International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 03, March-2015

Approaches to Optimize Bit Compression Algorithm

Prof. Swati Ringe,
Fr. C.R.C.E. Bandra,
University of Mumbai

Mr. Hardik Agrawal
Department of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering
University of Mumbai
Mumbai, India

Mr. Dylan Andrades
Department of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering
University of Mumbai
Mumbai, India

***Abstract*— Data compression involves reducing the statistical redundancies in data by using an alternative representation. This alternative representation usually involves substituting the original representation by using symbols to reduce the size required to represent the original information. In this article we propose certain techniques to optimize the performance of an already defined compression algorithm based on bit representation of data using inverted index notations, also known as Bit Compression Algorithm. There is found to be an improvement in compression percentage up to approximately 85.47% and 87.30% over the earlier 85.15% after two proposed improvements are applied to the technique.**

Keywords*— Bit-representation technique, data compression, inverted index

I. INTRODUCTION

Full text compression need an excellent data structure and an efficient algorithm to compress and decompress source data [1], [8]. Bit level principle has been used popularly for large scale data compression. Certain algorithms which provide efficient compression using bit level representation are [5], [6] and [7].

This research article involves study of a previously presented compression technique using bit representation [2]. It then proposes certain optimizations to improve the efficiency of the base approach. The logical implementation of the proposed optimizations and pseudo code for the same are provided. The statistics proving the improved performance are also presented.

The original technique proposed by Chovalit Khancome [2] proposes a new data structure and an algorithms for compression and decompression. Bits are used to store the positions of characters in the data. The given file is divided into documents and all characters in a documents have positions represented by bit form. It is found to give compressions ratios of 11.5% to 76.5% based on symbol set of 1 to 26 and 160,000 characters.

The first proposed improvement is called the 'Interpretation based reduction' where we reduce the number of bits required to represent the data based on bits required for previous positions. Next approach for optimizing called as 'reduction by subtraction' where we store the difference between the positions instead of the original positions to reduce the number of bits required for representation. The third approach called 'reduction by derivation' involves standardizing the number of positions mentioned in the compressed file and using derivation to remove the most redundant character. The three approaches are explained with an example in the further sections.

II. DESCRIPTION

First, we consider the compression algorithm proposed by Chovalit Khancome in [2].

Example 1. If the source data is
T = { aabaabcccaabbbaaabbbcccaaaababcccaaaabab } then T can be divided into four documents and given the occurring positions as follows.

TABLE I. DOCUMENTING THE DATA

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
D1	a	a	b	a	a	b	c	c	c	a
D2	a	b	b	b	a	a	a	b	b	b
D3	c	c	c	a	a	a	a	b	a	b
D4	c	c	c	a	a	a	a	b	a	b

As mentioned in [2], all positions of each character in each document are considered and then they are represented in the form of "character : <inverted lists form>" .This form is derived from the inverted index data-structure concept for files [3] and [4]. For the "inverted lists form" representation, it can be rewritten by the form of "<position : { documents }>".

Example 2. The inverted lists representation for example 1 is shown below:

TABLE II. INVERTED LIST REPRESENTATION

Characters	**Inverted Lists Form**
a	<1,{1,2}>, <2:{1}>, <4:{1,3,4}>, <5:{1,2,3,4}>, <6:{2,3,4}>, <7:{2,3,4}>, <9:{3,4}>, <10:{1}>
b	<2:{1}>, <3:{1,2}>, <4:{2}>, <6:{1}>, <8:{2,3,4}>, <9:{2}>, <10:{2,3,4}>
c	<1:{3,4}>, <2:{3,4}>, <3:{3,4}>, <7:{1}>, <8:{1}>, <9:{1}>

A. *Original Approach by Chovalit Khancome*

The bit-form representation is the form of "character : position : { D1 D2 … Dn }". For instance, if the character 'a' is considered (a:<1:{1,2}>), then it can be represented as a:<0001:110>. The bit-form of 0001 represents 1 in decimal and the bit-form of 110 represents the document numbers 1 and 2.

