DOI: 10.5604/01.3001.0010.5214

APPLYING A Q-GRAM BASED MULTIPLE STRING MATCHING ALGORITHM FOR APPROXIMATE MATCHING

Robert Susik
Lodz University of Technology

***Abstract**. We consider the application of multiple pattern matching (Multi AOSO on q-Grams) algorithm for approximate pattern matching. We propose the on-line approach which translates the problem from approximate pattern matching into a multiple pattern one (called partitioning into exact search). Presented solution allows relatively fast search multiple patterns in text with given k-differences(or mismatches). This paper presents comparison of solution based on MAG algorithm, and [4]. Experiments on DNA, English, Proteins and XML texts with up to k errors show that the new proposed algorithm achieves relatively good results in practical use.*

Keywords: text processing, approximate string matching, string algorithms, q-gram

ZASTOSOWANIE ALGORYTMU WYSZUKIWANIA WIELU WZORCÓW OPARTEGO O TECHNIKĘ Q-GRAMÓW DO WYSZUKIWANIA PRZYBLIŻONEGO

***Streszczenie.** Rozważamy zastosowanie algorytmu wyszukiwania wielu wzorców (Multi AOSO on q-Grams) do wyszukiwania przybliżonego. Proponujemy rozwiązanie on-line, upraszczające problem wyszukiwania przybliżonego do wyszukiwania wielu wzorców. Zaprezentowane rozwiązanie umożliwia relatywnie szybko wyszukiwać wiele wzorców dla odległości Levenshteina (lub Hamminga) z ograniczeniem do k. W artykule porównane jest rozwiązanie oparte na algorytmie MAG oraz [4]. Badania eksperymentalne przeprowadzone na zbiorach DNA, English, Proteins and XML z różnymi wartościami k wykazały, że zaproponowany algorytm osiąga relatywnie dobre wyniki w praktycznym zastosowaniu.*

Słowa kluczowe: przetwarzanie tekstu, wyszukiwanie przybliżone, algorytmy tekstowe, q-gram

Introduction

Approximate string matching is well known and widely used problem in stringology, with applications in spell checking, spam filtering, matching of nucleotide sequences, etc. Given pattern $P_{1..m}$, text $T_{1..n}$, both consisting of σ characters (called alphabet Σ), where $m \leq n$, find all positions in the text where the pattern matches the text up to k errors (which is the maximal number of differences/mismatches). We specify three operations that cause difference: insertion, deletion, and substitution. The difference between two strings is also called difference ratio which is defined as $\alpha = k / m$.

Multiple pattern matching is a classic problem with applications in bioinformatics, bibliographic data analysis, information retrieval, virus detection, data filtering, and other areas. The problem is to find positions of patterns $P = \{P_1, \ldots, P_r\}$ in the text T of length n where text and patterns are over common alphabet Σ of size σ. The patterns may be considered as of the same size or different sizes.

As stated before the approximate string matching problem is fundamental problem in text processing and heavily explored [1, 2, 4, 6–8, 10]. Most classical models are based on filtration method with verification, e.g., Levenshtein or Hamming distance. The pattern matching problems can be divided into two categories: on-line and off-line (index, semi-index). We consider the online approach in this paper.

Our method is closely related to previous work [4]. Authors deeply explore the use of multiple pattern matching algorithm in approximate pattern matching problem. The algorithm is optimal on average for low and intermediate difference ratios (up to 1/2). Authors present a couple of variants of this algorithm and we refer to all of the variants using single name AOSMASM.

The purpose of this research is to adapt MAG algorithm to approximate string matching with k-differences/mismatches, to examine its efficiency and to compare with other existing solution.

1. Our Approach

1.1. Multi AOSO on *q*-Grams (MAG)

Multi AOSO on *q-grams* is on-line algorithm designed for multiple pattern matching. It scans the text only once to find a set of patterns and returns its positions in the text. MAG is a complex algorithm based on Shift-Or (AOSO) with use of many techniques widely adapted in text processing such as *q-grams*, pattern superimposition, bit-parallelism and alphabet size reduction. We chose this algorithm because it achieves quite good results in practical use, and fit to our requirements.

MAG uses *q-grams* which are a contiguous (or non-contiguous [3]) substring (factor) of *q* characters of a string. The *q-grams* have been widely used in approximate (single and multiple) string matching [10] as a filtering method, but also to speed up exact matching of a single pattern by treating the *q-grams* as a super-alphabet [5].

