ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **18**, Number 4, 2015, 299–311

On Additional Constrains in Lossless Compression of Text Files

Radu RĂDESCU

"Politehnica" University of Bucharest,
Faculty of Electronics, Telecommunications and Information Technology,
Applied Electronics and Information Engineering Department,
1-3, Iuliu Maniu Blvd, sector 6, Bucharest, Romania

E-mail: radu.radescu@upb.ro

Abstract. Coding algorithms are generally aimed at minimizing the output code length encoding speed once the code has been designed. Moreover, most codes use a binary alphabet. This paper examines other issues related to coding, such as additional constraints imposed on the channel. Code generation will be considered where there is a limit on code words. Limits the application of such systems is a practical example of data compression where fast decoding is essential. When all code words correspond to a single word in memory (usually 32 bits, but there are situations that take 64-bit) can be used canonical decoding. If the deadline cannot be guaranteed, however, required the use of slower decoding methods. This paper also deals with the alphabetic code generation, where lexicographic arrangement of words by their code symbols must correspond to the original order in which the symbols were taken coding system. When an alphabetic code is used to compress a database that can be sorted in the same, order they would have had if the database records were first decompressed and then sorted. It also corresponds to alphabetic code trees binary search trees, which have applications in a wide variety of search problems. Assumption that the symbols are sorted by probability is not suitable for this scenario. The problem of finding codes for non-binary alphabets channel will be examined in detail. The subsequent experimental results cover the problem of alphabetic coding and of limited length coding.

1. Introduction

Unequal-cost coding is the task of finding a code symbols when channel alphabet symbols cannot be considered to be of unit costs. For example, in an attempt to minimize the power consumption of new communications devices based on new technologies, it tries calculation code, given that a zero bit requires 10% more energy than a bit to be transmitted of 1. In this case, the code should contain more bits than 1 bit 0; bit 0, which are required to be present anyway. Similarly, engineers might appeal to notions related to information theory to find a minimum cost code for multi-symbol alphabet channel or three or more different symbols channels possibly cost differential that cost to be measured by energy consumption or time after transmission of these symbols.

Due to the ubiquitous use of minimum redundancy codes in software for data compression, a significant variation on the standard prefix coding problem is to develop a code when applying an upper limit on the length of code words. Such codes ensure the integrity of applications that use canonical coding, are also used in applications with code trees of limited depth because the length of the path to the deepest leaf of the tree is limited. Search applications requiring a weighted tree, but only in a limited number of steps, can use the algorithms presented below to construct such a search tree.

2. The Alphabetical Coding Algorithm

All encryption mechanisms are independent of the exact values of the symbols that were to be coded, and this flexibility has proven to be very useful, especially when at the input a sorted distribution of probabilities was assumed. However, there are situations where it is desirable that the input symbol sequence is preserved, which implies that the source alphabet is not changed. Such a situation is alphabetical coding.

Suppose an ordering $\ll$ is defined of a source symbols so that $i < j$ implying $s_i \ll s_j$. We also assume that we want to expand to $\ll$ the words of code, so that when $i < j$ and $s_i \ll s_j$ to have $c_i \ll c_j$. In other words, if the code words are sorted, the resulting sequence must be in the same order as the order in which the source symbols were sorted. A canonical redundancy code for an alphabet with minimum probability will always be an alphabetical code. On the other hand, the three codes derived from a strict application of Huffman's algorithm are not guaranteed to be alphabetical, even if the probability distribution of the source is supposed to be sorted.

The problem consists in generating an alphabetic code of an unsorted distribution of probabilities. Hu and Tacker developed in 1971 an algorithm to solve the problem and that was executed in time $O(n^2)$. Later, Knuth improved this implementation, using a left tree data structure. This algorithm generates a non-alphabetic binary tree code word which corresponding to each symbol can be read. Once the lengths of the code words are known the alphabetic code words are easily assigned.

