http://dx.doi.org/10.35596/1729-7648-2020-18-1-29-34

Оригинальная статья
Original paper

UDC 681.142.2

A SEARCHING ALGORITHM FOR TEXT WITH MISTAKES

SARA NASR, OLEG V. GERMAN

The Belarussian State University of Informatics and RadioElectronics (Minsk, Republic of Belarus)

Submitted 17 May 2019

Abstract. The paper contains a new text searching method representing modification of the Boyer-Moore algorithm and enabling a user to find the places in the text where the given substring occurs maybe with possible errors, that is the string in text and a query may not coincide but nevertheless are identical. The idea consists in division of the searching process in two phases: at the first phase a fuzzy variant of the Boyer–Moore algorithm is performed; at the second phase the Dice metrics is used. The advantage of suggested technique in comparison with the known methods using the fixed value of the mistakes number is that it 1) does not perform precomputation of the auxiliary table of the sizes comparable to the original text sizes and 2) it more flexibly catches the semantics of the erroneous text substrings even for a big number of mistakes. This circumstance extends possibilities of the Boyer–Moore method by addmitting a bigger amount of possible mistakes in text and preserving text semantics. The suggested method provides also more accurate regulation of the upper boundary for the text mistakes which differs it from the known methods with fixed value of the maximum number of mistakes not depending on the text sizes. Moreover, this upper boundary is defined as Levenshtein distance not suitable for evaluating a relevance of the founded text and a query, while the Dice metrics provides such a relevance. In fact, if maximum Levenshtein distanse is 3 then how one can judge if this value is big or small to provide relevance of the search results. Consequently, the suggested method is more flexible, enables one to find relevant answers even in case of a big number of mistakes in text. The efficiency of the suggested method in the worst case is $O(nc)$ with constant c defining the biggest allowable number of mistakes.

Keywords: searching algorithm, text processing, finding text with mistakes, fuzzy Boyer–Moore method, metrics of similarity.

Conflict of interests. The authors declare no conflict of interests.

For citation. Nasr S., German O.V. A searching algorithm for text with mistakes. Doklady BGUIR. 2020; 18(1): 29-34.

Introduction

Text searching methods are widely required in modern text-based applications. Let us note CV (Curriculum Vitae) and paper abstracts processing, extracting an invention formula description from patents, ***e***-mails filtering and so on. The problem is stated generally as answering queries to some text(s). To find an answer different text searching techniques can be applied [1, 2]. Clearly, the efficiency of a text searching technique essentially depends on the text sizes, its structure and presence of syntax mistakes. From this viewpoint, the methods based on text tagging are not suitable if text is poorly organized and unstructured. The keyword-based seach methods are not

efficient in large text databases, for instance, on web-servers proposing job offers with billions of files. The regular expression based methods may be not efficient if text contains mistakes as it is difficult (if not possible) to envisage the concrete mistakes in text of different topics. So, we selected sequential text searching approach which seems to be most reasonable in the case of short, poorly organized texts with different topics and especially in case with a lot of text files to be processed. It is the Boyer–Moore method [3] we selected as a prototype in our approach. It can be efficiently used to find a substring in the text with computational complexity estimated as $O(n+m)$ where n is size of the text and m stands for a query length. However, this method presumes exact coincidence of the original substring and those which are being looked for. The methods to find substring occurences in text with mistakes can be found in [4, 5]. They are different from the Boyer–Moore scheme though. These methods suggest that some constant $k \geq 0$ restricts the utmost mistakes amount. However, it is difficult to define the value of k a priori especially if text is unknown and overloaded with mistakes. We suggest to use two constants $k_2 > k_1 \geq 0$ and organize a fuzzy searching procedure as follows. The modified Boyer–Moore procedure continues till the number of the detected mistakes not exceeds k_2, otherwise the right-side shift of the query is performed with computed shift value. If the number of mistakes does not exceed k_1 then the searched text word is recognized by means of the modified Boyer–Moore method. If the number of mistakes lays in the diapazone $[k_1 + 1, k_2]$ then the text word is processed with the help of fuzzy comparison based on Dice metrics [6]. If Dice metrics is greater than 0.5 then the answer is found. Otherwise the answer is not found and right-side shift is performed accordingly to the modified Boyer–Moore strategy. We consider a query string as a correct one and modify currently observed substring in the text as a copy of the query. We use Levenstein distance between the strings to estimate their similarity. It is used as a marker signalling that strings (or their parts) are equal or different. The main problem is to define the value of the shift (the number of symbols to omit) to be made if the compared substrings are different. The rest of the paper explains the details of how this and the other details are realized.

