Arithmetic Coding-An Enhanced Implementation Using Locality

Lakshmi Sasilal and V. K. Govindan
CSED, NIT Calicut, Kerala
lakshmy.sasilal@gmail.com, vkg@nitc.ac.in

***Abstract*—The amount of data that is being stored and transmitted is increasing day by day. This increasing rate affects the performance of every application which depends on storage devices and networks. It has become a necessity to represent more amount of information using the available units of resources. Hence data compression is playing an important role in the information and communication technologies. There are many advantages from using data compression, like gaining space on hard drives or lowering use of transmission bandwidth in the network. There are also many algorithms and tools that are used today in this field. This paper focuses on lossless data compression techniques and suggests a technique to improve the computational efficiency of arithmetic compression scheme employing locality. A comparative study with the basic algorithm demonstrates improved compression performance of the proposed algorithm.**

***Index Terms*— data compression, Huffman coding, arithmetic coding, locality, Burrows-Wheeler, LZ77, LZ78, LZW.**

I. Introduction

The beginnings of data compression began as early as 1838 with its use as more code. The method of compressing data involves the encoding of information using less number of bits than its original representation. Data compression has many important applications in the areas of data storage and transmission. It allows a user to store more information in the memory device available than otherwise possible. The more important use of data compression comes in the area of networks, where the size of data being transferred is a crucial factor affecting its quality of service. The primary goal in all these cases will be to represent more amount of information in less number of bits. Various methods have been put forwarded by the researchers, and the field of information theory has been promoted to a big research area.

Among the different techniques available to compress data, two categories can be clearly identified. They are lossless compression techniques and lossy compression techniques. Lossless compression is possible because most of the real world data possess statistical redundancy and it can be exploited to represent the data in a concise manner without any loss in the information. The application in which some loss of information is acceptable makes use of lossy compression techniques.

The efficiency of the compression largely depends on the length of code words used. [1] discusses arithmetic coding scheme and claims to be superior in most respects to the better-known Huffman method. The present work mainly focuses on the improvement of arithmetic coding.

In arithmetic coding [1], a message is represented by an interval of real numbers between 0 and 1. As the message becomes greater in length, the interval needed to represent it becomes smaller, and the number of bits needed to denote that interval increases. Successive symbols of the message reduce the size of the interval in accordance with the symbol probabilities generated by the model. The compression ratio of the method is better than many other lossless schemes.

The most important advantage of arithmetic coding is its flexibility: it can be used in conjunction with any model that can provide a sequence of event probabilities. Optimality is another important advantage of arithmetic coding. Arithmetic coding is optimal in theory and very nearly optimal in practice, in the sense of encoding using minimal average code length.

The main disadvantage of arithmetic coding is that it tends to be slow. The full precision form of arithmetic coding requires at least one multiplication per event and in some implementations up to two multiplications and two divisions per event. Another disadvantage of arithmetic coding is that it does not in general produce a prefix code. This precludes parallel coding with multiple processors. One final minor problem is that arithmetic codes have poor error resistance, especially when used with adaptive models. Even an error in a single bit in the encoded file causes the decoder output to be an error, making the remainder of the decoded file corrupted.

II. Related Works

Witten *et al.* [1] gives a good tutorial on arithmetic coding- how it works, its practical implementation and the efficiency of the arithmetic coding, which has become a *de facto* standard. Later in 1991, Alistair et al. [2] described a new implementation of arithmetic coding that incorporates several improvements over a widely used earlier version. These improvements include fewer multiplicative operations; greatly extended range of alphabet sizes and symbol probabilities, and the use of low-precision arithmetic. In the work proposed in this

Manuscript received January 13, 2013; revised March 15, 2013.

doi: 10.12720/lnit.1.3.136-139

paper, we make use of a modular structure that separates the modeling, coding and the probability estimation components of a compression system. To model an improved coder, we consider the requirements of a character and context based text compression algorithm. The encoding procedure of a symbol depends on the context of the previous symbol being encoded and thus makes use of "locality of reference" principle. The encoder assumes that the symbols of same context tend to appear together in the file being compressed.

One drawback of arithmetic coding is its slow speed, because arithmetic coding requires multiplications. Some research have been done to avoid multiplications and to improve the efficiency. Langdon and Rissanen [3] proposed a modified scheme for encoding a binary string by using of shift-and-add. Rissanen and Mohiuddin [4] proposed a multiplication-free algorithm for encoding general string. The method was further developed by Lei [5]. Howard and Vitter [6] described an efficient bit-level implementation that uses table lookup as a fast alterative to arithmetic operations.

