Phrase Based Document Retrieving by Combining Suffix Tree index data structure and Boyer- Moore faster string searching algorithm
B.Ganga, Research student, Email gangabalu@yahoo.com

ABSTRACT
Phrase has been considered as a more informative feature term for improving the effectiveness of document retrieval. This paper propose an Algorithm A Phrase Based Document Retrieval to retrieve the similar documents by combining two exiting algorithm suffix tree, index data structure and “The Boyer-Moore Algorithm”, faster string searching algorithm. The suffix tree is constructed based on E. Ukkonen, “on-Line Construction Of Suffix Trees For Strings, a most efficient string-matching algorithm. On the constructed suffix, “The Boyer-Moore Algorithm” is applied to check the presence of pattern i.e. the input phrase in order and without order to retrieve the similar documents. Furthermore, by studying the property of suffix tree and Boyer-Moore, we conclude that suffix tree data structure store huge documents and Boyer-Moore algorithm checks the presence of pattern fastly. This conclusion sufficiently explains why the Phrase Based Document Retrieval works much better than the other document retrieval.

KEYWORDS: Suffix tree, Boyer-Moore, Document retrieval.

1. INTRODUCTION
Phrase identification is an important task of Document Retrieval. Most of the retrieval techniques are designed and based on words or keywords and its occurrence. Concepts are often expressed as phrases consisting of multiple words whose meaning is substantially different from the meaning of the individual words. For example the phrase “Artificial Intelligence” is different than individually “Artificial” and “Intelligence”, and “Operating System” is different than the individual words “Operating” and “System”. Composite expressions are prevalent in natural language, hence there is a need in the information retrieval systems to have a methodology that identifies phrases and do a retrieval based on it.

Document representation is a major problem in the information retrieval. The Phrase Based Document retrieval algorithm combine advantages of two algorithm Ukkonen’s, “suffix tree”; for strings and boyer-moore string search algorithm. Suffix tree is used to represents the document as suffix of the tree and the “Boyer-Moore”, string searching algorithm is used to search input phrase in constructed suffix in order and without order to retrieve the similar documents together.

The purpose of implementing suffix tree is, it is efficient data structures to store the large text, faster and number of nodes created is less when compared to suffix trie, the previous algorithm and “The Boyer-Moore Algorithm”, is faster string searching algorithm.

This paper emphasizes on phases present in documents and do a retrieval based on it. The Phrase Based Document Retrieval Algorithm represent each document as suffix trees, a data structure where phrases of the documents are stored as suffixes of tree and Boyer Moore algorithm is used to check the presence of pattern i.e. the input phrase in order and with out order.

Suffix tree is constructed by using E.Ukkonen, “on-Line Construction Of Suffix Trees For Strings” and enhanced to represent suffix tree for any type of document namely Portable Document Format, MS Format Files like MS Word, MS PowerPoint, MS Excel and Text files. And here edges of suffix tree are suffix i.e. phrases of the documents. And once suffix tree is constructed, several operations can be performed quickly, for instance locating a substring in, locating matches for a regular expression or pattern etc. suffix tree retrieval performance is better compare to other form based retrieval and searching becomes faster with suffix tree construction.

The Boyer-Moore string search algorithm is a particularly efficient algorithm, and has served as a standard benchmark for string search algorithm ever since. This algorithm’s execution time can be sub-linear, as not every character of the string to be searched needs to be checked. Generally speaking, the algorithm gets faster as the target string becomes larger.

1.1 Objectives of the work
The objectives include
• Retrieving similar documents which has input phrase in order and without order by combining two methodologies, the suffixtree and Boyer-Moore algorithm.
• And enhanced to retrieve the documents of different form such as Portable Document Format, MS Format files and Text files.
• The approach was an experiment to observe the proposed methodology’s ability i.e. using phrase to improve the retrieval.

The rest of this paper is organized as follows: Section 2 discuss the literature survey. Section 3 starts with design of Phrase Based Document Retrieval. Section 4 illustrates the testing and experimental results. Finally Section 5 summarizes the contributions of work.
2. LITERATURE SURVEY

2.1 Document Retrieval

Document Retrieval is defined as the matching of some user stated query against a set of free-text records [1]. These records could be any type of mainly unstructured text, such as newspaper articles, real estate records or paragraphs in a manual. User queries can range from multi-sentence, full descriptions of an information need to a few words.

