



An optimized Code for Phrase Matching in Compressed Files in (S,C) Dense Code

Saurabh SharmaShobhit University, Meerut,
Uttar Pradesh, India**Dr. Shashank Awasthi**G. L. Bajaj Institute of Technology & Mgmt.
Uttar Pradesh, India**Dr. Niraj Singhal**Shobhit University, Meerut,
Uttar Pradesh, India

Abstract— Due to increase in the data dimension and inadequate network bandwidths there is urgent requirement of compressing the files. This technique of compression put aside the memory and faster transmission of data can be done over the network. Pattern matching on compressed files is one of the requirements for Information retrieval applications. Files that are compressed using (S,C) dense code compression technique helps significantly to reduce the time for searching as it avoids the decompression of the compressed file for finding the pattern. In this paper we propose an approach for phrase matching in the compressed files by modifying standard string matching algorithms like horspool and Sunday algorithm. This phrase matching can be used by search engines in relevant document retrieval for the given query. Pattern matching on (s, c) dense code compressed files had lots of advantages along with better compression ratios when compared to other standard algorithms of compression. Probing the text on the compressed files is up to 8 times faster when contrast to uncompressed file. We are going to present a new searching procedure for phrase searching in (S,C) dense code file. Frequency based codeword matching searching using standard algorithms with apt and appropriate modification in them can be applied to it.

Keywords- Phrase Matching; (S,C) Dense Code; Frequency based codeword searching

I. INTRODUCTION

The Problem pattern matching in compressed files can be defined as: given a pattern P and the text T where the pattern should be found in compressed text C which is obtained by some compression algorithm finding all the occurrence of P in C. The naïve approach to doing this pattern matching is to decompress the compressed text C and finding all the occurrences of the pattern P in the decompressed text. Due to unnecessary processing time in decompressing the text to find the pattern this solution is not the appropriate one. In order to reduce this I/O seek time a mechanism for finding the pattern in compressed file should be there. (S,C) Dense Code gives a improved compression ratio together with direct Barry-Moore type search on the compressed text [2]. In this paper we discuss the different byte oriented compression techniques in section 2. In the section 3 we mentioned pattern matching in compressed files. In upcoming section 4 we provide our approach to find the phrase in the compressed files using pattern matching algorithms with modification in them and had done an analysis. Finally in section 5 we had shown our experimental results and proved that our proposed technique is faster than the straight forward pattern matching.

II. BYTE ORIENTED COMPRESSION

The Huffman coding is a bit oriented coding applied to the characters. It gives the most approving prefix codes. Prefix codes can be referred to as the codes that ensure one code is not a prefix of the other. The compression is done taking characters as source symbols and assigning bit codes to characters [3]. On bit oriented coding, the decompression is slower and searching is a bit difficult. To overcome these semistatic statistical methods came into existence taking words as the source symbols for compression [4]. The use of byte codes taking words as the source symbols give better compression ratio [5].

The Plain Huffman codes are nothing but Huffman with source symbols as words and targets symbols are bytes. Tagged Huffman codes are one which uses the highest bit of each byte to notify the start of each codeword. So in a byte only 7 bits are useful for codeword and 1 flag bit is used to notify the starting of the codeword. The bit that signals make these codes a prefix code. In end tagged dense codes the signal bit is used to represent the end of the codeword instead of starting. This bit ensures that the codeword formed is prefix code words. So codes formed here are static which use only 7 bits. It was already realized that instead of making these codes static fixed to 128 the stoppers and continuers can be dynamic and can be decided based on the probability occurrences of the source symbols. These codes are named as (S,C) dense codes [6]. In the provided coding method the source symbols are assigned codeword in such a manner that the stopper is followed by continuer. We have to sort out the source symbols based on their frequencies and the words with more frequency are assigned to smaller byte codeword and less frequent words are assigned to larger byte codeword. Let b be the number of bits used for codeword then $S+C=2Y$. Hence $(2Y-1, 2Y-1)$ are nothing but end tagged dense codes. The variation in the (S,C) values is as per the variation with the size of the vocabulary and word frequency. The compression ratio of the (S,C) dense codes is dependent on the (S,C) values. The optimal (S,C) values can be found using the algorithm given in [5].

