Speeding up Pattern Matching by Optimal Partial String Extraction

TAN Jianlong[1], LIU Xia[1,2], LIU Yanbing[1,2],LIU Ping[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190
[2]Graduate School of Chinese Academy of Sciences, Beijing, 100049
e-mail:{tan,liuxia,liuping,liuyanbing}@software.ict.ac.cn

Abstract—**String matching plays a key role in web content monitoring systems. Suffix matching algorithms have good time efficiency, and thus are widely used. These algorithms require that all patterns in a set have the same length. When the patterns cannot satisfy this requirement, the leftmost m characters, m being the length of the shortest pattern, are extracted to construct the data structure. We call such m-character strings *partial strings*. However, a simple extraction from the left does not address the impact of partial string locations on search speed. We propose a novel method to extract the partial strings from each pattern which maximizes search speed. More specifically, with this method we can compute all the corresponding searching time cost by theoretical derivation, and choose the location which yields an approximately minimal search time. We evaluate our method on two rule sets: Snort and ClamAV. Experiments show that in most cases, our method achieves the fastest searching speed in all possible locations of partial string extraction, and is about 5%-20% faster than the alternative methods.**

Keywords: string matching; pattern matching; multiple string matching; SBOM algorithm; Wu Manber algorithm.

I. INTRODUCTION

The Internet allows rich user activities and produces an enormous volume of traffic owing to its high-degree connectivity and increasing scale. Because of its complexity and lack of central control, web content monitoring has inarguably become one of the most important techniques in detecting malicious attacks. String matching is a key component in such systems, which rely on the detection of certain patterns in web content that could potentially be harmful. There are many existing string matching algorithms, such as SBOM (Set Backward Oracle Matching)[3], Aho-Corasick[4], Set Horspool[5], Wu-Manber[6], SOG[7], etc. These matching algorithms are classified into two categories: prefix matching and suffix matching. In general, suffix matching is faster and more effective in handling long patterns than prefix matching; thus, it is used more widely.

Suffix matching algorithms require patterns of the same length, but this requirement is not satisfied all the time because of the randomness of rule sets. Most suffix matching algorithms adopt a very simple method to extract partial strings with the same length. Take the Wu-Manber algorithm as an example. This algorithm simply selects the leftmost m characters to construct the data structure (where m denotes the length of the shortest pattern). However, such a selection rule ignores the potential impact of partial strings on the searching time efficiency.

In this paper, we consider the problem of finding the optimal location to extract partial strings from these patterns, in terms of efficient searching time. In the training process, the length of partial strings is determined by the shortest pattern's length. Having located the best position for an equal partial string extraction, suffix matching algorithms can construct their basic data structure with these equal partial strings, by which searching texts can achieve the fastest speed.

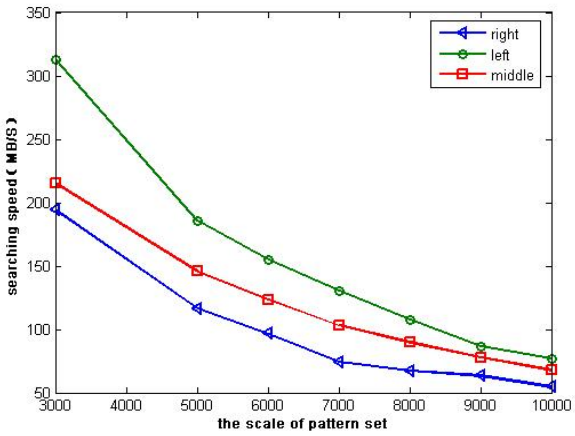

Figure 1. Partial strings at different locations result in different searching speed: right/ left/ middle means partial strings are extracted from the right/ left/ middle m characters. (pattern number range: 3000-10000, pattern length range: 5-18)

Before introducing our method, we first investigate the influence on searching time cost when using different strategies of partial string selection. With the Wu-Manber algorithm, we extract partial strings from three positions: the leftmost, the rightmost, and the middle of the pattern. Figure 1 shows the comparison of their corresponding searching speed. We can see that the speed of extracting partial strings from the leftmost m characters is the fastest, while extracting from the rightmost m is the slowest. Speed of extracting from the middle position lies between. The speed of the left extraction is about 50% faster

than that of the middle, and about 60% faster than that of the right. Such contrast demonstrates that extracting partial strings from different positions results in different searching time cost. Therefore, it is necessary to design an efficient partial-string extraction method to select proper partial strings of length m. With these partial strings extracted from the appropriate positions, we expect to achieve the fastest searching speed.

