A New Method of Predictive-substitutional Data Compression

Zdzisław Szyjewski, Jakub Swacha
University of Szczecin, Szczecin, Poland
jakubs@uoo.univ.szczecin.pl

Abstract: The growth of data interchange within computer networks, including the Internet, causes bandwidth requirements to increase. Instead of widening the bandwidth, the same may be obtained by limiting the amount of interchanged data through an application of data compression. There are several general compression methods, varying in speed and efficiency. In this paper a new method of data compression is described, that combines elements of known methods to achieve better compression efficiency in less time. Performed tests show that this method achieves good results, especially for text data, so it could be used in network-related applications in order to limit the amount of interchanged data.

Key words: bandwidth saving, data compression, compression methods

1. INTRODUCTION

The Information Technology development, comprising almost every sphere of life and science, effects in huge data volumes, which are stored in computers and transmitted between them. Nowadays, one of the basic data interchange channels is the Internet. Independently of providers' expenditures on technical infrastructure for efficient data transfer, users still struggle with bottle-necks, in which there is excess of data to be transmitted over the connection's bandwidth.

This problem may be solved twofold, by:

- further increase of connections' bandwidth
- minimalization of transmitted data stream.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35616-7_23](https://doi.org/10.1007/978-0-387-35616-7_23)

W. Cellary et al. (eds.), *Internet Technologies, Applications and Societal Impact*
© IFIP International Federation for Information Processing 2002

The increase of connections' bandwidth is a technical problem, with conditions of its own. It is not subject to our interest. The minimalization of transmitted data stream should not result in restrictions for communication between network users. The solution should be sought in such data processing methods, that are capable of limiting the bandwidth usage by reducing the transmitted data capacity in an efficient way. One of such methods is data compression, applied to transmitted data.

Studies made in prior research work in using caching architectures to compress data on the network demonstrate that significant share of the bandwidth can be saved by compressing the exchanged data [Velasco and Lucianez 1998]. Applying compression to a link between two computers improves its average transfer rate and reduces the number of transmitted data packets, so the likelihood of lost packets and retransmissions is also reduced.

There are many data compression methods known [Sayood 2000]. Some of them are appropriate for general data compression, others are intended for selected data types only, for instance multimedia. There is research going on, focused on looking for new efficient data compression methods. In this paper, we propose a new data compression method, which if applied in data transmission process may result in better transmission efficiency.

2. LOSSLESS DATA COMPRESSION METHODS

Lelewer and Hirschberg in their work *Data Compression* acknowledge data compression is *a branch of information theory in which the primary objective is to minimize the amount of data to be transmitted* [Lelewer and Hirschberg 1987]. The difficulty of doing this minimalization lies in the need of keeping the transmitted content unchanged. Considering whether the content is kept exact, or minor differences are allowed, there are lossless and lossy data compression methods. This paper refers to the former only.

The way to reduce the amount of data to be transmitted is to remove its excessive part, and leave the part which determines the meaning of information. Most data, interchanged between people every day, contains excessive elements. In real-life conditions redundancy may be a desirable property for reasons such as communication channel's activity rules, or elimination of transmission errors. In the digital world, redundancy is usually *redundant*.

A measure for data source's information content is its entropy H , which may be calculated using following expression [Shannon and Weaver 1949]:

$$H = \sum_{i=1}^n -P(a_i) \lg P(a_i)$$

where: n – number of symbols in the source, $P(a)$ – probability of symbol a occurrence, $\lg$ – logarithm of base 2.

Knowing the entropy, it is easy to find the redundancy:

$$R = \sum_{i=1}^n P(a_i) L_i - H$$

where: L – codelength, i.e. the number of bits it takes to encode i th symbol.

From these equations it appears that redundancy disappears ($R = 0$), if the codelength L is equal to $-\lg P(a)$. Accomplishing that is possible owing to statistical coding, an operation during which every source symbol is given a code, long adequately to its probability of occurrence.