The Boyer–Moore algorithm illustration

To be more concrete let us consider the following example.

Text string: *awbsycsdesacscdssqwass*. Substring to find: *sacs*.

Let's consider an approach of Boyer-Moore [3]. First, in the substring to find one defines the number of occurrences of each symbol and their corresponding positions. Thus, with respect to our substring one has: <*c* – 1(2); *a* – 1(3), *s* – 2(1,4)>. In the round brackets the symbol offsets from the end of the word are placed, for instance, *c* – 1(2) means that symbol *c* is encountered only once in the word *sacs* and takes the second position from the right. At the second step the searching procedure starts by sequential comparison of the last symbol in *sacs* (i.e. – *s*) with the currently observed symbol in the Text string until the coincidence takes place. Thus, we have:

a w b s y desacscdssqwass
s a c s

Now let's compare the corresponding symbols from right to left. Because *c* < > *b* the procedure stops. It is possible to shift word *sacs* to the right by 3 positions to superpose the first symbol *s* in *sacs* with the currently observed symbol *s*. Indeed, no other symbols *s* besides the leftmost and the rightmost ones in *sacs* are available. So the next comparison is realized as below

awb s y c s desacscdssqwass
s a c s
etc.

Mistakes in the strings and their correction

The next errors in words are possible:

– *S1*: a letter is wrong in the given position;

– *S2*: a letter is omitted;

– *S3*: extra letter is inserted;

– *S4*: two adjacent letters are interchanged

We need some criteria to reveal the above pointed states. Our consideration is restricted with these situations only.

Situation S1. To reveal this situation one needs to read the corresponding next symbols in both compared strings. If those symbols coincide then the resulting answer is YES (true) otherwise – NO (false). The NO answer is generated also if there are no next symbols (in both or some of strings).

Situation S1.
B R I D **G** E
B R I D **D** E

Situation S4.
B R I **D G** E
B R I **G D** E

Situation S2.
B R I D **G** E
B R I D E

Situation S3.
B R I D G E
B R I D **H** G E

While comparing the strings we compute the Levenstein distance between them. We use this distance to come to conclusion that two substrings are different or not. For this, we base ourselves on the experimental results [7] presented in Table 1 and giving value of k_1

Table 1. Definition of k_1

Word size	The uppermost number of text errors – k_1
≤ 3	0
4–6	1
7–8	2
9–10	3
≥11	4

We, however, change Table 1 to define k_2 as in Table 2.

Table 2. Definition of k_2

Scanned word size	The uppermost number of text errors – k_2
≤ 3	1
4–6	2
7–8	3
9–10	4
≥11	6

The text errors here correspond to situations *S1*, *S2*, *S3*, *S4* described above. The situation *S4* is considered as a single error.

Modified Boyer–Moore algorithm

The idea of method based on [4, 5] can be explained as follows. We try to simulate the Boyer-Moore algorithm comparing symbol by symbol from right to left in both strings with query under the text. The query string is supposed to be written correctly. If symbols do not coincide we admit that some mistake is encountered in the CV-text. We define a type of mistake (situations *S1*, *S2*, *S3*, *S4*). According to the Table 2 we test if the number of mistakes is still admissable, that is lies in diapason $Dp = [0, k_2]$. If the answer is positive, we make suitable shifts in both strings. In this case we replace an erroneous place in the CV-text by a supposedly correct symbol from the query. If the number of mistakes is beyond the diapason *Dp* we restore the corrected mistakes in the CV-text and make a right-side shift of the query. We explain how to define the shift value below by means of example.

Let us make some illustration for $k_1 = 1$, $k_2 = 3$.

