International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Fast Dictionary Construction using Data Structure and Numeration Methodology with Double Hashing

Safa S. Abdul-Jabbar[1], Loay E. George[2]

[1] Baghdad University, College of Science for women, Computer Science Department, Baghdad, Iraq

[2]Computer Science Department, College of Science, Baghdad University, Baghdad, Iraq

Abstract: ***The problem of text retrieval is continuously attracting more research attention; they still used for efficiently analyze text data. The unstructured text data take more importance in numerous fields such as business analysis, customer retention and extension, social media, information retrieval and legal applications, etc. This article considers the importance of exploratory dictionary construction for finding the concepts of interest, also it proposes a system for efficient dictionary construction, tuning. The re-use of these dictionaries across a large scale and different datasets still remain an unsolved problem. This paper employing different types of hash functions to conduct progressive multi-search stages, and reducing the time that required constructing the dictionary as much as possible while maintaining the accuracy of the information contained in it. Many text-mining tools, hashing functions, data structures concepts and numeration operations were utilized in the planned system in order to provide a dynamic word dictionary. This could be used for fast text retrieval systems as a result of its small size in comparison with the original dataset. The proposed algorithm was designed for improving the time complexity due to the ability to retrieve an accurate result in a short time. This could be done by obtaining the advantages of binary search; which lets the processing time replaced from being linear to logarithmic behavior. The obtaining result is considered the highest when compared with the results of other published works, especially those based on dealing with string as a sequence of characters. The proposed system extracts the important word information's which gave chance to text retrieval system for attaining accurate and fast results.***

Keywords: Fast String Operations, Binary Search, Double Hashing, Thesaurus Construction, Keyword Generation, Knowledge Dictionary, extract word Features, data structures concepts

1. Introduction

Due to an oversized quantity of information that is held on and utilized in computers, a number of this information is in style of text information, some of this data is in form of text data. The ability of fast and accurate retrieval of specific information from this kind of data is still challenging task. This system should deal with the storage, organization, representations and access to text documents. The users should have easy accessibility to the documents relevant to their queries, and this can be provided by the organization of text documents. Any dataset used for retrieval system may contain the different number of text documents. When a user asks for a specific document, he should formulate his question so as to be processed by the retrieval system, and the result should select a specific number of documents which contain the query keywords as basic elements [1].

In summary, string matching problem findings the incidence of a pattern P of lengths m characters from a text T of length (L) in characters, and the pattern P has a different number of substrings (words) each one has multiple characters [2]. This problem was studied by many researchers; different techniques and algorithms were introduced to solve it [3] [4].

Text mining had gained importance due to growing usefulness of data mining applications occurred in recent years. Till 2008, the evolution of text mining was even small [5], but was grown rapidly because of the awareness of business that hidden in unstructured information. Text mining aims to understand and organize large amounts of unstructured data that are available in any system which use it to gain useful information that used to solve real-world problems (e/g. information extraction) [6] [7] [8]. Information extraction is a commonly used process that depending on the dictionary [9], entity annotation [10], classification [8], and link analysis tasks.

The process of constructing dictionaries depends on what is called a practitioner's art [7]; it requires experience and a lot of trial and error with manual tuning for measures (e.g., precision and recall) [6]. The practitioner's domain knowledge becomes particularly important for tagging abstract concepts [11].

Two kinds of knowledge dictionaries are provided. One of them is called the key concept dictionary while the other is called the concept relation dictionary. Words are extracted from the documents by lexical analysis. Each word is checked to see whether it corresponds to an expression in a key concept dictionary. If it is a key concept corresponding to the expression then the concept class corresponding to the concept is extracted. If a dictionary can consider a concept class as an attribute and the text class given by the reader can be considered as a class attribute for a training example, then it is possible to generate a training example from a document. The dictionaries are made through trial and error, so it is a time-consuming process. In addition, the dictionaries must be created for each target problem. Consequently, the creation is a bottleneck for applying a text mining system to a new target problem [12].

Sakurai et al. have proposed a method that used for automatic building rules and their classes from the original data by employment an inductive learning method [12]. The result showed that the fuzzy inductive learning algorithm is appropriate for the acquisition of the rules by providing

higher accuracy through numerical experiments based on 10 fold cross validation and using daily business reports in retailing [12].

