Research Journal of Applied Sciences, Engineering and Technology 13(12): 885-894, 2016
DOI:10.19026/rjaset.13.3761
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: September 23, 2016 Accepted: November 15, 2016 Published: December 15, 2016

**Research Article**

# Building Words Dictionary List Using Symbol Enumeration and Hashing Methodology

Safa S. Abdul-Jabbar and Dr. Loay E. George
Computer Science Department, College of Science, Baghdad University, Baghdad, Iraq

**Abstract:** This study aims to introduce a new method to reduce the time needed for text retrieval systems by building word dictionary takes the advantage of enumerating each string, multi hashing methodology stop-words extraction and word stemming; dictionary-based text mining has an important role in understanding and analyzing large text datasets that used in any searching, matching and information retrieval systems. All of these systems mainly imply dealing with strings (i.e., undefined number of alphabet characters of each word and an undefined number of words in a sentence) and text processing operation. This has a significant effect on the execution time for the systems due to the overhead hidden-operations (like, symbols matching calculations and character conversion operations). Some of the attained experimental results are provided for these operations with a comparison between the proposed method results and those belong to the traditional method; which directly deals with strings only. Results comparisons are provided for each step to understand the advantage of the proposed approach. The results demonstrate the effectiveness of the proposed approach that reduces the execution time for each step, which in turn leads to improve the overall execution time for the whole system while maintaining the accuracy of the operations.

**Keywords:** And stop-words, data editors, hashing methodology, string enumeration, string hashing, stemming, string matching operation, word dictionary

## INTRODUCTION

Dictionary based text mining has a significant role in enabling practitioners in understanding and analyzing large text datasets. In addition, the dictionary commonly used in information extraction, entity annotation, classification and link analysis tasks. Users vary among experts and ordinary ones and the linguistic rule-sets and pre-packaged dictionary-based components are preferred to be utilized in real world applications. These components are used in text processing operations for making the text easily accessible, simplify browsing and facilitate the process of understanding corpora (Godboles *et al*., 2010). There are many text mining tools that can be used for building a text mining dictionary (e.g., lexical analyzer, stop-words filtering, hash-indexing system, etc.). Many studies have investigated the effect of these tools in the text pre-processing systems.

Yao and Ze-Wen (2011) have used three different filter algorithms that designed and implemented to stop-word filtering. Then, they compared the efficiency of these algorithms according to the experiments that have done. The results indicated that the hash-filter method was the fastest method (Yao and Ze-Wen, 2011). Ayral and Yavuz (2011) have proposed an automated method that used to identify stop words in order to improve classification of natural language content. Popova *et al*. (2013) have proposed methods to automatic stop list feeding due to the scope of interdisciplinary methods applied. This method allows for improving the quality of key phrase extraction on the stage of the candidate key phrase building (Popova *et al*., 2013).

Willett (2006) had improved the original Porter stemming algorithm. Also, he provided an overview of its subsequent use (Willett, 2006). Joshi *et al*. (2016) proposed a modified version of the Porter stemmer to overcome some of the limitations of the old algorithm version and provide some features that made it more useful in information retrieval (Joshi *et al*., 2016).

Stein and Potthast (2007) have presented two developed hashes-based indexing approaches and compared the performance improvements in real-world retrieval systems (Stein and Potthast, 2007). Also, Singh *et al*. (2009) have used the Modified Word Searching Algorithm (MWSA) that matches the hash value of the same length for text T and pattern P. The experimental results showed the proposed MWSA algorithm is much faster than the WSA algorithm (Singh *et al*., 2009).

Dictionaries not only store the important words for the datasets, but also rules which are composed of some

**Corresponding Author:** Safa S. Abdul-Jabbar, Computer Science Department, College of Science, Baghdad University, Baghdad, Iraq

