 doi:10.20944/preprints201608.0197.v1

Article

A Multi-Fusion Pattern Matching Algorithm for Signature-Based Network Intrusion Detection System

Manohar Naik S [1,*] and Geethanjali N [2]

[1] Research Scholar, Department of Computer Science & Technology, Sri Krishnadevaraya Univeristy, Anantapur, 515003 Andhra Pradesh, India
[2] Department of Computer Science & Technology, Sri Krishnadevaraya University, Anantapur, 515003 Andhra Pradesh, India; geethanjali.sku@gmail.com
* Correspondence: manoharamen@gmail.com; Tel.: +91-8008-255-400

Abstract: Security has become a critical issue in today's highly distributed and networked systems. Network intrusion detection systems (NIDSs), especially signature-based NIDSs, are being widely deployed in a distributed network environment with the purpose of defending against a variety of network attacks. Most of the commercially available NIDSs are software based and rely on pattern matching to extract the threat from network traffic. The increase in network speed and traffic may make existing algorithms to become a performance bottleneck. Therefore it is very necessary to develop faster and more efficient pattern matching algorithm in order to overcome the troubles on performance of NIDSs. Therefore, we propose a multi fusion pattern matching algorithm for Network Intrusion Detection Systems. The results obtained in percentages from the proposed fusion algorithm given better values in terms processing time in milliseconds than the existing algorithms when data English text are applied to evaluate the fusion performances.

Keywords: attack; network intrusion detection systems; pattern matching algorithms

---

1. Introduction

Network intrusions (e.g., malware, exploits) are becoming a critical issue for the whole network communications [1]. This can cause loss or harm of data on computers, and loss of sensitive information is something that occasionally happens. Companies spend billions of dollars on computer security each year, but still computers get infected or compromised by malicious traffic. Large companies and governmental organizations need the best tools available to prevent intrusions, since their companies are more exposed to malicious traffic. To mitigate this issue, network intrusion detection systems (NIDSs) [2,3] have been widely implemented in different kinds of network environments, aiming to enhance network security by defending against different kinds of network attacks. In addition, these intrusion detection systems have already deployed in a distributed environment (e.g., agent-based network, mobile ad hoc network-MANET) to perform detection of intrusions.

Network Intrusion Detection Systems (NIDSs) are designed to identify attacks or intrusions against networks. As these threats can be invisible to firewalls, NIDS provides an additional layer of security and is being widely deployed in various network environments.

There are two approaches that can carry out NIDS; first is anomaly detection and second is misuse detection. Anomaly detection can detect new attacks or potential attacks but the problem of accuracy is still open to research. . It generates a lot of false positive alarms. Signature detection (misuse detection) that has been used for detecting the known attacks; has the higher level of security than anomaly detection, but the major problem of Signature-based NIDS is that every attack signature should have an entry in the database in order to compare with the arrived packets; therefore, the process will be time-consuming and will slow down the throughput of the NIDS.

At the heart of almost every modern Signature-based NIDS, there is a PME (pattern matching engine). Essentially, the pattern matching algorithm compares the set of patterns in the rule set (also

 doi:10.20944/preprints201608.0197.v1

called signature database) to the payloads of the packets. Pattern matching is computationally intensive. The pattern matching routines in Snort, a famous open source lightweight NIDS [4], account for up to 70% of total execution time and 80% of instructions executed on real traces [5].

Because pattern matching is a critical building block for network security applications that inspect payloads, it is reported that string matching is one of the most expensive operations in Web application firewalls [6] and accounts for 70 to 80% of CPU cycles for IDS [7, 8]. An efficient PME is crucial to NIDS. Many studies have shown that string pattern matching is one of the primary performance bottlenecks for these systems [7, 8, 9, 6, 10].

The key challenge to pattern string matching is that its performance requirement has increased dramatically (e.g., from multi-Gbps [12] to multi-10s of Gbps [16]), outpacing the performance of existing solutions [8, 6]. If the capacity of NIDS cannot matching the speed of network, a passive NIDS will drop packets and thus miss attacks, while an inline NIDS will create a bottleneck for network performance. On the other hand, as the number of potential threats and their associated signatures is expected to grow, the cost of pattern matching is likely to increase further. Therefore, the pattern matching algorithm needs to be highly efficient to keep up with the increasing volume of network traffic, as well as the increasing number of patterns.