A variant of arithmetic coding scheme which can be used for data compression in supercomputers is discussed in [7]. It utilizes the co relational properties of data, i.e., the property that consecutive characters tend to be of same type (alphabets, digits, blanks, successive zeros etc). The scheme is based on the observation that in many data files, data records exhibit strong locality behaviour of character reference. Initially the encoding start on the basis of a particular set, (i.e. alphabets or numbers) and the basis of calculation changes when the type of character changes. At that point, the end-of-message character has to be encoded and then the rest of the message is coded starting with the initial interval.

Jianjun Zhang and Xingfang Ni [8] present a new implementation of bit-level arithmetic coding using integer additions and shifts. The algorithm has less computational complexity and more flexibility, and thus is very suitable for hardware design.

Several preprocessing techniques are discussed in [9], [10], [11]. In [9], they propose RIDBE (Reinforced Intelligent Dictionary Based Encoding), a Dictionary-based reversible lossless text transformation algorithm. The basic idea of the secure compression they have proposed is to preprocess the text and transform it into some intermediate form which can be compressed with better efficiency and which exploits the natural redundancy of the language in making the transformation. In [10], a transformation is done on every word in the file, such that the transformed words give shorter length for most of the input words and also retain some context and redundancy.

The method suggested in [11] processes a block of text as a single unit. The key idea behind the method is to apply a reversible transformation to a block of text to form a new block that contains the same characters, but is easier to compress by simple compression algorithms. The transformation tends to group characters together so that the probability of finding a character close to another instance of the same character is increased substantially.

III. Arithmetic Coding–Basic Algorithm

This section deals with the working and implementation of the arithmetic coding based on that of Witten et al. [1]. The algorithm for encoding a file using this method works conceptually as follows:

1. The "current interval" [L, H) is initialized to [0, 1).
2. For each symbol that has to be encoded, we do the following procedure:
    a. Subdivide the current interval into subintervals, one for each possible alphabet symbol. The size of a symbol's subinterval is proportional to the estimated probability that the symbol will be the next symbol in the file according to the model of the input.
    b. We select the subinterval corresponding to the symbol that actually occurs next in the file, and make it the new current interval.
3. We output enough bits to distinguish the final current interval from all other possible final intervals.

IV. Proposed Algorithm

While encoding large files or using complex models, multiple contexts can be used. The symbols can be assigned probabilities depending on the context to which they belong to, such as words, non-words or numbers. Some symbols may have high frequency of occurrence in a particular context, while it may not be present in another context. [2] is discussing about the use of multiple context in word based encoding scheme. The same can be used for character based encoding, but it include additional overhead to encode the pair (start, end) for every context change to denote which is the next context. This additional overhead can be eliminated if there is a mechanism to predict the context of the current symbol being encoded or decoded, depending on the previous symbols which have been encoded/ decoded. This method makes use of locality of reference, i.e., if symbols of same context appear together they can be encoded efficiently with least number of context changes and thereby less overhead.

The proposed algorithm for encoding process with single character based context prediction is as follows:

1. The "current interval" [L,H) is initialized to [0,1).
2. Decide the number of contexts and the symbols that belong to each context.
3. Build the probability model for each context such that in a particular context the probability of those symbols which does not belong to that context is scaled to a lesser value.
4. Initialize the variables, *current_context* (to denote the context in which a symbol has to be encoded) and *previous_context* (to denote the actual context of the last symbol encoded) as alphabetic context.

5. For each symbol that has to be encoded, we do the following procedure:
   a. Subdivide the current interval into subintervals, one for each possible alphabet symbol. The size of a symbol's subinterval is proportional to the estimated probability that the symbol will be the next symbol in the file, with the probability model as the one corresponding to the current value of variable *current_context*.
   b. We select that subinterval corresponding to the symbol that actually occurs next in the file, and assigns it to the new current interval.
   c. Find the context to which the actual symbol belongs to. If that context is same as *previous_context* (which implies two adjacent symbols belong to same context), assign it to the *current_context* (for a context change). Else no change occurs in the variable *current_context.*
   d. Even though we do not change the context for a single misprediction, we keep track of this misprediction in *previous_context*. Change the value of *previous_context* to the actual context to which the symbol belongs to and this is being used while determining the context for encoding the next symbol.
6. We output enough bits to distinguish the final current interval from all other possible final intervals.

