ISSN: 0067-2904

Fast Text Analysis Using Symbol Enumeration and Hashing Methodology

Safa S. Abdul-Jabbar*, Loay E. George
Department of computer science, College of science, University of Baghdad, Baghdad, Iraq

Abstract
This paper is focusing on reducing the time for text processing operations by taking the advantage of enumerating each string using the multi hashing methodology. Text analysis is an important subject for any system that deals with strings (sequences of characters from an alphabet) and text processing (e.g., word-processor, text editor and other text manipulation systems). Many problems have been arisen when dealing with string operations which consist of an unfixed number of characters (e.g., the execution time); this due to the overhead embedded-operations (like, symbols matching and conversion operations). The execution time largely depends on the string characteristics; especially its length (i.e., the number of characters consisting the strings plus the number of words in the sentence). In other words, the variable length of strings is an obstacle to achieve processing uniformity when manipulating strings. Many of string matching algorithms were introduced in the literature to deal with fixed length of characters of each string. In this paper, some test results are provided for a number of string operations (such as, simple string matching, hashing indexing systems, stop-words collection and text extractions). To understand the advantage of the proposed method, these operations were applied on different sizes of text files. A comparison is made with the results of using traditional methods that deal with strings only. The overall results demonstrate the positive effectiveness of the proposed approach.

Keywords: String matching operation, Data Editors, String Enumeration, String Hashing, and Text Analysis.

التحليل السريع للبيانات باستخدام طرق التجزئة و ترقيم الرموز

صفا سامي عبد الجبار*، لؤي ادور جورج
قسم الحاسبات، كلية العلوم، جامعة بغداد، بغداد العراق.

الخلاصة
يركز هذا البحث على تقليل الوقت المستغرق في عمليات معالجة النصوص عن طريق الاستفادة من عمليات ترقيم سلاسل الحروف **"Strings"** باستخدام نظام التجزئة المتعدد. حيث يعتبر تحليل النصوص من المواضيع المهمة لاي نظام يتعامل مع سلاسل الحروف (وهي سلسلة من الحروف المعتمدة من اي لغة) وبرامج معالجة النصوص (مثل برنامج معالجة الكلمات، برنامج تحرير النصوص وغيرها من الانظمة الخاصة بسلاسل الحروف). عند التعامل مع العمليات الخاصة بسلاسل الحروف ذات الطول الغير ثابت تظهر العديد من المشاكل (مثل وقت التنفيذ) بسبب العمليات الضمنية التي تحدث داخل المعالج (مثل مطابقة الرموز ،وعمليات التحويل). ويعتمد وقت التنفيذ بشكل كبير على خصائص سلاسل الحروف وبالاخص طول السلسلة (عدد الاحرف المكونة للسلسلة بالاضافة الى عدد الكلمات في الجملة). وبعبارة اخرى فان الطول المتغير للسلاسل يشكل عائقاً لتحقيق التوحيد لطرق معالجة النصوص. في الدراسات السابقة تم تقديم عدد من الخوارزميات الخاصة بمطابقة السلاسل التي تتعامل مع عدد ثابت من

*Email: safasami1988@gmail.com

الحروف لكل سلسلة. في هذا البحث تم توفير بعض من نتائج الاختبارت لعدد من العمليات الخاصة بسلاسل الحروف (مثل عمليات المطابقة البسيطة، انظمة الفهرسة باستخدام التجزئة ،جمع واستخراج كلمات التوقف **"Stop–words"**). ولتوضيح فائدة الطريقة المقترحة تم تطبيق هذه العمليات على احجام مختلفة من الملفات النصية، واجراء مقارنة للنتائج التي تم الحصول عليها مع نتائج الطرق التقليدية التي تتعامل مع السلاسل النصية فقط . حيث اظهرت النتائج بشكل عام فعالية ايجابية لهذه الطرق المقترحة.

