An improved Approach for Document Retrieval Using Suffix Trees

N. Sandhya
Associate Professor,
CSE Department,
GRIET,
Hyderabad, Andhra Pradesh, 500 072, India

Dr. A. Govardhan
Principal,
JNTUH College of Engineering
Jagityal, Andhra Pradesh 500 501, India

Y. Sri Lalitha
Associate Professor, CSE Department,
GRIET,
Hyderabad, Andhra Pradesh, 500 072, India

Dr. K. Anuradha
Professor and Head,
CSE Department,
GRIET,
Hyderabad, Andhra Pradesh, 500 072, India

***Abstract*—Huge collection of documents is available at few mouse clicks. The current World Wide Web is a web of pages. Users have to guess possible keywords that might lead through search engines to the pages that contain information of interest and browse hundreds or even thousands of the returned pages in order to obtain what they want. In our work we build a generalized suffix tree for our documents and propose a search technique for retrieving documents based on a sort of phrase called word sequences. Our proposed method efficiently searches for a given phrase (with missing or additional words in between) with better performance.**

Keywords-Document retrieval; Frequent Word Sequences; Suffix tree; Traversal technique.

I. INTRODUCTION

With the growth of web, hundreds of millions of people engage in information retrieval process every day when they use web search engine or search their emails. IR is fast becoming the dominant form of information access, overtaking traditional database style searching. IR process begins when user enters a query like search strings in web search engines, phrases etc. to identify the related documents or URLs.

Now almost all the documents have electronic copies. With the development of WWW it is an efficient technique to retrieve the documents using the web search engines based on a query. But this should not be time consuming. That is the reason precision of the retrieval of related documents for a given query is vital for the search engine. Cluster based information retrieval techniques also exist [11].

The next section deals with the Information Retrieval and its related work on text documents. Section 3 describes Suffix Tree. Section 4 deals with building generalized suffix tree. Section 5 explains traversal technique Algorithm used for quick retrieval of documents. Section 6 shows the experiment setting, results and analysis. Section 7 concludes and discusses future work.

I. RELATED WORK

Information Retrieval for a given query is retrieving relevant documents efficiently. An application needs to be developed that facilitates the user with an efficient retrieval of the information that is needed. Search engines are the keys to find specific information on the World Wide Web. Without sophisticated search engines, it would be virtually impossible to locate anything on the Web. A search engine is a program that searches documents for specified keywords and returns a list of the documents where the keywords are found based on certain algorithms. Document clustering has initially been investigated in Information Retrieval mainly as a means of improving the performance of search engines by pre-clustering the entire corpus [12].

The assumption (implicitly or explicitly) upon which most commercial information retrieval systems are based is that if a query and a document have a keyword in common, then the document is about the query to some extent and if there are more key words in common, then the document is about the query. In this respect, an IR System represents the documents and also query in separate vector space models as document - terms matrix, where each column indicates terms in documents and rows correspond to documents in IRS. This representation is also called “bag of words” mechanism. An IR system matches the bag of keywords in the user’s query with the bag of keywords representing the documents to identify related documents and this approach suffers from a number of problems.

- It does not handle lexical variation, i.e. different words are used to represent the same meaning or concept in queries and documents.
- It cannot deal with semantic variation, where a single word has multiple meaning.

- It does not handle properly syntactical variation, i.e. words that co-occur in multiple documents are probably related.
- Morphological variations i.e. words appearing in different numbers (singular or plural) and in different cases (Active, passive cases)

All the above problems hurt the retrieval system in terms of precision and recall [8].

To overcome the bag of words problems, here we choose to treat text documents as sequence of words and to retrieve documents that share frequent word sequences from text databases. The sequential relationship between the words and documents is preserved using suffix tree data structure.

II. SUFFIX TREE

A suffix tree is a data structure that admits efficient string matching and querying. Suffix trees have been studied and used extensively, and have been applied to fundamental string problems such as finding the longest repeated substring [6], strings comparing [4], and text compression [5]. The suffix trees became useful as the search time is independent of the length of the string. The following description of the suffix tree was taken from Dan Gus field's book on Strings, Trees and Sequences [7]. A suffix tree of a string is simply a compact trie of all the suffixes of that string. Here we treat documents as sequences of words, not characters. The main purpose of using Suffix trees is that it is used to identify the document IDs that contain the suffixes efficiently because the leaf nodes of suffix tree stores documents IDs. Suffix trees are useful in clustering [10].

