www.czasopisma.pan.pl PAN www.journals.pan.pl
INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2010, VOL. 56, NO. 4, PP. 351–355
Manuscript received May 11, 2010; revised October 2010. DOI: 10.2478/v10177-010-0046-8

Joint Source-Channel Coding in Dictionary Methods of Lossless Data Compression

Marcin Rodziewicz

***Abstract*—Limitations on memory and resources of communications systems require powerful data compression methods. Decompression of compressed data stream is very sensitive to errors which arise during transmission over noisy channels, therefore error correction coding is also required. One of the solutions to this problem is the application of joint source and channel coding. This paper contains a description of methods of joint source-channel coding based on the popular data compression algorithms LZ'77 and LZSS. These methods are capable of introducing some error resiliency into compressed stream of data without degradation of the compression ratio. We analyze joint source and channel coding algorithms based on these compression methods and present their novel extensions. We also present some simulation results showing usefulness and achievable quality of the analyzed algorithms.**

***Keywords*—Channel coding, joint source-channel coding, lossless data compression LZ'77, LZSS, source coding.**

I. Introduction

DATA which have to be transmitted through the channel characterized by limited capacity should be the subject of source coding (compression) in order to meet the limitations set by the transmission channel. However, compressed stream is very prone to transmission errors. Single error can cause many more errors in the decompression process, which leads to significant corruption of data. Therefore, in situations where errors are very probable the compressed data stream has to be protected by an appropriate channel code.

Lack of error resiliency in lossless data compression methods is a long-standing problem. Dealing with this problem is not a simple task. The goal of source coding is to decorrelate the data coming from the source of information by minimizing redundancy, whereas the aim of channel coding is to provide error resiliency by introduction of additional correlation. Transmission of unprotected compressed data is risky because single error anywhere in the compressed stream can prevent from decoding the data correctly. This flaw excluded lossless compression methods from being used in many applications. However joint source-channel coding is emerging as a possible solution to this problem.

The idea behind the joint source-channel coding based on dictionary methods of data compression is to provide error resiliency without serious degradation of compression ratio (ratio between compressed and uncompressed data size). It is achievable for instance by exploiting the fact that some of the dictionary based algorithms leave some redundancy in the compressed stream. Thanks to this seemingly undesirable feature it is possible to introduce some error resiliency to the compressed stream by embedding additional information into it. This information could be responsible for detecting and correcting errors in the compressed stream. However, the method for embedding the additional information depends on the compression algorithm used.

This paper addresses joint source-channel coding for two algorithms belonging to the family of dictionary compression methods. These are LZ'77 (Lempel-Ziv'77) and LZSS (Lempel-Ziv-Storer-Szymanski) algorithms. This paper is organized as follows. The next section describes the way for obtaining redundant bits in the aforementioned algorithms. Section III introduces different methods for embedding channel codes into compressed stream and describes the joint source-channel algorithms proposed by the author of this paper. Finally, in Section IV numerical results obtained from simulations are presented whereas conclusions are made in Section V.

II. Redundant Information in LZ'77 and LZSS Algorithms

A. *LZ'77 Algorithm*

The method for obtaining redundant bits in the LZ'77 algorithm originates from [1] and was further analyzed in [2] and [3]. Firstly, let us remind the operation of the standard LZ'77 scheme.

The LZ'77 algorithm belongs to the family of dictionary compression methods [4]. The encoder processes the input data by parsing the stream from left to right. During this process, it looks into the sequence of past symbols, stored in the search buffer (dictionary), to find a phrase matching to the longest prefix of the string starting in the current position i.e. at the beginning of look-ahead buffer. The search buffer and look-ahead buffer form a window. The match is substituted by a token consisting of three elements (*position*, *length*, *symbol*). As soon as the match is found the data in the window is shifted by $length + 1$ positions to the left i.e. the first $length + 1$ symbols of the look-ahead buffer are appended to the search buffer, and similarly a sequence of the same length of the following input symbols is appended to the look-ahead buffer. The LZ'77 algorithm is commonly known as a sliding window algorithm because the input stream is shifted from right to left in that window during the encoding process.

