Two-level directory based compression

Przemysław Skibiński
University of Wrocław, Institute of Computer Science, Przesmyckiego 20, 51–151 Wrocław, Poland, e-mail: inikep@ii.uni.wroc.pl

30 November 2004

This is a full-version of an article presented at DCC'05 poster session and published in *Proceedings of the IEEE Data Compression Conference (DCC'05)*, page 481

Abstract

In this paper we present a new dictionary-based preprocessing technique and its implementation called TWRT (Two-level Word Replacing Transformation). Our preprocessor uses several dictionaries and divides files into various kinds, what improves the compression performance in latter stage. TWRT also recognizes multilingual text files, but operates with almost the same speed as its predecessor – WRT. TWRT in connection with a popular bzip2 and modern compressors PAQ6 and PPMonstr improves the compression performance by about 5–8%. Moreover, TWRT improves the compression speed with PAQ6 and on larger files with PPMonstr.

1 Introduction

There are plenty of papers about universal compression, and nowadays it is very hard to improve this kind of algorithm. Several papers about specialized compression algorithms have appeared. They explore specific properties of text [4,5,17], XML [2], platform-dependent executable files [3] and record-aligned data [20].

In this work we decided to examine the impact of specialized modelling ideas on compression of textual files. There are two distinct approaches to the text compression. One is to design a "text aware" compressor; the other is to write a text preprocessor [5] (filter) which transforms the original input into a representation more redundant from the viewpoint of existing general-purpose compressors. Specialized text compressors are potentially more powerful, both from the viewpoint of compression performance and the compression speed at a given compression rate, as there are virtually no restrictions imposed on the implemented ideas. Nevertheless, text preprocessing is more flexible, as the filtered text can be well compressible with most existing general-purpose compressors, so with relatively little programming effort various compression speed/decompression speed/compression ratio compromises can be achieved.

The basic idea of preprocessing is to transform the text into some intermediate form, which can be compressed more efficiently. This kind of algorithm is known as "preprocessing algorithm", and its output is used as input of any existing general-purpose compressor. The decompression process has two stages, decompression using given compressor and reverse preprocessing transformation.

Dictionary-based preprocessing [4] is a preprocessing technique, which gives the most significant improvement in the compression performance. It is based on the notion of replacing whole words with shorter codes. The dictionary of words is usually static (for a given language) and given in advance, as the success of building the dictionary "on the fly" is possible only on very large input data, and thus practicality of such a solution is doubtful.

The organization of this paper is as follows. In Section 2 we briefly review the related work, mostly focusing on WRT algorithm developed by Skibiński et al. [16], which was the starting point of our research. Section 3 contains the description of our new algorithm, stressing on the extensions to WRT. Section 4 reveals the details of our two-level dictionaries. The next section presents the comparative results of extensive experiments with several compressors from PPM (Prediction by Partial Matching) and BWCA (Burrows–Wheeler Compression Algorithms) families. Finally, we draw several conclusions and suggest perspectives for the future.

2 Related work

Several specialized text compressors have been presented. We would like to mention WordHuff [10], word-based LZW [6], word-based PPM [19] and BWT compression on words [7]. Text filtering ideas have been described by, among others, Kruse and Mukherjee [8], Teahan [19], Grabowski [5], Franceschini et al. [4], and Sun et al. [17].

WRT [16] is an English text preprocessor, a successor of StarNT [17]. WRT replaces words in input text file with shorter codewords and uses several other techniques to improve performance of latter compression. In terms of compression, it brings an improvement on the order of 8–12% in comparison with ordinary PPM or BWCA compressors and even about 21–27% with LZ77 (Lempel–Ziv) compressors.

A word in WRT dictionary is a sequence of at least one symbol over the alphabet [a..z]. There is no need to use upper case letters in the dictionary, as the scheme makes use of the "capital conversion" technique: there are two one-byte tokens (reserved symbols) in the output alphabet to indicate that either a given word starts with a capital letter while the following letters are all lowercase, or a given word consists of capitals only. Those tokens, t_{cl} and t_{uw}, respectively, are inserted after the word. Finally, there is yet a collision-handling token, t_{esc}, used for encoding occurrences of tokens t_{cl}, t_{uw}, and t_{esc} in the text.

