



Improved N-gram Extraction and Substring Removal Algorithms with Efficient Dual-Pivot Sorting in Mining Data for UML Models

Bala Sundara Ganapathy Nadesan*, Dr.K.Alagarsamy

Computer Science, Madurai Kamaraj University
India

Abstract—Extracting data from raw corpus can be useful for designing software models like UML Model. Usually, the data from the raw corpus can be extracted using N-gram Algorithm and equal frequency n-gram can be removed using Statistical Substring Algorithm. In this paper, we have proposed new improved versions of N-gram Extraction Algorithm and Substring Removal Algorithm for extracting data from raw corpus. In both the proposed algorithms, efficiency can be improved by Dual-Pivot Quick Sort Algorithm and eliminating substring with equal frequency. Therefore the proposed n-gram algorithm has a time complexity of $O(n \log n)$, where n is the number of words in the input file. Similarly, the substring removal algorithm has a time complexity of $O(n \log n)$, where n is the number of word n-grams if the input file. Using these algorithms, it is evident that the automatic extraction or determination of words, compound words and collocations are useful for designing software models.

Keywords— UML Model, N-gram, Substring Removal, Raw Corpus, Dual-Pivot Quick Sort

I. INTRODUCTION

In order to develop quality software, it must be designed according to the requirements. UML diagram [1] is an ideal choice for software developers who need to demonstrate and deduce relationships, actions and connections of a software application using the Unified Modeling Language (UML) notation. The software designer must go through the software requirements specifications (SRS) and extract data for the UML Models. This could be achieved efficiently by employing N-gram Algorithm [2] [3] and Statistical Substring Algorithm [4]. But the N-gram Algorithm uses Comb Sort [5] for sorting the word N-grams and Statistical Substring Algorithm uses Radix Sort [6] for sorting the set of strings. But it is found that the efficiency of both the algorithms can be improved by utilizing Yaroslavskiy’s Dual-Pivot Quick Sort Algorithm [7], after it has been implemented as a standard sorting method for Oracle’s Java 7 Run-Time Library [8] [9] recently.

II. IMPROVED N-GRAM EXTRACTION ALGORITHM

The N-gram extraction algorithm can pull out n-grams for an arbitrary large number of n with a sensible memory size in a practical calculation time. The algorithm first gets a table of alphabetically sorted substrings of a text string [10] and then calculates the frequency of n-grams for all the existing n character strings from the sorted strings for a specific number of n . The improved N-gram extraction algorithm is given in Algorithm 1.

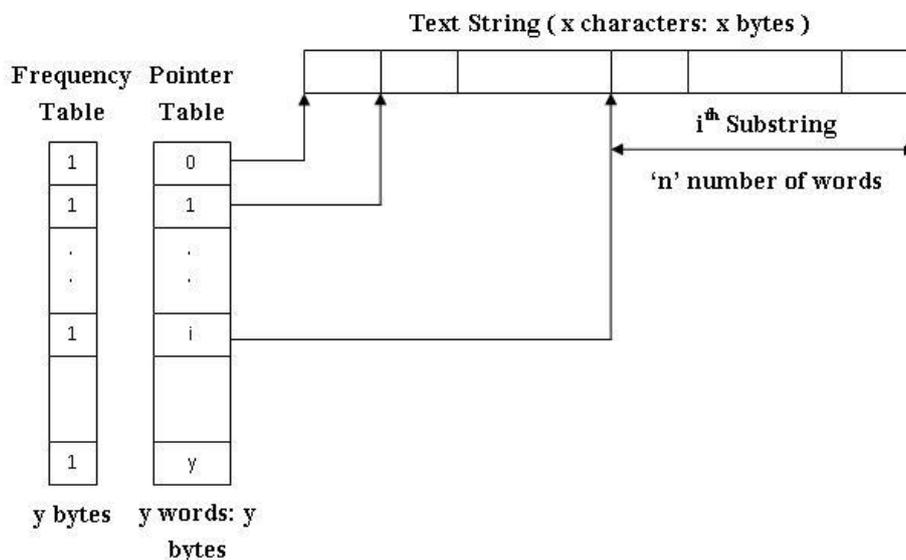


Figure 1: Text string, Pointer table and Frequency table to sub-string.