Knowing the principles of operation of the LZ'77 scheme it can be noticed that the algorithm leaves some implicit redundancy in the compressed stream which can be used for instance for ensuring error resiliency. This redundancy comes from the fact that for some phrases the encoder can issue more

M. Rodziewicz is with the Chair of Wireless Communications at Faculty of Electronics and Telecommunications, Poznań University of Technology, Polanka 3, 60-965 Poznań, Poland (e-mail: marcin.rodziewicz@put.poznan.pl).

than one possible token. In practice if there are q copies of the longest matching phrase it is possible to recover $\lfloor \log_2 q \rfloor$ bits by selecting one of q tokens. In order to embed additional information used for error correction, the LZ'77 algorithm needs to be slightly modified. The resulting algorithm allows embedding bits of message M formed from binary symbols. If the longest matching phrase for the current look-ahead buffer sequence has $q > 1$ matching phrases in the search buffer, the encoder can choose one of the multiple copies of that phrase. The next $\lfloor \log_2 q \rfloor$ bits of M will drive the selection of one token (Fig. 1).

The full description of the encoding algorithm can be found in [1]. This algorithm will be referred as the LZS'77 algorithm as in [1].

To exploit the embedding capability of the LZS'77 compression algorithm, the standard LZ'77 decompression algorithm needs to be modified. The standard LZ'77 decoder forms its own buffer as it reads the tokens on its input. This buffer is used to retrieve the uncompressed sequences of symbols from the input tokens. In order to be also able to retrieve embedded data the decoder has to parse its buffer in order to find possible multiple copies of the sequence that is currently being decoded. This requires additional time needed for decompression process. The more detailed description of LZS'77 decoder can be found in [1].

B. *LZSS Algorithm*

The LZSS method is based on the LZ'77 scheme and its principles are very similar. However, it introduces several changes to the original algorithm. The main difference is that the encoders' output token consists of two fields i.e. (*position*, *length*). In practice it means that the encoder allows to mix uncompressed data with compressed data. The uncompressed data is issued by the encoder when the bit-length of a token is longer than the bit-length of uncompressed data. For this to work the encoder has to indicate what the type of the output symbol is. It is done by adding an additional bit before each output symbol.

Since LZSS and LZ'77 schemes are similar, the method for embedding and recovering additional information into and from the LZSS compressed stream is basically the same.

III. Adding Error-Resiliency in LZ'77 and LZSS

This section describes how to use redundant bits for error correction and presents novel joint source-channel coding algorithms.

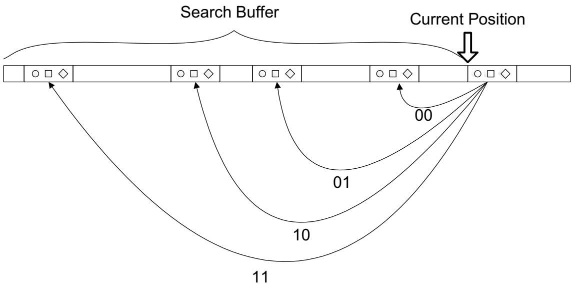

Fig. 1. Selection of one of the four available tokens recovers two additional bits [1].

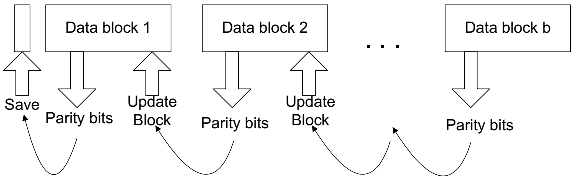

Fig. 2. The reverse order of encoding process [1].

Since both LZ'77 and LZSS algorithms operate on strings and as a result of encoding we obtain the stream of tokens i.e. the sequence of bytes, the primary selection for channel codes are Reed-Solomon codes over GF(2^8). An RS code is specified as RS(n,k) where n is the size of a codeword and k is the size of data block to be protected. The RS encoder adds $n-k$ parity symbols to data consisting of k symbols, resulting in a codeword of size n. The code rate is defined as k/n . The correction ability of a RS code is defined as $e = (n-k)/2$ which means that the decoder can correct up to e errors in the block. Reed-Solomon codes are codes capable of correcting burst errors so a single error occurs when one or more bits within the symbol are wrong.

