Finite Automata Approach to Computing All Seeds of Strings with the Smallest Hamming Distance

Ondřej Guth, Bořivoj Melichar *

Abstract—**Seed is a type of a regularity of strings. A restricted approximate seed w of string T is a factor of T such that w covers a superstring of T under some distance rule. In this paper, the problem of *all restricted seeds with the smallest Hamming distance* is studied and a polynomial time and space algorithm for solving the problem is presented. It searches for all restricted approximate seeds of a string with given limited approximation using Hamming distance and it computes the smallest distance for each found seed. The solution is based on a finite (suffix) automata approach that provides a straightforward way to design algorithms to many problems in stringology. Therefore, it is shown that the set of problems solvable using finite automata includes the one studied in this paper.**

Keywords: approximate seed, suffix automaton, Hamming distance, stringology

1 Introduction

Searching regularities of strings is used in a wide area of applications like molecular biology, computer–assisted music analysis, or data compression. By regularities, repeated strings are meant. Examples of regularities include repetitions, borders, periods, covers, and seeds.

The algorithm for computing all exact seeds of a string was introduced by Iliopoulos, Moore, and Park [1]. The first algorithm for searching all seeds using finite automata was introduced by Voráček and Melichar [2].

Finding exact regularities is not always sufficient and thus some kind of approximation is used. An algorithm for searching approximate periods, covers, and seeds under Hamming, Levenshtein (also called edit), and weighted Levenshtein distance was presented by Christodoulakis, Iliopoulos, Park, and Sim [3]. The algorithm for computing approximate seeds was originally introduced by these authors in [4]. An algorithm for searching all covers under Hamming, Levenshtein, and Damerau distance using finite automata was introduced by Guth [5], optimized algorithm for computing all covers with smallest Hamming distance was presented by Guth, Melichar, and Balík [6].

*Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Computer Science and Engineering. Email: {gutho1,melichar}@fel.cvut.cz

Finite automata provide common formalism for many algorithms in the area of text processing (stringology), involving forward exact and approximate pattern matching and searching for borders, periods, and repetitions [7], backward pattern matching [8], pattern matching in a compressed text [9], the longest common subsequence [10], exact and approximate 2D pattern matching [11], and already mentioned computing approximate covers [5, 6] and exact covers [12] and seeds [2] in generalized strings. Therefore, we would like to further extend the set of problems solved using finite automata. Such a problem is studied in this paper.

Finite automaton as a data structure may be easily implemented. Therefore, using it as a base for similar approach to many algorithms is not only theoretical problem, as it may make development of software related with above mentioned areas easier, faster, and cost-reduced.

This paper is organized as follows: in Section 2, basic definitions and previous works overview are placed. In Section 3, the algorithm for the problem being studied is presented. Its theoretical time and space complexity is derived in Section 4 and experimental results are shown in Section 5.

2 Preliminaries

An *alphabet* is a nonempty finite set of symbols, denoted by A. The symbol of the alphabet is denoted by a. A *string* over an alphabet is a finite sequence of symbols of the alphabet. Having string $T = a_1, a_2, \dots, a_{|T|}$, *reversed string* T is denoted by T^R and it is equal to $T^R = a_{|T|}, a_{|T|-1}, \dots, a_1$. Empty string is an empty sequence of symbols, denoted by ε. An *effective alphabet* of a string T is a set of symbols that occur in T, denoted by A_T. Only effective alphabet is considered in this paper. A *language* is a set of strings. A set of all strings over alphabet A is denoted by A^*. The length of a string w is denoted by $|w|$, the i–th symbol of w is denoted by $w[i]$. Result of an operation *concatenation* of strings $x, y \in A^*$ is equal to xy, it may be denoted by $x.y$. An operation *superposition* is defined in this way: $x = pu, y = us$, superposition of x and y is pus.

A *distance* is the minimum number of editing operations necessary to convert string x into string y. The maximum allowed distance is denoted by k. The *Hamming distance* between strings x and y, denoted by D_H, is equal to the minimum number of editing operations *replace* (of one symbol) that are necessary to convert x into y. Only Hamming distance is considered in this paper.

Suppose $p, p', s, s', u, w, x, T \in A^*$. p is a *prefix* of T if $T = pu$, s is a *suffix* of T if $T = us$, w is a *factor* (also called a substring) of T if $T = uwx$ (also T is a *superstring* of w). Set of all factors of T is denoted by $Fact(T)$. p' is an *approximate prefix* of T with maximum Hamming distance k if $T = pu$ and $D_H(p, p') \leq k$. Set of all approximate prefixes of T with maximum Hamming distance k is denoted by $Pref^k(T)$. An *approximate suffix* and set of all approximate suffixes of T with respect to k is defined by analogy and is denoted by $Suff^k(T)$.

We say that string w *occurs* in string T if $w \in Fact(T)$. Factor w *occurs at position* (*end-position*) i in string T if $\forall j \in \{1, \ldots, |w|\} : w[j] = T[i - |w| + j]$. An *end-set* is a set of all i such that w occurs at position i in T. String w occurs approximately with maximum Hamming distance k at position i in string T (or w has approximate occurrence at position i in T) if there exists factor x of T that occurs at position i in T and $D_H(x, w) \leq k$.