History of algorithms performing this task has begun in late 40s, with works of Shannon's, Fano's and Huffman's [Shannon and Weaver 1949; Fano 1949]. The last of them is an author of to-day's most widespread method for generation of optimal length codes, called Huffman coding [Huffman 1952].

Although Huffman codes are optimal, they are assigned to individual symbols, consequently their codelengths are integers, often inaccurately approximate to the desired value of $-\lg P(a)$. To make the codelengths optimal for a given set of symbols, a coding method is needed, which would allow fractional codelengths. And there is such a method – arithmetic coding [Abramson 1963].

Independently of the solution of the optimal codelength problem, a question remains how to estimate each symbol's occurrence probability. The simplest method, counting every symbol's number of occurrences, seems optimal but that is the case only when each symbol appears in the source independently of the others. In practice, data coming from a majority of typical sources is highly correlated internally. It means redundancy of symbol position, value or precedence information. Three most popular ways of solving this problem are: extension of an alphabet (applied in substitutional data compression methods), statistical modeling of intersymbol dependences (predictive methods), and decorrelation through block transformations.

A substitutional method of the simplest sort is run-length encoding (RLE). It enriches the alphabet with supersymbols, allowing encoding multiple occurrences of individual symbols. This method is invaluable in situations where one symbol is locally dominating over all the others. It is still used to compress peculiar types of data (computer generated graphics

for instance), or as an additional processing step in more complex compression methods.

In 1977-1978 Ziv and Lempel presented much more advanced substitutional methods (LZ77, LZ78) [Ziv and Lempel 1977][Ziv and Lempel 1978]. Like in RLE, supersymbols are added to the alphabet to represent symbol sequences. Yet this time the substituted sequences are not built of the same symbols. Their work is to substitute phrases (strings of symbols) with unique identifiers. If only the identifier's length is shorter than the substituted phrase's length, the total amount of data is reduced, i.e. the compression is working.

Only phrases contained within a dictionary have identifiers, so only such phrases may be substituted. The dictionary is composed of phrases that have already been seen in the source. In LZ77 family methods, a limited number of recently processed source symbols makes the dictionary. In LZ78 methods, the dictionary is built by the coder simultaneously with coding.

At present substitutional methods are the ones used most often. They are used in such areas as computer graphics compression, hardware data compression in modems, and first of all in general purpose file archiving utilities, *Zip* to be the most popular of them.

Despite the substitutional methods are inferior to the remaining two types in compression effectiveness, they surpass the others in respect of compression speed and memory requirements [Swacha 1999].

Predictive methods utilize statistics gathered through analysis of processed input data to calculate probabilities of each symbol's occurrence as a next one in the source. The Prediction-by-Partial-Match (PPM) family methods predict forthcoming symbol basing on its context, i.e. the symbols seen right before it [Cleary and Witten 1984]. The more intersymbol correlation there is, the better PPM's effectiveness is. The number of preceding symbols taken into account is the prediction's order. In the original PPM the order is constant, set for the entire data source. In PPM* the order is variable according to the processed data [Cleary and Teahan 1993].

Only the symbols already seen in the current context may be predicted using it. To let the coder handle symbols which appear in the current context for the first time special escape codes are used. The escape codes are given the probability of occurrence of any symbol that has not been seen in the current context so far. It means that the current context's length must be decreased by one to make the prediction possible. If the forthcoming symbol has not been found in any context, it eventually comes to a virtual context of length -1 in which all the alphabet symbols are available. Every time the context is reduced, prediction's uncertainty decreases also as all the symbols

which could be predicted using longer context and were not, are known to be the wrong ones, and may be excluded from further prediction.

Most usual data sources tend to be more correlated locally than globally. Statistics need to be maintained in a way which takes into account how old they are. A simple method of accomplishing that is a scaling procedure – all gathered frequencies are divided by some constant in time intervals.

At present, the best method of the PPM family is C. Bloom's PPMZ. M. Taylor's *RK* program using PPMZ is on the top of various archive comparison tests [Gilchrist 2001]. PPMZ is based on PPM* method, with Local Order Estimation (LOE) and Secondary Escape Estimation (SEE) algorithms added to achieve superior compression efficiency [Bloom 1998]. LOE uses most probable symbol's probability as a measure for context's reliability, whereas SEE uses separate statistical model (acknowledging context length, escapes run length and previous symbol) for escape prediction.