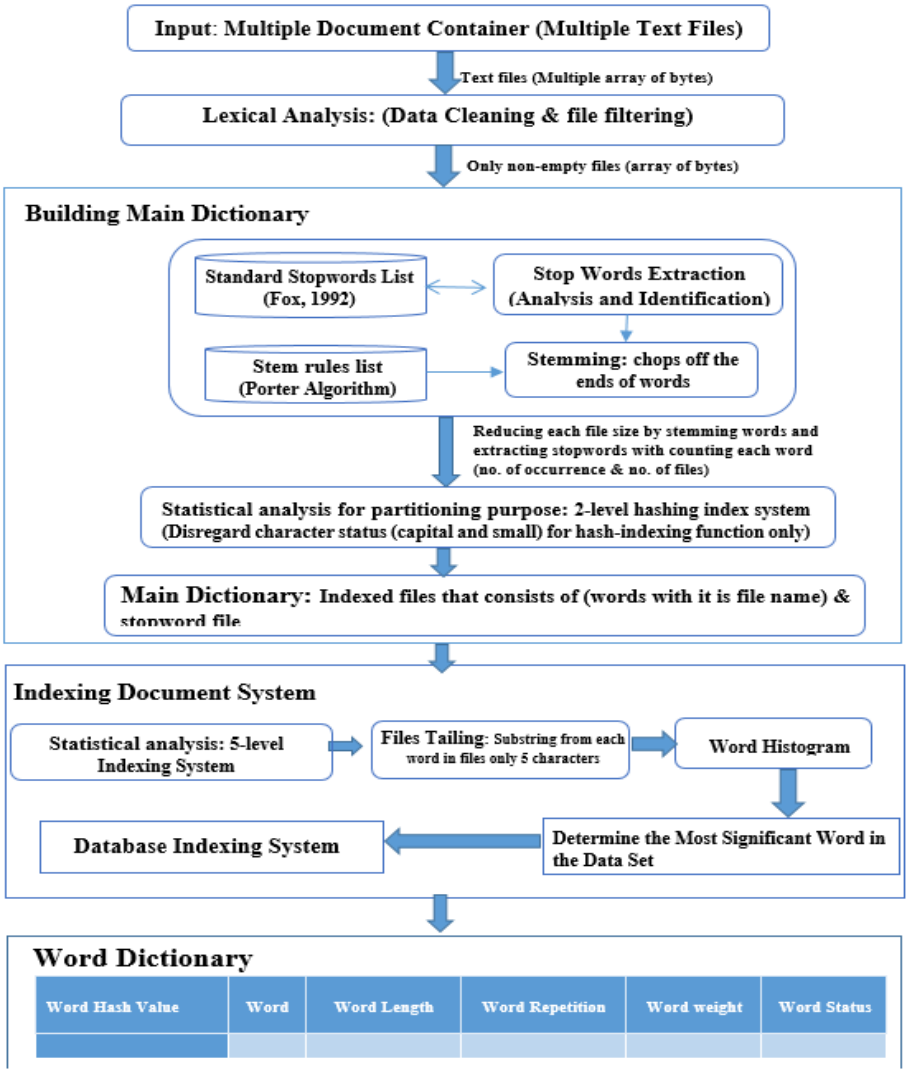


Fig. 1: Block diagram for overall system design

important words. Sakurai *et al*. (2001) have proposed a method that automatically builds the rules and their classes from the original text data by using an inductive learning method. The result showed that the fuzzy inductive learning algorithm is appropriate for the acquisition of the rules. In addition, this method acquires rules which provide higher accuracy through numerical experiments based on 10-fold cross validation and using daily business reports in retailing (Sakurai *et al*., 2001). The dynamic dictionary can be used for both Compression and data retrieval operations from large dataset. So, Bhadade and Trivedi (2011) proposed a pre-compression technique that can be applied to the original text files. The output of this technique could be utilized in the available standard compression techniques (e.g., BZIP2 and arithmetic coding) because the proposed method provides better compression ratio. The suggested algorithm used the dynamic dictionary that must be created at run-time. Also, it is suitable for retrieving the phrases from the compressed file (Bhadade and Trivedi, 2011).

The objective of this study is twofold: firstly, to build a dictionary to be utilized for analyzing a text with text mining mechanisms. This dictionary can be used to extract expressive concepts from documents in order to provide fast retrieval systems; secondly to speed up the text-mining techniques and reduce the memory and CPU consumption. Hence, a new method was proposed to handle the string operations as a sequence of numbers instead of a sequence of characters to reduce the hidden cost of the string operations.

## MATERIALS AND METHODS

The overall design of the proposed system is shown in Fig. 1; it consists of three stages:

- Lexical analysis
- Building main dictionary
- Indexing database system

Lexical Analysis stage is used to extract the useful data using enumeration operation for all input documents. While, the purpose of Building Main Dictionary is to extract a set of words, which can be used as attributes, from the original files to decrease the execution time of retrieval systems as a result of reducing the search space size. Finally, the document indexing stage was used to determine, automatically, the most and least significant words exist in any dataset to help the system in partitioning the search space to multiple nodes; this will be cause further reduce in the time that needs for doing any retrieval process.

**Lexical analysis stage:** Preprocessing methods play very substantial role in text mining system and its applications (Vijayarani *et al*., 2015). The first step in Lexical Analysis stage is the enumeration operation. This operation means changing the way in which the underlying data is stored for each variable. Whereas, the computer stores the string in the form of byte for each character (i.e., the number of bytes that used to represent a string is equivalent to the length of that string). While numbers are stored in a fixed length of bytes (such as 4 bytes for integer and 8 bytes for long), this will affect the storage required to store all documents in each dataset (Clapson, 2014).