In 2010, Godbole et al constructed a framework that provides a spread of interaction modes to the user to quickly build dictionaries over massive real-world datasets [11]. They adapt one or a lot of dictionaries across domains and tasks (e.g. social media mining). Thereby enabling reprocess of the information. They made a case study on real-life datasets; where the time and energy of the professional person will be preserved using the dictionary-based text mining tools. The proposed system is savings about 60% of the elapsed time and measures the control parameters (recall and Precision) according to different user words query.

The dynamic dictionary can be used for both compression and data retrieval operations from a large dataset. So, Bhadade and Trivedi 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 [13].

Global Dictionary (GDIC) data structure is used as a preprocessing model in 2011 by Park et al, they inspected 6,200 documents to find the similar text in huge document repositories [14]. In this method the frequency of occurrence of common non-stop words (i.e. stop-words like “I” and “to”) were measured. Then, they used the proportion of common non-stop words to determine either that or not to inspect documents. Based on this information, they extracted essential pairs of documents that compared. Each pair of these documents used in two different methods at same time: The first method used the common non-stop words which have high frequency.

The second method used the words with high proportions as common non-stop words.

This model reduced searching time to 64-87%, while the sensitivity was stood at 77-96%.

Also in 2011, Wang et al. proposed a new method for approximate string search. They used a dictionary in which the system can find words that are similar to the given misspelled word [15]. In other words, this method used the probabilistic approach to complete its search. This approach used log-linear model and an algorithm for finding the top number of candidates. The log-linear model can, definitely, used as the conditional probability distribution for the word. Also, they used the loss function with the learning method that employs the criterion in candidate generation. The tests indicated this algorithm is efficient and guaranteed for finding the best candidates.

In this paper, a dynamic word dictionary is generated using many text-mining tools, hashing functions, data structure concepts, and enumeration operations in which the word information’s are extracted from each file in any given dataset. This can be used for fast text retrieval systems because of its small size in comparison with the original dataset. This gave an effective role in increasing the system ability for retrieving the most relevant files for the inputted user query.

2. The Proposed System

The proposed system consists of three primary stages which are: (A) Lexical Analysis Stage, (B) Building the Primitive Dictionary Model Stage, and (C) Indexing Database System Stage.

A. Lexical Analysis Stage: is applied to extract the useful data using numeration operation for all input documents. This stage consists of two steps:

a) Data cleaning: is required to remove the noise and make it suitable for further stages.
b) File Filtering: is applied for managing the resulted files by passing only files that including useful data.

B. Building Main Dictionary Stage: is applied to extract a set of words to be used as features representing the original files to decrease the processing time of retrieval systems, because it will reduce the search space size. This stage implies the steps:

a) Reducing the Search Space: this step consists of two operations:
   1. Stop-word Extraction Operation: is used to extract all stop-words that appeared in the dataset, its frequencies and the number of files that containing each word. This information is provided using the data structures concepts to produce (hash0_StopwordsFile, Stop_wordsRec, and Filename_list):
      - Establishment of the first list contains the hash value that depends on the first two characters (first index), the number of words covered by this index and a pointer value refer to the start index in the second list.
      - Establishment of the second list contains the words that convey by each hash value with its frequency of occurrence and the start index in the document file list to each word.
      - Establishment of Short Document Record of Stop-words List: it is a list of packed records used for pointing to the series of documents conveys certain indexed word.
   2. Stemming Operation: is used for identifying the roots of all words in the dataset using the porter stemming with enumeration operations and lengths conditions in each step.

b) Statistical Analysis (2-level hashing index system): is applied for converting the unstructured text data to structured data; such that the arranged text becomes easy to use in any searching process (using the first two characters).

For the first two stages, we can use the same method that proposed by Abdul-Jabbar and George in 2016 [16]; Figure (1) shows the basic stages that composed the system.

