Text Preprocessing for Data Compression

Jürgen Abel and William Teahan

***Abstract*—Several preprocessing algorithms for text files are presented which are performed prior to the compression scheme. The overall gain of the compression rate when all the preprocessing algorithms are combined is compared between the BWT, PPM and LZ based compression schemes. The algorithms need no external dictionary and are language independent. The average overall compression gain is in the range of 3 to 5 percent for the text files of the Calgary Corpus and between 2 to 9 percent for the text files of the large Canterbury Corpus.**

***Index Terms*—algorithms, data compression, BWT, LZ, PPM, preprocessing, text compression.**

I. Introduction

TODAY the most popular schemes for lossless data compression are the Burrows-Wheeler Compression Algorithm (BWCA) [1], Prediction by Partial Matching (PPM) [2] and Lempel-Ziv (LZ) [3] based compression schemes. The first two schemes are context related, whereas the LZ scheme is based on repetitions. Even though each of these schemes can be used to compress text files, they do not consider the special properties of textual data, and the compression rate can be enhanced by using preprocessing algorithms specialized for textual data. Text preprocessing algorithms are reversible transformations, which are performed before the actual compression scheme during encoding and afterwards during decoding. Since textual data make up a substantial part of the Internet and other information systems, efficient compression of textual data is of significant practical interest.

This paper presents a number of text preprocessing algorithms, including a text recognition scheme and five separate algorithms: capital letter conversion, EOL coding, word replacement, phrase replacement and alphabet reordering. The basic ideas of many of these algorithms have already appeared in many variations. The approach presented here is a combination of strongly improved and more universal algorithms, which in contrast to many others need no fixed external information like dictionaries. It is language independent – as long as the text is based on Latin letters. Nevertheless, it achieves a remarkable compression gain. The first four algorithms work with most universal compression schemes, the last algorithm is especially designed for sort based schemes like the BWCA.

The impact of the text preprocessing algorithms is illustrated using the example of the 10 text files of the Calgary Corpus – `bib`, `book1`, `book2`, `news`, `paper1`, `paper2`, `progc`, `progl`, `progp` and `trans` – for BWCA, PPM and LZ compression schemes. The BWCA is represented by the program ABC 2.4 from Abel [4], the PPM algorithm is represented by the program PPMD from Teahan [5] and for the LZ scheme the program GZIP from Gailly is used [6].

II. Previous Work

The preprocessing of textual data is a subject of many publications. In some articles, the treatment of textual data is embedded within the compression scheme itself but could easily be separated into two independent parts: a preprocessing algorithm and a standard compression algorithm, which are processed sequentially one after the other.

Bentley et al. describe a word based compression scheme [7], where words are replaced by an index into an MTF list. The dictionary of the words is transmitted implicitly by transmitting the word during its first occurrence. This scheme can be divided into a parsing preprocessing part and a standard MTF ranking scheme. A word based variation of the PPM scheme is presented by Moffat [8]. He uses order-0, order-1 and order-2 word models to achieve better compression than the MTF scheme from Bentley et al. Similar schemes, which differentiate between alphanumeric strings and punctuation strings, and which also use an implicit dictionary, are presented by Horspool and Cormack [9]. Again, these schemes can be divided into a parsing part and a coding part using Huffman codes.

Teahan and Cleary describe several methods for enlarging the alphabet of the textual data [10]. Besides the replacement of common bigrams by a one symbol token, they propose methods for encoding special forms of bigrams called digrams (two letters representing a single sound as *ea* in "bread" or *ng* in "sing"). The replacements are processed using a fixed set of the frequently used bigrams in the English language, which makes this attempt language dependent. Teahan and Cleary [11] describe a word based compression scheme where the word dictionary is adaptively built from the already processed

· Jürgen Abel, University Duisburg-Essen, Department "Communications Systems", Faculty of Engineering Sciences, Bismarckstrasse 81, D-47057 Duisburg, Germany. E-mail: juergen.abel@acm.org.

· William Teahan, School of Informatics, University of Wales, Bangor, Gwynedd LL57 1UT, United Kingdom. E-mail: wjt@informatics.bangor.ac.uk.

input data. This can also be achieved by a preprocessing stage if the words are replaced by corresponding tokens. Teahan presents a further comparison between two different word based compression schemes in his PhD thesis [12]. The first scheme uses function words, which include articles, prepositions, pronouns, numbers, conjunctions, auxiliary verbs and certain irregular forms. The second scheme uses the most frequently used words in the English language. Both schemes require external dictionaries and are language dependent.

A special case of word encoding is the star encoding method from Kruse and Mukherjee [13]. This method replaces words by a symbols sequence that mostly consist of repetitions of the single symbol '*'. This requires the use of an external dictionary that must be known by the receiver as well as the sender. Inside the dictionary, the words are first sorted by their length and second by their frequency in the English language using information obtained from Horspool and Cormack [9]. All sorted words of the same length are then encoded by sequences "*…*", "A*…*", … , "Z*…*", "a*…*", …, "z*…*", "*A*…*", … where the length of the encoded sequence is equal to the length of the word being encoded. The requirement of an external dictionary makes this method again language dependent.

