International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 5, May - 2013

Enhancement Of Data Compression Using Incremental Encoding And Infix Substitution

Neeti Sangwan
Dept. of Computer Science, GGSIPU University, Delhi, India

Aanchal Nayyar
Dept. of Computer Science, GGSIPU University, Delhi, India

Meenakshi Kapoor
Dept. of Computer Science, GGSIPU University, Delhi, India

Abstract

In today's world scenario data has been multiplied to very large volumes and also the number of data storage applications has increased manifolds. At the same time, the communication networks are having in massive transfer of data through communication channels. In this paper a methodology has been proposed to reduce storage and thus reducing communication costs and also helps increase the capacity of channels and storage medium. The purpose indeed can be met by optimizing Incremental encoding algorithm which is a data compression technique to increase its performance even further. In this redundant/similar data between data entries are replaced by shorter codes. Also the previous research of optimizing incremental encoding algorithm with RLE algorithm has been compared with our proposed technique which will work more efficiently than the previous research done.

Keywords— Compression, Lossless compression, Lossy compression, Run length encoding, incremental encoding, Longest common substring, Pattern Matching

1. Introduction

When we have enormous amounts of data we would like to apply techniques to minimize the usage of space, such techniques are techniques of Data compression. It involves the development of a compact representation of information. Most representations of information contain large amounts of redundancy. Redundancy can be defined as the repetition of linguistic information inherent in the structure of a language, as singularity in the sentence it works. Therefore, one aspect of data compression is redundancy removal.

Data compression is an art of reducing the size of original data and data compression stores the same amount of data in few bits. This Paper specifies the study of efficient encoding and develops an algorithm to encode the data into as few bits as possible. The primary objective of data compression is to minimize the amount of data to be transmitted. The theme of this paper is to present and analyze a data compression technique.

At this stage we make use of the fact that in a database where a particular string of data that is common in entries of that database can be encoded as shorter codes. In the coding step we use shorter code words to represent letters that occur more frequently, thus lowering the average number of bits required to represent each letter. The design of a compression algorithm involves understanding the types of redundancy present in the data and then developing strategies for exploiting these redundancies to obtain a compact representation of the data. Redundancy or similarity can be present within a single input or between 2 inputs or both.

2. Compression Techniques

The Compression techniques can be basically classified into two types depending upon whether data is lost or not during the process of compression. These are as follows:

2.1. Lossless Compression

Lossless data compression technique use algorithms which usually exploit statistical redundancy to represent data more concisely without losing information. Lossless data compression is possible because most real-world data has statistical redundancy. Lossless compression techniques are used for transmitting textual data because we cannot afford losing any part of the text, because such loss can significantly change the meaning of the text, for example: Please do not attempt the first question of the paper, now let us say on compression the text becomes Please do attempt the first question of the paper, a loss of the “not” word changes the meaning of the original sentence completely, hence we use Lossless compression techniques for compressing textual data.

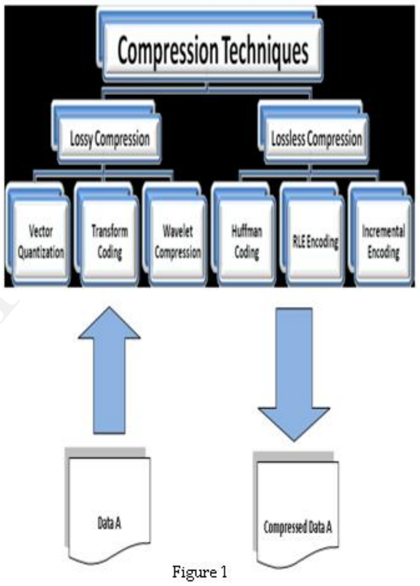

Figure 1

2.2. Lossy Compression

Lossy data compression is the converse of lossless data compression. In these schemes, some loss of information is acceptable. Dropping nonessential detail from the data source can save storage space. Lossy data compression schemes are informed by research on how people perceive bits of information. There is a corresponding trade-off between information lost and the size reduction. A number of popular compression formats exploit these perceptual differences, including those used in music files, images, and video.