A. Definition

A suffix tree T for an m-word string S is a rooted directed tree with exactly m leaves numbered 1 to m. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of S. The label of a node is defined to be the concatenation of the edge-labels on the path from the root to that node. No two edges out of a node can have edge labels beginning with the same word. For each suffix s of S, there exists a suffix node whose label equals s.

The suffix trees are fast, incremental and are constructed in linear time of the suffixes generated.

III. CONSTRUCTION OF SUFFIX TREES FOR DOCUMENTS

A text document D is viewed as a sequence of words, so that it can be represented as D = (w1, w2, w3 . . .), where w1, w2,w3, . . . are words appearing in D. Like a frequent itemset in the association rule mining of a transaction data set [9], a word set is frequent when at least the specified minimum number (or percentage) of documents contains this word set. A frequent word set containing k words is called frequent k-word set.

A frequent k-word sequence is an FS with length k, such as FS = (w1, w2. . . wk), and it has two frequent subsequences of length k - 1, which are (w1, w2, . ., wk-1) and (w2,w3, . . ,wk). [2].

In our work finding the frequent word sequences has two steps: finding frequent 2-word sets first, then finding frequent word sequences of all length by using the Generalized Suffix Tree (GST) data structure.

A. Finding frequent 2-word sets

The goal of this step is to reduce the dimension of the database (i.e. the number of unique words) by eliminating those words that are not frequent enough to be in a frequent k-word sequence, for k >= 2. This step is simple and straight forward. We use an association rule miner to find the frequent 2-word sets that satisfy the minimum support. All the words in frequent 2-word sets are put into a set. After finding the frequent 2-word sets, we remove all the words in the documents that are not in WS. After the removal, the resulting documents are called compact documents. Let us consider an example database

D = {d1, d2, d3}:

d1: Almost all children eat chocolates.
d2: Some of the children eat dry fruits.
d3: Children like to eat dry fruits and chocolates.

There are 13 unique words in this database D: {all, and, almost, children, chocolates, dry, eat, fruits, like, of, some, to, the}. If we specify the minimum support as 60%, the minimum support count is 2 for this case. The set of frequent 2-word sets is {{children, eat}, {children, chocolates},{eat, chocolates}, {children, dry}, {children, fruits}, {dry, fruits},{eat, fruits}, {eat, dry}, }; and WS = {children, eat, chocolates, dry, fruits}. After removing those words not in WS, the database D becomes D'={d1', d2', d3'} as follows, where the removed words are shown in parentheses.

• d1': (Almost all) children eat chocolates.
• d2': (Some of the) children eat dry fruits.
• d3': children (like to) eat dry fruits (and) chocolates.

Thus reducing the dimensions of documents, this has a considerable impact in the next step i.e. building of a generalized suffix tree. To find frequent word sequences of the database, we adopted the suffix tree [6], a well known data structure for sequence pattern matching, to find all the frequent word sequences. Each compact document is treated as a string of words and inserted into a generalized suffix tree (GST) one by one. Finally, by collecting the information stored in all the nodes of the GST, we can find all the frequent word sequences of the database.

A suffix tree for a string S is actually a compressed trie for the non-empty suffixes of S. A GST is a suffix tree that combines the suffixes of a set of strings. In our case, we build a GST of all the compact documents in the text database.

Each suffix node has a box attached, and it contains the document id set of the suffix node. After building the GST, we traverse it by depth-first. On the way down, the labels of the edges are concatenated to become the string L of each node. On the way up, each child node sends its document id set to its parent. The support count of the label (i.e. string L) of this

parent node is the size of the union of all the document id sets of its children. By checking the support count and the length of the label of each node, we can get the information about all the frequent word sequences in the database. In our example shown above, we have seven nodes in the GST, and the details are given in Table I.

Since the minimum support for frequent words is set to 60% in this example, the minimum support for frequent word sequences could not be smaller than 60%. Only those words whose support is at least 60% are kept in the compact documents, so that we can find only the frequent word sequences with that minimum support.

