Impact Factor (SJIF): 4.542

International Journal of Advance Research in Engineering, Science & Technology

e-ISSN: 2393-9877, p-ISSN: 2394-2444

Volume 4, Issue 4, April-2017

Time based data compression: Clock algorithm

Dhuliya Vandan K.[#a], Prof. Vatsal Shah[#b]

[#a] B.E. Student, Information Technology Department, Birla Vishvakarma Mahavidhyalay, V.V. Nagar

[#b] Assistant Professor, Information Technology Department, Birla Vishvakarma Mahavidhyalay, V.V. Nagar

Abstract - Data compression is an area where data are represented in such a way that they are not been lost though require minimum redundancy, less space to store them in any media. This area has being found very efficient compression techniques with higher compression ratio, space savings and lesser compression rate for many years. There are many data compression techniques reduce data redundancy with good compression ratio. In this paper I worked on a technique which doesn't reduce redundancy but changes representation of data in different form such that it compresses data. This method gives a data compression ratio inversely proportional to decompression time complexity.

Keywords - *Lossless data compression, time based data compression, clock algorithm, image compression, text compression.*

I. Introduction

Data compression is study of how data are optimized with less bits for any storage media or machine and how to eliminate redundancy present in data or transform them into another format. In simple words, data compression is used to reduce data size. Consider a 200 MB file is compressed to 100 MB file, so 50% compression is done reduced 50% file size, requires less bits to store it.

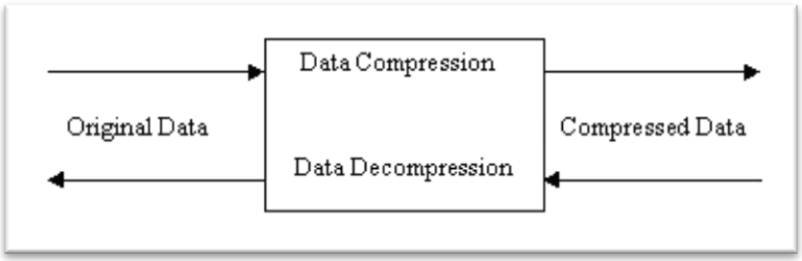

Figure A. Data compression block diagram [6]

Data compression techniques are categorized in two ways,

- Lossy data compression
- Lossless data compression

1.1. Lossy data compression

A lossy data compression method is one where the data retrieves after decompression may not be exactly same as the original data, but is "close enough" to be useful for specific purpose [3]. This method transfers data into such a form from that data cannot be recovered into original form and that compressed data are used instead

of original. For example, all real numbers of range -1 to 1 are represented as 0 then compressed file contains only 0s.

Source data sequence -> -0.3, 0.4, 0.64, 0.998, 1, -0.12

Decoded data sequence -> 0, 0, 0, 0, 0, 0

Decoder will never know about actual data as some information is lost.

This technique is preferable in such applications where speed is necessary than data lost. E.g., Audio/Video streaming. Video, audio, photo files are encoded with this technique. Lossy data compression methods are scalar and vector quantization.

1.2. Lossless data compression

Lossless data compression is a technique that allows the exact original data to be reconstructed from the compressed data [5]. Lossless compression is generally used for applications that cannot tolerate any difference between the original and reconstructed data [4].

This is analogous to an English man converses to a Gujarati man with the help of a translator man. That English man speaks “hello” then translator translates it to Gujarati word of “hello” and vice versa. Here translator knows mapping between English and Gujarati is analogous to an encoder. Data can be regained without any loss.

Examples of these techniques are Huffman coding, arithmetic coding, LZ77, LZ78, LZW, run-length encoding, Tunstall code and golomb code, etc [5].

1.3. Advantages of data compression

- Reduce storage requirement to store data
- Fast transmission over transmission media.
- Data encryption
- Avoid high bandwidth usage over internet

1.4. Disadvantages of data compression

- Requires more time to encode and decode data
- Data loss occurs in lossy data compression

II. Literature Survey

This section involves the Literature survey of various techniques available for Data compression and analysing their results and conclusions,

S.L. Bawa D.A.V, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010. This research paper provides survey of Huffman compression technique and comparisons of various lossless data compression methods with it. Comparisons provide necessary information when to use particular technique according to need of time complexity and space complexity.