2.2 Document Retrieval System

Document Retrieval system consists of a database of documents, a classification algorithm to build a full text index, and a user interface to access the database. The system finds information to given criteria by matching text records (documents) against user queries, as opposed to expert systems that answer questions by inferring over a logical knowledge database.

And two main classes of indexing schemata for document retrieval systems are form based (or word based), and content based indexing. The document classification scheme (or indexing algorithm) in use determines the nature of the document retrieval system.

Form based document retrieval addresses the exact syntactic properties of a text, comparable to substring matching in string searches. The text is generally unstructured and not necessarily in a natural language, the system could for example be used to process large sets of chemical representations in molecular biology.

A suffix tree algorithm is an example for form based indexing.

The content based approach exploits semantic connections between documents and parts thereof, and semantic connections between queries and documents. Most content based document retrieval systems use an inverted index algorithm.

2.3 Index Data Structures

Search engine architectures vary in the way indexing is performed and in methods of index storage to meet the various design factors. Types of indices include [2]:

2.3.1 Suffix Tree

Figuratively structured like a tree, supports linear time lookup. Built by storing the suffixes of words. The suffix tree is a type of trie. Tries support extendable hashing, which is important for search engine indexing. Used for searching for patterns in DNA sequences (Deoxyribonucleic acid is a nucleic acid that contains the genetic instructions used in the development and functioning of all known living organisms and some viruses) and clustering.

2.3.2 Tree

A tree is a widely-used data structure (way of storing and organizing data) that emulates a hierarchical tree structure with a set of linked nodes. And it is an acyclic connected graph where each node has a set of zero or more child nodes, and at most one parent node. An ordered tree data structure is used to store an associative array where the keys are strings. It's regarded as faster than a hash table but less space-efficient.

2.3.3 Inverted index

In information technology, an inverted index is an index data structure storing a mapping from content such as words or numbers, to its location in data base file allowing full text search.

And it stores a list of occurrences of each atomic search criterion, typically in the form of a hash table or binary tree.

2.3.4 Ngram index

Stores sequences of length of data to support other types of retrieval or text mining.

2.3.5 Term document matrix

Used in latent semantic analysis, stores the occurrences of words in documents in a two-dimensional sparse matrix (a matrix populated primarily with zeros).

2.4 Suffix Trees

2.4.1 Tries and Trees

A trie (from retrieval), is a multi-way tree structure useful for storing strings over an alphabet. It has been used to store large dictionaries of English (say) words in spelling-checking programs and in natural-language "understanding" programs. For example the given the data is BCABC then the corresponding trie would be as shown in Figure 2.2.

The advantage of suffix trie is that if you have an input text of length \( n \) and a search string of length \( m \), a traditional brute force search will take as many as \( nm \) character comparison to complete, the suffix trie demolishes this performance by requiring just \( m \) character comparisons, regardless of the length of the text being searched.

The disadvantage in trie is that more space is wasted as a lot of nodes near the edge of the trie will have most sub tries set to nil, number of nodes created is more.
2.4.2 Suffix Trees

Introduction
The suffix tree for a given block of data retains the same topology as the suffix trie, but it eliminates nodes that have only a single descendant. The tree has the same general shape as trie just far fewer nodes. By eliminating every node with just a single descendant, the count is reduced.

Suffix Tree Mechanisms
Suffix Tree Mechanisms start at longest suffix and work our way down to shortest suffix. Each suffix ends at a node which is of 3 types. First one is Leaf nodes which are defined as all the suffixes that are longer than the suffix defined by the active point. Second is explicit node which is the non-leaf nodes where 2 or more edges part way. Third is implicit node which is the non-leaf nodes whose prefixes all ends in the middle of the edges.