The majority of main commercial intrusion detection systems primarily use signature-based NIDS, so the progress of signature based intrusion detection system is vital. One of the prominent factors to achieve this goal is improving the performance of Signature based Network Intrusion Detection Systems in order to able process more traffic in less time, hence the speed of the processors should be increased. Several efforts have been done in this area such as improving the content matching algorithms, hardware acceleration, and parallel process.

There are some algorithms, which have been used for content matching. Aho and Corasick [13] provided an algorithm for searching multiple Patterns simultaneously in text, but it requires a significant amount of memory for the state machine. Several improvements have been presented in Commentz-Walter, B., [14]. The most widely used algorithm proposed by Boyer and Moore [15], is a single pattern matching algorithm that compares the string with the input starting from the rightmost character of the string. It provides the best performance when searching for a single signature, but scales poorly. Sun Wu and Udi Manber [16] proposed their multi pattern matching algorithm. . It mainly uses the bad character heuristic of Boyer-Moore algorithm, and requires less memory than Aho-Corasick and provides a better average case performance. Sun Kim and Yangon Kim [17] proposed their encoding and hashing based multi pattern matching algorithm, which is possible to encode the characters with fewer bits. It is the same idea as compression algorithms.

These algorithms are the most important between others, so we can conclude that the performance of signature based NIDS has been proven to be related by the speed of the string matching algorithms used to compare packets with signatures (Fisk, M. and G. Varghese, 2002) [18]. Implementing a different algorithm some-times causes an increase in performance up to 500 percent for snort 2.0; a widely used open source NIDS (Baker, Z.K. and V.K. Prasanna, 2004) [19].

The rest of the paper is organized as follows. Section II describes the contributed characterization of pattern matching in network security such as signature-based NIDS; Section III describes the existing pattern matching algorithms; Section IV gives the proposed algorithm; Section V covers the performance and analysis of proposed algorithm with some existing pattern matching algorithms; and finally Section VI conclude the paper.

2. Pattern Matching Algorithms for NIDS

Pattern matching is one of the most computationally intensive tasks of IDS. Performance of IDS suffers as signatures or rules grow in data volume. In order to increase the performance of NIDS, one of approach is to improve the performance of signatures detection engine by increasing the efficiency of pattern matching algorithm.

Pattern matching is a pivotal theme in computer research because of its relevance to various applications including intrusion detection systems (IDS), web search engines, computational biology, virus scan software, network security and text processing, and text mining [21]. Pattern

 doi:10.20944/preprints201608.0197.v1

matching focuses on finding the occurrences of a particular pattern P of length 'm' in a text 'T' of length 'n'. Both the pattern and the text are built over a finite alphabet set called Σ of size σ. . If more than one search strings are matched against the input string simultaneously, it is called multiple pattern matching. Otherwise, it is called single pattern matching. Generally, pattern matching algorithms make use of a single window whose size is equal to the pattern length. The searching process starts by aligning the pattern to the left end of the text and then the corresponding characters from the pattern and the text are compared. Character comparisons continue until a whole match is found or a mismatch occurs, in either case the window is shifted to the right in a certain distance [22].

Simone Faro and Thierry Lecroq 2010, 2013 [23] [22] provided two strong surveys, the first one [70] gave a comprehensive experimental evaluation for exact string matching algorithms. The second one [22] reviewed the string matching algorithms which have been proposed in the last decade 2000-2010 and presented experimental results in order to bring order among the dozens of articles published in recent years. Vidya SaiKrishna, Prof. Akhtar Rasool, and Nilay Khare 2012 [24] explored the various diversified fields where string matching has an eminent role to play and is found as a solution to many problems. Kamal Alhendawi and Ahmad Baharudin 2013 [25] introduced a short survey for five of well-known string matching algorithms, including theoretical analysis, empirical testing of the execution time based on the change of two factors (text size and pattern size), then it measured the efficiency of each string matching algorithm in term of estimated execution time. While Gulfishan Firdose Ahmed and Nilay Khare 2014 [26] presented a survey of several hardware based string matching algorithms such as Brute Force, KMP [27], and Aho-Corasicks [13] with their applications.