The ordering of words in the dictionary D, as well as mapping the words to unique codewords, is important for the compression effectiveness. The dictionary is sorted according to the frequency of words as more frequent messages should be represented with shorter codes than less frequent messages. WRT English dictionary have 80,000 words. Each word in D has assigned a corresponding codeword. Codewords' length is variable and span from one to four symbols. Ordinary text files, at least English ones, consist solely of ASCII symbols not exceeding 127, so codewords' alphabet has 128

symbols (ASCII values from 128 to 255). If there is a symbol from codewords' alphabet in the input file, then WRT outputs token t_{esc} and this symbol. Codewords' alphabet (128 symbols) is divided into four separate parts. WRT uses the mapping <101, 9, 9, 9> for codewords, and thus there are $101 + 101 \cdot 9 + 101 \cdot 9 \cdot 9 + 101 \cdot 9 \cdot 9 \cdot 9 = 82{,}820$ distinct codewords available. It is enough for 80,000 words WRT dictionary. The codeword bytes are emitted in the reverse order, i.e., the range for the last codeword byte has always 101 values.

WRT uses several additional techniques to improve the compression performance. First is q-gram replacement, which is based on substituting frequent sequences of q consecutive characters, i.e., q-grams, with single symbols [19]. In WRT the sequence length q does not exceed 3. It is natural to represent q-grams with characters hardly ever used in plain texts. In WRT, however, the values above 127 in ASCII code are already used as the codeword alphabet for the word dictionary. On the other hand, a nice side-effect of WRT capital conversion variant is that it get rid from all the capital letters, and therefore those 26 values are also used for encoding q-grams.

The next technique that improves the compression performance is End-of-Line (EOL) coding. The general idea is to replace EOL symbols with spaces and to encode information enabling the reverse operation in a separate stream. EOL symbols are converted to spaces, and for each space in the text, WRT writes a binary flag, to distinguish an original EOL from a space. The flags are encoded with an arithmetic coder, on the basis of an artificial context. The information taken into account to create the context involves the preceding symbol, the following symbol, and the distance between the current position and the last EOL symbol. WRT appends the EOL stream at the end of the processed file.

WRT presents a binary data filter technique to protect from a significant loss on data, which do not fit to its model (in particular, binary data). In WRT, the encoder (and the decoder) keep count of input characters from the textual, c_t, and nontextual, c_n, alphabet range. If $4 \cdot c_n < c_t$, then a special symbol p from the input is translated into two bytes: t_{esc} and p. In the opposite case (typical for binary files), the token t_{esc} is sent, and it is assumed that r successive characters are also non-textual. In other words, t_{esc} is used to mark the beginning of a non-textual data chunk, rather than encoding a single non-textual character. In the non-textual regime, the read characters are instantly sent to the output. The encoder, however, switches back to the textual regime if r preceding characters are all textual.

The last technique used by WRT is surrounding words with spaces, which converts all words to be surrounded by space characters. This technique gives gain only if there are at least a few occurrences of the word, because it joins similar contexts in PPM compressor (helps in better prediction of the word's first symbol as well as the next symbol just after the word). It is hard to predict if surrounding words with spaces should be used, but this technique works with the majority of large text files. WRT sets the minimum file size threshold for switching on this idea to 1 MB.

3 Two-level Word Replacing Transformation

The source file in any programming language, e.g., C, C++, Java, can be divided into a programming language commands and comments. To improve the compression

performance, WRT replaces comments in English language with shorter codewords. If we add the programming language commands to the dictionary, then the compression performance on source files should be improved, but the compression performance on the remaining files should be deteriorated. Based on this observation, we have developed Two-level Word Replacing Transformation (TWRT [15]), improvement of Word Replacing Transformation (WRT).

TWRT can use up to two dictionaries, which are dynamically selected before the actual preprocessing starts. The first level dictionaries (small dictionaries) are specific for some kind of data (e.g., programming language, references). The second level dictionaries (large dictionaries) are specific for natural language (e.g., English, Russian, French). If no appropriate first level dictionary is found, then the first level dictionary is not used. Selection of the second level dictionary is analogous. When TWRT has selected only the one (the first or the second level) dictionary, it works like WRT. If TWRT has selected both: the first and the second level dictionary, then the dictionaries are automatically joined (the second level dictionary is appended after the first level dictionary). If the same word exists in the first and the second level dictionary, then the second appearance of word is ignored to minimize length of codewords. Only the names of the dictionaries are written in the output file, so the decoder can use the same combination of the dictionaries.

