Burrows-Wheeler compression with Fountain codes

Bertrand Ndzana Ndzana and Amin Shokrollahi
EPFL, Switzerland
Phone number: +41 21 693{8132, 7512}
E-mail: bertrand.ndzanandzana@epfl.ch, amin.shokrollahi@epfl.ch

Jürgen Abel
Ingenieurbüro Dr. Abel GmbH
Lechstrasse 1
41469 Neuss - Germany
Phone number: +49 2137 999333
E-mail: juergen.abel@data-compression.info

Abstract

In this paper, we provide a lossless Burrows-Wheeler compression algorithm using Fountain codes. Our algorithm is based on the Burrows-Wheeler Transform (BWT), an invertible permutation transform for lossless data compression. The algorithm proceeds as following. First, we use the BWT. Second, we reduce the number of symbols by applying a Run Length Encoding (RLE) scheme. Third, we transform the local context of the symbols into a global context by the Incremental Frequency Count (IFC) stage. At the end, we separately encode the run length data stream with an Entropy Coder (EC) and the IFC symbols data stream with a layered Fountain Coding (FC). The proposed scheme follows the Closed-Loop Iterative Doping (CLID) algorithm together with the multilevel stage decoding Belief Propagation (BP) at the FC stage. Our algorithm offers encouraging compression rates performance for large files.

1. Introduction

Much of the research effort in universal coding has gone into minimization of the redundancy. Many times a compressed source needs to be transmitted over an unreliable channel thereby making an interplay between source and channel coding essential. In this correspondence, we propose an approach in the purely lossless text compression setting based on rateless Fountain Codes and block-sorting transform (also called Burrows-Wheeler transform [2]).

It has long been known that linear encoding can achieve the entropy rate of memoryless sources [27]. Very recently, it was shown this also holds for arbitrary sources [17-21]. To this end, the authors in [19-21] proposed fixed-length data compression algorithms based on the Burrows-Wheeler Transform (BWT) and Low-Density Parity-Check Codes (LDPC). Independently, in [17], an explicit scheme for variable-length data compression for binary sources based on the BWT and the Fountain Codes was proposed. This scheme was extended in [18] for non binary Markov sources with memory. In this paper, instead of considering binary sources or non binary Markov sources, we focus on the problem of data text compression. The BWT is the foundation of our algorithm. This transformation maps the source output of stationary ergodic tree into a sequence that can be decomposed

asymptotically into piecewise independent and identically distributed i.i.d. segments [24]. The challenge is to find a good segmentation model in order to estimate probabilities distribution at each segment of the BWT output. Compression systems studied in [18, 19, 21] follow adaptive segmentation via the Minimum-Description-Length (explained in [27]). We use a different segmentation model, the uniform segmentation introduced in [23].

In [18, 19, 20], to help reduce redundancy, the BWT, was followed by a Global Structure Transformation (GST) [Ref] called Move-To-Front (MTF) transform. We refer the reader to ([13, 30, 31]) for a detailed description of a number of possible GST. We adopt a new GST called Incremental frequency Count (IFC) [16], which is paired with a Run Length Encoding (RLE) stage between the BWT and the combination of an Entropy Coding (EC) and a Fountain Coding (FC) as depicted in Figure 2.

The rest of the paper is as follows. Section 2 introduces the BWCA algorithm. Section 3 presents background material for Fountain Codes and algorithmic tools. Section 4 describes in details the FC block of our algorithm. Finally, we present simulation results in Section 5 that show the superior performance of our algorithm for large files.

2. Burrows-Wheeler Compression Algorithm

The Burrows-Wheeler Compression Algorithm (BWCA) achieves strong compression rates and a high throughput. In contrast to many other compression approaches, the BWCA is a block oriented compression algorithm. A file to be compressed is first divided into data blocks of a fixed size and all blocks are processed separately. The size of a block is in the range of 1 to 10 MB in general. Since each block is processed separately, no context information of the previous block is used in the following blocks. The BWCA itself consists of several stages, which are performed sequentially. Each stage transforms the symbols of an input buffer into symbols of an output buffer, which is used as the input buffer for the next stage. A basic BWCA consists of three stages, one of which is the so called Burrows-Wheeler Transformation (BWT) stage, the second is the Global Structure Transformation (GST) stage, and the third part is the Entropy Coding (EC) stage as pictured in Figure 1. An introduction into BWCAs is given by Fenwick in [1].

