16th World Congress of the International Fuzzy Systems Association (IFSA)
9th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT)

Fast String Searching Mechanism

Petr Hurtik, Petra Hodakova, Irina Perfilieva

University of Ostrava, Centre of Excellence IT4Innovations,
Institute for Research and Applications of Fuzzy Modeling,
30. dubna 22, 701 03 Ostrava 1, Czech Republic

Abstract

The goal of this study is to introduce a novel exact string searching (i.e., matching) method based on the fuzzy transform, or F-transform (FTSS). The theoretical background of the F-transform, specifically the F^s-transform, $s \geq 0$, for functions of one variable is reviewed, and a string searching algorithm based on this theory is presented. The algorithm is demonstrated with several examples. The results are compared with those of three existing string searching algorithms: Knuth-Morris-Pratt (KMP), Boyer-Moore (BM) and Rabin-Karp (RK).

Keywords: Big data, Pattern searching, String searching, F-transform, F^s-transform, Generalized fuzzy partition

1. Introduction

As the computational power of computers (e.g., personal and servers) increases, the amount of stored data similarly increases. On the one hand, greater computational power allows higher processing speeds. On the other hand, higher computational speeds allow the processing of greater amounts of data. In fact, the volume of data has become so large that processing it has become problematic, and it has been named "big data". The term "big data" is not defined by a crisp value of size (see e.g., [3]). The size changes with time and depends on many factors such as the cost of storage, computational power, and the read/write speed. Moreover, these factors change rapidly over time. For example, Figures 1 and 2 illustrate the growth in the single hard-disk storage capacity and the computational power of computers in recent years.

There exist many different types of "big data" and different reasons to process them. In general, the common feature in processing big data is searching through the data. Therefore, in this study, we focus on search methods for big data sets. More specifically, we focus on the task of string searching.

Pattern searching can be separated into three types of approaches. The first type is finding an exact string match. For example, in the case of text strings, if we search for the word pattern "salami" in the given database, we obtain a positive answer if and only if the database contains the exact word "salami". The second type is finding a partial match and allows for minor typographical errors. For example, the word pattern "salami" can be matched with similar words such as "saLami", "slami", "alami". The third type is finding a semantic match, that is, all words with the same meaning as the pattern word. For example, the word pattern "salami" can be matched with words such as "ham" or "mortadela". These different types of pattern searching approaches can also be combined.

In this study, we focus on the first type of approach to pattern searching, the exact string match, and we develop a fast string searching method based on the fuzzy transform, or F-transform [8]. The main advantage of using the F-transform is that this technique gives us a simplified representation of the original data with a significantly shorter length, which simplifies the processing task. For reference, we also implement three well-known string searching algorithms: Knuth-Morris-Pratt (KMP) [5]; Boyer-Moore (BM) [6] and Rabin-Karp (RK) [7]. These algorithms were published in 1977 (KMP and BM) and 1987 (RK). We evaluate whether these algorithms remain applicable to big data in the 21^{st} century. We present several examples with different parameters, such as the length of the searched pattern, and we compare the results obtained using the four algorithms.

The paper is organized as follows: Section 2 formulates the problem of string searching and reviews the three common searching algorithms. In Section 3, we introduce a string searching method based on the F^s-transform, $s \geq 0$ and review several basic concepts of the F^s-transform. The examples and comparisons of the resulting search times are presented in Section 4. Conclusions and comments are provided in Section 5.

2. Exact string searching problem

As mentioned previously, searching for an exact string match is one type of pattern search. We are given a database that contains one type of data, for example, text, music, time series, or images. The data are in the form of a set of strings of numbers (symbols). We are also given a pattern, i.e., a short string of symbols. The task is to search for the pattern in the database in its exact form. A detailed description is given in the following.

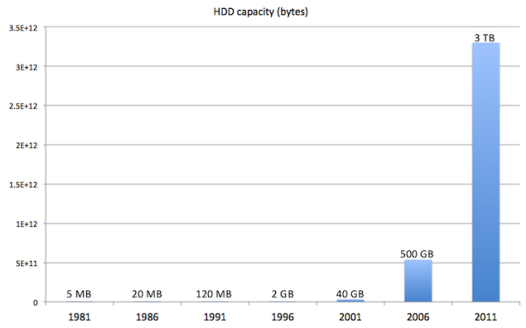

Figure 1: The growth in the single hard-disk storage capacity in recent years. Image was taken from [10].