Our method is based on the principle that searching texts by the data structure constructed with the extracted partial strings should incur a minimal time cost. In the training process, we first extract all the partial strings from different positions in the pattern set. Second, we collect all the substrings within one certain partial string, and record the corresponding time cost of searching the training data text. These corresponding time costs are summed up to obtain the total time cost for the partial string. At this point, we compare the total time costs of all partial strings from different positions, and select the location with the minimum time cost. We can find the optimal locations of other patterns in the same way. Finally, we obtain the optimal partial strings set from the original pattern set. We choose the training texts similar with the actual texts to ensure that the extraction is effective.

II. RELATED WORKS

In order to better present our method, we first define some symbols:

Table 1 Definitions of some symbols

$P=\{p_1, p_2,\ldots p_n\}$	The original pattern set, where pattern may have different lengths
q_{ij}	The j-th partial string in the i-th pattern
T	A text to be searched
m	The shortest pattern's length in P
d_i	When searching text with SBOM, the number of matched characters in a pattern within the i-th window of text.
k	When searching a text of length *t* by the factor oracle, the times of slipping windows
l_i	The length of the i-th pattern
x	q_{ij}'s offset from the beginning position of the i-th pattern
z	The length of $s^{p,x}_{y,z}$
y	The offset of $s^{1}_{j,k}$ from the beginning position in q_{ij}
$S_{sub}=\{s^{p,x}_{y,z}\}$	The substring set, with the substring defined by *x,y,z*,p
$C_{sub}=\{c^{p,x}_{y,z}\}$	The occurrences of S_{sub} in R
R	The text to be searched in the training process
$Q=\{q_{1i}, q_{2j}, q_{3k}\ldots q_{nh}\}$	The set of extracted partial strings of length m in P

SBOM[3] algorithm is adopted in our method to find the optimal partial string extraction location during the training process. It uses factor oracle as its data structure, which is based on weak factor recognition. [3] has defined this new automaton built on a pattern $p_i=c_1c_2\ldots c_m$ which acts like an oracle on the set of factors $c_i\ldots c_j$. If a string is recognized by this automaton, it may be a factor of p; On the other hand, if a string is rejected, it is surely not a factor.

Figure 2 shows how to search a text with the BOM algorithm. The factor oracle is constructed by one pattern p_i. This algorithm adopts suffix matching in one window of size *m*, and when the matching is completed, we slide the window by a proper distance to start the next round of matching. Within one window, the factor oracle built by p_i is used to search for all the factors of p_i from the right to the left of the window in the text. At first, two pointers point at the rightmost of the pattern and the last position of the window in the text respectively. Then we match the two pointed characters. If the match is successful, both two pointers are decreased by one; otherwise, the matching in this window is over. The next window is shifted by m-d_i if a text $t=t_it_{i+1}\ldots t_{i+di-1}$ is recognized by the oracle. If the pointers are shifted to the start of the whole window then the occurrences of p_i are added by one. The worst-case complexity of BOM is $O(nm)$. However, under a model of independence and equiprobability of characters, the BOM algorithm has an optimal average complexity of $O(\frac{n}{m}\log_{|\Sigma|} m)$.

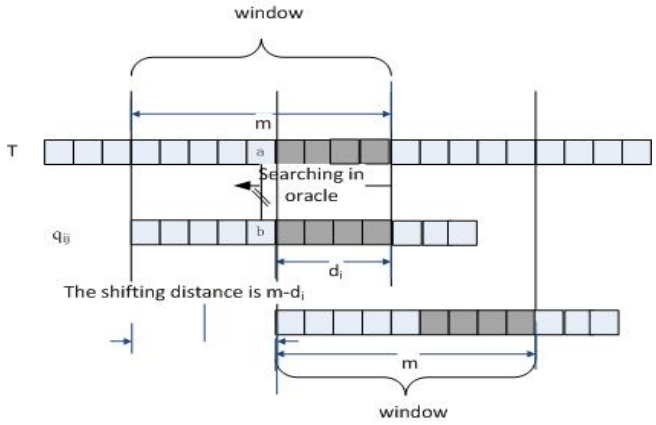

Figure 2. BOM searches for the factor of q_{ij} in the text and computes the next shift distance

SBOM is to find the occurrences of a pattern set instead of one pattern, and it is an effective suffix matching algorithm when handling long patterns. However, SBOM still requires a pattern set with elements of the same length (as with other suffix matching algorithms). This condition can seldom be satisfied in practice, and there has been no effective way to solve this problem. Hence we aim to design a method which will locate the optimal positions for partial strings extraction within a certain pattern set. In Section III, we will describe this method in details.

