**TABLE OF CONTENTS**

1. Introduction

2. Objective

3 .Methodlogy

4. Scope

5. Issue And Challenges.

6. Problem Statement.

**Introduction**

Information Retrieval:

Information retrieval is fast becoming the dominant form of information access. The Information Recovery Program is a part of any communication system. It is part of the Information Science, which studies the activities related to retrieval of information. The aim of information retrieval is to provide the right information, in the hands of a right user(s) at the right time. Retrieving information means to find some information resources that are relevant to some required information from a larger database of all kinds of information (relevant as well as irrelevant). This search can be based on metadata or on complete text. We can describe this process as:

http://www.lisbdnet.com/wp-content/uploads/2015/04/Capture.jpg

All kinds of structured as well as unstructured data (which does not have a clear and easy-to-read structure for computer like random data stored in a disk with no classifications) and information problems beyond the above specified definition in its core may also be covered in information retrieving. Unstructured data is just the opposite of structured data, for example a relational databases used by organizations to maintain their records of employees and products and their specific details. But actually no data can be regarded as completely unstructured, especially if all the text data is in a linguistic structure of human languages. Even the text documents (considered to be unstructured) are somewhat structured by headings and paragraphs or by highlighting or explicitly marking them in bold/ underline and more such techniques like using bullets, points or inserting code in between.IR can also be used for analyzing “semi-structured” searches like searching a document where the title has HTML and the body has ‘table’ in their contents, so both values are structured in terms of title and rest of the body but still unstructured as all data lies in the form of plain text. Information retrieval program also includes support to users to browse or filter document collection(s) or even process a set of already retrieved documents.

Clustering is similar to arrangement of books in a bookshelf according to some specific topics. In a large set of documents and files, it means to come up with a good grouping (formation of clusters) of these documents according to the topic(s) and their contents. According to a given set of topics or information requirements, or categorical needs (such as relevancy of a product to someone who has interest in some other product), classification decides in which classes, every set of documents should be grouped into. An approach to classify them is to manually classify some documents and then let the program automatically classify all the new documents that join in. Information retrieval systems can be distinguished on the basis of scales at which they operate. Mainly three scales are useful for distinguishing, these are discussed below:

1. *Web search*: the system searches a massive amount (billions) of documents stored in all (millions) the accessible computer machines across the world. Issues faced here are need to gather document files (for indexing), low efficiency of present systems to work at such a huge scale, handling the exploitation of hypertext and dodging the tricky manipulation of data and document content (attempt to fool the search engine) by site providers in order to trend their site in search engines, and use the commercial importance of the internet in their favor.
2. On the other hand there is *personal information retrieval*. Nowadays, consumer operating-systems have a built in information retrieval system to personalize data for the user. IR for emails management system provides searching feature and also text classification: spam detection/ junk mails, and automatically classify each mail so that they can directly be put into respective classified directories. Distinctive issues faced here are managing the various kinds of documents on a typical PC (personal computer), building a search system that is light-weight (takes less space and time) to reduce loads on owners’ PC and does not require any kinds of maintenance.
3. Then there is *enterprise, institutional, and domain-specific search* lying between the above discussed points. Here, retrieval may be provided for collections such as a company’s confidential documents, patents, or research articles on various topics. In such a case, these documents are to be stored on centralized file management system(s) which is implemented on some machines dedicated to provide search for this collection.
4. **MAJOR I.R MODEL :**

Mainly the following models are used to retrieve information: **Boolean retrieval** model (BRM), **Statistical** **retrieval** model (includes probabilistic retrieval model the vector space retrieval model) and the **Linguistic and knowledge-based retrieval** models.

* Statistical Model :-

The *probabilistic* and *vector space* models are the two good examples of this statistics-based approach. These models use information from stats like the term frequencies and check the relevancy of a document according to a given condition or query. Though both of them differently use the term ‘frequency’, but both produce a ranked list of documents according to their relevancy with the query. Although, these statistical-retrieval models solve some limitations present in boolean methods, but they too have some problems associated with them:

* Boolean model :-

The BIR (**Boolean model of information retrieval**) methods are a classical way of information retrieval methods but still the first and most adopted one. Today these are used by many IR programs because of their core advantages.