Considering the position, it depends on the length of document D. For instance, if the bit representation uses 4-bits (e.g. 0001), then it covers all positions from 1 to 15. For the {corresponding documents}, the document numbers need to be prepared by keeping the bit which equals all numbers of documents. For example, 0000 represents the document numbers from 1 to 4 respectively.

The positions of the character in the above example are represented in bit-form below:

TABLE III. INVERTED LIST REPRESENTATION

Inverted Lists Form		⇔	Bit-Form
a:	<1:{1,2}>	⇔	< 0001 : { 1100 } >
	<2:{1}>	⇔	< 0010 : { 1000 } >
	<4:{1,3,4}>	⇔	< 0100 : { 1011 } >
	<5:{1,2,3,4}>	⇔	< 0101 : { 1111 } >
	<6:{2,3,4}>	⇔	< 0110 : { 0111 } >
	<7:{2,3,4}>	⇔	< 0111 : { 0111 } >
	<9:{3,4}>	⇔	< 1001 : { 0011 } >
	<10:{1}>	⇔	< 1010 : { 1000 } >
b:	<2:{1}>	⇔	< 0011 : { 1100 } >
	<3:{1,2}>	⇔	< 0100 : { 0100 } >
	<4:{2}>	⇔	< 0110 : { 1000 } >
	<6:{1}>	⇔	< 1000 : { 0111 } >
	<8:{2,3,4}>	⇔	< 1001 : { 0100 } >
	<9:{2}>	⇔	< 1010 : { 0111 } >
	<10:{2,3,4}>	⇔	< 0001 : { 0011 } >
c:	<1:{3,4}>	⇔	< 0010 : { 0011 } >
	<2:{3,4}>	⇔	< 0011 : { 0011 } >
	<3:{3,4}>	⇔	< 0111 : { 1000 } >
	<7:{1}>	⇔	< 1000 : { 1000 } >
	<8:{1}>	⇔	< 1001 : { 1000 } >
	<9:{1}>	⇔	< 0001 : { 1100 } >

Theoretically, the bit representation in the above table can be shown as a = 64 bits (8 bytes), b = 56 bits (7 bytes) and c = 48 bits (6 bytes). The total size is 168 bits (21 bytes) from a source data of 40 bytes (in ASCII).

B. *Proposed Approach 1 – Interpretation Based Representation*

The bit-form representation is the form of "character : position : { D1 D2 … Dn }". However we sort the positions for a particular character in the descending order. We first start with the highest position number which is represented by maximum bits required for it. For e.g. consider character a, the highest position is 10 which is represented as 1010. Next we read this position and determine the amount of bits required for representing the next position. E.g. consider for character a if we have just used 0111 to represent position 7 now when we move to next position we already know that all positions below 7 can be represented by just 3 bits. Thus the from position 6 onwards we use only 3 bits and so on reduce the number of bits as the position decreases, whenever possible. Also when we come across the last position number represented using a particular number of bits we can decrease the bits next position onwards e.g. once we represent 2 using 10 we know that all positions below it can be represented by a single bit.

Using this approach, the bit representation obtained corresponding to given example will be as follows:

TABLE IV. PROPOSED APPROACH 1 (INTERPRETATION BASED REDUCTION OF BITS)