3. Review of Incremental Encoding

Incremental encoding algorithm is a data compression technique employed where we replace common prefixes with shorter codes to save memory and avoid duplication. That is why it is also known as front compression, back compression, or front coding,This algorithm is particularly well-suited for compressing sorted data, e.g., a list of words from a dictionary. Following is the example of such words:

Table 1: Incremental Encoding

Input	**Common prefix**	**Compressed output**
psxa	No preceding word	0 psxa
psxorhvta	‘psx’	0 orhyta
psxorod	‘psxor’	5 od
cab	No common prefix	0 cab
cabbed	‘cab’	3 bed
cabbing	‘cabb’	4 ing
cabit	‘cab’	3 it
cabk	‘cab’	3 k
cabob	‘cab’	3 ob
50 bytes		**35 bytes**

The encoding used to store the common prefix length itself varies from application to application.

4. Related Work

Incremental encoding is a data compression technique where we save memory for storage of data by replacing longer words with shorter codes especially holds true for a dictionary encoder where the similarity between consecutive entries is exploited to save memory.

In the previous research paper incremental encoding algorithm has been enhanced by combining it with RLE algorithm. The methodology involved is 2-phased:

4.1. Phase 1

In the first phase of the algorithm, RLE algorithm is applied where repetitive patterns are replaced by a 2-byte code {no of times the particular letter appears repetitively, the repeated letter itself}, which means FFFFF can be encoded as 5F.Let us understand the concept by taking a simple example of 2 inputs, the first being FFFFGGGCCCDDE and the second being FFFFGGGCD. We can encode the first output as 4F3G3CDDE and the second output as 4F3GCD.

4.2. Phase 2

In the ^{2nd} phase of encoding data is encoded using incremental encoding wherein we replace common prefixes by the common prefix length that is of 1 byte. Let us understand the concept by taking a same example of 2 inputs in the 1st step, in the 1st phase of encoding we observe that the incremental output for the 2nd entry can be encoded as 8D where 8 is the common prefix length and the common prefix being FFFFGGGC. Finally both incremental encoding and RLE algorithm have been combined to increase the efficiency even further in the published research paper.

5. Proposed Technique

In our proposed technique an approach is used which comprises of 2 phases:-

5.1. Phase 1

The 1st phase involves the technique of encoding the data using incremental encoding algorithm where the common prefix between successive data entries is replaced by common prefix length in exactly the same way illustrated before in the 2nd phase of related work content.

5.2. Phase 2

The 2nd phase in the methodology proposed increases the efficiency by using concept of infix matching in any 2 successive data entries after the incremental encoding has been done.

5.3. Relative Analysis of Proposed Technique and Related Work

RLE algorithm works well where large amounts of repetitive character is available to us, whereas our methodology works more efficiently on record based datasets where repetitive pattern occurs. RLE algorithm does not cater to the needs of encoding real world data where similar patterns between successive data entries can be observed and exploited for saving more memory unlike our proposed methodology. The proposed Technique can be used to compress data of different organizations/firms/colleges.

Let us try to understand the comparison by illustrating the application of the proposed methodology on a dataset which is close to the kind of data we generally observe in the real scenario where in a firm we maintain the record of the computers in the firm where the nomenclature indicates the use of the computer and also its location:

CADC/AI/TPO/LG-11

CADC/ERP/TPO/LG-12

Taking up the above example, we can observe that applying the proposed methodology will lead to by far better results as compared to the RLE algorithm because in this case similar pattern is observed which can be treated as infix and can be encoded according to our proposed technique.

5.4. Implementation

The proposed technique is a 2-phased operation on data to be compressed. In the first phase we are using Incremental Encoding engine and in the second phase we are using Infix Substitution engine, which replaces the longest common substring between two consecutive words by its code which is of the form (Starting position length), after the incremental coding has been done. The procedure for encoding matched infix between successive data entries is as follows:-

After the 1st phase, to encode a matched infix, we use an algorithm by which we get the length of the common substring/infix which is the longest from the 2 successive entries, that are being compared, can be retrieved. The length is only taken into account when it is greater than 2 bytes because only in that situation we will save at least 1 byte of memory for the minimum case and much more in case in the length of the matched substring is considerably greater than 2 bytes. In the 2nd phase of encoding a pattern matching algorithm on the 2 data strings received as inputs has been applied by which the starting position of the matched substring/infix in the 1st word can be retrieved.

