Algorithm for Fast Finding High-Frequency Strings from Large-Scale Corpus[1]

Haijun Zhang
School of Computer Science and Technology Xinjiang Normal University, Urumqi China, 830054
Email: ustczhj@mail.ustc.edu.cn

***Abstract*—In high-frequency string extraction, there exists enormous time and memory waste in taking statistics of tremendous low-frequency strings, which causes low efficiency. Based on the incremental n-gram model, this paper puts forward Hierarchical Pruning Algorithm (HPA) to filter out low-frequency garbage strings and to extract candidate repeats for reducing I/O reading-writing times and enhancing efficiency of memory usage. On the basis of candidate repeats, external sort method is applied to merge all of them in order to obtain the final repeat set. For improving the efficiency of candidate repeats merging, this paper proposes to employ improved Radix Sort method to process strings in *O(dn)*. With 32 gigabyte plain text corpus, experiments show that the relationship between I/O reading-writing times of HPA and the corpus size is nearly linear, and the algorithm can efficiently extract repeats from corpus whose size is much larger than that of memory.**

***Index Terms*—repeats, hash table, low-frequency string, hierarchical pruning algorithm**

I. INTRODUCTION

With the progress of era, Chinese new words and phrases come forth constantly. For example: "非典"(SARS), "超女"(super girl) and so on. New words and phrases provide lots of convenience for conversations; at the same time, they also bring some difficulties[1] for automatic language processing. As the main feature of new words and phrases is the high frequency of occurrence, they belong to High-Frequency Strings (HFS). If HFS in Large-Scale Corpus (LSC) can be extracted quickly, we can efficiently find lots of new words and phrases, which are very helpful in Natural Language Processing (NLP). HFS extraction is an important technology for the fields of NLP such as Chinese Unknown Words Identification (UWI), information retrieval and information extraction etc. For example, the extracted HFS from LSC is regarded as candidate words [2-4] in repeat-based UWI.

Let Σ be a limited character set, and S be a character sequence $S = c_1c_2c_3\cdots c_n$, whose length is n. $S[i]$ denotes the ith character in S where $1 \le i \le n$, and $S[i, j]$ denotes a string in sequence S. $T = S_1\#S_2\#S_3\cdots\#S_m$ denotes a text corpus, where "#" is a Symbol of Text End (STE) used to indicate the end of a piece of text, for example, "!", "?"and "。" etc. can be taken as STEs. Let $R = c_1c_2\cdots c_k$ be a string, and there is no STE in R. If there are at least two positions p_1 and p_2 in corpus T, which meet $R=T[p_1, p_1+k-1]=T[p_2, p_2+k-1]$, string R is called repeat[5]. If the occurrence frequency of string R is equal to or higher than predefined threshold λ, R is called high-frequency string. The extraction of high-frequency strings is to find various lengths of repeats whose frequencies are equal to or higher than λ. For example, if there is conditions: $\Sigma = \{'a','b','c','d'\}$, S="*adbca*", and T="*adbca#db#ccc*", we have $S[2]$= "*d*", $S[1,4]$= "*adbc*" and R="*db*" with frequency 2. If $\lambda = 2$, the string R="*db*" will be extracted as a high-frequency string. Here the so-called high-frequency is a relative definition.

The main difficulty for HFS extraction is the statistics of repeats, in which most of time and memory are spent in taking statistics of low-frequency strings resulting in low efficiency and high memory usage, which is difficult to meet the requirement of fast extraction for repeats.

In this paper, we put forward a pragmatic HFS extraction algorithm, in which the mass of memory usage for once is converted to trifle usage for multi-times through scanning text corpus multi-times in order to effectively process LSC whose size is greater than that of memory. The specific ideas are as follows. First, we construct Pruning Characters Set (PCS) by scanning corpus once. The PCS includes characters whose frequency is less than the predefined threshold λ, which is used to filter out low-frequency garbage strings at the level of character. Second, hierarchical pruning method is employed to remove low-frequency garbage strings based on incremental n-gram model. That is, when extracting long repeats, the adjacent-length short repeat set as well as PCS is used to filter them for further improving efficiency. And finally, the external sort method is adopted to achieve the final set of repeats of the whole corpus.