Introduction

Text analysis is an important research area to extract meanings of text statements and phrases, patterns, and hidden structure(s) in unstructured text data and to investigate the problem of text comprehension, depending on the properties of texts; this would allow accurate to matching of the text to the needs of the user [1-2]. Text analysis applications are common in the business environment. Text analysis systems used in many industries. In recent years, text analytics has been heavily used for discovering trends in textual data (e.g., fraud detection, and crime prevention). Also, in hospitals that using the text analysis to provide better care. Scientists in the pharmaceutical industry are using this technology to mine biomedical literature to discover new drugs, and searching/matching systems. Text analyzers use many techniques that are useful for deriving insights from unstructured data. Some of these techniques are information retrieval, text extraction, text clustering, text summarization, text categorization and text filtering (that could be considered in the domain of text mining).

Some of text mining techniques in the field of textual information are using Natural Language processing and machine learning [2]. Many of these techniques can be considered as a time consuming process because it deals with string [3]. Many challenges must be handled when dealing with string operations, like:

1. The execution time is largely dependent on the string length, and it must be as fast as possible in all applications especially the real time applications.
2. The string length limitation (need to build a flexible application with unlimited string length) that is inversely proportional to execution time.
3. Reduce the hidden-operations (such as symbols matching calculations and conversion operations) which consuming RAM and CPU time.

In order to speed text-mining techniques and reduce the memory and CPU consumption this paper proposes a new method that handles string operations as a sequence of numbers to reduce the hidden cost of the string operations.

Relevant Background

Any dataset has many alphabetic characters, numbers, and punctuation characters such as dots and nonprintable characters that need to store as numbers form for reducing the computational load that required for dealing with blocks of characters stored as strings. This remark not currently discovered. Stata Corporation, its first release Sat-1.0 released with characters defined numerically only, discussed it for the first time in 1985. While the string variables were introduced in Stata 2.0 in 1988, it can deal with strings have even number of characters, specifically with those have length range between 2 to 80 characters for each string; while the cases of characters' sequences consist of an odd number of characters in string variables have been solved in Stata 2.1 in 1990.

In addition, the byte type submitted to join the existing numeric types of integer, long, float, and double. The next important change made in 2002 by raising the limit for string variables to 244 characters [4]. Although "destring" (conversion of string to numeric form) has a long history, it adopted in official Stata version 7 [5]. Then, "tostring" operation come later [6-7]. Still, "destring" has no practical implications, only users who started with Stata Technical Bulletin (STB) became accustomed to the way in which it works, however, it is easy to do the destring operation using command design. The initial design of the command was so easy to change any dataset without handling for special cases that may contain punctuation marks such as", replace". Destring operation allows a variable to be replaced by their numeric equivalents, except if users specifically abort that, while the existing numeric variables would be unchanged. This can be used for solve several dataset problems with no more than a single destring (no variable list, no options) [4].

Convert Characters to Numbers

The string used to define a sequence of characters that can used with their associated operations. The global system organizations consider strings similar to an array of characters. While a character is defined to be equivalent to a string of length one; for example, the string 'XYZ', is equivalent to an array of length three, contain 'X', 'Y' and 'Z' characters [7].

Computers store numbers and characters in different ways. Each character from the string is stored byte by byte, so the length of any string is equal to the number of bytes that used for each character (length of a string = number of characters for that string). On the other side, numbers are stored in the easiest way that uses less number of bytes (e.g., 4 bytes for integer & 8 bytes for long). Therefore, in case of numeric variables there is no need to represent each decimal digit as byte, like the case of representing each character element of a string [8].

Conversion operations mean changing the way in which the underlying data is stored, so it is necessary to understand what is going on behind the scenes in order to keep accuracy for these operations [9].

The Proposed System

In this paper, a method for enhanced text analysis system implemented to speed up the analysis process by using numeration and hashing methods. The overall design of the proposed system shown in Figure-1 ; it consists of three stages**:**

1. **Lexical Analysis.**
2. **Stop-words filtering.**
3. **Statistical Analysis.**

The input of this system can be of set consists of any number of text files with various sizes and can be taken from many resources. The main task of the first stage is to analyze the input file and extract words that contain only English alphabet characters with collective meaning.

