ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 2010. No. 1(97)
ELEKTRONIKA IR ELEKTROTECHNIKA

SYSTEM ENGINEERING, COMPUTER TECHNOLOGY
T 120
SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

Fibonacci Coding Within the Burrows-Wheeler Compression Scheme

R. Bastys
Faculty of Mathematics and Informatics, Vilnius University,
Naugarduko str. 24, LT-03225 Vilnius, Lithuania, phone: +370-674-45577, e-mail: rbastys@yahoo.com

Introduction

Burrows-Wheeler algorithm (*BWA* for short) is a lossless data compression scheme, named after authors Michael Burrows and David Wheeler. The classical work here is [1]. Also known as block-sorting, currently it is among the best textual data archivers in terms of compression speed and ratio. In this work we describe our *BWA* based data compressor implementation working principles and compare it with some other popular file archivers. As for prerequisites, the reader is expected to be familiar with basic lossless data compression techniques.

Burrows-Wheeler compression scheme

In this section we provide a detailed exposition of *BWA* and review some of the standard facts on lossless data compression.

Original Burrows-Wheeler scheme archives input string s (we use *"caracaras"* throughout our examples) in three major steps:

STEP 1: Calculate Burrows-Wheeler transformation *(BWT)* of s. We denote it briefly by $s' = BWT(s)$. Transformation permutes input string symbols as follows:

1. Build input string cyclic permutations matrix.
2. Sort matrix rows ascending.
3. Output last sorted matrix column (Fig. 1).

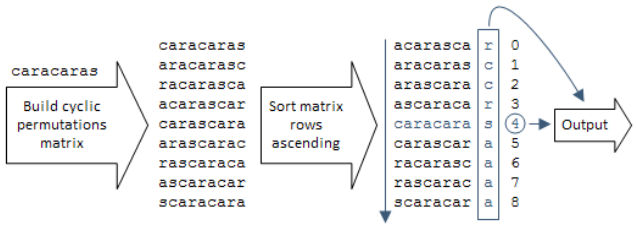


Fig. 1. *BWT* of string *"caracaras"*

To calculate the inverse transformation (restore s provided s') one must also know original string index in sorted matrix [1]. Hence the complete *BWT* output is

$$BWT("caracaras") = ("rccrsaaaa", 4).$$

STEP 2: Run Move-To-Front *(MTF)* transformation on *BWT* output: $s''_{MTF} = MTF(s')$. *MTF* algorithm renders *BWT* output into a sequence of integers:

1. Fix some s' alphabet permutation, e.g. sort it ascending.
2. Encode the next message symbol by its position in current alphabet permutation.
3. Move encoded symbol to the beginning of the alphabet.
4. Repeat steps 2 and 3 until the whole message is encoded (Fig. 2):

$$s''_{MTF} = MTF("rccrsaaaa") = (2,2,0,1,3,3,0,0,0).$$

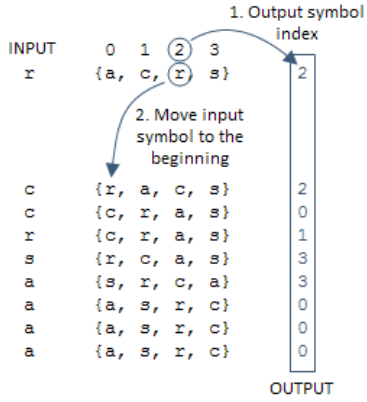


Fig. 2. *MTF* transformation of string *"rccrsaaaa"*

STEP 3: Encode *MTF* output s''_{MTF} by any entropy encoder (*EE*), e.g. Huffman [2] or Arithmetic [3].