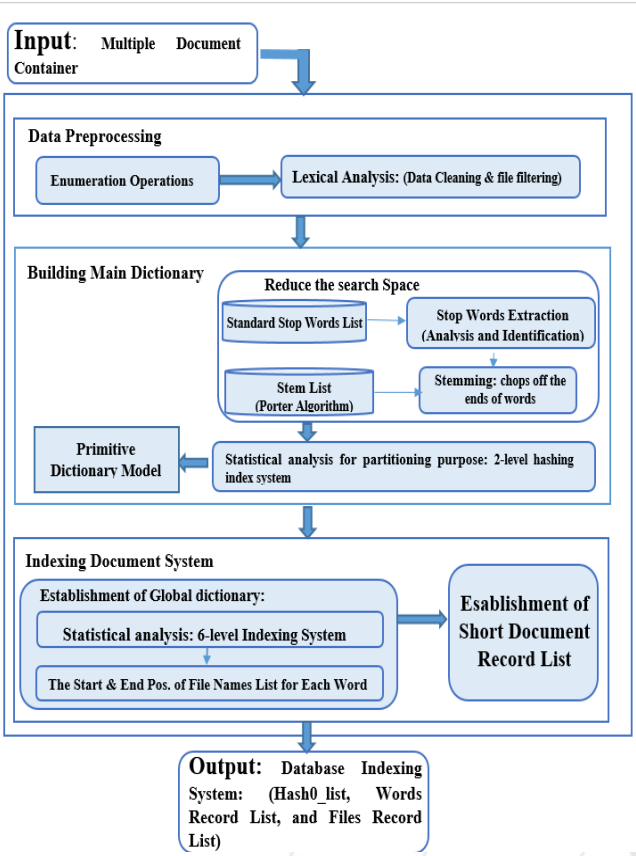

Figure 1: Basic system design

C. Indexing Database System Stage: it is applied for extracting the word features (such as the words weight), then determining the most and least significant words automatically in any dataset, and helping the system for partitioning the search space to multiple nodes; this will reduce the time that required for doing any retrieval process. This method is designed to implement this stage using some of the data structure concepts in order to produce the packed data list and the global dictionary:

- $Hash_0$_Record List: it contains the values of $hash_0$ which is using the first two characters of each word, $\boldsymbol{H_0}$; it is used as a pointing index for the starting position of searching operation in the global dictionary. Also, $Hash_0$ records list contains the frequency of occurrence of each $hash_0$ value to define the searching operation range.
- $Hash_1$_List: the global dictionary contains the $hash_1$ value of each word, $\boldsymbol{H_1}$; it uses the next four characters of each word appeared in the dataset. Also, this list holds the associated frequency of each H_1 value and the start pointing index in the packed list that contains the symbol numbers of document files containing this word.
- The short document file list of records of all document files containing each word listed in the global dictionary; in other words it holds a reference list of the document files names (or symbol numbers) convey the indexed words.

The computed Hash values depends on only 6 characters of each word because they cover around 83% of the whole datasets as depicted in Table (1).

Table 1: The Cumulative Average of Words that Available in All Lengths (for Words length range [1-16]

Words Len.	Dataset1	Dataset2	Dataset3	Dataset4
1	7.742%	9.245%	8.8235%	6.362%
2	9.962%	11.967%	11.600%	7.203%
3	25.466%	27.565%	27.983%	22.494%
4	52.389%	53.515%	54.810%	44.713%
5	70.645%	72.398%	73.723%	64.623%
6	**83.374%**	**84.638%**	**85.222%**	**79.115%**
7	90.971%	92.258%	92.267%	88.356%
8	95.186%	96.267%	96.113%	93.279%
9	97.222%	98.281%	98.079%	95.671%
10	98.339%	99.276%	99.108%	97.173%
11	98.903%	99.656%	99.542%	97.772%
12	99.250%	99.846%	99.771%	98.647%
13	99.455%	99.943%	99.892%	99.733%
14	99.594%	99.975%	99.940%	99.828%
15	99.709%	99.987%	99.965%	99.875%
16	99.781%	99.994%	99.980%	99.915%

The indexing method has been used in order to speed up the process of access to certain information. Nowadays, all modern computers have large RAM size which is enough to avoid the way back and forth to the hard disk each time the CPU need relatively large information for doing some operations; taking into consideration the repetitive data movement between hard disk and the RAM will take a lot of time. Each operation relevant to indexing operation requires:

- **CPU Time:** the time needed for processing each operation.
- **Storage Time:** the time required for accessing storage in order to get the required information.

Hence, to reduce the elapsed time during any searching operation by we need to reduce the number of visiting to non-volatile storage instead of going directly to RAM (when required); this was done through providing the required information as packed in the RAM. This was implemented in the developed method by establishing a global dictionary and a packed data list that contains words information in the search space.

As an example, if we have a dataset containing 1000000 words; for each word a record with length 12 bytes will be assigned to store the word information. So, the required storage will be 12 * 1000000 =12 MB; this is a small size in comparison with the memory size that can be easily provided by the new versions of computers; such that they can hold these lists of information in the volatile storage part. This will significantly accelerate the time access to the vital information. It is worth to mention that the process of gathering words information for the whole search space will be accomplished once at a time at the initialization phase of the program execution.

