УДК 004:021
DOI: 10.25559/SITITO.16.202001.41-49

Constrained Approximate Search Algorithms in Knowledge Discovery

S. Petrović[a*], J. Sidorova[b]
[a] Norwegian University of Science and Technology, Gjøvik, Norway
22 Teknologiveien, Gjøvik 2815, Norway
[*] slobodan.petrovic@ntnu.no
[b] Blekinge Institute of Technology, Karlskrona, Sweden
SE-37179 Karlskrona, Sweden

Abstract

Knowledge discovery in big data is one of the most important applications of computing machinery today. Search is essential part of all such procedures. Search algorithms must be extremely efficient, but at the same time knowledge discovery procedures must not produce too many false positives or false negatives. False positives require post-processing, which reduces the overall efficiency of the knowledge discovery procedures, while false negatives reduce the sensitivity of such procedures. To reduce the false positive and false negative rate, in this paper, constrained approximate search algorithms are proposed to be applied. An overview of search theory, exact and approximate, is given first, exposing fundamentals of dynamic programming-based and bit-parallel-based approximate search algorithms without constraints. Then, introduction of constraints specific for various knowledge discovery procedures is explained, together with the subtleties of various applications, such as SPAM filtering and digital and network forensics (file carving, intrusion detection in hosts and networks). Advantages and disadvantages of applications of such constrained search algorithms in knowledge discovery procedures are also discussed. A potential application in bioinformatics is outlined.

Keywords: Knowledge discovery, Big data, Search, Constraints, Intrusion detection, Digital forensics, SPAM filtering, bioinformatics, activity alerts, chemical activity.

For citation: Petrović S., Sidorova J. Constrained Approximate Search Algorithms in Knowledge Discovery. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2020; 16(1):41-49. DOI: https://doi.org/10.25559/SITITO.16.202001.41-49

© Petrović S., Sidorova J., 2020

Контент доступен под лицензией Creative Commons Attribution 4.0 License.
The content is available under Creative Commons Attribution 4.0 License.

Ограниченные алгоритмы приближенного поиска при обнаружении знаний

С. Петрович[1*], Ю. Сидорова[2]
[1] Норвежский университет естественных и технических наук, г. Йёвик, Норвегия
2815, Норвегия, г. Йёвик, Технологический пр., д. 22
[*] slobodan.petrovic@ntnu.no
[2] Технологический институт Блекинге, г. Карлскруна, Швеция
SE-37179, Швеция, Округ Блекинге, г. Карлскруна

Аннотация

Обнаружение знаний в области больших данных является одним из наиболее важных приложений вычислительной техники сегодня. Поиск является неотъемлемой частью всех таких процедур. Алгоритмы поиска должны быть чрезвычайно эффективными, но в то же время процедуры обнаружения знаний не должны давать слишком много ложных срабатываний или ложных отрицаний. Ложные срабатывания требуют последующей обработки, что снижает общую эффективность процедур обнаружения знаний, в то время как ложные отрицания снижают чувствительность таких процедур. Чтобы уменьшить количество ложноположительных и ложноотрицательных результатов, в этой статье предлагается применять ограниченные приближенные алгоритмы поиска. Краткий обзор теории поиска, точной и приблизительной, дается вначале, раскрывая основы алгоритмов приближенного поиска на основе динамического программирования и на основе бит-параллелизма без ограничений. Затем объясняется введение ограничений, специфичных для различных процедур обнаружения знаний, а также тонкостей различных приложений, таких как фильтрация спама, цифровая и сетевая экспертиза (разделение файлов, обнаружение вторжений в хосты и сети). Также обсуждаются преимущества и недостатки применения таких ограниченных алгоритмов поиска в процедурах обнаружения знаний. Намечено потенциальное применение в биоинформатике.

Ключевые слова: обнаружение знаний, большие данные, поиск, ограничения, обнаружение вторжений, цифровая криминалистика, фильтрация спама, биоинформатика, оповещения об активности, химическая активность.

Для цитирования: Петрович, С. Ограниченные алгоритмы приближенного поиска при обнаружении знаний / С. Петрович, Ю. Сидорова. – DOI 10.25559/SITITO.16.202001.41-49 // Современные информационные технологии и ИТ-образование. – 2020. – Т. 16, № 1. – С. 41-49.

Introduction

Search in large data sets has always been one of the most important tasks for computing machinery. These data sets, together with the algorithms that are used for their processing, are nowadays often referred to as *big data*. As data quantities that the mankind produces every day grow at so high rates, new more efficient search algorithms are needed to discover knowledge in such enormous amounts of data. In addition, since data to analyze often arrive from heterogeneous sources and may contain errors, it is necessary to introduce tolerance in search in order to avoid false negatives in knowledge discovery procedures. To this end, many exact search algorithms have been modified in order to be able to detect data patterns with certain error tolerance.

