A Fast Block–Sorting Algorithm

Dipl-Ing. Michael Schindler

hm for Lossless Data Compression

Vienna University of Technology
Karlsplatz 13/1961, A–1040 Wien, Austria

Symbol		Context BWT	Context Order 3	BWT	Sort 3	Context	
11	A	BRAC…	BRA$_{11}$	A$_1$	A$_1$	♦…	
10	B	RAC…	RAC$_{10}$	R$_2$	R$_2$	A♦…	
9	R	AC…	ACA$_9$	D$_5$	♦$_0$	ABRA♦	*
8	A	C…	CAD$_8$	♦$_0$	D$_5$	ABRAC…	*
7	C	AD…	ADA$_7$	R$_9$	R$_9$	AC…	
6	A	D…	DAB$_6$	C$_7$	C$_7$	AD…	
5	D	ABRA♦	ABR$_5$	A$_4$	A$_4$	BRA♦…	*
4	A	BRA♦…	BRA$_4$	A$_{11}$	A$_{11}$	BRAC…	*
3	B	RA♦…	RA♦$_3$	A$_8$	A$_8$	C…	
2	R	A♦…	A♦A$_2$	A$_6$	A$_6$	D…	
1	A	♦…	♦AB$_1$	B$_3$	B$_3$	RA♦…	
0	♦	ABRAC…	ABR$_0$	B$_{10}$	B$_{10}$	RAC…	

Advantages of **limited order sorting**

fast compression and decompression

memory efficient

suited for **hardware** implementation

methods for **structured data** available

The Transformation

The limited order sort transformation sorts contexts similar to the Burrows–Wheeler transformation. The difference is the sort criterion after a prefixed number of context characters compare equal. BWT continues to compare contexts, whereas this transformation uses the position in the input block to determine sort order. Due to this output can only differ from BWT if the limited contexts are equal (marked * in the table). This feature is exploited in the backtransformation, where the first pass uses the Burrows-Wheeler unsorting method to count the contexts. Loops and shortcuts are possible in this step, so care must be taken to backtransform the entire text.

Compression Implementation

Two methods were found to implement this compression. Both are very fast, but differ in memory requirement and complexity.

The first method is to directly sort pointers to the input buffer using radix sort. It is preferable to count the size of each bucket in a separate pass; these counts can then be used for each pass. The first pass sorts by the least significant context character(s); the last pass by the most significant context character(s). This method requires two pointers for each input character plus one pointer for each radix sort bucket.

The other method is to count how often each context appears. Those counts should be stored in a hash–table together with a pointer to the first occurrence of the context.
Then the hash–table is sorted using radix sort. With this sorted table start locations for each context are calculated.
A second pass over the to be compressed data puts the symbols in the correct place and increments the start location of their context.

This method requires one counter and two pointers for each context that appears in the data. The number of contexts is smaller than the number of symbols; additionally the blocksize can be adjusted if the hash–table fills early.

Decompression Implementation

Decompression is similar to the second compression method. First the contexts are counted (using a variant of the inverse BWT) and startpoints are calculated. The next pass then produces the output by looking up the next symbol at the startaddress for the current context, incrementing this startaddress upon usage. The memory requirement is the same as in the second compression method.

A critical step is the context counting; I suggest you fetch an implementation from ftp://eiunix.tuwien.ac.at/pub/michael/st/

Compression Performance (bit/byte):

	blocksort model		bzip	bred2	gzip
	BWT	new transformation (order 4)	BWT + P. Fenwick´s struct model	BWT + MTF + Huffmann (D. Wheeler)	LZ- based
bib	1.94	2.08	1.95	2.19	2.52
book1	2.34	2.45	2.40	2.98	3.26
book2	2.01	2.12	2.04	2.51	2.71
geo	4.41	4.30	4.48	4.89	5.35
news	2.46	2.57	2.51	2.94	3.07
obj1	3.83	3.78	3.87	3.91	3.84
obj2	2.42	2.53	2.46	2.67	2.65
paper1	2.47	2.55	2.46	2.58	2.79
paper2	2.41	2.48	2.42	2.58	2.90
pic	0.75	0.82	0.77	0.82	0.88
progc	2.51	2.54	2.50	2.58	2.68
progl	1.71	1.82	1.72	1.79	1.82
progp	1.73	1.88	1.71	1.78	1.82
trans	1.49	1.69	1.50	1.56	1.62
Average	2.32	2.40	2.34	2.56	2.71
Overall	2.06	2.15	2.09	2.44	2.61

Comparing the two colums that use my blocksort model gives an estimation of the loss caused by using the new transformation.

Comparing bzip and BWT with my blocksort model shows the performance of this blocksort model.

bred2 and gzip are given as reference values.