Preprocessing methods, specialized for a specific compression scheme, are presented by Chapin and Tate [14] and later by Chapin [15]. They describe several methods for alphabet reordering prior to using the BWCA in order to place letters with similar contexts close to one another. Since the Burrows-Wheeler transformation (BWT) is a permutation of the input symbols based on a lexicographic sorting of the suffices, this reordering places areas of similar contexts at the BWT output stage closer together, and these can be exploited by the latter stages of the BWCA. The paper compares several heuristic and computed reorderings where the heuristic approaches always achieve a better result on text files than the computed approaches. The average gain for BWCA using heuristic reorderings over the normal alphabetic order was 0.4% on the text files of the Calgary Corpus. Balkenhol and Shtarkov use a very similar heuristic alphabet reordering for preprocessing with BWCA [16]. A different alphabet reordering for BWCA is used in the paper from Kruse and Mukherjee [17]. It also describes a bigram encoding method and a word encoding method which is based on their star encoding.

Grabowski proposes several text preprocessing methods in his publication [18], which focuses on improvements for BWCA but some techniques can also be used for other compression schemes. Besides the already mentioned techniques like alphabet reordering, bigram-, trigram- and quadgram replacement, Grabowski suggests three new algorithms. The first one is capital conversion. An escape symbol and the corresponding lower letter replace capital letters at the beginning of a word. If the second letter of the word is capitalized too, the replacement is omitted. This technique increases context dependencies and similarities between words, which can be exploited by standard compression schemes. The second algorithm is space stuffing, where a space symbol is placed at the beginning of each line in order to change the context that follows the end of line symbol (EOL) to one space instead of various symbols. The last algorithm is EOL coding, which replaces EOL symbols by space symbols and separately encodes the former EOL positions, which is represented by the number of blanks since the previous EOL symbol. These numbers are encoded either within the symbol stream itself or in a separate data stream. Grabowski suggests using either space stuffing or EOL coding for preprocessing text files, but because of unstable side-effects, he decides to omit EOL coding in his comparisons. His preprocessing algorithms without EOL coding achieve an average gain for BWCA of 2.64% on the 10 text files of the Calgary Corpus. Since he uses a set of fixed bigrams, trigrams and quadgrams, his proposal requires an external dictionary and is language dependent.

Franceschini et al. extend the star encoding method by using different schemes for the indices into the dictionary [19], called Length-Preserving Transform (LPT), Reverse Length-Preserving Transform (RLPT) and Shortened-Context Length-Preserving Transform (SCLPT). All of these require an external dictionary and are language dependent. Franceschini reported for SCLPT, which achieves the best results, a gain for BWCA of 7.1% and for PPMD+ a gain of 3.8% for the files of the Calgary Corpus (including the files `paper3`, `paper4`, `paper5` and `paper6`). A further improvement of the star encoding method, presented by Awan et al. [20], is called Length Index Preserving Transform (LIPT). LIPT encodes a word as a string that can be interpreted as an index into a dictionary. The string consists of three parts: a single symbol '*', a symbol between 'a' and 'z', and a sequence of symbol from the set 'a'…'z', 'A'…'Z'. The second part of the string, the single symbol, represents the length *l* of the word, where 'a' stands for length 1 and 'z' for length 26. The third part is the encoded index inside the set of words with length *l*. They are encoded as a number representation of base 52 decremented by 1, where 'a' represents 0, …, 'z' represents 25, 'A' represents 26, …, and 'Z' represents 51. An empty substring represents the number 0. Therefore, a word of length 3 with index 0 is encoded as "*c", a word of length 3 with index 1 as "*ca", a word of length 3 with index 27 as "*cA" and so on. LIPT achieves a gain for BWCA on the Calgary Corpus of 4.1% and of 5.6% for GZIP.

Isal and Moffat present different text preprocessing schemes for bigrams and words [21] using internal and external dictionaries. In their paper, tokens are used with values above 255, so they can be used together with normal symbols, as the compression scheme needs to handle alphabets with more than 8 bits. For text files, the word based schemes with internal dictionaries give the highest compression gain. Later Isal et al. combine the word preprocessing scheme with different global structure transformations and entropy coding schemes [22]. Because of the use of an internal dictionary, where each word is spelt out the first time it occurred, the schemes of Isal and Moffat are all language independent.

Teahan and Harper propose a switching algorithm for combining both dynamic and static PPM models that also involves an initial text preprocessing step [23]. In this step that occurs prior to the encoding step, the text is essentially marked up by additional switch symbols to indicate when the compression algorithm should switch to another model. A greedy search algorithm which minimizes the overall code length of the encoded stream (of both the original symbols and additional switch symbols) is used to determine the positions of the markup symbols. This scheme is only relevant to context based schemes such as PPM, and it requires a modification of the subsequent PPM compression scheme.