The encoding process of (S,C) dense codes is as follows:

- Sort out all the words in the decreasing order of their frequencies.
- Assign one byte code word to the first s frequent words starting from c to $s+c-1$.
- Next s to $s+sc-1$ words in the vocabulary are assigned with two byte codewords. The very First byte in the range $(0, C-1)$ and the second byte in the range $(C$ to $S+C-1)$. Likewise it follows the three byte codewords. Byte oriented codeword assignments are easier to assign and there is no need to store the codeword corresponding to the word. Using the index of the word these codewords are can be generated dynamically. with the byte oriented coding the encoding and decoding are faster [6]. The most vital advantage with bytecodes is they offer direct search on the compressed files. The Boyer-moore type of search can be easily performed which skips certain byte codes while searching.

III. RELATED WORK

The pattern matching in compressed files is one of the key benefits of byte oriented compressed schemes. In these the byte compressed files the Boyer-Moore type search can be applied direct with modification in them [2]. The change that should be added is to identify the false match of a codeword in (S,C) dense codes. When a match is found the previous codeword of the match should be inspected in order to check it is suffix of other codeword or matching of exact codeword. If earlier byte of the match is checked and if it is a continued then it is a false match. If it is a stopper then the match is correct match for the given pattern.

While we are going to match the pattern we should ensure that the following situations don't occur:

- i) It is granted that there is a matching between pattern P matches and some code word
- ii) Suffix of some other code word matches pattern P.
- iii) Pattern P is formed by the combination of two codewords. In prefix codewords case (i) don't occur. But in plain Huffman there is a chance of a case (ii) and (iii) to occur. In (S,C) dense code compressed file case (iii) doesn't occur but there is a chance of a case (ii) to occur. Consider (S,C) dense codes byte encoding scheme with $b=8$ bits and $(S,C) = (160,96)$. Let us consider some pattern P and Compressed text C. The process of matching the pattern over the compressed text is shown in Fig 1.

IV. PROPOSED METHOD

A. Phrase Matching in Compressed File:

Phrase matching is vital in an information retrieval system to give the interrelated consequences. This matching must be efficient with the intention of giving better results in minimum time. It is one of the challenges to match the given phrase in the compressed file. Semi static codes explore this benefit of direct matching in the compressed files. Due to its word based byte encoding scheme it's easy to make a direct search of any substring or phrase or pattern, in the compressed text.

Phrase matching in (S,C) dense code compressed files can be performed in two ways.

i) Straight forward code word searching In the straight forward method the codeword for the phrase is to be framed as shown in Fig 2. After obtaining the codeword of the phrase it is searched using some conventional pattern matching algorithm. By using this predictable pattern matching algorithm for codewords is same as for character but with proper modification is done to fit to byte codes.

ii) Frequency based code word searching In frequency based codeword searching the codeword for the phrase is to be framed as shown in Fig 2. After getting the codeword of the phrase they are maintained in low to high frequency order. Codeword of the phrase is searched based on the frequency in the searching algorithm. First the codeword with less frequency are searched within the pattern aligned with the coded text window. If the codeword is matched with its corresponding position in the coded text window we will match the next lower frequent codeword. If all the codewords of the phrase are matched with the coded text window then the phrase is occurring in the file. All the occurrences of the phrase are reported.

B. Analysis:

We analyzed that considering the frequency of the codewords rather than straight forward searching reduces the evaluation. Because we are searching in the aligned window based on word frequency the probability of the codeword matching is less when compared to left to right or right to left comparison of the window. We can conclude the pattern aligned window comparison loop instantly after a mis match and can decide the skip distance. Hence the number of comparisons in the aligned window is reduced. So the overall comparisons in the frequency based searching are less than straight forward searching. This reduces the search time in frequency based method than straight forward method. When the number of phrases to be searched is less then there will not be much difference in the performance of both the searching procedure. Frequency based searching gives better results if the number of phrases are more. In both the methods the phrase must not be too undersized or too large to obtain the maximum skips. If the phrase is too small then the number of skips will be fewer. Although the phrase is too big then the codeword of one word may be the suffix of the other codeword reducing the bounce distance. When no codeword of the word in the phrase is a suffix of the other codeword then the best case occurs. The first match of the codeword doesn't occur at all in the pattern also gives best case of the technique. There exists at least one single word in the phrase whose frequency is very less to make the loop terminate immediately from the matching window. These cases give the maximum number of skips with a small number of comparisons. The worst case of the algorithm take place when codewords of the words in the phrase are suffix of other codewords and the codeword in text appears in the pattern. This reduces skip distance of the pattern. Let the length of the