* Linguistic and knowledge-based :-

In its simplest form, user enters the keywords that are needed to be searched; these keywords are searched in the inverted indexes of each document’s keywords. In other words, this approach is based solely on the concept whether the exact word (specified in the searching query) is present or absent in the document. If found, it collects this document in its result; else it ignores this document and checks another. Clearly, due to its inability to understand the true meaning of word(s) in the query, this approach may not consider some relevant documents to be a good output and ultimately fail. This problem is well handled and tried to be completely removed by a smarter approach in *Boolean* *and Statistical information retrieval methods*.

But still, methods like morphological, syntactic and semantic analysis [Lancaster and Warner 1993] can also work to defeat this problem and retrieve information much more effectively after being integrated into Linguistic and knowledge-based approaches.

In a morphological analysis, presence of parts of speech (nouns, verbs, adverbs, adjectives etc.) in the query are analyzed that help to identify root and affixes of word(s), then syntactic analysis parses complete phrases. Based on some semantic relationships between words, the linguistic methods finally resolve the words’ ambiguities and produce possible synonyms or quasi-synonyms. Developing a good and sophisticated retrieval system based on linguistic features of the language is difficult as it needs knowledge of complex structures and basics of semantic information and retrieval implementation on the language. Therefore, this approach needs a much more advanced techniques such as AI (Artificial Intelligence) or expert system techniques.

1.**2 Boolean Reterival Model :-**

The term “information retrieval***”*** may have a very broad meaning. But, to define it we can say, Information ***retrieval*** (IR) is searching material (of various unstructured features) from large collections, to fulfill an information need. For query processing, there can be two possible outcomes– TRUE and FALSE – “exactly-matched” retrieval is the simplest form of ranking. Boolean Operators (specified in query) like AND, OR, NOT or proximity operators are used for matching the relevant documents.

**Advantages –**

1. Gives predictable outputs.

2. Relatively low complexity, and hence easier to understand.

3. Can be integrated with various other features.

4. Effective process management as many documents are eliminated accordingly.

**Disadvantages –**

1. Dependency on user for efficiency.

2. Simple query input is usually not very efficient.

3. Complex queries cannot be handled easily.

**1.2.1 Narrowing and Broadening Techniques:-**

We can describe a Boolean query on the basis of 4 terms: coordination type and its degree, its proximity, degree of stemming and field specifications in terms of word or string specifications. To formulate or reformulate a query, informed choices should be made by the user, according to these 4 dimensions, for this query to be sufficiently broad or narrow as per requirement of the information. Usually, broadening techniques raise the recall but lower the precision while narrowing techniques raise the precision but lower the recall. Ideally, all queries can be reformed and achieve the desirable precision and recall, but usually it is not easy to achieve both simultaneously. All the four terms in the query (re)formulation have some particular operators, out of which some tend to give the narrowing effect while others give broadening effect. For each operator with one type (narrow/broad) of effect, there can be one or more inverse operators which give the opposite effect [Marcus 1991].

**1.2.2** **SMART BOOLEAN:-**

Traditional Boolean retrieval techniques give some disadvantages (as mentioned above), to cover up those we discovered a newer method, called Smart Boolean (Marcus [1991, 1994]). This method helps the users in constructing and modifying a Boolean query and make better choices through the four dimensions of query’s character. It explains some of the best possible ways to convert traditional Boolean retrieval methods into a much more user-friendly and effective technique for information retrieval.

**1.2.3 Extended Boolean Models:-**

Various methods have been (and are being) developed for extending the traditional Boolean model to terminate some issues like:

1) The Boolean operators can be very rigid (strictly according to the rule) without an extended development that can soften them.

2) There is no provision for ranking the results in standard Boolean model (unlike statistical frequency methods).

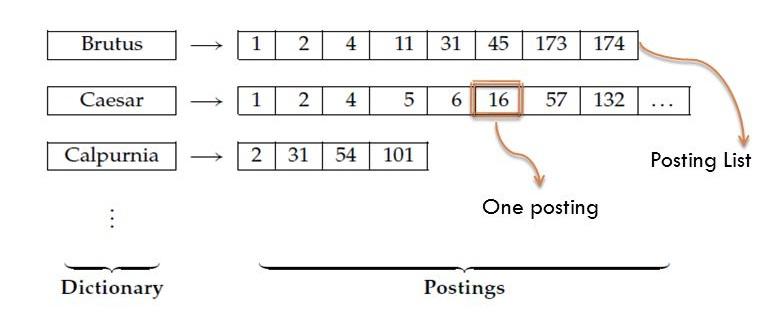
Extended methods provide users the relevancy documents with the query so as to provide a basis for ranking of results.