Using the LZS'77 algorithm it is possible to embed $2e$ extra bytes into the compressed stream. Because the number of redundant bytes highly depends on the type of data to be compressed, there are different methods for achieving error-resiliency. These methods share some common features though. The encoder, name it LZRS [1], has to process the data in two steps. Firstly, it has to encode the input stream with a standard algorithm. After this operation, it is possible to embed the parity bits into the stream. This is because the parity bits have to be known before they can be embedded. Secondly, the encoder divides the compressed stream into blocks of a certain size and then it processes the blocks in reverse order beginning from the last one. The size of the block depends on the channel code used. During processing of the block i the encoder computes the parity bits for the block $i+1$ and then embeds them into the block i using the method described in LZS'77 algorithm. The first block of the compressed stream is a special case because there is no block to embed the additional bytes so the parity bits for this block are added at the beginning of the stream. The encoding process is shown in Fig.2.

As mentioned before, the number of redundant bits available for channel coding is highly dependent on input type i.e. entropy of the source, alphabet size etc., therefore, error resiliency introduced by the aforementioned algorithm is also dependent on the same factors. In the encoding process the data compressed in the first phase is later divided into blocks. Let us consider some internal structures of these data blocks. Authors of paper [1] assumed the data block structure as in Fig. 3a. In this case the data block consists only of the compressed symbols and all the bits responsible for error correction are embedded into the block. The size of each block is constant. Therefore the encoding process is the same as described above. Let us denote the method exploiting this block structure as the ELZ algorithm for the LZ'77 and ELZSS for the LZSS algorithm. This solution has an advantage of

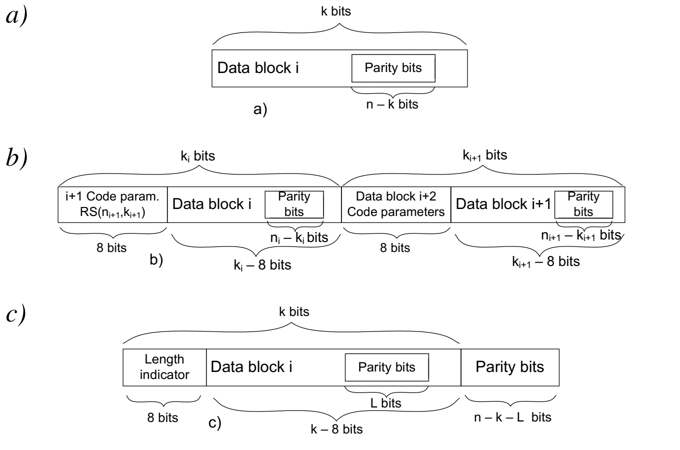

Fig. 3. Data block structures for a) ELZ algorithm b) VRLZ algorithm c) CLZ algorithm.

being backward compatible, i.e. data compressed using this block structure can be decompressed by a standard LZ'77 decoder. However, such organization of the blocks leads to inefficient use of available redundant bits. It is so, because the maximum number of redundant bits that can be embedded into the block is determined by the data block with the smallest embedding capability. Since the algorithm assumes only one channel coder, the number of parity bits is constant. It may occur that in some data blocks not all redundant bits available are utilized. Due to this the number of errors that can be corrected can be relatively small.

More efficient utilization of available redundant bits can be achieved by changing the assumption that we can use only one channel coder with fixed parameters, to the assumption that we can use a set of coders. Therefore for each block a different channel code can be used. In this case the lengths of data blocks are variable. The length of the block i $(i \geq 1)$ is determined by the embedding capability of the block $i-1$. In order to achieve variable correction capability we have to slightly modify the encoding algorithm. The proposed algorithm will be referred to as the VRLZ algorithm. The encoding process still takes place in two phases, but during the first phase the encoder, apart from compressing the input stream, determines the embedding capability of each block and stores this information in its memory. This information is further used for determining parameters of the channel code. In the second phase data blocks are processed in reverse order in the following manner. While processing data block $i+1$ the encoder reads the information about channel code parameters associated with data block i. Based on this information it chooses the proper channel coder and computes the parity bits for block $i+1$ (comprising of the compressed stream and code parameters indication field). These bits are later embedded into the block i. The exceptions in this process are the last and the first block of the stream. The last block consists only of compressed symbols and no other additional information, whereas the parity bits for the first block are added to the beginning of encoder's output stream. The parameters of the code used for the first block are set at the beginning of the encoding process. These parameters also have an impact on the number of redundant bits in the following blocks. If the first block code rate is small (i.e. the data block size is smaller), then the number of bits that can be embedded into the block decreases. On the other hand if the code rate is high, then the number of redundant bits will be greater, but the protection of this data block will be lower. Therefore, it is important to choose the right code rate. The structure of data blocks for the VRLZ algorithm is presented in Fig.3b. As we can see, at the beginning of each data block (except for the last block) there is an 8-bit field holding the information about the code parameters for the next block. This information is necessary for the correct decoder operation, because it allows us to separate consecutive blocks and determine the channel code used in each block. As it was mentioned before, the advantage of VRLZ algorithm is that it efficiently utilizes the available redundant bits. However, the drawback of the VRLZ is that the stream compressed with this algorithm cannot be decompressed with the standard LZ'77 decoder. It would be possible if we omitted the 8-bit code parameters information, but then no error correction is possible. Nevertheless, the gain in higher error resiliency and better utilization of available redundant bits seems to be worth sacrificing backward compatibility.