The stop-words, which are common words appearing within a collection of texts; These words compromise up to 40% of words in any text document and significantly affect the performance of text retrieval systems. Therefore, the main target of the second stage, stop-words filtering, is to improve the performance through reducing the searching space in order to reduce the required response time for any retrieval system. The proposed system not only deletes these stop-words, but also extracts them with statistical analysis for each one (counting its frequency and the number of files in which each word can appeared).

The main purpose of the third stage, statistical analysis, is to perform text categorization process; that is indexing all the resulted words from the previous stage, according to the first two characters with attaching file names (or the file index number) for each word for facilitation partitioning purpose, speeding up text retrieval operations, and sorting files in order to take advantage of the binary search.

Hence, the output for the proposed system consists of two files:

1. Stop-words file: that contains information about all words in the stop words list.

2. Indexed files: contain sorted words with reference to the file that contains each word.

The steps shown in Figure-1 were implemented with the help of using string to numeric conversion operations in order to reduce the text symbols matching calculations and the associated hidden operations. Then, to find the effect of the proposed method, it will be compared to the traditional methods can be used to conduct each step. In the following sub-section, the taken steps to implement each stage of the proposed system are given below.

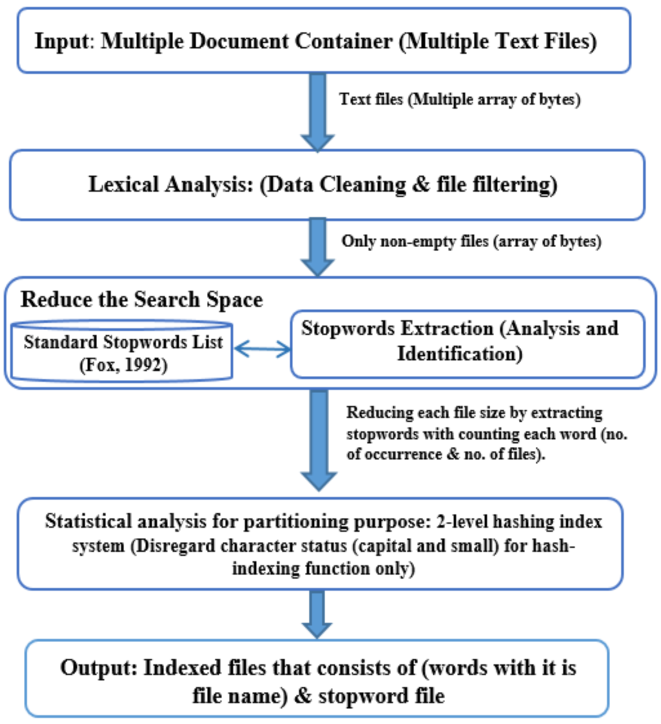
Input: Multiple Document Container (Multiple Text Files)
Text files (Multiple array of bytes)
Lexical Analysis: (Data Cleaning & file filtering)
Only non-empty files (array of bytes)
Reduce the Search Space
Standard Stopwords List (Fox, 1992)
Stopwords Extraction (Analysis and Identification)
Reducing each file size by extracting stopwords with counting each word (no. of occurrence & no. of files).
Statistical analysis for partitioning purpose: 2-level hashing index system (Disregard character status (capital and small) for hash-indexing function only)
Output: Indexed files that consists of (words with it is file name) & stopword file

Figure 1- Block diagram for overall system design

1. Lexical Analysis

When data collected from various resources (e.g., files) it may consist of a number of letters, non-printable characters, digits and punctuation. So, a need for extracting only the words from these resources should handle. Algorithm (1) illustrates the taken steps for extracting the words from the coming text streams loaded from input resources; in this work, the resource considered a collection of text 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. **Step3: Copying to a buffer.** \\ Copy each letter to buffer (a temporary array) which assigns 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. **Step4: Buffers Filtering.** \\ Passes only non-empty buffers. **Step5: Print in files.** \\ Print the resulted buffers in files that have corresponding names of the buffers.
End;