Therefore, it is necessary to understand how each data type is stored in order to keep accuracy for string converting operations (Clapson, 2014).

To illustrate how to convert any string to numbers, there are two methods to perform this operation: The first one is simply using the ASCII value of each character, while the second method uses a coding system (for handling numbers, punctuation, all alphabet characters and other non-printable characters). This system involves performing a mapping operation for each character to a unique number which can be considered as an ID that uses to identify each character in the range [0-26] for 26 alphabet characters plus one number for non-printable characters and spaces.

The main goal for this operation is to reduce the hidden cost of the internal operations (calculations and mapping operations) that consuming both CPU time and memory space. It is important to mention that this step will be used in the next stages of the proposed system for speeding up the string operations.

The input data files are collected from various resources which may contain unwanted symbols. The second step in the lexical analysis stage consists of two operations: data cleaning and file filtering. The first operation used to extract words from the input character stream (i.e., Removing: ",", ".", "?", "&", etc.) with special processing for some cases such as "we're" → "we are" and "B.S." → "BS" and soon. The resulted files from this operation may or may not contained at a that can be used in the next stages. Hence, file filtering operation must be done on all resulted files; exclusively those contain data with collective meaning for using their content in the next stages while rejecting other files.

**Algorithm (1):** Lexical Analysis
**Objectives:** Extract useful data from the input stream as a sequence of tokens (words).
**Input:** Text files.
**Output:** Text files with only words.

**Step1:** Read Text files \\ For each Text file Read all the content of this file as an array of bytes.
**Step2:** Filtering undesired symbols.\\ For each word check each letter for handling some situations such as (we 're → we are, don't → do not, bi-cycle → bicycle, B.S. → BS and up/down → up down) and collect the words in buffer to speed up the process of word extracting.
**Step 3:** Copying to a buffer. \\Copy each letter to buffer (temporary array) which has a numeric name for each file (i.e., The first file → 0, the second file→ 1… and so on) to speed up the process of word extracting.
**Step 4:** Buffers Filtering.\\Passes only non-empty buffers.
**Step 5:** Print in files. \\Print the resulted buffers in files that have corresponding names of the buffers.
End;

The result of this stage is a group of files with different sizes because each contains a variable number of words with variable length for each word.

**Building main dictionary stage:** This stage implies the following two steps:

1. Reduce the search space step:
- Stop-word extraction operation.
- Stemming operation.
2. Statistical analysis step (2-level hashing index system).

The first step consists of two operations: The first operation used to extract all stop-words that appeared in the dataset because:

- The retrieval operations consume time that depends on the search space size.
- These words appeared in all documents. This will cause retrieval nomination of all documents. Hence, the system accuracy will be reduced either.
- These words are not discriminating information that the user need.

This operation was implemented using the pre-complied stop-words lists such as Van Rijsbergen

(1979) and Fox (1992) list (Van Rijsbergen, 1979). In this study the Fox (1992) list was used for extracting and computing the stop-words frequency and to give each one of these words weight values according to two bases:

- The number of documents contains the stop-word
- The word frequency in all documents.

The reason for using the pre-complied stop-word list beside to the use of automatic stop-word detection system in the next stage is many words such as (preposition, conjunction, interjection, etc.) do not have any relevant meaning even it has less or high frequency; it also spread in a large number of documents. Hence, it is better to remove these words at this stage to satisfy the best system performance.

**Algorithm (2): Stop words Removal Algorithm**
**Objectives:** Reduce the search space that leads to reduce the required time for overall execution using the new proposed method.
**Input:** Arbitrary number of text files.
**Output:** The stop-word file plus the same number of input text files with reduced-size.

**Step 1:** Determine stop-words list (Fox, 1992). //Define the pre-complied a list of stop-words (i.e., An array of bytes of two dimensions) that defined by Fox (1992).
**Step 2:** Read text files as blocks of bytes. //For each Text file read its content as blocks of bytes with size about 4 MB for each block till reaching the end of file.
D ←0 //the initial value to the array size that contain ordinary words
Set Size← 4000000 //the maximum size of reading block
A()← Reading block (Size)/// Array of bytes
**Step 3:** Check the taking block //To ensure that the last word in the taking block was completed
Set kk←size
While A(KK) ≠ 32 Do
kk=kk-1
End
**Step 4:** Count Length for each word. //Determine the start (s) and the end (e) of each word (array of bytes) for counting its length (g), to compare it with the largest word length in the pre-compiled list for excluding words that across the largest length (i.e., 11) to reduce the number of comparisons that required
if *g*< 11 then
Call Function1
else
For I = s to e step 1 do
B(d) =A(I): d = d+1
End For
read next word
end if.
**Step 6:** Print in files. //When reaching to the end of each file Print Array B in a file with a numeric name, d = 0, then go to step2.
**Step 7:** Create stop word file. //Print all stop-words with its repetition and no. of files that appeared on it.
End;
**Fuction1:** Check Words. //Check if the word is exactly matched one of the words that listed in "Stop-words"; if the matching result is true then increased the count of the similar word. Then check the file name if this word appears in this file for the first time, increase the count of files for this word by one. Else if the result is false:
For I = s to e step 1 do
B(d) = A(I): d = d+1
End For
Return