The shift value, the direction of the sliding window and the order in which comparisons are made varies in different pattern matching algorithms. To reduce the number of comparisons, the matching process is usually divided into two phases. The pre-processing phase and the searching phase. The pre-processing phase determines the distance (shift value) that the pattern window will move. The searching phase uses this shift value while searching for the pattern in the text with as minimum character comparisons as possible. A good pattern matching algorithm aims to decrease the searching phase during each attempt and to increase the shifting value of the pattern. Hence several pattern matching algorithms have been developed with a view to enhance the searching processes by minimizing the number of comparisons performed [28-30]. About string length of Snort pattern, M. Aldwairi et al [32] suggest that the average string length is 14 bytes and the majority of the strings are shorter than 26 bytes; it is also clear that there is a non-negligible number of strings longer than the 40 bytes.

In this paper, we propose a new method to improve the average performance of our algorithm. The main idea behind the proposed algorithm is the order of comparisons is carried out by comparing the last character of the window and the pattern, and after a match, the algorithm further compares the first character of the window and the pattern. We will discuss in detail in next section.

3. Existing Algorithms

Many pattern matching algorithms are available with their own merits and demerits based on the pattern length, periodicity and alphabet set. One of the most viable approaches to this problem is to compare the text and the pattern in an effective pre-defined order. An efficient way is to move the pattern on the text using the best shift value. To this end, several algorithms have been proposed to get a better shift value, for example, Boyer-Moore [15], Quick Search [33], Raita [34] and Berry-Ravindran [35] algorithms.

3.1. Boyer-Moore Algorithm (BM)

The algorithm preprocesses the pattern and creates two tables, which are known as Boyer-Moore bad character (bmBc) and Boyer-Moore good-suffix (bmGs) tables. For each character in the alphabet set, a bad-character table stores the shift value based on the occurrence of the

 doi:10.20944/preprints201608.0197.v1

character in the pattern. On the other hand, a good-suffix table stores the matching shift value for each character in the pattern. The maximum of the shift value between the bmBc (character in the text due to which a mismatch occurred) dependent expression and from the bmGs table for a matching suffix is considered after each attempt, during the searching phase. This algorithm forms the basis for several pattern-matching algorithms.

3.2. Raita Algorithm

Raita algorithm searches for a pattern in a given text by comparing each character of pattern in the given text. Searching will be done as follows. First, last character of the pattern is compared with the rightmost character of the window. If there is a match, first character of the pattern is compared with the leftmost character of the window. If they match again, it compares the middle character of the pattern with middle character of the window. If everything is successful, then the original comparison starts from the second character to last but one. If there is a mismatch at any stage in the algorithm, it performs the bad character shift function which was computed in pre-processing phase. Bad character shift function is similar to the one proposed in Boyer-Moore algorithm.

3.3. Berry-Ravindran Algorithm (BR)

The Berry-Ravindran algorithm is a composite of the Quick Search algorithm and another variant of the Boyer-Moore algorithm, the Zhu-Takaoka algorithms. It performs the window shifts by considering the "bad-character shift" for the two consecutive text characters to the right of the window. The shift values are obtained from a two-dimensional array, computed in the preprocessing stage. This is the only algorithm which uses two consecutive characters to compute the shift value of the pattern on text.

The maximum of the shift value between the bmBc (character in the text due to which a mismatch occurred) dependent expression and from the bmGs table for a matching suffix is considered after each attempt, during the searching phase. This algorithm forms the basis for several pattern-matching algorithms.

3.3. SSABS and TVSBS Algorithms

These algorithms are a combination of the shifting method of Quick Search algorithm and the searching method of the Raita algorithm. This done by comparing the rightmost and leftmost characters first, and then continuing the comparisons of other characters from right to left until a complete match or a mismatch occurs. After each search, the shift of the window is computed by the Quick-Search bad character rule for the next character to the window.

4. Proposed Algorithm

The proposed algorithm is a fusion of Berry-Ravindran and Raita algorithms. The Berry-Ravindran bad character (brBc) function is proved in many cases effective in preprocessing phase. So same has been implemented with slight modifications. The searching phase is almost same as Raita algorithm.