The smallest sub-trees available will be packed, like Huffman's algorithm, except

that there are two differences. The first difference is that the new package format is not immediately inserted into a new list of packages sorted by size, but it replaces the component located in the leftmost. A second-important difference is the fact that only packets that are not separated from the leaves of the tree can be combined. To select two packages whose amount is minimal and are not separated by a leaf, Knuth uses a set of priority queues to keep the clues of the packages from the word limit L, which can be combined. Each priority queue in the set will be a collection of packages in which any object can be combined with any other. The queue is itself represented by its candidate pair (the two packs of lowest value) with a key equal to the sum of the two packages. In addition, a global *heap* is used to enable rapid identification of candidate pairs showing queues with the smallest key. That pair is then extracted and joined to a new package, all the structures being is updated.

3. Alternative Alphabet Channels

Until the alphabetical coding, the main target was to generate codes for a binary alphabet of a channel where bits of "0" and "1" are considered equal in terms of cost, cost measured in time, money or other such external quantities. Further, on, consider the code developed by Samuel Morse in 1835 for telegraph transmissions. Morse uses an alphabet of the channel consisting of "points" and "lines" where a line is, by definition, three times longer than a point.

The complete definition of Morse code also provides the permission to broadcast a one-point space between characters and a longer space between words. But in this case we ignore these additional difficulties, simply because for any message, the additional cost will be constant and will not be influenced by codes assigned to characters that include the message, once the spaces were removed.

In other words, if the messages consist of characters to be encoded in a way to spend more time, then a code must be built a code in which the cost of a "line" must be regarded as three times the cost of a "point". An immediate application of Morse code is used for data communication channels. For such a channel is a customary combination of 8-bit numbers is reserved for the control channel, and these bits will not appear when data is transmitted.

The cost of transmitting these special characters should be twice the cost of other combinations of 8 bits (bytes), if you want to maximize the channel capacity. In this case, the alphabet channel will consist of 256 symbols, where some subgroups have the cost of two units of time, while others have the cost of one unit of time. [5] The tree construction algorithm supposes that at each iteration of the *while* loop, the smallest sub-trees available are packed like in the Huffman algorithm.

4. The Limited Length Encoding Algorithm

Limited length encoding algorithm implemented in the application accompanying this paper is about the reversal joint mechanism package *(package reverse merge),* developed by Turpin and Moffat [5]. This algorithm derives from the simple combi-

nation mechanism package *(package merge),* the difference being faster operational space.

5. Experimental Results

In this paragraph, experimental results obtained from the above encoding methods on various types of input files will be observed. Compression experiments using the method of alphabetical coding were performed on a workstation with an Intel Core 2 Duo clocked at 1.66 GHz, 2 MB cache and 2 GB RAM at a frequency of 667 MHz. For testing, we used the following test corpora: *Calgary Corpus* and *Canterbury Corpus.* They are composed of a collection of files designed specifically to test applications of lossless compression methods (see Table 1). In addition to these algorithms, there will be performed tests on a number of files in Romanian. [11] *Calgary* corpus consists of 18 text files, which consist of more than 3 200 000 bytes.

Table 1. Type Sizes for Camera-Ready Papers

Name	Size [bytes]	Description	Type
glossa.txt	2 134	Eminescu Text	Text
luceafarul.txt	10 416	Eminescu Text	Text
stire.html	420	Source HTLM	Source
codul_ penal.txt	232 269	Text	Text
xml.xml	210 309	XML source	Source

It was collected by many researchers in 1987 to develop, test and compare different compression methods. Since a long time we worked on the same corpus, it was assumed that some compression methods have been constructed to be optimal in this case. Standard file format change has led to the development of a new corpus that is *Canterbury.* [12]

Besides the two corpora, already established, used in general testing of compression, we choose an additional corpus consisting of files in Romanian, that will note the alphabetical trees for the alphabetical coding, based on the frequencies of symbols in the Romanian language.

As you can see, we tried applying compression on many different types of files in order to see this behavior on their various compression algorithms. The further results will be refined on compression rate and compression time.

A. Compression Rate

To illustrate the compression coding algorithms using alphabetical coding a procedure implying compressing multiple files from the available corpora was accomplished. After applying the alphabetical coding, the following results were obtained from corpora files.

To measure performance compression algorithm the most suggestive index was chosen the compression rate, representing the output file size divided by the size of the input file. A good code will attempt to assign a high probability symbol a smaller

word length (due to repetition, we can economize) and those with a lower probability a longer word length.

As it can be seen in Fig. 1, by encoding the source with imposing an upper limit on the words, has led to compression rates of about 1.4÷1.5 for various test files from corpora.