The remainder of this paper is organized as follows: section 2 describes the Hierarchical Pruning Algorithm (HPA), which is used to filter and extract high-frequency strings; section 3 carries out experiments and discussions;

[1] This work is supported by Xinjiang Uygur Autonomous Region Natural Science Foundation Under grant NO. 2012211A057.

doi:10.4304/jsw.9.8.2154-2159

section 4 introduces the related studies; in the end of this paper, we present the conclusions and future works.

II. Related Studies

So far there are many repeat extraction algorithms, which typically include sequitur algorithm, the suffix index based algorithm and incremental n-gram algorithm.

The main idea of sequitur algorithm[8] is to employ production rules, which represent repeat structure in corpus, and to construct rule library of grammar and extract repeats from rule library. The current rule library must be adjusted when a character is read. The rules in the rule library of grammar must satisfy two principles as follows: (1) any rule made of adjacent characters must be unique; (2) the application times of any rule must be more than once. Sequitur algorithm commences with the starting-point of corpus, reads character one by one, and adds character to the right end of current string. After that, the rule library should be adjusted in order to make all rules satisfy the above two principles. Sequitur algorithm is used to find hierarchical structure of corpus quickly and both its time and space complexity is *O(n)*. However, sequitur algorithm might omit some repeats with special structure[3], which is the main problem that affects its application in repeat extraction.

Suffix index-based algorithm includes suffix tree and suffix array[9]. Let $T = t_1 t_2 \cdots t_n$ denote the text corpus, and $C_i(T) = T[i,n]$ denote the suffix starting from index *i* of corpus *T*. For corpus $T = t_1 t_2 \cdots t_n$, there are *n* non-empty suffixes: $C_1, C_2 \cdots C_n$, such as $C_1 = t_1 t_2 \cdots t_n, C_2 = t_2 t_3 \cdots t_n, C_3 = t_3 t_4 \cdots t_n$ etc. The all non-empty suffixes can construct the suffix tree from which the repeats can be extracted by finding the longest common sub-strings. However, the constructing time has a non-linear relationship with the size of character set. Moreover, as the space needed to construct suffix tree is $O(n \log |\Sigma|)$, it will be inefficient to process language of large character set, such as Chinese. Someone had proposed to replace the suffix tree with the suffix array. So far, the suffix array algorithm, whose complexity for both time and space is *O(n)*[10], is a efficient algorithm used to extract repeats. The main problem of suffix index based algorithms is that they need corpus size much less than that of memory, but in practice the corpus size is generally much greater than memory capacity, which results in the failure of this kind of methods.

For the naïve incremental n-gram algorithm, it first extracts two-character strings within the scope of the whole corpus, then extracts three-character strings, until n-character strings. This algorithm is simple and easy to implement, but inefficient. The time complexity of this algorithm is $O(n^2)$, which will cause difficult to process LSC.

According to the above discussion, the three kinds of algorithms cannot effectively deal with corpus with size greater than memory capacity. Researchers have tried a number of corpus partition methods to resolve this problem. Martin et al.[11] presented a viable corpus partition method, in which the corpus was scanned multi-times and for each scanning all the suffixes beginning with certain character were added to the suffix tree. The time complexity of constructing tree is $\Theta(n^2)$ for this method. Chen et al.[12] first sequentially scanned the corpus and generated the temporary files to store all the suffixes, then sorted them and established suffix trees for each group respectively. This method can handle LSC, but the workload of suffix sort is very huge. Based on Martin's algorithm, Schumann and Stoye[13] put forward clustered algorithm and employed hash function to locate the position of sub-tree. Clustered algorithm is faster than [11], but it demands several-times of memory available, at least several more times volume than the size of corpus. Clifford and Sergot[14] proposed the distributed suffix tree, which copied corpus to different nodes and used Martin's method to construct suffix tree beginning with certain character for each node.

