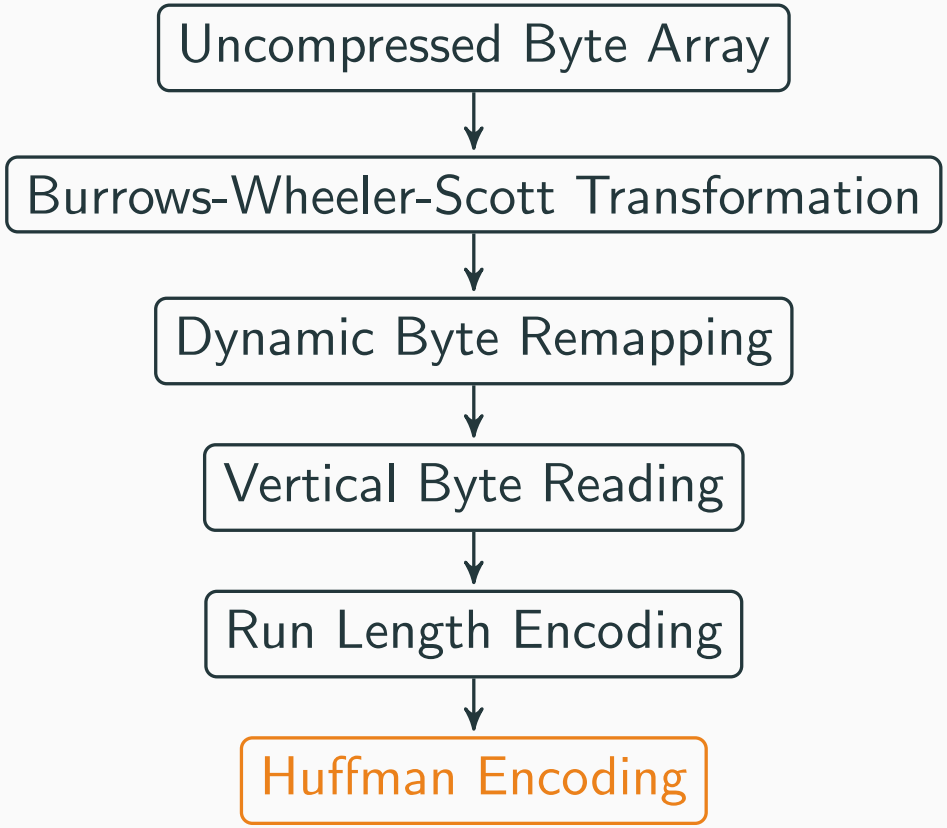


Example

$\langle 36, 1, 1, 1, 5, 1, 2, 1 \rangle$

$\underbrace{000000\ 000000\ 000000\ 000000\ 000000\ 000000}_{36}\ \overset{\uparrow}{1}\overset{\uparrow}{0}\overset{\uparrow}{1}\overset{\uparrow}{0}\underbrace{00\ 0}_{5}\overset{\uparrow}{1}\underbrace{\ 00}_{2}\ \overset{\uparrow}{1}$

Proposed Algorithm



Example

$1 \mapsto 0, 2 \mapsto 10, 5 \mapsto 110, 36 \mapsto 111$

111 0 0 0 110 0 10 0

Evaluation: Comparison with RLE

file	original size [kB]	RLE s. [kB]	RLE r.s. [%]	RLE [*bps*]	proposed algorithm s. [kB]	proposed algorithm r.s. [%]	proposed algorithm [*bps*]	impr. [%]
alice29.txt	152.1	604.9	397.70	31.82	65.4	43.00	3.44	**89.19**
asyoulik.txt	125.2	514.8	411.18	32.90	59.2	47.28	3.79	**88.50**
cp.html	24.6	98.9	402.03	32.16	11.0	44.72	3.60	**88.88**
fields.c	11.2	44.6	398.21	32.01	5.1	45.54	3.72	**88.57**
grammar.lsp	3.7	14.8	400.00	31.89	1.9	51.35	4.13	**87.16**
kennedy.xls	1029.8	1820.3	176.76	14.14	229.8	22.32	1.79	**87.38**
lcet10.txt	426.8	1749.7	409,96	32.80	170.5	39.95	3.20	**90.26**
plrabn12.txt	481.9	1944.9	403.59	32.29	215.6	44.74	3.58	**88.92**
ptt5	513.2	136.6	26.62	2.12	82.1	16.00	1.28	**39.90**
sum	38.2	99.4	260.21	20.80	19.6	51.31	4.10	**80.28**
xargs.1	4.2	17.7	421.43	33.50	2.5	59.52	4.76	**85.88**
all files	2811.9	7046.6	250.60	20.05	862.7	30.68	2.45	**87.76**
∅ values per file		-	337.06	26,95	-	42.34	3.40	**83.18**

Files from Canterbury Corpus; absolute file size in kB after compression, relative file size in %, defined as (size after compression)/(original size), and bits per symbol (bps).