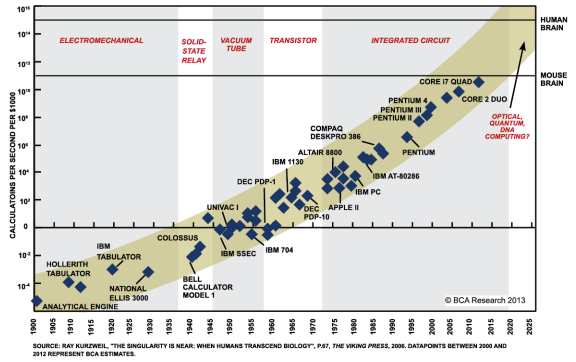

Figure 2: The computational power of computers in recent years. Image was taken from [4].

2.1. Problem description

Formally, we define o as a string of symbols represented by $o : T \to \Sigma$, where $T = \{1, \ldots, t_{max}\}$ is a set of positions of symbols and Σ is a predefined alphabet of symbols. More specifically, we define two types of strings o:

1. long string *database* $\mathbf{o^D} : \mathbf{T^D} \to \mathbf{\Sigma}$ where $T^D = \{1, ..., t^D_{max}\}$,
2. short string *pattern* $\mathbf{o^P} : \mathbf{T^P} \to \mathbf{\Sigma}$ where $T^P = \{1, ..., t^P_{max}\}$,

such that $t^P_{max} \leq t^D_{max}$.

If there is a match, we say that an occurrence of o^P in o^D is true at a position $\mathbf{s} \in \mathbf{T^D}$ if $(s + t^P_{max} - 1) \leq t^D_{max}$, and the following holds:

$$\forall j \in T^P : o^D_{s+j-1} = o^P_j \,,$$

where o^D_{s+j-1} and o^P_j denote the symbols at the $(s + j - 1)$-th and j-th positions in o^D and o^P, respectively. Otherwise, we say that an occurrence of o^P in o^D is *false*.

The problem is to find all occurrences of the pattern o^P in the database o^D. In the following, we will refer to this problem as *string searching*.

In the *naive approach*, a sliding comparison is performed between the pattern o^P and the database o^D. For this comparison, the following measure of closeness is used:

$$Cl(o^D, o^P) = \sum_{j=0}^{|T^P|} |o(t + j) - o^P(j)|, \; t \in T^D. \quad (1)$$

The complexity of the naive approach is then $O(t^D_{max} \cdot t^P_{max} - t^P_{max})$. This method is very computationally complex and therefore time consuming. In general, the naive approach (also called the "brute force" approach) represents the worst case (the slowest solution) from the perspective of the time complexity of string searching algorithms. One study[2] showed that searching by the naive approach can be very time consuming. For example, searching for a sound pattern with $t^P_{max} = 6.4$ s in a database of sound recordings with $t^D_{max} = 13200$ s required approximately 11 hours. Therefore, we seek a faster method for data processing problems such as this.

In the following section, we review three standard string searching algorithms.

2.2. Reference algorithms

In this section, we briefly describe the basic principles of three commonly used string searching algorithms. These algorithms were developed at roughly the same time as the first personal computers, when computers were not in common use. Computers at that time were intended for commercial/professional use and lacked "user-friendly" interfaces. Currently, the amount of data is increasing rapidly, and thus, the problem of big data is attracting increasing attention.

All of the algorithms presented in the following consist of two parts: preprocessing and searching. We note that the preprocessing step depends on the pattern form, i.e., any change in the pattern influences the preprocessing step and its computation time. Therefore, preprocessing must be performed separately for each pattern.

2.2.1. Knuth-Morris-Pratt (KMP) algorithm

This algorithm was developed independently by Knuth, Morris and Pratt and published jointly in 1977; see [5]. The principle of the algorithm is that each symbol in a database has to be inspected only once. A shift table is created in the preprocessing step. The preprocessing time is t^P_{max}. Each value in the shift table characterizes the length of the maximal proper prefix of that part of the pattern that is the suffix. For example, if the pattern AAABC is searched for in the database AAACBAFVAC..., after first matching AAA, the search continues with C, not B. For this position, the value in the shift table is 2, and the comparison with the database continues at AFVAC.... The time required for constructing the shift table is $O(t^P_{max})$. The complexity of the main string search is $O(t^D_{max})$.

2.2.2. Boyer-Moore (BM) algorithm

Similar to the KMP algorithm, the BM algorithm (cf. [6]) preprocesses the data with a shift table known as the good suffix shift. The preprocessing step includes the creation of a bad character shift table. Unlike the KMP and RK algorithms, the BM algorithm processes from right to left. The time complexity of this algorithm is $O(t^D_{max}+t^P_{max})$, but the worst-case complexity is equal to that of the naive approach. Usually, the complexity is sublinear, and the original study [6] showed that the BM algorithm works faster for lengthier alphabets.