III. Recognizing Text Files

Preprocessing algorithms are specifically designed for the properties of textual data. Since other types of data like pictures or numerical data have different statistical properties, using a text preprocessing algorithm on non textual data leads to a misinterpretation of the context and to worse compression rates. Therefore, a simple text recognition scheme prior to the text preprocessing step is presented, which has the task of classifying files as text files or non-text files respectively. It is assumed that the text files are based on Latin letters represented by the normal ASCII character set, supplemented if needed by some language dependent characters above ASCII 127, like the Umlaute 'ä', 'ö' 'ü' in German for example.

In order for a file to be categorized as a text file, it has to fulfill two requirements:

1. The percentage frequency share of letters (A,..., Z, a,..., z), digits (0,..., 9) and the space symbol compared to all symbols should be greater than 66%.
2. The percentage frequency share of the space symbol compared to letters (A,..., Z, a,..., z) and digits (0,..., 9) should be greater than 10%.

The first condition confirms that letters, digits and spaces are adequately represented inside the file, since these symbols dominate in normal text over punctuation symbols like ',' and non printable symbols like bell (0x07). By the second condition is taken into account that the space symbol is a very frequent symbol, for example as a delimiter of words.

This text recognition scheme categorizes the following 10 files of the Calgary Corpus as text files: `bib`, `book1`, `book2`, `news`, `paper1`, `paper2`, `progc`, `progl`, `progp` and `trans`. The files `geo`, `obj1`, `obj2` and `pic` are classified as non-text files.

Clearly, the assumptions on which these conditions are taken are a great simplification because of the lack of an official definition of a text file. This simple approach can lead in some cases to false classifications; furthermore, languages not based on Latin letters, like Russian and Chinese, have not been taken into account. However, the advantage of the approach is its simplicity and speed, and experiments show that the method is extremely effective across a wide range of sample text files.

IV. Universal Preprocessing Algorithms

A. Capital Letter and Upper Word Conversion

In order to improve context similarities, Grabowski suggests replacing capital letters at the beginning of a word by a capital-letter escape symbol and the corresponding lower letter [18]. If the second letter of the word is capitalized too, the replacement is omitted.

Table I displays the result of this technique on the text files of the Calgary Corpus for the three investigated compression schemes. BWCA and PPM get a small compression boost on average, whereas the LZ based scheme is hampered. The file `trans` especially shows a much worse compression rate by this technique for all three compression schemes.

In this paper, a different technique is revealed for the first time. In the first phase, a ternary search tree (a tree for which every node represents a symbol and has up to three edges for: less than, equal, or greater than [24]) is constructed containing all words of the text with length of size 2 or more. Hereto a word is defined as a sequence of lower or upper case letters surrounded by non-letter symbols. In the second phase, only words that start with a capital letter and occur with a lowercase letter elsewhere are considered for capital letter conversion. This phase ensures that words that always start with a capital letter are unconverted, since this conversion would decrease context similarities. Additionally, it is checked if the second letter is a lowercase letter. Only if both conditions are fulfilled, then the capital letter is converted into a capital-letter escape symbol, followed by a space symbol and the corresponding lowercase letter. Even though the original symbol is now replaced by three symbols, there is a gain on average for the text files of the Calgary Corpus, since the capital-letter escape symbol has a unique suffix consisting of the space character, which supports context-based

TABLE I
Percentage gain for simple capital letter algorithm

File	BWCA % gain	PPM % gain	LZ % gain
`bib`	0.12	-1.34	-0.20
`book1`	0.30	0.16	0.17
`book2`	0.38	0.26	0.30
`news`	0.26	-0.42	-0.13
`paper1`	0.35	0.70	0.34
`paper2`	0.33	0.59	0.55
`progc`	0.55	0.39	0.18
`progl`	0.15	0.12	0.02
`progp`	-0.36	-0.36	-0.52
`trans`	-0.72	-1.35	-4.84
Avg.	**0.14**	**-0.13**	**-0.41**

algorithms.

A further enhancement is achieved if words, consisting only of uppercase letters, are converted into a capital-word escape symbol followed by the word in lowercase letters. This word conversion is processed only if the number of lowercase letters inside the text is larger than the number of uppercase letters. This test ensures that files, consisting mainly of upper letters, are unchanged.

Both conversions have the goal of supporting the compression scheme by recognizing similarities between words with lower and upper letters, since plain compression schemes are unable to recognize the relationship between lowercase and uppercase letters.

The symbols for the capital-letter escape and capital-word-escape are not fixed escape symbols but are calculated as follows. In principle, any symbol, which has a frequency count of zero, could be used as an escape symbol. In order to save symbols with a zero frequency count for phrase and word tokens, a different approach is used instead. Inside normal text, uppercase letters rarely occur as a separate single letter (except the article 'A' and the personal pronoun 'I'), therefore this approach uses two uppercase letters as escape symbols. Hereto, the two uppercase letters are determined, which occur most rarely after a non-letter symbol. All occurrences inside the original text, where these two letters occur after a non-letter symbol, are replaced by two of the respective letters in order to be able to decode the escape symbols inside the decoder later. If for example the letters 'Q' and 'X' are chosen for the escape symbols, all occurrences of '.Q' and '.X' are replaced by '.QQ' and '.XX' respectively.