The time complexity of the most efficient exact search algorithms is sub-linear [1], but even that is not enough when it comes to analyzing big data, where the size of the search strings can easily reach exabytes. In addition, these search algorithms are difficult to implement on classical computer architectures with limited computer word sizes if there is a need to detect longer patterns. In these cases, concatenation of computer words is necessary to process the search string, which significantly reduces the efficiency of such implementations (see, for example, [2]). Different computer architectures perform better in these cases. Typical examples are reconfigurable hardware platforms, such as Field Programmable Gate Arrays (FPGA), where it is possible to define a computer word, whose size is only limited by the memory capacity of the concrete FPGA card. On such architectures, the advantage of bit-parallelism can be fully exploited even for very long search patterns.

By incorporating error tolerance in search algorithms, we can reduce the probability of false negatives in search. However, very often we need to reduce the probability of false *positives* in such procedures as well. False positives are annoying for the user of a search algorithm since they require post-processing, which can eliminate them or reduce their number and these procedures often influence the overall efficiency of a search algorithm in a negative way. To reduce the number of false positives, it is useful to introduce certain constraints in search algorithms, which take into account a priori knowledge about the process that is analyzed [3]. Then it is possible to ignore the patterns that would otherwise be accepted by the (approximate) search algorithm, but for such patterns the a priori knowledge about the analyzed process indicates that they are impossible to happen.

In this paper, we present several constrained approximate search algorithms, where the constraints reflect the specific features of certain application fields, such as intrusion detection, cryptanalysis of stream ciphers, SPAM detection and so on. A common characteristic of these applications is possibility of searching for long patterns, which may cause difficulties in implementation on various computer architectures. In some cases, these pattern lengths are extremely long, such as the case when we want to break a stream cipher by means of a generalized correlation attack as well as in all the cases where we have to decide between several hypotheses and set a threshold of acceptance of one of them. Then the length of the search pattern directly and significantly influences the number of false positives in approximate search.

We discuss two categories of constrained approximate search algorithms: dynamic programming-based and bit-parallel algorithms. The advantages and disadvantages of these groups of algorithms are discussed for several (potential) fields of application.

The search problem

The basic definition of the search problem is the following: given the search pattern $w = w_1 w_2 \cdots w_m$ and the search string $S = s_1 s_2 \cdots s_n$, find the locations of all occurrences of w in S. The naïve algorithm to solve this problem making use of a sliding comparator (see an example in Fig. 1) has quadratic time complexity. Strictly mathematically speaking, the search problem does not belong to the category of the most difficult problems (it is in the class P), but our challenge is the length of the search string S, which may be extreme. In the case of an extremely long S, even the best search algorithms, whose time complexity is sub-linear are not efficient enough to process big data in a reasonable amount of time. Then combinations of theoretically efficient search algorithms and sophisticated implementations on parallel and distributed computer architectures give the best results.

```
      para
     para
    para
   para
  para
 para
papara zzo
```

Fig. 1. The naïve exact search algorithm, w ="para", S ="paparazzo"

Efficiency improvements from quadratic to linear time complexities are achieved by varying the way of sliding the window along the search string and the way how the pattern is searched for in the window. On the other hand, efficiency improvements from linear to sub-linear time complexities are achieved *on average*, by avoiding (skipping) the search in the regions of the search string, in which the search pattern is impossible to occur.

Exact search

Let $X = x_1 x_2 \cdots x_n$ be a string of symbols from an alphabet $\mathcal{A}$. Then the substring $V = v_i v_{i+1} \cdots v_j$ of S is a *prefix* of S if $i = 1$, a *suffix* of S if $j = n$, and a *factor* of S if $1 < i \leq j < n$.

The exact search algorithms are usually divided into three groups, according to whether their approach is prefix, suffix, or factor based. The most prominent prefix-based algorithms are Knuth-Morris-Pratt [4] and the bit-parallel algorithms Shift-AND [2], and Shift-OR (see, for example, [1]). The suffix-based algorithms that are most often used in practice are the Boyer-Moore algorithm [5] and its variants, Horspool [6] and Sunday [7]. The algorithms from the third, factor-based, group achieve the best performance on average, provided independent and uniformly distributed characters from the alphabet $\mathcal{A}$ make the search string S. The often-used algorithms from this group are Backward DAWG[1] Matching (BDM) (see, for example, [8]) and the bit-parallel Backward Non-deterministic DAWG matching (BNDM) [9].

[1] DAWG = Directed Acyclic Word Graph, see [8]

Even though there exist exact search algorithms (so-called skip algorithms) that are fastest on average, in some applications these algorithms must not be used. An example is intrusion detection, where an attacker could deliberately produce traffic that makes the search algorithm used in the Intrusion Detection System (IDS) perform poorly (much slower than on average) and exploit that to cause IDS packet drop, which opens the way to false negatives. Such attacks are called *algorithmic attacks* [10]. In these cases, the fastest algorithms that can be used are the ones from the non-skip category, such as the forward-based bit parallel algorithms, Shift-AND, and especially Shift-OR.