TABLE I: WORD SEQUENCES ASSOCIATED WITH THE NODES IN FIGURE 1

Node no	*Word sequence*	*Length of word sequence*	*Document Ids*
1	chocolates	1	D1.txt D3.txt
2	dry fruits	2	D2.txt D3.txt
3	dry fruits chocolates	3	D3.txt
4	fruits	1	D2.txt D3.txt
5	fruits chocolates	2	D3.txt
6	children eat	2	D1.txt D2.txt D3.txt
7	children eat chocolates	3	D1.txt
8	children eat dry fruits	4	D2.txt D3.txt
9	children eat dry fruits chocolates	5	D3.txt
10	eat	1	D1.txt D2.txt D3.txt
11	eat chocolates	2	D1.txt
12	eat dry fruits	3	D2.txt D3.txt
13	eat dry fruits chocolates	4	D3.txt

After building the suffix tree as mentioned above, we traverse the tree for a given word sequence " eat chocolates ", which should retrieve all the documents that contain **"children eat chocolates", "children eat dry fruits and chocolates" " children of four years eat many chocolates".** To perform this we propose SuffixTree algorithm for sequence of words. Here we used level order traversal to find the word "eat", after getting to the node with word "eat", perform depth first search to get all the strings that start with node "eat" applying k-mismatch method to get the document ids of all the documents in which the word sequence occurs.

IV. ALGORITHM

Step 1: Given word sequence to be searched is tokenized first.

Step 2: Initialize n to 1
Search the nodes of root for first token
If there is a match
Perform step3
Else
If n is last node of root +1
Return search failed
Else
Increment n and again search with next node of root

Step 3: Consider only the Level1 nth node sub tree
Compare next token with Level2 first node
do
{
If there is match
(i) perform depth first search traversal on the tree comparing with remaining tokens
(ii) applying K-mismatch retrieve the documents that contain the word sequence
Else
If all the Level1 nth node sub tree nodes are traversed word sequence is not present
Else
apply K-mismatch, perform DFS traversal and compare with the next token
}
While all the tokens of word sequence are not completed or entire Level1 nth node sub tree nodes are not traversed

V. EXPERIMENTAL SETUP

Suffixes of the phrases are generated [1, 10]. We treat the documents as a sequence of words instead of bag of words. Then the similarity measurements are done based on the shared frequent word sequences among the documents. Each document is reduced to a compact document by keeping only the frequent words [1]. A generalized suffix tree for all the compact documents is built. The frequent word sequences and the documents sharing them are found. We proposed an approach to improve the precision of retrieval. We used level order traversal with depth first traversal of the GST to search for the related documents based on word sequence.

A. Cleaning of documents and generating suffixes

Preprocessing of documents involves removal of all the special symbols called nonword tokens (such as numbers, HTML tags, and most punctuation) from each of the documents, splitting the document contents line wise, removing unwanted characters, removal of stop words and stripping other text. Sentence boundaries are marked. Identifying sentence boundaries is an important task as the approach used here is word sequence based method for finding suffixes.

B. Generating Suffixes and Building GST

The suffix of each line of the document is generated. After finding all the suffixes of all the documents we extracted unique suffixes and built compact suffix tree with these unique suffixes. All Suffixes and Unique Suffixes for the given example are shown in the Table II. We pruned the tree with nodes that do not satisfy the user specified threshold, thus the

tree contains only those nodes that are frequent word sequences. A Generalized Suffix Tree (GST) for the given example is shown in Fig 1. Then we apply our algorithm for traversing the GST in order to retrieve documents when a phrase with missing or additional words is given.

