RESEARCH INTO THE EFFICIENCY OF LOSSLESS COMPRESSION METHODS WHEN USED TOGERHER

Aydın CARUS
aydinc@trakya.edu.tr
Computer Engineering Department
Trakya University, TURKEY

Altan MESUT
altanmesut@trakya.edu.tr
Computer Engineering Department
Trakya University, TURKEY

Abstract

In this paper, we showed how the efficiencies of most frequently used lossless compression methods are changed when they are used together. The results of this study would be used in evaluation of performance of newly discovered lossless data compression approaches. We used Calgary Corpus, Canterbury Corpus and Trakya Corpus, which is made up by us, for testing the lossless compression methods.

Keywords: Arithmetic Coding, Huffman Coding, LZ77, LZW, lossless compression.

INTRODUCTION

Nowadays, compression of data provides very important advantages for storing and transmitting data. Therefore, various data compression techniques have been developed. These techniques, generally, are classified into two groups: Lossless Methods and Lossy Methods respectively.

Lossy data compression reduces the size of the source data by permanently eliminating some redundant information. The data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly. Lossy data compression is generally used for image, video and sound, where a certain amount of information loss will not be detected by most users.

In lossless compression, every single bit of data that was originally in the file remains after the file is uncompressed. All of the information is completely restored. This is generally the technique of choice for text documents, spreadsheet files, executable programs and source code. Sometimes this technique is also used for image compression.

There are two types of lossless compression coding techniques; variable length coding, also known as statistical-based coding, and dictionary-based coding. In variable length coding, compression takes place based on using shorter codewords for symbols that occur more frequently. The most well known variable length coding techniques are; Huffman Coding, and Arithmetic Coding. Dictionary-based coding techniques replace input strings with an index to an entry in a dictionary. The most well known dictionary-based techniques are Lempel-Ziv Algorithms and their variants.

USING METHODS TOGETHER

Combining dictionary-based techniques like LZ77, LZ78 and LZW with variable length coding techniques like Huffman Coding and Arithmetic Coding can increase compression ratio. Therefore, the performance of lossless compression methods that are used together has become more important than the performance when they are used alone. Most of the compression packages use variable length coders after a dictionary-based technique. The methods that are used together in this comparison are described briefly in this section.

Huffman Coding: Huffman codes are optimal prefix codes generated from a set of probabilities by a particular algorithm, the Huffman Coding Algorithm. David Huffman developed this algorithm as a part of an assignment when he was a student in a class on information theory at MIT in 1950 [1]. The class was the first ever in the area of information theory and was taught by Robert Fano. The algorithm is now probably the most prevalently used component of compression algorithms, used as the back end of GZIP, JPEG and many other utilities.

Arithmetic Coding: Shannon in his original 1948 paper [8], mentioned something very similar in the proof of the coding theorem. Peter Elias, another member of Fano's first information theory class at MIT, came up with recursive implementation for this idea. In 1968, the idea of arithmetic coding is further developed as an example of variable length coding in an appendix of a book written by Jelinek [2]. Modern arithmetic coding owes its birth to the independent discoveries in 1976 of Pasco [4] and Rissanen [5] that the problem of finite precision could be resolved. Rissanen and Langdon produced the most well known practical arithmetic coding algorithm in 1979 [6].

LZ77: Abraham Lempel and Jakob Ziv created the first of what we now call the LZ family of substitutional compressors in 1977 [11]. The LZ77 algorithm is based on the idea of a sliding window. The algorithm only looks for matches in a window a fixed distance back from the current position. Some text compression and archiving programs, such as PKZip, Gzip, ZIP, Lharc (LHA), ARJ and a modem protocol V.42bis are all based on LZ77 Algorithm. Some of them use a variable length coder, such as Huffman, after LZ77 compression.

LZW: Lempel and Ziv created a new substitutional compression scheme in 1978 [12], which works by entering phrases into a dictionary and then, when a reoccurrence of that particular phrase is found, outputting the dictionary index instead of the phrase. Several algorithms are based on this principle, which are known as LZ78 family, differing mainly in the manner in which they manage the dictionary. The most well known modification of LZ78 Algorithm is Terry Welch's LZW Algorithm [9]. The LZW compression algorithm is more commonly used to compress binary data, such as bitmaps. UNIX compress, and the GIF and TIFF image compression formats are both based on LZW.