2. Stop-words Filtering

Any text file contains words belong to certain natural language has specific properties. Some of these properties are same for all such text files; for example, the existence of stop-words. The earliest work of stop-words removal assigned to Hans Peter Luhan in 1957, who suggested that the words in any natural language texts can divided into keyword terms and non-keyword terms that defined latter

as a stop-words. The stop-words are the most frequent words such that they become not significant (or descriptive) for the text container in compared with other words. As an example of stop words are prepositions, interjections, conjunctions, numerals, etc. The main purpose of removing stop-words using pre-compiled stop-words is to reduce the noise in any textual data. In addition, the impact of removing stop-words is reducing the level of data sparseness; which in turn reducing the size of the feature space and, consequently, improving the classification performance [10].
For handling this stage, the following two remarks considered in the relevant literature:

1. Discarding Stop-words: This method based on the idea that discarding unimportant words in order to reduce the feature space of the classifiers and assist to produce more accurate results [11].
2. Power of Stop-words: Many researchers claim that the removing of stop-words harms the performance of system classifies. For solving this limitation, several approaches have appeared in the areas of document retrieval and classification that aim to build a stop-words list. These approaches measure the discriminating power of each word using various methods such as the distribution of the occurrence frequency of words (i.e., considering the relative the occurrence frequencies over all text files). According to this aspect this tested word is considered as a stop-word when it has low discrimination power [10].

In this paper the [12] stop-word list suggested by Fox adopted; this list collected using the second approach that depends on the impact power of these words. This operation performed by calculating the frequency of each word (the repetitions of each word for finding the discrimination power).
Algorithms (2) and (3) illustrate the taken steps for extracting the stop-words with statistical analysis for each one of the coming text streams loaded of the previous stage.

Algorithm (2): Stop words Removal Algorithm (numeric method)

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.

Step1: Determine stop-words list (Fox, 1992). // Define the pre-compiled a list of stop-words (i.e. An array of bytes of two dimensions) that defined by Fox in 1992.
Step2: 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
Step3: 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
Step4: 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
Step5: 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.
Step6: Create stop-words 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
```

Algorithm (3): Stop-words Algorithm (Traditional method)

Objectives: Reduce the search space that leads to reduce the required time for the overall computation time using traditional methods.

Input: Arbitrary number of text files.
Output: The stop-word file plus the same number of input text files with reduced size.

Step1: Determine Stop-words list (Fox, 1992). // Define the pre-compiled a list of stop-words (i.e. Array of strings) that defined by Fox in 1992.
Step2: Read Text files as blocks of characters. // For each Text file Read its contents as blocks till reaching the end of file.

```
    d ←0 //the initial value to the size of the array for the ordinary words
    Ch () ← file Stream. Read Block (40000)    // Array of characters
      String s ←null
      For all elements in Ch
          If (Ch ≠ ‘ ‘) then
              s=s & Ch
          else
            Call Function 1
          end if
      End For
```

Step3: Print in files. // When reaching to the end of each file Print Array B in a new file with a numeric name, d=0, then go to step2 for reading the next file (in case of there is another text file).
Step4: Create stop-words file. // Print all stop-words with its repetition and no. of files that appeared on it.

End;

Function1: Check Words. // Check each word whether it matches any word listed in the stop words list or not; if the matching result is true, then increased the count of the similar word and increase the number of files for this word if it appears in this file for the first time. Else if the result equal false then do:

