DYNAMIC WORD BASED DATA COMPRESSION ALGORITHM FOR TEXT USING TAGGED SUB OPTIMAL CODE (TSC)

Nitin Arora
Department of Computer Science & Engineering
Women Institute of Technology, Dehradun

Esha Arora
Department of Computer Science & Engineering
Doon Institute of Engineering & Technology, Rishikesh

Abstract—
This paper presents a compression algorithm that can be used to compress the data as and when they are written rather than after they are stored. In this scheme the data is stored in compressed form rather than the data being compressed later. It is an efficient data compression technique comprising of Dynamic Word Based Tagged Sub – optimal Code (TSC) technique. This technique doesn’t use a single code for the text that it encounters. Instead it gives variable codes to the text as they come according to their frequencies. The major advantages of this algorithm are the speed of compression and the elimination of space complexity. This algorithm is suitable for real time applications for small digital devices such as Personal Digital Assistants (PDA) which have very limited capacity. Further, the compressed text can be decompressed in a simple and efficient manner. The paper also presents the design of the required data structures for our algorithm.

Keywords: Data Compression; Data Decompression; Dynamic word based Tagged Sub-optimal Code.

I. INTRODUCTION

Text is a major data form in computer and communication applications. Many compression algorithms developed so far have include text as a major experiment target[1]. Data compression is the process of encoding information using fewer bits (or other information-bearing units). Compression is useful because it helps reduce the consumption of expensive resources, such as hard disk space and transmission bandwidth. On the downside, compressed data must be decompressed to be used, and this extra processing may be detrimental to some applications. Many compression methods have been proposed till date which reduces the size of the data by various degrees [2], [3], [4].

1.1. Categories Of Data Compression Techniques (According To Method)

There are two main categories of data compression techniques: Those imporing statistical models, and those that require the use of a dictionary. Both type of techniques are widely used but dictionary based compression scheme tend to be used more for archiving applications (sometimes in conjunction with other methods), while real time situations typically require statistical compression scheme. This is because dictionary based algorithm tend to have slow compression speeds and fast decompression speeds while statistical compression tend to be equally fast during the compression and decompression.

1.1.1. Statistical Methods

The first systems used statistical methods in a variety of ways. The principle is that some characters appear more frequently than others. If a table is set up to grade the characters in order of frequency of use, then we can allocate minimum-bit codes to the characters with a high probability rate (which means the most frequently used ones). The lower the probability rate of a character, the more bits can be used to represent it.

In statistically-based systems [8] each collection of data is analyzed and a unique probability table created. Now, that does not mean the system is guessing; the table allocates variable-bit codes to characters according to the statistical analysis. The probability table has to be attached to the beginning of the compressed file, otherwise the receiver will not be able to decode it. The next step is to encode the data, which also requires that another collection of information has to be attached to the file.

1.1.2. Huffman Coding

This is probably the most widely used Statistical compression technique. It is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of
occurrence for each possible value of the source symbol.
The time complexity of an Adaptive implementation of Huffman encoding is linear: \( N [n + \log (2n + 1)] + Sn \) where \( N \) is the total number of input symbols, \( n \) is the current number of unique symbols and \( S \) is the time required, if necessary to rebalance the tree.

1.1.3. Arithmetic Coding

The Huffman system had one serious problem. Statistical results usually don't work out in whole numbers, and that makes for less efficient compression. Arithmetic coding takes advantage of the floating point ability of modern processors and represents a quantum step in compression efficiency.

The most basic statistical model would have a linear time complexity of \( N [\log (n) + a] + Sn \) where \( N \) is the total number of input symbols, \( n \) is the current number of unique symbols and \( a \) is the arithmetic to be performed and \( S \) is the time required if necessary to maintain internal data structures.

1.1.4. Dictionary Methods

The concept of dictionary compression is something we encounter daily. The ubiquitous acronym is a token for something much longer - for example, ASCII - and IBM has long since abandoned its original name for the initial letters.