Evaluation: Comparison with ZIP

file	original	ZIP			proposed algorithm			impr.
	size [kB]	size [kB]	r.s. [%]	[*bps*]	size [kB]	r.s. [%]	[*bps*]	[%]
dickens	9956	3780	37.96	3.03	3964	39.82	3.19	-4.87
mozilla	50020	18604	37.19	2.98	24808	49.60	3.97	-33.35
mr	9740	3608	37.04	2.96	2908	29.86	2.39	**19.40**
nci	32768	3128	9.55	0.76	2900	8.85	0.71	**7.29**
ooffice	6008	3028	50.40	4.03	3904	64.98	5.20	-28.93
osdb	9852	3656	37.11	2.97	3260	33.09	2.65	**10.83**
reymont	6472	1816	28.06	2.25	1960	30.28	2.42	-7.93
samba	21100	5336	25.29	2.02	7012	33.23	2.66	-31.41
sao	7084	5208	73.52	5.88	5604	79.10	6.33	-7.60
webster	40488	11920	29.44	2.36	12012	29.67	2.37	-0.77
xml	5220	676	12.95	1.04	868	16.63	1.33	-28.40
xray	8276	5900	71.29	5.70	4840	58.48	4.68	**17.97**
all files	206984	66660	32.21	2.58	74040	35.77	2.86	-11.07
∅ values per file		-	37.48	3.00	-	39.47	3.16	-7.31

Files from Silesia Corpus; absolute file size in kB after compression, relative file size in %, defined as (size after compression)/(original size), and bits per symbol (bps).

Thank You!

Improving Run Length Encoding by Preprocessing

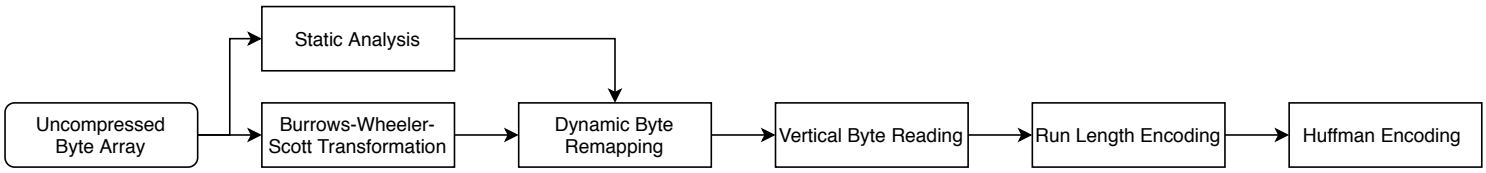
Sven Fiergolla and Petra Wolf

University Trier
Universitätsring 15
54296 Trier, Germany
{fiergolla,wolfp}@informatik.uni-trier.de

Full version

Implementation

Schematic Illustration of the Proposed Algorithm



Main Idea

Problem of binary RLE Does not contain long runs of repeating bits.

Idea of improvement First, reading all most significant bits of all bytes, then all second most significant bits and so on, $\Rightarrow$ much longer average runs.

Obtain algorithm Combine vertical byte reading with several preprocessing steps to obtain RLE based compression algorithm comparable to ZIP.

In the first step, the uncompressed byte array is analyzed and for each byte its number of occurrences is counted. In parallel, a bijective Burrows-Wheeler-Scott Transform is applied to the input byte array, which produces a reversible permutation of the input byte array with long repetitions of similar symbols. Afterwards, each byte is remapped, where the most frequent byte values are mapped to the lowest binary values. The resulting byte array is then interpreted in a specific way, called vertical byte reading, at first all most significant bits of all bytes are read, then all second most significant bits and so on, resulting in long average runs of identical bits. On this representation, a run length encoding (RLE) is applied and the runs are counted to generate a Huffman tree. Using this, the runs are output with a variable length code, together with the mappings needed to decompress the file.

Example

- **Input String**: *abraca*
  Binary representation:
  01100001 01100010 01110010 01100001 01100011 01100001
- **Burrows-Wheeler-Scott Transformation** yields *caraab*
  Binary representation:
  01100011 01100010 01110010 01100001 01100001 01100001
- **Dynamic Byte Remapping**:
  $a \mapsto 00000000$, $b \mapsto 0000001$, $c \mapsto 00000010$, $r \mapsto 0000011$
  00000010 00000000 00000011 00000000 00000000 00000001
- **Vertical Byte Reading**:
  00000010 00000000 00000011 00000000 00000000 00000001