* 1. **INVERTED INDEXING :**

Now, if we consider a scenario to familiarize ourselves with the dimensions of various kinds of problems that we might need to face in realistic situations. Let’s say, we have 1 million amount of documents containing all kinds of data and each of them is approx. 1000 words long. Here, *data* can be whatever units about which we are making a retrieval system (for example individual memos), Data collection is the group of huge amount of data on which we have to apply our retrieval system (sometimes referred as *corpus*). Let’s assume that it takes 6 bytes per word (plus space and marks) on an average then the size of this data collection is more than 6 GB. Possibly there can be about 500,000 different terms present in these documents. These numbers are common while handling real problems and might vary in magnitude when we implement the system in reality.

The inverted *index* is the first major idea for information retrieval. Actually this term (*inverted index*) is redundant: the parts of document are always indexed backwards from the term. But still, in IR, ‘*inverted index’* (also known as ‘inverted file’) has evolved as a standard term. See the figure (a) below for the basic idea of inverted indexing.

First, a dictionary containing some terms is maintained (also known as *vocabulary*). Now, for each key in the dictionary, a list of documents in which this term occurs is maintained. Conventionally, a *posting* refers to each item present in this list that holds that the term has occurred in this document (and, later, often, the positions in the document). This list is called a *postings list* (also known as *inverted list*), and the postings lists, altogether are known as *postings*. Now, each list is sorted by their document ID while the whole dictionary containing terms can be sorted alphabetically.





* 1. **Increasing Performance Of Boolean Retrieval Model Using Data Parallelism Technique :-**

Information retrieval (IR) is to identify documents of non uniform behavior that fulfill information requirement from the huge repository (maintained in computer systems). Different models have been defined to retrieve/fetch information. For example **Boolean** model, the **Statistical** model, this focuses the vector space, probabilistic and the Linguistic and Knowledge-based retrieval models. The Boolean model is defined as the "perfect match" model. If the queries are not accurate, they retrieve/fetch some irrelevant documents. This is called the *precision (p) rate*, which is the proportion of the relevant retrieved documents. The Boolean method provides good techniques to elaborate or concise a query. The Boolean method works good for the search process, because of the clarity and exactness with which relationships between concepts can be represented. The *Boolean retrieval* model process the queries in which terms of the queries are in the form of Boolean expressions, that is, in which terms of the user query combined with AND(&), OR(||), and NOT(!) operators. The model views documents in the form of inverted indexes. The key concept of an inverted index is to maintain a *dictionary* of terms. For every term, a list of documents in which the term occurs. Each item in the list – which records that a particular term present in a specific document – is called a *posting*. The list known as *postings list* (or inverted list), and all the postings lists collectively called as *postings*.

But as the number of documents are increased the postings of documents are also increased and processing these documents becomes time consuming, so, to resolve this problem a multithreaded model is proposed in which the postings list is break down into different chunks and processes parallel by this performing Boolean operations between postings in accordance with Boolean query becomes faster. Using this data parallelism technique **1.5** performance of Boolean Retrieval Model increased.

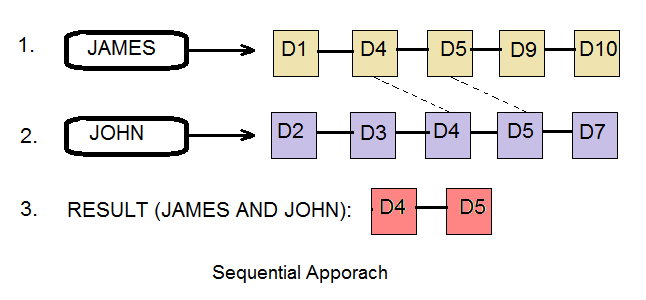
* 1. **Issue And Challenges Of BRM :-**

It is effective to use Boolean Retrieval Model for information retrieval, as the Boolean operation is performed between the posting lists. But these things become much more complex when the size of the posting list increase, it takes much longer time for performing the Boolean operation between the postings. So, there is a need for an enhanced BRM that can reduce the time complexity of Boolean operation between the posting lists.

“**In This Paper We Suggest A Multithreaded Data Parallelism Model To Reduce The Complexity Of Boolean Operation B/W The Posting List**.”