2.2.3. Rabin-Karp (RK) algorithm

The third algorithm [7] was published in 1987. The main principle of this algorithm is to use hash values. Before computing the hash values, the symbols are converted to numbers using ASCII, UTF-8, or a similar character set. The preprocessing step consists of computing the hash values of the pattern, which has complexity $O(t^P_{max})$. The complexity of the main string searching step is $O(t^D_{max})$.

3. Proposed approach - string searching based on the F-transform

The main principle of our approach is to apply the F-transform to the string pattern o^P and to the data string o^D. In doing so, we obtain simplified representations with much shorter lengths. Then, rather than comparing o^P with o^D, we compare only the F-transform components of o^P and o^D. Specifically, rather than comparing $O(t^D_{max}\cdot t^P_{max}-t^P_{max})$ combinations, we compare only $O(\frac{t^D_{max}}{h}.\frac{t^P_{max}}{h}-\frac{t^P_{max}}{h})$ combinations. Therefore, this approach significantly reduces the computational complexity. We would like to emphasize that our approach can be applied to many types of data such as time series, text, DNA sequences, or SETI project data.

In the following sections, we review the formal definition of the F^s-transform, $s \geq 0$, and then, we describe the proposed algorithm based on this theory in more detail.

3.1. F^s-transform, $s \geq 0$, for functions of one variable

In this section, we assume that the reader is familiar with the main concept of the ordinary F-transform [8]. The F-transform with constant components can be extended to the F-transform of a higher degree - the F^s-transform, $s \geq 0$ - with s-degree polynomial components [9]. With respect to this extension, the original F-transform can be denoted as the F^0-transform.

Generally, the F-transform depends on a chosen fuzzy partition. In the following, we review the definition of a generalized fuzzy partition [1]. Then, we introduce a particular Hilbert space as a background for the definition of the F^s-transform, $s \geq 0$.

3.1.1. Generalized Fuzzy Partition

Definition 1 *Let $x_0, x_1, \ldots, x_n, x_{n+1} \in [a,b]$ be fixed nodes such that $a = x_0 \leq x_1 < \ldots < x_n \leq x_{n+1} = b$, $n \geq 2$. The fuzzy sets $A_1, \ldots, A_n : [a,b] \to [0,1]$ constitute a generalized fuzzy partition of $[a,b]$ if for every $k = 1, \ldots, n$, there exist $h_k^{'}, h_k^{''} \geq 0$ such that $h_k^{'}+h_k^{''} > 0$, $[x_k-h_k^{'}, x_k+h_k^{''}] \subseteq [a,b]$ and the following conditions are satisfied:*

1. *(locality) – $A_k(x) > 0$ if $x \in (x_k - h_k^{'}, x_k + h_k^{''})$ and $A_k(x) = 0$ if $x \in [a,b] \setminus (x_k - h_k^{'}, x_k + h_k^{''})$;*
2. *(continuity) – A_k is continuous on $[x_k - h_k^{'}, x_k + h_k^{''}]$;*
3. *(covering) – for $x \in [a,b]$, $\sum_{k=1}^{n} A_k(x) > 0$.*

By the *locality* and *continuity* conditions, it follows that

$$\int_a^b A_k(x)dx > 0.$$

If the nodes $x_0 = x_1, x_2, \ldots, x_{n-1}, x_n = x_{n+1}$ are h-equidistant, i.e., for all $k = 2, \ldots, n$, $x_k = x_{k-1} + h$, where $h = (b-a)/(n-1)$, $h^{'} > h/2$, and if two additional properties

4. $h_1^{'} = h_n^{''} = 0$, $h_1^{''} = h_2^{'} = \cdots = h_{n-1}^{''} = h_n^{'} = h^{'}$, and $A_k(x_k - x) = A_k(x_k + x)$ for all $x \in [0, h^{'}]$, $k = 2, \ldots, n-1$;
5. $A_k(x) = A_{k-1}(x-h)$ and $A_{k+1}(x) = A_k(x-h)$ for all $x \in [x_k, x_{k+1}]$, $k = 2, \ldots, n-1$;

are satisfied, then the fuzzy partition is called a $(h, h^{'})$-*uniform generalized fuzzy partition.* Moreover, if $h = h'$, then we say that a fuzzy partition is h-uniform.

3.1.2. Space $L_2(A_k)$, subspace $L_2^s(A_k)$

Let interval $[a,b]$ be a universe and $\{A_k \mid k = 1, \ldots, n\}$ be an $(h, h^{'})$-uniform generalized fuzzy partition of $[a,b]$. Throughout this section, we fix one A_k from the chosen fuzzy partition.