TABLE II. RESULTS

All suffixes	*Unique suffixes*
children eat chocolates	children eat chocolates
children eat dry fruits	children eat dry fruits
children eat dry fruits chocolates	children eat dry fruits chocolates
chocolates	chocolates
chocolates	dry fruits
dry fruits	dry fruits chocolates
dry fruits chocolates	eat chocolates
eat chocolates	eat dry fruits
eat dry fruits	eat dry fruits chocolates
eat dry fruits chocolates	fruits
fruits	fruits chocolates
fruits chocolates	

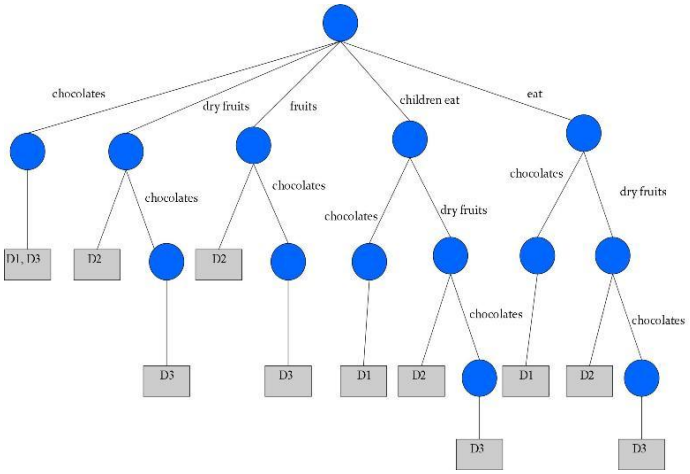

Figure 1. Generalized Suffix Tree:

VI. CONCLUSIONS

Our method is an efficient method over phrase based retrieval. Phrase based retrieval requires the complete phrase to appear in documents. We relaxed this condition and matched related phrases using k-mismatch method. Here by using k-mismatch the words sequence is maintained with some additional words/missing words in between. Our method proved efficient compared to simple phrase based retrieval in two aspects: a) we pruned the infrequent terms thus reduced the dimensions and b) proposed algorithm for efficient suffix tree traversal. Our results have shown better performance than simple phrase matching. Suffix trees can be constructed incrementally. In our future work we would like to apply this technique on Opinion Mining. Also we would like to extend this technique for concept retrieval. We also would like to investigate the application of this algorithm in cluster based information retrieval with hierarchal, hybrid and incremental clustering.

REFERENCES

[1] Yanjun Li, Soon M. Chung, John D. Holt Text document clustering based on frequent word meaning sequences.

[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th VLDB Conference, 1994, pp.

[3] Horatiu Mocian Text Mining with suffix trees.

[4] A. Ehrenfeucht and D. Haussler. A new distance metric on strings computable in linear time. Discrete Applied Math, 1988.

[5] M. Rodeh, V. R. Pratt, and S. Even. Linear algorithm for data compression via string matching. In Journal of the ACM, pages 28(1):16–24, 1981.

[6] P. Weiner. Linear pattern matching algorithms. In The 14th Annual Symposium on Foundations of Computer Science, pages 1–11, 1973.

[7] D. Gusfield. In Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press, 1997.

[8] A.T. Arampatzis, T.Tsoris, C.H.A. Koster, Th.P. Vander Weide, RIAO 97 Proceeding, Phrase based information retrieval.

[9] Wikipedia. (2006). *English on the Internet*. http://www.wikipedia.org Accessed on 1st December 2006.

[10] Oren Eli Zamir. Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results

[11] Anastasios Tombros. The effectiveness of query-based hierarchic clustering of documents for information retrieval

[12] Jardine, N. and van Rijsbergen, C. J. The use of hierarchical clustering in information retrieval. *Information Storage and Retrieval*, 7:217-240, 1971.

AUTHORS PROFILE

N.Sandhya B.Tech, M.Tech (Ph.D).
Working as an associate professor in the CSE Dept of GRIET. Has 11years of experience in teaching. My areas of interest are Databases, Information Retrieval, Data Mining and Text Mining.

Y.Srilalitha M.Tech (Ph.D).
Working as an associate professor in the CSE Dept of GRIET. Has 16 years of experience in teaching. My areas of interest are Information Retrieval, Data Mining and Text Mining.

Dr.K.Anuradha M.Tech, Ph.D.
Working as a professor and Head of the CSE Dept in GRIET.
Has an experience of 25 years in teaching. My areas of interest are Information Retrieval, Data Mining and Text Mining.

Dr.A.Govardhan M.Tech, Ph.D.
Working as a professor and Principal of JNTU, Jagityal. Has an experience of 20 years in teaching. My areas of interest are Information Retrieval, Databases, Data Mining and Text Mining.