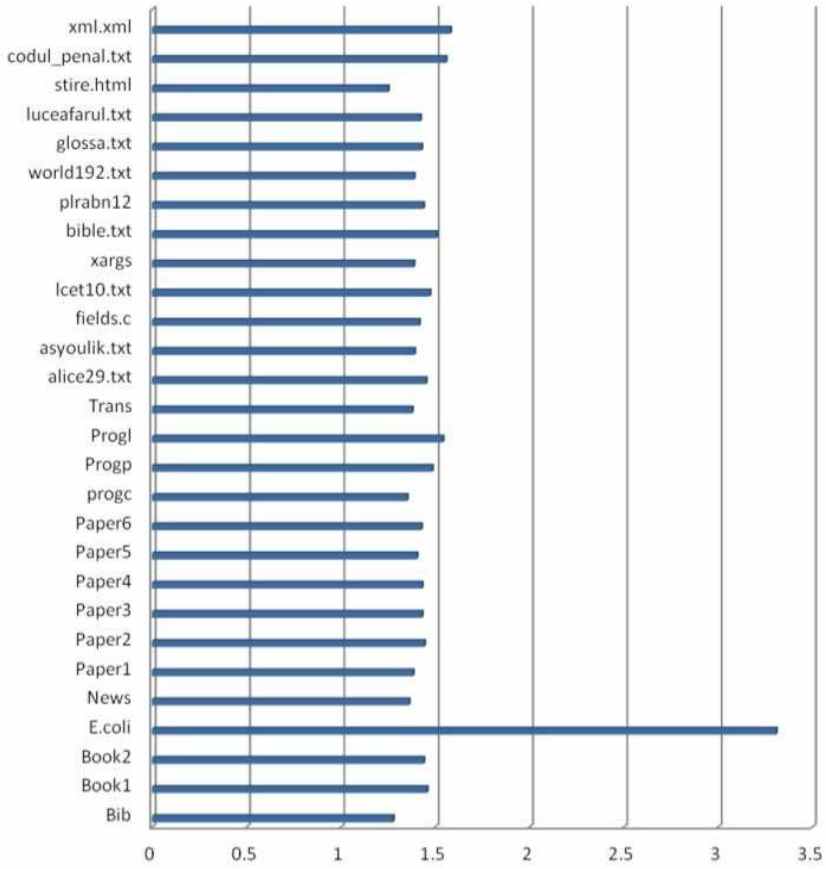

Fig. 1. Graphical representation of different file compression rates for alphabetical coding.

The file that is an exception and that achieved a much higher compression rate is the file E.coli. It contains the bacteria E.coli genome and the good obtained compression results can be explained by the small alphabet the file has, in which pairs of characters have much higher probabilities of occurrence than the rest of the files. The formed probability distribution for such a file will result in the formation of lists of coding that will produce much lower cost for words. For the rest of the files that are present in different formats (.txt, .c, .xml, .html), we obtained similar results as they are in English or Romanian language that will have an approximately similar alphabet, and therefore will present a uniform distribution of probabilities. However,

even in this case, imposing an L limit on the word led to satisfactory results even for such files. After compression by alphabetically encoding the source, the following results were obtained from the corpora files.

Regarding the alphabetical coding mechanism, the results for such compression are lower than the above, where an upper limit was imposed. Unlike the first case, the probability distribution of alphabetical encoding symbols will not be sorted leaving the alphabet unsorted and the order in the sequence of output symbols is the same as at the input. Thus, because the encoding mechanism is not independent of the values of the source symbols to be encoded and does not benefit from the flexibility offered by the previous algorithm, the compression rate is lower, especially where in the corpus files there are groups of characters that have a higher probability of occurrence. As can be seen in the graphic below, the experimental results were about equal, the file compression had a higher rate than most being the .xml file type that contains repetitive data.