1. **Working of the proposed Boolean Retrieval Model for IR:**

**2.1. Sequential execution of this model:**

Here the search is implemented sequentially (word by word). In the picture below, the word ‘JAMES’ is searched first, then it searches the word ‘JOHN’ among the documents named as D1,D2,D3,D4,D5,D6,D7,D8,D9,D10 and stores the names of those documents which have the exact word present in them in the respective posting list. Then in the third step, it finds out the intersection of JAMES’s posting list and JOHN’s posting list. Here, D4 and D5 are the common postings so, it returns then the list of documents which contain both the words, JAMES and JOHN. 

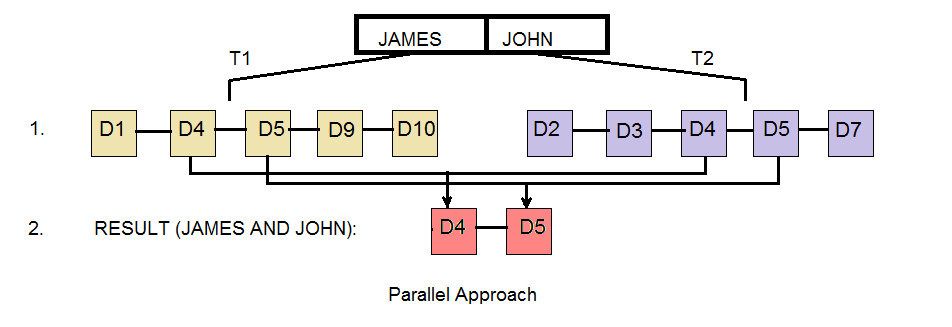
|  |
| --- |
| **2.1.1. Module – 1. Storing Files**  **Theory: -**  Storing files module is the first part of both sequential as well as the parallel sequential model. Purpose of this module is to first find out the directory whose path is given and where all the documents files lie at. Then it one by one picks up all the files and puts their content to a Tree-Map (data structure) and also it does the same for the search file where all the words (query’s data) to be searched are present. By this means, it speeds up the searching logic and the controller never has to go back to the secondary memory again and again for every document to be loaded to primary memory for searching.  **Algorithm:-**  1. Initialize a fixed variable FOLDER\_PATH storing directory’s path.  2. Paths.get (FOLDER\_PATH): loads up the path of directory.  3. Initialize a Tree-Map, say "linesOfFiles": to store files’ names and their content.  4. Initialize a Tree-Map, say "freqOfWords": to store searched words corresponding to files.  5. Initialize a Dictionary named "word": to store frequency of each word that is to be searched.  6. Initialize an ArrayList named "fileNames" to store names of files in directory.  7. For each directory Stream in Files.newDirectoryStream(Paths.get(FOLDER\_PATH))  A. Add files path into "fileNames" as its name.  (end of for loop)  8. For file in "fileNames"  A. Initializing the String List named "lines".  B. lines = Files.readAllLines(folderPath.resolve(file))  (end of for loop)  9. Initializing an ArrayList of String type named "st" to store searching words.  10. Initializing a FileReader for "SEARCH\_FILE\_PATH".  11. For each line of above file  A. Put word in "st" ArrayList  (end of loop)    **2.1.2. Module – 2. Data Pre-processing**  **Theory: -**  Symbol Remover and Queuing algorithm comes second in sequential model, right after the storing files. It's work is also very straight forward just to reduce down the time taken for searching out the result, what all this model does is that, it removes off all the common grammatical words like are, is, the and so on, along with all the punctuations which would make our task more difficult to find out our desired word. It also eliminates out the next line characters i.e. new line characters and replaces them by a space.  **Algorithm: -**  1. For each file in "linesOfFiles".  A. Initializing a dictionary "word" to store searched words along its frequency.  B. For each word 'temp' in "st"  i. Initializing d = temp as LowerCase.  ii. Initializing counter i to 0.  iii. For line as 'line' in "lines".  a. line = " " + line (Add space in beginning of each line).  b. line = line as LowerCase.  c. For each character c in 'line'.  I. If c is other than alphabet or number,replace it with " "(space)  (end of for loop)  d. For each line containing d.  I. Increase counter i by 1.  (end of for loop)  (end of loop)  iv. Put searched word "d" along its frequency "i" to "word".  (end of loop)  C. Put file name "fileName" along resulted dictionary "word" to "freqOfWords".  (end of loop)  **2.1.3. Module – 3. Creation of Indexes and Posting lists (String w)**  **Theory: -**  Creation of Indexes means finding out the result throughout the directory along with their frequencies. And posting list is the storing of each result as a recorded list with each file along with filename. Basically, what all is taking place here in this model is, it first retrieve throw content of all the files and read the content word by word and matches them out with the required word if it is matched up, frequency variable has been upgraded and this process continues till every file is being retrieved out from TreeMap, then stores up the result in the form of dictionary attached with the respective file name as tag.    **Algorithm: -**  1. Initializing an Integer array of size to that of Dictionary "word".  2. Fill the array with 0.  3. Initialize counter ‘i’ to 0.  4. For each word "t" in "st".  A. Initialize d to t in lower case.  B. For each file "fileNames" in "linesOfFiles".  i. Initialize counter c to 0.  ii. For each line 'line' in "fileNames"  a. Add " "(space) on before and after the line.  b. Convert line to LowerCase.  c. For each 'line' contains "d"  I. Increase counter c by 1.  (end of loop)  (end of loop)  iii. If counter is greater than 0.  a. Increase value of countNode at index i by 1.  b. Putting the “fileNames” into ArrayList “InsertFile”.  (end of loop)  C. Increase counter i by 1.  D. Put word “t” and ArrayList “InsertFile” to Dictionary “InsertNode”.  (end of loop)  **2.1.4. Module - 4 Boolean Intersection**  **Theory:-**  Intersection itself tells us what is common among two or more things. So, this intersection module basically checks out the common nodes that entered query has, that is, it gives the name of the nodes (files) containing that particular query. This module uses up an ArrayList to store the resulted file names. It traverses through each entity presented in dictionary " InsertNode ", and then matches the entity's Node list with the previously stored one.  **Algorithm:-**   1. Creating a String arrayList “result” of size of Dictionary “word”. 2. Initializing “result” to first key’s value of “InsertNode”. 3. For each word value “p” in “InsertNode”. 4. Set “result” to the intersection of “result” and “p”.   (end of loop) |
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|