The algorithm tries to predict the context of the next symbol being encoded on the basis of the previous symbol that has appeared in the input file. If the characters of same context tend to appear close in the file, the file can be compressed efficiently using this algorithm. In a particular context, a particular symbol can have high frequency of occurrences, but the same symbol can be exhibiting poor frequency of appearance in another context. So we can reduce the number of bits used to represent a particular symbol by assigning them to a particular context which contains the symbols that tends to appear together in the input files and representing those symbols with a larger interval in those contexts and with smaller intervals in other contexts. For example, in most of the text files the symbol "." is often followed by a space, digits tends to appear continuously etc. This property of symbols appearing together at certain portions of the input is made use of in this algorithm.

Since the interval selection and context change is based on the prediction mechanism which depends on the *previous_context*, some miss predictions can happen. We can reduce the rate of mispredictions if we have the knowledge about the nature of the input files and handling the mispredictions accordingly. We can define different context change mechanisms for different contexts, i.e. change the context to alphabetic if single misprediction happens, or change the context to special characters only if two mispredictions happen and so on. The algorithm described above changes the context only if two adjacent symbols belong to the same context.

After encoding the whole symbols in the file, the bits representing the final interval (or a fraction in the final interval) can be used to reproduce the entire file contents unambiguously. The decoding process is exactly the reverse of the encoding process. As done in the encoding process, the initial interval is chosen as [0, 1]. Define the contexts and probability models as the same that has been used for encoding process.

1. Initialize the variables, *current_context* (to denote the context in which a symbol has to be encoded) and *previous_context* (to denote the actual context of the last symbol encoded) as alphabetic context.
2. Until the "end of character" symbol is decoded, we do the following procedure:
   a. Subdivide the current interval into subintervals, one for each possible alphabet symbol. The size of a symbol's subinterval is proportional to the estimated probability that the symbol will be the next symbol in the file, with the probability model as the one corresponding to the current value of variable *current_context.*
   b. We select the symbol that contains the encoded value within its range and print it.
   c. Find the actual context to which the symbol that has been decoded belongs to. If that context is same as *previous_context* (which implies two adjacent symbols belong to same context), assign it to the *current_context* (for a context change). Else no change occurs in the variable *current_context.*
   d. Even though we do not change the context for a single misprediction, we keep track of this misprediction in *previous_context*. Change the value of *previous_context* to the actual context to which the symbol belongs to and this is being used while determining the context for decoding the next symbol.

V. Experiment Results

The above algorithm was implemented with single misprediction causing a context change. Three different contexts were defined as alphabetic, numeric and special characters. The performance was evaluated using Calgary and Large corpus files. The compression ratio has been calculated using the formula

%Compression Ratio = (Original Size- Compressed Size)* 100/ Original Size (1)

Most of the files showed an improvement over the basic method. The basic method has an average compression ratio of 36.98% for Calgary Corpus and 50.2 % for Large Corpus, while the results of proposed method showed an average compression ratio 39.42% for Calgary corpus and 55.18% for Large Corpus. Table I shows the results with Calgary Corpus and Table II shows the results with Large Corpus.

A. Factors Affecting the Results

The context definition has significant effects on the compression ratio. If we know the nature of the contents of the files we are trying to compress, we could have a guess on the characters that tend to appear close in that

particular file and thus define the contexts accordingly. This can reduce the number of mispredictions while performing the compression and have considerable improvement in the compression ratio.

The strategy we use to decide whether we have to change a context or not is also important. Commonly used text files with paragraph formatting usually include single space between two words. So if we change the context by seeing a single space, for each occurrence of space two mispredictions will be occurring and it will be costly. Hence the context change strategy has to be decided carefully, after the study of nature of the files we are compressing.