1.2. Counting filter

Counting filter [2, 6, 7] is a filtration method used to discard most of the space which does not meet a certain criterion. It is the one of the most popular algorithms used for finding approximate patterns with k-differences (=matching with up to k Levenshtein errors) or k-mismatches (=matching with up to k Hamming errors). The filter is based on simple idea, the algorithm counts the number of each symbol of the alphabet in the pattern and then moves the window (of size m) through the text checking how much the number of corresponding symbols existing in the window differs from the pattern. If this number is less or equal k then verification is triggered and possible match reported.

Lemma ([3]): *If there are $i \leq j$ such that $ed(T_{i..j}, P) \leq k$, then $T_{j-m+1..j}$ includes at least $m - k$ characters of P.*

1.3. MAG for approximate pattern matching (MAGA)

We present a solution that allows relatively fast searching for the pattern in text with given k-differences (or mismatches). Our solution is based on following Lemma:

Lemma: *If pattern $P_{1..m}$ can be divided into $k+1$ sub-patterns, the pattern with k-differences (or mismatches) can be found by searching all of the sub-patterns in the text $T_{1..n}$ and verifying all found positions for k-difference (or mismatches) matching.*

In other words, we search k+1 pieces of the pattern in the text using multiple pattern matching and when one of them is found we verify if the pattern with k-differences exists on found position. For example, if P = *"abcdemogpcba"* and k = 1, then we divide the pattern P into two pieces (sub-patterns) P_1 = *"abcdem"*, P_2 = *"ogpcba"*. As may be noticed now, if one character inside the pattern is modified, deleted or added, one of these two sub-patterns changes while the second one is exactly the same.

For instance we change the second position in P so that we have P' = *"axcdemogpcba"* which is giving P'_1 = *"axcdem"* and P'_2 = *"ogpcba"*. Comparing sub-patterns of P and P' we find that P_2 and P'_2 are still the same. These sub-patterns may be found with a single pattern matching algorithm by running it k+1 times, but a faster solution involves using a multiple pattern matching algorithm.

We use the algorithm Multi AOSO on q-Grams (MAG) [9], described in section 1.1, for searching all the k+1 pieces of the pattern; if any piece is found in the text the verification is triggered. The verification uses the Counting Filter combined with dynamic programming (for Levenshtein distance). In order to find the exact position of match algorithm needs to determine where exactly the filter should be started (1) and finished (2):

$$start = pos - offset - k \quad (1)$$

$$end = start + m + 2 * k \quad (2)$$

where *pos* is the position of sub-pattern in the text, *offset* is the position of sub-pattern in the original pattern, and *k* is number of allowed differences. When filter finds position where the pattern with *k*-differences may occur, then verification method is executed. For example, for given *k* = 2 the pattern P = "*GGACACCAGAGGCGGGGA*" is divided into three sub-patterns P_1 = "*GGACAC*", P_2 = "*CAGAGG*", P_3 = "*CGGGGA*" which are merged into single pattern P" = *[CG][AG][AG][ACG][AG][ACG]* where each of symbol is super symbol in super alphabet. The original sub-patterns are stored with the offset and used for later verification. If such pattern P" is found then algorithm looks up the sub-pattern that matches the position and then verifies if the pattern P matches with up to *k*-differences in text window that starts on (1) and ends on (2) position (Fig. 1).

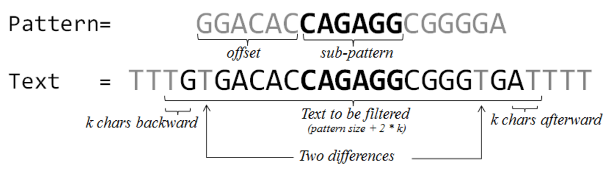


Fig. 1. Verification of searching pattern P=GGACACCAGAGGCGGGGA where the sup-pattern CAGAGG was found

We use Levenshtein implementation to validate the position. If the validation is successful (the number of differences is less or equal *k*) then the match is reported. We can easily adapt it to *k*-mismatches by running the Hamming distance verification in place of Counting Filter on *pos* – *offset* position.