Thus, the entire Burrows-Wheeler compression scheme may be written as

$$BW(s) = EE\left(MTF\left(BWT(s)\right)\right). \quad (1)$$

From now on, s stands for a string over the alphabet $\langle s\rangle = \{\sigma_1, \ldots, \sigma_h\}$. Assume each σ_i appears n_i times in s, which fixes s length to be $|s| = \sum_{i=1}^{h} n_i$. The *entropy* of s, denoted by $H(s)$, is defined to be the sum

$$H(s) = -\sum_{i=1}^{h} \frac{n_i}{|s|} log_2 \frac{n_i}{|s|}. \quad (2)$$

Also referred to as Shannon's entropy, $H(s)$ represents the minimum average number of bits required to encode one symbol of s. An equivalent formulation of this fact is

$$|EE(s)|_8 \geq \|s\|\,, \tag{3}$$

where $|EE(s)|_8$ denotes archiver output file size in bytes and $\|s\|$ is given by

$$\|s\| = \frac{|s|H(s)}{8}. \tag{4}$$

On the other hand any good *EE* algorithm achieves the reverse inequality

$$|EE(s)|_8 \leq \|s\| + \frac{C_{EE}}{8}, \tag{5}$$

the constant C_{EE} being relatively small and independent of s. For instance, $C_{HUFFMAN} = 1, C_{ARITHMETIC} \approx 0{,}02$ (see [2] and [3] for more details).

EXAMPLE 1: Let $s =$ "*aaaaabbbbbccccddddd*". Then

$$H(s) = \sum_{i=1}^{4} p_i \log_2 \frac{1}{p_i} = 4 \cdot \frac{5}{20} \log_2 \frac{20}{5} = \log_2 4 = 2\,.$$

Now let us run *MTF* on s and check how it affects the entropy.

$$s''_{MTF} = MTF(s)$$
$$= (0,0,0,0,0,1,0,0,0,0,2,0,0,0,0,3,0,0,0,0)\,,$$

$$H(s''_{MTF}) = \frac{17}{20} \log_2 \frac{20}{17} + 3 \cdot \frac{1}{20} \log_2 \frac{20}{1} \approx 0{,}8576\ldots$$

This straightforward example demonstrates rather strikingly a couple important facts. First thing, the structure of s''_{MTF} makes it obvious that *MTF* renders recurring consecutive symbols into series of zeroes, furthermore, the entropy of s significantly exceeds that of s''_{MTF}. Loosely speaking, scattered alphabet symbols probabilities decrease string entropy, therefore it is to be expected that $H(s''_{MTF}) \ll H(s)$ (Fig. 10), hence that $\|s''_{MTF}\| \ll \|s\|$ and finally that $|EE(s''_{MTF})|_8 \ll |EE(s)|_8$. *BWA* elegantly brings these properties together. It is not obvious on short "*caracaras*" example, so let us take another case. *BWT* of a long text input, such as the Wikipedia article on the Caracara (http://en.wikipedia.org/wiki/Caracara), should look similar to that in Fig. 3.

It is evident that the transformation groups together symbols preceding similar contexts, thus producing long homogenous chains in the last sorted matrix column. *MTF* converts them into series of zeroes, which reduces string entropy, and, in consequence, makes it a fitter input for any entropy encoder.

It is left to show that *BWT* can be calculated within a reasonable amount of time and does not require excessive PC memory usage. Compressing, say, ordinary 1 MB file involves sorting permutations matrix of size $10^6 \times 10^6$, which may become quite an expensive operation in case an improper sorting algorithm is applied. The two leading *BWT* sort approaches are *Bentley-Sedgewick sort* (modification of quick sort) and *suffix sort*. Both have their advantages and disadvantages, but we will not develop this point here. For a deeper discussion of these algorithms we refer the reader to [5] – [8].

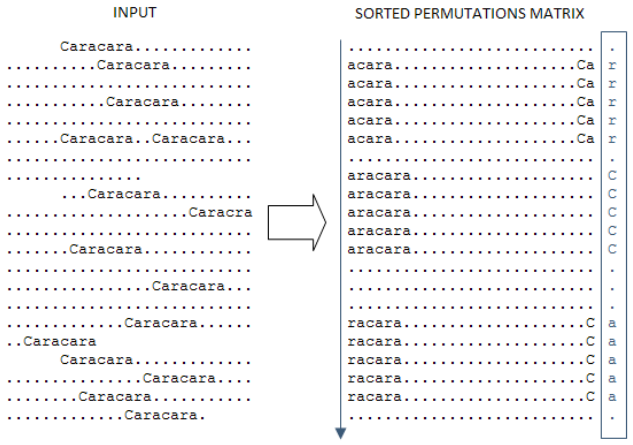


Fig. 3. Schematic *BWT* view of a long text input

One may mistakenly conjecture that *BWA* is suitable for compressing any input. Of course, technically it is possible to calculate both *BWT* and BWT^{-1} of any file, but the overall *BWA* efficiency highly depends on the source characteristics, especially the amount of consistent patterns (to put it simply – words) in it. As a rule *BWT* works well on plain text files (Fig. 4), yet it is rather useless when applied to large entropy binary files (Fig. 5).