**2.2. Parallel Execution of BRM**

**Theory:-**

This Parallel Sequential Model for searching of Boolean Retrieval Model is basically works on the basic principles of Multi-threading. Here we, let's suppose have to search out any two literals, so what we do is, we create a new thread for each search. So, for two searches there are two threads for this. And these two threads will work out like they both start at same time let us suppose t1, and ends up approximately together, depending upon their search through directory. In this way, rather searching them one by one, we found the result for both only at the cost of one's execution time.

Consider the picture below, there are two words, ‘JAMES’ and ‘JOHN’ to be searched along the entire range of documents named D1,D2,D3,D4,D5,D6,D7,D8,D9,D10. For achieving actual parallelism we split the searching into 2 threads T1 and T2 through multi-threading. T1 will handle the process of searching ‘JAMES’ while T2 will do the same for ‘JOHN’. After searching is done and we have documents with occurring keywords in their respective posting lists the second step is to find the intersection of these two posting lists of JAMES and JOHN similar to the sequential approach. Documents with both JAMES and JOHN are D4 and D5. See how this method only took 2 steps instead of 3, which is the main reason of reduction in net time taken by the approach.

****

**Algorithm:-**

A1. Creating two Threads for executing two word's searching in parallel

1. Retrieve all files names from folderPath to FileNames.

2. Put the contents of each file having their names in FileNames into linesofFiles.

3. Read the file containing words to be searched and store content in searchWords.

4. For each entity in linesofPath (filename, lines )

a. For each item in searchWords (search word)

i. Convert this word into lower case.

ii. Initialize a counter to 0.

iii. For each line in lines

A. Convert this line into lower case

B. For each character of this line: Check if character is other than alphabet or a number, replace it with a space (‘ ‘).

C. For this line check if this search word is present, if present, increase the counter by 1.

iv. Put word along with its counter as its value into wordDictonary.

b. Put filename and wordDictionary into freqofWords.

5. Set i=0

6. For each word in searchWord.

a. Convert word into lower case.

b. Assign the following code to an idle thread.

c. For each lines of LinesOfFiles (filename, lines).

i. Initialize counter to 0.

ii. For each line in lines

A. Convert line into lower Case

B. For this line check if this search word is present in lines

I. Increase the counter by 1.

II. Terminating the current loop

C. If Counter is greater than 0

I. Increase counterNode by one.

D. Generate the results of each word stored in searchWord along with its counterNode value.

E. Increase the counter I by 1.

7. Run the intersection module (discussed earlier in sequential approach).

1. **Result Analysis:**

**Table 4.1:** shows names and number of keywords of the document files used for testing this model. The maximum number of files used at once for testing is 30. ‘Keywords fetched’ refers to number of meaningful words (not the grammatical words) present in the document.