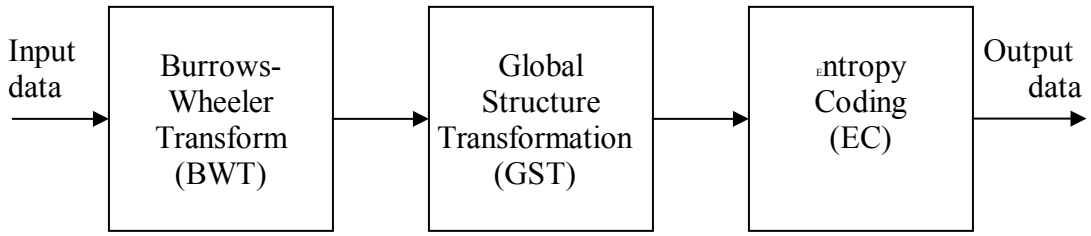

Figure 1: The basic Burrows-Wheeler Compression Algorithm

The first stage – the BWT – performs a permutation of the input symbols, which is the basis for the following stages [2]. The symbols are reordered according to their following context. The output of the BWT stage contains many runs of repeated symbols. During the last years, many fast and efficient algorithms for the BWT have been presented by Larsson and Sadakane [3], Sadakane [4], Itoh and Tanaka [5], Kao [6] and Kärkkäinen and Sanders [7].

The second stage of the BWCA transforms the local structure of the BWT output stream

into an index stream with a global structure and is called a Global Structure Transformation (GST). Some GST stages use a distance measurement between the occurrence of same symbols like Inversion Frequencies (IF) from Arnavut and Magliveras [8] or Distance Coding (DC) from Binder [9] but most GST stages use a more or less simple recency ranking scheme similar to the list update algorithm [10]. The most common approach is the Move To Front (MTF) stage, which was used in the original scheme by Burrows and Wheeler [2] and which is very fast. Better compression rates are achieved by the more complex Weighted Frequency Count (WFC) stage from Deorowicz [13] and the Incremental Frequency Count (IFC) stage by Abel [16]. The MTF stage uses a list with entries for each alphabet symbol. Each time, an input symbol is processed, the entry of that symbol is moved to the front of the list and the former position of the entry inside the list is output. This way, runs inside the input stream are transformed into a sequence of zeros from the second symbol on. Symbols, which occur often within the near past, are transformed into a sequence of small indices. A weak point of the MTF is the fact that the entry of the symbol is always moved to the front no matter how seldom the symbol appeared in the past. Even if the symbol appears the first time, it is moved to the front and removes other symbols, which might be more frequent, from the top of the list. Some MTF derivates try to extenuate this effect like the MTF-1 and MTF-2 approaches of Balkenhol et al. [11] or a sticky version by Fenwick [12]. The WFC stage from Deorowicz uses a different technique. For each alphabet symbol a counter is used, which represents the frequency of the symbol in the near past. The counters are sorted in descending order inside a list. Each time a symbol is processed, the current index of that symbol inside the list is output, and the counters are recalculated and sorted [13]. The calculation of the counters takes into account not only the absolute frequency but also the distance of the occurrences. Occurrences of near symbols are weighted stronger than occurrences of more distant symbols. By weighting the distances of the symbols, the WFC achieves strong compression rates. The drawback is the high complexity by the recalculation of the whole list for each symbol. The IFC stage from Abel tries to solve this problem by using a counter list, in which only one counter is changed for each symbol processed, which makes the IFC very fast [16]. The calculation of the counter takes several statistical properties into account, one of which is the average index value of the near past. Symbols, which occur inside a stable context, are weighted stronger than symbols inside a new context. The compression rates of the IFC stage are in the range of the results of the WFC stage, while the speed is similar to the MTF stage. In order to improve the compression rates and the speed further, a Run Length Encoding (RLE) stage can be used. The RLE cuts the run length of all runs to a fixed size. Especially large runs tend to hamper the symbol probability inside the following stages. Balkenhol and Shtarkov name this phenomenon "the pressure of runs" [15]. Most the time, the RLE stage is placed after the GST stage. This implementation places the RLE stage it in front of the GST [16] as shown in Figure 2. A typical RLE stage would place the length information of the runs in the same output stream than the symbols. Here, all runs of size 2 or more are cut to 2 symbols by an RLE-2 stage and the length information is transmitted into a separate run length data stream and compressed separately by an arithmetic coder. This way, the context of the main output stream is not disturbed by the length information.