Recently, D. Shkarin presented his PPMII (PPM with Information Inheritance) scheme in which statistics for longer contexts are inherited from shorter ones [Shkarin 2001]. This method proved to be fast, and effective also for binary data. It is one of the methods included in our tests.

The last of the three general compression methods, is based on reversible block transformations. Such transformations change data order in the source, making it possible to compress the output efficiently with simple statistical coding. The first and most important Burrows-Wheeler Transformation (BWT) starts with creation of a rotations matrix. Every row of this matrix is a copy of the original data block, but beginning from a different position, hence the rotations [Burrows and Wheeler 1994]. The rows are sorted in a lexicographic order, and the result of the transformation is the rightmost column of the sorted matrix. The data there is very repetitive.

The second step of transformation-based compression methods is decorrelation of transformed data. Move-To-Front (MTF) was the algorithm proposed originally by Burrows and Wheeler. It substitutes symbols with their indexes on a LIFO stack. Nowadays, the best results are obtained using Distance Coding (DC) algorithm [Gilchrist 2001]. DC outputs only distances between beginnings of each symbol's runs [Binder 2000].

The last step of transformation-based compression methods is order-0 statistical coding. Any known method may be used here: Huffman coding was proposed originally, popular *Bzip* program uses Huffman coding with multiple codetables and DC-based programs use arithmetic coding.

The progress in string sorting algorithms of the last few years [Bentley and Sedgewick 1997], made it possible for BWT-based programs to work with speed not much inferior to *Zip*, but with much superior efficiency for most data types.

All the three mentioned methods, although achieving compression in different ways, are based on the same ground – data repetitiveness. Hence the idea of combining compression methods of different sort has already been considered to be worth research. One of good examples of such a mix is C. Bloom's LZIP [Bloom 1996]. LZIP uses current context to find the most probable phrase in the dictionary. According to dictionary's size and context length, LZIP's efficiency was close to substitutional or even some PPM methods, and LZIP was faster. On the contrary, the method presented by Volf and Willems in 1998 is very slow, but it achieves very high compression ratios [Volf and Willems 1998]. Their main contribution was to allow switching between different algorithms (LZ77 and Context-Tree Weighting in the original paper) during compression.

A new approach to the subject of combining predictive and substitutional methods is presented below.

3. PREDICTIVE-SUBSTITUTIONAL DATA COMPRESSION

Predictive methods are, even though most efficient, not popular. Their two basic short-comings are unsatisfying compression speed and high memory requirements. Substitutional methods are fast and popular, but lacking the efficiency of others.

The new method proposed here aims at mixing elements of the two to achieve better results than LZ while keeping the time consumption low. Even though there are applications in which limiting the amount of data is the sole purpose, in case of data transmission the compression speed is of primary concern, as the delay caused by applying compression must not exceed the time gained by acceleration of the transmission.

The basic compression engine used in our method is PPM-based. Its main difference from the standard PPM concerns encoding of order escapes. Instead of encoding special escape symbols, context lengths will be stored along with the encoded data. It could seem to be codelength expensive, but in reality context lengths are highly predictable, especially if the maximum context length is low.

PPM-based compression runs fast and uses less memory only when its order is low (2-4). Low order makes it impossible to model long phrases properly, so it hurts compression efficiency. It hurts it badly if there is a lot of long phrases in the source. And that is the case with many files, for example text – see Table 1.

In order to limit the efficiency loss caused by the use of low prediction orders, substitutions will be employed. It will be possible to substitute a

phrase provided it is found in a dictionary indicated by one or two previously processed symbols. Such dictionary access is analogous to LZP, but in our method all dictionary items are examined at once, not one after another. The dictionaries' size may be set as an user's parameter allowing control between compression efficiency and time and memory usage - all these variables grow with dictionary size.

Table 1. Maximum context length for sample text file *dow.txt* symbols, 536,049 bytes long.