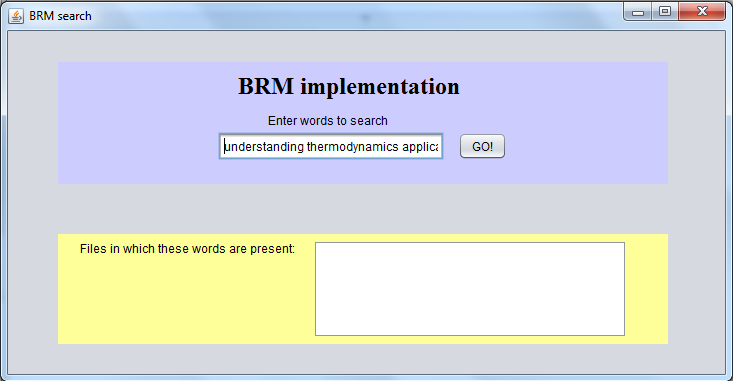
|  |  |
| --- | --- |
| **Document name:** | **Number of keywords fetched:** |
| Doc1.txt | 34 |
| Doc2.txt | 40 |
| Doc3.txt | 47 |
| Doc4.txt | 33 |
| Doc5.txt | 52 |
| Doc6.txt | 41 |
| Doc7.txt | 67 |
| Doc8.txt | 55 |
| Doc9.txt | 62 |
| Doc10.txt | 49 |
| phy\_unit1.txt | 4396 |
| phy\_unit2.txt | 3679 |
| phy\_unit3.txt | 2158 |
| phy\_unit4.txt | 4841 |
| phy\_unit5.txt | 5416 |
| phy\_unit6.txt | 5122 |
| phy\_unit7.txt | 4067 |
| phy\_unit8.txt | 3690 |
| phy\_unit9.txt | 2903 |
| phy\_unit10.txt | 2235 |
| chem1.txt | 5811 |
| chem2.txt | 4962 |
| chem3.txt | 4608 |
| chem4.txt | 4380 |
| chem5.txt | 3795 |
| test.txt | 153 |
| aml.txt | 1206 |
| index.txt | 114 |
| index copy.txt | 114 |

**Table 4.2:** Shows how the average posting lists’ size changes when number of documents and dictionary size vary. Numbers of documents used for testing differ from 30 to 20. Dictionary size means number of words searched in one query. Average posting list size refers to the average calculated by division of each posting list’s size (each word’s occurrences) and number of words searched in that query.

|  |  |  |
| --- | --- | --- |
| **No. of documents:** | **Dictionary size :** | **Average posting list size** |
| 30 | 5 | 7 |
| 30 | 8 | 5 |
| 30 | 4 | 8 |
| 25 | 4 | 7 |
| 20 | 4 | 5 |
| 20 | 3 | 8 |

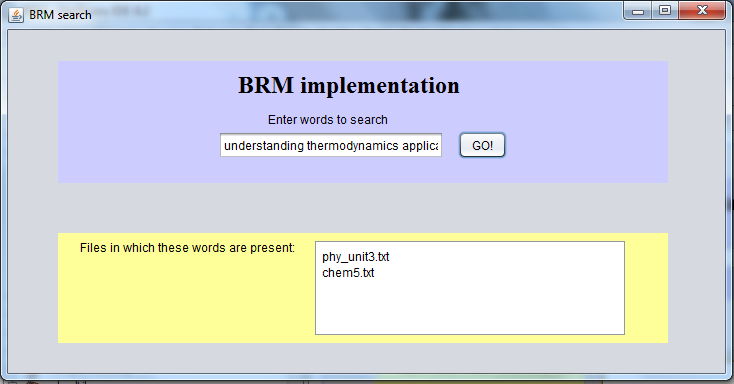
**Output screen shots of program:**

**Input: “**understanding thermodynamics applications”



On clicking the “GO!” button;