The symbol data stream which is the main output of the RLE-2 stage is processed by the GST stage and finally compressed into a bit stream. Some stages use Huffman coding like BZIP2 [14], others variable length codes [12] and many use arithmetic coding [13, 16]. In this correspondence, the output of the GST stage is compressed by a layered Fountain

Coding (EC) and the run length data stream is compressed by an Entropy Coding (EC) as depicted in Figure 2.

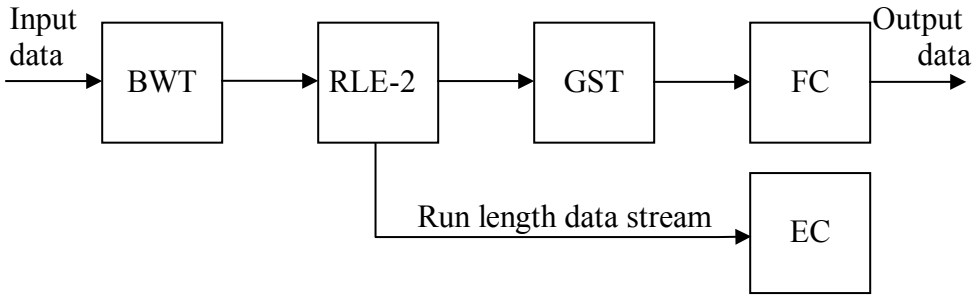

Figure 2: The BWCA with an RLE-2 stage in front of the GST stage

3. Fountain codes and algorithmic tools

Fountain codes are rateless codes used to combat noise when the noise level is not known. For example, these codes can be used over Binary Erasure Channels (BEC(p)) when the parameter p is unknown. LT codes are one of the first classes of Fountain codes [31]. Raptor codes [32] are an extension of this family of codes with linear time encoding and decoding.

In this work, we concentrate on LT codes. An LT code is a Fountain code with parameter (k, D), where k is the length of input sequence and $D = (D_1, ..., D_k)$ denotes the probability distribution that the value j is chosen. Without loss of generality, we assume that the input sequence is binary.

Each coded bit is the exclusive-or of a subset of the input bits. The number of input bits in this set is called the degree of the output. To generate an output bit, the encoder works as follows. First, choose the degree j of the output bit from the degree distribution D. Second, choose uniformly at random j distinct input bits as neighbors of the output bit. The value of the output bit is the exclusive-or of the j input bits.

The performance of LT codes with respect to a given decoding algorithm is measured in terms of the error rate as a function of the reception overhead. The overhead is the number of output symbols that the decoder needs to collect in order to recover the input symbols with high probability. The decoder collects output symbols and estimates for each received output the amount of information in that symbol. When output symbols are bit, this measure of information can be obtained from the Log-Likelihood Ratio (LLR) of the received bit. The receiver stops collecting output bits as soon as the accumulated information carried by the observed channel outputs exceeds $(1 + g)k$, where g is the overhead associated with the LT codes, and k is the number of input symbols.

The decoding graph of length n of an LT code with parameters (k, D) is a bipartite graph with k nodes on one side (called input nodes which correspond to the input symbols) and n nodes on the other side (called output nodes which correspond to output symbols). The decoder uses the Belief Propagation (BP) algorithm (see [33]) to recover the input symbols from the information contained in the output symbols.

To make the rest of the exposition easier, let formalize some notations. Consider a q-ary

(non binary alphabet) source S with source alphabet A= {0, ..., (q-1)}. Without loss of generality, the alphabet cardinality $q = 2^d$ for some integer d. The probability distribution of a source output X is assumed to be $P(X = a) = P(a)$ for $a \in A$.

We note $B(a) = (b_1(a), ..., b_d(a))$ as the binary representation of the symbol $a \in A$. We assume that we can not observe source symbols, but instead, we sequentially their binary representation $b_j(a)$. Let the binary representation $B(c) = (b_1(c), ..., b_\tau(c), ..., b_d(c))$ of the symbol c. ? is called the binary representation level of c and $b_\tau(c)$ is the bit corresponding to this level. For example 11101 is the binary representation of the symbol 23, with 1 being the bit corresponding to the first level and 0 the one corresponding to the fourth level. Note that, for our data compression, d = 8.