Let $L_2(A_k)$ be a Hilbert space of square-integrable functions $f : [x_{k-1}, x_{k+1}] \to \mathbb{R}$ with the inner product $\langle f, g \rangle_k$ given by

$$\langle f, g \rangle_k = \int_{x_{k-1}}^{x_{k+1}} f(x)g(x)A_k(x)dx. \qquad (2)$$

Remark 1 *The functions $f, g \in L_2(A_k)$ are* orthogonal *in $L_2(A_k)$ if*

$$\langle f, g \rangle_k = 0. \qquad (3)$$

In the sequel, we denote by $L_2([a,b])$ a set of functions $f : [a,b] \to \mathbb{R}$ such that for all $k = 1, \ldots, n$, $f|_{[x_{k-1}, x_{k+1}]} \in L_2(A_k)$, where $f|_{[x_{k-1}, x_{k+1}]}$ is the restriction of f on $[x_{k-1}, x_{k+1}]$.

Moreover, let space $L_2^s(A_k)$, $s \geq 0$, be a closed linear subspace of $L_2(A_k)$ with an orthogonal basis given by the polynomials

$$\{S_k^i(x)\}_{i=0,\ldots,s},$$

where i denotes the degree of the polynomial and orthogonality is defined in terms of (3). For example, $L_2^1(A_k)$ is a linear subspace of $L_2(A_k)$ with an orthogonal basis given by the polynomials:

$$S_k^0(x) = 1,\ S_k^1(x) = x - x_k.$$

The following lemma characterizes the orthogonal projection of a function $f \in L_2([a,b])$ or the best approximation of f in the space $L_2^s(A_k)$.

Lemma 1 *Assume $f \in L_2([a,b])$ and let $L_2^s(A_k)$ be a closed linear subspace of $L_2(A_k)$, as specified previously. Then, the orthogonal projection F_k^s of $f|_{[x_{k-1},x_{k+1}]}$ on $L_2^s(A_k)$, $s \geq 0$, is*

$$F_k^s = \sum_{i=1}^{s} c_k^i S_k^i \tag{4}$$

where

$$c_k^i = \frac{\langle f, S_k^i\rangle_k}{\langle S_k^i, S_k^i\rangle_k} = \frac{\int_{x_{k-1}}^{x_{k+1}} f(x)S_k^i(x)A_k(x)dx}{\int_{x_{k-1}}^{x_{k+1}} (S_k^i(x))^2 A_k(x)dx}. \tag{5}$$

The proof can be found in [9].

3.2. Direct F^s-transform, $s \geq 0$

Let $[a,b]$ be the universe and let $\{A_k \mid k = 1,\ldots,n\}$ be the (h,h')-uniform generalized fuzzy partition of $[a,b]$. Moreover, assume $f \in L_2([a,b])$ and let $L_2^s(A_k)$, $s \geq 0$, $k = 1,\ldots,n$, be a space with a basis given by

$$\{S_k^i(x)\}_{i=1,\ldots,s}.$$

Next, we define the direct F^s-transform of the given function f.

Definition 2 *Assume $f \in L_2([a,b])$, and let F_k^s, $s \geq 0$ be the orthogonal projection of $f|_{[x_{k-1},x_{k+1}]}$ on $L_2^s(A_k)$, $k = 1,\ldots,n$ given by (4). We say that the n-dimensional vector $\mathbf{F_n^s}[f] = (F_1^s,\ldots,F_n^s)$ is the direct F^s-transform of f with respect to $\{A_k \mid k = 1,\ldots,n\}$, where F_k^s, $k = 1,\ldots,n$ are called the F^s-transform components.*

3.2.1. Direct F^1-transform

In this section, we briefly discuss the (direct) F^1-transform of functions from $L_2([a,b])$. The F^1-transform will be used subsequently in this study.

Let $L_2^1(A_k) \subseteq L_2(A_k)$ be a linear span of the set consisting of two orthogonal polynomials

$$S_k^0(x) = 1,\ S_k^1(x) = x - x_k, \tag{6}$$

where A_k from the chosen fuzzy partition is assumed to be symmetric with respect to x_k, $k = 1,\ldots,n$.

Analogous to the general F^s-transform, $s \geq 0$, in the following we introduce the F^1-transform, which has components in the form of linear polynomials.