This new algorithm compares the right most character of the pattern and the sliding window, if it is matched then the leftmost character of the pattern and the sliding window is compared, if it also matched then the middle character of the pattern is compared to the corresponding position of the text window, if it is matched then it is again compare the characters string from the second last character of the pattern to the text window. In case of match or mismatch the skip of the window is achieved by the Berry-Ravindran (brBc) shift value for the character that is placed next to the window.

 doi:10.20944/preprints201608.0197.v1

The pseudo code given below illustrates overall strategy.

Algorithm 1 Preprocessing Phase

```
preProcess(x,brBc[][])
begin
//Create a bad character table
//x is the pattern of length m
//ASIZE alphabet size
for i:=1 to ASIZE do
for j:=1 to ASIZE do
brBc[i,j]:= m+2
for i:=1 to ASIZE do
brBc[i,x[1]]:= m+1
for i:=1 to m do
brBc[x[i],x[i+1]]:= m-i
for i:=1 to ASIZE do
brBc[x[m-1],i]= 1
end
```

The pre-processing phase assists the searching phase to improve the overall efficiency of the proposed algorithm. In addition, the maximum shift for the window reduces the number of character-character comparisons and hence increases the performance.

Algorithm 2 Searching Phase

```
search(x,y,m,n)
begin
//x is the pattern of length m
//y is the text of length n
preBrBc(x,brBc);// returns shift value
while(shift<=n-m) do
begin
if x[m-1] equals y[shift+m-1] then
if x[0] equals y[shift] then
if x[m/2] equals y[shift+m/2] then
i:= m-1
k:= 0
while(i>=2 and x[i] equals y[shift+i])
k:= k+1
i:= i-1
if k equals m-2
print "Pattern found"
shift:= shift+brBc[y[shift+m],y[shift+m+1]]
end
```

5. Performance and Evaluation

In this section we present experimental results in order to evaluate the performances of our new algorithm and to compare it against the best existing algorithms known in literature for multiple string matching problems. The newly proposed algorithm is referred as "Multi-Fusion Pattern Matching" (MFPM) algorithm.

 doi:10.20944/preprints201608.0197.v1

The existing and proposed algorithms are implemented in C programming language and have been compiled with the GNU C Compiler, using the optimization options -O3. The experiments were executed locally on a MacBook Pro with 4 Cores, a 2 GHz Intel Core i5 processor, 4 GB RAM 1333 MHz DDR3, 256 KB of L2 Cache and 6 MB of Cache L3. Algorithms have been compared in terms of running times, including any preprocessing time, measured with hardware cycle counter, available in modern CPUs. In particular we compared the performances of our proposed algorithm against existing algorithms specifically – Berry-Ravindran (BR) TVSBS algorithms. For the evaluation, we use a natural language text (English language), all sequences of 4MB. We have generated sets of 10, 100, 1000 and 10000 patterns of fixed length for the tests. In all cases the patterns were randomly extracted from the text. For each case we reported the mean over the running times of 200 runs. Table. 1 lists the timings achieved on englishTexts dataset. Running times are expressed in milliseconds.

Table 1. Running Times of Pattern Matching Algorithms. (in Milliseconds).

Algorithm	2	4	6	8	10	12	14	16	18	20
BR	2.99	2.3	1.89	1.62	1.47	1.32	1.21	1.17	1.16	1.12
TVSBS	2.92	2.15	1.78	1.54	1.39	1.25	1.18	1.11	1.09	1.02
MFPA	2.71	2.1	1.71	1.50	1.32	1.21	1.12	1.08	1.00	0.94

Fig.1 shows that our newly proposed "Multi-Fusion Pattern Matching" (MFPM) algorithm's performance is better than the existing algorithms. As can be viewed from the results, the newly proposed solutions are in general faster than the competitors. Notice that the gain in speed is more significant in the case of natural language English text.