We define the conditional marginal probability at level $? \in \{1, ..., d\}$ as

$p_\tau(b_1, ..., b_{\tau-1}) = p(b_\tau(X), = 1| b_1(X), = b_1, ..., b_{\tau-1}(X) = b_{\tau-1}) = Q_1/Q$

with $Q1 = \sum_{x \in A,\, b1(x)= b1(x),\, ...,\, b\tau-1(x) = b\tau-1,\, b\tau(x)= 1} P(x)$ and $Q = \sum_{x \in A\ b1(x)= b1(x),\, ...,\, b\tau-1(x) = b\tau-1} P(x)$ The conditional entropy $H(b_\tau(x)/b_1(x), ..., b_{\tau-1}(x))$ is given by

$\sum_{x \in A\ b1(x)= b1(x),\, ...,\, b\tau-1(x) = b\tau-1} P(x)\ h(\ p_\tau(b_1, ..., b_{\tau-1}))$

We note |A| as the cardinality of the source alphabet.

4. Fountain Coding (FC) algorithm

In this section, we present the Fountain Coding (FC) algorithm used in our compression system. The FC block is a static algorithm which requires two passes, first the modeling part to learn the distributions probabilities and the second pass is to accomplish the encoding. we describe the modeling part and the coding part in detail.

4.1. Modeling

Let's assume that the source sequence $x = (x_1, ..., x_n)$ takes values on a given q-ary alphabet.

If the sources are i.i.d. , we can simply estimate the distribution empirically and plug the estimates into the formulas for entropy. If the sources are known piecewise i.i.d., the BWT maps the source output of stationary ergodic tree into a sequence that can be decomposed asymptotically into piecewise independent and identically distributed i.i.d. Segments [24] as mentioned before. Source statistical modeling consists first of finding an efficient way of segmenting the source, next estimating the first order distribution on each segment, and thus we can estimate the empirical entropy of our source.

A source statistics model is given by the number of segments, the distinct transitions points between segments and by the model segment distributions. The cost of such a statistical model to represent x is the total number of bits needed to describe x. In [17, 18], to find the most efficient piecewise i.i.d. source model, authors implemented a segment merging algorithm (explained in [27]) via the Minimum description length principle that learn an approximation to the source tree, identifying segments by their context. Cai et al. [23] used different approaches to approximate the source tree and proposed two different segmentation models. The first segmentation method called the adaptive segmentation which estimates the location of the transitions of the the BWT output sequence based on the empirical distribution of the symbols. This adaptive algorithm, first obtain rough estimates for transition locations, and refines the locations of the estimates at the second pass. The number and lengths of the segments are adapted to the realization from the

source, and this results in general in segmentation of different lengths. The second approach to segment is the uniform segmentation , in which the BWT output is partitioned such that each segment contains an equal number of symbols from the sequence according to which segmentation is done. We denote this number of symbols by w(n). By taking w(n) as □(√n) , it has been established that, as n tends to ∞, the entropy estimator converges to the entropy rate with high probability for stationary ergodic sources. From experimental results, it has been established that uniform segmentation method performs almost as well as the the adaptive method.
We will follow the uniform segmentation. The advantage of using this segmentation is that the encoder do not need to send the transitions points between segments to the decoder. The decoder needs only to know the length of the first segment.
The source modeling algorithm works as follows:

- First apply the BWT on the sequence, following by a Run Length Encoding (RLE) . The RLE stage replaces all runs of repeated symbols, which have a length of two or more symbols, by a run consisting of exactly two symbols. The output of the RLE is consisting of two separate data stream, the run symbols data stream (RSDT) and the run length data stream (RLDT).
- Apply a Global Structure Transformation (GST) on the RSDT output.
- For each binary representation level ? from 1 to d,
  - Partition the BWT-RSDT-GST output sequence into |?| = n/w(n) segments, where w(n) = $c_\tau\sqrt{n}$. Where c_τ depends of the binary representation level. □ = {1, ..., |□| }, where |□| is the number of segments.
  - Estimate the first-order distribution within each segment. We estimate the number of occurrences of symbol x in the kth segment by $N_k(x)$, and the probability estimate of symbol x in the kth segment by P(x, k), with
    $P(x, k) = (N_k(x) + □)/ \sum_{y \in A}(N_k(y) + |A|□)$, with □ > 0
    We defined $p^k_\tau(b_1, ..., b_{\tau-1})$ as the conditional marginal probability on the kth segment, with P(x) = P(x, k) . The contribution to the conditional entropy estimate of the empirical distribution in the kth segment is summarized by
    $\log_2 q(k) = \sum_{x \in A}(N_k(x) + |A|□)\log_2 p^k_\tau(b_1, ..., b_{\tau-1})$
  - Average the individual estimates. The estimate conditional entropy is
    $H(b_\tau(X)/b_1(X), ..., b_{\tau-1}(X)) = (-1/n)\sum_{k \in \Gamma} \log_2 q(k)$
  - The conditional probability Log Likelihood Ratio (LLR) associated to a symbol x_h the hth symbol of the sequence x is defined as
    $\log((1 - p^k_\tau(b_1(x_h), ..., b_{\tau-1}(x_h)))/p^k_\tau(b_1(x_h), ..., b_{\tau-1}(x_h)))$
  - The estimation of the entropy is $\sum_{\tau \in \{1, ..., d\}} H(b_\tau(x)/b_1(x), ..., b_{\tau-1}(x))$, using quantized versions of $p^k_\tau(b_1, ..., b_{\tau-1})$