Bit-parallelism and exact search

Bit-parallel search algorithms simulate the Non-deterministic Finite Automaton (NFA) assigned to the given search pattern w. The automaton consists of states and has a simple linear register-like structure. For each input character from the search string S, such an automaton makes a transition from its current state into the next state, if it is possible, and makes a copy of itself continuing to receive the characters from S. Each such copy starts receiving the characters from the initial state (0). If for some input character from S, a transition from the current state into the next state for some copy of the NFA is not possible, that copy of the NFA becomes *inactive*, i.e. it stops receiving future characters. If the rightmost (double circled) state is reached by any of these copies for some input character, an occurrence of w is found in S (see, for example [11]). Instead of analyzing such an NFA as a set of copies of the basic linear structure, the concept of an *active state* is often used. Then, instead of modeling the functioning of the NFA by making copies and maintaining some of these copies active, an active state replaces the corresponding active copies of the basic structure. Each active copy is replaced by a single state. For example, consider the search pattern w ="attack" consisting of symbols from the English alphabet. The corresponding NFA without ϵ-transitions (the transitions without consuming any input character, see [11]) allowed is given in Fig. 2.

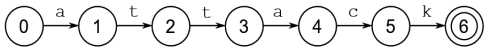


Fig. 2. The NFA assigned to w ="attack", without ϵ-transitions

If we allow ϵ-transitions, then the form of the NFA for the same search pattern is given in Fig. 3.

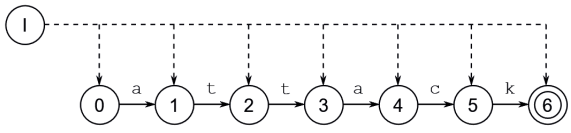


Fig. 3. The NFA assigned to w ="attack", with ϵ-transitions

Suppose the search string is S ="attentionattack". After receiving the prefix "att", the state 3 of the machine from Fig. 2 will be active, but after receiving the next character 'e' from S, no state will be active until the next character 'a' arrives. The machine from Fig. 3 that allows ϵ-transitions jumps immediately to the active state that corresponds to the prefix of S that it has processed. Thus, both representations, with and without ϵ-transitions, are equivalent.

The operation of the theoretical NFA described above assumes infinite parallelism. In practice, we can only simulate such an NFA, and to this end it is necessary to limit the number of copies of the basic linear structure present at a time to the length m of the pattern w. Baeza-Yates and Gonnet [2] were the first to describe an efficient simulation of this NFA. Suppose S ="paparazzo", $n = |S| = 9$, and w ="para", $m = |w| = 4$. As the new characters from S arrive, the created copies of the basic linear structure assigned to the search pattern w get inactive or remain active. The fact of being active or inactive is the only one that matters and since the activity status of one copy of the basic structure is binary, it can be encoded with a 0 (inactive) or a 1 (active). Since we can only have up to m basic structures at a time, we put the status bits of each such structure present at some time instant in a computer word, the *search status word* D. In our example, after 4 processed characters from S, the status word $D = 0010$ (Fig. 4).

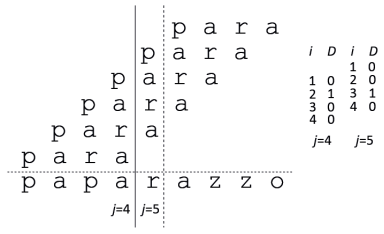


Fig. 4. Operation of the simulated NFA (see text)

The next character from S to process is 'r'. Since we cannot keep more than m basic structures at a time, we have to eliminate the oldest one. It is easy to see that by shifting the search status word D by one position to the left, we achieve this goal. The creation of the new copy of the basic structure reflects on D by OR-ing D with $0^{m-1}1$. The crucial step is then updating the search status word, all the bits at the same time (therefore bit-parallelism), by AND-ing D with a bit mask corresponding to the current processed character from S. The bit masks are defined in advance, by putting a 1 in the bit mask at the position where the corresponding character is located in the reversed search pattern w. Defining the bit masks in advance is possible since the basic structure, whose status bit is located at certain position in the search status word D always waits for the same character.

For the example from Fig. 4, the bit masks assigned to each character from the search pattern w are

$$B['p'] = 0001,\ B['a'] = 1010,\ \text{and}\ B['r'] = 0100. \quad (1)$$

Then, for the new search status word D' obtained after processing the next character 'r' from S we have

$$D' = \big((0010 \ll 1) \vee 0001\big) \wedge 0100 =$$
$$= 0101 \wedge 0100 = 0100 \quad (2)$$

By generalizing the equation (2), we get the Shift-AND search status word update formula

$$D' = \big((D \ll 1) \vee 0^{m-1}1\big) \wedge B[S_j], \quad j = 1, \ldots, n \quad (3$$

After each update of the search status word D, it is necessary to check whether the MSB of D is equal to 1, which would mean that an occurrence of the search pattern w would be found in the search string S. This completes the Shift-AND algorithm described in [2]. The inherent bit-parallelism of a computer word where the search status word is stored enables checking the status of all the simulated basic structures at the same time, which reduces the

time complexity of the search algorithm from essentially quadratic ($O(nm)$) to linear ($O(n)$).