When a text is given it is stored in a computer as one long character string. It may include spaces, newlines, sentence precincts, paragraph precincts and so on if they are regarded as components of text. The string length and the number of words on the string are calculated. When a text is composed of 'x' characters it occupies 'x' byte memory because an ASCII character is encoded by 8 bit code. When a string is composed of 'y' words, we need to prepare another table of the size (y), each entry of which keeps the pointer to a substring of the text string. This is illustrated in Figure 1.

In natural language, the number of words is lesser than the number of characters in any particular string; hence the value y must be smaller than the x. Therefore space complexity of the above operation is $O(x+y)$. Here, x is the number of characters on the string and y is the number of words on the string.

```
begin
  Read the whole document as a single string
  Calculate the length and number of words of the string
  Allocate length amount of memory for n-gram calculations
  Prepare number of words size of another table to keep the pointer to a substring
  Read n-gram size from the user
  while not an end of a string do
    begin
      Extract n number of words as a substring i from the source string
      Store this substring pointer address in the pointer table of location i
      Remove first word from the source string and update it
    end {while not end of string}
  Sort the extracted substring by utilizing the pointer in the pointer table
  Prepare number of words size of another table to keep the frequency of the substring
  Initialize frequency count to 1 to all the entries of frequency table
  while not end of pointer table
    begin
      Compare two adjacent entries in the pointer table
      if they are same then
        Remove one entry from both the pointer table and frequency table
        Add 1 to count of the respective entry in the frequency table
      else
        Move to the next entry in the pointer table
      endif
    end {while end of pointer table}
  end
```

Algorithm 1. Improved N-gram Extraction Algorithm

In order to calculate n-gram, let us fix n to a certain number. A substring pointed by (i-1) is composed of n number of words from i^{th} positioned word (see Figure 1). The first substring is the text string which starts from first word to n^{th} word, and the second substring is the string which starts from the second word and ends at the n number of words of the text string. Similarly, the last substring is the final n words of the text string.

As the text size is y words a pointer must have at least p bits where $2^p \geq y$. In our program, we set $p = 32$ bits so that we can accept the text size up to $2^{32} = 4$ giga words. The pointer table represents a set of y substrings. We need to apply the dictionary sorting operation to this set of y substrings. It is performed by utilizing the pointers in the pointer table. We have to employ Yaroslavskiy's dual pivot Quicksort, which is an improved version of quick sort. The sorting time is the order of $O(y \log y)$. When the sorting is completed the result is the change of pointer positions in the pointer table, and there is no replacement of actual words. As we are interested in n-grams of n less than 255, actual sorting of substring is performed for the leftmost 255 or less words of substrings.

The calculation of n-gram frequency table is done by using the pointer table. Allocate number of word size of frequency table for calculating frequency of the substring. Hence, the memory space required for this proposed algorithm is $x + y + y = x + 2y$ bytes. The space complexity of this algorithm is $O(x+2y)$, where x is the number of characters on the string and y is the number of words on the string.

Initialize the frequency count to 1 for all the entries of the frequency table. Compare two adjacent entries of the pointer table. If they are same, increase the frequency count and remove single entry from the pointer table as well as frequency table. Thereby, the pointer table and frequency table size is decreased. This process is continued until all the entries of the pointer table are evaluated. These operations result in the n-gram table of the given text. The running time of the algorithm is calculated as below.

The first while loop needs to be executed number of words(y) amount of time. The second while loop also needs to be executed y number of times. But the sorting algorithm needs $O(y \log y)$ time. Therefore time complexity of this algorithm is $O(y \log y)$, where y is the number of words available on the string.

III. DETERMINING POINTER TABLE SIZE

Let us consider the sentence “This too will pass away.” The number of words available on this sentence is five. Therefore, one can extract n-gram of size maximum to five. The following table illustrates the number of n-grams possible on this sentence.

TABLE I
N-GRAM LIST

Source String	This too will pass away.				
n-gram	1-gram	2-gram	3-gram	4-gram	5-gram
	This	This too	This too will	This too will pass	This too will pass away.
	too	too will	too will pass	too will pass away.	
	will	will pass	will pass away.		
	pass	pass away.			
	away.				
No. of n-gram	5	4	3	2	1

Hence, the maximum number of substring of some specified size is the number of words on the sentence. Therefore, number of words amount of pointer table is sufficient for handling substrings.