Amarjit Kaur, Navdeep Singh Sethi, Harinderpal Singh, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 1, January 2015. This paper gives a review on various lossless data compression techniques. It provides advantages and disadvantages of them makes easy

way to compare them. It is shown that no algorithm does not provide promising result that can be used in practical applications to compress the data.

III. Proposed Algorithm

3.1. Abbreviations

OPV - operator value

TV - time value

TTV – timer time value

SEG – A segment

CSEG – A complete segment

ISEG – An incomplete segment

BSS – binary source string

BDS – binary destination string

SEGD – segment distribution

OPD – operator distribution

SRC_B – source bits

DEST_B – destination bits

OPV_B – operator value bits

TV_B – time value bits

CSEG_B – number of bits contained by a complete segment

ISEG_B – number of bits contained by an incomplete segment

SEGD_B – redundant bits for segment distribution

OPD_B – redundant bits for operator distribution

OP_T – number of possible operator values from OPV_B bits (2^{OPV_B})

TV_T – number of possible time values from TV_B bits (2^{TV_B})

ITV_T – number of time values present in ISEG

CSEG_N – number of complete segments

ISEG_N – number of incomplete segments

SEG_N – number of segments (complete + incomplete)

Following are some terms related to technique.

1. Operator value - It's an atomic (smallest) part of BSS nothing but a number.
2. Time value - It's also an atomic (smallest) part of BDS nothing but a number.
3. BSS – It's a collection of operator values.

4. BDS – It's a collection of time values.
5. Segment – It's a collection of operator values in a BSS.
6. Operator – Operator is a collection of time values in a BDS. All operators are distinguished from one another.