**Output:** “phy\_unit3.txt”, “chem5.txt”



**Table 4.3: Time taken for BRM in sequential execution**

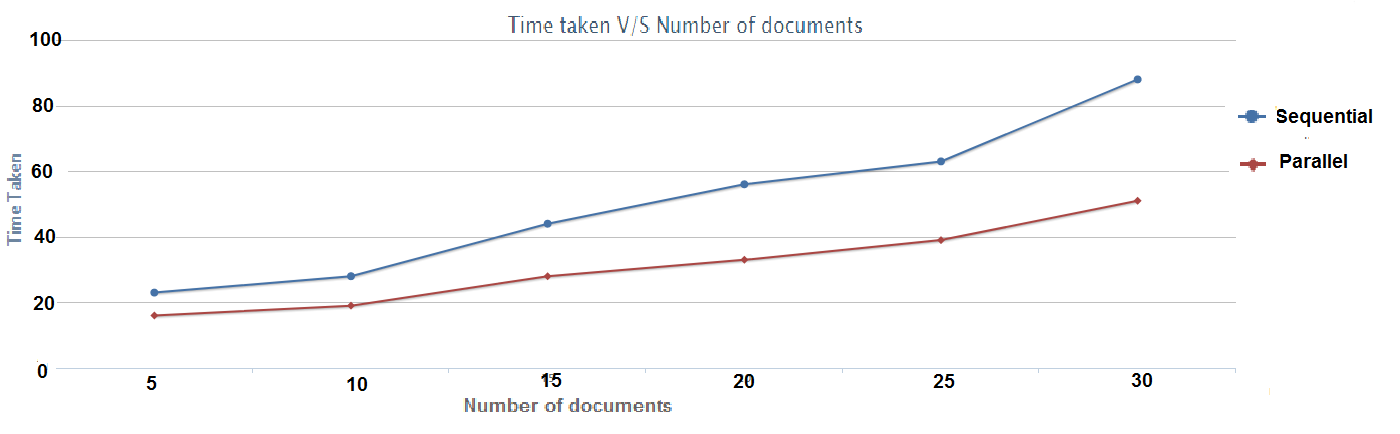
Boolean Query refers to the operation (AND here) and operands used for the particular test. Time taken gives the time elapsed while execution of the programs in milliseconds.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No. of documents** | **Dictionary size** | **Ave. posting list size** | **Boolean Query** | **Time taken(ms)** |
| 5 | 2 | 2 | work AND doctor | 23.45 |
| 10 | 2 | 4 | Man AND Woman | 28.94 |
| 15 | 2 | 4 | Classification AND acting | 44.23 |
| 20 | 2 | 6 | Hello AND create | 56.87 |
| 25 | 2 | 10 | Functions AND practice | 63.14 |
| 30 | 2 | 4 | Culture AND local | 88.42 |

**Table 4.4: Time taken for BRM in parallel execution via multi threading**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. of documents | Dictionary size | Ave. posting list size | Boolean Query | Time taken(ms) | No. of threads created |
| 5 | 2 | 2 | work AND doctor | 16.09 | 2 |
| 10 | 2 | 4 | Man AND Woman | 19.17 | 2 |
| 15 | 2 | 4 | Classification AND acting | 28.44 | 2 |
| 20 | 2 | 6 | Hello AND create | 33.91 | 2 |
| 25 | 2 | 10 | Functions AND practice | 39.55 | 2 |
| 30 | 2 | 4 | Culture AND local | 51.28 | 2 |

**Graph 4.1: shows how time taken differs in sequential and parallel approach**



**Analysis:**

Correct identified documents 🡪 C

Wrong identified documents 🡪 W

Non- identified documents 🡪 M

**Precision:**

Precision tells about the fraction of the retrieved documents that are actually relevant to the user's information need.

P = C/ (C + W)

**Recall:**

Recall tells about the fraction of the documents that are relevant to the query that are successfully retrieved.