2. Preliminary experimental results

The performance of the proposed solution was evaluated on datasets from the widely used Pizza & Chili corpus (http://pizzachili.dcc.uchile.cl/); we used 200MB files of *DNA*, *English*, *Proteins* and *XML* texts.

The codes of competitors were obtained from the authors and compiled as suggested. All our codes were implemented in C++ and compiled with g++ -O3. The computer was equipped with an Intel i3-2100 CPU 3.1 GHz (128KB L1, 512KB L2 and 3 MB L3 cache) and 4 GB of 1333MHz DDR3 RAM, and running Debian 3.2.63 x86 64.

In all experiments we ran MAGA with AOSO parameters set to *U* = 4 and *K* = 2. The parameter *q* (i.e., the *q-gram* size) used in all tests was set to {2, 4, 6, 8}. We decided to choose two variants of MAG with different alphabet mapping. We used combined alphabet mapping (*q-grams* creation is done on the fly – without mapping table) *mag_dna_lx* for *DNA* and *mag_lx* for the other datasets, where *x* is the value of *l* parameter which specifies the size of super alphabet (2^l). We used different values of *l* parameter for *DNA* and other datasets as follows: for *DNA* we set *l* = 2 for *m* > 32 and *l* = 3 for *m* ≤ 32, for all other datasets the *l* parameter was fixed to 3 for *k* = 1 and *m* = 128 but *l* = 4 for other *m* and *k* permutations.

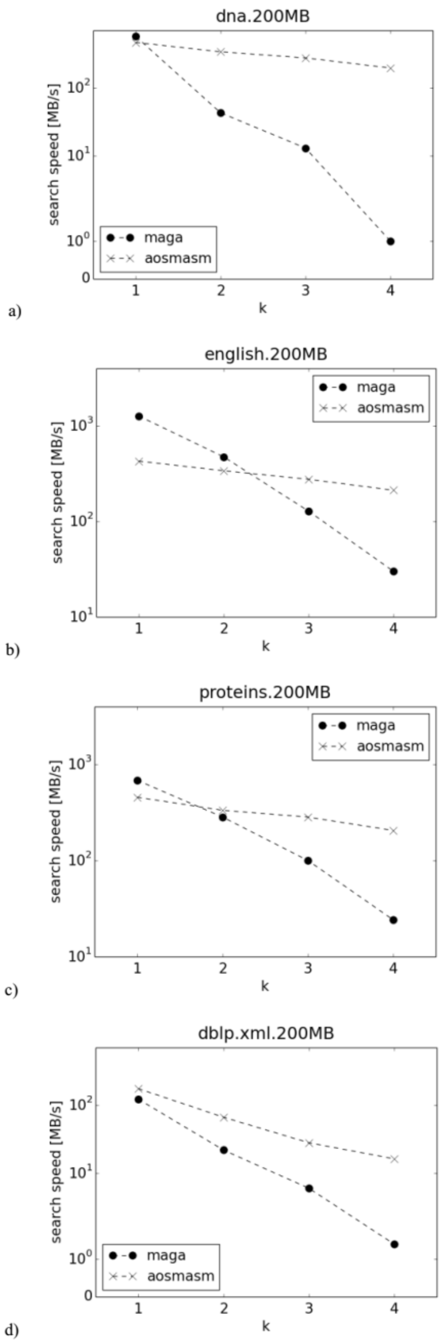
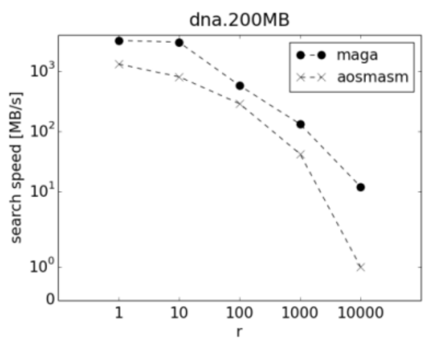
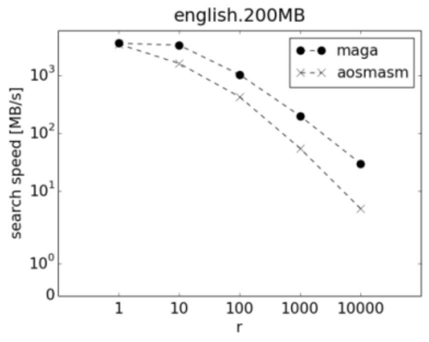
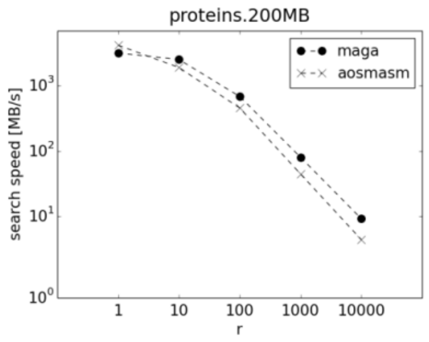