Definition 3 *Assume $f \in L_2([a,b])$ and let F_k^1 be the orthogonal projection of $f|_{[x_{k-1},x_{k+1}]}$ on subspace $L_2^1(A_k)$, $k = 1,\ldots,n$, with a basis given by (6).*

The n-dimensional vector $\mathbf{F_n^1}[f] = (F_k^1)$, $k = 1,\ldots,n$ is the F^1-transform of f with respect to $\{A_k \mid k = 1,\ldots,n\}$, and F_k^1 is the corresponding F^1-transform component represented by

$$F_k^1(x) = c_k^0 + c_k^1(x - x_k), \tag{7}$$

where the coefficients c_k^0, c_k^1 are given by (5).

The following theorem provides approximations of the function f and its first derivative f' using the coefficients of the F^1-transform and estimates of the quality of the approximations.

Theorem 1 *Assume $f \in L_2([a,b])$, and let $\{A_k \mid k = 1,\ldots,n\}$, $n \geq 2$, be an (h,h')-uniform generalized fuzzy partition of $[a,b]$. Let $\mathbf{F_n^1}[f] = (F_1^1,\ldots,F_n^1)$, where $F_k^1 = c_k^0 + c_k^1(x - x_k)$, $k = 1,\ldots,n$ is the F^1-transform of f with respect to the given partition.*

- *Let functions f, A_k, $k = 1,\ldots,n$ be twice continuously differentiable on $[a,b]$. Then, for every k:*
$$c_k^0 = f(x_k) + O(h'^2).$$
- *Let functions f, A_k, $k = 1,\ldots,n$, be four times continuously differentiable on $[a,b]$. Then, for every k:*
$$c_k^1 = f'(x_k) + O(h'^2).$$

The proof is analogous to that for the h-uniform fuzzy partition introduced in [9].

3.2.2. Discrete case of F^1-transform

Let us consider the discrete case, where the value of the original function is known only at discrete points. The data used in the examples in this study are represented by discrete elements.

The discrete (direct) F^1-transform is defined as follows.

Definition 4 *Let a function $f : [a,b] \to \mathbb{R}$ be defined at discrete points $P = \{(p_i) \mid i = 1,\ldots,N\}$. Let $\{A_k \mid k = 1,\ldots,n\}$ be a fuzzy partition of $[a,b]$ where $x_0, x_1, \ldots, x_{n+1} \in [a,b]$ are fixed nodes. Suppose that the set P is sufficiently dense with respect to the chosen partition, i.e.,*

$$\forall k\ \exists i;\ A_k(p_i) > 0.$$

We say that the n-dimensional vector $\mathbf{F_n^1}[f] = (F_1^1,\ldots,F_n^1)$ is the discrete F^1-transform of f with respect to the chosen partition if for all $k = 1,\ldots,n$ and $p_i \in P$

$$F_k^1(p_i) = c_k^0 + c_k^1(p_i - x_k), \tag{8}$$

where the coefficients c_k^0, c_k^1 are given as follows

$$c_k^0 = \frac{\sum_{i=1}^{N} f(p_i)A_k(p_i)}{\sum_{i=1}^{N} A_k(p_i)},$$

$$c_k^1 = \frac{\sum_{i=1}^{N} f(p_i)(p_i - x_k)A_k(p_i)}{\sum_{i=1}^{N} (p_i - x_k)^2 A_k(p_i)}.$$

3.3. Proposed algorithm based on the F^1-transform (FTSS)

The principle of our approach is as follows. We apply the F^1-transform to the strings o^D and o^P. Then, we compare the components of the F^1-transforms $\mathbf{F^1_n}[o^D]$ and $\mathbf{F^1_m}[o^P]$ by computing the closeness of the coefficients c^1_{k,o^D} and c^1_{k,o^P} (see (8)); i.e., we compute $Cl(c^1_{k,o^D}, c^1_{k,o^P})$ (given by (1)). We use the F^1-transform because it is sensitive to local changes in the analyzed string (unlike the F^0 transform, which produces only average values and thus does not follow the linear structure of a string).

The measure $Cl(c^1_{k,o^D}, c^1_{k,o^P})$ is used to detect potential occurrences of the pattern. By this we mean that if $Cl(c^1_{k,o^D}, c^1_{k,o^P})$ is less than the value of the predefined threshold θ, then the corresponding substring of o^D is close to the pattern o^P and is therefore a candidate for a match.

The algorithm iterates with respect to the parameter h from the chosen $(h, 2h)$-uniform generalized fuzzy partition (h denotes the distance between nodes from the partition). More specifically, we use a large value of h (e.g., $h = 10000$) in the first step of the algorithm, and we find and mark the "suspicious" sub-strings of o^D as described above. In the second step, we focus only on the chosen sub-strings from the previous step and create a new partition with the smaller value of h. Then, we repeat the procedure with decreasing values of the parameter h. This approach enables us to process less data, and therefore we can achieve a faster search by making the algorithm sequentially more accurate.