Compression Speed (sec on HP9000/705):

	raw size	new trans-formation		blocksort model		bzip		gzip		bred2	
		hash	radix	comp	decomp	comp	decomp	comp	decomp	comp	decomp
bib	111261	3.2	0.6	1.1	1.5	3.4	2.0	1.3	0.2	1.3	0.5
book1	768771	17.8	5.6	10.4	13.9	36.4	17.7	12.3	1.2	8.0	3.9
book2	610856	11.0	4.0	6.9	9.3	25.0	12.4	7.5	0.9	6.3	2.8
geo	102400	6.2	0.8	1.9	2.6	4.5	3.3	2.9	0.2	2.2	1.2
news	377109	11.1	2.5	4.6	6.2	13.7	8.0	4.0	0.6	4.2	1.9
obj1	21504	1.1	0.2	0.3	0.4	0.8	0.5	0.1	0.0	0.4	0.1
obj2	246814	6.7	1.7	2.7	3.6	7.9	4.7	2.8	0.3	3.2	1.3
paper1	53161	1.5	0.3	0.6	0.8	1.6	0.9	0.5	0.0	0.6	0.3
paper2	82199	2.3	0.5	1.3	1.4	2.6	1.7	1.0	0.1	0.9	0.3
pic	513216	6.8	2.1	3.5	3.6	4.8	3.3	2.9	0.5	1.6	1.1
progc	39611	1.2	0.3	0.6	0.7	1.2	0.8	0.3	0.0	0.4	0.2
progl	71646	1.5	0.3	0.7	0.9	1.9	1.0	0.5	0.0	0.8	0.3
progp	49379	1.4	0.3	0.5	0.6	1.4	0.7	0.4	0.0	0.8	0.2
trans	93695	2.2	0.5	0.9	1.1	2.5	1.3	0.5	0.0	1.5	0.4
.tar	3152896	67.9	17.2	43.1	47.0	108.7	52.0	38.4	4.6	31.6	12.0

Output Processing (also for BWT)

One of the drawbacks of bock–sorting compressors is that they are not able to catch longer identical sequences like the LZ, PPM or ACB compressors. Here is a solution for that: Consider that if once contexts are neighbours and they continue identical they will always stay neighbours. This can be exploited in the following: if the continuations are the same instead of another instance of the symbol how many symbols are the same and the first non-equal symbol is output.

Example: ABRACADABRA•ABRACADABRA♦

BW Transformed	A	A	R	*10*♦	D	R	C	A	A	A	B	*2*•
Context	♦	•	A	A	A	A	A	B	C	D	R	R

Entropy Coding (also for BWT)

Typical BWT implementations separate the move–to–front recoding and the entropy coding. When doing so the entropy coder never has a chance to see the real BWT output and thus synchronise with context switches in the BWT output. Such context switches usually change symbol distribution, so it is preferable to flush the coders statistics at this point.
If one includes the MTF operation or similar (a cache) into the entropy coder this can be done. A good heuristics is to flush the statistics when 3 out of the last 5 original symbols are *new*. Fill the statistics with the symbols since the first new one.

Example: …ABAAABBAACCDCC… would cause a flush after seeing D and statistics would contain CCD and escape.

Legal matters (patent rights):

This sorting (including hardware realisations) has been filed for patent in a PCT country (Austria, Nov. 15, No. A2001/96). I am the (not yet) patent holder, and I will grant free limited evaluation and research rights to anyone upon request. Since austrian family support law (wife & 3 kids) doesn't use actual income but ´could have earned´ income I am forced to reserve sellable rights even through I think it is wrong. Fees will be surprisingly low and be based on what you want to do. :-)

For evaluation rights mail to: michael@eiunix.tuwien.ac.at
(or in case your maildaemon doesn't like the .at domain:
michael@hpkloe01.lnf.infn.it)

I am looking for a new challenge!

My university job expires April 30, so I am free for YOUR problems.

Career: Study of computer science at the Vienna University of Technology, Vienna, Austria. Degree Dipl–Ing. (that is after 5 years minimum time) obtained in 1991. The thesis was on real-time-systems, employing some control theory. I studied two years of technical mathematics (no degree). Since student time I have been working as a teaching (programming, algorithms) and research assistant at this university. Two out of last 4 years I was a stay-at-home dad to allow my wife to finish her PhD in technical chemistry. Playgrounds and housework are fine places to think (this was developed at a playground near my home), but publishing and programming is a night–time job.

Interests: Developing/choosing algorithms best suited for a task. Locating possible bottlenecks and find other (better) ways in any application of computer science.

I have worked on various fields (real time control, computergraphics, text compression, specific compression for high energy physics) and would like to see some more fields.

Skills: good overall knowledge (natural sciences preferred); this makes it easy to work in different fields. Good ability to see things at a larger scale and to understand new things. Native language German, familiar with English. I never had good marks in ´lingua latina´.

Due to the long stay-at-home time my programming is out of exercise. I know C++, C, Pascal, Fortran well and have used some others.

Contact: Michael Schindler, Billrothstr.39/21 A–1190 Wien Austria

michael@eiunix.tuwien.ac.at

michael@hpkloe01.lnf.infn.it