Algorithm (1): Global Dictionary Algorithm
Objectives: Taking the advantage from data structure tools for building global dictionary that can used for searching proposes.
Input: 729 files contain all words in the dataset. **Output: The hash value for each word, the start and end index for each word in the dependent list that**

contain words freq., besides the files number (Global Dictionary).

```
Step1: Define struct (records for storing information about each word in the dataset)
        Struct WordRecType
          int Hash = 0,  StCnt = 0,  Freq = 0
        End struct
Step2: Generate the Indexing Table for Hash Value Determination
      Byte V[255]
      For I = 0 to 255 step 1, V[I] = 0, Next I
      For I = 33 to 127 step 1,  V[I] = 27, Next I
      For I = 65 to 90 step 1, V[I] = I – 64, Next I
      For I = 97 to 122 step 1, V[I] = I – 96, Next I
Step3:Calculate H1 & Load the File Data
  H1 = 27 * V(Asc(Mid(Fil$, 1, 1))) + V(Asc(Mid(Fil$, 2, 1)))
  A←read the data file
Step4: File Scan the determine H2 & Frequency
    For I = 3 to L
     If (A(I) = 44 & A(I + 5) = 32) then
      WrdLen = I - St
      If WrdLen = 2 then
       H2 = 0
      ElseIf WrdLen = 3 then
       H2 = 19683 * V(A(St + 2))
      ElseIf WrdLen = 4 then
       H2 = 19683 * V(A(St + 2)) + 729 * V(A(St + 3))
      ElseIf WrdLen = 5 then
       H2 = 19683 * V(A(St + 2)) + 729 * V(A(St + 3)) + 27 *
       V(A(St + 4))
      ElseIf WrdLen >= 6 then
       H2 = 19683 * V(A(St + 2)) + 729 * V(A(St + 3)) + 27 *
       V(A(St + 4)) + V(A(St + 5))
      End If
        Freq(H2) = Freq(H2) + 1, St = I + 6, I = I + 5
     End If
    Next I
Step5: Construct the Words Record List
    M = -1, St = 0
    For H2 = 0 to 531440
     If (Freq(H2) > 0) Then
        M = M + 1
       WordList(M).H1 = H1, WordList(M).H2 = H2,
       WordList(M).Freq = Freq(H2),
       WordList(M).St = St + FileOfs, St = St + Freq(H2)
     End If
    Next H2
    FilLstLen = St - 1
Step6: Print in file. \\ Print the resulted Hash0 list values in
        files that have corresponding names to the buffers.
       countt = M,  total = start + M
       open ssss for binary writer as bfile
            bfile.write(H1, Fil, start,total)
       close file
       start = total + 1
Step7: Construct the Reference File List
    For I = 0 to M
     H2 = WordList(I).H2,    Pnt(H2) = WordList(I).St - FileOfs
    Next I
    OldFileOfs = FileOfs
    FileOfs = FileOfs + FilLstLen + 1
    St = 3, L = L - 5
    For I = 3 To L
     If (A(I) = 44 & A(I + 5) = 32) then  WrdLen = I - St
      If WrdLen = 2 then
       H2 = 0
      ElseIf WrdLen = 3 then
       H2 = 19683 * V(A(St + 2))
      ElseIf WrdLen = 4 then
       H2 = 19683 * V(A(St + 2)) + 729 * V(A(St + 3))
      ElseIf WrdLen = 5 then
       H2 = 19683 * V(A(St + 2)) + 729 * V(A(St + 3)) + 27 *
       V(A(St + 4))
      ElseIf WrdLen >= 6 then
       H2 = 19683 * V(A(St + 2)) + 729 * V(A(St + 3)) + 27 *
       V(A(St + 4)) + V(A(St + 5))
      End If
      FilNo = 16777216 * A(I + 4) + 65536 * A(I + 3) + 256&
       * A(I + 2) + A(I + 1)
      FilLst(Pnt(H2)) = FilNo, Pnt(H2) = Pnt(H2) + 1
      St = I + 6: I = I + 5
     End If
    Next I
Step8: Save the File List
        open ssss for binary writer as bfile
            bfile.write(FilLst)
        close file

End;
```

The size of the short document list can be reduced by avoiding repeating the same file name that containing each word. This was implemented using an additional integer number to refer for the number of repeated times for each file name. The influence of implementation of this operation is as follow:

- Equal Storage Size Case: this case is realized when we have words that occurred twice in the given file; in this case in the original file 2 integer numbers (4 bytes for each integer number in C#) should be used to store the files ID number, and in the compressed method we also use 2 integer numbers (the first number used to refer for containing file for that word, and the second number is used to refer for the number of same word repetitions in that file). So, the number of used bytes will be same in both (the old and the new) file.
- Lost Storage Case: lost storage is possible only if we have words that appeared once in the given file; in this case we lost 4 bytes (i.e., for the word frequency number in the file) there is no need for this number in such case because consuming an integer value to refer that the word was appeared once in this file is meaningless; it is important to mention that this case has low incidence.
- Gain in Storage Case: for repeated words in the given file, no matter how repetitive words are in each file because for any repetition case only 2 integer numbers will needed (the first number uses to contain the file name, and the second number uses to register the number of repetitions).

Figure (2) shows the illustrated proportions of these three cases as follows (the old size=4639612 byte, new size=2526832 byte, the loss size=-831916 byte, the gain size= 2944696). So, the Net Gain= 2112780, the compression ratio=1.8361, and the Size Profit=45.5379%.

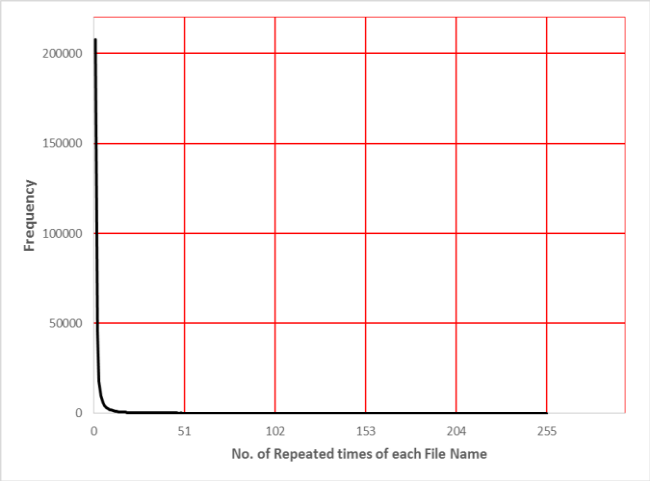

Figure 2: The cumulative rate of file frequency according to the number of recurrence (as an example 0 to 255 no. of repeated times)

After that, depending on these statistical analysis results, it was noted that the profit size will be recorded at a rate of **45.53%,** which leads to the success of applying the proposed compression method. In order to implement this compression operation Algorithm (2) was applied in this stage on the resulted files from the previous operation.

Algorithm (2): Compress the Short Document Record of Words List Algorithm

Objectives: Reduce the Suze that need to store the information of any given dataset

Input: Short Document Record of Words List, and Global Dictionary Files.
Output: The same number of input text files with reduced size.

Step1: Define struct (records for storing information about each word in the dataset)

```
Struct WordRecType
   int Hash = 0, StCnt = 0, Freq = 0
End struct
Struct WordRecType2
   int Hash = 0, StCnt = 0, Freq = 0,    Total_Freq=0
End struct
Struct File_list
   int File_Name = 0, File_Freq = 0
End struct
```

Step2: Read Text files

A ← read the file hash0_list content as a sorted array of structures
B ← read the file global dictionary content as a sorted array of structures
Read the file_list file as array of bytes

Step3:Compress the Files Name List //Compress this file by removing repeated file name with maintain the total freq. of each word that considered as useful information for compute weights of each word. This will change the information of each word that stored in the global dictionary. Then store the files information (filename, and frequency) in a different structure called File_list.

```
For j=0 to B.length step 1
  For I = 0 to B[j].Freq step 1
     if (File_list.File_Name is the similar to the pervious one) then
                File_list[File_Name]. File_Freq = File_list[File_Name]. File_Freq +1
          else
                WordRecType2[j]. Freq= WordRecType2[j]. Freq+1
          end if
       Next I
        WordRecType2[j].Hash= WordRecType[j].hash //The hash value does not changed
        WordRecType2[j]. StCnt = WordRecType[j]. StCnt +WordRecType2[j]. Freq +1
        WordRecType2[j]. Total_Freq= WordRecType [j]. Freq   //To save the freq. of each word in the dataset
     End For
Step5: Save the Output Files
       byte[] By = new byte[4]
       set ssss ← file path for hash values
       open ssss for binary writer as bfile
         bfile.write(A)
       close file
       set ssss ← file path for the structures of word records
       open ssss for binary writer
         For u = 0 to WordRec2.Length step 1
           By = BitConverter.GetBytes(WordRec2[u].Hash), b.Write(By)
           By = BitConverter.GetBytes(WordRec2[u].Freq), b.Write(By)
           By = BitConverter.GetBytes(WordRec2[u].StCnt), b.Write(By)
           By = BitConverter.GetBytes(WordRec2[u].Total.Freq), b.Write(By)
         next u
       close file
       set ssss ← file path for the structures of word records
       open ssss for binary writer
         For u = 0 to File_list.Length step 1
           By = BitConverter.GetBytes(File_list [u].File_Name), b.Write(By)
           By = BitConverter.GetBytes(File_list [u]. File_Freq), b.Write(By)
         next
       close file
```