3.2. Data Formats

Source Data Format

Graphical format

Figure 1A. Source string

Figure 1B. Complete segment

Figure 1C. Incomplete segment

Logical/Actual format

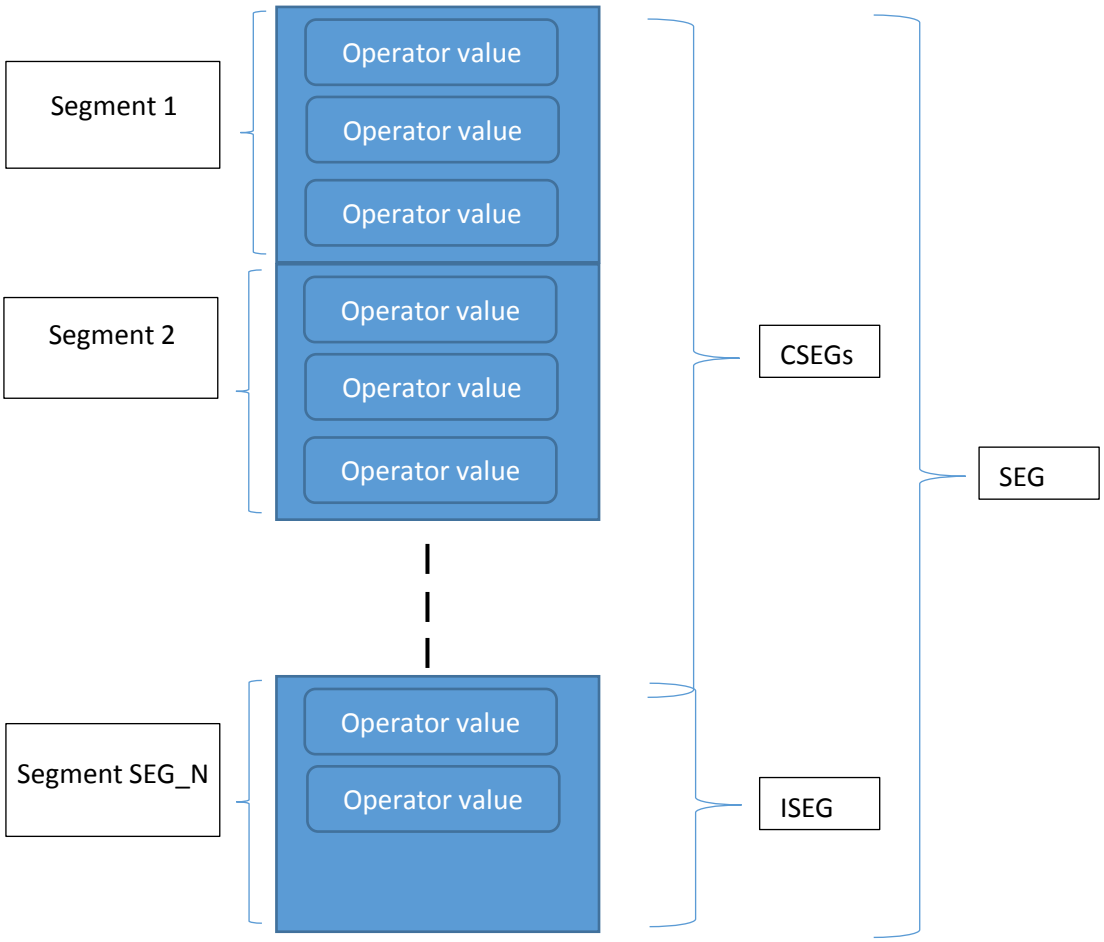

Figure 2. BSS format

Destination Data Format

Graphical format

Figure 3A. Destination string

Figure 3B. Complete segment

Figure 3C. Incomplete segment

Logical/Actual format

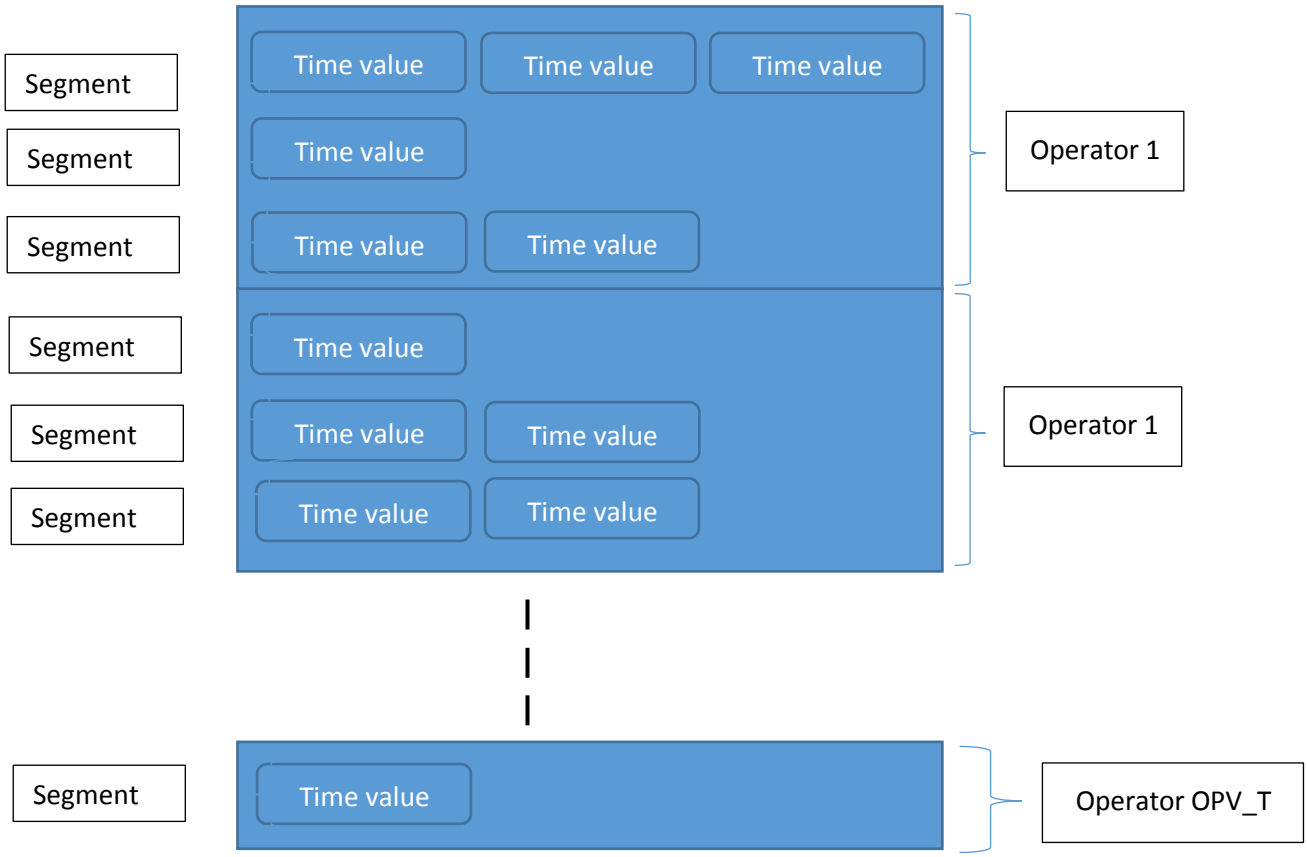

Figure 4. BDS format

Some formulas according to definitions

- SEG_N = ceil (SRC_B / (TV_T * OPV_B)) (equation – d1)
- CSEG_N = SRC_B / (TV_T * OPV_B) (as integer) (equation – d2)

- ICSEG_N = SRC_B % (TV_T * OPV_B) (as integer) (equation – d3)

Type of segments

There are two types of segments,

1. Complete segment – A segment which contains TV_T = 2^{TV_B} operator values or time values.
2. Incomplete segment – A segment which contains ITV_T operator values or time values.
   (0 < ITV_T < TV_T) (constrain – 1)

Incomplete segment is always only one in BSS and BDS according to above constrain.

ISEG_N = 1 (equation – 1)

And segment is summation of complete and incomplete segment.

SEG_N = CSEG_N + ISEG_N (equation – 2)

- A BSS contains only OPVs in particular segments and a BDS contains TVs in particular operators and segments.
- Distribution – Any two things are differentiated with something. Here something is some value/offset.
- Segment Distribution – segments aren't distributed neither in BSS nor in BDS, however decoder needs segment distribution for decoding in BDS so that actual data can be gained. It can be done by predictions and backtracking in decoding.
- Operator distribution – Operators are distributed in BDS by some redundant data.