The result of this step is files corresponds to the original files with less size beside one additional file contains the stop-words list, its frequency and the number of files that appears in.

Ones the resulted files are obtained from the previous step it is appropriate to use the stemming operation on them in identifying the roots of all words. In this study, the Porter Stemming Algorithm used which is considered as the most popular stemming method prepared by Martin Porter in 1980. Porter Stemming Algorithm is a conflation Stemming Algorithm. It is based on the idea of the suffixes in the English language are mostly made up of a combination of smaller and simpler suffixes, so it is a suffix removal algorithm. It has six steps, in each step, certain rules are applied until one of them passes the conditions. If a rule is accepted, the suffix is removed accordingly and starts executing the next step. The first two steps deal with plurals and past participles, the next two steps are designed for words that contain double suffixes, such as:

FUZZIFICATION→FUZZIFI→FUZZY

Last two steps do the tidying up; i.e., Making it presentable (Jivani, 2011; Ramasubramanian and Ramya, 2013). Figure 2 presents a simple example for stemming process.

The limitation of this algorithm is the result stem words which are not always the original words. The algorithm implies sixty rules; for this reason, it is considered a time consuming algorithm (Jivani, 2011). In this study, the system used the porter algorithm with a difference in the nature of its input; the algorithm was converted in a simple way to deal with input numbers that represent strings (not the strings themselves).

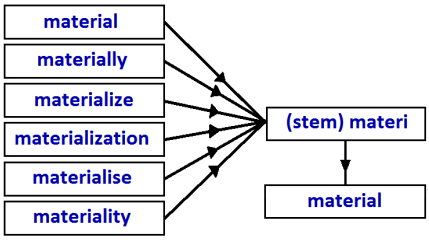


Fig. 2: Stemming process (Ramasubramanian and Ramya, 2013)

The result of this process consists of files corresponding to the original, but with less size due to the stop-words extraction and stemming operations. After reducing the search space, the system must be able to partition the search space for providing efficient access to information based on a key number. This can be done using a proper hashing algorithm because there is a strong evidence that ensures hash tables are much faster than others tree-structured indexes. It allows the system to look up the required information in constant time for access. Hashing is also simple to implement, safe to use an arbitrary method as a black box and expect good performance (Richter *et al*., 2015).

The basic idea of hashing resides in transforming the data points from the original search space into a Hamming space with binary hash codes. So, the storage cost for each transition will be reduced. Hence, the query speed can be improved (Zhang and Li, 2014).

Hash functions are different in their behavior in terms of collisions. This situation will appear when two keys trying to reach the same hash table location. There are two techniques that are mainly used to deal with collisions. One of them tries to handle the occurring conflicts, while the other technique is to completely avoid collisions. This will be achieved through using different techniques such as changing. Figure 3 presents the most important hashing data structures and surveys their behavior (Grill, 2014).

Thus, the next step in this stage is statistical analysis, which means performing the hashing index operation for all files using the first two letters of each word; they are used to define a hash function (H) value; which must be bounded within the range [0....N-1]. Where, N is the number of possibilities for the first character multiplied be the number of possibilities for the second character) minus 1.H(M) is an integer number within the range [0…N-1]; M is the string that must mapped to a unique number lay in the defined range of this function; H is a hashing index function.

In this step, the hash function was used to enumerate the first two letters of each word. The used equation of this function is:

$$H(M) = 27xch_1+ch_2 \quad (1)$$

where,
$ch_1$ = The first character of the word
$ch_2$ = The second character of the word

**Algorithm (3): Hashing Algorithm**
**Objectives:** Convert unstructured data files to structured data files using hash-index function of providing very fast access.
**Input:** Text files including stop-words file.