4.2. Coding

In this part, the compression and decompression algorithm is described. The main ideas of our text compression scheme is the main building block of the source coding scheme using Fountain Codes outlined before for binary sources and for non binary Markov sources [17, 18] . Let ? = { D_1, ..., $D_{|\Psi|}$} be a finite ensemble of LT codes distributions optimized for the erasure channel [33]. After the GST and the modeling stages, the output symbols is the sequence (y_1, ..., y_n). Let the sequence ($b_\tau(y_1)$, ..., $b_\tau(y_n)$) be the binary representation of the sequence (y_1, ..., y_n) at the ?th binary representation level (see subsection A).

The coding strategy works as follows.

- For each binary representation level ? from 1 to d:
  - Calculate from the binary symbols $(b_\tau(y_1), ..., b_\tau(y_n))$ a vector of binary intermediate symbols $(z_{\tau 1}, ..., z_{\tau n})$ through a random linear invertible k * k matrix G:
    $(z_{\tau 1}, ..., z_{\tau n}) = G\ (b_\tau(y_1), ..., b_\tau(y_n))$.
  - For each distribution D_s of ?, m_s symbols $(y_{(n+1)}, ..., y_{(n+ms)})$ is generated from $(z_{\tau 1}, ..., z_{\tau n})$ through encoding with an LT-code with parameters (n, D_s).
  - A bipartite graph is set up between the nodes corresponding to $(z_{\tau 1}, ..., z_{\tau n})$ and on the other side the nodes corresponding to $(b_\tau(y_1), ..., b_\tau(y_n))$ and the nodes corresponding to $(y_{(n+1)}, ..., y_{(n+ms)})$ obtained previously.

The BP algorithm is applied to this graph. The objective of the BP algorithm is to decode the symbols $(z_{\tau 1}, ..., z_{\tau n})$ using the full knowledge of the symbols $(y_{(n+1)}, ..., y_{(n+ms)})$ and the absolute values of LLR values coming from the modeling part. During the BP-algorithm, the Closed-Loop Iterative Doping (CLID) algorithm is applied: In a given level ?, every f-th round of the iteration the, intermediate bit with smallest reliability is marked, ans its LLR is set to $+\infty$ or $-\infty$ depending on whether its value is 0 or 1. f is a design parameter of the algorithm. The BP-decoding together with CLID is continued until the intermediate bits satisfy all the parity check equations of the LT code. These m_s output symbols, together with the doped symbols obtained from the CLID algorithm constitute the output of the compressor; m_s should be as close as possible to $n(H(b_\tau(y)/b_1(y), ..., b_{\tau-1}(y)) + ?)$, where ? is a small bias > 0. For each level, the encoder send the code distribution (and a seed) of ? for which the number of doped symbols is smallest , a seed for the matrix G, and LLR (its quantized version) values to the decoder using the adaptive Huffman coding [28]. The decompression is achieved level by level in several steps which closely mimic the compression steps. After receiving sequence $(y_{(n+1)}, ..., y_{(n+ms)})$, quantized values of LLR coming from the modeling part, the corresponding doped bits, the decompression is applied to the level ? sequence $(b_\tau(y_1), ..., b_\tau(y_n))$, all the sequences at lower levels 1 though ?-1 have been already recovered and the conditional LLR needed by the BP decoder at level ? are known.