```
        B(d)=s
         d=d+1
Return
```

3. Hash-Index System

Searching/Matching tasks through a large dataset needs to spend a lot of time to extract the required response (either positive or false positive) from the dataset [13]. For this problem, the information about each word in the considered dataset must provide. Therefore, in this stage the first process is the creation of lookup table that contains a hash value of the first two characters of each word. The lookup table contains the unique values for each character pair regardless the case of these characters (small or capital), while the character case will maintain when putting the complete words in files. The used equation of the proposed function is:

$$H(M) = 27 \times ch_1 + ch_2 \qquad (1)$$

Where, ch_1 is the first character of the word, ch_2 is the second character of the word. As example, for the word “about”, the character “a” is equal to 1 and b equal 2 so the hashing equation for this word will be equal 27x1+2 = 29. Hence, the values of matching table are calculated using simple linear hash-index function that limits the range of searching operations from any number of files in the original dataset to 729 files from 0 to 27*26+26; the base 27 represents the number of possible characters that have the range (0 … 26), and 26 is the maximum value that represents the numeric index values represents the alphabets + other non-alphabet text characters). Using this method will arrange the input files' contents in a limited number of text files (i.e., 729 file) with names are sorted; each file holds the words have same initial pairs of characters. This step is very useful to reduce the searching time by taking uniform jumps according to the hash index of each word.
Algorithms (4) illustrate the taken steps for converting the unstructured data files to structured data files depending on the value of the first two characters of each word that loaded from the previous stage.

Algorithm (4): Hashing Algorithm (numeric method)

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.
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
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
Step2: Read Text files. // For each Text file Read all the content of this file as an array of bytes.
Step3: End Trim. // Remove spaces from the end of the file.
Step4: Remove Double Space. // Convert the encounters double spaces to single space.
Step5: 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))
Step6: 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.
Step7: 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.
Step8: Print in files. // Print the results in files that have the names of the first two characters.
End;
```

The traditional method used the same steps of the algorithm (4) except that reading all the file content as an array of strings, then using the conventional substring functions for each word to obtain the first two characters that used by the lookup table to calculate the hash value of this word.

Experimental Setup

Implementation of the proposed algorithms was using C sharp (2015) programming language; it is chosen as the implementation language for its efficiency in designing data structures and hashing function programs. The tests conducted and reported on the English alphabet (capital and small) contents, which extracted from two large sets of text document files. The first dataset was collected from a collection of books contains 3015 (researches, books and articles) belong to many libraries and websites. The size of the extracted text of this set reached 4.64 GB.

The second dataset constructed from the University of Oxford Text Archive; it collects data comprising a number of texts from different sources. It usually compiled for the purposes of linguistic research, which form 541 MB. It is designed to represent a wide cross-section of current British English [14].

Experimental Results:

In order to study the effectiveness of using enumeration and hashing methodology in text analysis system, the proposed system was tested on the two above-mentioned dataset. The conducted experiments in the area of text analysis could be evaluated from the point of view of data pre-processing. It is a methodology designed and recommended for reliable data acquisition from text files in the process of discovering the words features and reducing the files size.

The conducted statistical analysis results indicated that when using the proposed method to implement text analysis operations (i.e. Lexical analysis, Stop-words filtering, and Hash indexing) the execution time of the overall system would be reduced by 80% compared with the traditional method. The attained results are shown in Figures- 2 and 3.