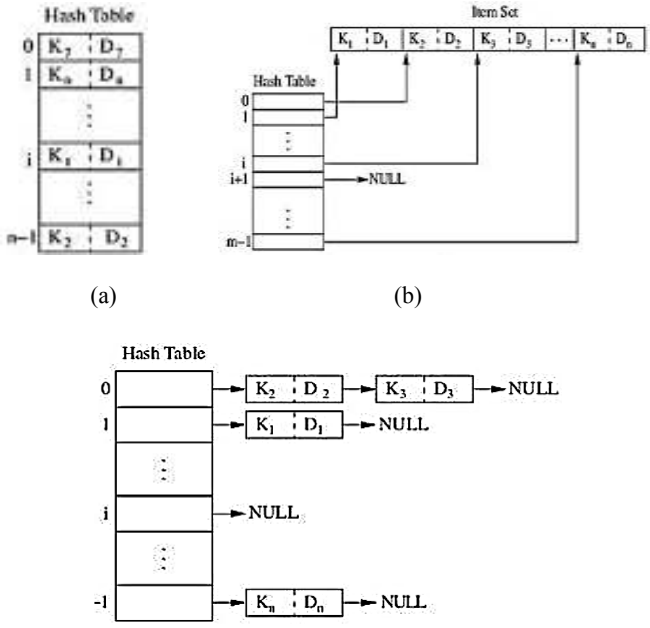


Fig. 3: (a): Example for minimal perfect hashing; (b): Open hashing techniques; (c): Chaining techniques (Botelho *et al*., 2011)

**Output:** Text files (729 file only) including stop words file.

**Step1:** Establish the Coding Table. //Building a coding system.
Tbl1() // Array of the integer numbers
For I = 0 to 255 step 1 do
Set Tbl1(I) ← 26 //an initial value for all printable and non-printable characters
End For
For I = 0 to 26 step 1 do // Establish the Coding Table
Set Tbl1(I + 65) ← I, Set Tbl1(I + 97) ← I
End For
For I = 0 to 26 do
For J = 0 to 26 do
tbl2(I, J)← 27 * I + J//filling the matching table for each pair of characters
End For
End For

**Step 2:** Read Text files. // For each Text file Read all the content of this file as an array of bytes.
**Step 3:** End Trim. //Remove spaces from the end of the file.
**Step 4:** Remove Double Space. // Convert double spaces to single space.
**Step 5:** Hash Indexing. // Mapping each pair of two characters with a unique value depending on matching table as follows:

$$H(M) = Tbl2(Tbl1(ch1), Tbl1(ch2))$$

**Step 6:** Mapping Words. //Mapping each word to 729 file that has the same name of the first two characters according to the $H(M)$ that calculated using the above equation.
**Step 7:** Mapping files. // Specify a file name that contains each word, convert it to an array of bytes, then combined it with each word in the resulted files.
**Step 8:** Print in files. //Print the results in files that have the names of the first two characters.
End;

The resulted files of the hashing function are organized according to the hashing value. This simple function *provides an* indexing technique such that the resulted values *can be used as* leading address for each data item. This allows for *fast access* for specific information with the possibility of using parallel searching for several patterns as one disconnected pattern.

At the end of this step, the first prototype of the main dictionary was completed by converting the unstructured text data into structured data. This makes the text retrieval operations and other text mining processes faster and easier to be used. In order to complete the proposed system for discovering most words properties, the next stage contains extracts other word properties.

**Indexing of database system:** This stage consists of four steps:

- Statistical analysis step (3-level hashing index system)
- File Tailing
- Word Histogram
- Determine the most frequent words in this dataset.

In the first two steps, the hash function was used for indexing and sorting all words in each file using the first five letters by taking the advantage of the first two letters which have taken from the previous stage and used a file name. Thus, in this stage the next three letters are used with respect to the index of each file, they were obtained from the last stage:

$$V_0 = 27^4 x fn_0 + 27^3 x fn_1 \quad (2)$$

$$H(M) = (27^2 x ch_3 + 27 x ch_4 + ch_5) + V_0 \quad (3)$$

Where, $fn_0$ is the value of the first character taken from the file name; $fn_1$ is the value of the second character taken from the file name; {cg3, ch4 & ch5} are the indexes of third, fourth and fifth characters taken from the word.

File tailing is used in the proposed system for selecting words with (1 to 5) characters and truncates the first five letters from the words that have length more than 5 letters with flag for each word. This indicates that whether the length of the word was larger than five letters or not. This step was used for two reasons:

- Reduce the storage space that required to store the dataset.
- Reduce the search time as possible by reducing the search space, taking in consideration the system efficiency.

The reason for limiting the word length to five letters for encoding purpose is due to the results of conduct analysis that was made to define the relation between the lengths of all words and their occurrence frequencies using three big datasets; it was found that the most frequent word lengths are ranging from 1 to 5; the total of occurrences of theses length constitutes around (%70.64) for dataset1, (%72.39) for dataset 2 and (%73.72) for dataset3, as shown the Fig. 4.