The Shift-OR algorithm is often considered just a more efficient implementation of the Shift-AND algorithm (see, for example, [1]). Namely, if the bit masks and search status word D are complemented and a 0 is considered the encoding of an active basic structure in D then there is no need for OR-ing with $0^{m-1}1$ in the update formula (3) for D, which makes the algorithm 33% faster than the Shift-AND algorithm. The update with the bit masks is performed by OR-ing of the bit masks instead of AND-ing. Thus, the Shift-OR search status word update formula becomes very simple (4).

$$D' = (D \ll 1) \vee B[S_j], \quad j = 1, \dots, n \qquad (4)$$

The MSB in the expression (4) is checked for being a 0, which would indicate an occurrence of w in S.

In the rest of this paper, we only use the variants of the Shift-OR algorithm because of its efficiency.

Approximate search

In a big data environment, errors in data occasionally happen and if exact search is applied on such data sets then false negatives will occur. To avoid that, certain error tolerance can be included in the search algorithms. The two most often used methods of performing error tolerant search are dynamic programming-based search and approximate bit-parallel search.

Dynamic programming in approximate search

It is well known (see for example [12], [13], [14]) that it is possible to determine the minimum number of elementary edit operations (insertions, deletions, and substitutions) that transform the given string $X = x_1x_2 \cdots x_n$ into the given string $Y = y_1y_2 \cdots y_m$ of symbols from an alphabet $\mathcal{A}$ in an iterative way, without enumerating all the possible transforms, by filling a special matrix W, which means that the time complexity of such an algorithm is essentially quadratic, i.e. $O(nm)$. This minimum number of elementary edit operations needed to transform X into Y is called Levenshtein or edit distance and the matrix W is called the matrix of partial edit distances. In this matrix, a cell $W[i,j]$ contains the (partial) edit distance between the prefix X_i of the string X and the prefix Y_j of the string Y. To compute the element $W[i,j]$ it is necessary to use the elements $W[i-1,j]$ (insertions), $W[i,j-1]$ (deletions), and $W[i-1,j-1]$ (substitutions), and check which of these elementary edit operations gives the minimum increase in partial edit distance. This operation determines the value of $W[i,j]$. To each elementary edit operation, a cost is assigned. In most cases, this cost is equal for all deletions and insertions (the value is usually 1). The substitutions by the same symbol usually contribute 0 to the edit distance, while the substitutions by a different symbol can be set differently for every pair of symbols (this is usual in computational biology) or equal for all such substitutions. Let d_e be the elementary edit distance assigned to a deletion of a symbol and let d_i be the elementary edit distance assigned to an insertion of a symbol. To represent deletions and substitutions, an "empty symbol" ϕ is often used. Thus, the cost assigned to the deletion of a symbol x is $d(x,\phi)$, and the cost assigned to the insertion of the symbol y is $d(\phi,y)$. Let $d_s(x,y)$ be the elementary edit distance assigned to a substitution of the symbol x by the symbol y. The dynamic programming algorithm for computing the edit distance $d(X,Y)$ between the strings $X = x_1x_2 \cdots x_n$ and $Y = y_1y_2 \cdots y_m$ is given below (Algorithm 1, [13])

Algorithm 1

$W[0,0] = 0$
for $i = 1, \dots, n, W[i,0] = i$
for $j = 1, \dots, m, W[0,j] = j$
for $i = 1, \dots, n$
for $j = 1, \dots, m$
$W[i,j] = min\{W[i,j-1] + d_e, W[i-1,j] + d_i, W[i-1,j-1] + d_s\}$
$d(X,Y) = W[n,m]$ □

To illustrate the operation of Algorithm 1, let us produce the edit distance matrix given the strings X ="bigram" and Y ="monogram", assuming the following:

$\forall x, d_e = d(x,\phi) = 1,$
$\forall y, d_i = d(\phi,y) = 1,$
$\forall (x,y), d_s = d(x,y) = \begin{cases} 0, x = y \\ 1, x \neq y \end{cases}, x, y \in \mathcal{A}.$

The matrix of partial edit distances produced by Algorithm 1 for this case is given below:

		m	o	n	o	g	r	a	m
	0	1	2	3	4	5	6	7	8
b	1	1	2	3	4	5	6	7	8
i	2	2	2	3	4	5	6	7	8
g	3	3	3	3	4	4	5	6	7
r	4	4	4	4	4	5	4	5	6
a	5	5	5	5	5	5	5	4	5
m	6	5	6	6	6	6	6	5	4

The representation of deletions and insertions by means of the "empty symbol" ϕ enables presentation of so-called edit sequences, which show the order of elementary edit operations in the transform of the (prefix of) string X into the (prefix of) string Y. The optimal transform, i.e. the transform, whose cost is minimal, is not unique. For example, the following two optimal transforms have the same overall cost ($d(X,Y) = 4$) under the assumptions on d_e, d_i, and d_s defined above.

$$C_1(X,Y) = \begin{bmatrix} b & i & \phi & \phi & g & r & a & m \\ m & o & n & o & g & r & a & m \end{bmatrix}$$
$$C_2(X,Y) = \begin{bmatrix} b & \phi & i & \phi & g & r & a & m \\ m & o & n & o & g & r & a & m \end{bmatrix}$$

Edit sequence reconstruction is necessary in many applications, such as computational biology, cryptanalysis etc. The algorithm to reconstruct an optimal edit sequence is based on backtracking through the whole partial edit distance matrix, starting from $W[n,m]$. The need for such a backtracking requires maintaining the whole partial edit distance matrix that yields space complexity of the dynamic programming edit distance computation algorithm $O(nm)$. If there is no need for edit sequence reconstruction, the space complexity is reduced to $O(n)$ since, obviously, to compute the edit distance only two columns of the partial edit distance matrix are necessary to maintain at every moment.

