Compression transformation for small text files

Jan Platoš
Department of Computer Science, FEI
VSB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
Email: jan.platos@vsb.cz

***Abstract*—The world of electronic communication is growing. Most of our work today is done using electronic devices. The amount of messages, voice and video calls, and other electronic communication is enormous. To make this possible, data compression is used everywhere. The amount of information must be compressed in order to reduce the requirements imposed on communication channels and data storage. This paper describes a novel transformation for compression of small text files.Such type of data are produced every day in huge amounts, because emails, text messages, instant messages, and many other data types belong to this data type. Proposed transformation leads to significant improvement of compression ratio in comparison with standard compression algorithms.**

Keywords: data compression, boolean minimization, small files

I. Introduction

The idea of data compression dates back to the 19th century. The first widely used compression techniques may be found in the Morse code for telegraphy from the year 1832 and in the Braille system for the blind from 1820. The main development of compression methods dates back to the 20th century, when computers and electronic telecommunication techniques were invented. At the beginning, the evolution of compression methods was motivated by the need to transmit messages through very slow channels (physically through metallic wires or by radio waves). It was necessary to send data between universities, tactical information for military forces, pictures and sensor data from satellites and probes, etc. These days, data compression is used everywhere in the information society.

Two main reasons for the usage of data compression exist. Large amounts of data must be stored into data storage or transmitted through limited communication channels. The application of data compression leads to a significant reduction in the disk space used and the necessary bandwidth. The data which must be stored or transmitted may be text documents, images, audio and video files, etc. Moreover, stored data are usually backed up to prevent data loss.

Text compression is a special task in the field of data compression, because the world is full of textual information. Textual data has existed since the start of computers, because information written in text is more suitable for humans than information written as binary code or a sequence of integers. Today, many places where textual information is used - books, forms, web pages, laws, personal notes, chats, and many other types - exist. A great effort is being dedicated around the world to the building of electronic libraries. It is not only about real books, but it is necessary to create data stores with laws, archives of web pages, data stores for emails, etc. Any of the universal compression methods may be used for the compression of text files, but practice has shown that it is better to use an algorithm developed especially for text, such as PPM methods [4], [12], a Burrows-Wheeler transformation-based method [2], or others.

The efficiency of the compression methods is based on the context information contained in the data or on the differences in probability between the occurrences of symbols. A good compression ratio is achieved for files with sufficient context information or with enough differences between the probability of symbols. Context information in random data is not sufficient. Moreover, symbols have equal probability and therefore it is not possible to gain any compression. Compression of very small text files, such as text messages, chat messages, messages in instant messaging networks, and emails, is even more specialized task, because such files have different attributes than normal and large text files. The main difficulty is insufficient context information.

Several algorithms for the compression of small files have been developed. The first group of algorithms use a model prepared before compression. The data used for the creation of models are the same or similar to the compressed one. This solves the problem with modeling the context in short messages, but the model used must be part of the decompressor of the compressed data. This approach was used by Rein et.al. [16], which use a PPM-based method with a strict memory constraint and a static model for the compression of text messages in mobile devices. Korodi et.al. [10] use the Tree Machine for compression. The Tree Machine uses a trained model with a limited size. Hatton [8] presents a different approach for estimating symbol probability, based on the counts of previous occurrences. An SAMC compression algorithm is based on this computation of probability, multi-pass model training, and a variable-length Markov model. The results achieved are better than with using PPM, with a smaller memory requirement for very small files.

The second group of algorithms solve the problem of insufficient context using application of transformation. El-Qawasmeh and Kattan in 2006 [5] introduce Boolean minimization in data compression. A modification of this approach focused on small text files was published in 2008 by Platos et.al. [15]. In that paper, several new transformation was suggested as well as other encodings for the final stage of

the compression. In this technique, the compression process splits the input data stream into 16-bit blocks. After that, the Quine-McCluskey approach is used to handle this input in order to find the minimized expression. The minimized expressions of the input data stream are stored in a file. After minimization, El-Qawasmeh uses static Huffman encoding for obtaining redundancy-loss data.