After the 2nd phase, using the above techniques the values for the starting position of the matched infix in the first word as well as its length have been already retrieved. Using these 2 values the final encoded form can be represented by replacing a matched infix by a combination of 2 bytes code that is [starting position, length of infix matched]. The form of encoding can be understood using the following example of let us say any 2 consecutive data entries in a database:

AAAEL127091310

AAAEL172709032

Using the proposed technique the final encoded output for the 2nd entry is represented as [6]7[7,4]032 where 6 represents the length of common prefix in the 2 entries,7 is written as it is since it is not similar to the 1st word, then 2709 which appears as infix in both the strings has been encoded by replacing 2709 as [7,4] where 8 indicates the starting position of the matched infix with respect to the 1st word and 4 corresponds to the length of the matched infix, also the technique takes into account that a particular matched infix will only be encoded if its length is greater than or equal to 3.

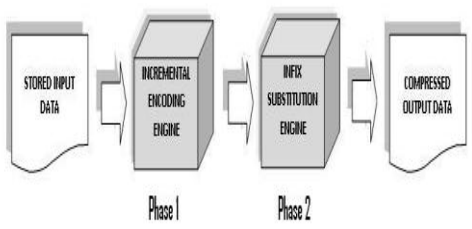

Figure 2 : Proposed Technique

5.4.1. Phase 1: Algorithm for Incremental Prefix Encoding

```
Step 1:   Initialize Queues (Q1, Q2, Q3 )
              and String s1
          T.count <- count1 <- count2 <- 0
          alphabet1 <- alphabet2 <- NIL

Step 2:   Q1 <- NULL;
```

```
            Q2 <- Q3 <- First Word

Step 3:     if ( Q1 == EMPTY )
             then  Dequeue and print all
            element of Q2 and save in s1
                Go to Step 5.
             else Dequeue one element of Q1
                  and Q2
             for i = 1, 2 if ( Qi(E) ==
                              Alphabet )
             then if ( counti == 0 )
                   then counti <- 1
                     alphabeti <- Qi(E)
            else counti <- Qi(E)
               Repeat Step 3.

Step 4:      if ( alphabet1 == alphabet2 )
              then T.count <- T.count +
                         count2
                    if (count1 == count2)
                      then Go to Step 3.
                    else print " T.count "
                      Deque all elements of
                        Q2 and save in s1.
              else Go to Step 5.

Step 5:       Q1 <- Q3; Q2 <- Q3 <- Next
                                 Word
               T.count <- count1 <- count2 <-
                    0
               alphabet1 <- alphabet2 <-
                   NULL

Step 6:        if (Q2 == EMPTY )
                then EXIT.
                else Go to Step 3.
```

5.4.2. Phase 2: Algorithm for matched Infix Encoding

```
Step 1:       Initialize String (s2,s3,s4,s5,s6)
               s2 < - First Word
               s3 < - Next Word [1…n]
               s4 <- s5 <- s6 <- NULL

Step 2:      function Infix(String1[1..m],
                           String2[1..n])
              L <- array(1..m, 1..n)
              z <- 0
             inf <- {}
             for i <- 1..m
             for j <- 1..n
             if s1[i] = s2[j]
             if i = 1 or j = 1
                L[i,j] <- 1
             else
                L[i,j] <- L[i-1,j-1] + 1
             if L[i,j] > z
                z <- L[i,j]
                inf <- {s1[i-z+1..i]}
             if L[i,j] = z
                inf <- inf ∪ {s1[i-z+1..i]}
             else L[i,j]=0;
              return inf

Step 3:   function PatMatch(String1,String2)
                j<- 1;
                while j <- n-m+1 do begin
                i<- 1;
                while (i<=m) and
                (String1[i]=String2[j]) do
                 begin
                i<- i+1;
                j<- j+1
                end;
                if i<=m then j<- j-i+2
                else return j-i+1;
                end.

Step 4:        s4 <- Infix (s1,s2)
                p <- s4.length
               if(p > 2)
                q <- PatMatch(s2,s4)
                 r <- PatMatch(s3,s4)
               else
                  Go to step 7.
```

```
Step 5:    s5 < - s3(T.count…r-1)
           s6 <- s3(p+r-1….n)

Step 6:    print"s5 '.' q ',' p'.' s6"

Step7:     s2 <- NULL
           s2 <- Next Word
           s3 <- Next to Next Word
           s4 <- s5 <- s6 <- NULL

Step 8:    if (s2== NULL)
               EXIT
           else
               Go to step 2.
```