The two uppercase letters are transmitted as the first two symbols of the transformed text. Besides saving possible token symbols, using uppercase letters as escape symbols has the advantage that the escape symbols are sorted closely to the other upper letters, which plays a positive role for sort based compression algorithms such as BWCA.

Table II shows the impact of the improved capital letter and upper word conversion. All compression schemes now achieve a gain on average. The gain is greater for BWCA and PPM than for LZ, since BWCA and PPM are able to exploit context similarities whereas LZ schemes are based on simple repetitions.

B. EOL Coding

The coding of End-Of-Line (EOL) symbols is used in several archivers and described by Grabowski [18] who attributes the idea to Taylor. The principle of EOL coding is that EOL symbols in comparison to space symbols hamper the context, since words are usually divided by space symbols. Therefore, Grabowski suggests replacing all EOL symbols by space symbols and to encode their former positions. The positions are either transmitted within the normal symbol stream or as a separate data stream that is encoded using a different compression scheme.

Table III shows the impact of a simple EOL scheme, where the positions of the EOL symbols are encoded in a separate stream using the number of space symbols that occurs since the last EOL. The stream is encoded with variable length integer codes similar to the Elias codes [25]. The binary lengths of the integers are encoded first with an arithmetic coder. The bits of the integer are encoded with a binary arithmetic coder, with the most significant bit first. The size of the respective stream is listed in the second column of the table and added to the size of the compressed symbol stream for comparison. Except for file `book1`, which gets a compression gain between 1% and 2%, all files get a worse compression rate. Obviously, this scheme gives in some cases a noticeable gain, but needs to be enhanced before it will lead to reliable results.

TABLE II
PERCENTAGE GAIN FOR IMPROVED CAPITAL LETTER AND UPPER WORD ALGORITHM

File	BWCA % gain	PPM % gain	LZ % gain
`bib`	0.27	-0.57	-0.79
`book1`	0.49	0.26	0.27
`book2`	0.53	0.25	0.33
`news`	0.44	-0.66	0.01
`paper1`	0.66	0.75	0.37
`paper2`	0.70	0.79	0.50
`progc`	0.79	0.70	0.78
`progl`	0.78	0.42	0.78
`progp`	-0.07	-0.29	-0.35
`trans`	0.06	-0.60	-1.51
Avg.	**0.47**	**0.11**	**0.04**

TABLE III
PERCENTAGE GAIN FOR THE SIMPLE EOL ALGORITHM

File	EOL stream size	BWCA % gain	PPM % gain	LZ % gain
`bib`	1,940	-7.28	-7.40	-5.16
`book1`	6,786	1.64	1.76	1.39
`book2`	6,806	-0.70	-0.86	-0.24
`news`	4,689	-2.07	-2.17	-1.31
`paper1`	610	-0.94	-0.86	-0.85
`paper2`	864	-0.41	-0.30	-0.26
`progc`	712	-5.88	-5.92	-4.83
`progl`	1,053	-7.04	-7.19	-5.86
`progp`	937	-8.19	-7.27	-6.92
`trans`	907	-4.79	-4.31	-3.73
Avg.		**-3.57**	**-3.45**	**-2.78**

The second column contains the size of the EOL positions stream.

Four main problems can be identified:

1. Shall every EOL be encoded?
2. Shall every text file be EOL encoded?
3. What symbol shall be encoded as the EOL symbol?
4. How shall the EOL positions be encoded?

The answers to these problems are quite subtle and represent the key to a more successful and reliable EOL coding.

The coding of a position of an EOL symbol requires code space. Therefore, only EOL symbols, which lead to a more predictable context, should be replaced by space symbols and their positions encoded. In this approach, only EOL symbols that are surrounded by lowercase letters are considered as valid and are replaced. In other words, only a part of all EOL symbols inside the file are considered for replacement and the others are ignored. If the percentage share of the valid EOL symbols is too small, the whole file is regarded as unvalid and will be not EOL encoded. Here a threshold of 10% valid EOL symbols is used. The two symbols "Carriage Return" (CR) and "Linefeed" (LF) are treated as EOL symbols together with the combination CR/LF, which is treated as one EOL symbol. Some files not only have a unique EOL symbol, like CR, LF or CR/LF, but some files like `trans` have a set of different EOL symbols. Therefore, the EOL symbol, which has the highest frequency count inside the file, is regarded as the EOL symbol that is being replaced. The combination CR/LF is treated in a particular manner. In this case, where the combination CR/LF is most frequent, all CR/LF combinations are replaced foremost by a single EOL symbol, which is then treated as the EOL symbol to replace.

The implementation of the last question – the coding of the positions – has a very strong influence on the result. Since in many text files the number of symbols per line is on average quite similar, this property can be exploited with a sophisticated encoding scheme. Hereto, the length of the current line is compared with the average length of the last 16 lines. If the current length is greater or equal to the average line length, then the numbers of spaces between the average line length and the current EOL positions are counted and encoded. If the current length is smaller, an escape symbol is encoded as a minus sign and the numbers of spaces between the current EOL position and the average line length is encoded. If the current EOL is an unvalid EOL – in the sense of the former definition – two escape symbols are encoded and the EOL is left unchanged.