This paper proposes a modification of the second approach, but a variant on the first approach is also described. The common phase is transformation of the data using Boolean minimization, but, for effective compression, it is necessary to solve a few other problems.

Organization of this paper is as follows. The Section II contains short description of the algorithm for compression of small text files as well as its new modification. The Section IV describes the testing data collection and results of the experiments are discussed in Section V. The Section VI present a summary of this paper.

II. Compression of small text files

The compression algorithms, as was published in [15], may be described in the following steps. The input messages are split into 16-bit vectors. Each vector corresponds to a 4-variable truth table. Vectors are transformed using mapping table into mapped vectors. If the semi/adaptive mapping is used, mapping table is stored into output. Mapped vectors are minimized using the Quine-McCluskey algorithm into product terms. Product terms are transformed into its numeric representation. Finally, compression algorithm is used for storing product terms numbers into output. The details about Boolean minimization, Quine-McCluskey algorithms and product terms may be found in [15], [7], [13], [17]. Application of Boolean minimization leads to the reduction of the alphabet size to 83 symbols [5], [15] and to the increasing of the number of symbols (data volume). Therefore, reduction of the number of symbols must be applied. Improvement of the final step - compression of the number, may be also improved by mapping function for increasing of the context information.

A. Reduction of the number of symbols

Several techniques solve this problem. The main technique is the reduction of the products that are generated. But the volume of data may also be reduced using the elimination of some parts of the generated data. Both techniques are described in the following paragraphs.

1) Reduction of the number of products: Reduction of the number of products must originate from the nature of the data. Because this method is focused on the compression of small text files, the determination of the distinct vectors count may be useful. Text files have very low numbers of distinct vectors - typically less than 5,000 out of 65,536 vectors $\approx 8\%$; e.g. the *bible.txt* file from the Canterbury Compression Corpus [1] has 1121 distinct vectors and *latimes.txt* from TREC [6, 7, 8] has 4984 distinct vectors.

Moreover, the number of products used for describing the vectors is not identical for each vector. Some vectors are described by only one product and some use more than four.

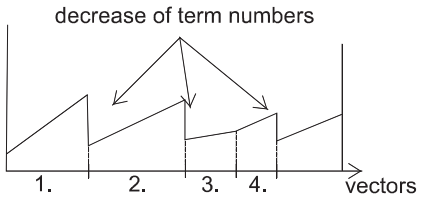

Fig. 1. Elimination of separators

Since only a small number of distinct vectors is present in the data, a mapping transformation may be applied. This transformation will map the most used vectors on the vectors with the lowest number of products, and the least used vectors on the vectors with the highest number of product terms. This mapping may be static or dynamic. The advantage of the static variant is that the mapping table is part of the compressor/decompressor and not a part of the compressed data. On the contrary, dynamic mapping better matches the content of the data, but it also needs two-pass processing and the mapping table must be stored as a part of the compressed data.

2) Separator elimination: Another reduction can be achieved with separator elimination. Products which describe vectors may be sorted from lowest to highest, according to their numeric representation, without loss of commonness. The passing between the products of adjacent vectors can be detected by reducing the term number. This principle works in most cases but not in all. The problems and advantages of this method may be seen from the following example. More details and example may be fount in [15].

a) Example:: Let the message may be encoded using five vectors $v_1, \ldots, v_5$. These vectors are processed using Boolean minimization and the results products are these $v_1 = \{4, 5, 24\}, v_2 = \{14, 25, 34, 35\}, v_3 = \{11, 22\}, v_4 = \{33, 43\}, v_5 = \{16, 23, 45, 56\}$. These product lists are shown in Figure 1. As may be seen, decreasing of product number may not be seen between vectors 3 and 4. In other cases, the decreasing is presents. Therefore, three from four separator may be eliminated.