End;

3. Experimental Results

In this section, the results of some conducted tests are presented and discussed to evaluate the performance of the established system. The programming language C# (Microsoft Visual Studio 2015) was utilized to develop the programs. In order to test the proposed system performance, in this thesis work the tests were conducted using four big datasets; they are:

- **Dataset-1:** It was collected from papers, books and articles with the possibility of recurrence of partial files contents. The file sizes ranging within [1KB-20374KB]. The total size of this dataset (Loay & safa dataset) is 4.26GB [17].
- **Dataset-2:** It was constructed using Oxford University Text Archive; it was collected by comprising a number of texts taken from different sources. They usually compiled for purposes of linguistic research. It forming text with size 541 MB. It was designed to represent a wide cross-section of current British English [18].

- **Dataset-3:** It is a standard dataset obtained from Pizza & Chili [19]. After downloading this dataset, it was partitioned into files for mining processing according to its content. The resulted file sizes ranging from 1KB to 29.633KB. The whole files cover the size 2.10GB.
- **Dataset-4:** This dataset is a collection of public USENET postings, which was collected between Oct 2005 and Jan 2011. It covers 47,860 groups of English language, non-binary-file news. It contains very small percentage of non-English words and non-words; this corpus is a raw text [20].

The proposed system was designed using some concepts of data structures discipline; according to the method, the resulted files of this stage are: (i) the short stop-word list of records; (ii) stop-words dictionary; (iii) short list of records for documents files; (iv) single complete dictionary; and (v) the non-duplicate words in the search space. Generally, this stage was applied to extract words characteristic from large-scale textual data and supports the operations of visualizing them with high performance.

The elapsed time for implementing this stage is shown in table (1) for all tested datasets in order to demonstrate the impact of each mentioned stage. In this section, each step of the proposed system was tested using enumeration. Table (1) shows that the baseline classification performance of lexical analysis is better than other stages depending on the noise removing. The stop-word extraction operation has a larger time compared with other operations because of the required matching operations between each word and all words contained in the stop-word list.

The size of resulted files for each stage was listed in table (2). This may considered as a preliminary file compression stage, which can be used to consolidate storage that contains only the most important information of the dataset.

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

Stage	Lexical Analysis Stage	Building Primitive Dictionary Stage	Indexing Database Stage
Dataset1	16.35	116.38	3.02
Dataset2	1	13.51	0.408
Dataset3	2.25	58.08	1.7
Dataset4	137.28	1193.9	35.5

The overhead operations for each string operation are: conversion of each string to its equivalent ASCII value, doing other operations which are relevant to characters' case, and then recover the string form to display the result on the screen. As a result of avoiding these operations by using the proposed method, the profit ratio in both (the execution time and the search space) was shown in Table (3) & Table (4).

The attained test results showed that the proposed system can significantly reduce the required time of each operation; as shown in Tables (1). Hence, this system can save up to 92.05% of the execution time for the dictionary construction operation computed with controlling the accuracy of the resulted files; this can be considered as an investment in both time and hardware. The Net time profit of each step was computed using the following equation which depends on the execution time of each one:

$$\text{Net time Profit Percentage} = \frac{T_1 - T_2}{T_1} * 100\%$$

Where, T1 is the elapsed time when using the traditional method; T2 is the elapsed time when using the numeration method.

While, the saved space percentage was computed using the same equation with one difference: ***T1*** is the size of the original dataset; ***T2*** is the size of the dataset after processing operations. For comparison purposes, the size of each dataset was computed when building the main dictionary using the traditional operations (i.e. lexical analysis, stop-word extraction, stemming, and 2- level hashing system) then compared with the size resulted from the proposed system, as shown in table (4); in order to find the most efficient method for reducing the search space size with maintaining the relevant information of each word.