Suffix is represented by defining its character. As first character starts at node 0 i.e. Suffix object defines the last character in a string by starting at a specific node then following the string of characters in the input sequence pointed to by the first character index and last character index members

Definitions
The suffix tree for the string S of length n is defined as a tree such that the paths from the root to the leaves have a one – to – one relationship with the suffixes of S and edges spell non- empty strings. And all internal nodes (except the root) have at least two children[3].

The suffix tree for the string BCABC is as follows and compared to suffixtrie the number of nodes created is less.

Definitions


* Weiner was the first to show that suffix tree can be built in linear time.

* McCreight introduced a more space efficient linear-time algorithm in 1976.

McCreight’s original algorithm for constructing a suffix tree had a few disadvantages, principle among them was the requirement that the tree be built in reverse order, and meaning characters were added from the end of the input. This ruled the algorithm out for the on-line processing, making it much more difficult to use for applications such as data compression.

* Ukkonen developed a simpler to understand linear-time algorithm in 1995.

Ukkonen’s algorithm was slightly modified version of the algorithm that works from left to right [5]. For a given string of text, T, Ukkonen’s algorithm starts with an empty tree, then progressively adds each of the n suffixes of T to the suffix tree. For example, when creating the suffix tree for BANANAS, b is inserted into the Tree, then BA, then BAN, and so on. When BANANAS is finally inserted, the tree is complete.

Algorithm for Updating the Suffix [4]

Build suffix tree T for string S [1...m]

Begin

Figure 2.2 Trie for string BCABC

Figure 2.3 Suffix tree for String BCABC

Figure 2.4 Progressively Building the Suffix Tree
1) Build the tree in \( m \) phases, one for each character. At the end of phase \( i \), we will have tree \( T_i \), which is the tree representing the suffix \( S[1...i] \) an online construction.

2) In each phase \( i \), we have \( i \) extensions, one for each character in the current suffix. At the end of extension \( j \), we will have ensured that \( S[j...i] \) is in the tree \( T_i \).

The algorithm begins with an implicit suffix tree containing the first character of the string. Then it steps through the string adding successive characters until the tree is complete. This order addition of characters gives Ukkonen's algorithm its "on-line" property, earlier algorithms proceeded backward from the last character.

### 2.5 String Matching Algorithm

- **Given a pattern** of length \( m \) and a body of text of length \( n \), return true if the pattern is found in text, or false otherwise. Including computational biology, computer science, and mathematics [6].
  - Applications – Obvious – find text in a document, on a webpage, etc.

#### 2.5.1 Brute Force

The brute force algorithm consists in checking, at all positions in the text between \( 0 \) and \( n-m \), whether an occurrence of the pattern starts there or not. Then, after each attempt, it shifts the pattern by exactly one position to the right. The brute force algorithm requires no preprocessing phase, and a constant extra space in addition to the pattern and the text. During the searching phase the text character comparisons can be done in any order. The time complexity of this searching phase is \( O(mn) \) (when searching for \( a^m \) in \( a^n \) for instance). The expected number of text character comparisons is \( 2n \).

#### 2.5.2 Horspool’s Algorithm

Step 1 – For a given pattern \( P \) of length \( m \), compute the shift table.
Step 2 - Align the pattern against the beginning of the text.
Step 3 – Starting with the last character in the pattern, compare text and pattern.
  - If pattern matches, return success
  - If pattern does not match, use the mismatched character from the text as \( c \). Lookup \( c \) in the shift table, and align the pattern to that position.
  - Repeat step 3 until the end of string (or a match is found).

#### 2.5.3 Boyer-Moore

- The B-M algorithm takes a _backward_ approach: the target string is aligned with the start of the check string, and the last character of the target string is checked against the corresponding character in the check string [7].
- In the case of a match, then the second-to-last character of the target string is compared to the corresponding check string character. (No gain in efficiency over brute-force method)
- In the case of a mismatch, the algorithm computes a new alignment for the target string based on the mismatch. This is where the algorithm gains considerable efficiency.