The frequency of each word length is calculated in order to:

- Determine the list of most frequent words (which have no discriminating weight for matching/ searching or retrieval systems); these words have a

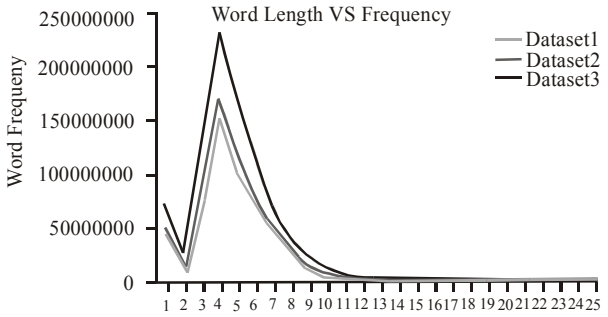


Fig. 4: Histogram summarizing the statistical analysis of words according to length

negative effect on the operations costs in terms of time and space beside to its bad effects on the overall system efficacy.

- Compute the weight for each word in the dataset according to its frequency and the number of documents holding this word.

In this study, the list of most frequent words was produced, taking the highest 20-word frequency for each file in the dataset. It is worth mentioning that word weights can be used to determine the least and most significant words in each dataset. This feature is very useful for any retrieval system.

**Algorithm (4): indexing document system**

**Objectives:** produce an indexing database file for calculating words weight and determine the non-significant word list.

**Input :** Text files with any number of files including stop word file.

**Output:** Three Text files only.

**Step 1 :** Establish the Coding Table. //Building a coding system.

Tbl1() // Array of the integer numbers

For I = 0 to 255 step 1 do

Set Tbl1(I) ← 26 //an initial value for all printable and non-printable characters

End For

For I = 0 to 26 step 1 do //Establish the Coding Table Set Tbl1(I + 65) ← I, Set Tbl1(I + 97) ← I

End For

Tbl2(,) //Array of integer numbers represent the matching table for the hashing function

For I = 0 to 26 step 1 do

For J = 0 to 26 step 1 do

tbl2(I, J)← ((27 * I) + J) //filling the matching //table for each pair of characters

End For

End For

**Step 2:** Read Text files.// For each Text file Read all the content of this file as an array of bytes except the stop word file that will not change.

**Step 3:** End Trim. //Remove spaces from the end of the file.

**Step 4:** Remove Double Space. //Convert double spaces to single space.

**Step 5:** Find v0. //For each indexed file using the name of each file such as

$$V0=27\text{^}4*ch1+27\text{^}3*ch2$$

**Step 6:** Determine hash value: //For each word in the file using v0 and the next three letters in each word.

$$H(M) =(27\text{^}2*ch3+27*ch4+ch5) +v0$$

**Step 7:** File Tailing. //Select the first five letters only from each word with flag for words with length>5.

**Step 8:** Word Histogram.//On the word hash value counting the frequency for each word.

**Step 9:** Counting files. //Counting the files that contain each word depending on the hash value of each word.

**Step 10:** Sorting words.//Depending on words frequency using the bubble sort algorithm.

**Step 11:** Determine the most frequent words.// Finding the 20 maximum repetitions of all words in each file and fill the list of the most frequent words.

**Step 12:** Print in files. //Print the original stop word file and the remaining results will show as follows (word, length flag, no. of files, word freq.) in two files.

- Most frequent word files
- Remaining words file //contain all words except most Freq. words.

End;

The results of this stage were registered in three files (i.e., Stop-words file, most frequent words file (common non-stop words file) and the remaining words file). The first file, which is the same file that extracted at the second stage, contains the pre-complied stop-words list with their frequency. The second file contains the most frequent words that appeared in the dataset with their frequency and the number of files holding that word. The last file contains all words that appeared in the dataset with the same information as in the most frequent words file for each word.