CV-text: x x x ... x ◊□□◊◊□□ Θ Θ Θ Θ ... Θ
query: □□□□□□□

We use the symbol □ to designate exactly coinciding symbol with the corresponding one in the query; the symbol ◊ is used to designate a mistaken symbol which was changed; Θ (x) is used to designate the rest right-side (left-side) symbols in the currently observed part of CV**.** We admit maximum 3 non-coincidences in both substrings. There are 3 mistakes in the example above. So we should compute the Dice distance (metrics) between two strings. Suppose, that the Dice metrics is less than 0.5. This means that we need to make a right-side shift of the query. Now let us suppose that there are no the same (repeating) symbols in the query. Our task is to define the shift value. Let us make one-symbol right-side shift of the query. The situation will change as shown below

CV-text: x x x ... x □ ◊◊**??**◊ ◊**?** Θ Θ Θ ... Θ
query: □□□□□□□

Here we use symbol **?** to designate a new position of the symbol which should be tested. As one can see the number of mistakes has not reduced (there are even 4 mistakes designated as ◊). As we admit maximum 3 mistakes a new shift is required with the following result:

CV-text: x x x ... x □□◊**??**◊◊**??** Θ Θ ... Θ
query: □□□□□□□

This time there remain 3 mistakes. One should compute the Dice metrics again in this new configuration. Suppose the Dice metrics is less than 0.5. Again one symbol right-side shift is required:

CV-text: x x x ... x □□□**??**◊◊**???** Θ Θ ... Θ
query: □□□□□□ □

Now there are 2 mistakes remained and 5 positions marked with **?** to be tested. So, verification should resume with the rightmost symbol **?**. The common **rule A** to define shift value when no repeating symbols occur in the pattern can be defined as follows. Let us consider the next configuration

CV-text: x x x ...x◊ □□◊◊□□ Θ Θ Θ Θ ... Θ
query: ***** □□□□□□□□

Let there are allowed two symbols ◊ in the upper string. Let ***** stand for the leftmost symbol in the query. The required shift value to be done defines a new position of the ***** where the number of ◊ in the text does not exceed the allowable value $k_2 = 2$. This position can be exactly found as it corresponds to the first symbol □ in the text such that total number of ◊ to the right of it does not exceed the allowable amount of mistakes, i.e. 2. Accordingly to this rule we get new configuration as below

CV-text: x x x ... x ◊ □□◊◊□□ΘΘΘΘ ... Θ
query: *****□□□□□□□□

The shift value is 5.

Suppose now that there are repeating symbols in the query. Let us illustrate the situation as follows

CV-text: x x x ... x◊ □□◊◊□□ ΘΘΘΘ ... Θ
query: ***** □♦□□□□♦□

We designated repeating symbol as ♦. One can apply the above given technique for the non-repeating symbols. As before one should define a new position of ***** with two (or less) pairs ◊ – □ in both strings after *****. The desired configuration is shown below

CV-text: x x x ... x ◊□□ ◊◊□ □ΘΘΘΘ ... Θ
query: *****□♦□□□□♦□

Again it required of us to make 5 shifts. Notice that configuration preceding the previous one was that one below with 3 mistakes what exceeds the allowed number of mistakes

CV-text: x x x ... x ◊ □□◊◊□□ΘΘΘΘ ... Θ
query: *□♦□□□□♦□

Here are 3 mistakes and one more right shift is required.

We estimate the worst case complexity of the method for the unrepeating symbols case admitting that the mean number of mistakes allowed is *c*.

The worst case means *c* + 1ending miscorrespondings with only one shift to the right each time. This gives $n\cdot(c+1)$ comparisons at the worst. Suppose the query length is $k > 3$. Then at the worst we make *k* comparisons and 3 shifts to the right. We conclude these considerations with the worst case shown below

CV-text: x x x ... xx xxx◊◊◊Θ ΘΘΘ ... Θ
query: □□□□□□

Let us suppose that 2 mistakes are allowable and all the mistakes are forming the final subsequence. Evidently, it is possible only one right shift resulting in

CV-text: x x x ... x xx xx**????** ΘΘΘ ...Θ
query: □□□□□□

and so on.

Addressing the Dice metrics

Let $k_1 = 1$ and $k_2 = 2$.

1. *a s b c y esaceescdssqwass*
   s a c d e

There may be at the most two mistakes. Here are 3. As no repeating symbols are contained in the query the right shift value is 1:

2. *a s b c y e s a ceescdssqwass*
   s a c d e

There are 2 mistakes. Accordingly to our algorithm one shoud find the Dice distance between two strings: *s b c y e* and *s a c d e*. The Dice metrics is defined by the formula $P = \frac{2 \cdot |X \cap Y|}{|X| + |Y|}$, where $|X|$ ($|Y|$) stands for the set *X* (*Y*) size. For example, for *X* = *s b c y e* and *Y* = *s a c d e* one has $P = 2\cdot 3 / (5+5) = 0.6$. We adopt the rule accordingly to which two words are considered alike if the Dice measure is 0.5 or greater. From this, in the example above the words *X* and *Y* are considered to be the same.