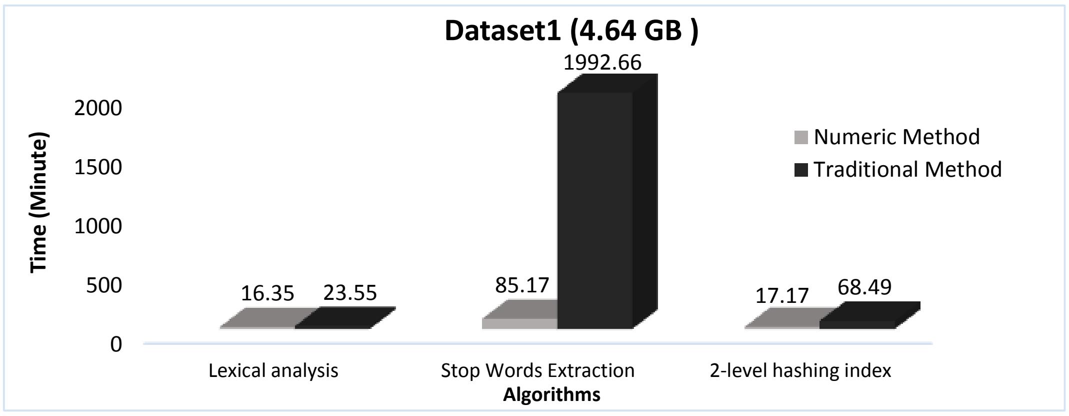

Figure 2- Traditional vs. Numeric Methods with Respect to Time for the First Dataset.

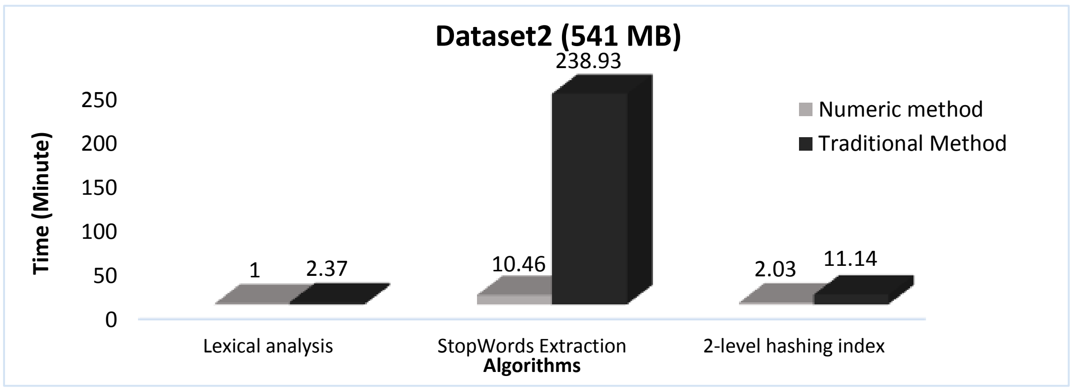

Figure 3- Traditional vs. Numeric Methods with Respect to Time for the Second Dataset.

These figures show the baseline classification performance in lexical analysis is better than the other stages depending on the noise (numbers, non-printable characters, punctuation) in a given dataset. While, the consumed time to stop-words extraction stage is increasing, dramatically, according to the number of the matching operations between each word and the stop-words list. The net profit of the execution time for each stage (excluding the time of reading and writing in files) was shown in Table-1. As a result of reducing the hidden-operations on the CPU that consume both the CPU time and memory space (i.e. converting each string to its equivalent ASCII value, doing other operations relevant to characters case, then recover the string form and display the result on the screen). The achieved speeds up results agree with the results given by other research articles [4].

Table 1- Net Time Profit Percentage for Each Stage Comparing with the Traditional Method

	Stages	**Net time Profit Percentage**
Dataset1	Lexical analysis	83.36 %
	Stop-words Extraction	99.28%
	2-level Hashing index	96.07 %
Dataset2	Lexical analysis	84.70 %
	Stop-words Extraction	94.35%
	2-level Hashing index	95.42%

Conclusion and Future Work:

This paper examined the text analysis performance and explained the rationale analysis of hash-index and enumeration methods. The starting point was the development of a unified view of these methods. The presented idea opens new possibilities to derive theoretical bounds for the performance of hashing and enumeration methods. This idea introduced to Text analysis systems by defining a new formulation of strings when converting each character in strings to a unique number using the coding system and sometimes just using the ASCII code of each character. The new technique was tested, and the results indicated it is particularly noteworthy, such that the obtained results are consistently better when compared with the traditional method. The dataset used to validate our method is sufficiently large, and at this point, it is enough to conclude that the proposed algorithm is efficient and faster than the traditional ones. Hence, it can possibly implement in all applications related to text analysis systems.