RSSDC: In digram coding, the dictionary consists of all letters of the source alphabet followed by as many pairs of letters, called digrams, as can be accommodated by the dictionary [7]. Recursive Semi-Static Digram Coding [3] is an algorithm based on digram coding which uses a multi-pass approach. Because of multi-pass mechanism, the compression is slow, but the decompression is always done in one-pass, and it is considerably fast.

COMPARISON OF COMBINED TECHNIQUES

The c codes that are used in this comparison are; for Huffman Coding "codhuff.c" by David Bourgin (1995), for Arithmetic Coding "ari.cpp" by Mark Nelson (1996), for LZ77 "lz77.c" by Rachad Alao (1997), for LZW "codlzw.c" by David Bourgin (1995), and for RSSDC "rssdc.c" by Altan Mesut (2004).

We used Calgary Corpus [10], Canterbury Corpus [10] and Trakya Corpus, for testing the lossless compression methods. The Calgary Corpus is containing 18 files that are 3.251.493 bytes in size while The Canterbury Corpus is containing 11 files that are 2.788.958 bytes in size. Trakya Corpus, which is made up by us, includes 10 files and the total size of Trakya Corpus is 3.412.121 bytes. It includes four word files (roman.doc, solvsamp.xls, yuksek.ppt, north.mdb), two ASCII-based text files written in Turkish (yonet.txt, hikaye.txt), two c-code files (tasm2msg.c, dos.h), an image file (taner.bmp) and an HTML file (yasa.htm).

In this comparison, firstly all of the algorithms are used individually to compress corpuses. After that, the compressed files that are compressed by LZ77, LZW and RSSDC algorithms, are made inputs for Huffman Coding and Arithmetic Coding algorithms and compressed again. The results are shown in Table 1.

	Calgary	Canterbury	Trakya	TOTAL
Uncompressed Size	3.251.493	2.788.958	3.412.121	**9.452.572**
LZW	1.581.117	989.520	1.911.587	**4.482.224**
LZW + ARITHMETIC	1.567.817	982.605	1.857.207	**4.407.629**
LZW + HUFFMAN	1.579.823	991.855	1.875.276	**4.446.954**
LZ77	1.524.946	1.157.435	1.756.548	**4.438.929**
LZ77 + ARITHMETIC	1.462.213	1.070.347	1.723.286	**4.255.846**
LZ77 + HUFFMAN	1.474.959	1.082.438	1.742.230	**4.299.627**
RSSDC	1.500.417	975.613	2.101.476	**4.577.506**
RSSDC + ARITHMETIC	1.467.466	930.911	2.025.630	**4.424.007**
RSSDC + HUFFMAN	1.479.899	941.947	2.048.752	**4.470.598**

Table 1. Compression efficiency of combined techniques

EVALUATION OF COMPARISON

When the total size of corpuses evaluated the following results are found;

1. The use of Huffman Coding and Arithmetic Coding after LZW are provided 0,8% and 1,7% more compression than the use of only LZW, respectively.
2. The use of Huffman Coding and Arithmetic Coding after LZ77 are provided 3,1% and 4,1% more compression than the use of only LZW, respectively.
3. The use of Huffman Coding and Arithmetic Coding after RSSDC are provided 2,3% and 3,4% more compression than the use of only LZW, respectively.

Consecutive usage of statistical-based Arithmetic Coding algorithm after dictionary-based LZ77 algorithm is given the best compression ratio. The reason is that, LZ77 is not using all of the ASCII characters generally. Therefore, another compression approach can make more compression after LZ77.