The main problem for TWRT is selection of the dictionaries before preprocessing. The simplest idea is to use multi-pass preprocessing to preprocess the input file step by step with all dictionaries and finally to choose the smallest output file. Nevertheless, this idea is very slow. We propose to read only the first f (e.g., 250) words from each of n dictionaries (small and large) and create one, joined dictionary (only in computer's memory). The joined dictionary consists of n parts (from n dictionaries). If there is the same word in different dictionaries, then all occurrences of this word are skipped. When we have created the joined dictionary, we have to preprocess the input file only with this dictionary. While preprocessing we should only count occurrences of words in n parts of the joined dictionary. In the end, we have to find the large dictionary with the biggest count of words' occurrences and the small dictionary with the biggest count of words' occurrences. These dictionaries, with big probability, will give the best results in latter compression. We call this technique "dictionaries detection". Moreover, we propose, what is experimentally checked, that the small dictionary should not be used if count of words for the large dictionary is greater than five times count of words for the small dictionary.

WRT is designed to be a one-pass preprocessor, but dictionaries detection mechanism in TWRT needs an additional preprocessing pass. To improve the preprocessing speed, we can preprocess only some part of the input file (e.g., 50 kB). This can cause that TWRT will sometimes select (on heterogeneous files) wrong dictionaries. A better idea is to read data from random part of the input file. TWRT uses another idea: it divides file into several (e.g., 5) equal blocks and reads only a few kB (e.g., 10 kB) from each block. This technique is called "faster dictionaries detection" as it speed ups in high degree the preprocessing speed.

Some languages use Latin alphabet with added national characters (e.g., French, German, Polish). These characters use ASCII values over 127, so they should not be used as TWRT codewords alphabet. Moreover there are several encoding standards for

national characters (e.g., cp850 and ISO 8859-1 for German language). For each word from the joined dictionary, if the word includes one or more national characters, TWRT adds to the joined dictionary this word in all possible encodings. To find the suitable dictionary, it is necessary to use words in all encodings as for example Russian language does not use Latin alphabet at all. TWRT selects large dictionary with the most probable encoding, which is used later, when it reads entire large dictionary.

"Faster dictionaries detection" stage gives additional opportunities. TWRT can decide if the binary data filter, surrounding words with spaces, and EOL coding should be used in latter preprocessing stage. It also gives possibility to find record-based files. Record preprocessing [1] is a preprocessing technique, which exploits the structure of record-based files. The scheme detects symbol repetitions that occur at the same position inside a sequence of records with fixed length. TWRT uses record preprocessing and transposes the record-based files by the record length. Record preprocessing is not used in connection with any other preprocessing technique.

4 The first and the second-level dictionaries

We have collected sets of files of each kind to create the dictionaries. Each dictionary is created by counting words' occurrences in set (e.g., 3 GB English text files). The dictionary is sorted with words' frequency (starting from greater). Full description is available in Table 1. TWRT uses six small dictionaries (C++, CompSc, Lisp, Math, Politics, Ref) and five large dictionaries (DE, ENG, FR, PL, RU).

Dictionary file name	Short name	File size in bytes	Number of words	Description
wrt-short-c++.dic	C++	700	85	Most frequent C++ words
wrt-short-comp_sc.dic	CompSc	164,381	18,596	Computer science dictionary
wrt-short-lisp.dic	Lisp	629	74	Most frequent LISP & Pascal words
wrt-short-math.dic	Math	196,368	21,301	Words from Encyclopedia of Math
wrt-short-politics.dic	Politics	134,127	16,845	Based on CIA World Factbooks
wrt-short-ref.dic	Ref	68,940	8,546	References
wrt-de.dic	DE	463,918	45,599	German dictionary
wrt-eng.dic	ENG	1,023,195	107,680	English dictionary
wrt-fr.dic	FR	648,314	67,667	French dictionary
wrt-pl.dic	PL	1,479,847	163,843	Polish dictionary
wrt-ru.dic	RU	4,954,251	473,524	Russian dictionary

Table 1. Detailed information about the first and the second-level dictionaries

5 Experimental results

We carried out the experiments to compare TWRT with its predecessor, WRT, and available dictionary-based compressors: Durilca [14] and RKC [18]. We have used TWRT and WRT in connection with bzip2 [11] and modern compressors PAQ6 [9] and PPMonstr [13], which are currently at the top when we look at the compression performance.