Most popular compression utilities are based on the LZ77 [9] algorithm, which uses a moving window technique. Imagine a page of text covered by a mask, and the mask has a slot that can be moved along each line in turn to expose a few words at a time. In other words, a window of fixed length allows the reader (in this case, the compression program) to view some of the data. The program looks for repeating groups (usually called phrases), substitutes tokens for them, writes the result into a dictionary, and encodes the data from the window. Unlike Huffman, which makes two passes, the window-dictionary method does everything in a single pass.

1.1.5. Lempel-Ziv

LZ77 [9] and LZ78 [10] are the names for the two lossless data compression algorithms published in papers by Abraham Lempel and Jacob Ziv in 1977 and 1978. They are also known as LZ1 and LZ2 respectively. These two algorithms form the basis for most of the LZ variations including LZW [2] [3] [4] [11] [12] [13], LZSS and others. They are both dictionary coders, unlike minimum redundancy coders. LZ77 is the "sliding window" compression algorithm, which was later shown to be equivalent to the explicit dictionary technique first given in LZ78.

1.2. CLASSIFICATION OF DATA COMPRESSION ALGORITHMS (ACCORDING TO TYPE)

1.2.1. Lossless Data Compression

Lossless compression algorithms [5] usually exploit statistical redundancy in such a way as to represent the sender's data more concisely with fewer errors. Lossless compression is possible because most real-world data has statistical redundancy. For example, in English text, the letter 'e' is much more common than the letter 'z', and the probability that the letter 'q' will be followed by the letter 'z' is very small.

1.2.2. Lossy Data Compression

Another kind of compression, called lossy data compression or perceptual coding, is possible if some loss of fidelity is acceptable. Generally, a lossy data compression will be guided by research on how people perceive the data in question [9]. For example, the human eye is more sensitive to subtle variations in luminance than it is to variations in color. JPEG image compression works in part by "rounding off" some of this less-important information. Lossy data compression provides a way to obtain the best fidelity for a given amount of compression. In some cases, transparent (unnoticeable) compression is desired; in other cases, fidelity is sacrificed to reduce the amount of data as much as possible.

Lossless compression schemes are reversible so that the original data can be reconstructed, while lossy schemes accept some loss of data in order to achieve higher compression.

1.3. Adaptive Coding

Adaptive coding refers to variants of entropy encoding methods of lossless data compression. They are particularly suited to streaming data, as they adapt to localized changes in the characteristics of the data, and don't require a first pass over the data to calculate a probability model. The cost paid for these advantages is that the encoder and decoder must be more complex to keep their states synchronized and more computational power is needed to keep adapting the encoder/decoder state.

In this paper for implementing the dynamic compression algorithm we have used Tagged Sub Optimal Code (TSC) [6] [14] which is a very fast and simple algorithm for data compression as compared to the commonly used data compression schemes such as Byte pair encoding and Huffman Coding.

The remainder of this paper is organized as follows: Section 2 introduces the Tagged Sub Optimal Code (TSC) and advantages of TSC, Section 3 describes the implementation requirements and flow charts for our proposed dynamic data compression and decompression algorithm. Section 4 describes the complexity analysis of compression and decompression algorithm, in section 5 code generation scheme in TSC, section 6 describes some application of dynamic compression.
algorithm, section 7 describes the conclusion and future work.

II. TAGGED SUB OPTIMAL CODE (TSC)

TSC [6] is based on traversing a binary tree that generates variable length code words every two levels of the tree and delimited by the sequence of 01 or 10 bits. The simplicity of mapping the source message to static predefined code words from the sub optimal code, improves the compression process elapsed time drastically. TSC is faster and simpler than most of the other algorithms that are available today.

2.1 Advantages of TSC

- Experimental results have shown that TSC is 8.9 times faster in generating compressed text than huffman encoding.
- TSC is 14 times faster than Byte pair encoding (BPE) in the compression process.
- TSC is simple to implement for both compression and de-compression.
- TSC does not use a tree for generating the codes as huffman coding, so it takes less space. Moreover as no such tree is needed only the codes and the compressed text needs to be send.