Fig. 4. *BWT* fragment of this article in .tex format: numerous runs of consecutive identical symbols

Fig. 5. *BWT* fragment of this article in .pdf format: chaotic structure, occasional runs of consecutive identical symbols

The next section is devoted to the study of Distance Coding algorithm, which is in all likelihood the most successful *MTF* alternative at the second *BWA* step.

Distance Coding

Distance Coding (*DC*) was originally proposed by Edgar Binder at comp.compression newsgroup [9] in 2000. There is no official paper on *DC* by Edgar, so we provide the algorithm (Fig. 6) here:

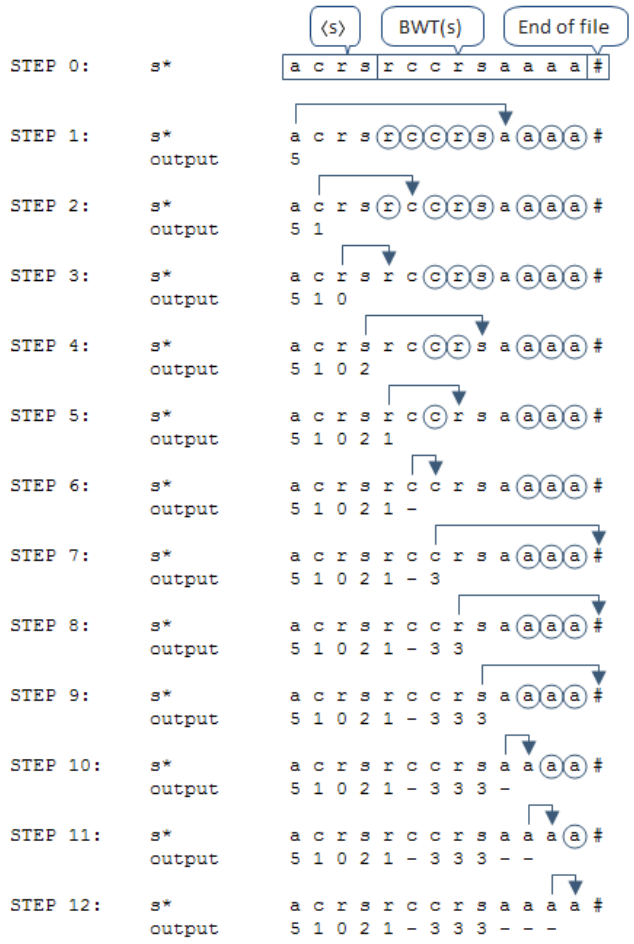


Fig. 6. *DC* algorithm for string *"rccrsaaaa"*

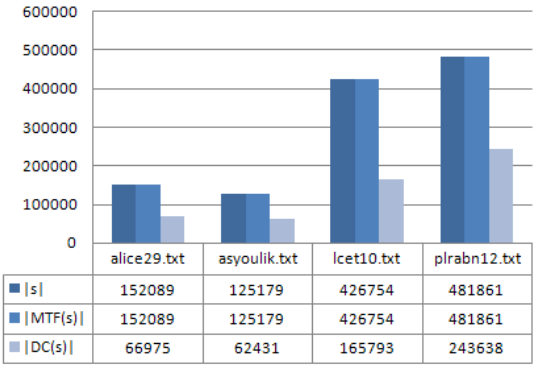
1. Fix $s^* = \langle s \rangle + s' + eof\_pointer$. Someway mark s' symbols yet to be encoded (encircled in our example).
2. Encode the next message symbol σ by the number of unencoded (encircled) characters till the next occurrence of σ (or end-of-file pointer if there are no σ-as left).
3. Unmark the second σ.
4. Repeat steps 2 and 3 until the whole message is encoded.