The allowed number of errors in approximate search can now be defined as the minimum edit distance k that is tolerated between the search pattern w and the distorted version of the portion of the search string S where the search pattern is located. The

approximate search of the search pattern w in the search string S with the error tolerance k can be performed by means of the same dynamic programming procedure that is used for the edit distance computation (Algorithm 1), but with a different initialization (see [1]). Namely, by setting all the elements of the 0-th row of the partial edit distance matrix W to 0, we allow the search pattern to commence at any position of the search string S. The insertions before that position and the insertions that (may) occur after the last symbol of the search pattern (except a single insertion that may appear immediately after this symbol) do not contribute to the overall cost. The dynamic programming approximate search algorithm with the same input as in the example above and with $k = 3$ gives the following partial edit distance matrix:

		m	o	n	o	g	r	a	m
	0	0	0	0	0	0	0	0	0
b	1	1	1	1	1	1	1	1	1
i	2	2	2	2	2	2	2	2	2
g	3	3	3	3	3	2	3	3	3
r	4	4	4	4	4	3	2	3	4
a	5	5	5	5	5	4	3	2	3
m	6	5	6	6	6	5	4	**3**	**2**

Every edit sequence that corresponds to an entry in the lowest row of the matrix W, whose total cost is $\leq k$ is acceptable. In our example (the figures in boldface in the last row of the matrix W), the acceptable values are 3 and 2, which means that the search pattern w is detected either at the position 7 or at the position 8 of the search string S. The corresponding edit sequences are given below:

$$C_1(X,Y) = \begin{bmatrix} \phi & \phi & \boldsymbol{b} & \boldsymbol{i} & \boldsymbol{g} & \boldsymbol{r} & \boldsymbol{a} & \boldsymbol{m} & \phi \\ m & o & \boldsymbol{n} & \boldsymbol{o} & \boldsymbol{g} & \boldsymbol{r} & \boldsymbol{a} & \boldsymbol{\phi} & m \end{bmatrix}, d = 3$$

$$C_2(X,Y) = \begin{bmatrix} \phi & \phi & \boldsymbol{b} & \boldsymbol{i} & \boldsymbol{g} & \boldsymbol{r} & \boldsymbol{a} & \boldsymbol{m} \\ m & o & \boldsymbol{n} & \boldsymbol{o} & \boldsymbol{g} & \boldsymbol{r} & \boldsymbol{a} & \boldsymbol{m} \end{bmatrix}, d = 2$$

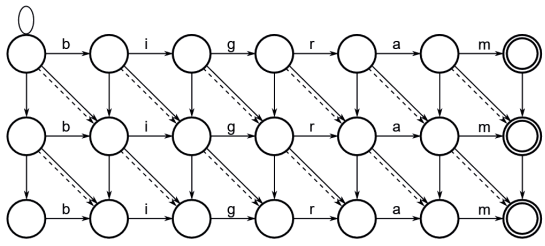
Only the symbols of the edit sequence given in boldface letters contribute to the overall cost of the edit sequence.

Bit-parallelism in approximate search

Attempts have been made to parallelize the dynamic programming-based approximate search algorithm (see, for example [15]), since if no edit sequence reconstruction is needed then it is possible to encode in binary the transition from one column of the partial edit distance matrix W into the other column that is maintained. If the reconstruction of the edit sequence is needed, then the whole matrix W is necessary to maintain, and the parallelization of these transitions becomes difficult.

Another approach to the parallelization of approximate search is through the extension of the bit-parallel exact search [16]. Suppose the search tolerance is k. Then we can simulate an NFA having $k + 1$ rows, each corresponding to the search status word D assigned to the pattern w. The transitions in this NFA can be horizontal (a match, which is a substitution by the same character treated separately), vertical (an insertion), and diagonal (deletions and substitutions by a different character). An example of such an NFA is presented in Fig. 5, where $w =$"bigram" and $k = 2$.

The NFA from Fig. 5 has 3 rows. The diagonal transitions that correspond to deletions are presented in the form of dashed lines – they are ϵ-transitions, since such transitions do not consume any input character. The zero state in the 1-st row of the NFA has a loop and is always active since the detection of the first character of the search pattern can occur at any position in the (distorted) search string.