- Compression principle – Algorithm simply converts OPVs into TVs. If OPV_B (bits) are less than TV_B (bits) then then compression may be possible (redundant data must be considered) as total number of OPVs are equal to total number of TVs.

- Figure 1A shows binary source string is divided into SEG_N number of segments.
- Figure 1B shows CSEG is further divided into TV_T number of operator values.
- Figure 1C shows ISEG into ITV_T number of operator values.

So, formulas are,

CSEG_B = OPV_B * TV_T (equation-3A)

ISEG_B = OPV_B * ITV_T (equation-3B)

Calculating bits,

SRC_B = CSEG_B * CSEG_N + ISEG_B * ISEG_N

SRC_B = CSEG_B * CSEG_N + ISEG_B (from equation – 1)

SRC_B = (OPV_B * TV_T * CSEG_N) + (OPV_B * ITV_T)

SRC_B = OPV_B (TV_T * CSEG_N + ITV_T) (equation-3C)

- Figure 3A shows binary destination string is divided into SEG_N number of segments.
- Figure 3B shows CSEG is further divided into TV_T number of time values.
- Figure 3C shows ISEG into ITV_T number of time values.

So, formulas are,

CSEG_B = TV_B * TV_T (equation-4A)

ISEG_B = TV_B * ITV_T (equation-4B)

Calculating bits,

DEST_B = CSEG_B * CSEG_N + ISEG_B * ISEG_N

DEST_B = CSEG_B * CSEG_N + ISEG_B (from equation – 1)

DEST_B = (TV_B * TV_T * CSEG_N) + (TV_B * ITV_T)

DEST_B = TV_B (TV_T * CSEG_N + ITV_T) (equation-4C)

3.3. Algorithm

Encoding

- Encoding changes OPVs into TVs.
- Encoding algorithm initializes a timer of certain bits from TTV=0 and increases TTV after reading and encoding an OPV into TV. When TTV reaches its maximum value TV_T, it is the end of one segment, then timer is again initialized to TTV=0 and process of reading and encoding continues until end of BSS.
- OPV is encoded with current TTV (timer time value) and are sent to an operator, corresponding to OPV.