[8] proposed a novel alphabet sampling technique that chooses a subset of the alphabet and selects the corresponding subsequence of text. This method can speed up both online and indexed string matching because the searching is carried out on the subsequence. The candidate matches are then verified in the full text. [8] and our method share a commonality of speeding up the searching process by selecting representative subset.

However, [8] selects subset both from original pattern set and text, while our method only selects subset from pattern set. Furthermore, [8] can select the letters from noncontinuous positions, but ours just selects a section of partial string from some certain position in one pattern. Both these two methods can be applied in suffix matching algorithms.

[9] optimizes matching algorithms by compressing DFAs, based on the observation that the names of states are actually meaningless in practice. This paper encodes states in such a way that all transitions to a specific state can be represented by a single prefix that defines a set of current states. Hence the problem of pattern matching is reduced to the well-studied problem of Longest Prefix Matching. But our paper aims to optimize matching by speeding up the searching process. Furthermore, [9] is applied to the automaton based algorithms, while our method is used in the suffix algorithms.

III. OUR METHOD: OPTIMAL PARTIAL STRING EXTRACTION

The basic idea behind our method is a minimum time cost in searching training texts by the factor oracle built by Q.

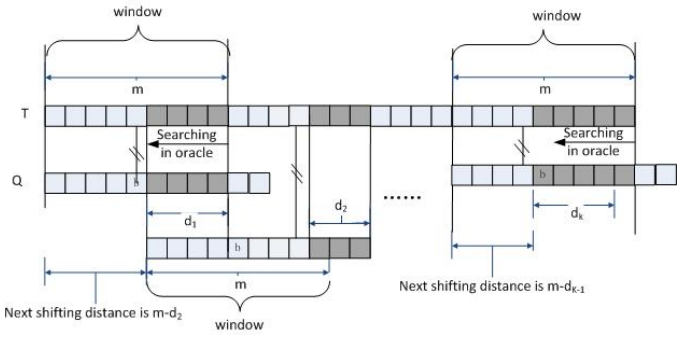

Figure 3. SBOM finds all the occurrences of the pattern set Q in text T.

A. Searching Text by The Factor Oracle Built by Pattern Set

Figure 3 illustrates the searching process by SBOM with a certain pattern set.

In Figure 3, two pointers have moved d_1 off the last position in this window, which means there are only d_1 characters that were successfully matched. The window is then shifted after the failed search by the oracle. The two pointers are set to the last position of the new window again. These two pointers are decreased until the pointed characters fail to match. These steps are iterated until the end of the text, with the total number of times in which the window is slid being *k*.

Within the i-th window, the comparisons between the corresponding two characters have been completed for d_i times. So the total time cost is $\sum_{i=1}^{k} d_i$, and the equation $km - \sum_{i=1}^{k} d_i = t$ holds for k. Formula 1 is the total time cost.

$$totalcost = km - t \quad \text{(Formula 1)}$$

$$averagecost = \frac{km}{t} - 1 \quad \text{(Formula 2)}$$

The average time cost on each letter is determined using Formula 2. We could change the value of *k* to achieve the minimum average time cost. Apparently, when *k* is equal to the minimum value, the average time cost is also the minimal. *k* is determined by Q. Therefore, we need to enumerate all possible values of Q to find the minimum *k*.

However, the potential number of Q is exponential, because it could be obtained by any combination of different locations for partial strings of length m in each pattern. Every pattern p_i has $l_i - m + 1$ partial strings, and the number of different combinations is $\prod_{i=1}^{n}(l_i - m + 1)$. Hence it is impractical to enumerate them all. Thus a compromise solution is to design a method that could select Q with approximately minimum searching time cost. Section B will detail the method.