The effect of the enhanced EOL scheme can be seen in Table IV. The share of valid EOL symbols for the files `bib`, `progc`, `progl`, `progp` and `trans` is too small, and therefore these files are omitted for EOL coding (the EOL stream size of 1 represents a byte flag, which is transmitted in front of the actual data, giving information if the following data is EOL encoded or not). All other files achieve a compression boost with the three compression schemes.

C. *Token based Replacement*

1) *Determination of available Tokens*

TABLE IV
PERCENTAGE GAIN FOR THE ENHANCED EOL ALGORITHM

File	EOL stream size	BWCA % gain	PPM % gain	LZ % gain
`bib`	1	0.00	0.00	0.00
`book1`	3,882	1.82	1.81	1.16
`book2`	3,776	0.46	0.25	0.37
`news`	856	0.26	0.31	0.09
`paper1`	194	0.58	0.78	0.52
`paper2`	334	1.23	1.38	0.97
`progc`	1	-0.01	-0.01	-0.01
`progl`	1	-0.01	-0.01	-0.01
`progp`	1	-0.01	-0.01	-0.01
`trans`	1	-0.01	-0.01	-0.01
Avg.		**0.43**	**0.45**	**0.31**

The second column contains the size of the EOL positions stream.

Token based replacement refers to the following technique. Given an alphabet A, the set of letter symbols $L = \{a,...,z, A,...,Z, \_\}$ and the set of non-letter symbols $N = A \setminus L$, all occurrences except the first one of a substring X inside the input text are replaced by a token t in the output text. Two kinds of substrings are considered for replacement. The first kind is a word. A word is a sequence of letter symbols from L, which are surrounded in the input text by non-letter symbols from N. The second kind of substrings are either bigrams or trigrams, which consist of two or three letter symbols from L in the input text respectively. Bigrams and trigrams are normally part of a word. For t, a symbol is used which has a frequency count of zero inside the text.

Before the actual replacement starts, the set of tokens is determined. Hereto, all symbols with a frequency count of zero and ordinal values between 0 and 31 and between 127 and 255 are identified. This token set is encoded as the sequence T:

$$T = nq_0n_0q_1n_1...$$

where n is the total number of tokens, q_i is a token, and n_i is the number of continuous tokens that follow. The q_i are ordered in ascending order. For example if the tokens with ordinal value 2, 3, 4, 5, 127, 128, 129, 150, 151, 152, 200, 201, 202, 203 have a frequency count of zero, T would be:

$$T = 14,2,4,127,3,150,3,200,4\,.$$

The sequence T is transmitted at the beginning of the text. The first two tokens t_0 and t_1 (in the former example 2 and 3) are used for trigram replacement, the next two tokens t_2 and t_3 (in the former example 4 and 5) for bigram replacement and the rest of the tokens are used for word replacement. If there are less than 6 tokens, the word replacement is omitted; if there are less than 4 tokens, the phrase replacement is omitted too.

2) *Word Replacement*

Word replacement is the preprocessing algorithm that gives the most compression boost on average. The basic idea is to replace frequently used words by tokens, which are indices into a word dictionary. The size s_D of the dictionary is given by the number of tokens available for word replacement as described in the last section. The token t_4 is used as a first-occurrence escape symbol. In order to keep the approach language independent, no fixed dictionary is used. Instead, the dictionary is built adaptively and transmitted together with the data stream. In the first phase, the frequencies of all words of the file are calculated using a ternary search tree for words with length of size 2 or more. In many languages, the stem of a word is extended with a separate letter at the end for declination like the nominative plural in English and the genitive singular in German or for conjugation like the 's' for the third person singular in English. Therefore, in the second phase, all words W are omitted for replacement where the shorter stem S occurs inside the text with a frequency of at least 25% of W. In this case, S is used instead for replacement. Furthermore, a value for each word W is calculated, which represents the value for replacing W by a token inside the whole file. The value v_W is calculated by:

$$v_W = (f_W - 1) \cdot (l_W - 1),$$

where f_W is the frequency of W and l_W is the length of W. The set of all words is then sorted descending by their respective values and the first – most valuable – s_D words inside the file, which have a value of at least 16, are replaced by tokens in the third phase. When a word occurs for the first time, it is transmitted as is. After the word t_4 is output, indicating that the prior word should be added into the dictionary. During further occurrences, the word is replaced by the token t_{i+5}, where i is the number of words already encoded in the adaptively growing dictionary. Since the decoder is able to rebuild the dictionary from the transmitted data, no external dictionary is needed and the approach stays independent from a specific language.