5.5. Comparison Analysis

Let us take an example to understand the proposed technique and its advantages in terms of compression ratio with RLE encoding, incremental encoding and combination of these two in related work. "." is used to differentiate data from encode output. In this example we are taking 4 words as input and compressing the input words using RLE encoding, Incremental encoding, related work and our proposed technique. Below is the comparison table of the algorithm:

Table 2: Comparison Analysis Table

Input Word	**RLE Output**	**Incremental Output**	**RLE + Incremental Output**	**Proposed Technique Output**
AABCEAAACYBBC	AABCE4ACYBBC	AABCEAAACYBBC	AABCE4ACYBBC	AABCEAAAACYBBC
AABCBAAACYBBEA	AABCB3ACYBBEA	4BAAACYBBEA	4B3ACYBBEA	4B7,7EA
AABBAAACYBDCBE	AABB3ACYBDCBE	3BAAACYBDCBE	3B3ACYBDCBE	3.5,7DCBE
AABCAAACYBDCCD	AABB3ACYBDCCD	3CAAACYBDCCD	3C3ACYBDCCD	3C5,8CD
Takes 56 bytes	Takes 51 bytes	Takes 49 bytes	Takes 44 bytes	Takes 33 bytes
Compression %	**9%**	**12.5 %**	**21.5%**	**41%**

We can see that uncompressed data takes 56 bytes of memory and RLE, Incremental and combination of these two take 51 bytes, 49 bytes and 44 bytes of memory respectively while with the help of our proposed technique we are using only 33 bytes of memory to save data and thus the compression % is raised to 41 %. Hence it is the best among all the four types of technique to compress real world data.

5.6. Key Features

The proposed technique has the following key features:

- ❖ It is a lossless compression technique.
- ❖ Increases the efficiency of Incremental encoding algorithm considerably.
- ❖ Increase the transmission rate because of high compression.
- ❖ Increase the disk storage capacity.
- ❖ Increases the capacity of input/output channels and communication channels.
- ❖ It is a 2-phased enhanced encoding procedure.

6. Conclusion

The proposed technique gives an excellent result of data compression among Lossless compressions. As already illustrated in the table it is evident that the proposed methodology saves 41% of memory which is the highest among all the 4 results that are from incremental encoding, RLE algorithm, Incremental encoding, combination of these two and our proposed methodology respectively.

Thus, we can conclude that our proposed theory saves 20 % more than that in the related work, in the example we have illustrated. Also by replacing “infixes” we have broadened the horizon of its application beyond a dictionary to an organization based database which is primarily record-based that exhibits similar relationship among data in the database.

7. References

[1]Data Compression Khalid Sayood university of Nebraska, Lincoln
[2] Enhancement of Data Compression Using Incremental Encoding by Ajit Singh and Yogita Bhatnagar.
[3] http://en.wikibooks.org/wiki/Data_ Compression
[4] http://en.wikipedia.org/wiki/Data_ Compression
[5] http://www.ics.uci.edu/~dan/pubs/ Data Compression.html
[6] http://books.google.co.in/books?id=ChSOjgiY84YC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false
[7] http://datacompression.info/
[8]http://www.data-compression.com/index.shtml
[9] http://www.cs.cmu.edu/~guyb/ Realworld/compression.pdf
[10] http://www.webopedia.com/TERM/D/ data_compression.html
[11] Introduction To Data Compression by Khalid Sayood (http://books.google.co.in/ books?id=ChSOjgiY84YC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)
[12]http://www.amazon.com/Data-Compression -Book MarkNelson/dp/ 1558514341#reader_1558514341
[13] http://www.cs.cmu.edu/~guyb/real world/compress.html

[14]http://en.wikipedia.org/wiki/Run-length _encoding
[15] http://en.wikipedia.org/wiki/Run_ Length Limited
[16] http://www.fileformat.info/mirror/egff/ ch09_03.htm
[17] http://www.datacompression.info/ Algorithms/RLE/index.htm
[18] http://www.ijser.org/researchpaper% 5CEnhancement-of-Data-Compression-Using-Incremental-Encoding.pdf