<i>Length</i>	<i>Quantity</i>	<i>Share %</i>	<i>Length</i>	<i>Quantity</i>	<i>Share %</i>
-1 and 0	2418	0%	11	12124	2%
1	12946	2%	12	8883	1%
2	40557	7%	13	6506	1%
3	73140	13%	14	4860	0%
4	85280	15%	15	3781	0%
5	76529	14%	16	3019	0%
6	62373	11%	17	2399	0%
7	47418	8%	18	1972	0%
8	34412	6%	19	1629	0%
9	24198	4%	20 and more	14608	3%
10	16997	3%			

A substitution though possible will not be done unless recording it (its position, dictionary item index, match length and optionally dictionary index) requires less memory than encoding substituted symbols normally. This way execution time will increase but a situation in which introducing substitutions makes output data longer is practically excluded.

Substitution information will not be encoded as a supersymbol (like *Zip* does it), but independently of the symbol data. Every symbol (or chunk of them) will be preceded by a tag containing information whether there is a substitution or not. As the number of substituted symbols is very low compared to those left alone, the tag should lengthen symbols' codelengths very slightly. The substitution tag's codelength will be long, but it is taken into account when the decision about doing the substitution or not is being made.

The ideas presented above were applied into a test application compressing data using the described method.

4. PRACTICAL APPLICATION OF THE METHOD AND AN ANALYSIS OF THE RESULTS

As the main issue addressed by the proposed method is saving the bandwidth, possible implementation points seem to be Internet servers, caching architectures and client applications. Moreover, incorporating it as an extension to transmission protocols would be advisable. Introducing a new method on such a large scale could be interesting, but extensively effort- and time-consuming and its acceptance seems questionable. However, the example of *Zip* shows that it is the general compression software which makes the compression methods popular. Therefore we have implemented our method in a stand-alone compression program first, thus acquiring a tool for testing the proposed method in practice.

A program for data compression using predictive-substitutional method should consist of standard functions like: loading up the buffer with source data, memory allocation for dictionaries and statistical model, data encoding (see below), saving data, freeing the memory and exit. The data encoding process should be going on within a loop containing following phases:

1. Find in dictionary pointed to by previous symbol(s) the longest phrase consistent with the phrase being currently processed.
2. If the found phrase's length is longer than minimum one (constant), current state of statistical coder is stored. PHRASE_END is set to "input position + phrase length". Phases number 1 and 2 are executed only when there is no active phrase, i.e. PHRASE_END is before the input position.
3. Find the maximum context length (update the contexts without the currently encoded symbol).
4. According to whether the current context is shorter, equal or longer than the previous one, respectively 2, 0 or 1 is encoded preceded by "no substitution" tag. Statistics for encoding these numbers are counted independently of the context's value, but subject to previous context's length.
5. If the current context is shorter, the difference from the previous one must be encoded, again independently from the context's value, but subject to previous context's length.
6. The current symbol is encoded using the maximum context found.
7. The maximum context found is updated.
8. Move input position one symbol forward.
9. If input position is PHRASE_END then output growth since the phrase begin is compared to a predicted total cost of recording the substitution.
10. If the cost is lower, the coder state is restored and substitution data is written ("make substitution" tag, dictionary index, phrase length).

According to this specification *PLZ* (Prediction + LZ) program has been created. Results of tests carried out with this program on a set of carefully selected files of different types, and discussion of the results show the method's efficiency.

For the purpose of making this comparison we chose the metric of compression ratio defined as:

$$CR = \frac{L_{in} - L_{out}}{L_{in}}$$

where: L_{in} - amount of data input, L_{out} - amount of data output.

Besides *PLZ* test program (maximum prediction context length set to 3, dictionary size set to 16 items for order 1, 8 items for order 2 contexts), following compression utilities were used in the experiment: based on substitutional methods - *Zip* of Info-Zip's (options: maximum compression), PPMII-powered *PPMd* (ver. I, standard options – order 4) and J. Seward's *Bzip2* (ver. 1.0) based on BWT. More information about these programs is available on *ACT* web page [Gilchrist 2001].