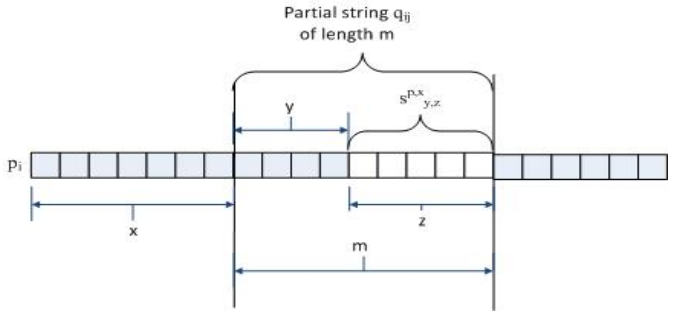

Figure 4. One example of substring $s^{i}_{j,k}$ in q_{ij}

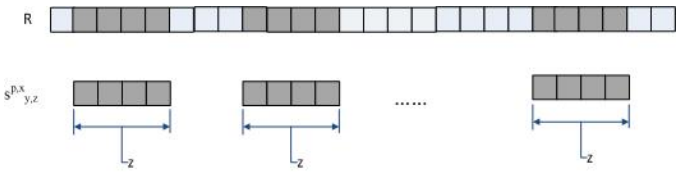

Figure 5. Looking for all the occurrences of $s^{p,x}_{y,z}$ in the training text R.

B. Our Extracting Location Method

We have addressed the impossibility of enumerating all possible combinations of the various locations for partial strings of length m in P in Section A. However, if there is only one pattern in P, it is easy to find the best location in this pattern which yields the minimum time cost. Thus, we propose locating the proper partial string in each pattern, respectively. In this approach, Q is the union of all these partial strings. Although this method may not provide the best extraction, it could obtain approximately optimal one. Experiments show that the extraction can achieve good searching time efficiency.

In our method, for one pattern, we focus on the time cost of searching all the substrings of each partial string in the training text. The time cost is computed as follows: All of the substrings in one partial string of length *m* are used to build an Aho-Corasick automaton. By searching the training data text on this automaton, their corresponding occurrences can be obtained. The time cost of each substring is computed according to the formula discussed in Section A. Then we can sum these time costs up to get the total time cost of this partial string. Similarly, we can compute other total time costs which correspond to other partial strings of length *m* in the same

pattern. After comparing these total time costs, we select the partial string which yields the minimum time cost. Other locations for partial strings are determined in the same way. Finally, Q consists of a union of the partial strings incurring the minimum time costs.

Figure 4 shows an example of substring of length z in one certain partial string. After finding all occurrences of the substring denoted by $c^i_{j,k}$, the time cost of $s^i_{j,k}$ is: $c^i_{j,k}*z$, which equals to $F(x, y, m, R) * z$ ($c^i_{j,k}$ is determined by the value of function F(x, y, m, R)). Thus we can obtain the formula of time cost:

$$F(x, y, m, R) * z \qquad \text{(Formula 3)}$$

The values of x, y and z can be changed to obtain all of the substrings and their corresponding time cost. When these values are summed up, we finally obtain the total time cost of the whole partial string. Similarly, we can compute the total time costs of other partial strings. By comparing these time costs we can identify the best location for partial string extraction in one pattern. Proper locations for partial strings in other patterns are obtained in the same way. Finally, Q is the union of all partial strings extracted from P.

In the training process, our location algorithm is presented as follows:

```
Input: P(Original Pattern Set), R(Train Data)
Output: Q(the set of extracted partial strings of length m in P)
1.  for each pattern pt in P
2.      S_sub ← {s^{p,x}_{y,z}}
3.      Ac ← Build_AC(S_sub)
4.      C_sub={ c^{p,x}_{y,z} } ← Search_InAC(R)
5.      for each partial string ppt in pt
6.          minx ← 0; mincost ← +∞ ; ps ← ξ
7.          for x=0 to x<|ppt|-m
8.              costx ← Σ_{y=0}^{m} Σ_{z=1}^{m-y} c^{ppt,z}_{y,x}
9.              if costx<mincost do
10.                 mincost=costx; Minx=x; ps=ppt
11.             end of if
12.         end of for
13.     end of for
14.     Add ps to Q
15. end of for
```

This pre-process does not cost much time, with its complexity O(|P|*(l-m)*m +|R|). In the next section, we will present experiment results and analysis.

IV. EXPERIMENTS AND ANALYSIS

In the experiments, our training pattern datasets are collected from both Snort[1], an intrusion detection system, and ClamAV[2], an open source antivirus toolkit in Unix. All experiments are carried out on a Window XP platform with a 2.93Hz dual core CPU and 2GB RAM.

In the following figures, right (left/middle) was extracted from the rightmost (leftmost/middle) m characters. *Mincost* represents the extraction as computed by our method.