TABLE I. RESULTS WITH CALGARY CORPUS

Name	Size(kb)	% Comp. Ratio(existing)	% Comp. Ratio(proposed)
bib	111.3	34.65	34.78
book1	768.8	47.19	43.37
boo2	610.9	39.9	41.68
geo	102.4	26.28	28.72
news	377.1	34.96	37.7
obj1	21.5	25.36	22.34
obj2	246.8	17.4	21.45
paper1	53.2	27.78	37.2
paper2	82.2	39.65	42.16
paper3	46.5	41.88	42.75
paper4	13.3	37.43	39.5
paper5	12	36.68	36.16
pape6	38.1	36.7	36.7
pic	513.2	84.46	84.72
progc	39.6	34.41	34.35
progl	71.6	35.8	41.42
porgp	49.4	36.61	48.75
trans	93.7	30.44	35.87

TABLE II. RESULTS WITH LARGE CORPUS

Name	Size(mb)	% Comp. Ratio(existing)	% Comp. Ratio(proposed)
bible.txt	4	43.16	47.9
E.coli	4.6	74.05	76.07
world192.txt	2.4	33.41	41.54

VI. CONCLUSION

A modified version of basic arithmetic compression scheme is proposed in this paper. The whole character set is divided into different contexts and probability models are built independently for each context. A prediction mechanism is also suggested to choose which probability model has to be used for compression. The algorithm has been implemented and the comparison of results with basic scheme demonstrates performance improvements of the proposed algorithm using the file of Calgary and Large Corpus.

REFERENCES

[1] I. H. Witten, R. M. Neal, and J. G. Cleary. "Arithmetic coding for data compression," *Communications of the ACM*, 1987, vol. 30, no. 6, pp. 520-540.

[2] A. Moffat, R. M. Neal and I. H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256–294, July 1998

[3] G. G. Langdon and J. J. Rissanen, "A simple general binary source code," *IEEE Trans. Inf. Theory*, vol. 28, no. 3, pp. 800-803, 1982.

[4] J. J. Rissanen and K. M. Mohiuddin, "A multiplication-free multialphabet arithmetic code," *IEEE Trans. Commun*, vol. 37, pp. 93-98, 1989.

[5] S. M. Lei. "Efficient multiplication-free arithmetic codes," *IEEE Trans. Commun*, vol. 43, pp. 2950-2958, 1995.

[6] P. G. Howard and J. S. Vitter, "Arithmetic coding for data compression," in *Proc. IEEE.* 1994, vol. 82, pp. 857-865.

[7] M. A. Bassiouni, N. Ranganathan, and A. Mukherjee, "A scheme for data compression in supercomputers," in *Proc. ACM/IEEE Conference on Supercomputing*, 14-18 Nov 1988, vol. 1, pp. 272-278.

[8] J. J. Zhang and X. F. Ni, **"**An improved bit-level arithmetic coding algorithm," *Journal of Information and Computing Science*, vol. 5, no. 3, pp. 193-198, 2010.

[9] S. Senthil, S. J Rexiline, and L. Robert, "RIDBE: A lossless, reversible text transformation scheme for better compression," *International Journal of Computer Applications*, vol. 51, no. 12, August 2012.

[10] F. S. Awan, N. Zhang, N. Motgi, R. T. Iqbal, and A. Mukherjee, "LIPT: A Reversible Lossless Text Transform to Improve Compression Performance," in *Proc. IEEE Data Compression Conference, IEEE Computer Society Press, Los Alamitos, California*, 2001, vol. 481.

[11] M. Burrows, D. J. Wheeler, M. Burrows, and D. J. Wheeler, "A block-sorting lossless data compression algorithm," *Tech. Rep.*, 1994.

Lakshmi Sasilal is currently doing final semester of MTech in Computer Science and Engineering in the National Institute of Technology, Calicut. She has received Bachelor's degree in Computer Science and Engineering from Government Engineering College, Thrissur (University of Calicut) in the year 2011. She was born in Thrissur, Kerala on 3rd May 1989. Her research areas include data processing and information security.

V K Govindan was born on 30th April, 1950 at Malappuram. He has received Bachelor's and Master's degrees in electrical engineering from the National Institute of technology Calicut in the year 1975 and 1978, respectively. He was awarded PhD in Character Recognition from the Indian Institute of Science, Bangalore, in 1989. His research areas include Image processing, pattern recognition, data compression, document imaging and operating systems. He has more than 100 research publications in international journals and conferences, and authored ten books. He has produced six PhDs and reviewed papers for many Journals and conferences. He has more than 34 years of teaching experience at UG and PG levels and he was the Professor and Head of the Department of Computer Science and Engineering, NIT Calicut during years 2000 to 2005. He is currently working as Professor in the Department of Computer Science and Engineering, and Dean Academic at National Institute of Technology Calicut, India.