The algorithm consists of the following steps:

Input: Strings o^D and o^P, h, θ.

S 1: Compute $\mathbf{F^1_n}[o^D] = (F^1_1, \ldots, F^1_n)_{o^D}$ w.r.t. the $(h, 2h)$-uniform generalized fuzzy partition.

S 2: Compute $\mathbf{F^1_m}[o^P] = (F^1_1, \ldots, F^1_m)_{o^P}$ w.r.t. the same fuzzy partition.

S 3: Compute the closeness $Cl(c^1_{k,o^D}, c^1_{k,o^P})$.

S 4: Find and mark sub-strings of o^D for which $Cl(c^1_{k,o^D}, c^1_{k,o^P})$ less than the chosen threshold θ.
If no sub-strings are found, go to *Output a).*

S 5: Replace o^D by a union of the marked sub-strings (from the previous step); reduce the value of the parameter h, e.g., $h = h/10$.

S 6: Repeat steps *S 1* - *S 5* until $h = 1$.

S 7: Choose the sub-strings of o^D with $Cl(c^1_{k,o^D}, c^1_{k,o^P}) = 0$.
If no sub-strings are found, go to *Output a).*

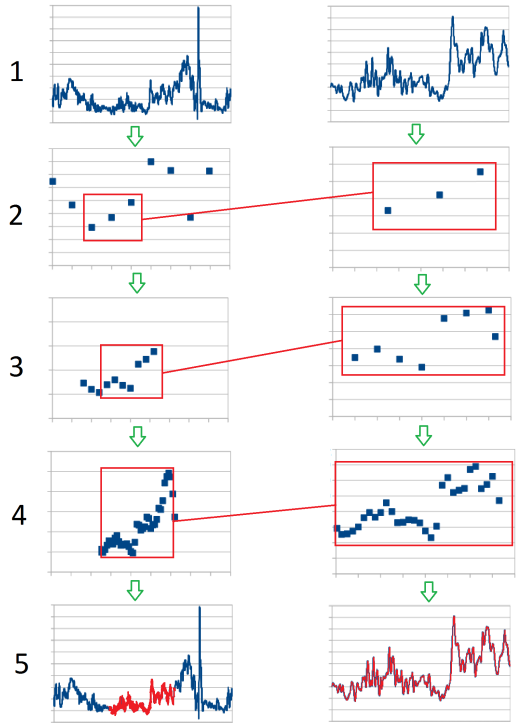

Figure 3: An example of recognition of pattern o^P (on the right side) in signal o^D (on the left side). **1.** signals representations, **2.-4.** iterative procedure with decreasing values of h (and increasing number of the corresponding components), **5.** the found and highlighted occurrence of o^P in o^D.

Output:
a) The pattern has no occurrences in the database.
b) The first coordinate in o^D that corresponds to an occurrence.

In step *S 4)*, the threshold θ can be given by user arbitrarily, or it can be computed as:

$$\theta = m \cdot \max_{i=1,\ldots,m-1} |c^1_{i,o^P} - c^1_{i+1,o^P}|,$$

where m denotes the number of components of $\mathbf{F^1_m}[o^P]$.

An example of the iterative procedure for searching for the pattern o^P in the signal o^D with decreasing values of h is illustrated in Figure 3.

In Figure 4, we show the steps of the algorithm in the form of a flow chart. This diagram indicates the dependencies between the partial steps of the algorithm.

4. Experiments and results

The four search algorithms - RK, BM, KMP, and FTSS - were implemented on a notebook computer: an ASUS with an i7 4500U CPU and a clock speed of 1.8 GHz. These algorithms were implemented in C++ with the QT framework.

For our experiments, we chose two alphabets with different lengths. The first one, $|\Sigma| = 5$, simu-

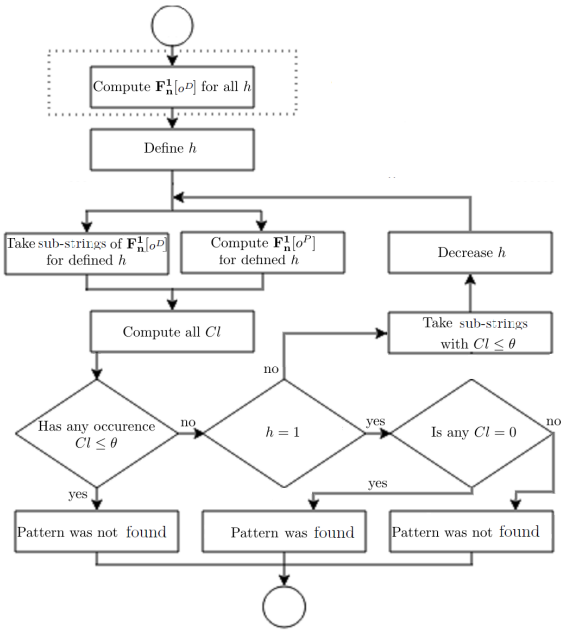

Figure 4: Demonstration of partial steps of the algorithm in the form of a flow chart.