Assume we are encoding symbol σ_1, whose right neighbor σ_2 is not yet encoded. This clearly forces $\sigma_1 = \sigma_2$, since otherwise some other symbol would have already pointed to σ_2. Such deduction allows us not to encode it any extra (steps 6, 10, 11 and 12), this way shortening *DC* output:

$$s''_{DC} = DC(\text{"}rccrsaaaa\text{"}) = (5,1,0,2,1,3,3,3).$$

Let us compile some basic facts on s''_{MTF} and s''_{DC} provided s satisfies the conditions of the previous section ($\langle s \rangle = \{\sigma_1, \dots, \sigma_h\}, |s| = \sum_{i=1}^{h} n_i$). We illustrate each property by our calculations on the Canterbury Corpus files [10].

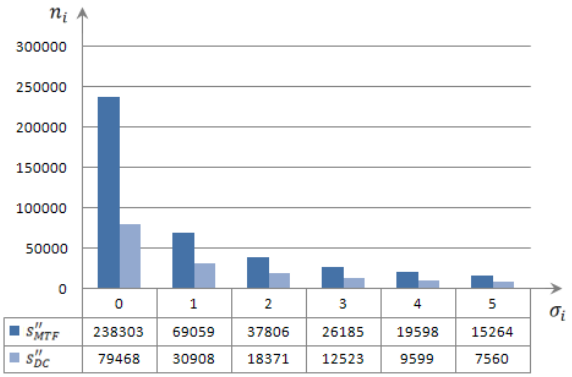
1. Unlike *MTF*, *DC* truncates input on the account of homogenous chains in s' (Fig. 7)



	alice29.txt	asyoulik.txt	lcet10.txt	plrabn12.txt
|s|	152089	125179	426754	481861
|MTF(s)|	152089	125179	426754	481861
|DC(s)|	66975	62431	165793	243638

Fig. 7. Canterbury Corpus .txt files: $|s''_{MTF}| = |s|; |s''_{DC}| \ll |s|$

2. s''_{MTF} is a sequence of non-negative integers; so is s''_{DC}. Small numbers prevail in both (Fig. 8).



	0	1	2	3	4	5
s''_{MTF}	238303	69059	37806	26185	19598	15264
s''_{DC}	79468	30908	18371	12523	9599	7560

Fig. 8. Canterbury Corpus plrabn12.txt file: $\langle s''_{MTF} \rangle$ and $\langle s''_{DC} \rangle, \sigma_i \leq 5$

3. Since *MTF* encodes each symbol by its index in $\langle s \rangle$, we have $|\langle s''_{MTF} \rangle| \leq h$. *DC* measures distance between identical symbols, and hence $\langle s''_{DC} \rangle$ can contain numbers up to $|s| - 1 \gg h$ (Fig. 9)

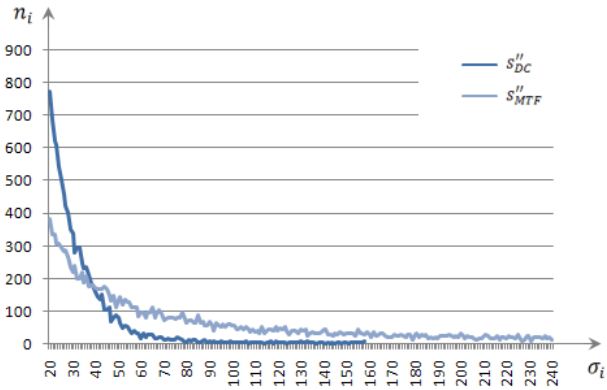
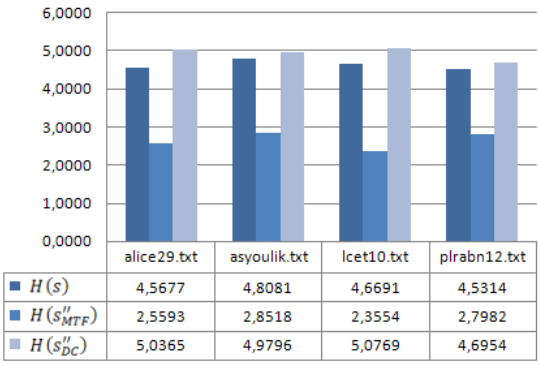


Fig. 9. Canterbury Corpus ptt5 file: $\langle s''_{DC} \rangle$ tail is longer and heavier than that of $\langle s''_{MTF} \rangle$