Although the application of the VRLZ algorithm increases error resiliency, it is still limited by the number of available redundant bits. Another approach goes around this problem by allowing to add some parity bits to the end of each block of the compressed data. Fig. 3c depicts this block structure. The proposed corresponding encoding algorithm, called CLZ, looks as follows: the first phase of the CLZ algorithm is the same as in the ELZ. The second phase is different in such a way that if the encoder cannot embed all the parity bits computed for the block i into the block $i-1$ then the remaining part of the parity bits is appended to the block $i-1$. At the beginning of each data block there is a length indicator that holds the information how many bytes of not embedded parity bits are added at the end of the block. This method allows to use channel codes of fixed but higher rates, and even though adding some parity bits directly to the compressed stream reduces the compression ratio, the achieved error resiliency could be much higher than in the case of the ELZ algorithm. Another drawback, of course, is the lack of backward compatibility.

Although the aforementioned algorithms use different block structures, the basics of the decoding process are identical for each of the algorithms. The decoder decodes the compressed stream and retrieves the parity bits embedded into the block

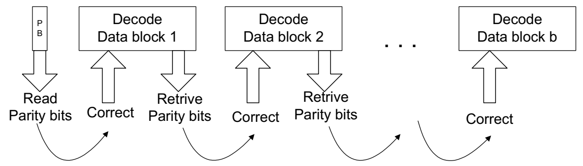

Fig. 4. Decoding process. Retrieving of the embedded bits allows correcting of potential block errors.

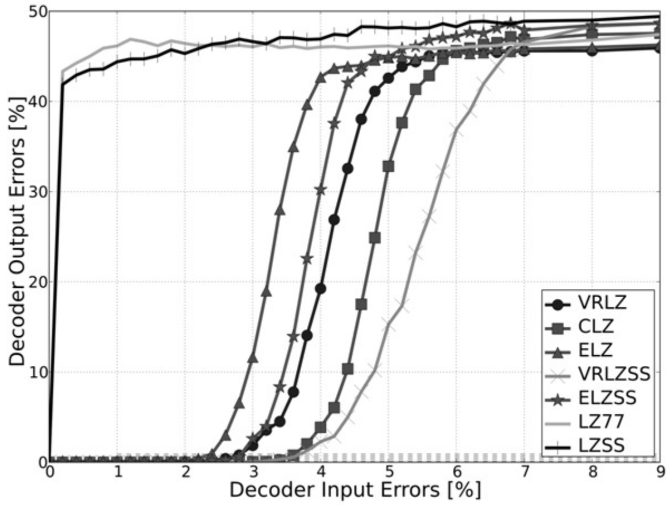

Fig. 5. Average decoder output errors as a function of decoder input errors.

i and uses them to correct the block $i+1$ if it is possible. Fig.4 presents the basics of the decoder's operation. Obviously the operations executed by the decoder for the CLZ and VRLZ algorithms include more than just retrieving embedded parity bits. In the VRLZ method the decoder at first has to read the code parameters field and separate the corresponding blocks. Only after that it is possible to retrieve the embedded information correctly. In the CLZ algorithm the decoder has to read the block's length indication field and append the remaining parity bits to the ones retrieved from the block.

IV. Experimental Results

In the evaluation of the proposed solutions a simple scenario was simulated. This scenario included the encoder and decoder corresponding to the selected algorithm and a file corruption block. The function of the file corruption block was to inject statistically independent errors into the compressed stream. The number of errors was expressed as a percentage of bit errors in the compressed stream. The file used for performing compression and decompression was a bit map picture (.bmp) file of size of approximately 130 KB. The selection of this file was based on previous studies carried out in [5]. The bit map file had a decent number of redundant bits available, therefore better error resiliency can be achieved. This allowed for better presentation of the advantages of the proposed algorithms.