Fig. 2. Search speed of MAGA and AOSMASM for varying number of differences k = {1, 2, 3, 4}, r = 100, and m = 64. Results for a) english.200MB, b) dna.200MB, c) proteins.200MB, d) dblp.xml.200MB

The pattern size is constrained by two major factors which are the *q-gram* size and number of differences (*k*), therefore we narrowed the parameters as follows: for *k* = 1, *m* ≥ *16*, for *k* = {2, 3}, *m* ≥ 32, and for *k* = 4, *m* ≥ 64. The AOSMASM algorithm was tested with all possible parameters described by authors (see [1] for more detail). There are too many variants of mentioned algorithms to present on chart so we decided to present only the most efficient variants (the best result) of the solution.

a) 

b) 

c) 

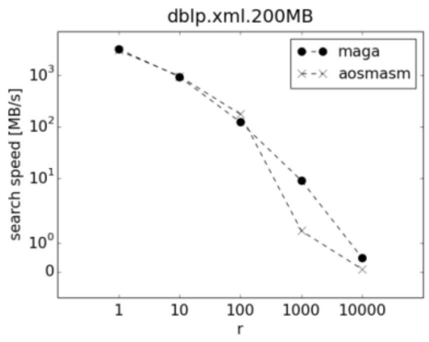
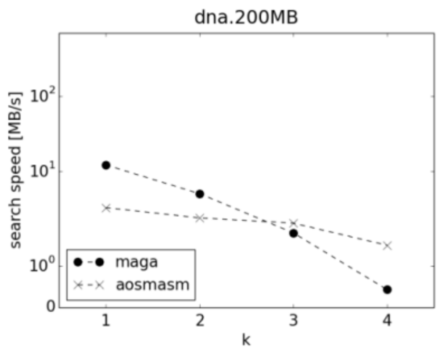
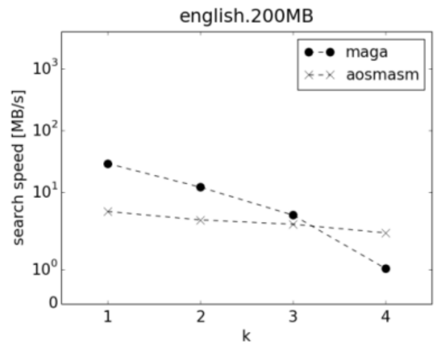
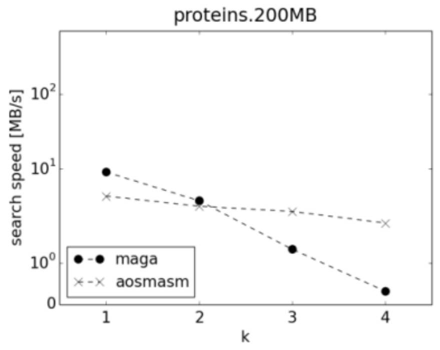
d) 

Fig. 3. Search speed of MAGA and AOSMASM for varying number of patterns r = {1, 10, 100, 1000, 10000}, k = 1, m = 64. Results for a) dna.200 MB, b) english.200 MB, c) proteins.200MB, d) dblp.xml.200 MB

Figure 2 presents search speed (in MB/s) of 100 patterns in 200 MB file of *DNA*, *English*, *Proteins*, *XML* texts in function of *k*. MAGA is much more effective than AOSMASM for *k* less than 3 for English alphabet (Fig. 2b) but function of our solution is decreasing much faster giving worse speed for *k* equals 3. The results for *DNA* and *Proteins* (Fig. 2a, 2c) are worse so that AOSMASM is only little worse for *k* equals *1* but much better for bigger *k* (up to two orders of magnitude). It can be reason of quite small alphabet. A very small alphabet as in the case of DNA may cause that adjacent q chars practically never produce unique q-grams, which in turn triggers the verification more often. The worst case is for *XML* (Fig. 2d) file where MAGA has worse result for all k. This may be caused by the nature of *XML* files where tags many times repeat in the text. This phenomenon has impact on uniqueness of *q-grams* causing many verifications.