B. Increasing of context information

Increasing the amount of contextual information was achieved using Boolean functions, but it may still be improved. One method is reducing the size of the alphabet. Another is modification of the mapping function, which was introduced in the previous section.

1) Reduction of the number of symbols: The context information is increased by the reduction of the number of symbols, but this reduction may not be final. We may reduce the alphabet and compress data by smaller parts than products - logical variables $(A, A', B, \ldots D')$. In this case, the separator between products must also be compressed. This final alphabet contains 12 symbols $(A, A', B, \ldots, D', 0, 1, +, comma)$. This reduction again leads to an increase in the data volume. The ratio between increasing the contextual information and

increasing the data volume may not be as good as in the case of the representation of vectors as Boolean functions. This will be proven by the following experiments.

2) Modification of the mapping function: The mapping function introduced in Section II-A1 substitutes vectors in data with vectors with the lowest number of the representing products. This reduces the size of the data but the products of the result vectors are within the scope of the whole alphabet, because at the beginning of the mapping table are vectors which have only one product in their minimization, as may be seen in the following notation.

$$(0, 1, A, A', B, \ldots, D', AB, AB', \ldots, A'B'C'D', A + B, \ldots)$$

The mapping function may be modified to move vectors with the same products in their representation to the front. The ordering of the representation of the vectors is shown in the next notations.

$$(0, 1, A, A', B, A + B, A' + B, B', A + B', A' + B', C, \ldots)$$

This mapping function is not so effective in reducing the volume of the data, but it reduces the number of symbols used.

C. Encoding of the product terms

Encoding the output is the final part of the algorithm. El-Qawasmeh and Kattan in 2006 [5] used semi-adaptive Huffman encoding in order to achieve the compression for the product terms. This approach has many disadvantages - this encoding needs two passes over the file and the storing of the Huffman table into a compressed file, and the compression ratio of the Huffman encoding is usually lower than when using other compression methods.

In our experiments, many compression methods will be tested. The context information was increased by the Boolean minimization and other transformations, therefore, any of the algorithms described in may be used without any modification. Statistical compression methods may use improved contextual information for the better modeling of symbol probabilities and dictionary-based methods find longer matches. Burrows-Wheeler transformation generates longer sequences of symbols.

Moreover, the sorting of products which represent vectors allows the creation of a better statistical model, because the probability of the product in the data depends not only on the overall probability of this product but also on the previous product. These models are also called models with context length 1 or models of order-1. In addition, this model may be initialized prior to encoding more accurately, because after each product only the product with a higher number or *comma* symbol may be present.

III. The Compression framework

The compression algorithm for small text files was derived into compression framework. This framework has several parts, but in this section only few of the will be described. More information about this framework may be found in [14].

A. Static mapping

The static mapping is used for reducing the generated product terms by using the mapping of the vectors that are used into other vectors according to some criteria. The ideas are described in Sections II-A1 and II-B. Three possible variants were implemented.

The first variant map vectors into vectors with the smallest number of the representing products. The vectors that were originally present are mapped according to their numeric representations independent on the frequency. The vectors that were originally present are mapped according to their numeric representations. Let the sorted list of vectors be called S, the first vector in S, i.e. with the smallest number of product terms, S_1, the second S_2, etc. When the data contain vectors (123, 43, 65, 234), then vector 43 is mapped into vector S_1, vector 65 is mapped into vector S_2, vector 123 is mapped into S_3 and vector 234 into vector S_4. This variant is called *smf*.

The second variant is similar to the previous one, but the vectors that were originally present are mapped according to their frequency in the data. When the data contain vectors (123, 43, 64, 123, 64, 123, 234), then vector 123 is mapped into S_1, vector 64 into vector S_2, vector 43 into vector S_3 and vector 234 into vector S_4. This variant will be called *sms*.

The last variant sorts vectors according to the criterion described in Section II-B2 for improving context information using a reduction in the number of symbols. The vectors that were originally present are mapped in the same way as in *smf*. This variant will be called *con*.