5. Experiments

We experimentally evaluated the compression rate of our scheme with Calgary Corpus and large Canterbury Corpus files and compare it to leading text compression system GZIP–B, BZIP-9 and PPMD5 (see [35]) in the following tables. IFOUNT05 is the approach presented in this paper, where the GST stage is the IFC.

Table1: Compression rates for the Calgary Corpus in bits per symbol

File	**Size**	**IFOUNT05**	**GZIP-B**	**BZIP-9**	**PPMD5**
Bib	111261	**To fill**	2.51	1.95	1.89
book1	768771	**To fill**	3.25	2.40	2.34
book2	610856	**To fill**	2.70	2.04	1.98

File	Size	IFOUNT05	GZIP-B	BZIP-9	PPMD5
Geo	102400	**To fill**	5.34	4.48	4.96
news	377109	**To fill**	3.06	2.51	2.42
obj1	21504	**To fill**	3.84	3.87	3.70
obj2	246814	**To fill**	2.63	2.46	2.35
paper1	53161	**To fill**	2.79	2.46	2.36
paper2	82199	**To fill**	2.89	2.42	2.34
progc	39611	**To fill**	2.68	2.50	2.40
progl	71646	**To fill**	1.80	1.72	1.69
progp	49379	**To fill**	1.81	1.71	1.72
trans	93695	**To fill**	1.61	1.50	1.50
Avgerage	**202186**	**To fill**	**2.84**	**2.46**	**2.43**

Table2: Compression rates for the Large Canterbury Corpus in bits per symbol

File	Size	IFFO05	GZIP-B	BZIP-9	PPMD5
Bible.txt	4047392	**To fill**	2.24	2.13	1.99
E.coli	4638690	**To fill**	2.33	1.65	1.58
World192.txt	2473400	**To fill**	2.33	1.57	1.52
Average	**3719828**	**To fill**	**2.30**	**1.79**	**1.70**

Performance ...
These results can be considerably improved [34] by making a text preprocessing which takes into consideration the special properties of textual data.

6. Acknowledgments

We would like to thank Giussepe Caire, Emre Telatar and Payam Pakzad for several remarks and discussions on this work.

References

[1] Khalid Sayood (Editor). Lossless Compression Handbook. Academic Press, 2003.
[2] Burrows, M, Wheeler, D J. A Block-Sorting Lossless Data Compression Algorithm. Technical report, Digital Equipment Corporation, Palo Alto, California, 1994, URL (October 2005): http://citeseer.ist.psu.edu/76182.html.
[3] Larsson, N J, Sadakane, K. Faster Suffix Sorting, Technical report, 1999, URL (October 2005): http://citeseer.ist.psu.edu/larsson99faster.html.
[4] Sadakane, K. Unifying Text Search and Compression – Suffix Sorting, Block Sorting and Suffix Arrays, Ph.D. Dissertation, Department of Information Science, Faculty of Science, University of Tokyo, 2000.