Durilca 0.3a, created by Shkarin in 2003, is general-purpose compressor. It is essentially PPMII [12] (PPM variation) with built-in models (selected for given data via a data detection routine). If the model fits to data (e.g., to English text), then PPM starts compression with lots of useful knowledge. What makes this scheme similar to the preprocessing approach is employing the PPM engine directly without any modification, a very attractive feature. What makes it different to preprocessing techniques, and is another benefit, is that Shkarin's idea needs a single pass over input data, while the "preprocess-and-compress" approach requires (at least) two passes. Durilca, together with its faster version, Durilca Light, achieve excellent compression, but the main drawback of this approach is limitation to PPM family. Only English and Russian built-in models are available. In our experiments we have used Durilca with options "–t2 –m200 –o16" (what means: turn on built-in models, set memory limit to 200 MB, and use PPM order 16 respectively).

RKC 1.02 is general-purpose compressor with built-in English, Russian and Polish dictionaries. It is based on PPMZ with added variation of StarNT transformation. Unfortunately for our experiments, preprocessing stage and compression stage can not be separated in RKC, so we can not use RKC only as preprocessor. We have used RKC with no additional options.

All compressors, used in the experiments, are based on character based PPM and its variations, except PAQ6 v2, which is binary based PPM, and bzip2 1.0.2, a popular BWT based compressor. PPMonstr var. I implements PPMII compression algorithm. Durilca 0.3a can be considered as PPMonstr var. I with added preprocessing and built-in models (it is not true for Durilca 0.4 and newer, where the compression algorithm has been modified). We have chosen PAQ6 and PPMonstr as they yield the best compression rates among the compressors with no preprocessing techniques. It assures that our transformation routines in TWRT and WRT do not interfere with possible preprocessing used by the compressor. We have used PPMonstr with options "–m200 –o16" (what means: set memory limit to 200 MB and use PPM order 16 respectively), PAQ6 with option "–6" (usage of 202 MB memory) and bzip2 with option "–9" (maximum compression).

TWRT has separate optimizations for four compression algorithms: LZ77, BWT, PPM and PAQ. The optimizations for the compression algorithm implemented in the compressor (used after the preprocessing stage) should be selected in TWRT. It gives the best compression performance.

All tests were carried out on an Athlon 750 MHz machine equipped with 256 MB RAM, under Windows 98 SE operating system.

5.1 The Calgary Corpus experiments

We have carried out our experiments on well-known set of files – the Calgary Corpus[1]. Because all the files in Calgary Corpus are English or binary, TWRT uses only English large dictionary. Entire improvement over WRT is from usage of small dictionaries or record preprocessing. As can be seen in Table 2, TWRT with PAQ6 gives the best average compression performance. It beats the previous best – WRT with PAQ6. It has the best compression on 7 from 14 files and is very close on remaining 7 files. TWRT

[1] http://corpus.canterbury.ac.nz/descriptions/#calgary

improves the compression performance for more than 5%, comparing PAQ6 and TWRT + PAQ6. The main problem with PAQ6 is that it is very slow (6 kB/s) and high memory consuming. Among faster and not so memory consuming (character based) compressors, TWRT + PPMonstr give the best average compression performance at almost the same speed as PPMonstr alone. TWRT improves the compression performance for about than 5%, comparing PPMonstr and TWRT + PPMonstr. In Calgary Corpus experiments, TWRT with PPMonstr beat Durilca, which can be considered as PPMonstr with added preprocessing and built-in models.