2. Implementation Requirements And Flow Charts For Our Proposed Compression And Decompression Algorithm

2.1. Data Structures Used For Implementation Compression:

Hash table used to storing and searching the word.

```
struct hash {
    char *str;    /* Stores the word */
    int wrank;   /* Stores the word rank of the word*/
    struct hash *next; /*Pointer to the next hash*/
};
```

The Wtable is used for storing the word and the frequency of the word and is indexed by the word rank.

```
struct wtable {
    int frequency; /*Stores the frequency of the word*/
    struct hash *ptr2; /*Stores the address of the corresponding hash table structure*/
};
```

The array top contains the index to top of the word ranks having a particular frequency.

```
int top[MAX_SIZE];
```

**DECOMPRESSION:**

In this case we need only a modified wtable. The wtable is indexed by word rank and is responsible for assigning dynamic codes.

```
struct wtable {
    int frequency; /*Stores the frequency of the word*/
    char *str; /*Stores the string*/
};
```

The array top contains the index to top of the word ranks having a particular frequency.

```
int top[MAX_SIZE];
```

2.2. Flow Chart

**COMPRESSION:**

```
DECOMPRESSION:
```

© IJATER (IOCRSEM- 2014) 168
III. COMPLEXITY ANALYSES

FOR THE COMPRESSION STAGE:
- Complexity for calculating the hash value of word = O(string length of the word) * n (for n entries)
- Complexity for code generation for a wordrank = O(length of code generated) * n (for n entries)
(The code generation algorithm consists of the following major functions: String reverse, Decimal to binary conversion, code generation)
- Complexity for insertion into the hash table = O(1) * n (for n entries)
- Complexity of searching from the hash table = O(s) * n (for n entries) (s is the no of slots in a bucket of a hash table, for a good hash function) s \approx n/b
Where, n is the number of entries and b is the numbers of buckets used in the hash table
- Complexity of updating the wtable = O(1) * n (for n entries)
- Complexity for updating the top data structure = O(1) * n (for n entries)
- Complexity of compression into the string = O(b) * n (for n entries)

FOR THE DECOMPRESSION STAGE:
- Complexity for insertion into the wtable = O(1) * n (for n entries)
- Complexity for updating of the wtable = O(1) * n (for n entries)
- Complexity for modification of the top data structure = O(1) * n (for n entries)
- Complexity for generating word rank from code = O(length of code generated) * n (for n entries)
(The code generation algorithm consists of the following major functions: String reverse, Decimal to binary conversion, code generation)
- Complexity for searching from the wtable = O(1) * n (for n entries)
- Complexity of decompression = O(b) * n (for n entries) where b is the no of bytes in the string inserted

IV. APPLICATIONS

The applications of a dynamic compression scheme:
- This compression scheme can be used where real time compression is used, for example whenever a user uses a chat application, the messages are first compressed and then encrypted. Compression reduces the regular pattern in the messages so that the messages cannot be decrypted easily by any adversary. The dynamic nature of the compression scheme makes it suitable for such an application.
- In PDA’s and other handheld devices the amount of memory is very limited. So instead of first saving a file and then compressing it (which causes redundancy for that instant) we can directly save the file in the compressed format and then decompress the file to retrieve it as and when it is needed

V. CONCLUSION AND FUTURE WORK

The dynamic compression algorithm is very useful where real time compression needs to be done and speed of compression is a constraint. The dynamic word based tagged suboptimal coding scheme uses the features and advantages of tagged suboptimal coding (TSC) for its operation. The DTSC offers many advantages such as simple encoding scheme and uses minimum space and time complexity for its operation. The future work in the compression scheme involves many other aspects such as increasing the compression ratio using an even better compression technique, removing redundancies in the compression scheme and improving the space and time complexity of compression.

REFERENCES
Journal of Computer Science and Engineering Vol 1 No 4 416-425.


