

Compressing an Inverted File with LCS

Fang-Yie Leu

leufy@mail.thu.edu.tw

Yao-Chung Fan

G912810@student.thu.edu.tw

Computer Science and Information Engineering

Tung-Hai University, Taichung, Taiwan

1. Introduction

Constructing a document index, a mechanism for locating the given terms in the document addressed, for future convenient, efficient and accurate retrieval is one of the most important concerns in designing an information retrieval system. The most common index structure used in document retrieval is the inverted file which consists of inverted lists holding lists of pointers to all the locations of the given terms in the documents collected.

Nevertheless, an inverted file of full text retrieval often occupies space more than 50% of the indexed text [1]. Given an information system retaining N documents and an index containing M pointers in average, the space required for the inverted list is $M * \lceil \log N \rceil$ bits. Retrieving information from a document system, search engine is the most popular one used. Google claims that their system involves two billions documents. Assuming that each only extracts five keywords for indexing, such a massive index system requires 40GB memory storage. So far, most current computers and workstations cannot process such enormous data efficiently. However, the size of an inverted file can be reduced by exploiting compression techniques. A wide range of index compression approaches have been developed over years. However, conventional inverted lists are often passively compressed with delta coding or difference coding based on the locality of documents to be indexed. Instead, [2] proposes an algorithm to permute the document numbers by using document similarity in order to generate the locality for an inverted index.

Let $F = \{I_1, \dots, I_m\}$ be an inverted file, where m is the number of terms identified from the n documents collected, $I_i = \{d_{i,1}, \dots, d_{i,|I_i|}\}$ a list of documents containing term i , $1 \leq d_{i,j} \leq n$, $1 \leq j \leq |I_i|$, and $|I_i|$ the length of I_i . The ultimate

goal of the inverted-file compression with difference

code is to minimize $C(F) = \sum_{i=1}^m \sum_{j=1}^{|I_i|} (d_{i,j} - d_{i,j-1})$,

the cost of encoding F .

2. Algorithm

Given a document system $S(D, T)$ consisting of a document set D and a term set T , we can represent T and D each as a bipartite graph, and the edges are the relation "indexing" between D and T . Algorithmically, we can draw a term graph $G_t(T, E_t)$ and a document graph $G_d(D, E_d)$. Specifically, in G_t two terms are connected if and only if they at least appear in one document. Similarly, in G_d two documents are connected if and only if both of them have at least one common term.

Our compression algorithm comprises two modules: clustering and indexing (numbering). Firstly, we recursively partition the G_d , in the sense of balance, into subgraphs by exploiting randomized Minimum Spanning Tree (MST) partition algorithm [3][4]. The well-known graph theoretic divisive clustering algorithm is based on the construction of the MST of the data, and then deleting the edge with lowest weight to generate clusters. The MST algorithms commonly known as Prim's and Kruskal's algorithms are all deterministic and comparison-based. For sparse graphs, using such algorithms the fastest complexity can be run in $O(E(G_d) \log V(G_d))$. In this article, we exploit randomized MST algorithm proposed by [4] in which uses the spanning tree verification and randomized sampling. This algorithm gives MST an expected time $O(V(G_d) + E(G_d))$. With this algorithm, the MST of the given graph can be quickly calculated

and the following partition algorithm can be recursively performed.

Next the vertices (documents) in a local clique split are then indexed by BFS (Breadth First Search) graph traversal algorithm which systematically explores the edges of G_d to expansively visit all the vertices. That is, the algorithm discovers all the vertices at distance k from the starting vertex after finding out all the vertices at distance $k-1$. Conceptually, BFS gives the neighbors of an indexed vertex v the index numbers closer to v than other index strategies, e.g., Depth First Search, do. As a consequence, $C(F)$ based on difference code can be minimized as small as possible. This approach is somehow analogous to the exploitation of document similarity [2][5].

We perform the same operations stated above on G_t . F can be then separated into $F_1 \dots F_{c+1}$, where c is the number of partition performed on G_t . We can imagine that the terms in the same clique may be the synonym or some that are highly correlated. Meanwhile, given two terms list $l_i = \{d_{i,1} \dots d_{i,|l_i|}\}$ and $l_j = \{d_{j,1} \dots d_{j,|l_j|}\}$, $l_i, l_j \in F_k$, if term i appears in document D_a , there is a high probability that term j also exists in D_a resulting in the fact that l_i and l_j have a large longest common subsequence (LCS). In biological applications, the way to measure the similarity of two strings $s1$ and $s2$ is by finding a third string $s3$ in which the terms in $s3$ both appear in $s1$ and $s2$. These terms must appear in the same order, but not necessarily consecutively. The longer the string $s3$ we can obtain, the more similar $s1$ and $s2$ are. This LCS problem can be solved efficiently with dynamic programming. Its running time is $O(|s1|*|s2|)$, where $|s1|$ and $|s2|$ are the length of given strings.

Accordingly, let $LCS(F_k)$ be the LCS of all the terms in F_k . We can use two-way computational tree to yield $LCS(F_k)$ in $O(|F_k|*|l_{avg}|^2)$, where $|F_k|$ is the number of list in F_k and $|l_{avg}|$ the average length of each $l_i \in F_k$.

Namely, for each $l_i = \{d_{i,1} \dots d_{i,|l_i|}\} \in F_k$, let $l_i' = l_i - FCS(F_k)$, $C(F)$ can be again minimized as follows.

$$C(F) = \sum_{k=1}^{c+1} \left(\sum_{i=1}^{|F_k|} \sum_{j=1}^{|l_i'|} (d_{i,j} - d_{i,j-1}) + LCS(F_k) \right)$$

$$= \left(\sum_{i=1}^m \sum_{j=1}^{|l_i'|} (d_{i,j} - d_{i,j-1}) \right) + \sum_{k=1}^{c+1} LCS(F_k)$$

3. Conclusion

Our conclusion is that through the bipartite model, both G_d and G_t of an information system can be generated. Conceptually, by invoking the classification of G_d the difference code $(d_{i,j} - d_{i,j-1})$ can be optimized. Following, the length of inverted-list, say $|l_i|$, can be shortened by the classification of G_t and the replacement of l_i by l_i' . As a consequence, $C(F)$ can be tremendously reduced. In the future, we plan to implement our algorithm by using .gov2 data of TREC (Text REtrieval Conference) and give comparisons by similarity algorithm [2] and PI parse proposed by [6].

Reference:

- [1] Witten I.H, A. Moffat, and T.C. Bell, "Managing Gigabytes: Compressing and Index Documents and Image," Morgan Kaufmann, second edition, 1999.
- [2] D. Blandford and G. Bleeloch, "Index Compression through Document Redoering," Data Compression Conference, 2002, pp 342-351.
- [3] C. T. Zahn, "Graph-theoretical methods for detecting and describing gestalt clusters." IEEE Trans. Computer. C-20 (Apr.), pp 68-86.
- [4] D. R. Karger, P.N. Klein, and R.E. Tarjan, "A randomized linear time algorithm to find minimum spanning Trees," Journal of the Association for Computing Machinery, Vol 42, No 2, 1995, pp 321-328.
- [5] F. Y. Leu and Y. C. Fan, "Compressing a Directed Massive Graph using Small World Model," Data compression Conference, 2004, pp 550.
- [6] F. Y. Leu and Y. C. Fan, "Dealing with the structure of Real World Graph," International Conference on Computer Science and its applications, Proceeding ICCSA 2004, pp 40-49