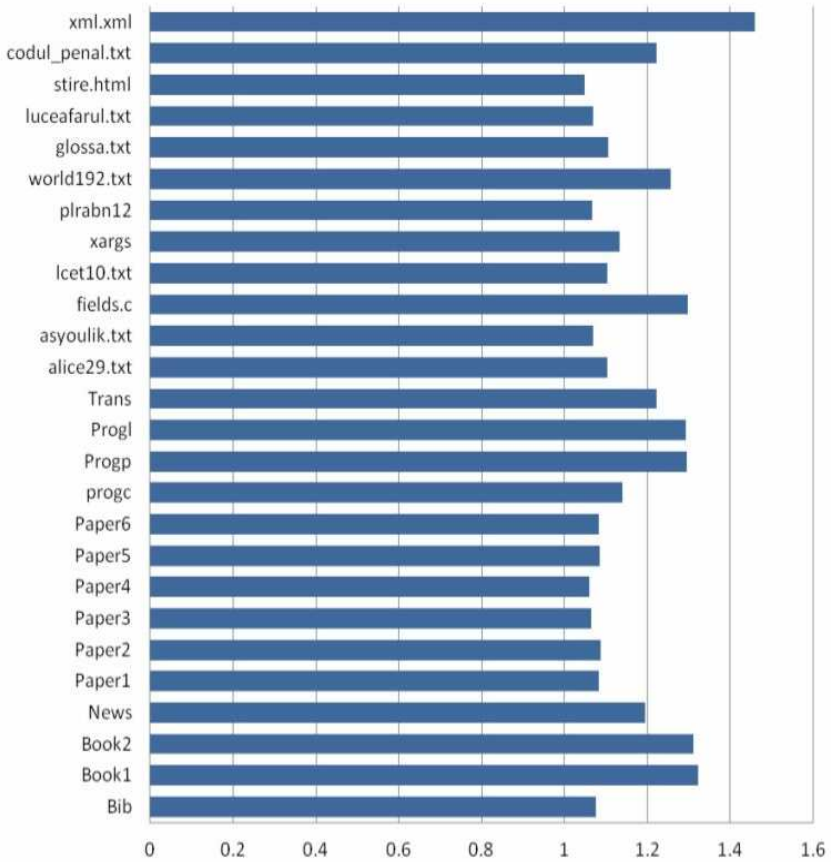

Fig. 2. Graphical representation of different file compression rates for limited length encoding.

To illustrate the compression coding algorithms using limited length, a procedure implying compressing multiple files from the available corpora was accomplished. After coding with limited length, the following results were obtained from corpora files.

To measure performance compression algorithm the most suggestive index was chosen the compression rate, representing the output file size divided by the size of the input file. A good code will attempt to assign a high probability symbol a smaller word length (due to repetition, we can economize) and those with a lower probability a longer word length. As it can be seen in Fig. 2, by encoding the source with imposing an upper limit on the words, has led to compression rates of about $1.4 \div 1.5$ for various test files from corpora. The file that is an exception and that achieved a much higher compression rate is the file E.coli. It contains the bacteria E.coli genome and the good obtained compression results can be explained by the small alphabet the file has, in which pairs of characters have much higher probabilities of occurrence than the rest of the files. The formed probability distribution for such a file will result in the formation of lists of coding that will produce much lower cost for words.

For the rest of the files that are present in different formats (.txt, .c, .xml, .html), we obtained similar results as they are in English or Romanian language that will have an approximately similar alphabet, and therefore will present a uniform distribution of probabilities. However, even in this case, imposing an L limit on the word led to satisfactory results even for such files.

B. Compression Time

Compression time obtained for this encoding method is exposed in Fig. 3 for the same files in the test corpora. In terms of obtained compression time, it is observed that the results differ quite significantly from one file to another.

For alphabetical encoding times are extremely varied, they mainly depend on the size of the files that have been compressed. Training trees, wrapping nodes and updating data structures are operations that prove to be time consuming compared to the formation of lists for the first mechanism of encoding.

In addition, the workstations on which the measurements were made have a significant influence on the obtained measurements. The strongest stations with high power can get much better results compared to stations that have weaker processing power. A graphic representation of the times obtained for the two algorithms is presented below.

Compression time obtained for this encoding method is exposed in Fig. 4 for the same files in the test corpora. In terms of obtained compression time, it is observed that the results differ quite significantly from one file to another. The more obvious advantage is for the limited-length coding in same particular cases.

For such an encoding the highest processing time was obtained for the file bible.txt file that is a file with a large number of bytes, the rest of the files resulting in very fast time which confirms once again that this encoding mechanism is extremely advantageous, especially when working with large files.