TABLE I
Reed-Solomon Codes Parameters and Corresponding Compression Ratios for Different Algorithms

Algorithm	Reed-Solomon Code parameters	Compression Ratio
LZ'77	No channel code	18.79%
LZSS	No channel code	14.20%
ELZ	(255,211)	18.83%
CLZ	(255,197)	19.34%
VRLZ	(255,197)	18.93%
ELZSS	(255,205)	14.24%
VRLZSS	(255,179)	14.33%

TABLE II
Comparison of VRLZ and VRLZSS Algorithms

First Data Block Code parameters		VRLZ	VRLZSS
RS(255,225)	Compression Ratio	18.91%	14.30%
	Total no. of bits available	61311	65836
	Average no. of bytes per block	59.04	76.70
	Min. no. of bytes in block	36	62
RS(255,195)	Compression Ratio	18.93%	14.32%
	Total no. of bits available	61315	65803
	Average no. of bytes per block	58.94	76.53
	Min. no. of bytes in block	40	56
RS(255,165)	Compression Ratio	18.96%	14.35%
	Total no. of bits available	61306	65815
	Average no. of bytes per block	58.93	76.62
	Min. no. of bytes in block	36	54

The graph in Fig. 5 shows the relation between percentage of bit errors on input of the decoder and average percentage of bit errors on output of the decoder. Curves representing the results for the LZ'77 and LZSS algorithms are given for reference. As it can be noticed, to some point the percentage of output errors is equal to zero or almost zero for the curves related to joint source-channel coding algorithms. However, if the percentage of errors on decoder's input exceeds this point, the percentage of output errors starts to rise dramatically to reach its maximum value of about 50%. This point is determined by the channel code used, whereas the parameters of the code depend on the number of available redundant bits. Different compression algorithms allowed using different channel codes. The channel code parameters for each algorithm are presented in Table I along with the corresponding compression ratio. The VRLZSS algorithm is a LZSS variant of the VRLZ algorithm. Since the channel codes used in the VRLZ and VRLZSS algorithms are variable, the code rates for these algorithms included in Table I are an integer part of a code rate averaged over all the codes used. Analysis of Table I and Fig.5 leads to the conclusion that the CLZ, the VRLZ and also VRLZSS algorithms achieve higher error resiliency than the ELZ and ELZSS algorithms. However, there is a cost of slightly lower compression ratio and lack of backward compatibility. The results presented in Table I show that the degradation of compression ratio due to application of the joint source-channel algorithms is minimal. The results presented in Fig. 5 and in Table I prove what was expected i.e. better utilization of available redundant bits by using the VRLZ or CLZ algorithm as compared to the ELZ algorithm.

Table II presents the comparison of the VRLZ and the VRLZSS algorithms in terms of compression ratio, total number of available redundant bits and average number of redundant bytes per block and minimal number of redundant bytes among all the blocks. This comparison is made for three channel codes of different rates. The minimum number of redundant bytes in a block is an important parameter because it determines the minimal number of errors in the block that can prevent from correctly decoding the data. The value of this parameter in Table II does not take into account the code

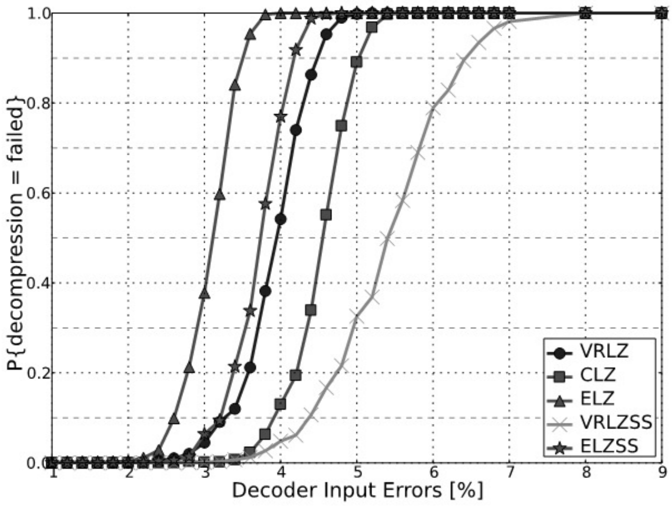

Fig. 6. Probability of decompression failure as a function of percentage of errors on decoder input.