Decoding

- Decoding changes TVs into OPVs.
- Decoding algorithm finds TTV starting from 0. Whenever it is found in BDS in particular operator it predicts that that operator was encoded, if prediction fails then it backtracks and do another prediction. And if prediction was true it increments TTV and continue above process until all segments are decoded. When it reaches to max value TV_T, it's the end of one segment and it again starts from 0. TTVs are incremented or decremented according to truth of prediction.
- TV is decoded with OPV where (in which operator) current TTV is found.

Algorithm

(Following pseudocodes are for ISEG=0.)

Encoding progresses by following pseudocode,

1. Set OPV_B and TV_B values.
2. Initialize TTV=0.
3. Read OPV_B bits.
4. Encode that OPV by sending TTV to corresponding.
5. If (TTV > TV_T) then TTV=0. (It's the starting of new segment).
6. Repeat 3 to 5 steps until BSS ends.
7. Stop.

Decoding progresses by following pseudocode,

1. Set OPV_B and TV_B values.
2. Initialize TTV=0.
3. Create NYT (not yet transmitted), AT (already transmitted) sets and a tree. (NYT and AT are sets contain TVs for predicted current segment for every operator)
4. Search OPVs which matches with TTV in NYT set.
5.

If (at least one value matches) then
(Go-Ahead)
Resolve (pick one of the operator values).
Remove that resolved match in NYT and add it to AT.
Spawn children of which have values as per searched OPVs.
Increment-TTV.

Else

(Backtrack)

Remove that previously resolved match in AT and add it to NYT.

Remove that child from tree.

If (only one child is there) than decrement-TTV.

6. Repeat steps 4 and 5 while (TTV <= TV_T)).
7. Set TTV=0. (It's the starting of new segment).
8. Update NYT set for next segment and empty AT set. (Updates sets according to next predicted segment.)
9. Repeat steps 4, 5, 6, 7 and 8 until BDS ends.
10. Stop.

Example

Source string -> ABACAAABCDC (ASCII chars – 8 bits)

So, SRC_B = 88(bits)

Suppose, OPV_B = 8 and TV_B = 1

So,

TV_T = 2^{TV_B}

TV_T = 2

Means one segment contains 2 values.

And,

SEG_N = ceil (SRC_B / (TV_T * OPV_B))

CSEG_N = SRC_B / (TV_T * OPV_B) (as integer)

and ICSEG_N = SRC_B % (TV_T * OPV_B) (as integer)

(from equation – d1,d2,d3)

Then, SEG_N = 6, CSEG_N = 5 and ISEG_N = 1

For destination CSEG_B = CSEG_N * TV_T * TV_B = 10 and ISEG_B = 1

(example has taken OPV_B = 8 though only 4 operators are shown a,b,c and d in diagram)

From above values,

Operator value	A	B	A	C	A	A	A	B	C	D	C
Time value	0	1	0	1	0	1	0	1	0	1	0
index	0	1	2	3	4	5	6	7	8	9	10
SEGs	CSEG1	CSEG1	CSEG2	CSEG2	CSEG3	CSEG3	CSEG4	CSEG4	CSEG5	CSEG5	ISEG

Encoding

Algorithms flows through following steps,

TTV initialized to 0,

1. TTV = 0, index = 0, operator value OPV = 'A', output = TTV = 0 (new segment)
2. TTV = 1, index = 1, operator value OPV = 'B', output = TTV = 1
3. TTV = 0, index = 3, operator value OPV = 'A', output = TTV = 0 (new segment)
4. TTV = 1, index = 4, operator value OPV = 'C', output = TTV = 1
5. TTV = 0, index = 5, operator value OPV = 'A', output = TTV = 0 (new segment)
6. TTV = 1, index = 6, operator value OPV = 'A', output = TTV = 1
7. TTV = 0, index = 7, operator value OPV = 'A', output = TTV = 0 (new segment)
8. TTV = 1, index = 8, operator value OPV = 'B', output = TTV = 1

9. TTV = 0, index = 9, operator value OPV = 'C', output = TTV = 0 (new segment)
10. TTV = 1, index = 10, operator value OPV = 'D', output = TTV = 1
11. TTV = 0, index = 11, operator value OPV = 'C', output = TTV = 0 (new segment ,last one)

Encoded bits without operator distribution -> 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0