In addition, the workstations on which the measurements were made have a sig-

nificant influence on the obtained measurements. The strongest stations with high power can get much better results compared to stations that have weaker processing power. A graphic representation of the times obtained for the two algorithms is presented below.

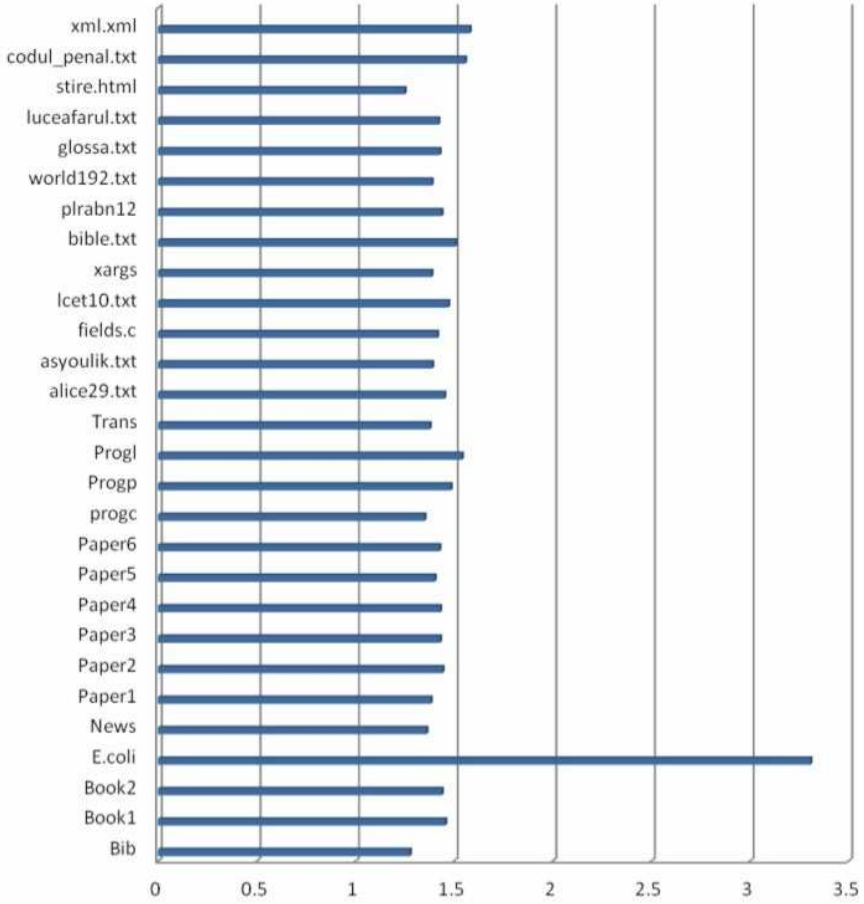

Fig. 3. Graphical representation of different file compression time for alphabetical coding.

C. Comparison to Other Compression Algorithms

The alphabetical coding was not subjected to comparison, because it represents a different mean of encoding besides the other methods that takes into account the order of the input symbols. In addition, results were already presented as weaker than the limited-length encoding algorithm, so this type of comparison like the one above is not suitable for this algorithm.

Next, we will evaluate the performance of the compression algorithm with limited length encoding relative to other compression algorithms. For comparison there will be used a commercial archive tool that uses different compression methods in order to optimize performance (WinRAR) and an algorithm using arithmetic coding. The results are displayed in Fig. 5.

As can be seen in the histogram, the encoding compression algorithm with limited length is surpassed by other methods of compression in all cases, even for the E.coli file for which it obtained a good result. The algorithm that has the highest average rate is WinRAR, followed by the arithmetic-coding algorithm and finally by the one studied in this paper. This result makes the old encryption algorithm to be seen as an inefficient algorithm, although it is a very fast algorithm. By far, the most efficient method is WinRAR, giving a compression rate of 54 for the file in Excel. The only file type that the result is worse is the *lcet10* file that is a text file containing technical data.