Summarization of the searching procedure can be represented in the next form.

1. At each iteration the procedure compares symbol by symbol from right to left in both strings with query under the text (the query string is supposed to be written correctly).

2. If all symbols in the query have been compared with CV-text fragment (*cvf*) then.

2.1 If the number of mistakes encountered is in diapason $Dp = [0, k_1]$ then *cvf* is accepted as successfully recognized and searching procedure resumes from the next right symbol following *cvf* (if any, otherwise the procedure finishes).

2.2 If the number of mistakes encountered is in diapason $Dp = [k_1+1, k_2]$ then *cvf* is compared with query by means of Dice metrics. If comparison is successfull the *cvf* is accepted as successfully recognized and searching procedure resumes from the next right symbol following *cvf* (if any, otherwise the procedure finishes).

2.3 Otherwise (1.1 and 1.2 failed) the algorithm restores the corrected mistakes in the CV-text and makes a right-side shift of the query with resuming searching procedure from step 5.

3. If symbols do not coincide then some mistake is encountered in the CV-text. If the number of mistakes is still admissable, that is, lies in diapason $Dp = [0, k_2]$ the algorithm produces suitable shifts in both strings and replaces an erroneous place in the CV-text by a supposedly correct symbol

from the query. Return to step 2.

4. If the number of mistakes is beyond the diapason *Dp* the algorithm restores the corrected mistakes in the CV-text and makes a right-side shift of the query.

5. If the rightmost symbol of the query is beyond CV-text then stop searching procedure, otherwise return to step 2.

Conclusion

The suggested method combines techniques of the fuzzy Boyer–Method with verifying string similarity on the basis of Dice metrics. This provides some important advantages in comparison with the Boyer–Moore method as it gives a possibility to increase the number of mistakes in the text and still to preserve the semantics of the original text. The other advantage of the suggested approach in comparison with the known fuzzy versions of the Boyer–Moore method consists in more sensitive regulation of the allowable upper number of errors in the word to be recognized. In existing approaches the number of mistakes is fixed and does not depend on the text length. Moreover, this number is defined in the procedure as Levenstein metrics (i.e. as absolute number of mistakes). This does not help to make a conclusion about processed text relevance to a query (on the contrary, the Dice metrics gives a clear answer). Indeed, if the Levenstein distance is equal to 3, then how one can judge if this number is big or, on the contrary, low. Therefore, the suggested method enables one to decrease k_1 and even increase k_2 to make the recognition process more efficient. The investigation of this and related questions remain the future landmarks of the suggested approach.

References

1. Manning Ch. D., Raghavan P., Schutze H. *An introduction to information retrieval*. Cambridge University Press; 2009.
2. Wu S., Manber U. Fast text searching allowing errors. *Communication of the ACM.* 2001;35:83-91.
3. Gusfield D. *Algorithms on strings, trees and sequences*. *Computer science and computational biology*. Cambridge university press; 1997.
4. Galil Z., Giancarlo R. Data structures and algorithms for approximate string matching. *Journal of Complexity*. 1988;4:33-72.
5. Gonnet G.H., Baeza-Yates R.A. *Handbook of algorithms and data structures*. Addison-Wesley. 1991.
6. Mandel I. [*Cluster analysis*]. M.: Finansy i statistika; 1988. (In Russ.)
7. Gourine N.I., German O.V. [Intelligent query analyzer to knowledge base of the multimedia electronic tutorial]. *Trudy BGTU=Proceedings of BSTU*. 2010;18(6):167-170. (In Russ.)

Authors contribution

Nasr S. has developed the detailed specification of the fuzzy substring searching method.
German O.V. suggested the common idea of the method.

Information about the authors

German O.V., PhD, Associate Professor of the Information Technologies in Automatized Systems Department of Belarussian State University of Informatics and Radioelectronics.

Nasr S., PG student of Information Technologies in Automatized Systems Department of Belarussian State University of Informatics and Radioelectronics.

Address for correspondence

220600, Republic of Belarus,
Minsk, P. Brovki str. 6,
Belarussian State University of Informatics and Radioelectronics
+375-29-612-42-32;
e-mail: ovgerman@tut.by
German Oleg Vitoldovich