Tian et al.[15] proposed Partition and Write Only Top Down (PWOTD) algorithm, in which they used inverted index to store all the positions of characters in corpus. When constructing suffix tree, all suffixes beginning with a certain character were obtained based on the appearing positions of character in the inverted index and added into suffix tree in turn. Gong et al.[5] made some improvements on PWOTD method. They divided all suffixes of LSC into many groups according to the first characters, and constructed an independent suffix tree for each group. Based on the above operations, the final set of repeats was obtained by merging all repeats from each independent suffix tree. The inverted index was used to partition the corpus. If some character *x* appears in position *p*, the *Postlist* of character *x* will be added the suffix beginning with character *x* in position *p*. The difference between Gong's and Tian's is that the *Postlist* of Gong's directly stores the suffixes rather than the emergence positions, by which the algorithm can reduce tremendous overhead of I/O operations for reading the suffixes into memory.

Though the above two algorithms can effectively decrease the memory usage, they need frequent and tremendous I/O operations. For Tian's method, it needs rather frequent I/O operations to load suffixes into memory because *Postlist* stores only the positions of characters; for Gong's, it requires reading the corpus as many times as the size of the character set, because obtaining suffixes beginning with a certain character needs to scan the whole corpus once. When the size of corpus is becoming greater and the memory cannot accommodate the suffixes beginning with a single character, the times of I/O operations will increase exponentially, which leads to difficulty in dealing with LSC.

III. Repeat Extraction Based On Hierarchical Pruning Algorithm (HPA)

The aim of HPA is to filter out as many low-frequency garbage strings as possible in order to improve the memory and time efficiency for repeats extraction.

Hierarchical pruning algorithm includes three steps, low-frequency character pruning, cascade pruning and candidate repeats merging.

A. Low-Frequency Strings Filtration Based on Low-Frequency Character Pruning

For incremental n-gram model, it will inevitably generate a large number of garbage strings whose frequency is less than threshold if directly used to extract repeats without any improvements. The garbage strings waste large mass of memory and debase the efficiency of repeat extraction. According to the characteristic of repeats, for string $S = c_1c_2 \cdots c_k$, if there is $f(S) \geq \lambda$, there must be $f(c_1) \geq \lambda, f(c_2) \geq \lambda, \cdots, f(c_k) \geq \lambda$, where function $f(x)$ indicates the occurrence frequency of string *x* and value λ denotes the predefined threshold used to restrict the minimum frequency of repeats.

However, this proposition is difficult to judge whether a string is a low-frequency garbage string or not. The equivalence of the above proposition is if there is $f(c_i) < \lambda, c_i \in \Sigma$, there must exist $f(S) < \lambda$. Through scanning the corpus once, we can obtain the set of characters which satisfy the condition $f(c_i) < \lambda, c_i \in \Sigma$. The set of these characters is called Pruning Character Set (PCS) $\sum_0$, which is used to pre-filter low-frequency strings. For any character C_x in candidate string *S*, if there is $c_x \in \sum_0$, the string *S* can be discarded because there must be $f(S) < \lambda$. According to the method proposed above, we can accomplish the filtration of low-frequency strings at the level of character and can effectively reduce the usage of memory by simple and fast operations.

For example, giving candidate string S ="自然语言", if '然' $\in \sum_0$, then f("自然语言 ") $< \lambda$. As a result, the candidate string *S* will be discarded without follow-up processing.

B. Low-Frequency Strings Filtration Based On Cascade Pruning

By studying the relationship among strings, we find that long strings can be filtered by adjacent-length short strings. For example, we can employ 2-character repeats to filter 3-character candidate strings. The proof is described as follows.