F i g. 5. An NFA used in bit-parallel approximate search

Instead of the search status word D used i n e xact b it-parallel search, a search status array R is used, consisting of $k + 1$ rows. As the characters of the (distorted) search string S arrive, t he simulated NFA makes the transitions (if possible) from all the current active states from each row at the same time. The influence of the previous rows on the active states of the current row is taken into account in the search status array update formula by the superposition law. Equation (5) (converted to the Shift-OR form from [16]) is the search status array R update formula that determines which states are active after processing a symbol from S. The formula is the extension of the Shift-OR search status word update formula (4). If the final state of the 0th row of the NFA becomes active after processing of a symbol from S, then an occurrence of the search pattern without errors (exact match) is detected. If this happens in the 1st row, then an occurrence is found with 1 error and so on.

$$\begin{aligned} &R'_0 \leftarrow (R_0 \ll 1) \vee B[S_j] \\ &R'_i \leftarrow \Big((R_i \ll 1) \vee B[S_j]\Big) \wedge && \text{(match)} \\ &R_{i-1} \wedge && \text{(insertion)} \\ &(R_{i-1} \ll 1) \wedge && \text{(substitution)} \\ &(R'_{i-1} \ll 1) && \text{(deletion)} \\ &i = 1, \ldots, k \end{aligned} \qquad (5)$$

Constrained approximate search

The unconstrained approximate search algorithms (dynamic programming-based and bit-parallel) explained in the previous sections can be generally applied, regardless of the a priori knowledge about the search pattern/search string properties. In many applications, not all the possible transforms of the search pattern into a distorted version of the search string take place. If we possess some knowledge about the distortion process, then we can modify the search algorithm to take into account its subtleties. In such a way we exclude the transforms of the original search string into its distorted version that are impossible to happen. Consequently, the false positive rate of the search procedure is reduced.

A typical and very simple constraint that is possible to apply concerns the elimination of certain elementary edit operations. For example, if insertions never happen, we can simplify the search status array R update formula (5) used in the bit-parallel version of the unconstrained approximate search by eliminating the part related to insertions, which simplifies the formula and

saves the processing time. We can do the same with the dynamic programming-based search formula, by allowing only the diagonal and horizontal transitions in the matrix of partial edit distances. In the dynamic programming-based search case, this has another positive consequence. Namely, if we transform the coordinates in such a way that instead of the counters of symbols in the strings X and Y we use the numbers of elementary edit operations (i for insertions, e for deletions, and s for substitutions), the Algorithm 1 obtains a form having a very complicated initialization and the dynamic programming array becomes 3-dimensional [14]. However, if we know a priori that no insertions (or deletions) are used, then the dimension of the dynamic programming array remains 2 and the initialization of the algorithm remains relatively simple. This form of the dynamic programming-based search with transformed coordinates is used in cryptanalysis of stream ciphers [17].

Other, more complex, types of constraints can be introduced and the search status array update formula in the bit-parallel approximate search algorithm can be modified by introducing special counters and/or bit masks. In the dynamic programming-based algorithms, this is achieved by adding counters and additional loops in the Algorithm 1. The constraints that are introduced are determined by the application of the search algorithm and the a priori knowledge that is at the disposal of the search algorithm designer. In the sequel, we explain certain scenarios that determine specific sets of constraints in approximate search.

Applications in SPAM filtering and file carving

SPAM still represents a great deal of today's E-mail traffic. To eliminate SPAM without producing too many false positives and/or false negatives, various algorithms are used and many of them include search for typical SPAM words (see, for, example, [18]). To avoid elimination by SPAM filters, whose operation is based on exact search, the spammers often use algorithms that modify these words by substituting, inserting and/or deleting symbols. At the same time, the intelligibility of these words must be preserved in order to achieve the spammers' goals – the victim must be able to understand these words, even though they are modified. Consequently, some constraints must be defined to the numbers of inserted/deleted/substituted symbols and the distribution of the changes. Being aware of this fact, the defensive side can introduce the corresponding constraints in approximate search for SPAM words. The effect on reducing the number of false positives in search is better if the a priori information about the modification process parameters that the spammer uses is more accurate. In [3], such a scenario was studied, and a set of constraints was defined that limited the total number of so-called indels (insertions and deletions) in the edit transforms of the original SPAM words. By using these constraints, both the dynamic programming-based search algorithm and the bit-parallel approximate search algorithm were modified, and their performances were compared. It was shown experimentally that the bit-parallel version of the approximate search algorithm is more efficient than the dynamic programming-based one when the number of indels is greater than the number of substitutions.

In digital forensics, file carving procedures are used to try to reconstruct files, whose fragments are still present in permanent memory, but at the operating system level these files have been erased and therefore the metadata is missing. This is a natural field of application for approximate search algorithms and in some scenarios (for example, when the files have been erased by means of a tool with intention to reconstruct them at a later time) the introduction of constraints in search may help in improving the efficiency of the carving and reducing the false positive rate. The quality of a priori information about the deletion tool parameters again contributes to improving the efficiency and accuracy of the constrained approximate search algorithm used in the file carving procedure.