#### 2.5.4 Comparisons of Brute Force, Horspool, Boyer-Moore

![Fig 2.5 Comparisons of Brute Force, Horspool, Boyer-Moore](image)

### 3. DESIGN

#### 3.1 Theoretical background

The design is considered to be one of the most important phases of software development. The first phase is the development of a suffix tree, data structure for storing the documents of different format such as Portable Document Format, MS Format files and Text files. In constructed suffix tree, the edges are stored in hash table. Hashing is a very efficient way to store and retrieve data [9].

The second phase is the development of the Phrase Based Document Retrieval. For the input, phrase either in order or without order is taken. And the last phase is the development of the proper user interface to assist the user in using the Phrase Based Document retrieval application [10].

#### 3.2 Problem Formulation

The focus of work is to combine the advantages of two methodologies suffix tree and Boyer-Moore in document retrieval. As a result the Phrase Based Document retrieval Algorithm represent each document as suffix trees, where phrases of the documents are stored as suffixes of tree in hash table for efficient storing and retrieval and Boyer Moore algorithm is used to check the presence of pattern i.e. the input phrase in order and with out order.

#### 3.3 Phase 1: Algorithm for Constructing The Suffix Tree

Suffix Tree is constructed for the document by using E.Ukkonen, “On-line construction of suffix trees” and enhanced to construct the suffix tree for document of different types. The constructed suffix tree store the edges
in a hash map, using a hash key based on their starting node number and the first character of the sentences. This implementation competently handles huge amount of data. A simple suffix tree for File1.pdf [8] is shown in figure 3.1.

![Suffix Tree Diagram]

Figure 3.1 Example of a Suffix Tree with Sentences

**Algorithm for Constructing the Suffix Tree**

1. Read the Document.
2. Tokenize the Document into sentences.
3. Start constructing the suffix tree for each sentence.
   i. For Every sentence, add suffix is done to add the suffix to construct suffix tree.
   ii. In add_suffix
       If the node is explicit node
       Then the suffix are added into EDGE_KEY
       Else if the node is implicit node
       Then
       The suffix are added into EDGE_KEY class
       And the split edge is done.
   iii. A node is created.
   iv. The details of the suffix are added into edge class.
   v. To add the suffix an edge is inserted.
   vi. Canonize is done to move to next smaller suffix.
4. The constructed suffix is displayed.

3.4 Phase 2: Algorithm for Retrieving the Documents with input phrase in order.

On the constructed suffix, Boyer Moore algorithm is applied to check the presence of pattern i.e. the input phrase in order. **The Boyer-Moore string search algorithm** (match a pattern of length \(n\) in a text of length \(m\))

i. Calculate the length of pattern and Text.
ii. Preprocess the pattern for the right-to-left-scan and bad-character-shift rules by finding the right-most positions of all characters in the pattern.
iii. Align \(p\) and \(t\), starting on index and shift \(p\) to the left, until we reach the end of \(t\).
iv. Scan the pattern from right to left, comparing the aligned characters at the current position in the text \(x\) and at the current position in the pattern \(y\).
v. If the pattern is longer than the text, we have no match.
vi. In the case of a mismatch, we do the shifting
   - Retrieve the right-most index of the mismatching text-character in the pattern.
   - If the mismatching character in the text is not in the pattern, shift until we are aligned behind the mismatch-position.
   - Else we shift the pattern to the right until the right-most occurrence of \(x\) in the pattern is under the mismatch position in the text.
   - If the characters are equal and the pattern has been scanned completely from right to left, we have a match. We store the match and shift the pattern one position to the right.

**Proposed Algorithm Combines Suffix Tree and Boyer-Moore for Documents with input phrase in order**

1. The suffix tree constructed for every document is obtained.
2. The suffixes are sorted ignoring case.
3. Then The Boyer-Moore string search algorithm is applied.
4. Input phrase is defined as pattern and stored suffix as text.
5. Then The Boyer-Moore string search algorithm is applied.