a) 

b) 

c) 

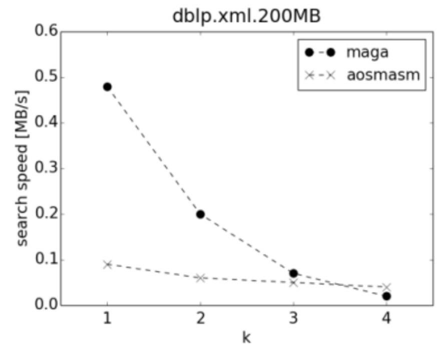
d) 

Fig. 4. Search speed of MAGA and AOSMASM for varying number of differences k = {1, 2, 3, 4}, r = 10 000, and m = 64. Results for a) english.200 MB, b) dna.200 MB, c) proteins.200 MB, d) dblp.xml.200 MB

Figure 3 illustrates the effectiveness of mentioned algorithms in function of *r* (number of patterns). The chart shows that the performance of both solutions is almost the same for one pattern, but the advantage of MAGA grows together with a growing number of patterns. The performance ratio (search speed of MAGA divided by speed of AOSMSAM) of both algorithms equals 1.03 for *r* = 1, while for *r* = 10000 it equals 6 for *English* (Fig. 3b) and up to 12 for *DNA* (Fig. 3a). MAGA having worse results (perf. ratio 0.76) for *r* = 1 for *Proteins* (Fig. 3c) got performance ratio on the level of 2.09 for *r* = 10000. The results are optimistic also for *XML* file (Fig. 3d) what is weak point in Fig. 2 but has much higher effectiveness for *r* = 1000 and *r* = 10000.

Figure 4 presents search speed of AOSMASM and MAGA in function of *k*. In this figure we examine the behaviour of both solutions when the number of patterns is increased from $r = 100$ (Fig. 2) to $r = 10000$ (Fig. 4). As expected, the performance of MAGA is much improved (in comparison to AOSMASM) when number of pattern is increased. For *DNA* (Fig. 4a) and *Proteins* (Fig. 4c) MAGA is more efficient than AOSMASM for *k* less than 3, but for *English* and *XML* the results are higher for *k* less or equal 3. Overall, it cannot be clearly specified which solution is better for given *k* because many other parameters (i.e. alphabet size, number of patterns) have much influence on the performance.

Figure 5 shows performance results of searching 100 patterns in *DNA*, *English*, *Proteins* and *XML* text in function of pattern length. The results show that MAGA achieves better results for patterns longer than 32 for English and DNA (Fig. 5a, 5b), for longer than 64 for XML (Fig. 5d) and longer than 16 for Proteins (Fig. 5c). MAGA is designed on the top of MAG algorithms which is using *q-grams* that have major impact on the performance. The speed may be raised by increasing *q* size what on the other hand is limited by pattern length. This enforces use of smaller *q* size for shorter patterns causing performance issue. This is one of the reasons why MAGA achieves much better results for long patterns than small ones.

3. Conclusions and future work

Experiments show that the proposed algorithm achieves relatively good results in practical use. MAGA is more efficient than AOSMASM if *k* is relatively small, but it can by improved if large number (i.e. 10k) of patterns is searched (6-fold speedup). Taking into account that MAGA handles searching of large number of patterns better than competitors the results may be more optimistic for a couple of tests we did for 100 patterns. MAGA achieves satisfactory results in all cases where is need to search large number (1000, 10000, etc.) of long (>32) patterns in datasets with quite big alphabet (>4) and small number of differences (≤3). We found that MAGA algorithm may be applied to different problems of approximate pattern matching. There is still a lot of research and experimental work to be done in the future, concerning using various AOSO parameter combinations, testing on a larger number of patterns, using different alphabet mapping (other variants of MAG) and different datasets. We believe there is a significant potential in the proposed approach, which should stimulate future research.

References

[1] Baeza-Yates R.A., Navarro G.: New and faster filters for multiple approximate string matching. Random Structures and Algorithms 20(1), 2011, 23–49.