Table 3: Net time Profit Percentage for each Stage Compared with the Traditional Method

	Stages	Net time Profit Percentage
Dataset1	Lexical analysis	87.37
	Stop-words Extraction	99.285
	Stemming operation	99.60
	2-level Hashing index	96.07
Dataset2	Lexical analysis	84.70
	Stop-words Extraction	94.35
	Stemming operation	99.80
	2-level Hashing index	95.42
Dataset3	Lexical analysis	87.37
	Stop-words Extraction	80.6
	Stemming operation	98.50
	2-level Hashing index	92.41
Dataset4	Lexical analysis	65.05
	Stop-words Extraction	93.83
	Stemming operation	99.74
	2-level Hashing index	98.76

Table 4: The Reduced Search Space Size Percentage

	Net Saved Size Percentage using the Building Main Dictionary Stage	Net Saved Size Percentage using the Proposed System
Dataset1	23.49%	99.85%
Dataset2	24.85%	98.89%
Dataset3	12.26%	99.69%
Dataset4	4.30%	99.73%

Table (5) presents comparisons between the system performances of the proposed system with other recently published work for the words with lengths up to 16 characters. The listed results in this table indicate the effectiveness of the proposed system.

Table 5: The comparisons between the elapsed times between the proposed system and the previous works

Method name	Author name	Data size	Computation Time		Used techniques
			Searching time(s)	Preprocessing time(s)	
GDIC (Global DICtionary)	Park et al.(2011)	6,200 documents	9,705.33	12,624.32s	Dictionary
Fast and Accurate Method for Approximate Search	Wang et al. (2011)	973,902 word	Did not mentioned	Did not mentioned	Dictionary
The Proposed Method	Safa & Loay (2016)	4-datsets with different sizes	$O(m \log_2 n)$	save up to 92.05% of the traditional execution time	Hashing & Dictionary

These comparisons are difficult because each method uses different techniques. The elapsed time for each method is depending on the structure of the program (the reflection of the programmer ideas) and the computer hardware. So, in order to compute the required time for each method the computation complexity was used as an indicator to the required time of each method. Where n is the dataset size; L is the number of pattern P; m is the number of words.

As showed in Table (5) the proposed method gives as excellent performance (time and maintain the system accuracy) on retrieval systems because the computation complexity has a logarithmic base, this success is entirely as a result of utilize the numeration and hashing methodology and using the data structure concepts and binary search strategies. While, the traditional method have a linear time complexity that directly depends on the dataset and the given query size.

4. Conclusions and Future Work

In this paper, a new dictionary based on data structure concepts and hashing methodology has been proposed. This dictionary is more flexible for providing the dynamic range of words because more number of characters can be involved for defining each word using multi hashing operation. It uses the data structure concepts which enable us to use the binary tree structure as a strategy for performing the searching operation; this will reduce the elapsed time for searching operation from O(n) to O(Log2n); The partitioning operation will be easier for preparing the distributed search space which consists of many space search tiles such that each tile can be conducted by certain server in the system. In addition, the number of most redundant words for each slot can be set variable; depending on the nature of the text which entirely depends on the language of the dictionary. In other words, in the case of using Arabic, it will reflect other most redundant words distribution in comparison with the case of dealing with the English language. After accomplishing the dictionary construction operations, the overall performance is improved due to reducing the searching space. The attained reduction percentage for the total search space when using the traditional string operations for building the dictionary was 16.22%. While, the use of the proposed system led to size reduction percentage around 99.54% with preserving the search accuracy for the relevant information of each word.. Porter stemming algorithm was slightly modified by reducing the checking operations applied on each condition by taking the advantage of words length (i.e., checking the length of the given word at the beginning of each condition to avoid wasting time for processing word cases which does not meet the similarity condition). The required length for each condition will equal the word suffix plus one for the minimum remaining characters, in addition, the using of the numeration operation in each step leads to reducing the processing time taken by this algorithm. A double hashing indexing system was introduced, it depends on the first six characters of each word, the first two characters are used to establish the first hash level and the next four characters for building the second level. The process of counting the frequency of occurrence of each word will be very useful to accomplish proper flexible segmentation and set the position of segmentation pivot points along the word list and the corresponding file pointing list.