CONCLUSION

We have seen that using a variable length coding algorithm after a dictionary-based technique is providing approximately 4% more compression. But if they are used together in one algorithm the compression ratio will be higher. For example, the LZARI Algorithm (the algorithm of LARC compression program), which was developed by Haruhiko Okumura in 1988, uses LZSS (a variant of LZ77) followed by Adaptive Arithmetic Coding. Haruyasu Yoshizaki replaced LZARI's adaptive arithmetic coding with a version of adaptive Huffman coding to increase speed. Based on this algorithm, which he called LZHUF, he developed another archiver, LHarc.

Some type of files cannot compressed by variable length coding techniques after they compressed dictionary-based techniques. This situation is clearly seen in Appendix.

REFERENCES

[1] D. A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. Proceedings of IRE, 40(9):1098-1101, September 1952.

[2] F. Jelinek. Probabilistic Information Theory. New York, McGraw-Hill, 1968.

[3] A. Mesut, A. Carus. A New Approach to Dictionary-Based Lossless Compression. UNITECH'04 International Scientific Conference, Gabrovo, November 2004.

[4] R. Pasco. Source Coding Algorithms for Fast Data Compression. Ph.D. Thesis, Stanford University, 1976.

[5] J. J. Rissanen. Generalized Kraft Inequality and Arithmetic Coding. IBM Journal of Research and Development, 20:198-203, May 1976.

[6] J. J. Rissanen, G. G. Langdon. Arithmetic Coding. IBM Journal of Research and Development, 23(2):149-162, March 1979.

[7] K. Sayood. Introduction to Data Compression. San Francisco, California, Morgan Kaufmann, 1996.

[8] C. A. Shannon. The Bell System Technical Journal, 27:379–423, 623–656, July, October, 1948.

[9] T. A. Welch. A Technique for High-Performance Data Compression. IEEE Computer, 17(6):8-19, June 1984.

[10] I. H. Witten and T. Bell. The Calgary/Canterbury text compression corpus. Anonymous ftp from ftp.cpsc.ucalgary.ca: /pub/text.compression.corpus/

[11] J. Ziv, A. Lempel. A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory, IT-23(3):337-343, May 1977.

[12] J. Ziv, A. Lempel. Compression of Individual Sequences via Variable-Rate Coding. IEEE Transactions on Information Theory, IT-24(5):530-536, September 1978.

APPENDIX

DETAILED RESULTS OF COMPRESSING CALGARY CORPUS

File Name	File size (byte)	LZW	LZW + ARITH	LZW + HUFFMAN	LZ77	LZ77 + ARITH	LZ77+ HUFFMAN	RSSDC	RSSDC + ARITH	RSSDC + HUFFMAN
bib	111.261	60.307	59.953	60.533	52.532	51.012	51.570	50.469	49.610	50.141
book1	768.771	419.111	416.551	418.202	432.320	414.854	416.543	382.582	375.207	376.686
book2	610.856	325.274	323.598	324.948	289.331	281.485	282.706	321.220	313.889	315.417
geo	102.400	79.287	76.418	77.107	88.347	82.663	83.392	63.174	62.543	63.102
news	377.109	231.578	229.529	230.575	204.170	198.112	199.230	218.376	210.917	212.030
obj1	21.504	13.684	13.363	13.804	13.125	13.039	13.475	13.407	13.009	13.452
obj2	246.814	134.562	132.331	133.288	110.314	108.325	109.164	149.614	146.273	147.675
paper1	53.161	28.960	28.840	29.327	25.072	24.682	25.154	27.312	26.780	27.272
paper2	82.199	42.705	42.517	43.045	40.608	39.539	40.059	39.890	39.288	39.798
paper3	46.526	25.288	25.206	25.677	24.084	23.542	24.013	23.243	22.936	23.387
paper4	13.286	7.143	7.156	7.585	7.291	7.273	7.706	6.912	6.816	7.247
paper5	11.954	6.802	6.807	7.235	6.539	6.534	6.960	6.493	6.389	6.817
paper6	38.105	20.543	20.494	20.945	17.824	17.646	18.095	19.682	19.312	19.775
pic	513.216	65.842	65.330	65.930	122.266	103.225	104.751	65.304	63.756	64.382
progc	39.611	21.143	21.073	21.530	18.101	17.921	18.366	19.720	19.346	19.804
progl	71.646	30.499	30.489	30.954	24.033	23.808	24.270	29.512	29.059	29.549
progp	49.379	21.502	21.511	21.955	16.400	16.334	16.781	19.366	19.126	19.584
trans	93.695	46.887	46.651	47.183	32.589	32.219	32.724	44.141	43.210	43.781
TOTAL	**3.251.493**	**1.581.117**	**1.567.817**	**1.579.823**	**1.524.946**	**1.462.213**	**1.474.959**	**1.500.417**	**1.467.466**	**1.479.899**