IV. NEED FOR BETTER STRING SORT ALGORITHM

The Nagao and Mori’s N-gram algorithm uses comb sort (also called Dobosiewicz sort) [11] for sorting substring which is available as a pointer table. The algorithm claims that the sorting time of comb sort is $O(n \log n)$, where n is the number of sub-stings available for sorting. But the citation for this claim is a paper on GPU processing [12] that mentions this result offhand without providing any analysis or citation. One of the original papers on comb sort doesn't even attempt to prove a worst-case bound, and in fact not a single source we have found has attempted to analyze the worst-case or even average-case complexity of comb sort. Therefore, we are going to use most efficient [13] latest quick sort algorithm called Dual Pivot Quicksort. It was proposed by Yaroslavskiy in 2009, and recently, it was chosen as standard sorting method for Oracle’s Java 7 runtime library. The decision for the change was based on empirical studies showing that on average, the new algorithm is faster than the formerly used classic Quicksort. The average case analysis of this new sorting algorithm showing is $O(n \log n)$ for a random permutation of length n . The advantages of this algorithm are as follows:

- The Dual-Pivot Quicksort algorithm is faster than classical quick sort algorithm
- This algorithm follows "divide and conquer" strategy
- Usage of two pivot elements instead of one
- More effective sorting procedure
- Importantly, it was implemented as a standard sorting method for Oracle’s Java 7 Runtime Library.

V. NEED FOR BETTER SUBSTRING REMOVAL ALGORITHM

The Statistical Substring Reduction is a mechanism, which can remove equal frequency n-gram substrings from an n-gram set. For example, if n-grams “the premium member can update”, “the premium member can update the program” and “the premium members can update the programs” occur five times in a corpus, the first two should be removed from the n-gram set, for it being the substring of the third n-gram.

When the initial n-gram set contains n n-grams, traditional SSR algorithm has an $O(n^2)$ time complexity (Han et al., 2001), and is actually intractable for large corpus. Le Zhange et al., (2005) proposed an algorithm for statistical substring reduction in linear time. In this algorithm, all steps have a time complexity of $O(n)$ except two steps(3 and 9), which perform sorting on n statistical strings. The sorting was implemented with radix sort(most preferably Most Significant Digit Radix Sort), an $O(n)$ operation. Therefore this algorithm has an ideal time complexity of $O(n)$. If the strings are of finite size, the radix sort algorithm runs in $O(n)$ asymptotic time. But the string extracted from raw corpus may not be in fixed size. So the entire Radix Sort procedure takes $O(nk)$ time, where n is the number of elements to sort and k is the number of characters in each element. If the string length are all distinct, then $\log n \geq k$, radix sort on distinct strings therefore has a time complexity of $O(n \log n)$. The following are the some of the pitfalls of radix sort that motivate to consider better sorting algorithm for statistical substring algorithm.

1) The pitfall for MSD string sort is that it can be relatively slow for sub-arrays containing large numbers of equal keys. The keys that fall into a small range and small arrays where MSD string sort runs slowly.

2) The main challenge in getting maximum efficiency from MSD string sort on keys that are long strings is to deal with lack of randomness in the data. Typically, keys may have long stretches of equal data, or parts of them might fall in only a narrow range. For example, an information-processing application for student data might have keys that include graduation year (4 bytes, but one of four different values), state names (perhaps 10 bytes, but one of 50 different values), and gender (1 byte with one of two given values), as well as a person’s name (more similar to random strings, but probably not short, with non-uniform letter distributions, and with trailing blanks in a fixed length field). Restrictions like these lead to large numbers of empty sub-arrays during the MSD string sort.

3) Like quicksort, MSD string sort partitions the array into sub-arrays that can be sorted independently to complete the job, but it partitions the array into one sub-array for each possible value of the first character, instead of the two or three partitions in quicksort.

VI. IMPROVED SUBSTRING REMOVAL ALGORITHM

In order to consider the above said problems, instead of radix sort, we are going to use the latest Dual-Pivot Quick Sort for our Substring Removal Algorithm. This improved substring removal algorithm is given in Algorithm 2.

```
Get the set of input strings in X
Sort all statistical strings in X in ascending order using Dual Pivot Quick Sort algorithm
for i=1 to n - 1 do
    if  $X_i$  is a substring of  $X_{i+1}$  then
        Remove  $X_i$  from string set
        Reduce n by one
for i=1 to n do
     $X_i = \text{reverse}(X_i)$ 
    Sort all statistical strings in ascending order using Dual Pivot Quick Sort algorithm
for i=1 to n - 1 do
    if  $X_i$  is a substring of  $X_{i+1}$  then
        Remove  $X_i$  from string set
        Reduce n by one
for i=1 to n do
     $X_i = \text{reverse}(X_i)$ 
    output  $X_i$ 
```

