

# A Flexible Compressed Text Retrieval System Using a Modified LZW Algorithm

Nan Zhang, Tao Tao, Ravi Vijaya Satya, and Amar Mukherjee<sup>1</sup>

With an increasing amount of text data being stored in compressed format, being able to randomly access and partially decode the compressed data is highly desirable for efficient retrieval is required in many applications. For example, in a library information retrieval system, only the records that are relevant to the query are displayed. The efficiency of these operations depends on the compression method used. We present a modified LZW algorithm that supports efficient indexing and searching on compressed files. Our method performs in a sublinear complexity since we only decode a small portion of the file. The proposed approach not only provides the flexibility for the dynamic indexing in the different granularity of the text, but also provides the possibility for parallel processing in both encoding and decoding sides independent on the number of processors available. It also provides good error resilient property. The compression ratio is improved using the proposed modified LZW algorithm. Test results show that our public trie method has a compression ratio of 0.34 for TREC corpus and 0.32 with text preprocessing using star transform with an optimal static dictionary, which is very close to the efficient word Huffman and phrase based word Huffman schemes but has a more flexible random access ability.

In contrast to the current online LZW, our approach is to build a common dictionary trained by a large corpus. All the other texts are compressed using the common dictionary which is also called public trie. This dictionary only needs to be transmitted once and is stored in both the encoding and decoding side. During the construction of index file and decompression, the node numbers corresponding to index table entries will refer to the public dictionary. In our algorithm, we can decode any part of the text given the index of the dictionary entry and stop decoding when a certain tag is found or decode a given number of symbols. Moreover, the modified LZW uses fixed length code for each node. We choose the length of the code to be 16 bits/node. That is, the code is at byte level so that we can still perform fast random access when the indexing granularity increases or decreases.

We add tags or use symbols in the text for different levels of details instead of a fixed level of granularity. The pointers in the inverted index file will be built on the tags. The decoding will be performed by the nature of the query or by user defined level. For fixed level partitions if we need to access more details than the fixed blocks, the new pointers are needed in the index table pointing to the bit level in the compressed text instead of at byte level which may make the pointer size larger. For example, it needs an extra byte per pointer to achieve the level of granularity as our approach does. In our coding scheme, each node is encoded with fixed number of bits. In case there is an error occurring at some position, the error is limited within the range of subsequence represented by the corresponding node. Further decoding will not be affected.

---

<sup>1</sup> School of Computer Science, University of Central Florida {nzhang, ttao, rvijaya, amar}@cs.ucf.edu. This research is partially supported by NSF grant number: IIS-9977336 and IIS-0207819.