## DATASETS

In order to test the system performance, in this study work a set of tests applied on three big datasets were conducted. The datasets are:

Table 1: The time that took in each step for traditional and enumeration methods

| | Dateset1 | | Dateset2 | | Dateset3 | |
|---|---|---|---|---|---|---|
| Step | Enumeration method (minute) | Traditional method (minute) | Enumeration method (minute) | Traditional method (minute) | Enumeration method (minute) | Traditional method (minute) |
| Lexical analysis | 16.35 | 23.55 | 1 | 2.37 | 2.25 | 8.42 |
| Stop words | 79.17 | 516.21 | 10.23 | 51.01 | 44.29 | 230.24 |
| Extraction stemming | 20.04 | 1.18 for 1 MB | 1.25 | 1.21 for ≈ 1MB | 8.21 | 1.12 for≈1 MB |
| 2-level hashing index | 17.17 | 68.49 | 2.03 | 11.14 | 5.58 | 33.49 |
| Indexing database | 60.35 | 0.5 for 1 MB | 53.25 | 0.5 for 1 MB | 62.03 | 0.5 for 1 MB |

Table 2: The size for each dataset after and before systems stage

| | Dataset 1 size | | Dataset 2 size | | Dataset 3 size | |
|---|---|---|---|---|---|---|
| Stages | Before | After | Before | After | Before | After |
| Date preprocessing stage | 4.64GB | 4.27GB | 541MB | 527MB | 2.12GB | 2.01GB |
| Building main dictionary stage | 4.27GB | 3.55 GB | 527MB | 408 MB | 2.01GB | 1.86 GB |
| Indexing database system stage | 3.55 GB | 13.1MB | 408 MB | 3.82MB | 1.86 GB | 4.98MB |

Table 3: Net time profit percentage for each stage comparing with the traditional method

| | Stages | Net time profit percentage (%) |
|---|---|---|
| Dataset1 | Lexical analysis | 87.37 |
| | Stop-words Extraction | 99.29 |
| | 2-level Hashing index | 96.07 |
| Dataset2 | Lexical analysis | 84.70 |
| | Stop-words Extraction | 94.35 |
| | 2-level Hashing index | 95.42 |
| Dataset3 | Lexical analysis | 87.37 |
| | Stop-words Extraction | 80.6 |
| | 2-level Hashing index | 92.41 |

**Dataset 1:** Collected data from papers, books and articles with the possibility of recurrence of the file content and the size of files ranging within [1KB-20,374KB]. The overall size of all files are formed 4.26 GB.

**Dataset 2:** It is a standard dataset taken from the training and test text corpora by Burnard (1976). The size of files ranging within [1KB-6,250KB]. The whole files occupy 541MB.

**Dataset 3:** It is standard dataset obtains from Pizza and Chili (Ferragina and Navarro, 2005). The size of files ranging between [1KB-29.633KB). The files cover the size 2.10GB.

## RESULTS AND DISCUSSION

In this study, all algorithms were implemented using visual C# 2015 and operated on computer with Intel(R) Core(TM) i7-2600; CPU @ 2.60 GHz; 16 GB RAM; OS: Microsoft Windows 10.

Each step of the proposed system was tested through two experiments. The first one was made for testing the system performance using numeration operations rather than dealing with string operations.

The second experiment conducted by excluding the use of string operation so it works with the string as the most systems doing (traditional method).

Table 1 shows the comparison results between the two experiments (of three datasets). The results show the effectiveness of the proposed method when using an enumeration operation for dealing with string. In stemming step the same result attained as the original Porter stemming do, but with less runtime required.

Table 2 presents the size reduction at each stage in the dataset. This may consider as a preliminary file compression stage, which can be used to consolidate storage in both block storage and file system that contains only the most important information of the dataset.

The test results showed that the designed system will reduce the execution time around 80% for the overall system while maintaining the results accuracy. Also, we can notice that the baseline of the execution time will increase dramatically with the size and the matching operations that required in each execution for the traditional methods. So, it can be noticed that the larger elapsed time was in stop-words extraction stage for both experiments (traditional and numeric based methods); this due to the number of the matching operations that required to decide about whether reach word is a stop-word or not. The net profit time obtained in each text operation with excluding the writing and reading operation for all files are shown in Table 3. This table emphasis that the optimum saved time is occurring in the stop-words step due to speeding-up comparisons and converting operations.

The achieved speeds up results agree with the results given by other research articles (Cox, 2002).

## CONCLUSION

The proposed system provides fast and efficient instrument for string operations, which is based on the string conversion ability. The attained test results showed that the proposed system significantly reduces the required time of each operation; as shown in Table 1. Hence, this system will save up to 50% of the execution time with controlling the accuracy of the resulted files; this can save investments in both time and hardware.

Also, this study builds a text-mining dictionary using many text-mining tools, hashing functions and numeration operations in order to provide a fixable dictionary. This dictionary can be used for fast text retrieval systems because of its small size comparing with the original dataset, as shown in Table 2. Then, the third stage is added to the system to extract words characteristic of large-scale textual data, visualizing them with high performance and extracts the most and least significant word for each dataset.

For future work, more advance numeric analysis can be applied to the resulted dictionary files to improve the response time for retrieval systems. This can be done using a combination of two or more similarity measures with complementary weighting methods.