If the length and frequency of string *S* are *k* and *m* respectively, for string $X = c_iS$ or $X = Sc_j$ whose length and frequency are *k+1* and m' where $c_i \in \sum, c_j \in \sum$, there must be $m' \leq m$. i.e., the frequency of a string is lower than or equal to the frequency of its sub-string. Further, if the frequency of *S* is less than predefined threshold λ, i.e. $m < \lambda$, the frequency of *X* must be also less than λ, i.e. $m' < \lambda$. It is obvious that if sub-string *S* is a garbage string whose frequency is less than λ, string *X* must be a low-frequency garbage string. According to above discussions, it is feasible to filter candidate strings by adjacent-length short strings whose frequency is less than threshold λ. Here λ is the predefined threshold used to restrict the minimum frequency of high-frequency strings.

Let set Ω denote the set of all the strings with length *k* in corpus, set $\overline{\varphi}$ denote the set of strings with frequency bellow threshold λ in set Ω, and set $\varphi = \Omega - \overline{\varphi}$ denote the set of repeats with frequency above or equal to λ in Ω. Because there exists $R \in \Omega$ and $\varphi \cap \overline{\varphi} = \phi$, one of the conclusions between $R \in \varphi$ and $R \in \overline{\varphi}$ must be held. According to this mutex relationship, as there exists $|\varphi| << |\overline{\varphi}|$ when $k \geq 2$, we consider replacing $\overline{\varphi}$ with $\overline{\varphi}$ to perform filtration of low-frequency strings in order to save memory and accelerate repeat extraction. The filtering principle is that, for string X whose structure is $X = c_iS_1$ and $X = S_2c_j$ where $C_i, C_j \in \sum$ and $S_1, S_2 \in \Omega$, if $S_1 \in \varphi$ and $S_2 \in \varphi$, string X is taken as a candidate repeat, otherwise discarded as a garbage string.

For example, concerning candidate string X ="自然语言", there exists S_1 ="然语言" and S_2 ="自然语 ". If $S_1 \in \varphi$ and $S_2 \in \varphi$, string X ="自然语言" will be regarded as a candidate repeat, otherwise, it will be discarded immediately.

C. The Repeats Merging Based on External Sort

To process corpus whose size is much greater than that of memory, we need to partition the corpus into blocks to find repeats within memory capacity. In this paper, the corpus is simply divided into blocks whose size is below a certain scale, and the repeats are extracted from each block one by one. Based on the two pruning methods discussed in previous sections, we can effectively remove the low-frequency strings and facilitate the follow-up works.

The candidate repeats from each block are stored in an independent temporary file to reduce memory usage, but they are only a part of the entire repeats of the whole corpus. Therefore all candidate repeats from blocks need to be merged to achieve the final set of repeats. In this paper, external sort method is employed to merge all candidate repeats.

The external sort includes two stages, internal sort and external merging. We employ an efficient method to enhance the speed of internal sort. For incremental n-gram model based repeat extraction, it only uses a fixed-length window to scan the whole corpus for a certain length of repeats. According to this characteristic, we propose to use the Radix Sort to improve the efficiency of internal sort. Radix Sort[6, 7], whose time complexity is *O(dn)*, has higher efficiency, but it is only suitable for sorting numerical data with identical digit. For applying this method to strings, we consider taking integer arrays, whose length is identical to that of Chinese strings, to represent Chinese strings. As each string is corresponding to a unique integer array, if radix sort for integer arrays comes into realization, so does for

Chinese strings.

D. The Implementation Frame for Hierarchical Pruning Algorithm

According to the foregoing descriptions of HPA, here gives the implementation frame of the algorithm, shown in Figure Ⅰ.