encoded text be 'n' and 'm' be the length of the codeword of the phrase. Applying the naïve pattern matching algorithm for the given phrase takes $O(mn)$ search time complexity and shifts the window exactly by 1 position. Straight forward horspool algorithm takes $O(m+\sigma)$ time for a preprocessing stage where σ is the space required to store the bad character shift array. It takes $O(\sigma)$ space complexity and $O(mn)$ search time complexity [7]. In case of frequency based horspool algorithm the space complexity is $O(\sigma +\alpha)$ where α is the space required to store the indexes of the codewords of the phrase in their frequency order. The Straight forward Sunday algorithm takes $O(m+\sigma)$ time for a preprocessing stage where σ is the space required to store the bad character shift array. It takes $O(\sigma)$ space complexity and $O(mn)$ search time complexity [8]. In case of frequency based Sunday algorithm the space complexity is $O(\sigma +\alpha)$ where α is the space required to store the indexes of the codewords of the phrase in their frequency order. The time complexity remains same as the straight forward approach. A Sunday algorithm looks at the codeword next to the encoded text window as this is the codeword to be compared next so it gives bit more shifts [8].

V. EXPERIMENTAL RESULTS

In this section we present the experimental results on phrase searching in (S,C) dense code compressed files. We used some text collection from the Calgary Corpus1 (CALGARY) to perform compression and phrase matching in them. We compressed the files using (S,C)-Dense Code(SCDC) with bytes as the target symbols ($b=8$). We performed searching for phrase on compressed files using naïve pattern matching algorithm, straight forward horspool algorithm, frequency based horspool algorithm, straight forward Sunday algorithm and frequency based Sunday algorithm with proper modification on them. The difference between the performances of straight forward horspool and frequency based horspool are significantly less when compared to the difference in performance between straight forward Sunday algorithm and frequency based Sunday algorithm. We performed the search for different number of phrases and presented the results.

VI. CONCLUSION

We have presented an approach for phrase matching in (S,C) dense code compressed files. The approach presented in this paper is using the statistics of the given phrase and based on that the search is performed. So our approach is having an advantage of reduced comparisons and faster search when compared to the straight forward search. We have presented the space and time complexity requirements of the proposed approach. From the experimental results it can be seen that our proposed "Frequency based Sunday algorithm" performs better than other algorithms. Based on our analysis we have given the best and worst cases in phrase matching for a given phrase. We had shown the as the number of phrases increase, the frequency based searching gives better results than straight forward searching.

REFERENCES

- [1] Turpin, A. and Moffat, A., "Fast file search using text compression", Proceedings of the 20th Australian Computer Science Conference, pp. 1–8, 1997.
- [2] Brisaboa, N., Farina, A., Navarro, G., and Parama, J., "Lightweight natural language text compression", Infor. Retrieval. 10, 1, pp.1-33, 2007.
- [3] Huffman, D. A., "A method for the construction of minimum redundancy codes", Proceedings of the Institute of Electronics and Radio Engineers (IRE) 40(9), pp.1098–1101, 1952.
- [4] Moffat, A., "Word-based text compression", Software - Practice and Experience 19(2), pp.185–198,1989.
- [5] E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates, "Fast and flexible word searching on compressed text", *ACM Transactions on Information Systems*, 18(2),pp.113–139, 2000.
- [6] Brisaboa, N., Fariña, A., Navarro, G. and Esteller, M., "(S,C)-dense coding: An optimized compression code for natural language text databases", Proceedings of the 10th International Symposium on String Processing and Information Retrieval (SPIRE'03), LNCS 2857, Springer-Verlag, pp. 122–136, 2003a.
- [7] Horspool, R. N., "Practical fast searching in strings", *Software Practice and Experience* 10(6), pp.501–506, 1980.
- [8] SUNDAY D.M., A very fast substring search algorithm, *Communications of the ACM*. 33(8), pp.132-142, 1990.