## REFERENCES


Ayral, H. and S. Yavuz, 2011. An automated domain specific stop word generation method for natural language text classification. Proceeding of the IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA). June 15-18, pp: 500-503.

Bhadade, U.S. and A.I. Trivedi, 2011. Lossless text compression using dictionaries. Int. J. Comput. Appl., 13(8): 27-34.

Botelho, F.C., A. Lacerda, G.V. Menezes and N. Ziviani, 2011. Minimal perfect hashing: A competitive method for indexing internal memory. Inform. Sciences, 181(13): 2608-2625.

Burnard, L., 1976. The University of Oxford Text Archive. University of Oxford, Retrieved from: http://ota.ox.ac.uk/catalogue/index.Html.

Clapson, A., 2014. A Note on Type Conversions and Numeric Precision in SAS®: Numeric to Character and Back Again. Statistics Canada. Paper No. 1752-2014. Retrieved from: http://support.sas.com/resources/papers/proceedings14/1752-2014.pdf.

Cox, N.J., 2002. Speaking stata: On numbers and strings. Stata J., 2(3): 314-329.

Ferragina, P. and G. Navarro, 2005. Pizza&Chili Corpus-Compressed Indexes and Their Testbeds. Retrieved from: http://pizzachili.dcc.uchile.cl/2013.

Fox, C., 1992. Lexical Analysis and Stoplists. In: Frakes, W.B. and R. Baeza-Yates (Eds.), Information Retrieval: Data Structures & Algorithms. Prentice Hall Inc., Uppar Saddle River, NJ, USA, pp: 102-130.

Godboles, S., I. Bhattacharya, A. Gupta and A. Verma, 2010. Building re-usable dictionary repositories for real-world text mining. Proceeding of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10). Toronto, Ontario, Canada, October 26-30, pp: 1189-1198.

Grill, B., 2014. A Survey on Efficient Hashing Techniques in Software Configuration Management. White Paper. Retrieved from: http://old.iseclab.org/people/bgrill/papers/grill_hashing_2014.pdf.

Jivani, A.G., 2011. A comparative study of stemming algorithms. Int. J. Comput. Technol. Appl., 2(6): 1930-1938.

Joshi, A., N. Thomas and M. Dabhade, 2016. Modified porter stemming algorithm. Int. J. Comput. Sci. Inform. Technol., 7(1): 266-269.

Popova, S., L. Kovriguina, D. Mouromtsev and I. Khodyrev, 2013. Stop-words in keyphrase extraction problem. Proceeding of the 14th Conference of Open Innovations Association (FRUCT), pp: 113-121.

Ramasubramanian, C. and R. Ramya, 2013. Effective pre-processing activities in text mining using improved porter's stemming algorithm. Int. J. Adv. Res. Comput. Commun. Eng., 2(12): 4536-4538.

Richter, S., V. Alvarez and J. Dittrich, 2015. A seven-dimensional analysis of hashing methods and its implications on query processing. Proc. VLDB Endowment, 9(3): 96-107.

Sakurai, S., Y. Ichimura, A. Suyama and R. Orihara, 2001. Acquisition of a knowledge dictionary for a text mining system using an inductive learning method. Proceeding of the Workshop on Text Learning: Beyond Supervision, pp: 45-52.

Singh, B., I. Yadav, S. Agarwal and R. Prasad, 2009. An efficient word searching algorithm through splitting and hashing the offline text. Proceeding of the IEEE International Conference on Advances in Recent Technologies in Communication and Computing. Kottayam, Kerala, India, pp: 387-389.

Stein, B. and M. Potthast, 2007. Applying hash-based indexing in text-based information retrieval. Proceeding of the 7th Dutch-Belgian Information Retrieval Workshop, pp: 29-35.

Van Rijsbergen, C.J., 1979. Information Retrieval. Department of Computer Science, University of Glasgow. Retrieved from: citeseer.ist.psu.edu/vanrijsbergen79information.html.

Vijayarani, S., J. Ilamathi and Nithya, 2015. Preprocessing techniques for text mining - An overview. Int. J. Comput. Sci. Commun. Netw., 5(1): 7-16.

Willett, P., 2006. The porter stemming algorithm: Then and now. Program-Electron. Lib., 40(3): 219-223.

Yao, Z. and C. Ze-Wen, 2011. Research on the construction and filter method of stop-word list in text preprocessing. Proceeding of the IEEE 4th International Conference on Intelligent Computation Technology and Automation (ICICTA), pp: 217-221.

Zhang, D. and W.J. Li, 2014. Large-scale supervised multimodal hashing with semantic correlation maximization. Proceeding of the 28th AAAI Conference on Artificial Intelligence, 1(2): 2177-2183.