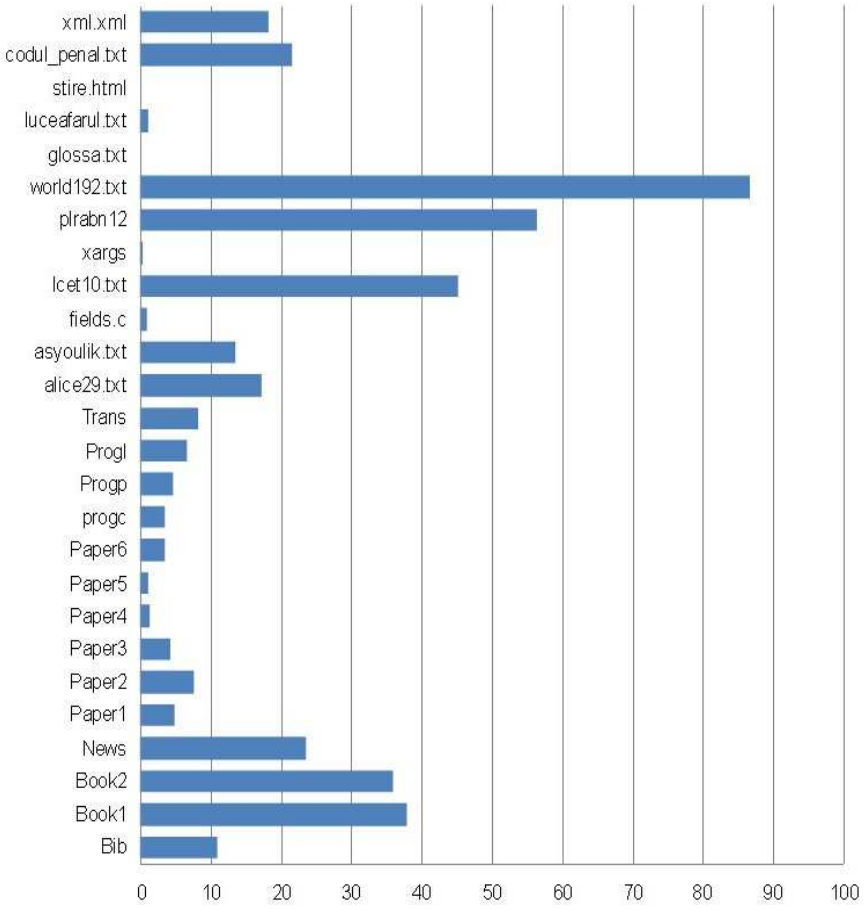

Fig. 4. Graphical representation of different file compression time for limited length coding.