[5] Itoh, H, Tanaka, H. An Efficient Method for in Memory Construction of Suffix Arrays. Proc. IEEE String Processing and Information Retrieval Symposium (SPIRE'99), 81–88, September 1999.
[6] Kao, T H. Improving Suffix-Array Construction Algorithms with Applications, Master's thesis, Gunma University, Kiryu, 376–8515, Japan, 2001.
[7] Kärkkäinen, J, Sanders, P. Simple Linear Work Suffix Array Construction. 30th International Colloquium on Automata, Languages and Programming, number 2719 in LNCS, 943–955, Springer, 2003.
[8] Arnavut, Z, Magliveras, S S. Block Sorting and Compression. Proceedings of the IEEE Data Compression Conference 1997, Snowbird, Utah, J. A. Storer and M. Cohn, Eds. 181–190, 1997.
[9] Binder, E. Distance Coder, Usenet group: comp.compression, 2000, URL (October 2005): http://groups.google.com/groups?selm=390B6254.D5113AD2%40T-Online.de.
[10] Bentley, J, Sleator, D, Tarjan, R, Wei, V. A locally adaptive data compression scheme. Communications of the ACM, 29, 320–330, 1986.
[11] Balkenhol, B, Kurtz, S, Shtarkov, Y M. Modifications of the Burrows and Wheeler Data Compression Algorithm. Proceedings of the IEEE Data Compression Conference 1999, Snowbird, Utah, J. A. Storer and M. Cohn, Eds. 188–197, 1999.
[12] Fenwick, P. Burrows Wheeler Compression with Variable Length Integer Codes. Software – Practice and Experience, 32(13), 1307–1316, 2002.
[13] Deorowicz, S. Second step algorithms in the Burrows-Wheeler compression algorithm. Software – Practice and Experience, 32(2), 99–111, 2002.
[14] Seward, J. On the performance of BWT sorting algorithms. Proceedings of the IEEE Data Compression Conference 2000, Snowbird, Utah, J. A. Storer and M. Cohn, Eds., 173–182, 2000.
[15] Balkenhol, B, Shtarkov, Y M. One attempt of a compression algorithm using the BWT. SFB343: Discrete Structures in Mathematics, Falculty of Mathematics, University of Bielefeld, Preprint, 99–133, 1999, URL (October 2005): http://citeseer.ist.psu.edu/balkenhol99one.html.
[16] Abel, J. A fast and efficient post BWT-stage for the Burrows-Wheeler Compression Algorithm. Proceedings of the IEEE Data Compression Conference 2005, Snowbird, Utah, J. A. Storer and M. Cohn, Eds., 449, 2005.
[17] G. Caire, S. Shamai A. A. Shokrollahi, and S. verdu, Universal variable-length data compression of binary sources using Fountain Codes, Information Theory Workshop, 2004. IEEE, 24-29 Oct. 2004 Pages(s):123 -128
[18] G. Caire, S. Shamai A. A. Shokrollahi, and S. verdu, Fountain Codes for lossless data compression, in Dimacs Series in Discrete Mathematics and Theoretical Computer Science, A. Barg and A. Ashkhimin, Eds. American MathematicalSociety, 2005
[19] Caire G., Shamai S., Verdu, S.,A new data compression algorithm for sources with memory based on error correcting codes, Information Theory Workshop, 2003. Proceedings. 2003 IEEE 31 March-4 April 2003 Page(s):291 - 295
[20] Caire Giuseppe, Shamai Shlomo Shitz, Verdu Sergio, Universal data compression with LDPC codes , 3rd International Symposium on Turbo Codes and Related Topics, 1-5 September 2003, Brest, France
[21] Caire G., Shamai S., Verdu S., Lossless data compression with error correction codes, 2003 IEEE Int. symp. on Information Theory, p. 22, June 29-July 4, 2003

[22] Caire G., Shamai S., Verdu S., Noiseless data compression with low-density parity-check codes, in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, P. Gupta and G. Kramer, eds. American Mathematical Society, 2004
[23] Haixiao Cai, Kulkarni S.R., Verdu S., Universal entropy estimation via block sorting, Information Theory, IEEE Transactions on Volume 50, Issue 7, July 2004 Page(s): 1551 - 1561
[24] Visweswariah K., Kulkarni S, Verdu S., Output distribution of the Burrows-Wheeler transform , Information Theory, 2000. Proceedings. IEEE International Symposium on 25-30 June 2000 Page(s):53
[25] Baron D., Bresler Y., An O(N) semipredictive universal encoder via the BWT, Information Theory, IEEE Transactions on Volume 50, Issue 5, May 2004 Page(s): 928 - 937
[26] I. Csiszar and J. Korner, Information Theory: Coding Theorems for Discrete memoryless Systems, Academic, New york, 1981.
[27] I. Csiszar and J. Korner, Information Theory: Coding Theorems for Discrete memoryless Systems, Academic, New york, 1981.
[28] David Salomon, Coding for DATA AND COMPUTER COMMUNICATIONS, Springer Science, 2005, pages 68-110
[29] J. Abel, Improvements to the Burrows-Wheeler Compression Algorithm: After BWT stages, Preprint dated 31.03.2003
[30] S. Deorowicz, Universal lossless data compression algorithms, Doctoral dissertation, Silesian University of Technology, Gliwice, Poland (2003)
[31] M. G. Luby, LT codes, Proc.43rd IEEE Symo. Foundations of computer science, pp. 271-280, 2002
[32] A. Shokrollahi, Raptor Codes , Preprint, 2003
[33] O. Etesami, M. Molkaraie, and A. Shokrollahi, Raptor codes on symmetric channels, Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on 27 June-2 July 2004 Page(s):38
[34] Bertrand Ndzana Ndzana, Amin Shokrollahi and Juergen Abel, Universal Lossless Source Coding for text document using Fountain Codes, in preparation
[35] The Canterbury corpus: http://corpus.canterbury.ac.nz/details/calgary/RatioByRatio.html