parameters of the first block. An average number of redundant bytes per block serves as a reference to a coder with fixed parameters. By analyzing the content of Table II we can come to the conclusion that there exists a certain channel code used for the first block that maximizes the minimal number of bytes in a block for the VRLZ algorithm. The situation is a little bit different for the VRLZSS algorithm, namely, the higher the code rate the greater the minimal number of redundant bytes in the block. However, the higher the code rate the less protected the first block is. Comparing other parameters we can see that the VRLZSS offers better compression ratio and also greater number of available redundant bits than the VRLZ algorithm. It is so, because the LZSS algorithm parses the input stream more slowly, i.e. for the matching phrase of length l the window content for the LZSS algorithm is shifted by l positions to the left, whereas for the LZ'77 algorithm this shift is equal to $l+1$ positions. Since the LZSS encoder step is smaller, the dictionary (search buffer) fills up more slowly so there is higher probability of finding more copies of the same phrase than in the LZ'77 algorithm. This leads to a higher number of redundant bits available.

Fig. 6 presents the probability of decompression failure as a function of percentage of errors on decoder's input. As we can see, the worst error resiliency is achieved by the ELZ algorithm. The probability of successful decompression is above 0.9 if the errors on input do not exceed 2.6% for this algorithm. For the CLZ algorithm the percentage of input errors, if we aim to achieve more than 0.9 probability of successful decompression, cannot exceed 4%, whereas for the VRLZ and VRLZSS algorithms this percentage cannot exceed 3.4% and 4.4% respectively. What is more, the probability of an unsuccessful decompression increases more slowly for the VRLZSS algorithm in comparison to other algorithms. These results show that for the LZ'77 algorithm the optimal joint source-channel coding algorithm in terms of error resiliency and compression ratio is the VRLZ algorithm. The same applies to the LZSS algorithms.

V. Conclusion

In this paper joint source-channel coding in dictionary methods of data compression was analyzed. It was shown that it is possible to exploit some redundancy left by compression algorithms for error correction capabilities without the degradation of compression. Several solutions on how to embed information responsible for error correction, were presented.

In experimental studies the algorithms proposed in the paper were compared. They proved that the ELZ algorithm based on the algorithm described in [1] does not utilize the redundant bits efficiently, but still it can be used in situations where we are sure that the number of errors will be low and we want to keep the backward compatibility of the compression process. The results also showed that by sacrificing the backward compatibility we can ensure better error resiliency by allowing using channel codes with variable parameters. The proposed VRLZ algorithm realized these assumptions. Another approach was to use a channel code with high error correction capabilities and embed as many parity bits as we can into the compressed stream (CLZ algorithm). It guaranteed better error resiliency at the cost of a slightly degraded compression ratio.

Simulations were carried out on two compression algorithms belonging to the family of dictionary methods, namely LZ'77 and LZSS. The result presented in this paper are limited to one file for better and clearer comparison of presented algorithms. The conclusions made for tested file was verified in more simulation carried out in [5].The obtained results proved that the LZSS algorithm offered both better compression ratio and more available redundant bits for the studied file. The VRLZSS and ELZSS algorithms behaved better compared to their LZ'77 variants VRLZ and ELZ, respectively.

References

[1] W. Szpankowski and S. Lonardi, "Joint source-channel LZ'77 coding," in *IEEE Data Compression Conference*, 2003.

[2] W. Szpankowski, S. Lonardi, and M. D. Ward, "Error resilient LZ'77 scheme and its analysis," in *International Symposium on Information Theory*, 2004.

[3] W. Szpankowski, S. Lonardi, and M. D.Ward, "Error resilient LZ'77 data compression: Algorithms, analysis and experiments," *IEEE Transactions on Information Theory*, vol. 53, no. 5, May 2007.

[4] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transaction on Information Theory*, May 1977.

[5] M. Rodziewicz, "Investigation of source coding supported with channel coding," Master's thesis, Dept. Electronics and Telecommunications, Poznan University of Technology, Poznan, Poland, 2009.

[6] D. Salomon, *Data Compression: The Complete Reference*. Springer-Verlag, 2004.

[7] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley & Sons, 2006.