A. Speed comparison on partial string extractions at different positions.

In our first experiment, we compare different searching speed of SBOM when partial strings are extracted from different positions.

In Figure 6, we can see that extracting Q from P at the position computed by our method yields the fastest searching speed. The speed is about 20% faster than the second best speed which is obtained by the right Q, and is about 30% faster than the slowest one.

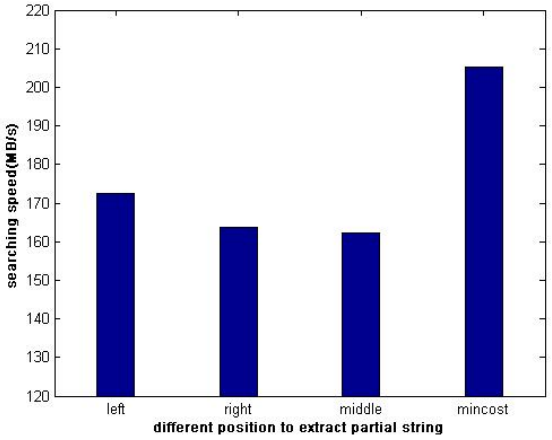

Figure 6. Speed comparison of partial string extraction at different positions. (pattern set is from Snort, pattern number:4656, pattern length range: 5-18).

B. Speed comparison on partial strings extraction at different positions with pattern set changing

In this experiment, we evaluate the variation of searching speed as the scale of pattern set changes. In Figure 7, we can see the speed at various locations for partial string extractions decreases as the size of pattern set scales up. Moreover, our algorithm consistently has the best searching speed as the pattern set changes, which remains about 20% faster than the next best speed. The speed of the other three partial string extractions is on a similar level.

To illustrate the universality of our algorithm, we choose another pattern set from ClamAV. As demonstrated in Figure 8, we can also conclude that the searching speed of the extracted partial strings obtained by our algorithm is faster than any others. The other three speeds remain comparable.

Figure 9 illustrates the length of the shortest pattern's impact on the searching speed. As the shortest length increases, the searching speed is also increasing. It indicates that the length of the shortest pattern is an important factor in degrading the performance of searching. When the shortest length of patterns are larger than 12, the searching speed of our method drops. The reason is that when the shortest length is

large enough, these partial strings are so effective that they are almost the same in the searching speed.

Our algorithm allows the proper partial strings extraction from a pattern set, with random pattern lengths meeting most suffix matching algorithms' requirements. Each of the experiments discussed here shows that choosing the partial strings extracted by our method can achieve better searching speed than those by other methods. Moreover, the advantage of our method is more obvious on the Snort pattern set. When the scale of pattern set increases, our method can achieve more graceful degradation.

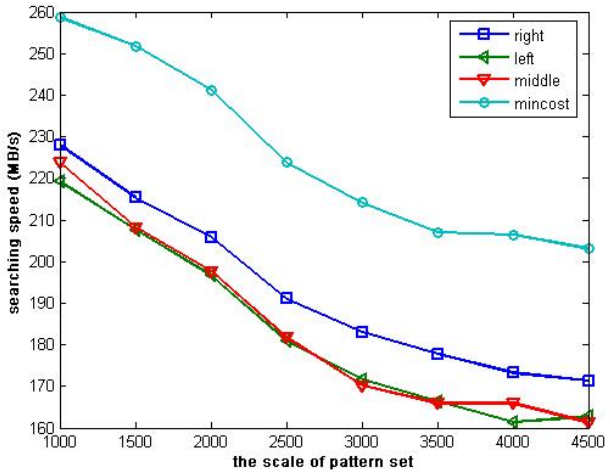

Figure 7. Speed comparisons of partial string extractions at different positions with pattern sets changing. Pattern sets are from Snort. (pattern number range: 1000-3500, pattern length range: 5-18).

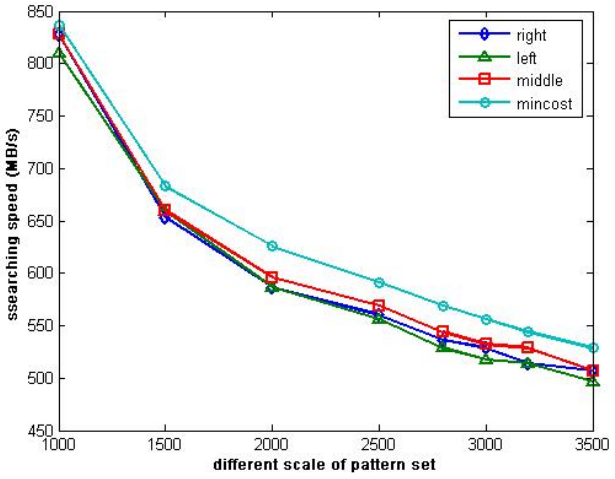

Figure 8. Speed comparison on partial string extractions at different positions with pattern sets of various scales. The pattern sets are selected from ClamAV (pattern number range: 1000-3500, pattern length range:4-16).