	Operator value	A	B	A	C	A	A	A	B	C	D	C
	Time value	0	1	0	1	0	1	0	1	0	1	0

Encoded bits with operator and segment distribution->

Segments\Operators	**A**	**B**	**C**	**D**
CSEG 1	0	1		
CSEG 2	0		1	
CSEG 3	0, 1			
CSEG 4	0	1		
CSEG 5			0	1
ISEG 6			0	

Encoded bits with operator distribution->

Segments\Operators	A	B	C	D
SEGs	0, 0, 0, 1, 0,	1, 1	1, 0, 0	1

Decoding

Algorithms flows through following steps,

1. TTV=0,
   NYT: {{},{},{},{}}
   AT: {{},{},{},{}}
2. TTV=1,
   NYT: {{0},{1},{1},{1}}
   AT: {{},{},{},{}}
3. TTV=0,
   NYT: {{},{1},{1},{1}}
   AT: {{0},{},{},{}}

   ('A' predicted)
4. TTV=1,
   NYT: {{},{},{1},{1}}
   AT: {{0},{1},{},{}}

   ('B' predicted)
   (first segment completed)
5. TTV=0,
   NYT: {{0},{1},{1},{1}}
   AT: {{},{},{},{}}
6. TTV=1,
   NYT: {{},{1},{1},{1}}
   AT: {{0},{},{},{}}

   ('A' predicted)
7. TTV=0,
   NYT: {{},{},{1},{1}}
   AT: {{0},{1},{},{}}

   ('B' predicted)
   (second segment completed)
8. TTV=1,
   NYT: {{0, 1},{},{1},{1}}
   AT: {{},{},{},{}}

9. TTV=0,
   NYT: {{1},{},{1},{1}}
   AT: {{0},{},{},{}}

   ('A' predicted)
10. TTV=1,
    NYT: {{},{},{1},{1}}
    AT: {{0, 1},{},{},{}}

    ('A' predicted)
    (third segment completed)
11. TTV=0,
    NYT: {{0},{},{1},{1}}
    AT: {{},{},{},{}}
12. TTV=1,
    NYT: {{},{},{1},{1}}
    AT: {{0},{},{},{}}

    ('A' predicted)
13. TTV=0,
    NYT: {{},{},{},{1}}
    AT: {{0},{},{1},{}}

    ('C' predicted)
    (forth segment completed)
14. TTV=1,
    NYT: {{},{},{0},{1}}
    AT: {{},{},{},{}}

    ('C' predicted)
15. TTV=0,
    NYT: {{},{},{},{1}}
    AT: {{},{},{0},{}}
16. TTV=1,
    NYT: {{},{},{},{}}
    AT: {{},{},{0},{1}}

    ('D' predicted)
    (fifth segment completed)
    (CSEGs are completed)
    (next is ISEG)
17. TTV=0,
    NYT: {{},{},{0},{}}
    AT: {{},{},{},{}}
18. TTV=1,
    NYT: {{},{},{},{}}
    AT: {{},{},{0},{}}

    ('C' predicted)
    (ICSEG is completed)

Source string -> ABACAAABCDC

In above example backtracking is not happened. But at some point if algorithm does not find any next prediction, it is required to backtrack.

3.4. Complexity

Time Complexity

Encoding - $\partial(n)$ linear average case time complexity

Decoding - $\omega(b^{TV\_B})$* $\omega(2^{OPV\_B}) = \omega(s^{TV\_B} * 2^{OPV\_B})$ exponential worst case time complexity

(Where, b = average branching factor of tree structure)

Space Complexity

Compression Ratio

$$\mathbf{r} = \frac{BSS}{BDS}\left(\frac{\text{compressed}}{\text{uncompressed}}\right)$$

1. Ideal Compression Ratio,

$$r1 = \frac{DEST\_B}{SRC\_B}$$

$$r1 = \frac{\text{TV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}}{\text{OPV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}} \quad \text{(from equation – 3C, 4C)}$$

$$r1 = \frac{TV\_B}{OPV\_B} \quad \text{(equation 5A)}$$

2. Practical Compression Ratio,

$$r1 = \frac{DEST\_B + OPD\_B + SEGD\_B}{SRC\_B}$$

(SEGD redundancy comes from depth ambiguity, it's explained in next session.)

$$r2 = \frac{\text{TV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)} + OPD\_B + SEGD\_B}{\text{OPV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}}$$

$$r2 = \frac{TV\_B}{OPV\_B} + \frac{OPD\_B + SEGD\_B}{\text{OPV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}}$$

$$r2 = r1 + \frac{OPD\_B + SEGD\_B}{\text{OPV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}} \quad \text{(from equation – 5A)}$$

$$r2 = r1 + REDF$$

$$\text{Where, redundancy factor } REDF = \frac{OPD\_B + SEGD\_B}{\text{OPV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}}$$

Analysis

In equation 4,

1. If SEGD_B=0, OPD_B→ 0, REDF→ 0 and r2=r1.

   Means, redundancy decreases, ratio also decreases.