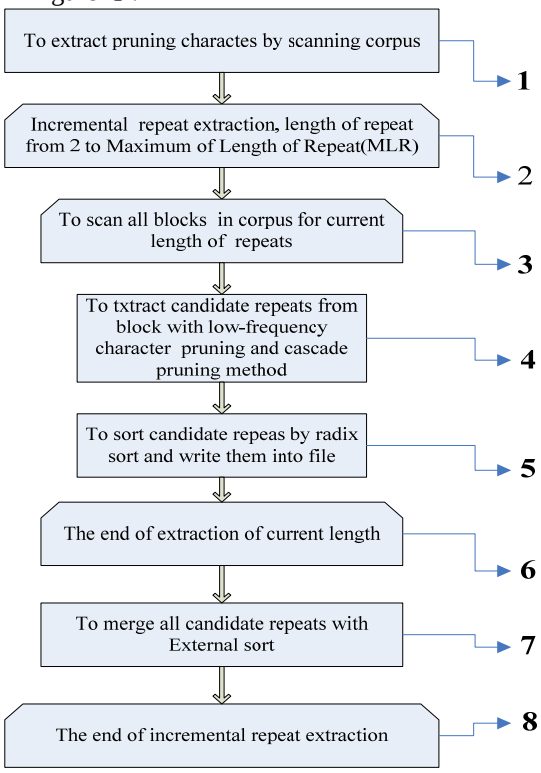

Figure Ⅰ. Frame of High-Frequency String Extraction Algorithm

Module 1 is used to scan the whole corpus to obtain the pruning characters whose frequency is less than predetermined threshold λ ; Module 2 is used to control loop of incremental n-gram algorithm to obtain various lengths of repeats through scanning the whole corpus multi-times. The modules from module 3 to module 6, being inner loop, are used to extract a certain length of candidate repeats from every text blocks in the whole corpus. Module 7 is used to merge all candidate repeats of a certain length to obtain the final repeat set by external sort method, and the final repeat set of a certain length is taken as cascade pruning set used to filter adjacent-length long strings. When the entire repeats of the corpus are obtained, the total works are done.

E. Analysis for Time Complexity of HPA

The time consumption of HPA consists of three parts. The first comes from module 1 in which the PCS is constructed through scanning the whole corpus once; the second is from extracting candidate repeats in the whole corpus and storing them in temporary files after sorting; the last comes from merging all candidate repeats to constitute the final repeat set by external sort.

For the first part, let the size of the whole corpus be N, as the time complexity of retrieval and insertion of hash structure is $O(1)$, the time consumption of this part is: $N\times O(1)+Fr(N) = O(N)+Fr(N)$,where $Fr(N)$ is the time complexity of reading data of size N from files.

For the second part, the time consumption is composed of candidate repeats extracting, sorting and storing. For length k, the time consumption of candidate repeats extracting is $N\times O(1)+Fr(N)=O(N)+Fr(N)$ because the frequency statistics of strings is based on the hash structure. Let the amount of candidate repeats be M, and the maximum of M is $N-k+1$, where k is the length of current extracting repeats. By hierarchical pruning, there must exist $M<<N$ and $O(M) <<O(N)$. As the efficiency of radix sort is $O(dn)$ for data with size n and length d, the time consumption of sort for candidate repeats of length k is $O(kM)$[17]. After accomplishing sorting, the candidate repeats will be written into temporary files. As a whole, the above operations need to be performed $K-1$ times, where K is the Maximum of Length of Repeats (MLR). So the total time consumption of this part is:

$(O(N)+Fr(N)+O(kM))\times(K-1)+\gamma\times Fw(M)$
$=O(N)+(K-1)\times Fr(N)+\gamma\times Fw(N)$,

where $Fw(N)$ is the time complexity of writing data of size N into files and γ needs to be determined by experiments.

For the third part, it merges all candidate repeats by external sort, which needs lots of I/O reading-writing operations. As the time consumption has close relations with the size of corpus and the length of strings, it cannot be evaluated independently. We give the total time consumption for this part as: $\alpha\times Fr(N)+\beta\times Fw(N)$, where the parameters α, β are determined by experiments.

