Indian Journal of Science and Technology, Vol 8(24), DOI: 10.17485/ijst/2015/v8i24/80242, September 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A New Variable-Length Integer Code for Integer Representation and Its Application to Text Compression

J. Nelson Raja[1*], P. Jaganathan[2] and S. Domnic[3]

[1]Department of Computer Science, Bharathiyar University, Coimbatore - 641046, Tamil Nadu, India; raja_nelson@yahoo.com
[2]Department of Computer Applications, PSNA College of Engineering and Technology, Dindigul – 624622, Tamil Nnadu, India; jaganathodc@gmail.com
[3]Department of Computer Applications, National Institute of Technology, Tiruchirappalli – 620015, Tamil Nadu, India; domnic@nitt.edu

Abstract

Data Compression plays an important role in reducing data storage space in computer memory and in achieving minimum data transmission time in communication networks. There are two types of data compression: lossless and lossy. In lossless data compression, decompression reproduces data that is exactly match with the original data and in lossy data compression, the decompression reproduces data which is an approximation of the original data. Variable length integer codes such as Elias Gamma Code, Elias Delta Code, Golomb Code, have been used for data compression (i.e. integer compression, text compression, etc). In this paper, a new variable length integer code is proposed based on radix conversion and it is used with Burrows Wheeler Transform for text data compression. The performance of the proposed code is compared with Elias Gamma Code, Elias Delta Code and Golomb Code. For evaluation, Calgary corpus is used in the experiments, which contains both text file and binary files. Experimental results show that the Fibonacci code gives better compression rate on an average than all other coders and Elias Gamma Code gives better compression rate for text files. The other coders perform well for binary files compared to Elias Gamma Code.

Keywords: Burrows-Wheeler Compressione, Elias Delta Code, Elias Gamma Code, Golomb Code, Variable-Length Integer Code

1. Introduction

The main objective of data compression is to store the data with the minimum number of bits in storage devices and transmit the data in low band width communication channels. Data-compression methods can be divided into two types i.e. lossy and lossless. In lossless compression methods, data can be decompressed as exactly same as the source data. In lossy compression, decompressed data is not 100% identical to the source and has a certain loss of information. Lossless data compression techniques are used to compress text data like financial data, executable programs, text documents, source code, etc. Lossy data compression technique is used to compress images, video and audio. Various variable-length codes[1] have been used for data compression. In contrast with fixed length codes, statistical coding methods achieves compression by assigning short-length codes to more frequently occurring symbols and long-length codes to rarely occurring symbols of the source file which needs to be compressed. The statistical methods require the probabilities of the input symbols to generate variable-length codes. Huffman coding[2] and Shannon-Fano[3] methods are examples for statistical methods which use symbol tables while decoding the compressed data. The two passes approach of the Statistical method is very slow for storage systems[4],

*Author for correspondence

sensory systems[5] etc. There are other coding methods such as Elias Gamma Code (EGC)[6], Elias Delta Code (EDC)[6], Golomb Code (GC)[7] which do not require the probability values of the input data to produce variable-length codes and these methods are called as variable-length integer coding methods or variable-length integer codes. Since variable-length integer codes does not require symbol table and probability values, these codes are more preferable in those applications which require fast encoding and storage. In this paper, a new variable-length integer code is proposed based on radix conversion and it is used as the final stage coder in Burrows-Wheeler compressor[8,9] for text data compression. The compression performance of the proposed code is compared with the existing state-of-the art methods such as EGC, EDC and GC. The experiments are carried out on Calgary corpus[10].

Section II describes state-of-the art variable length integer codes. In Section III, the new code is presented. In section IV, the performance of proposed code with Burrows-Wheeler Transform (BWT) for text data compression is studied and evaluated on Calgary Corpus data set. Section V concludes this work.

2. Variable Length Integer Codes

Variable-length Integer Codes, which are prefix codes, are used for the compact representation of non-negative integers. Since these codes are easy to be constructed, they have been used for image compression, video compression and text compression. In this section, state-of-the art Variable-Length Integer Codes: such as GC, EGC, EDC are presented, which are used to represent non-negative integers.

2.1 Unary Code (UC)

UC[1] is a universal variable-length code and satisfies the prefix property. Unary code of integer n is defined as $(n-1)$ zeros or ones followed by a single one or zero. So, the length of the UC for an integer n is thus n bits. UCs for 10 integers from 1 to 10 are given in Table 1.