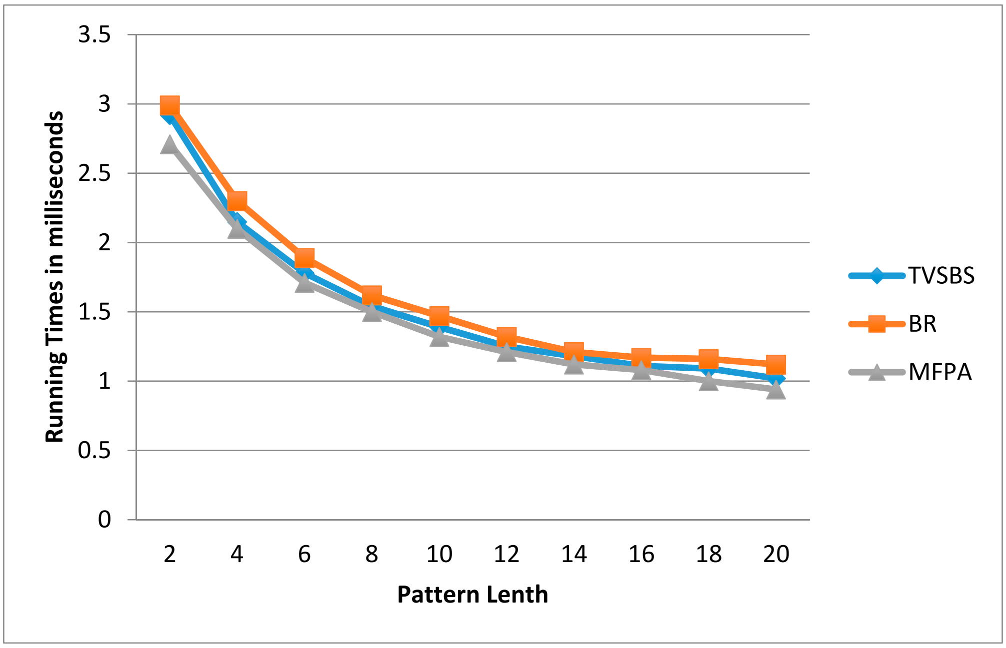

Figure 1. Pattern matching algorithms comparison by englishTexts dataset.

6. Conclusions

Since pattern matching algorithms are very essential in the current scenario of Intrusion Detection Systems, in this paper we proposed MFPA algorithm for fast intrusion detection. The pre-processing phase assists the searching phase to improve the overall efficiency of the proposed algorithm. In addition, the maximization of the skip for the window reduces the number of

 doi:10.20944/preprints201608.0197.v1

character-character comparisons and skip the undesirable characters efficiently and takes only few iterations to find the pattern in the given text string, and eventually it gives better performance.

Author Contributions: M.N conceived designed and performed the experiments. G.N helped to draft the manuscript. All read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Microsoft Security Intelligence Report (SIR), vol. 13, June2012. <http://www.microsoft.com/security/sir/default.aspx>.
2. V. Paxson, Bro: a system for detecting network intruders in real time, Computer Networks 31 (23-24) (1999) 2435–2463.
3. K. Scarfone, P. Mell, Guide to Intrusion Detection and Prevention Systems (IDPS), NIST Special Publication 800-94, February 2007.
4. M. Roesch. “Snort: Lightweight intrusion detection for networks.” Proc. 1999 USENIX LISA Systems Administration Conference, 1999.
5. S. Antonatos, K. G.. Anagnostakis, and E. P. Markatos. “Generating realistic workloads for network intrusion detection systems.” Proc. ACM Workshop on Software and Performance, 2004
6. Chris Ueland. Scaling cloudflare’s massive waf. http://www.scalescale.com/scaling-cloudflares-massive-waf/ [accessed 25-Jul-2016].
7. Spyros Antonatos, Kostas G. Anagnostakis, and Evangelos P.Markatos. Generating Realistic Workloads for Network Intrusion Detection Systems. ACM Special Interest Group on Software Engineering (SIGSOFT) Software Engineering Notes (SEN), 29(1), January 2004
8. Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. Kargus: A Highly-scalable Software-based Intrusion Detection System. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2012.
9. Po-Ching Lin, Zhi-Xiang Li, Ying-Dar Lin, Yuan-Cheng Lai, and F.C. Lin. Profiling and Accelerating String Matching Algorithms in Three Network Content Security Applications. IEEE Communications Surveys and Tutorials, 8(2):24–37, 2006.
10. Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P. Markatos, and Sotiris Ioannidis. Gnort: High Performance Network Intrusion Detection Using Graphics Processors. In Proceedings of the International Symposium on Research inAttacks, Intrusions and Defenses (RAID), 2008.
11. Capacity Planning for Snort IDS: Bilbous, Not Tapered. http://mikelococo.com/2011/08/snort-capacity-planning/. [accessed 01-Aug-2016].
12. Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer.Operational Experiences with High-volume Network Intrusion Detection. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2004.
13. Aho, A. and M. Corasick, 1975. Fast pattern matching: an aid to bibliographic search Common, ACM, 18(6): 333.340.
14. Commentz-Walter, B., 1979. A string matching algorithm fast on the average, In Proceedings of ICALP'79: 118-132.
15. R. S. Boyer, J. S. Moore, “A fast string searching algorithm,” Communications of ACM, vol. 20, no. 10, pp.762–772, 1977.
16. Wu, S and U. Manber, A fast algorithm for multi-pattern searching, Technical Report TR-94-17, University of Arizona, 1994.
17. Kim, S. and Y. Kim, A fast multiple string pattern matching algorithm, In Proceedings of 17th AoM/IAoM Conference on Computer Science, 1999.
18. Fisk, M. and G. Varghese, An analysis of fast string matching applied to content-based forwarding and intrusion detection, Technical Report CS2001-0670 (updated version), University of California - San Diego, 2002.
19. Baker, Z.K. and V.K. Prasanna, Time and area efficient pattern matching on fpgas,” In FPGA '04: Proceedings of the 2004 ACM/SIGDA 12 th international symposium on Field programmable gate arrays, pp.223-232, New York, NY, USA, ACM Press, 2004.