The effectiveness of the proposed method was evaluated using two relative datasets, the preliminary results are encouraging to go forward toward developing new method for multiple pattern matching/searching algorithms to meet the challenges and demands of fast and efficient searching systems, also to continue this evaluation by using other different distance measures (such as: Edit distance, Levenshtein distance... etc..) for making the search operation as fast as possible.

References

1. Green, G. M., Kantor, R. N., Morgan, J. L., Stein, N. L., Hermon, G., Salzillo, R., Sellner, M. B., Bruce, B.C., Gentner, D. and Webber, B.L. **1980**. Problems and techniques of text analysis. *Center for the Study of Reading Technical Report* no. 168. Available on: https://www.ideals.illinois.edu/bitstream/handle/2142/17541/ctrstreadtechrepv01980i00168opt.pdf?sequence=1.
2. Chakraborty, G., Pagolu, M. and Garla, S. **2014**. Text mining and Analysis: Practical Methods, Examples, and Case Studies Using SAS, *SAS Institute*, ISBN: 978-1-62959-086-8.
3. Linoff, G.S. and Berry, M. J. **2011**. *Data Mining Techniques: for Marketing, Sales, and Customer Relationship Management*. John Wiley & Sons.
4. Cox, N. **2002.** Speaking Stata: On Numbers and Strings. *The Stata Journal*, Number **3:** 314–329.
5. Cox, J. **1999.** Changing String Variables to Numeric: Update. *Stata Technical Bulletin* 49: 2, In Stata Technical Bulletin Reprints, **9**(14), College Station, TX: Stata Press.
Available on: http://www.stata.com/products/stb/journals/stb49.pdf.
6. Cox, J. and Wernow, j. **2000.** dm80: Changing numeric variables to string. *Stata Technical Bulletin* **56**: 8–12. In Stata Technical Bulletin Reprints, **10:** 24–28. College Station, TX: Stata Press.
7. Cox, J. and Wernow, J. **2000.** dm80.1: Update to Changing Numeric Variables to String. *Stata Technical Bulletin* 57: 2. In Stata Technical Bulletin Reprints, **10**: 28–29. College Station, TX: Stata Press.

8. Klint, P. **1985.** *A Study in String Processing Languages*. Springer Science & Business Media, No. 205, ISBN: 3-540-16041-8.
9. Clapson, A. **2014.** A Note on Type Conversions and Numeric Precision in SAS: Numeric to Character and Back Again. *SAS Conference Proceedings: SAS Global Forum 2014 March 23-26*, 2014, Washington, DC, 443 papers, Paper ID 1752-2014.
Available on: http://support.sas.com/resources/papers/proceedings14/1752-2014.pdf
10. Saif, H., Fernandez, M., He, Y., and Alani, H. **2014.** On Stop words Filtering and Data Sparsity for Sentiment Analysis of Twitter. *9th International Conference on Language Resources and Evaluation*. Proceedings, European Language Resources Association, pp: 810–817.
11. Silva, C. and Ribeiro, B. **2003.** The Importance of Stop Word Removal on Recall Values in Text Categorization. Proceedings of the International Joint Conference, **3**: 1661–1666. *IEEE*, In Neural Networks.
12. Fox, C. **1992.** *Lexical analysis and Stop lists. Information retrieval: Data structures and algorithms [M]*.Upper Saddle River, New Jersey: Prentice Hall, pp:102-130, and ISBN: 0-13-463837-9.
13. Kanuga, P., and Chanuhan, A. **2014.** Adaptive Hashing Based Multiple Variable Length Pattern Search Algorithm for Large Data Sets. In Data Science & Engineering (ICDSE), 2014 International Conference, pp: 130-135, *IEEE*.
14. Burnard, L., the University of Oxford Text Archive, **1976**, University of Oxford, http://ota.ox.ac.uk/catalogue/index.html.