2.2 Elias Gamma Code (EGC)

EGC was proposed by Peter Elias[6] in the year of 1975. EGC of an integer n contains two parts: unary part $UC(L)$ and binary part ~ $B(n)$, where $UC(L)$ is the unary code for the length (L) of the binary representation of n and $\sim B(n)$ is the binary representation of n without its most significant bit. EGC is generated as UC (L) | ~ B(n). The bit length of EGC for an integer n is $2\lfloor \log_2 n \rfloor + 1$ bits. EGCs for 10 integers are given in Table 2.

2.3 Elias Delta Code (EDC)

EDC developed by Peter Elias[6]. EDC has two parts: Gamma Part and binary part ~ $B(n)$. The Gamma part is the Elias Gamma Code of the bit length (L) of $B(n)$ and the binary part is the binary representation of the integer n without its most significant bit. So, EDC is generated as EGC (L) | ~ $B(n)$. The bit length of EDC for an integer n is $1+\lfloor \log_2 n \rfloor + 2\lfloor \log_2 (1+\lfloor \log_2 n \rfloor) \rfloor$ bits. EDCs for 10 integers are given in Table 3.

2.4 Golomb Code (GC)

GC[7] was developed by Solomon W. Golomb. In GC, the given number $n>0$ is first divided by a divisor d. The

Table 1. UC for the integers 1 to 10

Integer (n)	UC (n)	Bit Length
1	0	1
2	10	2
3	110	3
4	1110	4
5	11110	5
6	111110	6
7	1111110	7
8	11111110	8
9	111111110	9
10	1111111110	10

Table 2. EGC for the integers 1 to 10

Integer (n)	B(n)	EGC: UC(L) |~B(n)
1	1	1
2	10	10 | 0
3	11	10 | 1
4	100	110 | 00
5	101	110 | 01
6	110	110 | 10
7	111	110 | 11
8	1000	1110 | 000
9	1001	1110 | 001
10	1010	1110 | 010

quotient part q and the remainder part r are then used to code the given number $n(>0)$. The quotient q and the remainder r are obtained using equation (1).

$$\left.\begin{aligned} q &= \frac{n-1}{d} \\ r &= n - qd - 1 \end{aligned}\right\} \quad (1)$$

The GC contains two parts. The first part of GC is the value of $q+1$ which is coded in unary and the second part is binary value of r. For example, when divisor d = 4, it produces four possible remainders, 0, 1, 2 and 4 are coded as 00, 01, 10 and 11 respectively, and are given in the Table 4. Table 4 shows the GC for divisors d = 3 and d = 4.

3. Proposed Variable-Length Integer Code

In this section, the proposed variable-length integer code is presented. It is proposed based on radix conversion to represent integers. Compact representation of integers is essential for reducing the storage space requirements and achieving fast retrieval of integers. The proposed code for an integer is constructed based on the conversion of its radix-2 representation to radix-r representation. The idea is that the number of digits required to represent a number in higher radix-r *is* equal or less than that of lower radix-r. Hence, the number of bits used to represent higher radix-r in the length part of the proposed code will be less. Based on this idea, the proposed code has two parts: length part and data part. Length part is the number of digits required to represent the integer to be represented in the selected (higher) radix-r. Data part of the proposed code is the radix-2 (i.e. binary) representation (i.e. bits) of the integer. Since higher radix representation requires minimum number digits to represent an integer than lower radix representation, length part of the integer needs less number of bits to store the length part of the proposed code. This idea is used to achieve compact representation.

The state-of-the art methods: Elias Gamma Code, Elias Delta Code also represents integers in two parts: length part and data part. Length part denotes the number of binary digits (bits) required for binary representation (data part) of the integer. But, the unary coding of length part leads to poor representation of mid-range and large integers by these methods. The difference between the proposed code and Elias Gamma Code is that the proposed code uses length part to denote the number of radix-r digits required to represent the integer where as Elias code uses length part to denote the number of binary (radix-2) digits required for the integer representation. In both proposed code and Elias codes, data part is the binary representation of the given integer.

Table 3. EDC for the integers 1 to 10

n	EDC	Bit Length
1	1	1
2	100 | 0	4
3	100 | 1	4
4	101 | 00	5
5	101 | 01	5
6	101 | 10	5
7	101 | 11	5
8	11000 | 000	8
9	11000 | 001	8
10	11000 | 010	8

Table 4. Remainder table for divisor d = 3 & 4

Remainders	Binary codes	
	d = 3	d = 4
0	0	00
1	10	01
2	11	10
3	-	11

3.1 Algorithm for Encoding

The new variable-length integer code for an integer n is generated using the following steps:

- Select any radix-r to represent the given integer n.
- Calculate the number of digits required to represent n in the radix-r using the equation (2).

$$k = \left\lfloor \log_r^n \right\rfloor + 1 \quad (2)$$

- Code k in unary (k-1 zeros followed by 1 or k-1 ones followed by 0).
- Generate binary representation (b) of the integer n in $k\left\lfloor \log_r^n \right\rfloor$ bits.
- Attach binary code without its most significant bit if r=2, otherwise attach binary code of the integer n to the code of k generated in step 3.