Inverted Lists Form		⇔	Bit-Form
a:	<10:{1}>	⇔	< 1010 : { 1000 } >
	<9:{3,4}>	⇔	< 1001 : { 0011 } >
	<7:{2,3,4}>	⇔	< 0111 : { 0111 } >
	<6:{2,3,4}>	⇔	< 110 : { 0111 } >
	<5:{1,2,3,4}>	⇔	< 101 : { 1111 } >
	<4:{1,3,4}>	⇔	< 100 : { 1011 } >
	<2:{1}>	⇔	< 10 : { 1000 } >
	<1:{1,2}>	⇔	< 1 : { 1100 } >
b:	<10:{2,3,4}>	⇔	< 1010 : { 0111 } >
	<9:{2}>	⇔	< 1001 : { 0100 } >
	<8:{2,3,4}>	⇔	< 1000 : { 0111 } >
	<6:{1}>	⇔	< 110 : { 1000 } >
	<4:{2}>	⇔	< 100 : { 0100 } >
	<3:{1,2}>	⇔	< 11 : { 1100 } >
	<2:{1}>	⇔	< 10 : { 1000 } >
c:	<9:{1}>	⇔	< 1001 : { 1000 } >
	<8:{1}>	⇔	< 1000 : { 1000 } >
	<7:{1}>	⇔	< 111 : { 1000 } >
	<3:{3,4}>	⇔	< 011 : { 0011 } >
	<2:{3,4}>	⇔	< 10 : { 0011 } >
	<1:{3,4}>	⇔	< 1 : { 0011 } >

Theoretically, the bit representation in the above table can be shown as a = 56 bits, b = 50 bits and c = 41 bits. The total size is 147 bits from a source data of 40 bytes (ASCII).

C. *Proposed Approach 2 – Reduction by Subtraction*

In this approach we reduce the number of bits used for representation of the positions by first sorting the positions in descending order. Next we use normal method to represent the highest position and all the rest positions are represented by bits corresponding to their difference with the immediate previous position. For e.g. if we consider character a from the

previous example position 10 will be represented as 1010 however its immediate next position i.e. 9 will be represented as 01 (since 10-9=1). This procedure is followed to represent all the following positions. Also notice in the table that for chars 'a' and 'b' we use 2 bits to represent differences while in 'c' we use 3 bits. The number of bits used to represent the difference is determined by the maximum difference present for a given character.

Using this approach for the given example the table obtained is as follows:

TABLE V. PROPOSED APPROACH 2 (REDUCTION BY SUBTRACTION)

Inverted Lists Form		⇔	Bit-Form
a:	<10:{1}>	⇔	< 1010 : { 1000 } >
	<9:{3,4}>	⇔	< 01 : { 0011 } > (10-9=1)
	<7:{2,3,4}>	⇔	< 10 : { 0111 } > (9-7=2)
	<6:{2,3,4}>	⇔	< 01 : { 0111 } > (7-6=1)
	<5:{1,2,3,4}>	⇔	< 01: { 1111 } > (6-5=1)
	<4:{1,3,4}>	⇔	< 01: { 1011 } > (5-4=1)
	<2:{1}>	⇔	< 10 : { 1000 } > (4-2=2)
	<1:{1,2}>	⇔	< 01 : { 1100 } > (2-1=1)
b:	<10:{2,3,4}>	⇔	< 1010 : { 0111 } >
	<9:{2}>	⇔	< 01 : { 0100 } > (10-9=1)
	<8:{2,3,4}>	⇔	< 01 : { 0111 } > (9-8=1)
	<6:{1}>	⇔	< 10 : { 1000 } > (8-6=2)
	<4:{2}>	⇔	< 10 : { 0100 } > (6-4=2)
	<3:{1,2}>	⇔	< 01 : { 1100 } > (4-3=1)
	<2:{1}>	⇔	< 01 : { 1000 } > (3-2=1)
c:	<9:{1}>	⇔	< 1001 : { 1000 } >
	<8:{1}>	⇔	< 001 : { 1000 } > (9-8=1)
	<7:{1}>	⇔	< 001 : { 1000 } > (8-7=1)
	<3:{3,4}>	⇔	< 100 : { 0011 } > (7-3=4)
	<2:{3,4}>	⇔	< 001: { 0011 } > (3-2=1)
	<1:{3,4}>	⇔	< 001: { 0011 } > (2-1=1)

Theoretically, the bit representation in the above table can be shown as a = 50 bits, b = 44 bits and c = 43 bits. The total size is 137 bits from a source data of 40 bytes (ASCII).