R = C/ (C + M)

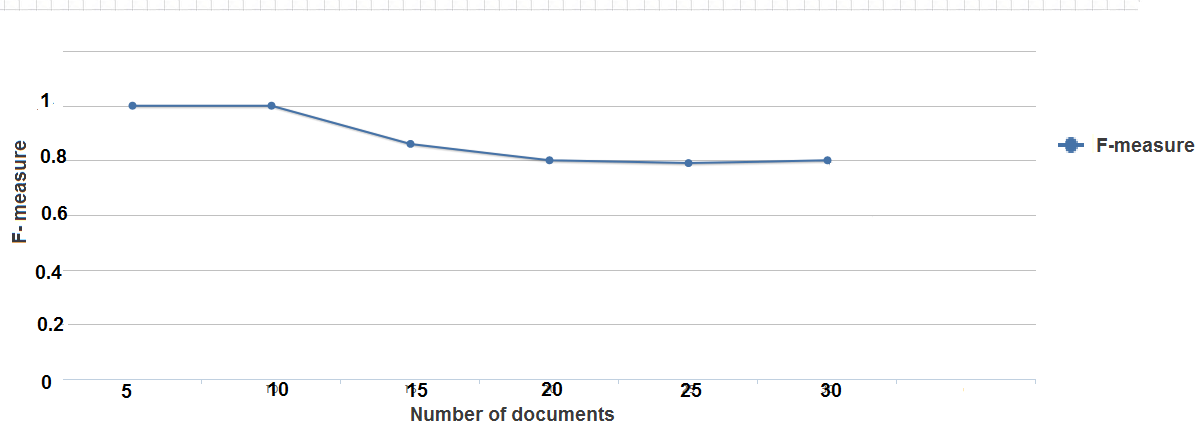
**F-measure:**

F-measure is the fraction that gives both precision and recall equally weighted. Its value varies from 0 to 1. F = 2 \* P \* R / (P + R)

**Table 4.5:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No. of documents** | **C** | **W** | **M** | **P** | **R** | **F** |
| 5 | 1 | 0 | 0 | 1 | 1 | 1 |
| 10 | 3 | 0 | 0 | 1 | 1 | 1 |
| 15 | 3 | 0 | 1 | 1 | 0.75 | 0.86 |
| 20 | 4 | 1 | 1 | 0.8 | 0.8 | 0.8 |
| 25 | 6 | 1 | 2 | 0.85 | 0.75 | 0.79 |
| 30 | 2 | 0 | 1 | 1 | 0.67 | 0.8 |

**Graph 4.2: shows F-measure as the number of documents increase**



**Conclusion:**

This enhanced Boolean Retrieval Model (BRM) focuses on retrieving information much faster to previously discussed methods. Proposed parallel execution of Boolean operations enables quicker calculation of results unlike sequential methods. In this paper, multithreading using java 8 is used but other methods like Hadoop’s MapReduce or any distributed processing systems which have multi-core processing architecture can implement the model on the basis of above discussed algorithm. Implementation of sequentially executable algorithm is much easier on general multi-purpose PC’s but due to unavailability of multiple CPU’s it is impossible to achieve actual parallelism unless there is an interconnected multiple processor architecture present within the system. But even if such multi-processing environment is unavailable, the model would still work in sequential mode but with reduced speed. This multiple environment support allows users independency over any type of structure, while its best performance (in terms of speed) can be seen on multi-core processing architecture. For example, in above algorithm, only 2 threads are used for processing each posting as the testing data was smaller in comparison to real life data, for huge data input, number of threads may be increased up to the number of processors available. Proportionally, the speed will vary with number of threads i.e. increasing number of threads will increase the speed (or decrease the time taken by each processor to complete the search).

Today, time is considered to be the most precious asset. So, the priority of the system is to reduce the time complexity to as low as possible over space conservation. Future Scope in this work may include compression techniques to reduce the memory used in execution of the program. The space required for each thread to search a word increases with the number of documents available for search, if documents are huge in number or heavy by size, each thread will take up large spaces to execute searching. In such cases, it may be needed to construct a technique that compresses or manages the present data in such a way that it takes minimum space on primary memory without exploiting the time complexity of the proposed system.

Applications of the proposed model (for information retrieval) can be search engines (web), recommendation systems, digital libraries, email/file management, social media searches, news retrieval etc. In web search engine this model can be used as a module which takes a preprocessed input (i.e. reduction of words which may have same meaning or adding words very relevant to the user input etc.) to reduce the ignorance to some relevant documents which have the word in other forms as it is a exact match based model (it searches the word for its exact match in documents). Spam detection in email management is another major possible application of this model as it will return all those documents which contain all spam recognized words, in this case, a Boolean model may reduce false accusations of spam on a genuine document hence increasing its performance than other methods. Another example of its use is news retrieval, for multiple agencies like news channels it is a daily task to study very old news for investigation and reports, this model will provide the exact search for such tasks even before user blinks again.