Repeat step 1-5 for all integers.

Example: let $n = 25$ and $r = 4$
$b(n) = 011001$
$k = 3$ ∴ Use equation(2)
Proposed code of 25 = unary (3) | $b(n)$.
= 001|011001.

The bit length (BL) of the proposed for an integer n is give by:

$$BL = \left(\lfloor \log_r n \rfloor + 1\right)\left(1 + \lceil \log_2 r \rceil\right) \quad (3)$$

3.2 Algorithm for Decoding

The compressed integer is decoded using the following steps.

- Read the binary bits until bit 1/0 (unary terminator) is encountered and count the bits read so far as k. Read $(l-1)$ bits if $r = 2$, otherwise, l bits further. The value of l is calculated using equation (4).

$$l = \lceil k \log_2 r \rceil \quad (4)$$

- Convert 'l' binary bits into decimal equivalent value. The decimal equivalent value of k binary digits is used to determine the number of binary digits required to represent the integer n.
- Repeat step1-2 for all integers.

Example:
Let Proposed code = 001|011001, with $r = 4$
Read 001 and decode.
$k = 3$,
Compute $l = 6$ using equation (4).
Read 6 bits further and decode to get n
$b = 011001$,
$n = 25$.

Designing variable length codes applicable to all the probability distribution of the symbols (integers) for best compression is very difficult. The proposed code represents an integer n in $\left(\lfloor \log_r n \rfloor + 1\right)\left(1 + \lceil \log_2 r \rceil\right)$ bits. So our code is the best when integers follow the probability distribution of $P_n = \frac{1}{2^{\left(1+\lfloor \log_r n \rfloor\right)\left(1+\lceil \log_2 r \rceil\right)}}$. The parameter r is the key value in determining the suitability of our code for the given distribution. Table 6 shows the proposed code for integers 1 to 10 for $r = 2$, 3 and 4

Table 5. GC for the integers 1 to 10

Integer n	Golomb Code	
	d = 3	d = 4
1	0 | 0	0 | 00
2	0 | 10	0 | 01
3	0 | 11	0 | 10
4	10 | 0	0 | 11
5	10 | 10	10 | 00
6	10 | 11	10 | 01
7	110 | 0	10 | 10
8	110 | 10	10 | 11
9	110 | 11	110 | 00
10	1110 | 0	110 | 01

Table 6. Proposed code for the integers 1 to 10

Proposed code			
n	r = 2	r = 3	r = 4
1	1	1 | 01	1 | 01
2	01 | 0	1 | 10	1 | 10
3	01 | 1	01 | 0011	1 |11
4	001| 00	01 | 0100	01 | 0100
5	001| 01	01 | 0101	01 | 0101
6	001| 10	01 |0110	01 | 0110
7	001|11	01 | 0111	01 |0111
8	0001| 000	01 | 1000	01 | 1000
9	0001|001	001 | 01001	01 | 1001
10	0001 | 010	001 | 01010	01 |1010

4. Experimental Results and Discussion

In many applications, EGC, EDC, GC have been used. These codes have been used for compressing indices in database[11], information retrieval system[12], and compressing environmental data in wireless sensor networks[13,14]. In this work, the proposed code is used for text compression with BWT compressor. BWT compressor has four phases as shown in Figure 1. In first phase, the BWT computes the permutation of the input. In the second phase, the Move-To-Front (MTF) coder encodes the output of BWT. In the third phase, MTF output will be encoded by RLE. In the fourth phase, RLE output will be encoded by the Variable Length Integer Codes (VLC).

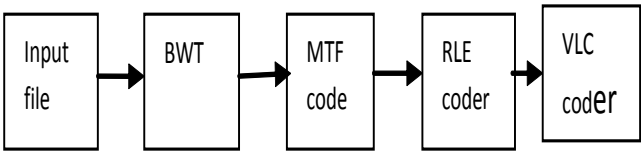

Figure 1. Stages of Burrows-Wheeler Compressor.