Static mapping is used globally for all messages from the collection without the necessity of storing anything to output, because the mapping function is part of the compressor/decompressor. The information needed for the mapping is gained from the whole collection before encoding.

B. Vector mapping

Vector mapping is used for each message. This means that for each used message an individual mapping function is constructed and it is stored into the output for the decompression process. The mapping table is stored as a map of the vectors, where the distances between vectors are stored using Fibonacci encoding [6]. The size of the mapping table may be reduced using an adaptive mapping scheme. This adaptive mapping scheme is used before the vector mapping and it is updated after each message using a Move-to-Front scheme. The content of the adaptive mapping scheme is based on all the previously encoded messages, but this may be achieved in practical usage.

Two basic variants of the vector mapping were implemented - the *smf* and *con*. Both variants were described in the previous section. Each variant was used with and without a global adaptive scheme. The first variant is called *smf* without a global mapping scheme and *smf_g* with a global mapping scheme. The second scheme is called *con* and *con_g* respectively.

C. Boolean minimization

Boolean minimization is used for converting vectors into a list of product terms. The vectors are transformed according to the algorithm described at the beginning of this chapter, i.e. a vector is taken as a truth table and minimized using the Quine-McCluskey algorithm, and the final list of products is the shortest possible list covering all the true values from the table. Product terms for each vector are sorted according to their numeric representation.

D. Product transformation

Transformations for product terms were described in Subsections II-A2 and II-B1. The separator elimination is used for reducing the number of products and is called *sepElim*. The decomposition to logical values is used for the reduction of the number of symbols and increasing the context information. This transform is called *logValue*.

E. Product encoding

Encoding of the list of products is the final stage of the algorithm. Several algorithms were used. The size of the alphabet was set to 83. Direct Encoding means the the product number were stored directly into out using 7-bit per number. Other encoding is based on compression algorithms: Semi-adaptive and Adaptive Huffman Encoding, LZW algorithm, Burrows-Wheeler based encoding, Order-O and Order-1 Range Encoding [11]. The Order-1 Range encoder use the knowledge about sorting to improve prediction of the following symbol.

1) Direct encoding: Direct encoding is the simplest possible encoding of the product terms. The products are stored with only the necessary number of bits. When the size of the alphabet is 83, only 7 bits are necessary. For an alphabet with a size of 12, only 4 bits are used. This encoding is called *min_dc*.

2) Semi-adaptive Huffman encoding: This algorithm is used in its standard form. It needs two-pass processing and storing of the Huffman tree into an output file. This algorithm was used in the original work of El-Qawasmeh and Kattan [5]. This encoding is called *min_sh*.

3) Adaptive Huffman encoding: This algorithm was also used in its standard form. In comparison with the semi-adaptive version, it needs only one-pass processing and it is not necessary to store the encoding tree to the output. This encoding is called *min_ah*.

4) LZW algorithm: The LZW algorithm was used in its standard form with the size of the dictionary set to 2^{20} symbols. LZW uses storage with an increasing number of bits. At the beginning only 83 phrases are used and therefore only 7 bits are necessary to store the phrase number. When the number of phrases exceeds 127, 8 bits is used. This encoding is called *min_lzw*.

5) Burrows-Wheeler transformation based encoding: Burrows-Wheeler transform was used for encoding the product in two variants. The first variant uses BWT with Move-to-Front and adaptive Huffman encoding. This variant is called *min_bwt_ahc*. The second variant uses BWT with Move-To-Front, run-length encoding, and adaptive Huffman encoding. This variant is called *min_bwt_ahc_rle*.

6) Order-0 Range encoding: This encoding uses an Order-0 model with range encoding. Range encoding [11] is mathematically equivalent to arithmetic encoding. The principle of the algorithm is very similar to arithmetic encoding and may be used instead of it. The used model is tuned for better performance. The tuning consists of the initialization of the model. Each symbol is initialized at the same frequency, but the *Comma* symbol has a higher frequency than the other symbols, because each list of products must be followed by the *Comma* symbol. When the *sepElim* transform is used, the preference of the *Comma* symbol is lower. This encoding is called *min_0r*.