The general architecture of Phrase Based Document retrieval retrieving input phrase in order is shown in figure 3.2. The general description of the algorithm is, it takes input as set of documents in the form of Portable Document Format, MS Format files and Text files and input phrase and implements suffix tree algorithm which represent document as suffix and then Boyer-Moore algorithm is applied and output is obtained as the set of documents which contains the input phrase in order.
The general architecture of Phrase Based Document retrieval retrieving input phrase in order is shown in Figure 3.3. The general description of the algorithm is, it takes input as set of documents in the form of Portable Document Format, MS Format files and Text files and input phrase and implements suffix tree algorithm which represent document as suffix, and then Boyer-Moore algorithm is applied as shown in algorithm for retrieving the documents with input phrase in order. And then following steps are also implemented to search input phrase present in any order.

**Algorithm for Retrieving the Documents with input phrase without order**

1. The suffix tree constructed for every document is obtained.
2. The suffixes are sorted ignoring case.
3. The input phrase is tokenized into words and stored.
4. Each word in the input is defined as pattern individually and stored suffix as text.
5. Then the Boyer-Moore string search algorithm is applied as shown in algorithm for retrieving the Documents with input phrase in order. And then following steps are also implemented to search input phrase present in any order.

\[ \text{Begin} \]
\begin{itemize}
    \item Check for match and store the match.
    \item Check if input phrase is equal to store match with same suffix.
    \item If so then store the name of the document after checking for the presence of any duplication of document name.
\end{itemize}
\[ \text{Else} \]
\begin{itemize}
    \item Check in next suffix
\end{itemize}
\[ \text{End.} \]

3.5 Document Collection
Input is set of documents in form PDF files, MS Format Files such as MS Word, MS Excel, MS PowerPoint and Text File.

4. TESTING AND EXPERIMENTAL RESULTS
The development and deployment of the Phrase Based Document retrieval is done in Eclipse 3.4.2 mix of Java 1.4 and Java 5 VMs[11][12]. For reading Portable Document Format PDF Box java API and for reading MS Format files such as MS Word, MS Excel, MS PowerPoint Apache POI java API is used.

Comparative study of Phrase Based Document retrieval Algorithm and previous algorithm Suffix tree and Boyer-Moore Algorithm.

Previous algorithm Suffix tree construct suffix tree for strings and numbers of nodes created are more as suffixes are constructed for strings and Boyer-Moore Algorithm search pattern inside a text where as Phrase Based Document retrieval Algorithm Combines advantages of both previous algorithms Suffix Tree, Boyer-Moore and enhanced to represent different types of documents, searching is done in documents and retrieve them based on input phrase in order and without order. Numbers of nodes created are less as suffixes are constructed for sentences in documents.

4.1 Performance Metrics
The performance of Phrase Based Document retrieval Algorithm is evaluated using performance coefficient such as

- Number and types of Files available for retrieval.
- Size of Files available for retrieval.
- Time taken to retrieve the similar documents.
The graphical representation of the performance of Phrase Based Document retrieval Algorithm is shown in figure 4.1 and 4.2.

![Figure 4.1 Time taken to retrieve documents in minutes with respect to Number of Documents](image1)

![Figure 4.2 Size of Documents in MB with respect to Number of Documents](image2)

5. CONCLUSION AND FUTURE WORK

The project was about retrieving similar documents which has input phrase in order and without order by combining two methodologies, the suffix tree and Boyer-Moore algorithm and enhanced to retrieve the documents of different form such as Portable Document Format, MS Format files and Text files. The approach was an experiment to observe the proposed methodology’s ability to improve the retrieval, which partially succeeded with the given samples. Thorough testing and further improvements are required which are given below.

5.1 Limitations of the work

The problems observed were difficulty to retrieve the documents when input phrase is huge. The time taken to retrieve the documents also grows in direct proportion with size of documents.

5.2 Future Enhancements

The size of suffix tree is growing in direct proportion to size of the documents. And the time taken to retrieve the documents also grows in direct proportion. To provide more efficient way of retrieval suffix tree can be converted into suffix arrays. And some page ranking algorithm can be applied to rank the retrieved documents.

REFERENCES


About The Author

B.Ganga native of Tamilnadu. I have received my B.E degree in computer science and engineering from Karunya Institute of Technology, Tamilnadu and M.Tech in software Engineering from M.S.Ramaiah Institute of Technology, Bangalore. I have seven years of lecturing experience in engineering college and presently doing research in data mining. My area of interest helping human kind through technology.