4. The previous property together with (2) also yield $H(s''_{DC}) \gg H(s''_{MTF})$ (Fig. 10)



	alice29.txt	asyoulik.txt	lcet10.txt	plrabn12.txt
$H(s)$	4,5677	4,8081	4,6691	4,5314
$H(s''_{MTF})$	2,5593	2,8518	2,3554	2,7982
$H(s''_{DC})$	5,0365	4,9796	5,0769	4,6954

Fig. 10. Canterbury Corpus .txt files: $\boldsymbol{H(s''_{DC}) \gg H(s''_{MTF})}$

Comparison of $\|s''_{MTF}\|$ and $\|s''_{DC}\|$ (which is basically a combination of properties 1 and 4) shows that *DC* has a higher potential (Fig. 11). Unfortunately, classical entropy encoders are not well adapted to compressing s''_{DC} due to a very large $\langle s''_{DC} \rangle$ (property 3). Storing the alphabet creates significant overhead, thus decreasing *BWT + DC + EE* scheme efficiency. This difficulty disappears entirely if we replace entropy encoder by some universal code [11], such as Fibonacci.

Fibonacci Coding

Any positive integer N can be represented as the sum

$$N = \sum_{i=1}^{k} d_i F_i, \qquad (6)$$

where F_i is the i-th Fibonacci number (1,1,2,3,5,8, ...), $d_i \in \{0,1\}, d_k = 1$. Fibonacci Code (*FC*) [11] of N is defined by

$$FC(N) = \overline{d_1 \dots d_k 1}. \qquad (7)$$

The important point to note is that no two adjacent coefficients d_i can equal 1, therefore token 11 (=$d_k 1$) immediately indicates the end of code. Thus, *FC* transforms any sequence of integers into uniquely decodable binary string. The structure of *FC* also implies that smaller numbers are being mapped into shorter code words (Table 1):

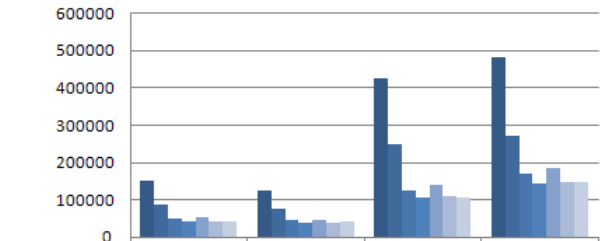
Table 1. Fibonacci Codes, $\boldsymbol{N \le 10}$

N	$d_1F_1 + \dots + d_kF_k$	*FC(N)*
1	$1 \cdot 1$	11
2	$0 \cdot 1 + 1 \cdot 2$	011
3	$0 \cdot 1 + 0 \cdot 2 + 1 \cdot 3$	0011
4	$1 \cdot 1 + 0 \cdot 2 + 1 \cdot 3$	1011
5	$0 \cdot 1 + 0 \cdot 2 + 0 \cdot 3 + 1 \cdot 5$	00011
6	$1 \cdot 1 + 0 \cdot 2 + 0 \cdot 3 + 1 \cdot 5$	10011
7	$0 \cdot 1 + 1 \cdot 2 + 0 \cdot 3 + 1 \cdot 5$	01011
8	$0 \cdot 1 + 0 \cdot 2 + 0 \cdot 3 + 0 \cdot 5 + 1 \cdot 8$	000011
9	$1 \cdot 1 + 0 \cdot 2 + 0 \cdot 3 + 0 \cdot 5 + 1 \cdot 8$	100011
10	$0 \cdot 1 + 1 \cdot 2 + 0 \cdot 3 + 0 \cdot 5 + 1 \cdot 8$	010011