As a future work, a suitable stop-words statistical distribution model can be developed by computing the stop-words frequencies over whole documents files; this can be done over each language to be covered by the system. Also, this improvement will increase the retrieving ability of the system. The adopted stemming operation can be substituted by other algorithms (i.e., lemmatization operation); that involve many tasks such as understanding the context and determining the meaning of word in a sentence. Taking into consideration the used stemming method achieved relatively rational results. Beside that, another level of hashing step can be used and its effectiveness on the system accuracy can be studied. Since, the proposed system deals only with documents consist of English words; it can be developed to be a multilingual system can retrieve the documents consist of any natural language.

References

[1] Baeza-Yates, R., and Ribeiro-Neto, B., “Modern information retrieval”, New York: ACM press, 1999.
[2] Fredriksson, K.; and Grabowski, S.,”Average-Optimal String Matching”, Journal of Discrete Algorithms 7(4), pp. 579–594, 2009.
[3] Charras, C., and Lecroq, T., “Handbook of Exact String Matching Algorithms”, King’s College London Publications Pp. 1-17, Vol. 2, ISBN-13: 978-0954300647, 2004.
[4] Navarro, G., and Raffinot, M., ”Flexible Pattern Matching in Strings”, Cambridge University Press, ISBN: 0521813077, 2002.
[5] O’Dowd, S., 2008. Unstructured Data and Text Analytics in Capital Markets. Financial Insights Report-IDC report.
[6] Bhattacharya, I., Godbole, S., Gupta, A., Verma, A., Achtermann, J., and English, K., “Enabling Analysts in Managed Services for CRM Analytics”. In Proceedings of the 15th ACM SIGKDD International Conference on

Knowledge Discovery and Data Mining, pp. 1077-1086, ACM, 2009.
[7] Takeuchi, H., Subramaniam, L.V., Nasukawa, T., and Roy, S.,"Getting Insights from the Voices of Customers: Conversation Mining at a Contact Center", Information Sciences 179(11), pp.1584-1591, 2009.
[8] Godbole, S., and Roy, S.,"Text Classification, Business Intelligence, and Interactivity: Automating C-Sat Analysis for Services Industry" In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 911-919, ACM, 2008.
[9] Riloff, E., and Jones, R.,"Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping". In AAAI/ IAAI, Pp. 474-479, 1999.
[10] Bunescu, R. C.,"Learning for Information Extraction: from Named Entity Recognition and Disambiguation to Relation Extraction", ProQuest, 2007.
[11] Godbole, S., Bhattacharya, I., Gupta, A. and Verma, A., "Building re-usable dictionary repositories for real-world text mining". In Proceedings of the 19th ACM international conference on Information and knowledge management, pp. 1189-1198. ACM, 2010.
[12] Sakurai, S, Y, Ichimura, A, Suyama, and R, Orihara, " Acquisition of a knowledge dictionary for a text mining system using an inductive learning method", In Proceedings of IJCAI 2001 Workshop on Text Learning: Beyond Supervision, pp. 45-52, 2001.
[13] Bhadade, U.S. and A.I. Trivedi, "Lossless text compression using dictionaries", Int. J. Comput. Appl. 13(8). pp. 27-34, 2011.
[14] Park, S.Y., Kim, S.Y., Kim, S.H., and Cho, H.G., "A Global Dictionary Based Approach to Fast Similar Text Search in Document Repository", In Computer and Information Technology (CIT), 2011 IEEE 11th International Conference, Pp. 526 – 532, IEEE, 2011.
[15] Wang, Z.; Xu, G.; Li, H.; and Zhang, M., "A Fast and Accurate Method for Approximate String Search", Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Vol 1, pp. 52-61,2011.
[16] Abdul-Jabbar, S., S. and George, L. E.,"Building Words Dictionary List Using Symbol Enumeration and Hashing Methodology", Research Journal of Applied Sciences, Engineering and Technology 13(12), pp. 885-894, 2016.
[17] Sami, S., Loay E., 2016. "Loay & Safa Dataset". Mendeley Data, v2 http://dx.doi.org/10.17632/ggh75fd25f.2.
[18] Burnard, L., the University of Oxford Text Archive, 1976, University of Oxford, http://ota.ox.ac.uk/catalogue/index. html.
[19] Ferragina, P. and Navarro, G., 2005. Pizza & Chili Corpus—Compressed indexes and their testbeds. http:// pizzachili.dcc.uchile.cl/ 2013.
[20] Shaoul, C. & Westbury C., "A reduced redundancy", USENET corpus (2005-2011) Edmonton, AB: University of Alberta, http://www.psych.ualberta.ca/~westburylab/downloads/usenetcorpus.download.html.