DETAILED RESULTS OF COMPRESSING CANTERBURY CORPUS

File Name	File size (byte)	LZW	LZW + ARITH	LZW + HUFFMAN	LZ77	LZ77 + ARITH	LZ77+ HUFFMAN	RSSDC	RSSDC + ARITH	RSSDC + HUFFMAN
alice29.txt	148.481	75.953	75.587	76.212	73.783	71.566	72.172	68.820	67.685	68.278
asyoulik.txt	125.179	67.351	67.065	67.642	67.128	64.943	65.527	61.302	59.964	60.532
cp.html	24.603	12.798	12.773	13.207	11.314	11.196	11.634	10.834	10.534	10.972
fields.c	11.150	4.965	4.998	5.427	4.228	4.262	4.692	4.530	4.512	4.945
grammar.lsp	3.721	1.813	1.830	2.262	1.722	1.735	2.178	1.865	1.859	2.296
kennedy.xls	1.029.744	269.791	266.867	269.920	388.494	342.302	347.340	303.191	271.995	276.899
lcet10.txt	419.235	216.272	215.119	216.161	199.302	193.211	194.152	204.348	200.265	201.344
plrabn12.txt	471.162	252.359	250.839	252.007	267.887	256.872	258.062	231.758	227.138	228.193
ptt5	513.216	65.842	65.330	65.930	122.266	103.225	104.751	65.304	63.756	64.382
sum	38.240	20.036	19.833	20.286	18.947	18.655	19.115	21.302	20.856	21.326
xargs.1	4.227	2.340	2.364	2.801	2.364	2.380	2.815	2.359	2.347	2.780
TOTAL	**2.788.958**	**989.520**	**982.605**	**991.855**	**1.157.435**	**1.070.347**	**1.082.438**	**975.613**	**930.911**	**941.947**

DETAILED RESULTS OF COMPRESSING TRAKYA CORPUS

File Name	File size (byte)	LZW	LZW + ARITH	LZW + HUFFMAN	LZ77	LZ77 + ARITH	LZ77+ HUFFMAN	RSSDC	RSSDC + ARITH	RSSDC + HUFFMAN
dos.h	17.286	7.004	7.032	7.460	5.836	5.850	6.278	5.859	5.818	6.248
hikaye.txt	6.387	3.610	3.629	4.061	4.196	4.199	4.631	3.726	3.686	4.121
north.mdb	1.585.152	832.677	814.207	819.550	818.080	801.381	807.260	1.040.775	988.985	1.001.674
roman.doc	665.600	379.457	363.151	369.803	356.557	348.714	355.750	357.986	351.495	355.004
solvsamp.xls	125.952	48.913	48.402	48.950	46.688	45.979	46.581	56.874	55.234	55.940
taner.bmp	453.654	389.897	373.524	375.477	321.496	317.030	318.883	411.361	399.079	401.952
tasm2msg.c	17.650	7.330	7.358	7.784	6.394	6.414	6.842	6.482	6.408	6.840
yasa.htm	246.794	95.438	95.366	95.893	73.353	71.545	72.180	76.936	75.935	76.577
Yonet.txt	35.086	16.698	16.719	17.151	12.774	12.658	13.097	14.483	14.329	14.770
yuksek.ppt	258.560	130.563	127.819	129.147	111.174	109.516	110.728	126.994	124.661	125.626
TOTAL	**3.412.121**	**1.911.587**	**1.857.207**	**1.875.276**	**1.756.548**	**1.723.286**	**1.742.230**	**2.101.476**	**2.025.630**	**2.048.752**