Applications in network forensic and intrusion detection

In attacks against computer networks and hosts, very often the new attack traffic is obtained by slightly modifying the known attack traffic. Since most Intrusion Detection Systems (IDS), which are the tools used to detect such attacks, employ exact search for known attack patterns, new attack patterns may pass unnoticed by these systems. A single bit of change of the known attack traffic is enough to make such a signature-based IDS to miss the attack. On the other hand, since the attacks exploit known small vulnerabilities of the computer networks and hosts, changing the attack pattern too much may make the newly produced traffic incapable of exploiting these vulnerabilities. Consequently, the changes that are performed on the known attack patterns are very often very small. In addition, since the traffic rate and the number of potential victims steadily grow, manual changes are very rarely used. Instead, special tools are used to modify such traffic and the parameters that determine their behavior may be known to the defensive side by threat intelligence (where information about this may be obtained through various channels). In [19], a bit-parallel constrained approximate search algorithm was described, in which the constraints limit the total numbers of elementary edit operations (insertions, deletions, and substitutions). The resulting algorithm CRBP-OpCount (Constrained Row-Based Bit-Parallel-Operations Count) produced up to 6 times fewer false positives under some scenarios (concerning the percentage of applied deletions, insertions, and/or substitutions) than the unconstrained approximate bit-parallel search, maintaining at the same time reasonable efficiency. With this constrained approximate search algorithm in place, a signature-based IDS can detect new attacks originating from the known ones with a reduced number of false positives compared to a system employing unconstrained approximate search.

Potential Applications in Bioinformatics

The methods hold a big potential in the field of bioinformatics, such a search of genome for a motif mining or a database with chemical compounds for the ones containing a fragment of interest. This is useful in the light of automatic search, where via the statistical analysis based on frequencies of occurrences in active and inactive chemicals, it is possible to discover relevant fragments [20], but the occurrence of their "distorted expressions" are not frequent enough to be discovered. Yet it is known that a

small distortion, say the substitutions of atoms within the column in the periodic table or the substitutions of within the functional group, does not change the activity of the compound. The expert knowledge is incorporated into the modeling of the distortions of the search pattern in the form of probabilities for the edit operations. The first tuning parameter in the inexact retriever is the number of permitted indels (insertions/deletions and substitutions) on the input, and the second one is the probabilities associated to the edit operations. The search should be performed over parsed strings [21-25], not raw ones. The mechanism permits to retrieve molecules that contain fragments similar to the exact fragments, which were known or discovered to be critical to the activity. They are obtained from an exact fragment via the substitution of the groups of atoms known to make the compound engage in certain reactions (functional groups), and via applying a number of insertions and deletions.

Conclusion

In this paper, we have given an overview of constrained approximate search algorithms. We first exposed the elementary concepts of exact and approximate unconstrained search. Then we explained the constraints that could be introduced in order to reduce the number of false positives in certain big data applications. We enumerated typical environments, in which it is possible to achieve better results if we introduce such constraints in search. Consequences on search efficiency have also been discussed. Some experimental results that indicated the circumstances under which it is worth using constrained approximate search have been discussed. If the adequate constraints are introduced, these results show that the false positive rate in knowledge discovery procedures can be significantly reduced.

References

[1] Navarro G., Raffinot M. Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences. Cambridge: Cambridge University Press; 2002. (In Eng.) DOI: https://doi.org/10.1017/CBO9781316135228

[2] Baeza-Yates R., Gonnet G.H. A new approach to text searching. *Communications of the ACM*. 1992; 35(10):74-82. (In Eng.) DOI: https://doi.org/10.1145/135239.135243

[3] Chitrakar A., Petrović S. Approximate search with constraints on indels with application in SPAM filtering. In: Proceedings of Norwegian Information Security Conference (NISK-2015), Ålesund, Norway; 2015. p. 22-33. Available at: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2380987 (accessed 21.01.2020). (In Eng.)

[4] Knuth D.E., Morris Jr. J.H., Pratt V.R. Fast Pattern Matching in Strings. *SIAM Journal on Computing*. 1977; 6(2): 323-350. (In Eng.) DOI: https://doi.org/10.1137/0206024

[5] Boyer R.S., Moore J.S. A fast string searching algorithm. *Communications of the ACM*. 1977; 20(10):762-772. (In Eng.) DOI: https://doi.org/10.1145/359842.359859

[6] Horspool R. Practical Fast Searching in Strings. *Software: Practice and Experience*. 1980; 10(6):501-506. (In Eng.) DOI: https://doi.org/10.1002/spe.4380100608

[7] Sunday D.M. A very fast substring search algorithm. *Communications of the ACM*. 1990; 33(8):132-142. (In Eng.) DOI: https://doi.org/10.1145/79173.79184

[8] Crochemore M., Rytter W. Text algorithms. Oxford University Press, Inc., USA; 1994. (In Eng.)

[9] Navarro G., Raffinot M. Fast and flexible string matching by combining bit-parallelism and suffix automata. *ACM Journal of Experimental Algorithmics*. 2000; 5:4-es. (In Eng.) DOI: https://doi.org/10.1145/351827.384246

[10] Zhang Y., Liu P., Liu Y., Li A., Du C., Fan D. Attacking Pattern Matching Algorithms Based on the Gap between Average-Case and Worst-Case Complexity. *Journal on Advances in Computer Network*. 2013; 1(3):228-233. (In Eng.) DOI: https://doi.org/10.7763/JACN.2013.V1.45

[11] Sipser M. Introduction to the Theory of Computation, 2nd ed. Thomson, USA; 2006. (In Eng.)

[12] Levenshtein V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady.* 1966; 10(8):707-710. (In Eng.)