lates special computer applications such as DNA sequences. The second one, $|\Sigma| = 30$, simulates standard text. For both alphabets, we randomly created three types of databases, $t^D_{max} \in \{1 \cdot 10^6, 1 \cdot 10^7, 1 \cdot 10^8\}$, and two types of patterns, $t^P_{max} \in \{1 \cdot 10^4, 1 \cdot 10^5\}$, based on different lengths. For each trial of the algorithms, we randomly created one type of database and one type of pattern as characterized previously. We tested two cases, one in which the pattern occurs in the database and one in which it does not.

The processing times for each of the algorithms are given in the following Tables 1 - 4. The processing times are the average values of ten trials of the algorithm. The execution time is measured in ms.

For the three standard algorithms - KMP, BM, and RK - the preprocessing time is included in the processing time because the preprocessing part is dependent on the chosen pattern. In the case of the FTSS algorithm, we distinguish the preprocessing time (FSSS $\mathbf{F^1_m}[o^P]$, the computation of the F^1-transform components of the pattern o^P), the processing time (FTSS closeness, the computation of $Cl(c^1_{k,o^D}, c^1_{k,o^P})$) and their summation (FTSS total). The computation of $\mathbf{F^1_n}[o^D] = (F^1_1, \ldots, F^1_n)_{o^D}$ is not included because this step is executed only one time and the result can be used for searching for all of the patterns.

All of the tests were successful in that all actual occurrences of the pattern were found and marked in the correct place in the database, and no false occurrences of the pattern were found or marked.

Tables 1 and 2 show the results for the databases created from the alphabet with length $|\Sigma| = 30$. We can observe that the fastest results were obtained by the BM algorithm. However, in the case of $t^D_{max} = 1 \cdot 10^7$ and $t^D_{max} = 1 \cdot 10^8$, the FTSS algorithm was significantly faster than the KMP and RK algorithms and was in several cases comparable with the BM algorithm.

Tables 3 and 4 show the results for the databases created from the alphabet with length $|\Sigma| = 5$. In this case, we can observe that the BM algorithm was slower than in the previous case where $|\Sigma| = 30$, whereas the FTSS algorithm had processing times similar to those in the previous case. Moreover, the results for the FTSS algorithm were comparable to those of the BM algorithm and in some cases better.

We can conclude that the BM algorithm is very fast for "long" (i.e., many symbols) alphabets, whereas for "short" (i.e., few symbols) alphabets, the results are comparable to those of the other algorithms. The performance of the proposed FTSS algorithm is less sensitive to the length of the alphabet; i.e., it is very fast and the length of alphabet does not affect the processing speed.

Table 1: Processing time for the pattern $t^P_{max} = 3 \cdot 10^4$ and the alphabet $|\Sigma| = 30$.

Algorithm	$t^D_{max} = 1 \cdot 10^6$	$t^D_{max} = 1 \cdot 10^7$	$t^D_{max} = 1 \cdot 10^8$
Pattern is in db			
KMP	**1**	44	442
RK	**1**	40	441
BM	**1**	**1**	**31**
FTSS $\mathbf{F^1_m}[o^P]$	1	1	1
FTSS closeness	1	1	30
FTSS total	2	2	**31**
Pattern is not in db			
KMP	6	42	442
RK	3	32	328
BM	**1**	**1**	**32**
FTSS $\mathbf{F^1_m}[o^P]$	3	1	1
FTSS closeness	1	16	281
FTSS total	4	17	282

5. Conclusion

We proposed a new string searching algorithm that is based on the F-transform (FTSS). In short, this is a brute-force-style algorithm applied to transformed data using the F-transform. Three standard searching algorithms, the KMP, RK and BM algorithms, were compared to the FTSS algorithm. The comparison was performed on examples with various databases, patterns and alphabets. The results of these examples showed that, among the standard algorithms, the best results were achieved by the BM algorithm. However, the BM algorithm was sensitive to the length of the alphabet. All three algorithms have linear complexity with a multiplier of approximately 1. The proposed FTSS algorithm has sub-linear complexity with a multiplier of approximately 0.5. Moreover, this algorithm is not sensitive to the length of an alphabet. As a result,

Table 2: Processing time for the pattern $t^P_{max} = 3 \cdot 10^5$ and the alphabet $|\Sigma| = 30$.