   (Practically, some redundancy must be there, so it's not possible.)

2. If SEGD=0, CSEG_N $\gg OPD\_B$, REDF→ 0 and r2=r1.

Means if BSS contains more number of complete segments (or segments) then ratio becomes lesser.

(Practically, it's possible as large data is needed to be compressed.)

Data Savings

$$ds = 1 - \frac{SRC}{DEST}$$

$$ds = 1 - \frac{OPV\_B}{TV\_B + \frac{OPD\_B + SEGD\_B}{\text{(TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}}}$$

$$ds = 1 - \frac{OPV\_B}{TV\_B + REDF}$$

3.5 Depth Ambiguity

Typically, decoding algorithm do segment distribution in BDS except in one case.

Depth – A segment number/index is called segment depth.

Clone – Any two segments have same number of TVs and same value of TVs in the same order are clones of each other.

Any two cloned segments with equal depth can't be predicted by decoding algorithm is called depth ambiguity. This occurs when segment distribution fails. This ambiguity is dependent on source data.

It requires some extra data to distinguish these segments. Extra bits are SEGD_B. By some mathematic calculation SEGD_B has a constant upper bound value.

Source Type/Data Independency

$$\text{Ratio r} = \frac{TV\_B}{OPV\_B} + \frac{OPD\_B + SEGD\_B}{\text{OPV\_B (TV\_T} * \text{CSEG\_N} + \text{ITV\_T)}}$$

Compression ratio contains two independent parameters OPV_B, TV_B, and five dependent parameters CSEG_N, TV_T and ITV_T, OPD_B and SEGD_B.

1. CSEG_N, TV_T and ITV_T are dependent on SRC_B and TV_B.
2. OPD_B is dependent on OPV_B and operator distribution format.
3. TV_B and OPV_B are independent can be taken anything by following TV_B < OPV_B constrain.
4. SEGD_B is one dependent on TV_B and source type.

However, SEGD_B depends on source type, it is bounded by a constant upper limit. So SEGD_B can be considered as a constant value.

Thus ratio becomes almost constant, source type independent. So for every type of data constant amount of compression is definite.

IV. Conclusion

Proposed time based lossless data compression technique is about a new type of technique than existing techniques which are statistical, arithmetic and dictionary based. This technique does not eliminate redundancy present in data but transforms data into another form which is compressed. Work on this shows data can be compressed without redundancy is one of the advantages of this technique. That means, it is a source independent algorithm.

Time based data compression technique compresses data very fast and decompresses them slowly. So, this method is applied on such data that are important, must not be destroyed and are not needed persistently like important documents, proofs, audios, videos, etc.

V. Future work

In future, this algorithm can be extended with composition of another techniques or algorithms to eliminate redundancy present in data which compresses data more.

VI. References

1. S.L. Bawa D.A.V, "Compression Using Huffman Coding", IJCSNS International Journal of Computer Science and Network Security, VOL-10 No-5, May 2010.
2. Amarjit Kaur, Navdeep Singh Sethi, Harinderpal Singh,"A Review on Data Compression Techniques" International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 1, January 2015.

3. Amandeep Singh Sidhu [M.Tech] and Er. Meenakshi Garg [M.Tech], "Research Paper on Text Data Compression, Algorithm using Hybrid Approach" International Journal of Computer Science and Mobile Computing (IJCSMC), Vol. 3, Issue. 12, December 2014, pg. 01 – 10.
4. Khalid Sayood, "Introduction to Data Compression, Second Edition".
5. Wikipedia - https://en.wikipedia.org/wiki/Category:Lossless_compression_algorithms.
6. Oracle - http://www.oracle.com/technetwork/articles/java/compress-1565076.html