7) Order-1 Range encoding: This encoding uses an Order-1 model with range encoding. The model is again tuned for better performance. Because of the sorting of the products, it is not possible to find a product with a lower numeric representation than the actual one, except after the *Comma* symbol. The probability of each symbol with a lower numeric representation than the actual one is set to 0. The *Comma* symbol is also preferred, as in the previous encoding. When the *sepElim* transform is used, only the *Comma* symbol is preferred, because the rule about the presence of a product with a lower numeric representation is not valid. This encoding is called *min_1r*. Pure encoding is the general name for standard algorithms used to store messages in their original form - without Boolean minimization.

TABLE I
STATISTICS OF THE TESTING FILES

	smsCorpus	smsLarge
Message count	854	10 117
Char. count	93 288	588 161
Avg. Message length	109.24	58.14
Vector count	46 860	296 582

IV. DATA COLLECTIONS

Two collections of text messages (SMS) were chosen as the main test data. The first one contains 854 messages and was collected by Monojit Choudhury and his team [3]. The second one contains 10,117 messages and was collected at the Department of Computer Science at the National University of Singapore by students [9]. These collections are called *smsCorpus* and *smsLarge*.The statistics of used files is shown in Table I.

V. EXPERIMENTS

Several experiments were performed. The total number of possible variants may be calculated from four possible static mappings (*none*, *SMS*, *SMF*, *CON*), five possible vector mapping (*none*, *SMF_G*, *SMF*, *CON_G*, *CON*) and tree possible transformations (*none*, *SepElim*, *LogVal*). The total number is $4 \times 5 \times 3 = 60$ and all were performed on all collections. This section will contain only the most interesting result.

all results are published in [14]. The configuration of the testing computer is not important, because all experiments were focused on the best possible compression ratio and not on the compression time.

A. Experiments without static mapping

The first experiments were performed without static mapping. Therefore, each mapping function must be stored into the output. The most effective mapping function is *smf_g* for both collections, but the variant without the adaptive mapping scheme *smf* produces almost the same low count of symbols. The difference between the *con_g* and *con* mapping is higher but still very small. Any of the mapping functions produces a significantly lower count of symbols than the amount produced without mapping functions. The most interesting results for both collections are summarized in Table II. When the BWT-based compression method was used, better results were achieved without *sepElim* transformation. This transformation reduces the number of clues, which leads to longer sequences of symbols and, consequently, to better compression. The mapping with minimization leads to an improvement in the compression ratio from 1 to 6 percent.

As may be seen, the methods based on Boolean minimization without static mapping are more suitable for the compression of short messages. The improvement in the compressed ratio for short messages was up to 7%. The result size is greatly affected by the size of the mapping function and, therefore, it is necessary to reduce the size of the mapping function as far as possible. The transformations used for improving efficiency were also important, because the compression ratio achieved is much better when the transformations are used.

B. Experiments with static mapping

The second group of experiments was focused on static mapping. Three different types of static mapping were tested. The main advantage of the usage of static mapping is that the mapping function is part of the compressor/decompressor and not a part of the compressed data. This has one main advantage and one main disadvantage. The advantage is that the mapping function is not stored in the data and, therefore, the data are much smaller than without static mapping. The disadvantage is that the mapping function is more general than the mapping function generated for each message, because it must produce the best mapping for all the messages in the collections.

The significant attribute for static mapping is the source of the knowledge used for the creation of the mapping function. This source must have the same characteristics as the compressed data to achieve good results. In our experiments, the mapping function was generated from the compressed data themselves.

The summarization of the results for SMS files is in Table III. The best results for SMS files were achieved with *sms* static mapping and *sepElim* transformation, but the results without *sepElim* transformation are very close to the best results. The improvement achieved is more than 20% in comparison with pure algorithms for both files, but not as good as may be expected from results without static mapping, because the static mapping is modeled according to all the messages, as mentioned above.