  00000010
  00000000
  00000011
  00000000
  00000000
  00000001

  000000 000000 000000 000000 000000 000000 101000 001001
- **Run Length Encoding**: $\langle 36, 1, 1, 1, 5, 1, 2, 1 \rangle$
  000000 000000 000000 000000 000000 000000 101000 001 00 1
  36 1 1 1 5 1 2 1
- **Huffman Encoding** of RLE runs:
  $1 \mapsto 0, 2 \mapsto 10, 5 \mapsto 110, 36 \mapsto 111$
  111 0 0 0 110 0 10 0

Evaluation: Comparison with ZIP

file	original size [kB]	ZIP size [kB]	ZIP r.s. [%]	ZIP [bps]	proposed algorithm size [kB]	proposed algorithm r.s. [%]	proposed algorithm [bps]	impr. [%]
dickens	9956	3780	37.96	3.03	3964	39.82	3.19	-4.87
mozilla	50020	18604	37.19	2.98	24808	49.60	3.97	-33.35
mr	9740	3608	37.04	2.96	2908	29.86	2.39	**19.40**
nci	32768	3128	9.55	0.76	2900	8.85	0.71	**7.29**
ooffice	6008	3028	50.40	4.03	3904	64.98	5.20	-28.93
osdb	9852	3656	37.11	2.97	3260	33.09	2.65	**10.83**
reymont	6472	1816	28.06	2.25	1960	30.28	2.42	-7.93
samba	21100	5336	25.29	2.02	7012	33.23	2.66	-31.41
sao	7084	5208	73.52	5.88	5604	79.10	6.33	-7.60
webster	40488	11920	29.44	2.36	12012	29.67	2.37	-0.77
xml	5220	676	12.95	1.04	868	16.63	1.33	-28.40
xray	8276	5900	71.29	5.70	4840	58.48	4.68	**17.97**
all files	206984	66660	32.21	2.58	74040	35.77	2.86	-11.07
∅ values per file		-	37.48	3.00	-	39.47	3.16	-7.31

The Silesia Corpus encoded with ZIP v3.0 (cmd: `zip -evr`) and the proposed algorithm. For each method, absolute file size in kB after compression, relative file size in %, defined as (size after compression)/(original size), and bps are listed. The last column shows the improvement (bold if > 0) of the proposed algorithm over ZIP in % as $1-$ (size proposed algorithm)/(size ZIP) in %.

file type	ZIP ∅ rel. size [%]	ZIP ∅ [bps]	proposed algorithm ∅ rel. size [%]	proposed algorithm ∅ [bps]	improvement [%]
PGM	76.77	6.14	70.54	5.64	**8.11**
PPM	76.37	6.11	70.13	5.61	**8.17**
DNG	87.09	6.96	85.12	6.80	**2.26**
OBJ	26.92	2.15	36.40	2.91	-35.21
STL	43.93	3.51	64.30	5.14	-46.36
PLY	33.88	2.71	41.87	3.34	-23.58

Average relative file size after compression of a random selection of files of different file types compressed with ZIP v3.0 in comparison with the proposed algorithm.

Evaluation: Comparison with RLE

file	original size [kB]	RLE s. [kB]	RLE r.s. [%]	RLE [bps]	proposed algorithm s. [kB]	proposed algorithm r.s. [%]	proposed algorithm [bps]	impr. [%]
alice29.txt	152.1	604.9	397.70	31.82	65.4	43.00	3.44	**89.19**
asyoulik.txt	125.2	514.8	411.18	32.90	59.2	47.28	3.79	**88.50**
cp.html	24.6	98.9	402.03	32.16	11.0	44.72	3.60	**88.88**
fields.c	11.2	44.6	398.21	32.01	5.1	45.54	3.72	**88.57**
grammar.lsp	3.7	14.8	400.00	31.89	1.9	51.35	4.13	**87.16**
kennedy.xls	1029.8	1820.3	176.76	14.14	229.8	22.32	1.79	**87.38**
lcet10.txt	426.8	1749.7	409,96	32.80	170.5	39.95	3.20	**90.26**
plrabn12.txt	481.9	1944.9	403.59	32.29	215.6	44.74	3.58	**88.92**
ptt5	513.2	136.6	26.62	2.12	82.1	16.00	1.28	**39.90**
sum	38.2	99.4	260.21	20.80	19.6	51.31	4.10	**80.28**
xargs.1	4.2	17.7	421.43	33.50	2.5	59.52	4.76	**85.88**
all files	2811.9	7046.6	250.60	20.05	862.7	30.68	2.45	**87.76**
∅ values per file		-	337.06	26,95	-	42.34	3.40	**83.18**

The Canterbury Corpus encoded with RLE and the proposed algorithm.
For each method, absolute file size in kB after compression, relative file size in %, defined as (size after compression)/(original size), and bps are listed. The last column shows the improvement of the proposed algorithm over RLE in % as $1-$ (size proposed algorithm)/(size RLE).