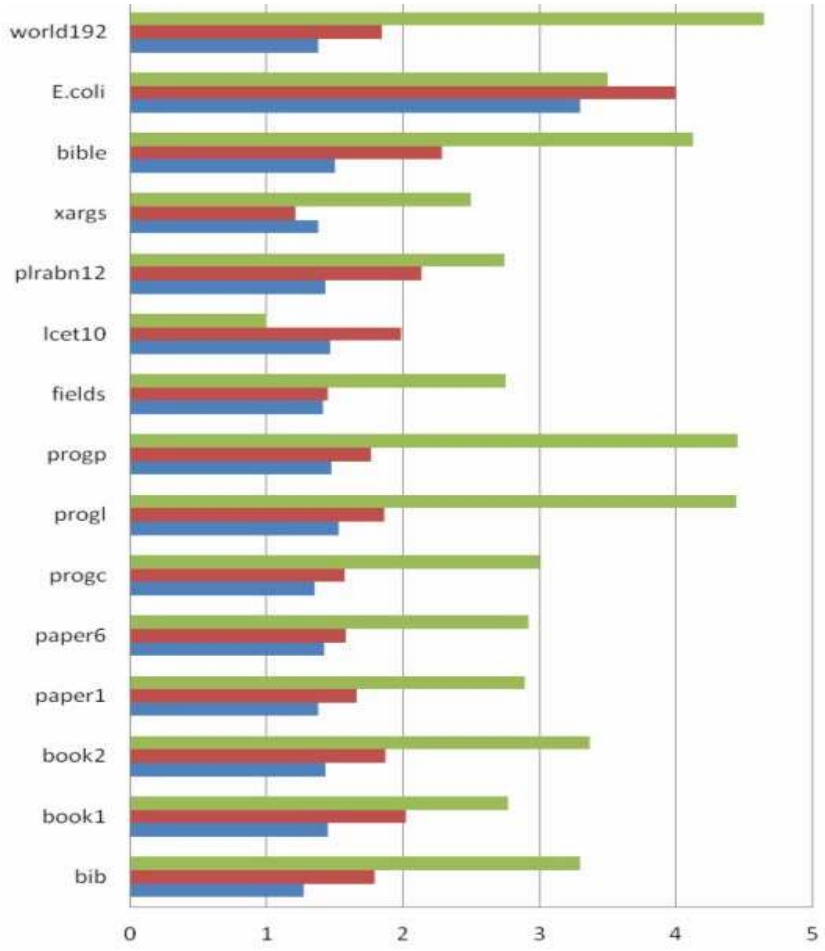

Fig. 5. Graphical representation of the compression rate for various compression algorithms: (1) green - WinRAR, (2) red - arithmetic coding, (3) blue - limited length coding.

6. Conclusions

This paper was meant to study the lossless compression when it is imposed with additional constraints on the source to be encoded. Thus, with the help of encoding algorithms, the arising lossless compression could be seen in the cases of both alphabetic and length-limited algorithms. An application with a graphical interface was created, allowing the user to view the method performances for different types of files that are selected using the dialog window. From the analysis, implementation and testing of both algorithms several conclusions can be drawn regarding the advantages and disadvantages of these methods.

The first thing that stands out is the high compression time, representing the main disadvantage of the alphabetic coding algorithm. However, as it is sensitive to context, the probability distributions are unsorted and obtaining an output sequence of symbols in the same order as at the input, this disadvantage can be ultimately minimized.

The compression rate for the alphabetic coding algorithm has values lower than the previous algorithm so it can be said that the overall compression mechanism has a weaker performance. In terms of implementation, it was more difficult to apply from an algorithmic point of view, requiring numerous data structures such as priority queues in which each element had a "priority" associated or global heap tail used to identify candidate pairs that have the smallest key value.

The Huffman algorithm likeliness and the almost similar way of building a tree, recommend this lossless data compression algorithm only if it is to retain the syntax. Otherwise, it is preferred to use a different compression method to achieve better compression rates and compression time.

Regarding the limited length encoding mechanism, it can be said that it gives a good performance and is relatively easy to implement and offers reduced execution times. The formation of lists on a descending sorted probability distribution, in which the items are packed according to their total impact on the overall cost, compiling packages using a recursive mechanism and simple data structures, such as queues, guarantees a good to very good compression of a variety of files.

For files that have a high frequency of symbols, such as the E.coli file, excellent results were obtained both in terms of compression rate and compression time. For the remaining files, especially for the plain text files, there have been achieved good results, even if we are talking about large files that do not have very high frequencies of occurrence of symbols. In terms of losses the results for the limited length coding method, we can say that it is generally very small for such a code length that is never less than a minimum redundancy code.

References

[1] SALOMON D., *Data Compression – The Complete Reference*, 3rd Edition, Springer, 2003.

[2] RĂDESCU R., *Lossless compression – methods and applications*, Matrix Rom Publishing, Bucharest, 2003.

[3] `http://www.cs.cmu.edu/~guyb/realworld/compression.pdf`, accessed on 18.02.2012.

[4] NELSON M., GAILLY J.L., *The Data Compression Book*, 2^{nd} Edition, M&T Books, 1995.

[5] MOFFAT A., *Compression and Coding Algorithms*, 2002.

[6] TELL T.C., CLEARY J.G., WITTEN I.H., *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[7] `http://www.ginfo.ro/revista/13_1/serial.pdf`, accessed on 12.05.2012.

[8] RĂDESCU R., *Lossless Compression Tool for Medical Imaging*, Proceedings of the 6th International Symposium on Advanced Topics in Electrical Engineering, ELTH & AIEER, pp. 265–267, November 20-21, 2008, Bucharest, Romania, Printech Press.

[9] ALLEN B., MUNRO I., *Self-organizing search trees*, J. ACM. **25**, 4 (Oct. 1978), pp. 526–535.

[10] RĂDESCU R., *Star-Derived Transforms in Lossless Text Compression*, Proceedings of the IEEE CAS International Symposium on Signals, Circuits and Systems – ISSCS2009, pp. 291–296, July 9-10, 2009, Iaşi, Romania, ISBN 9781-4244-3786-3.