The compression method based on Boolean minimization with static mapping are suitable for messages of any-size, but better results were achieved for shorter messages. The best combination of the parts of the compression framework is *sms* static mapping without per-message mapping and transformation for very short messages represented by both *sms* collections.

VI. Conclusion

The proposed algorithm is designed for the compression of very small text files. Very small text files are text files with a length up to 500 B. The compression of these files is very difficult, because all compression algorithms lack enough information about the compressed data to achieve good compression. The proposed algorithm introduces a new type of transformation for compression - Boolean minimization, in combination with additional transformation for solving this problem. Briefly described, the proposed transformations lead to the reducing of the size of the alphabet and increasing of the context information in the data. Additional transformations may be divided into 3 groups - static mapping, per-message mapping, and symbol transformation. When no static mapping is used, the algorithm needs no previous knowledge about the compressed data; otherwise pre-compression analysis must be performed. A comparison was performed of standard compression algorithms without and with the proposed techniques. When no static mapping was used, the compression achieved was up to 8% better than without the proposed transformation. When static mapping was used, the improvement achieved was up to 25%.

References

[1] R. Arnold and T. Bell. A corpus for the evaluation of lossless compression algorithms. In J. A. Storer and M. Cohn, editors, *Proc. 1997 IEEE Data Compression Conference*, pages 201–210. IEEE Computer Society Press, Los Alamitos, California, Mar. 1997.

[2] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital SRC Research Report, 1994.

[3] M. Choudhury. Sms collection. electronic, March 2010. http://www.mla.iitkgp.ernet.in/˜monojit/sms.html.

[4] J. G. Cleary, Ian, and H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32:396–402, 1984.

[5] E. El-Qawasmeh and A. Kattan. Development and investigation of a novel compression technique using Boolean minimization. *NEURAL NETWORK WORLD*, 16(4):313–326, 2006. 4th International Multiconference on Computer Science and Information Technology, Amman, JORDAN, APR 05-06, 2006.

[6] A. S. Fraenkel and S. T. Klein. Robust universal complete codes for transmission and compression. *Discrete Applied Mathematics*, 64:31–55, 1996.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[8] E. Hatton. Samc - efficient semi-adaptive data compression. In *CASCON '95: Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, page 29. IBM Press, 1995.

[9] Y. How and M. F. Lee. The nus sms corpus. electronic, March 2010. http://www.comp.nus.edu.sg/ rpnlpir/downloads/corpora/smsCorpus/.

TABLE II
PART OF THE RESULTS FOR BOTH SMS COLLECTIONS WITH *smf_g* MAPPING. CR MEANS COMPRESSION RATIO, CS MEANS COMPRESSED SIZE. THE BEST RESULTS ARE HIGHLIGHTED IN EACH COLUMN.

File	smsCorpus				smsLarge			
Orig. size	93 288 B				588 161 B			
Transform	none		sepElim		none		sepElim	
	CS [B]	CR [%]	CS [B]	CR [%]	CS [B]	CR [%]	CS [B]	CR [%]
min_dc	104 925	112.47	83 834	89.87	723 629	123.03	588 459	100.05
min_0r	77 216	82.77	72 817	78.06	568 790	96.71	540 107	91.83
min_1r	**72 503**	**77.72**	**72 178**	**77.37**	538 591	91.57	536 568	91.23
min_ah	79 843	85.59	72 593	77.82	580 675	98.73	**526 788**	**89.57**
min_bwt_ahc	88 357	94.71	84 488	90.57	694 522	118.08	658 542	111.97
min_bwt_ahc_rle	73 936	79.26	81 916	87.81	594 611	101.10	641 305	109.04
min_lzw	101 989	109.33	83 235	89.22	725 605	123.37	592 715	100.77
min_sh	108 696	116.52	105 069	112.63	765 400	130.13	742 058	126.17
pure_0r	77 859	83.46	77 859	83.46	547 006	93.00	547 006	93.00
pure_ah	78 109	83.73	78 109	83.73	**533 557**	**90.72**	533 557	90.72
pure_bwt_ah	91 868	98.48	91 868	98.48	674 159	114.62	674 159	114.62
pure_bwt_ah_rle	92 497	99.15	92 497	99.15	678 849	115.42	678 849	115.42
pure_lzw	83 623	89.64	83 623	89.64	577 294	98.15	577 294	98.15