So the time complexity of the high-frequency string extraction is the sum of above 3 parts, that is:

$O(N)+Fr(N)+O(N)+(K-1)\times Fr(N)+\gamma\times Fw(N)+\alpha\times Fr(N)+\beta\times Fw(N)$
$=O(N)+K\times Fr(N)+\alpha\times Fr(N)+(\gamma+\beta)\times Fw(N)$
$=O(N)+K\times Fr(N)+\alpha\times Fr(N)+\beta'\times Fw(N)$ (1)

In formula (1), the parameters K , α , β' are the times of reading or writing the whole corpus. As the time complexity of I/O reading-writing is much greater than that of memory-based operations, the times of I/O reading-writing becomes the key to extract high-frequency strings. If an algorithm can effectively reduce the times of I/O reading-writing in extracting strings, it will achieve better performance.

IV. Experiments and Discussions

A. The Conditions of Experiments

Based on the above descriptions of HPA, we implement the repeat extraction algorithm in computer. The computing environment is as follows: the main frequency of CPU is 2.66GHz, the memory size is 2GByte and the operating system is Windows XP. The corpus used in experiments is provided by Sogou Lab and it is composed of tremendous web pages in compressing style. On this basis, we obtain 32 gigabyte plain text corpus from about 800 gigabyte web pages. The predefined frequency threshold λ is 100 and MLR is 10 in experiments.

B. Analysis and Discussion on Experimental Data

In order to analyze the effects of HPA, taking None Filtration (NF) repeat extraction method as the baseline, we carry out a number of parallel experiments with incremental size of corpora. The results are shown in Table Ⅰ.

TABLE Ⅰ.
COMPARATIVE EXPERIMENTS BETWEEN TWO METHODS WITH INCREMENTAL SIZE OF CORPORA

Size (GB)	method	reading bytes	writing bytes	FR
1	NF	30597850261	19945043593	91.9%
	HPF	12264013408	1612032424	
2	NF	69105021161	47532297464	90.1%
	HPF	26284318853	4712245955	
4	NF	149180690491	105981792558	87.9%
	HPF	56020756684	12822498097	
8	NF	320093118126	234164934675	85.6%
	HPF	119697130280	33776558000	

Note: NF denoting None Filtration method, HPF denoting Hierarchical Pruning Filtration method and FR denoting Filtration Ratio.

From Table Ⅰ, the use of HPA can significantly reduce bytes of I/O reading-writing compared with the baseline method for the same size of corpus. As the difference of writing bytes between two methods can well measure the function of HPA, we define Filtration Ratio (FR) to facilitate analysis as follow.

FR=(Writing bytes of NF method – Writing bytes of HPF method)/(Writing bytes of NF method)×100% (2)

According to the data of FR in Table Ⅰ, the method based on HPA can filter more than 85% of low-frequency strings out, which shows that the filtration effect of HPA is very significant. However, FR will gradually decrease with the growth of corpus size. We think it reasonable because the increment of the corpus size brings both higher complexity of character combinations and more low-frequency strings into the set of candidate repeats. It is just the low-frequency strings which are not filtered out that decreased the filtration effect of HPA. We believe that FR will remain constant if the corpus size is large enough. For middle-scale corpus, the function of HPA is very significant.

To verify the overall performance of HPA, we carry out some comparative experiments with incremental size of corpora. The data are shown in Table Ⅱ.

TABLE Ⅱ.
DATA OF EXPERIMENTS OF I/O OPERATION

Size (GB)	reading bytes	writing bytes	reading times	Writing times	grads
1	12256691564	1612032136	11.4	1.5	----
4	56013080572	12822437898	13.0	3.0	0.533
6	87476205463	22809723148	13.6	3.5	0.300
8	119697130280	33776558000	13.9	3.9	0.150
12	187937434348	58817845729	14.6	4.6	0.175
16	258507654500	86408517723	15.0	5.0	0.100
20	332442393465	116739061731	15.4	5.4	0.100
26	445600056347	165308069442	15.9	5.9	0.083
32	561839627606	216958530202	16.3	6.3	0.067

Note: reading times denoting the times of reading the whole corpus and writing times denoting the times of writing the whole corpus

As can be seen in Table Ⅱ, the times of I/O reading-writing is increasing with the growth of corpus size, but growth rate gradually slows down. From the trend of grads, we think that the writing times will converge to a constant when the size of corpus becomes large enough. It means that all kinds of combinations among characters have all emerged for a certain frequency threshold and the amount of repeats will keep invariant with the growth of corpus size.