Reading the tables below it should be taken into account that *PLZ* results were confronted with programs of comparable speed. However, results of *RK* (ver. 1.04a) program were included in Table 2 on a special basis to demonstrate how close these fast programs are to the best.

Table 2 shows results of compression for several test files selected by the authors (see below). The reason for choosing our own corpus instead of one of existing ones (*Calgary Corpus* for example) was to include file types that are more consistent with actual network traffic [Velasco and Lucianez 1998].

Table 2. Comparison of compression ratios for the authors' set of test files and the Calgary Corpus.

Name	Size	PLZ	Zip	PPMd	Bzip2	RK
Act.html	195 304	94,50%	94,26%	92,53%	95,50%	96,18%
Angband.exe	696 832	52,72%	51,14%	56,22%	52,37%	63,94%
Dow.doc	769 024	69,95%	62,50%	69,90%	67,36%	74,62%
Dow.html	1 869 314	90,39%	85,56%	89,11%	89,52%	91,03%
Dow.txt	536 049	71,60%	59,53%	69,67%	67,62%	70,06%
Screensht.bmp	1 440 054	98,45%	98,49%	97,70%	99,17%	98,73%
Xmlgen.xml	2 000 209	78,06%	64,13%	76,91%	74,91%	79,50%
Total	7 506 786	81,83%	75,14%	81,19%	80,35%	83,88%
Average	1 072 398	79,38%	73,66%	78,86%	78,06%	82,01%
Calgary Crp	3 251 993	75,24%	67,37%	74,95%	72,51%	77,22%

Act.html file is a list of programs tested in *ACT* with some description in a tabular form. It has a highly repetitive content, dominated by long phrases. PPM order 4 turns to be not enough to model this properly – *PPMd* loses here even with *Zip*. The advantage of substitutions availability in *PLZ* appears obvious. The best result from *Bzip2* could be easily predicted as BWT knows no context bounds, though it must be noticed that this program worked on this file (as on *dow.html* and *screensht.bmp*) much slower than other programs (as slow as *RK!*). The result of compression of a single file should not be generalized to entire class, but one must remember that many Internet files have a form similar to *act.html*, especially those generated automatically by database interfaces.

File *angband.exe* is a *Windows* 32-bit executable. Our method's weak point appears here for the first time, that is performance fall on heterogenous binary data. Results presented in Table 3 show that this file could be compressed better with lower prediction order. Unfortunately, unlike *PPMd*, *PLZ* has no information inheritance implementation to deal with such situations. To *PLZ*'s advantage, *Zip* and *Bzip2* are not any better at all.

Files *dow.doc*, *dow.html* and *dow.txt* are one file saved in three file formats: *MS Word* with binary structural information, tagged HTML with structural information in XML, and plain text without structural information. It contains a list of jokes in Polish, so it is an example of text file in natural language. Prediction wins here with other methods, and *PLZ* is better than *PPMd*. Once again binary contents of *dow.doc* affect compression ratio, but *PLZ* manages to retain its superiority over *PPMd* this time.

Screensht.bmp is a *MS Windows* screenshot, some active applications on it. The repeating phrases here are many and very long. Pre-processing this file with simple RLE compression improved *PLZ*'s result to 98,78% (5 KB gained). Although it is uncommon to interchange computer graphics in BMP format, this file has been included as a representative of any binary files with very low entropy.

Xmlgen.xml is a XML file generated using *XmlGen* program [Schmidt et al. 2001]. *PLZ* seems to be very good at compressing XML. Increasing its maximum context length by one improved the result on this file to *RK*'s level.

Conclusions after an analysis of results from Table 2 are as follows: the presented method works fine on text files, including the most popular file types like HTML and XML. Its compression ratio for executable files is worse than *PPMd*'s, but still acceptable.

Table 3 shows *PLZ*'s results after reducing its maximum context length by one. Compression was significantly faster this way, but efficiency has suffered for most of the test files. The table contains also share of symbols in each file compressed by substitutions (the remaining compressed by

prediction). These values differ between files, and as expected quotas for context length 2 are higher than those for context length 3 – the better the prediction model, the less the need to employ substitutions.