[11] The testing Calgary Corpus: `ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/`, accessed on 10.06.2012.

[12] The testing Canterbury Corpus: `http://corpus.canterbury.ac.nz`, accessed on 02.04.2012.

[13] RĂDESCU R., LICULESCU G., *Efficient Implementation of Adaptive Huffman Methods in Lossless Compression*, Proceedings of the Fifth International Workshop on Optimal Codes and Related Topics OC2007, Balchik, Bulgaria, 16-22 June, 2007, pp. 209–215, ISSN 1313-1117.

[14] RĂDESCU R., *Lossless Compression – Algorithms and Applications*, MatrixRom, Press Bucharest, 2003.

[15] RĂDESCU R., *Digital Transmission of Information. Data lossless compression – Practical Guide*, Politehnica Press, Bucharest, 2009.

[16] RĂDESCU R., BONTAŞ C., *Design and Implementation of a Dictionary-Based Archiver*, Scientific Bulletin, Electrical Engineering Series C, Polytechnic University of Bucharest, vol. **7**, nr. 3, pp. 21–28, 2008.

[17] RĂDESCU R., *Transform Methods Used in Lossless Compression of Text Files*, Romanian Journal of Information Science and Technology (ROMJIST), Publishing House of the Romanian Academy, Bucharest, vol. **12**, nr. 1, pp. 101–115, 2009, ISSN 1453-8245.

[18] RĂDESCU R., *Lossless Compression Tool for Limited Number of Colors*, Scientific Bulletin, Electrical Engineering Series C, Polytechnic University of Bucharest, vol. **71**, nr. 2, pp. 49–54, 2009, ISSN 1454-234X.

[19] RĂDESCU R., *New Table Look-Up Lossless Compression Method Based On Binary Index Archiving*, International Journal of Engineering, Science and Technology (IJEST), Vol. **1**, No. 1, pp. 283–290, 2009, e-ISSN 0975-5462.

[20] RĂDESCU R., KING R., *Table Look-Up Lossless Compression Using Index Archiving*, Proceedings of the IEEE International Geoscience & Remote Sensing Symposium IGARSS2009: Earth Observation – Origins to Applications, Vol. **1**, pp. 5–8, July 12-17, 2009, Cape Town, South-Africa, ISBN 978-1-4244-3395-7.

[21] RĂDESCU R., *Star-Derived Transforms in Lossless Text Compression*, Proceedings of the IEEE CAS International Symposium on Signals, Circuits and Systems – ISSCS2009, pp. 291–296, July 9-10, 2009, Iaşi, Romania, ISBN 9781-4244-3786-3.

[22] RĂDESCU R., *A New Operating Tool for Coding in Lossless Image Compression*, Proceedings of the Sixth International Workshop on Optimal Codes and Related Topics (OC2009), pp. 163–167, Varna, Bulgaria, 16-22 June 2009, ISSN 1313-1117.

[23] RĂDESCU R., *An Interactive Application for the Study of Delta Compression*, Proceedings of The 6th International Seminar on the Quality Management in Higher Education

(QMHE 2010), Book II, pp. 625–626, 8-9 July 2010, Tulcea, Romania, UT Press, Cluj-Napoca, ISBN 978-973-662-566-4.

[24] RĂDESCU R., *Building An E-Course For Interactive Learning Of Lossless Compression*, Proceedings of The 6th International Seminar on the Quality Management in Higher Education (QMHE 2010), Book II, pp. 627–630, 8-9 July 2010, Tulcea, Romania, UT Press, Cluj-Napoca, ISBN 978-973-662-566-4.

[25] RĂDESCU R., *LIPT-derived Transform Methods Used in Lossless Compression of Text Files*, Scientific Bulletin, Polytechnic University of Bucharest, Series C: Electrical Engineering and Computer Science, Vol. **73**, Issue 2, 2011, pp. 63–72, ISSN 1454-234x.

[26] RĂDESCU R., *LIPT-Derived Transform Methods Used in Lossless Compression of Text Files*, Romanian Journal of Information Science and Technology (ROMJIST), Publishing House of the Romanian Academy, Bucharest, vol. **14**, nr. 2, pp. 149–158, 2011, ISSN 1453-8245.