Moreover, for each size of the corpus, the reading times is associated with the writing times, and the difference between them is MLR (in this paper is 10). i.e., there will exist $\alpha = \beta'$ and K=MLR in formula (1). Why? To obtain all repeats, the corpus must be scanned MLR times, while the redundant reading times is caused by reading the extra written data. The relationship between reading times and writing times is shown in Figure Ⅱ, which can obviously reveal above conclusion.

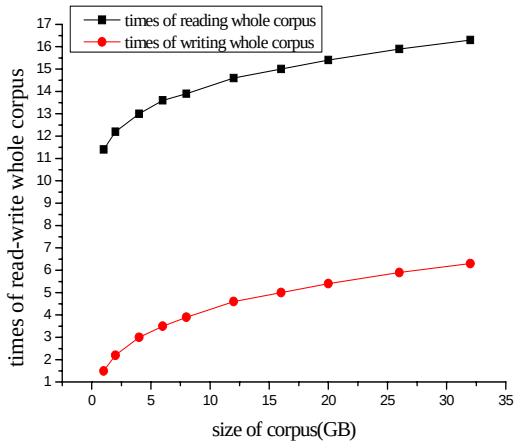

Figure Ⅱ. Relationship between I/O Reading-Writing Times and Incremental Size of Corpora

From Figure Ⅱ, the relationship between the reading times (or writing times) and the size of corpus is nearly linear (when size greater than 12 gigabyte), and the grads between them is very small, about 0.1, even much smaller.

C. Comparisons with Other Works

The method of Gong et al.[5] is a classical method used to extract repeats from LSC in Chinese. However, it is not comparable between Gong's method and HPA in a quantitative style because the conditions and corpus are not comparable. Some qualitative comparisons are given as follows.

The times of I/O reading-writing are the key factor to decide the efficiency of repeat extraction because the speed of I/O operations is much slower than that of memory operations. For the method of Gong et al., when the size of corpus is greater than that of memory, the times of reading the whole corpus will be the size of character set, for example, if processing Chinese corpus, the reading times will be more than 6000; When the size of corpus is much greater than memory capacity, the I/O reading-writing times of Gong's method would increase exponentially, while that of HPA is nearly linear with the

size of corpus, for example, the reading times of HPA is 16.3 when the size of corpus is 32 gigabytes.

On the other side, the method of Gong et al. has better parallel and extensible performance, but HPA is difficult to be parallelized because it requires knowing the final set of short repeats before processing long strings.

V. Conclusions and Future Works

In this paper, we put forward hierarchical pruning algorithm to extract high-frequency strings. By HPA, the extraction method can effectively decrease the usage of memory, greatly reduce the times of I/O reading-writing, and improve the efficiency of repeat extraction in LSC. Experiments have shown that HPA can filter more than 85% low-frequency garbage strings out and the times of I/O reading-writing for the whole corpus has a nearly linear relationship with the corpus size. As a practical application, HPA has been employed in the Unknown Words Identification (UWI) system of our lab, and it can provide repeats as candidate words effectively.

Through a large number of studies, we have gotten a bold prediction: though the number of character combinations is theoretically infinite, when the size of corpus grows to a tremendous threshold size, the total amount of repeats will remain constant, i.e., the set of repeats has contained all reasonable combinations among characters. The reason for this is that, the combinations of Chinese characters must follow Chinese language habits, which leads to the finite number of repeats. Accordingly, for the restriction of corpus size, repeats that we have obtained are just a subset of total set of repeats under the current circumstance.