The following table reflects the optimization in form or bit-reduction obtained when the proposed techniques are applied:

TABLE VI. COMPARISON OF PROPOSED APPROACHES TO ORIGINAL APPROACH

Characters	Bit Compression Algorithm (bits)	Proposed Approach 1 (bits)	Proposed Approach 2 (bits)
a	64	62	50
b	56	52	44
c	48	43	43
Total	**168**	**157**	**137**

III. STATISTICS

The experiments were performed on Lenovo G505S notebook with AMD A10 Quad-core 2.5 GHz processor and 8 GB DDR3 RAM, running Windows 8.1 Pro (64-bit). The programs were written in Java in JDK 1.8 Build 20 and implemented using NetBeans IDE 8.0.1.

The test files used for conducting the test is 160,000 bytes and uses varied character sets ranging from 1 to 26. The number of documents for the original and proposed approaches are set to optimum value of 64. The results obtained are represented by the following table:

TABLE VII. STATISTICAL ANALYSIS ON ORIGINAL AND PROPOSED APPROACHES

No. of Characters	Bit Compression Algorithm (%)	Proposed Approach 1 (%)	Proposed Approach 2 (%)
1	85.156	85.474	87.303
2	70.305	70.948	74.606
3	55.455	56.421	61.909
4	66.195	66.935	70.108
5	64.199	64.985	67.986
6	63.273	64.088	67.161
7	61.693	62.545	65.686
8	60.647	61.527	64.632
10	59.743	60.650	63.766
12	59.107	60.039	62.933
14	59.004	59.949	62.811
16	58.131	59.106	61.892
20	57.540	58.550	61.356
26	57.056	58.107	60.601

The above table proves that both the proposed approaches are better than the original approach, hence improving the compression technique originally developed. Proposed approach 2 is even better than proposed approach 1 in terms of compression.

The optimum number of documents may vary according to data. Hence, these proposed approaches can be further optimized based on the number of documents.

IV. CONCLUSION

Based on the statistics we can see that the two proposed approaches provide means to optimize the basic approach. The two approaches provide an average compression ratio of 63.52% and 66.62% as compared to the 62.67% average ratio provided by the basic approach. These statistics prove the effectiveness of the proposed optimization techniques to improve the performance of the proposed approach. Since the proposed approaches provide better efficiency at almost no extra timing cost, they can be used along with the basic approach while developing a compression system.

ACKNOWLEDGMENT

Authors Hardik Agrawal and Dylan Andrades express gratitude to Professor Swati Ringe for encouragement to take up this research topic and for continued guidance throughout the course of research. We would also like to thank Fr. Conceicao Rodrigues College of Engineering for providing the necessary resources. Many thanks to Mr. George Cherian for brainstorming ideas with us during lectures. Lastly, we would like to thank our families for consistent emotional support.

REFERENCES

[1] M. Crochemore & W. Rytter, "Text Algorithms," (2010) Available: http://monge.univ-mlv.fr/

[2] “Text Compression Algorithm Using Bits for Character Representation”, International Journal of Advanced Computer Science, Vol. 1, Chouvalit Khancome.

[3] C. Monz & M.D. Rijke, "Inverted Index Construction," (2006) Available: http://staff.science.uva.nl/~christof/courses/ir/transparencies/clean-w-05.pdf.

[4] O.R. Zaïane, "CMPUT 391: Inverted Index for Information Retrieval," (2001) University of Alberta. Available: http://www.cs.ualberta.ca/~zaiane/courses /cmput39-03.

[5] H. Al-Bahadili & S.M. Hussain, "An adaptive character wordlength algorithm for data compression," (2008) Computers & Mathematics with Applications, vol. 55, no. 6, pp. 1250-1256.

[6] S. Nofal, "Bit-level text compression" (2007) In Proceedings of the International Conference on Digital Communications and Computer Applications, Jordan, 1, pp. 486-488.

[7] A. Rababáa, "An Adaptive Bit-Level Text Compression Scheme Based on the HCDC Algorithm," (2008) M.Sc., dissertation, Amman Arab University for Graduate Studies, Jordan.

[8] Khalid Sayood, “Introduction to Data Compression” (The Morgan Kaufmann Series in Multimedia Information and Systems)