[13] Wagner R.A., Fischer M.J. The String-to-String Correction Problem. *Journal of the ACM*. 1974; 21(1):168-173. (In Eng.) DOI: https://doi.org/10.1145/321796.321811

[14] Oommen B.J. Constrained String Editing. *Information Sciences*. 1986; 40(3):267-284. (In Eng.) DOI: https://doi.org/10.1016/0020-0255(86)90061-7

[15] Hyyrö H., Navarro G. Faster Bit-Parallel Approximate String Matching. In: A. Apostolico, M. Takeda (ed.) Combinatorial Pattern Matching. CPM 2002. Lecture Notes in Computer Science, vol. 2373. Springer, Berlin, Heidelberg; 2002. p. 203-224. (In Eng.) DOI: https://doi.org/10.1007/3-540-45452-7_18

[16] Wu S., Manber U. Fast text searching: allowing errors. *Communications of the ACM*. 1992; 35(10):83-91. (In Eng.) DOI: https://doi.org/10.1145/135239.135244

[17] Golić J.Dj., Mihaljević M.J. A generalized correlation attack on a class of stream ciphers based on the Levenshtein distance. *Journal of Cryptology*. 1991; 3(3):201-212. (In Eng.) DOI: https://doi.org/10.1007/BF00196912

[18] Schryen G. Anti-SPAM Measures: Analysis and Design. Springer, Berlin, Heidelberg; 2007. (In Eng.) DOI: https://doi.org/10.1007/978-3-540-71750-8

[19] Chitrakar A., Petrović S. Constrained Row-Based Bit-Parallel Search in Intrusion Detection. In: Proceedings of Norwegian Information Security Conference (NISK-2016), Bergen, Norway; 2016. p. 68-79. Available at: https://ojs.bibsys.no/index.php/NISK/article/view/375 (accessed 21.01.2020). (In Eng.)

[20] Sagar S., Sidorova J. Sequence Retriever for Known, Discovered, and User-Specified Molecular Fragments. In: M.S. Mohamad, M.P. Rocha, F. Fdez-Riverola, F.J. Domínguez Mayo, J.F. De Paz (ed.) 10th International Conference on Practical Applications of Computational Biology & Bioinformatics. PACBB 2016. Advances in Intelligent Systems and Computing, vol. 477. Springer, Cham; 2016. p. 51-58. (In Eng.) DOI: https://doi.org/10.1007/978-3-319-40126-3_6

[21] Sidorova J., Fernandez A., Cester J., Rallo R., Giralt F. Predicting Biodegradable Quality of Chemicals with the TGI+.3 Classifier. In: Proceeding (717) Artificial Intelligence and Applications / 718: Modelling, Identification, and Control - 2011. Innsbruck, Austria; 2011. p. 108-115. (In Eng.) DOI:

https://doi.org/10.2316/P.2011.717-044

[22] Sidorova J., Anisimova M. NLP-inspired structural pattern recognition in chemical application. *Pattern Recognition Letters.* 2014; 45:11-16. (In Eng.) DOI: https://doi.org/10.1016/j.patrec.2014.02.012

[23] Sidorova, J., Garcia, J. (2015). Bridging from syntactic to statistical methods: Classification with automatically segmented features from sequences. *Pattern Recognition*. 2015; 48:3749-3756. (In Eng.) DOI: https://doi.org/10.1016/j.patcog.2015.05.001

[24] Dietterich T.G., Bakiri G. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research.* 1995; 2:263-286. (In Eng.) DOI: https://doi.org/10.1613/jair.105

[25] Anand R., Mehrotra K., Mohan C.K., Ranka S. Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*. 1995; 6(1):117-124. (In Eng.) DOI: https://doi.org/10.1109/72.363444

Submitted 21.01.2019; revised 10.03.2020; published online 25.05.2020.

Поступила 21.01.2020; принята к публикации 10.03.2020; опубликована онлайн 25.05.2020.

About the authors:

Slobodan Petrović, Professor of the Department of Information Security and Communication Technology, Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology (22 Teknologiveien, Gjøvik 2815, Norway), Ph.D. (Engineering), ORCID: http://orcid.org/0000-0002-4435-2716, slobodan.petrovic@ntnu.no

Julia Sidorova, Associate Professor of the Department of Computer Science, Blekinge Institute of Technology (SE-37179 Karlskrona, Sweden), Ph.D. (Engineering), ORCID: http://orcid.org/0000-0002-1024-168X, julia.a.sidorova@gmail.com

All authors have read and approved the final manuscript.

Об авторах:

Петрович Слободан, профессор кафедры информационной безопасности и коммуникационных технологий, факультет информационных технологий и электротехники, Норвежский университет естественных и технических наук (2815, Норвегия, г. Йёвик, Технологический пр., д. 22), кандидат технических наук, ORCID: http://orcid.org/0000-0002-4435-2716, slobodan.petrovic@ntnu.no

Сидорова Юлия, доцент кафедры информатики, Технологический институт Блекинге (SE-37179, Швеция, Округ Блекинге, г. Карлскруна), кандидат технических наук, ORCID: http://orcid.org/0000-0002-1024-168X, julia.a.sidorova@gmail.com

Все авторы прочитали и одобрили окончательный вариант рукописи.