	PAQ6	WRT + PAQ6	TWRT (PAQ) + PAQ6	PPMonstr	WRT + PPMonstr	TWRT (PPM) + PPMonstr	Durilca	RKC	Dictionaries used by TWRT
bib	1.617	1.519	**1.476**	1.663	1.553	1.501	1.542	1.647	Ref, ENG
book1	2.090	1.894	1.900	2.119	1.912	1.913	**1.853**	2.048	ENG
book2	1.691	**1.583**	1.597	1.742	1.611	1.614	1.649	1.723	CompSc, ENG
geo	**3.536**	3.537	3.542	3.869	3.872	3.831	3.870	3.674	
news	2.034	**1.919**	**1.919**	2.084	1.939	1.937	1.956	2.086	ENG
obj1	3.047	**3.011**	3.041	3.345	3.301	3.330	3.349	3.234	CompSc, ENG
obj2	**1.703**	1.717	**1.703**	1.898	1.912	1.898	1.899	1.803	
paper1	2.052	1.829	**1.770**	2.122	1.909	1.842	1.883	2.087	CompSc, ENG
paper2	2.056	1.798	1.788	2.112	1.848	1.836	**1.772**	2.062	ENG
pic	0.456	**0.445**	0.448	0.693	0.693	0.455	0.693	0.683	
progc	2.031	1.947	**1.919**	2.106	2.027	1.989	2.022	2.119	C++, ENG
progl	1.314	1.253	**1.246**	1.352	1.296	1.283	1.382	1.378	Lisp, ENG
progp	1.312	1.294	**1.283**	1.360	1.346	1.327	1.409	1.462	Lisp, ENG
trans	1.126	1.091	1.104	1.151	1.125	1.156	**1.067**	1.155	Ref, ENG
average	1.861	1.774	**1.766**	1.972	1.881	1.851	1.881	1.940	
ctime	488.6	382.8	389.4	25.0	26.3	26.7	26.6	76.2	

Table 2. Detailed comparison between WRT and TWRT on the Calgary Corpus. Results are given in bits per character (bpc). The compression times (ctime) are given in seconds.

5.2 Experiments with multilingual text files

We have selected multilingual text files from Project Gutenberg[2] as there is no well-known corpus with multilingual text files. Because all the selected files, in these experiments, are books, TWRT does not use small dictionaries at all. WRT uses only English dictionary so improvement on latter compression is relatively small, and we did not include the WRT results. As can be seen in Table 3, again TWRT with PAQ6 gives the best average compression performance. It has the best compression on 16 from 22 files. TWRT improves the compression performance for about 8%, comparing PAQ6 and TWRT + PAQ6. From faster and not so memory consuming (character based) compressors, TWRT + PPMonstr give the best average compression performance. TWRT improves the compression performance for over than 8%, comparing PPMonstr and TWRT + PPMonstr. With bzip2, TWRT improves the compression performance by about 8%. With both: PAQ6 and PPMonstr, TWRT improves the compression speed as compressor's input is smaller after preprocessing. It was not true for PPMonstr in the

[2] http://www.gutenberg.org/.

previous experiment as the Calgary Corpus files are smaller and faster dictionaries detection takes proportionally more time. The compression speed with bzip2 and TWRT + bzip2 is almost the same.

	language/ encoding	bzip2	TWRT (BWT) + bzip2	PAQ6	TWRT (PAQ) + PAQ6	PPMonstr	TWRT (PPM) + PPMonstr	Durilca	RKC
10055-8.txt[3]	German/cp850	2.312	2.095	1.910	**1.812**	1.946	1.826	1.914	1.868
10055-8.txt	German/ISO 8859-1	2.312	2.097	1.906	**1.811**	1.946	1.827	1.881	1.867
12267-8.txt[4]	German/cp850	2.302	2.107	1.943	**1.815**	1.986	1.829	1.956	1.974
12267-8.txt	German/ISO 8859-1	2.302	2.108	1.939	**1.815**	1.986	1.829	1.921	1.973
1musk10.txt[5]	English/ASCII	2.080	1.769	1.718	1.569	1.737	1.571	**1.494**	1.649
plrabn12.txt[6]	English/ASCII	2.417	2.211	2.104	1.968	2.138	1.982	**1.963**	2.123
11176-8.txt[7]	French/cp850	2.015	1.760	1.699	1.549	1.712	**1.548**	1.641	1.600
11176-8.txt	French/ISO 8859-1	2.017	1.759	1.694	1.553	1.712	**1.549**	1.596	1.599
11645-8.txt[8]	French/cp850	2.385	2.057	2.024	**1.815**	2.052	1.828	2.046	1.951
11645-8.txt	French/ISO 8859-1	2.386	2.057	2.019	**1.821**	2.052	1.831	1.982	1.950
rnpz810.txt[9]	Polish/cp1250	2.600	2.350	2.245	**2.069**	2.274	2.088	2.247	2.193
rnpz810.txt	Polish/cp852	2.599	2.348	2.246	**2.067**	2.274	2.086	2.297	2.193
rnpz810 .txt	Polish/ISO 8859-2	2.603	2.351	2.245	**2.067**	2.274	2.086	2.221	2.192
sklep10.txt[10]	Polish/cp1250	3.022	2.599	2.602	**2.281**	2.663	2.349	2.599	2.564
sklep10.txt	Polish/cp852	3.017	2.598	2.603	**2.280**	2.663	2.347	2.663	2.565
sklep10.txt	Polish/ISO 8859-2	3.020	2.599	2.601	**2.279**	2.663	2.347	2.558	2.564
master.txt[11]	Russian/cp1251	2.572	2.305	2.196	**2.036**	2.241	2.063	2.134	2.265
master.txt	Russian/cp866	2.572	2.311	2.202	2.039	2.241	2.063	**1.874**	2.271
master.txt	Russian/KOI8-R	2.574	2.307	2.197	**2.037**	2.241	2.063	2.134	2.265
misteria.txt[12]	Russian/cp1251	2.390	2.256	1.974	**1.929**	2.045	1.986	1.941	2.070
misteria.txt	Russian/cp866	2.390	2.261	1.985	1.933	2.045	1.986	**1.773**	2.075
misteria.txt	Russian/KOI8-R	2.391	2.259	1.972	**1.929**	2.045	1.986	1.941	2.070
average		2.467	2.207	2.091	**1.930**	2.133	1.956	2.035	2.083
ctime		19.2	19.8	2201.7	1462.5	113.4	107.7	117.6	323.2