3) *Phrase Replacement*

Besides word replacement, bigrams and trigrams are also used for token based replacement. Hereto, the two most frequent trigrams are determined and are encoded with the tokens t_0 and t_1 if their frequency inside the text is greater than 64. During the fist occurrence of a trigram, the trigram is transmitted as is and the respective token is transmitted afterwards. The most frequent bigrams are treated the same way, encoded with the tokens t_2 and t_3.

Both word and phrase replacement causes a compression of the text by assigning a single symbol to a frequent sequence of several determined symbols. Table V shows the result of the token based replacement for words and phrases. The result for the file `book2` shows an interesting effect: while the gain for the LZ based compression scheme reaches its highest value, the compression for BWCA is slightly hampered. If the capital letter and upper word conversion are processed together with the token based replacement, the negative result for BWCA disappears, since the conversion supports the word replacement by producing more words that consist of equal lowercase letters as illustrated in Table VI.

TABLE V
PERCENTAGE GAIN FOR THE TOKEN BASED REPLACEMENT ALGORITHM

File	BWCA % gain	PPM % gain	LZ % gain
`bib`	0.97	2.65	3.59
`book1`	0.98	1.46	5.53
`book2`	-0.07	3.57	6.72
`news`	0.53	2.05	4.02
`paper1`	0.70	0.43	4.60
`paper2`	1.22	1.06	6.31
`progc`	2.17	1.28	1.27
`progl`	2.36	5.96	2.84
`progp`	2.17	3.76	4.96
`trans`	-0.43	2.53	-1.47
Avg.	**1.06**	**2.48**	**3.84**

TABLE VI
PERCENTAGE GAIN FOR THE TOKEN BASED REPLACEMENT ALGORITHM IN CONJUNCTION WITH CAPITAL LETTER AND UPPER WORD ALGORITHM

File	BWCA % gain	PPM % gain	LZ % gain
`bib`	1.81	2.92	3.77
`book1`	1.76	2.20	6.20
`book2`	1.81	4.38	7.51
`news`	1.26	2.48	4.28
`paper1`	1.89	1.66	5.51
`paper2`	2.30	2.25	7.17
`progc`	2.99	2.49	2.50
`progl`	3.24	6.82	3.73
`progp`	3.16	3.75	4.84
`trans`	0.01	2.78	-1.20
Avg.	**2.02**	**3.17**	**4.43**

V. SPECIAL TEXT PREPROCESSING ALGORITHMS

For compression schemes that are based on sorting stages like BWCA, for example, the lexicographic order of the alphabet during the sorting process has an impact on the sorted output. During BWCA, the sorted data is usually processed by a Global Structure Transformation (GST), which transforms the local context of the different output segments to a global context [4, 17, 26]. This global context data is finally compressed by an Entropy Coding (EC) stage. If the lexicographic order of the alphabet is changed in a way that symbols with a similar context are grouped closer together, segments with similar context properties will also be grouped closer together. This can be exploited by the GST stage since context changes will be smoother and less abrupt.

In this paper, a heuristic alphabet reordering is used, which was determined by trying many hand-permutated orderings. It is different to former approaches of grouping all vowels together [14, 15, 16, 18]. This approach groups the vowels "aoui" in the middle of the consonants together and the 'e' at the end of the consonants. Furthermore, other characters like digits and punctuation symbols are also reordered. Figure 1 contains the complete alphabet order.

The result of the alphabet reordering can be seen in Table VII. The gain of 1.1% using the new alphabet order for the BWCA is more than double as much as that reported by Chapin and Tate [14], Kruse and Mukherjee [17], and Chapin [15], who achieved between 0.2% to 0.5%. Since PPM and LZ do not depend on a sorting stage, their compression rates are practically unaffected.

```
                 "´_", tab, "@", space, 0x1B, lf, cr,
              ".:?,];)|~&<=>{*+}[/!-"'\0123456789%$",
                   "SNLMGQZBPCFWRHAOUIYXVDKTJE",
                              "^#(",
                   "snlmgqzbpcfwrhaouiyxvdktje",
                         0x7F … 0xFF,
0x00 … 0x08, 0x0B, 0x0C, 0x0E … 0x1A, 0x1C … 0x1F
```

Fig. 1. New alphabet order

TABLE VII
PERCENTAGE GAIN FOR THE ALPHABET REORDERING ALGORITHM

File	BWCA % gain	PPM % gain	LZ % gain
bib	0.52	0.00	-0.01
book1	0.61	0.00	-0.01
book2	0.82	0.00	0.00
news	0.67	0.00	0.01
paper1	1.37	0.00	0.02
paper2	1.26	0.00	0.01
progc	1.65	0.00	0.01
progl	1.66	0.00	0.01
progp	2.06	0.00	0.02
trans	0.39	0.00	0.02
Avg.	**1.10**	**0.00**	**0.01**

VI. CONCLUSIONS

The compression rates of the popular compression techniques, which are based on BWT, PPM or LZ schemes, can be improved by using a text preprocessing algorithm beforehand.