TABLE III
PART OF THE RESULTS FOR BOTH SMS COLLECTIONS WITH *sms* STATIC MAPPING. CR MEANS COMPRESSION RATIO, CS MEANS COMPRESSED SIZE. THE BEST RESULTS ARE HIGHLIGHTED IN EACH COLUMN.

File	smsCorpus				smsLarge			
Orig. size	93 288 B				588 161 B			
Transform	none		sepElim		none		sepElim	
	CS [B]	CR [%]	CS [B]	CR [%]	CS [B]	CR [%]	CS [B]	CR [%]
min_dc	95 014	101.85	70 093	75.14	627 775	106.74	469 030	79.75
min_0r	65 342	70.04	59 635	63.93	463 230	78.76	426 173	72.46
min_1r	**59 241**	**63.50**	59 967	64.28	**420 471**	**71.49**	425 312	72.31
min_ah	67 526	72.38	**58 682**	**62.90**	470 806	80.05	**406 280**	**69.08**
min_bwt_ahc	81 416	87.27	72 709	77.94	622 491	105.84	549 635	93.45
min_bwt_ahc_rle	74 820	80.20	73 061	78.32	584 537	99.38	552 537	93.94
min_lzw	81 519	87.38	69 003	73.97	566 488	96.32	469 889	79.89
min_sh	89 618	96.07	83 905	89.94	649 102	110.36	609 379	103.61
pure_0r	77 859	83.46	77 859	83.46	547 006	93.00	547 006	93.00
pure_ah	78 109	83.73	78 109	83.73	533 557	90.72	533 557	90.72
pure_bwt_ah	91 868	98.48	91 868	98.48	674 159	114.62	674 159	114.62
pure_bwt_ah_rle	92 497	99.15	92 497	99.15	678 849	115.42	678 849	115.42
pure_lzw	83 623	89.64	83 623	89.64	577 294	98.15	577 294	98.15

[10] G. Korodi, J. Rissanen, and I. Tabus. Lossless data compression using optimal tree machines. In *DCC '05: Proceedings of the Data Compression Conference*, pages 348–357, Washington, DC, USA, 2005. IEEE Computer Society.

[11] G. N. N. Martin. Range encoding: an algorithm for removing redundancy from a digitised message. In *Video & Data Recoding Conference*, 1979.

[12] A. Moffat. Implementing the ppm data compression scheme. *IEEE Transactions On Communications*, 38(11):1917–1921, 1990.

[13] V. P. Nelson, H. T. Nagle, B. D. Carroll, and D. Irwin. *Digital Logic Circuit Analysis and Design*. Prentice Hall, 2nd edition, March 1995.

[14] J. Platos. *Scalability of the compression algorithms*. PhD thesis, VSB-Technical University of Ostrava, May 2010.

[15] J. Platos, V. Snasel, and E. El-Qawasmeh. Compression of small text files. *ADVANCED ENGINEERING INFORMATICS*, 22(3):410–417, JUL 2008.

[16] S. Rein, C. Guhmann, and F. H. P. Fitzek. Low-complexity compression of short messages. In *DCC '06: Proceedings of the Data Compression Conference*, pages 123–132, Washington, DC, USA, 2006. IEEE Computer Society.

[17] D. Zissos and U. K. A. E. Authority. *Logic design algorithms / by D. Zissos*. Oxford University Press, London ; Toronto :, 1972.