Table 3. Detailed comparison between WRT and TWRT on multilingual text files. Results are given in bits per character (bpc). The compression times (ctime) are given in seconds.

Concluding multilingual text files experiments, RKC uses English, Russian and Polish dictionaries, but they did not work well (especially Russian and Polish). TWRT with PPMonstr is on average better than Durilca. Durilca has the best compression performance on English and Russian files, but Russian built-in model works only with codepage cp866. There are, however, a few TWRT advantages over Durilca. First, Durilca's built-in model is limited to PPM compression and TWRT is ready to immediate

[3] http://www.gutenberg.net/1/0/0/5/10055/10055-8.txt
[4] http://www.gutenberg.net/1/2/2/6/12267/12267-8.txt
[5] http://www.gutenberg.net/etext98/1musk10.txt
[6] http://www.gutenberg.net/etext92/plrabn12.txt
[7] http://www.gutenberg.net/1/1/1/7/11176/11176-8.txt
[8] http://www.gutenberg.net/1/1/6/4/11645/11645-8.txt
[9] http://www.gutenberg.net/etext04/rnpz810.txt
[10] http://www.gutenberg.net/etext05/sklep10.txt
[11] http://www.lib.ru/BULGAKOW/master.txt
[12] http://www.lib.ru/URIKOVA/misteria.txt

use with any compressor (based on LZ77, BWT, PPM, or PAQ). Second, to add a new language to TWRT we do not have to modify TWRT at all. We only have to create an additional dictionary.

6 Conclusions

In this paper we have presented dictionary-based preprocessing technique, which is called "two-level dictionaries" and its implementation TWRT (Two-level Word Replacing Transformation). The first level dictionaries (small dictionaries) are specific for some kind of data (e.g., programming language, references). The second level dictionaries (large dictionaries) are specific for natural language (e.g., English, Russian, French). TWRT chooses the best combination of the dictionaries, one small and one large dictionary. Our preprocessor uses "faster dictionaries detection" mechanism before ordinary preprocessing. This technique gives additional opportunities. TWRT can decide if binary data filter, surrounding words with spaces, EOL coding, and record preprocessing should be used in ordinary preprocessing.

TWRT, comparing to its predecessor – WRT, divides files on various kinds, what improves the compression performance (after preprocessing). Our preprocessor also recognizes multilingual text files, but operates with almost the same speed. On the Calgary Corpus, TWRT improves the compression performance of PPMonstr, by about 6% on average. Even for the top compressor nowadays, PAQ6, the gain is significant – 5%. On the multilingual text files, gain is even bigger. TWRT improves the compression performance of bzip2, a popular BWT based compressor, by about 8%. With PPMonstr and PAQ6 the gain is 8% on average. Moreover, TWRT improves the compression speed with PAQ6 and on larger files with PPMonstr.

Please note that TWRT can be used in spell checking. It has multilingual dictionaries and can help a spell checker with automatically detecting language of text file. The preprocessor and the spell-checker can share the same dictionaries. Spell checker's dictionaries are probably bigger than adequate TWRT dictionaries so some last words from the dictionaries will not be used in preprocessing.