This paper puts forward a compound text preprocessing scheme that consists of a text/non-text recognition scheme and five different text preprocessing algorithms. The first algorithm is capital letter and capitalized word conversion, which converts the capital letter of a word to a lowercase letter and converts words, which consist only of uppercase letters, into words with lowercase letters. This transformation helps to increase the similarities of the contexts of the words. The second algorithm, EOL coding, replaces EOL symbols like "carriage return" and "linefeed" in the text with space characters and encodes the former position into a separate data stream by a special encoding method that uses the average line length in terms of the number of spaces. Replacing the EOL by space symbols helps to improve the predictability of the context, since words are usually separated by spaces. The third algorithm replaces words by tokens. The most valuable words, in relation to their frequency and length, are calculated and replaced by byte tokens, except for the first occurrence. The fourth algorithm replaces the most frequent two trigrams and bigrams. Both replacement algorithms result in a compression of the text. The last algorithm is alphabet reordering, which supports sort based compression schemes like BWCA. The reordering helps to place segments with similar contexts closer together after the sorting. All five algorithms are processed sequentially one after the other as illustrated in Figure 2 after the initial text recognition scheme.

Table VIII compares the compression rates for the text files of the Calgary Corpus between the raw files and the text preprocessed files for BWCA, PPM and LZ schemes. The compression gain on average is between 3% to 5%.

Table IX displays the respective results for the text files of the large Canterbury Corpus. The compression gain here is about 2% for the BWCA and between 7% to 9% for the LZ and PPM scheme. One reason for the smaller boost on the BWCA may be the fact that the BWCA has already a very strong compression (1.379 bps), which is more difficult to

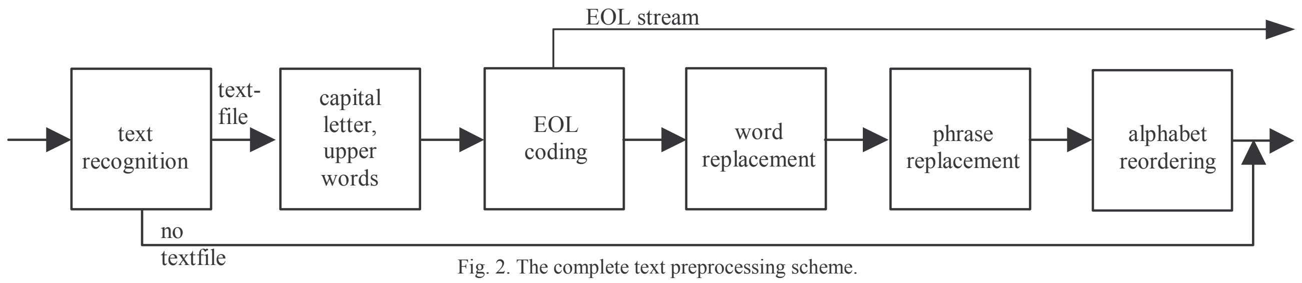

Fig. 2. The complete text preprocessing scheme.

TABLE VIII
COMPRESSION RATES IN BPS AND PERCENTAGE GAIN FOR THE TEXT FILES OF THE CALGARY CORPUS

File	BWCA raw [a]	BWCA preproc. [b]	BWCA % gain [c]	PPM raw [a]	PPM preproc. [b]	PPM % gain [c]	LZ raw [a]	LZ preproc. [b]	LZ % gain [c]
bib	1.888	1.840	**2.51**	1.901	1.845	**2.92**	2.516	2.421	**3.76**
book1	2.226	2.136	**4.05**	2.302	2.198	**4.52**	3.256	3.015	**7.42**
book2	1.929	1.869	**3.10**	2.017	1.913	**5.13**	2.702	2.491	**7.82**
news	2.400	2.347	**2.20**	2.408	2.338	**2.88**	3.072	2.938	**4.37**
paper1	2.381	2.286	**4.00**	2.341	2.280	**2.59**	2.791	2.630	**5.76**
paper2	2.331	2.230	**4.34**	2.317	2.226	**3.91**	2.880	2.649	**8.02**
progc	2.418	2.309	**4.49**	2.380	2.321	**2.48**	2.678	2.612	**2.48**
progl	1.658	1.585	**4.41**	1.739	1.621	**6.82**	1.806	1.739	**3.75**
progp	1.658	1.582	**4.61**	1.726	1.661	**3.74**	1.812	1.724	**4.85**
trans	1.439	1.433	**0.42**	1.530	1.488	**2.78**	1.611	1.630	**-1.20**
Avg.	**2.033**	**1.962**	**3.41**	**2.066**	**1.989**	**3.78**	**2.512**	**2.385**	**4.70**

[a] "raw" means file without preprocessing, compression rates in bps (bits per symbol).
[b] "preproc." means file with preprocessing, compression rates in bps (bits per symbol).
[c] Percentage gain between raw file and preprocessed file.