Algorithm	$t^D_{max} = 1 \cdot 10^6$	$t^D_{max} = 1 \cdot 10^7$	$t^D_{max} = 1 \cdot 10^8$
Pattern is in db			
KMP	10	62	414
RK	6	42	328
BM	**1**	**1**	**31**
FTSS $\mathbf{F^1_m}[o^P]$	39	31	32
FTSS closeness	3	8	30
FTSS total	42	39	62
Pattern is not in db			
KMP	12	48	449
RK	4	32	347
BM	**1**	**3**	**28**
FTSS $\mathbf{F^1_m}[o^P]$	25	24	22
FTSS closeness	1	15	22
FTSS total	26	39	44

Table 3: Processing time for the pattern $t^P_{max} = 3 \cdot 10^4$ and the alphabet $|\Sigma| = 5$.

Algorithm	$t^D_{max} = 1 \cdot 10^6$	$t^D_{max} = 1 \cdot 10^7$	$t^D_{max} = 1 \cdot 10^8$
Pattern is in db			
KMP	10	68	692
RK	6	34	328
BM	**3**	24	213
FTSS $\mathbf{F^1_m}[o^P]$	5	1	5
FTSS closeness	1	2	15
FTSS total	6	**3**	**20**
Pattern is not in db			
KMP	11	70	703
RK	**1**	34	390
BM	3	**18**	**172**
FTSS $\mathbf{F^1_m}[o^P]$	3	1	1
FTSS closeness	6	44	217
FTSS total	9	45	218

Table 4: Processing time for the pattern $t^P_{max} = 3 \cdot 10^5$ and the alphabet $|\Sigma| = 5$.

Algorithm	$t^D_{max} = 1 \cdot 10^6$	$t^D_{max} = 1 \cdot 10^7$	$t^D_{max} = 1 \cdot 10^8$
Pattern is in db			
KMP	17	74	724
RK	4	35	328
BM	**1**	**23**	187
FTSS $\mathbf{F^1_m}[o^P]$	39	26	46
FTSS closeness	4	8	5
FTSS total	43	34	**51**
Pattern is not in db			
KMP	17	79	703
RK	3	32	328
BM	**1**	**20**	191
FTSS $\mathbf{F^1_m}[o^P]$	18	26	27
FTSS closeness	3	32	81
FTSS total	21	58	**108**

the FTSS algorithm has better performance than the BM algorithm in cases with a short alphabet.

To conclude, the FTSS algorithm demonstrated universality and good potential in solving similar problems in various applications. Our future research will be focused on the analysis of the parameters that affect the performance of the proposed algorithm such as the rate of decrease of the parameter h, the multiplier in the estimation of complexity.

Acknowledgment

The research was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and SGS13/PRF/2015.

References

[1] Perfilieva, Irina. "F-transform." in: J. Kacprzyk, W. Pedrycz (Eds.), Handbook of Computational Intelligence, Springer, Berlin, Heidelberg, 2014, in press.

[2] Hodakova, Petra, Irina Perfilieva, and Petr Hurtik. "F-transform and Its Extension as Tool for Big Data Processing." Information Processing and Management of Uncertainty in Knowledge-Based Systems, Communications in Computer and Information Science, 444 (2014): 374–383.

[3] Jacobs, Adam. "The pathologies of big data." Communications of the ACM 52.8 (2009): 36–44.

[4] Kurzweil, Ray. "The singularity is near: When humans transcend biology." Penguin, 2005.

[5] Knuth, Donald E., James H. Morris, Jr, and Vaughan R. Pratt. "Fast pattern matching in strings." SIAM journal on computing 6.2 (1977): 323–350.

[6] Boyer, Robert S., and J. Strother Moore. "A fast string searching algorithm." Communications of the ACM 20.10 (1977): 762–772.

[7] Karp, Richard M., and Michael O. Rabin. "Efficient randomized pattern-matching algorithms." IBM Journal of Research and Development 31.2 (1987): 249–260.

[8] Perfilieva, Irina. "Fuzzy transforms: Theory and applications." Fuzzy Sets and Systems, 157 (2006) 993–1023.

[9] Perfilieva, Irina, Martina Daňková and Barnabas Bede. "Towards a higher degree F-transform." Fuzzy Sets and Systems, 180 (2011) 3–19.

[10] Hutchinson, Lee. Information explosion: how rapidly expanding storage spurs innovation. ArsTechnica [online]. 2011. Available at: http://arstechnica.com/business/2011/09/information-explosion-how-rapidly-expanding-storage-spurs-innovation/