Table 7. Performance (bits per symbol) of variable length integer codes on Calgary corpus

Files	EDC	EGC	Proposed code			GC	
			r=2	r=4	r=8	d=4	d=8
bib	2.419	2.219	2.219	2.626	2.885	2.362	2.576
book1	3.435	3.077	3.077	3.666	4.048	3.130	3.600
book2	2.843	2.573	2.573	3.095	3.430	2.751	3.097
geo	5.888	6.152	6.156	5.769	5.535	11.120	7.350
news	3.217	2.997	2.997	3.479	3.762	3.268	3.412
obj1	4.536	4.628	4.631	4.602	4.609	7.726	5.529
paper1	3.005	2.752	2.752	3.284	3.623	2.985	3.266
paper2	3.035	2.745	2.745	3.284	3.629	2.881	3.246
paper3	3.350	3.042	3.042	3.610	3.972	3.160	3.530
paper5	3.734	3.491	3.491	4.059	4.401	3.092	3.231
paper6	3.074	2.828	2.828	3.391	3.729	3.111	3.385
pic	0.922	0.884	0.884	0.931	0.959	0.942	0.861
progc	2.960	2.763	2.763	3.247	3.530	3.092	3.231
progl	2.063	1.901	1.901	2.330	2.575	2.145	2.352
progp	2.007	1.873	1.873	2.309	2.554	2.222	2.379
trans	1.791	1.669	1.669	2.067	2.297	1.983	2.127
Average	3.017	2.850	2.850	3.234	3.471	3.498	3.323

The performances of various coders (EDC, EGC, GC, proposed code) with BWT compressor are evaluated on calgary corpus dataset[13] which contains both text files (bib, book1, book2, news, paper1, paper2, paper3, paper6, progc, progl, progp, trans and binary files (geo, obj2, pic). The performance is measured in terms of bit-rate which is calculated using Eq.5. Experimental results are given in Table 7. It is noted from Table 7 that proposed code with $r = 2$ and EGC achieve lowest bit- rate on an average than EDC, and GC. The proposed code ($r = 4, 8$) gives better compression performance than EDC, EGC and GC for binary files (geo and obj1). The proposed code gives better results than EDC, GC and it also gives competitive results when compared to the results of EGC.

$$\text{Bit - rate} = \frac{\text{Size of Compressed file}}{\text{Number of symbols in the input file}} \quad (5)$$

5. Conclusion

In this paper, a new variable-length integer codes are proposed to represent integers compactly. The proposed method represents an integer by converting its radix-2 representation to its radix-r representation. The proposed method is also used as the final stage encoder in Burrows-Wheeler compressor for text data compression. Experiments are carried out on Calgary corpus files. The compression performance of the proposed is compared with the state-of-the art methods (EGC, EDC, GC). The experimental results show that the proposed code gives better results than EDC, GC and it also gives competitive results when compared to the results of EGC.

6. References

1. Salomon D. Variable-length codes for data compression. London: Springer-Verlag; 2007. p. 69–100.
2. Huffman DA. A method for the construction of minimum-redundancy codes. Proceedings of the Institute of Radio Engineers. 1952; 40:1098–101.
3. Shannon CE. A mathematical theory of communication. Bell System Technical Journal. 1948; 27:379–423, 623–56.
4. Ramana YV, Eswaran C. A new algorithm for BTC bit plane coding, IEEE Transactions on Communications. 1995; 43(6):2010–11.
5. Leon-Salas WD. Low-complexity compression for sensory systems. IEEE Transactions on Circuits and Systems - II: Express Briefs. 2015 Apr.; 62(4).
6. Elias P. Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory. 1975 Mar; 21(2):194–203.
7. Golomb SW. Run-length encodings. IEEE Transactions on Information Theory. 1966; 12(3):399–401.
8. Fenwick PM. Burrows wheeler compression with variable length integer codes. Software-Practice and Experience. 2002; 32(13):1307–16.
9. Burrows M, Wheeler D. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation; 1994.

10. Witten IH, Bell T. The calgary/canterbury text compression corpus; 1990.
11. Williams HE, Zobel J. Indexing and retrieval for genomic databases. IEEE Transactions on Knowledge and Data Engineering. 2002; 14(1):63–78.
12. Trotman A. Compressing inverted files. Information Retrieval. 2003; 6(1):5–19.
13. Stephen Chang et.al. Energy and storage reduction in data intensive wireless sensor network applications. Technical Report 15-07; 2007.
14. Mihajlovic B. Compression and security platform for the testing of wireless sensor network nodes [Thesis]. Montreal, Canada: McGill University; 2008.