TABLE IX
COMPRESSION RATES IN BPS AND PERCENTAGE GAIN FOR THE LARGE CANTERBURY CORPUS

File	BWCA raw [a]	BWCA preproc. [b]	BWCA % gain [c]	PPM raw [a]	PPM preproc. [b]	PPM % gain [c]	LZ raw [a]	LZ preproc. [b]	LZ % gain [c]
bible.txt	1.452	1.421	**2.14**	1.699	1.524	**10.29**	2.330	2.113	**9.35**
world192.txt	1.306	1.286	**1.51**	1.679	1.545	**7.99**	2.337	2.217	**5.15**
Avg.	**1.379**	**1.354**	**1.83**	**1.689**	**1.535**	**9.14**	**2.334**	**2.165**	**7.25**

[a] "raw" means file without preprocessing, compression rates in bps (bits per symbol).
[b] "preproc." means file with preprocessing, compression rates in bps (bits per symbol).
[c] Percentage gain between raw file and preprocessed file.

improve than the 1.689 bps from PPM and 2.334 bps from the LZ scheme.

Besides the pure compression gain, the independence from a specific language and from an external dictionary makes this universal approach very attractive for all three kinds of compression schemes.

ACKNOWLEDGMENT

The discussions with Sebastian Deorowicz, Szymon Grabowski, Uwe Herklotz, Maxim Smirnov and Vadim Yoockin have been very much appreciated.

REFERENCES

[1] M. Burrows and D. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm," Technical report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[2] J. Cleary and I. Witten, "Data Compression Using Adaptive Coding and Partial String Matching", *IEEE Transactions on Communications*, pp. 396–402, 1984.

[3] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, pp. 337–342, 1977.

[4] J. Abel, "Improvements to the Burrows–Wheeler Compression Algorithm: After BWT Stages," submitted for publication in *ACM Transactions on Computer Systems*, 2003.

[5] W. Teahan, " Probability estimation for PPM," *Proceedings of the New Zealand Computer Science Research Students' Conference*, University of Waikato, New Zealand, 1995.

[6] J. Gailly. (1993). GZIP – The data compression program – Edition 1.2.4. [Online]. Available: http://miaif.lip6.fr/docs/gnudocs/gzip.pdf

[7] J. Bentley, D. Sleator, R. Tarjan and V. Wei, "A locally adaptive data compression scheme," *Communications of the ACM*, 29, pp. 320–330, 1986.

[8] A. Moffat, "Word–based Text Compression," *Software – Practice and Experience*, pp. 185–198, 1989.

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

[10] W. Teahan and J. Cleary, "The Entropy of English using PPM–Based Models," *Proceedings of the IEEE Data Compression Conference 1996*, Snowbird, pp. 53–62.

[11] W. Teahan and J. Cleary, "Models of English Text," *Proceedings of the IEEE Data Compression Conference 1997*, Snowbird, pp. 12–21.

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

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

[14] B. Chapin and S. Tate, "Preprocessing Text to Improve Compression Ratios," *Proceedings of the IEEE Data Compression Conference 1998*, Snowbird, p. 532.

[15] B. Chapin, "Higher Compression from the Burrows–Wheeler Transform with new Algorithms for the List Update Problem," Ph.D. dissertation, Department of Computer Science, University of North Texas, 2001.

[16] B. Balkenhol and Y. Shtarkov, "One attempt of a compression algorithm using the BWT," *SFB343: Discrete Structures in Mathematics*, Falculty of Mathematics, University of Bielefeld, Germany, 1999.

[17] H. Kruse and A. Mukherjee, "Improving Text Compression Ratios with the Burrows–Wheeler Transform," *Proceedings of the IEEE Data Compression Conference 1999*, Snowbird, p. 536.

[18] S. Grabowski, "Text Preprocessing for Burrows–Wheeler Block–Sorting Compression," *VII Konferencja Sieci i Systemy Informatyczne – Teoria, Projekty, Wdrozenia*, Lodz, Poland, 1999.

[19] 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.

[20] F. Awan. N. Zhang, N. Motgi, R. Iqbal and A. Mukherjee, "LIPT: A Reversible Lossless Text Transform to Improve Compression Performance," *Proceedings of the IEEE Data Compression Conference 2001*, Snowbird, pp. 481–210.

[21] R. Isal and A. Moffat, "Parsing Strategies for BWT Compression," *Proceedings of the IEEE Data Compression Conference 2001*, Snowbird, pp. 429–438.

[22] R. Isal, A. Moffat and A. Ngai, "Enhanced Word–Based Block–Sorting Text Compression," *Proceedings of the twenty–fifth Australasian conference on Computer science*, pp. 129–138, 2002.

[23] W. Teahan and D. Harper, "Combining PPM Models Using a Text Mining Approach," *Proceedings of the IEEE Data Compression Conference 2001*, Snowbird, pp. 153–162.

[24] J. Bentley and R. Sedgewick, "Fast Algorithms for Sorting and Searching Strings," *Eighth Annual ACM–SIAM Symposium on Discrete Algorithms*, New Orleans, 1997.

[25] P. Elias, "Universal Codeword Sets and Representations of the Integers," *IEEE Transactions on Information Theory*, pp. 194–203, 1975.

[26] S. Deorowicz, "Improvements to Burrows–Wheeler Compression Algorithm," *Software – Practice and Experience*, pp. 1465–1483, 2000.