Table 3. Results for PLZ with maximum context length reduced to 2 and the share of symbols in each file compressed using substitutions (LZ Share).

	PLZ ctx length 3		PLZ ctx length 2	
	C.Ratio	LZ Share%	C.Ratio	LZ Share%
act.html	94,50%	72,67%	94,61%	78,39%
angband.exe	52,72%	2,21%	54,15%	8,69%
dow.doc	69,95%	6,62%	68,56%	8,87%
dow.html	90,39%	60,52%	90,08%	61,67%
dow.txt	71,60%	2,41%	70,12%	4,95%
screensht.bmp	98,45%	52,96%	98,20%	56,76%
xmlgen.xml	78,06%	2,87%	74,05%	8,57%
Total	81,83%	71,06%	80,52%	67,36%
Average	79,38%	28,61%	78,54%	32,56%

5. CONCLUSIONS

It appears that the search for new compression methods is a very important research problem and a progress in this field conditions efficient usage of data transmission technologies in computer networks.

The new compression method which was described above, combining predictive and substitutional elements fulfils its duty, achieving high compression efficiency combined with short execution time. Compared to other methods with similar time and memory requirements it distinguishes itself with very good text compression results. Texts make a huge part of data interchanged using Internet, so the method could find a practical use there.

REFERENCES

- Abramson, N. 1963. *Information Theory and Coding*. McGraw-Hill, New York.
 Bentley, J. L., Sedgewick R. 1997. Fast Algorithms for Sorting and Searching Strings. *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 360-369.
 Binder, E. 2000. *Distance coder*. Usenet newsgroup: comp.compression.

- Bloom, C. 1996. *LZP: a new data compression algorithm*.
<http://www.cbloom.com/lzp.zip>.
- Bloom, C. 1998. *Solving the Problems of Context Modeling*.
<http://www.cbloom.com/ppmz.zip>.
- Burrows M., Wheeler D. J. 1994. A block-sorting data compression algorithm. *SRC Research Report 124*. Digital Equipment Corporation, Palo Alto, Ca.
- Cleary, J. G., and Teahan, W. J. 1993. Unbounded length contexts for PPM. *The Computer Journal*, 36 (5).
- Cleary, J. G., and Witten, I. H. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32 (4), 396-402.
- Fano, R. M. 1949. *Transmission of Information*. M. I. T. Press, Cambridge, Mass.
- Gilchrist, J. 2001. *Archive Comparison Test*, Jun. 24. <http://compression.ca/>.
- Huffman, D. A. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proc. IRE* 40,9 (Sept.), 1098-1101.
- Sayood, K. 2000. *Introduction to Data Compression, Second Edition*. Morgan Kaufmann Publishers, San Francisco, Ca.
- Schmidt A.R., Waas F., Kersten M.L., Florescu D., Manolescu I., Carey M.J., Busse R. 2001. *The XML Benchmark Project*. <http://monetdb.cwi.nl/xml/Benchmark/benchmark.html>
- Shannon, C. E., and Weaver, W. 1949. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Ill.
- Shkarin, D. 2001. Improving the efficiency of PPM algorithm. *Problems of Information Transmission*, 34(3), 44-54. In Russian.
- Swacha, J. 1999. Popularne standardy kompresji danych. *Pro Dialog* 9, 23-32. In Polish.
- Velasco J.R., and Velasco Lucianez L.A. 1998. Benefits of compression in HTTP applied to caching architectures. *Proceedings of the Third International WWW Caching Workshop*.
- Volf P.A.J., and Willems F.M.J. 1998. Switching between two universal source coding algorithms. *Proc. IEEE Data Compression Conf.*, 491-500.
- Ziv, J., and Lempel, A. 1977. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inform. Theory* 23, 3 (May), 337-343.
- Ziv, J., and Lempel, A. 1978. Compression of Individual Sequences via Variable-Rate Coding. *IEEE Trans. Inform. Theory* 24, 5 (Sept.), 530-536.