Although in theory *FC* is not as effective as entropy encoders, we were interested in investigating the scheme

$$BW_{FIB}(s) = FC\left(DC\left(BWT(s)\right)\right).$$

The practical advantage of using Fibonacci Code within *BWT + DC* scheme lies in the fact that *FC* is universal code, hence there is no need to store bulky s''_{DC} alphabet. Let us compare our implementation of the *BWA* scheme to some other data compressors performance on the relatively large text files (Fig. 11):



	alice29.txt	asyoulik.txt	lcet10.txt	plrabn12.txt
$\|s\|$	152089	125179	426754	481861
$\|\|s\|\|$	86837	75234	249071	272936
$\|\|s''_{MTF}\|\|$	48655	44623	125647	168543
$\|\|s''_{DC}\|\|$	42165	38860	105214	142997
$\|zip(s)\|_8$	51797	46682	137842	184824
$\|bzip2(s)\|_8$	43178	39560	107683	145310
$\|BW_{FIB}(s)\|_8$	43295	39905	107439	146568

Fig. 11. Canterbury Corpus .txt files: $\boldsymbol{|s|}$ – length in bytes; $\boldsymbol{\|s\|}$, $\boldsymbol{\|s''_{MTF}\|}$ and $\boldsymbol{\|s''_{DC}\|}$ – theoretical lower bounds in bytes for various data compression techniques (4); *zip* – popular commercial file archiver [12]; *bzip2* – one of the best open source *BWA* implementations [13]

1. *BWA* based compressors are roughly twice as effective as plain entropy encoders:
$$|bzip2(s)|_8 \ll \|s\|, |BW_{FIB}(s)|_8 \ll \|s\|.$$
2. *bzip2* output is shorter than the classical *BWA* scheme theoretical lower bound:
$$|bzip2(s)|_8 < \|s''_{MTF}\|.$$
The reason behind that is *bzip2* includes several additional compression layers, the most important being Run-Length Encoding (*RLE*) (see [14] and the references given there).
3. Both BW_{FIB} and *bzip2* excel *zip* - the most popular commercial file archiver.

We were mostly surprised at finding out that BW_{FIB} nearly achieves *BWT + DC + EE* theoretical lower bound:

$$|BW_{FIB}(s)|_8 \sim \|s''_{DC}\|.$$

To sum up, *BWT + DC + FC* is a highly effective scheme for compressing plain text input (this also includes .html, .xml files, various programming languages source code, etc.). Compression and decompression algorithms are simple and relatively fast.

Compression scheme *BWT + MTF + FC* is also possible, but it is unlikely to achieve noticeable results, because even $\|s''_{MTF}\|$ is markedly greater than $|BW_{FIB}(s)|_8$. Besides, it is inexpensive using regular entropy encoder together with *MTF*, since $\langle s''_{MTF} \rangle$ is usually small.

Conclusions and Future Work

1. The main Distance Coding advantage over *MTF* is reduced input length; disadvantage – large output alphabet.
2. Fibonacci Code is very well adapted to compressing *DC* output – results obtained on text files are close to theoretical lower bounds. *BWT* + *DC* + *FC* encoded file requires very little metadata information, since *FC* is a universal code.
3. It is natural to try out other universal codes within Burrows-Wheeler scheme, possibly combining them with entropy codes and/or *RLE*.

References

1. **Burrows M., Wheeler D. J.** A block sorting lossless data compression algorithm // Technical Report 124, Digital Equipment Corporation, Palo Alto, California. – 1994.
2. **Huffman D. A.** A Method for the Construction of Minimum-Redundancy Codes // Proceedings of the Institute of Radio Engineers. – 1952. – P. 1098–1102.
3. **Witten I., Neal R., Cleary J.** Arithmetic Coding for Data Compression. Communications of the ACM. – 1987. – Vol. 30, No. 6. – P. 520–540.
4. **Shannon C. E.** A Mathematical Theory of Communication. Bell System Technical Journal. – Vol. 27. – 1948. – P. 379–423, 623–656.
5. **Bentley J. L., Sedgewick R.** Fast algorithms for sorting and searching strings // Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms. – 1997. – P. 360–369.
6. **Larsson J., Sadakane K.** Faster Suffix Sorting // Technical report LU-CS-TR:99-214, Department of Computer Science, Lund University, Sweden. – 1999.
7. **Sadakane K.** A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation // Proceedings of the IEEE Data Compression Conference, Snowbird, Utah. – 1998. – P. 129–138.
8. **Sadakane K.** A Comparison among Suffix Array Construction Algorithms. http://citeseer.ist.psu.edu/187464.html. – 1997.
9. **Binder E.** Distance Coding algorithm. http://groups.google.com/group/comp.compression/msg/27d46abca0799d12. – 2000.
10. **University of Canterbury.** http://en.wikipedia.org/wiki/Canterbury_Corpus. – 1997.
11. **Fraenkel A. S., Klein S. T.** Robust universal complete codes for transmission and compression // Discrete Applied Mathematics. – 1996. – Vol. 64, No. 1. – P. 31–55.
12. **Katz P.** ZIP data compression algorithm. http://en.wikipedia.org/wiki/PKZIP.
13. **Seward J.** *bzip2* data compression algorithm. http://bzip.org/.
14. *bzip2* data compression algorithm description. http://en.wikipedia.org/wiki/Bzip2.