Algorithm 2. Improved Substring Removal Algorithm

This Substring Removal Algorithm uses Dual-Pivot Quick Sort method for sorting all statistical strings and reverse of the same in ascending order. After sorting the string, two consecutive strings are compared. If i^{th} string is a substring of $(i+1)^{\text{th}}$ string, later will be deleted and n will be reduced by one. The same process will be repeated with the sorted reverse string of the same. This way we can remove equal frequency n-gram substrings from an n-gram set. In addition to the Dual Pivot Quick Sort algorithm efficiency, the Substring Removal algorithm efficiency is also increased by removing the substring from the string set and reducing the number of strings (n) available for processing in each iteration. Therefore the Substring Removal algorithm has a time complexity of $O(n \log n)$.

VII. CONCLUSIONS

We have developed new Word N-gram Extraction Algorithm for large corpus, that has time complexity of $O(n \log n)$ and the Substring Removal Algorithm, that has time complexity of $O(n \log n)$. Therefore, the maximum time complexity is $O(n \log n)$ making it a reasonably efficient method for extracting data from raw corpus. Experimentation and evaluation of the above suggested algorithms to find the quality of design model that leads to the quality of the software could be undertaken in future. As a continuation of this research, list of nouns could be extracted from raw corpus for constructing class diagram. Specific research can also be conducted for different types of natural languages.

REFERENCES

- [1] Bogdan D. Czejdo, Rudolph L. Mappus IV, Kenneth Messa, "The Impact of UML Class Diagrams on Knowledge Modeling, Discovery and Presentations", *Journal of Information Technology Impact*, 2003, Vol. 3, No. 1, p. 25-44.
- [2] William B. Cavnar and John M. Trenkle, "N-Gram-Based Text Categorization", *Proceedings of the Third Symposium on Document Analysis and Information Retrieval*, 1994.
- [3] Makoto Nagao, Shinsuke Mori, "A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases from Large Text Data of Japanese", *International Conference on Computational Linguistics*, In COLING-94, 1994, p. 611—615.
- [4] Xueqiang LÜ, Le Zhang, and Junfeng Hu. Statistical Substring Reduction in Linear Time. In *Proceeding of the 1st International Joint Conference on Natural Language Processing (IJCNLP-04)*, Sanya, Hainan Island, China, March 2004.
- [5] Stephen Lacey, Richard Box: *Nikkei BYTE*, November 1991, p.305-312.
- [6] Juha Kärkkäinen, Tommi Rantala, "Engineering Radix Sort for Strings", 15th International Symposium, SPIRE 2008, Melbourne, Australia, November 10-12, 2008, p 3-14.
- [7] Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch, "Dual-Pivot Quicksort", February 16, 2009.
- [8] Sebastian Wild, Markus Nebel, Raphael Reitzig, Ulrich Laube, "Engineering Java 7's Dual Pivot Quicksort Using MaLiJAn", *Society for Industrial and Applied Mathematics(SIAM)*, 2013, p-55-69.
- [9] Sebastian Wild and Markus E. Nebel, "Average Case Analysis of Java 7's Dual Pivot Quicksort", *Algorithms – ESA 2012, 20th Annual European Symposium*, Ljubljana, Slovenia, September 10-12, 2012, p. 825-836.

- [10] Ioannis Kanaris, Konstantinos Kanaris, Ioannis Houvardas, Efstathios Stamatatos, “ Words Vs. Character N-Grams For Anti-Spam Filtering”, International Journal on Artificial Intelligence Tools Vol. XX, No. X (2006), p. 1–20.
- [11] Wlodzimierz Dobosiewicz., “An efficient variation of bubble sort”, Information Processing Letters 11(1):5-6, August 1980.
- [12] Vincent Garcia and Frank Nielsen, “Searching High-Dimensional Neighbours: CPU-Based Tailored Data-Structures Versus GPU-Based Brute-Force Method”, A. Gagalowicz and W. Philips (Eds.): MIRAGE 2009, LNCS 5496, 2009, p. 425–436.
- [13] Jon L. Bentley, Robert Sedgewick, “Fast Algorithms for Sorting and Searching Strings”, Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, January, 1997, p.1-10.