TWRT can be used as an encoding converter in text files. It outputs postprocessed files with the same encoding as input file, but it can be easily changed. A more interesting idea is adding national characters to multilingual text files with stripped national characters. For this kind of files, TWRT reads proper dictionary and strips national characters from it. If we find words in stripped dictionary, then we can output original words, with national characters. But there is a problem with words that have the same form after stripping, e.g., Polish words "sad" and "sąd". We do not know if "sad" should be interpreted as "sąd" or "sad". It is an interesting subject for further research.

Acknowledgements

The author wishes to thank Szymon Grabowski for great help in improving this document.

Bibliography

[1] J. Abel. Record preprocessing for data compression. *Proceedings of the IEEE Data Compression Conference 2004*, Snowbird, Utah, pp. 521.

[2] J. Adiego, G. Navarro and P. de la Fuente. Lempel–Ziv Compression of Structured Text. *Proceedings of the IEEE Data Conference 2004*, Snowbird, Utah, pp. 112–121.

[3] M. Drinić and D. Kirovski. PPMexe: PPM for Compressing Software. *Proceedings of the IEEE Data Compression Conference 2002*, Snowbird, Utah, pp. 192–201.

[4] R. Franceschini, H. Kruse, N. Zhang, R. Iqbal and A. Mukherjee. Lossless, Reversible Transformations that Improve Text Compression Ratios. Preprint of the M5 Lab, University of Central Florida, 2000.

[5] Sz. Grabowski. Text Preprocessing for Burrows–Wheeler Block–Sorting Compression. *VII Konferencja Sieci i Systemy Informatyczne – Teoria, Projekty, Wdrożenia*, Łódź, Poland, 1999.

[6] N. Horspool and G. Cormack. Constructing Word–Based Text Compression Algorithms. *Proceedings of the IEEE Data Compression Conference 1992*, Snowbird, Utah, pp. 62–71.

[7] R. Y. K. Isal, A. Moffat and A. C. H. Ngai. Enhanced Word-Based Block-Sorting Text Compression. *Proceedings of 25th Australasian Computer Science Conference*, Melbourne, pp. 129–138, 2002.

[8] H. Kruse and A. Mukherjee. Preprocessing Text to Improve Compression Ratios, *Proceedings of the IEEE Data Compression Conference 1998*, Snowbird, Utah, pp. 556.

[9] M. V. Mahoney. The PAQ6 data compression program. 2004. http://www.cs.fit.edu/~mmahoney/compression/.

[10] A. Moffat. Word based text compression, *Software–Practice and Experience*, 19 (2), pp. 185–198, 1989.

[11] J. Seward, bzip2 1.0.2 (computer program). 2002. http://sources.redhat.com/bzip2/.

[12] D. Shkarin. PPM: one step to practicality. *Proceedings of the IEEE Data Compression Conference 2002*, Snowbird, Utah, pp. 202–211.

[13] D. Shkarin. PPMd and PPMonstr var. I (computer programs). 2002. http://compression.graphicon.ru/ds/.

[14] D. Shkarin. Durilca Light and Durilca (computer program). 2004. http://compression.graphicon.ru/ds/.

[15] P. Skibiński. TWRT (Two-level Word Replacing Transformation) sources. 2004. http://www.ii.uni.wroc.pl/~inikep/research/WRT40.zip.

[16] P. Skibiński, Sz. Grabowski and S. Deorowicz. Revisiting dictionary-based compression. To appear in *Software–Practice and Experience*, DOI: 10.1002/spe.678, 2005. http://www.ii.uni.wroc.pl/~inikep/papers/05-RevisitingDictCompr.pdf.

[17] W. Sun, A. Mukherjee and N. Zhang. A Dictionary-based Multi-Corpora Text Compression System. *Proceedings of the IEEE Data Compression Conference 2003*, Snowbird, Utah, pp. 448.

[18] M. Taylor. RKC v1.02 (computer program). March 2004. http://www.msoftware.co.nz.

[19] W. Teahan. Modelling English text. Ph.D. dissertation. Department of Computer Science, University of Waikato, New Zealand, 1998.

[20] B. D. Vo and K.-P. Vo. Using Column Dependency to Compress Tables. *Proceedings of the IEEE Data Compression Conference 2004*, Snowbird, Utah, pp. 92–101.