 doi:10.20944/preprints201608.0197.v1

20. Wang, Y. and H. Kobayashi, "High performance pattern matching algorithm for network security," IJCSNS, 6: 83-87, 2006.
21. Koloud Al-Khamaiseh, A Survey of String Matching Algorithms Int. Journal of Engineering Research and Applications www.ijera.com ISSN: 2248-9622, Vol. 4, Issue 7 (Version 2), pp.144-156, 2014.
22. S. Faro, T. Lecroq, "The Exact Online String Matching Problem: a Review of the Most Recent Results", ACM Computing Surveys (CSUR) Surveys Homepage archive, Volume 45 Issue 2, Article No. 13, February 2013.
23. S. Faro, and T. Lecroq, "The exact string matching problem: a comprehensive experimental evaluation", Report arXiv: 1012.2547, 2010.
24. V. SaiKrishna, A. Rasool, and N. Khare, "String Matching and its Applications in Diversified Fields", International Journal of Computer Science Issues (IJCSI), Volume 9 Issue 1, p219-226, Jan2012
25. K. Hendawi, and A. Baharudin, "String Matching Algoritms (SMAs): Survey & Empirical Analysis", Journal of Computer Sciences and Management, Volume 2, Issue 5, 2013.
26. G. Ahmed and N. Khare, "Hardware based String Matching Algorithms: A Survey", International Journal of Computer Applications, volume 88(11):16-19, February 2014.
27. D. Knuth, J. Morris, and V. Pratt, "Fast pattern matching in strings", SIAM Journal on Computing, volume 6(1), 322–350, 1977.
28. Charras, C. and T. Lecroq, Handbook of Exact String Matching Algorithms. First Edition. King's College London Publications. ISBN: 0954300645, 2004.
29. Hume, A. and D. Sunday, Fast string searching. Software Practice Experience, 21: 1221-1248, doi: 10.1002/spe.4380211105, 1991.
30. Lecroq, T., Experimental results on string matching algorithms. Software-practice and Experience, 25: 727-765, 1995. doi:10.1002/spe.4380250703.
31. Davies G., and Bowsher S., Algorithms for pattern matching, Software-Practice and Experience, 16:575-601, doi:10.1002/spe.4380160608, 1996.
32. Monther Aldwairi, Thomas Conte, Paul D. Franzon: Configurable string matching hardware for speeding up intrusion detection. SIGARCH Computer Architecture News 33(1): 99-107, 2005.
33. D. M. Sunday, A Very fast substring search algorithm, Commun. ACM, 33, PP 132-142, 1990.
34. Available:http://www-igm.univ-mlv.fr/~lecroq/string/node28.html [accessed 25.07.2016]
35. Available:http://www-igm.univ-mlv.fr/~lecroq/string/node29.html [accessed 25.07.2016]