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel method to extract optimal partial strings based on achieving the fastest searching speed. According to the theoretical derivation in Formula 2, we need to enumerate all possible positions of partial strings in original pattern sets to see which yields the fastest searching speed. The impracticality of this approach, however, leads us to look for another method to find the approximate location. Experiments show that our method can reach better searching speed in all the locations of partial string extractions in most cases, and is about 5%-20% faster than the closest one on the Snort and ClamAV data set.

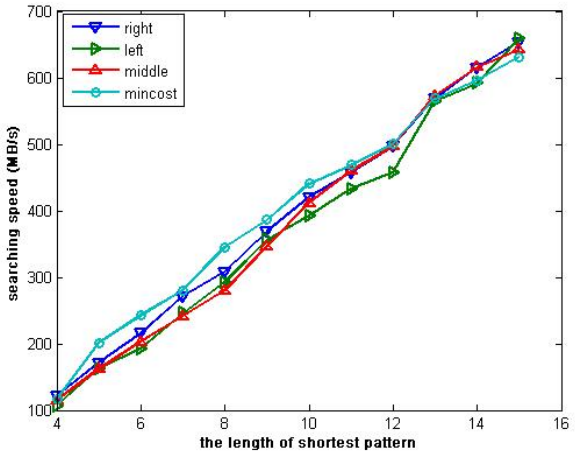

Figure 9. Speed comparison on partial string extractions at different positions with various lengths of the shortest pattern (pattern number:4656, the length of the longest pattern: 18).

We plan to survey this critical point in more depth in future work. We hope to find a way to avoid the combinatorial explosion, so that we can locate the exact optimal extraction.

ACKNOWLEDGMENT

This work is supported by the National Basic Research Program of China (973 Program) under grant No. 2007CB311100, the National High Technology Research and Development Program of China (863 Program) under grant No. 2009AA01Z436, and the National Natural Science Foundation of China under grant No. 61070026.

REFERENCES

[1] Snort Rule. http://www.snort.org/snort-rules

[2] ClamAV Rule. http://www.clamav.net/lang/en/download/cvd/

[3] C. Allauzen, M. Crochemore and M. Raffinot, "Efficient Experimental String Matching by Weak Factor Recognition", in *Proc. 12th Annu. Symp. on Combinatorial Pattern Matching*, Jerusalem, July 1–4, 2001, pp. 51-72.

[4] A. Aho and M. Corasick, Bell Laboratories. "Efficient String Matching:An Aid to Bibliographic Search", in *Communications of the ACM*, vol. 8, 1975, pp. 333-340.

[5] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings: Practical on-line search algorithms for texts and biological sequence*. Cambridge: Cambridge University Press, 2002.

[6] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching", Dept. of Computer Science, University of Arizona, Tucson, AZ, TR-94-17, 1994.

[7] L. Salmela, J. Tarhio and J. Kytojoki, “Multi-Pattern String Matching with q-Grams”, in *Proc. 14th Annu. Symp. on Combinatorial Pattern Matching*, Michoacan , June 25–27, 2003, pp. 211-224.

[8] F. Claude, G. Navarro, H. Peltola, L. Salmela and J. Tarhio, “Speeding Up Pattern Matching by Text Sampling”, in *Proc. 15th Intl. Symp. on String Processing and Information Retrieval* , Melbourne, November 10-12, 2008, pp. 87-98.

[9] A. Bremler-Barr, D. Hay and Y. Koral, “CompactDFA: Generic State Machine Compression for Scalable Pattern Matching”, in Proc. 29th IEEE INFOCOM, San Diego, March 14-19, 2010, pp. 1-9.

[10] C. Allauzen and M. Raffinot, “Factor oracle of a set of words”, Institute Gaspart-Monge, University de Marne-la-vallee, TR-99-11,1999.

[11] X. Wang, “The Design and Analysis of Computer Algorithms”, Beijing: Publishing House of Electronic Industry, 2001