[2] Baeza-Yates R., Navarro G.: New and Faster Filters for Multiple Approximate String Matching. Random Structures and Algorithms 20/2002, 23–49.

[3] Burkhardt S., Kärkkäinen J.: Better filtering with gapped q-grams. Fundam. Inform. 56(1-2)/2003, 51–70.

[4] Fredriksson K., Navarro G.: Average-optimal single and multiple approximate string matching. ACM J. Exp. Alg. 9(1.4)/2004, 1–47.

[5] Fredriksson K.: Shift–or string matching with super-alphabets. Information Processing Letters 87(1)/2003, 201–204.

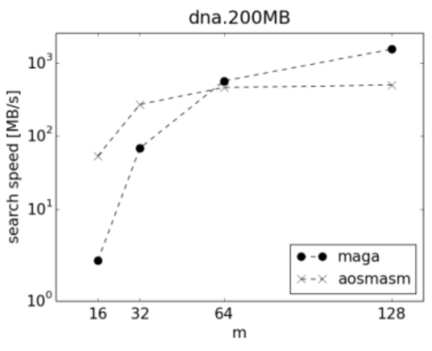
[6] Grossi R., Luccio F.: Simple and efficient string matching with k mismatches. Information Processing Letters 33(3)/1989, 113–120.

[7] Jokinen P., Ukkonen E.: Two algorithms for approximate string matching in static texts. Proc. MFCS 16/1991, 240–248.

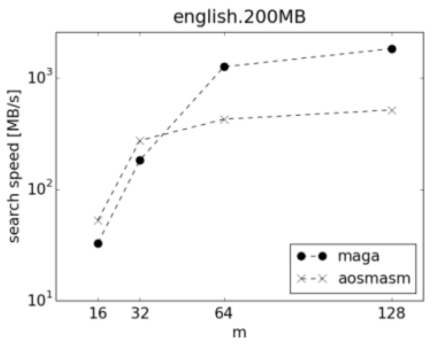
[8] Landau G.M., Vishkin U.: Fast string matching with k differences. Journal of Computer and System Sciences 37(1)/1988, 63–78.

[9] Susik R., Grabowski S., Fredriksson K.: Multiple Pattern Matching Revisited. Proceedings of PSC 2014, 59–70.

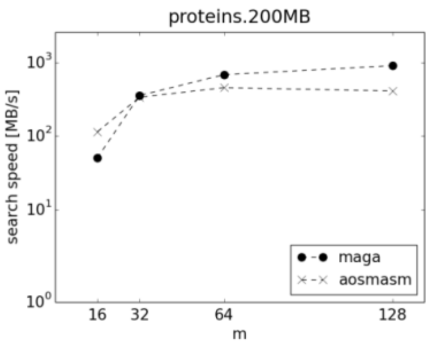
[10] Ukkonen E.: Approximate string-matching with q-grams and maximal matches. Theoretical Computer Science 92/1992, 191–211.



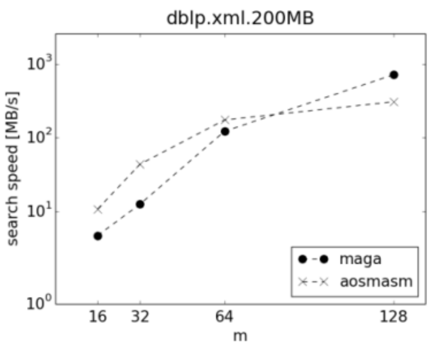
a)



b)



c)



d)

Fig. 5. Search speed of MAGA and AOSMASM for varying pattern length m = {16, 32, 64, 128}, r = 100, and k = 1. Results for a) english.200 MB, b) dna.200 MB, c) proteins.200 MB, d) dblp.xml.200 MB

M.Sc. Eng. Robert Susik
e-mail: rsusik@kis.p.lodz.pl; robert.susik@gmail.com

M.Sc. Eng. Robert Susik graduated from Lodz University of Technology, Faculty of Electrical, Electronic, Computer and Control Engineering. In 2012 he participated in the Erasmus student exchange programme and studied at the University of Eastern Finland. Currently Ph.D. student at the Institute of Applied Computer Science of Lodz University of Technology. He is interested in data processing, data archiving and string matching algorithms.

otrzymano/received: 06.10.2015 *przyjęto do druku/accepted: 14.08.2017*