Though the efficiency of repeat extraction is improved by HPA, concerning the filtration effects of low-frequency strings there still is some room for improvement, especially in large-scale corpus. For the merging of candidate repeats from blocks, there may be some room for reducing the times of I/O reading-writing further. Among our future work, we will research into a new data structure to further decrease the memory usage in candidate string filtration without debasing extraction efficiency. We also want to study even larger corpus in order to exploit the new laws and trends which have not been found so far.

Acknowledgement

This work is supported by Xinjiang Uygur Autonomous Region Natural Science Foundation (2012211A057).

References

[1] Huang C, Zhao H. Chinese Word Segmentation: A Decade Review. Journal of Chinese Information Processing. 2007, 21(3): 8-19.

[2] Zheng J, Li W. A Study on Automatic Identification for Internet New Words According to Word-Building Rule. Journal of Shanxi University(Nat Sci Ed). 2002, 25(2): 115-119.

[3] Zou G. Research on Chinese New Words and Expressions Identification. Beijing: Graduate University of Chinese Academy of Sciences; 2004.

[4] Cui S, Liu Q, Meng Y et al. New Word Detection Based on Large-Scale Corpus. Journal of Computer Research and Development. 2006, 43(5): 927-932.

[5] Gong C, He M, Chen H et al. Frequent-Pattern Discovering Algorithm for Large-Scale Corpus. Journal of Communications. 2007, 28(12): 161-166.

[6] Cormen TH, Leiserson CE, Rivest RL et al. In Introduction to Algorithms (2nd Ed). Cambridge MA: MIT Press; 2001.

[7] Lu K. In Introduction to Computer Algorithms: Design & Analysis (Second Edition). Beijing Tsinghua University Press; 1996.

[8] Nevill-Manning CG, Witten IH. Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm. Journal of Artificial Intelligence Research. 1997: 67-82.

[9] Yamamoto M, Churcht KW. Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a Corpus. Computational Linguistics, 2001, MIT Press. 2001.

[10] Larsson NJ, Sadakane K. Faster Suffix Sorting. Lund, Sweden: Department of Computer Science, Lund University; 1999.

[11] Martin F-C, Paolo F, S M. On The Sorting Complexity of Suffix-Tree Construction. Journal of ACM. 2000, 47(6): 987-1011.

[12] Chen Z, Fowler R, Fu AW-C et al. Fast Construction of Genaralized Suffix Trees over A Very Large Alphabet. In Proc. Proceedings of International Conference on Computing and Cobinatorics;Big Sky,MT; 2003. pp. 284-293.

[13] Schurmann K-B, Stoye J. Suffix Tree Construction and Storage with Limited Main Memory. Bielefeld, Germany: University of Bielefeld; 2003.

[14] Clifford R, Sergot M. Distributed and Paged Suffix-Trees for Large Genetic Databases. In Proc. Proceedings of 14th Annual Symposium on Combinatorial Pattern Matching; 2003. pp. 70-82.

[15] Tian Y, Tata S, Hankins RA et al. Practical Methods for Constructing Suffix Trees. The VLDB Journal. 2005, 14(3): 281-299.

[16] Anisa AH, Maxime C, Lucian I et al. A comparison of index-based lempel-Ziv LZ77 factorization algorithms[J].ACM COMPUTING SerVey,2012,45(1):5.

[17] Zhang HJ, Pan WM, Munina,.A String Sort Algorithm in Custom Character Order [J]. Journal of Chinese Computer Systems,2012,33(9):1968-1971.

Haijun Zhang is a member of Chinese Information Processing Society of China. He is an associate professor of the school of computer science and technology, Xinjiang Normal University and got his doctoral degree at the school of computer science and technology, University of Science and Technology of China in 2011. His research interests mainly focus on natural language processing, new words identification and terms extraction from Uygur-Han bilingual languages.