Received 2009 09 02

R. Bastys. Fibonacci Coding Within the Burrows-Wheeler Compression Scheme // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 1(97). – P. 28–32.

Burrows-Wheeler data compression algorithm (BWA) is one of the most effective textual data compressors. BWA includes three main iterations: Burrows-Wheeler transform (BWT), Move-To-Front transformation (MTF) and some zeroth order entropy encoder (e.g. Huffman). The paper discusses little investigated scheme when MTF is replaced by the less popular Distance Coding (DC). Some relevant advantages and downsides of such modified scheme are indicated, the most critical being heavy DC output alphabet. It is shown that applying Fibonacci Code instead of entropy encoder elegantly deals with this technical problem. The results we obtain on the Canterbury Corpus text files are very close to the theoretical lower bounds. Our compressor outperforms the most widely used commercial *zip* archiver and achieves sophisticated BWA implementation *bzip2* compression. Ill. 11, bibl. 14, tabl. 1 (in English; abstracts in English, Russian and Lithuanian).

Р. Бастис. Код Фибоначчи использование в схеме сжатия данных Барроуза-Вилера // Электроника и электротехника. – Каунас: Технология, 2010. – № 1(97). – С. 28–32.

Алгоритм Барроуза-Вилера (BWA) является одним из наиболее эффективных методов сжатия текстовых данных. BWA включает три основных итерации: преобразование Барроуза-Вилера (BWT), трансформацию Move-To-Front (MTF) и энтропийный код (например, Хафмана). В статье анализируется мало исследованная схема, когда MTF заменяется кодированием расстояний (англ. Distance Coding). Приводятся основные плюсы и недостатки модифицированного BWA, среди которых черезмерно большой алфавит идентифицируется как основной. Для решения этой технической проблемы предлагается универсальный код Фибоначчи. Его изполозование на третьем шаге BWA с текстовыми данными позволяет достичь результаты весьма близкие к теоретическим. Описываемый алгоритм сжатия данных также превосходит популярный коммерческий архиватор *zip* и близок по эффективности к намного более сложной BWA реализации *bzip2*. Ил. 11, библ. 14, табл. 1 (на английском языке; рефераты на английском, русском и литовском яз.).

R. Bastys. Fibonačio kodo panaudojimas Burrowso ir Wheelerio duomenų kompresijos schemoje // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – Nr. 1(97). – P. 28–32.

Burrowso ir Wheelerio duomenų kompresijos algoritmas (BWA) yra vienas efektyviausių tekstinių duomenų archyvavimo metodų. BWA jungia tris iteracijas: Burrowso ir Wheelerio transformaciją (BWT), Move-To-Front transformaciją (MTF) ir pasirinktą entropinį kodą (pavyzdžiui, Hafmano). Straipsnyje analizuojama mažai išnagrinėta schema, kai MTF pakeičiamas atstumų kodavimu. Nurodomi modifikuoto algoritmo pranašumai ir trūkumai, palyginti su klasikiniu BWA, identifikuojama pagrindinė techninė problema – perteklinė abėcėlė. Jai spręsti pasitelkiamas universalus Fibonačio kodas, leidžiantis išvengti abėcėlės saugojimo sąnaudas. Rezultatai, pasiekti koduojant tekstinius duomenis aprašoma schema, yra labai artimi teoriniams. Pateikiamo algoritmo efektyvumas pranoksta plačiai naudojamą komercinį archyvatorių *zip* ir yra panašus į vienos sudėtingiausių BWA realizacijų (*bzip2*) suspaudimo koeficientą. Il. 11, bibl. 14, lent. 1 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).