String w is a *cover* of T if T can be constructed by concatenations and superpositions of w. We also say that w *covers* T. String w is a *seed* of T if w covers some superstring of T. For example, *aba* and *ababa* are some seeds of *ababaab*.

String w is a *restricted approximate cover* of T with maximum Hamming distance k if w is a factor of T and there exist strings $s_1, s_2, \ldots, s_r; s_i \in Fact(T)$ such that:

1. $D_H(w, s_i) \leq k$ for all i where $1 \leq i \leq r$,
2. T can be constructed by superpositions and concatenations of copies of the strings $s_1, s_2, \ldots, s_r$.

String w is a *restricted approximate seed* of string T with maximum Hamming distance k if w is a factor of T and w is a restricted approximate cover of some superstring of T with maximum Hamming distance k.

The *smallest Hamming distance* of a restricted approximate seed w of string T is the smallest possible integer l_m such that w is a restricted approximate seed of T with maximum Hamming distance l_m.

A *finite automaton* M (also called finite state machine) is a quintuple $M = (Q, A, \delta, q_0, F)$, where Q is a nonempty finite set of states, A is an input alphabet, δ is a transition function, $q_0 \in Q$ is an initial state and $F \subseteq Q$ is a set of final states. Finite automaton is *deterministic* (abbreviated to DFA) if its transition function is $\delta : Q \times A \mapsto Q$, i.e. for each pair of state q_i and symbol a, there exists at most one state q_j such that $\delta(q_i, a) = q_j$. DFA is *partial* if there may exist a pair of state q_i and symbol a such that $\delta(q_i, a)$ is undefined. In this paper, partial DFA are considered in general. A deterministic *trie* is a DFA that may have its transition diagram represented as a tree, i.e. for each state q^j, there exists at most one state q^i such that for any symbol $a \in A$, $\delta(q^i, a) = q^j$. Finite automaton is *nondeterministic* (abbreviated to NFA) if its transition function is $\delta : Q \times A \mapsto \mathcal{P}(Q)$, i.e. for some pair of state q_i and symbol a, there may exist more than one state q_j such that $q_j \in \delta(q_i, a)$. A *successor* q^j of a state q^i for symbol a is state from result of a transition function, i.e. $q^j \in \delta(q^i, a)$ for NFA and $q^j = \delta(q^i, a)$ for DFA. An *extended transition function* denoted by δ^* is for a DFA defined for $a \in A, u \in A^*$ in this way:

$$\delta^*(q, \varepsilon) = q, \ \delta^*(q, ua) = \delta(\delta^*(q, u), a)$$

An extended transition function of an NFA is defined as:

$$\delta^*(q, \varepsilon) = \{q\}, \ \delta^*(q, au) = \bigcup_{q_i \in \delta(q,a)} \delta^*(q_i, u)$$

String w is *accepted* by a DFA when $\delta^*(q_0, w) = q$ for initial state q_0 and some final state q. String w is *accepted* by an NFA when $q \in \delta^*(q_0, w)$ for initial state q_0 and some final state q. We also say that the automaton *accepts* string w. Automaton M_1 is *equivalent* to automaton M_2 if M_1 and M_2 accept equal sets of strings. A *left language* of a state q of a DFA is set of all strings w for that holds $\delta^*(q_0, w) = q$ for initial state q_0. Left language of a state q of a trie contains one string, denoted by $factor(q)$.

A nondeterministic suffix automaton for string T and maximum Hamming distance k, denoted by $M^k_{S^N}(T)$, is an NFA that accepts all strings from $Suff^k(T)$ (see Figure 3 for an example of such automaton). Such an automaton $M^k_{S^N}(T) = (Q, A_T, \delta, q_0, F)$ may be constructed in this way:

1. Create a layer of $|T| + 1$ states:
   (a) each state q^0_i corresponds to a position i in T (plus initial state q_0, thus $0 < i \leq |T|$),
   (b) for each state q^0_i (but the last $q^0_{|T|}$) define transition $\delta(q^0_i, T[i]) = q^0_{i+1}$,
   (c) define the last state $q^0_{|T|}$ final (note that until now such automaton accepts exactly T).
2. Similarly, create a layer for each "number of errors" $l, 1 \leq l \leq k$ (the only exception: we do not need any state q^l_i for $l > i$).
3. For each state q^l_i (but the last $q_{|T|}$ in each layer and but the last layer) and for each symbol $a \in A_T, a \neq T[i]$ (not occurring in T at position i), define transition $\delta(q^l_i, T[i]) = q^{l+1}_{i+1}$.
4. Create "long" transitions from q_0: $\delta(q_0, a) = \{q^0_i : a = T[i], 1 \leq i \leq |T|\} \cup \{q^1_i : a \neq T[i], 1 \leq i \leq |T|\}$.

A *level* of a state of $M^k_{S^N}(T)$ corresponds to the number of errors (order l of the layer mentioned above), a *depth* of a state of this automaton is equal to the corresponding position in T (number i mentioned above).

A deterministic suffix automaton for string T and maximum Hamming distance k, denoted by $M^k_{S^D}(T)$, is a DFA that accepts all strings from $Suff^k(T)$ (see Figure 5). A *depth* of a state q of the automaton is length of the longest string w such that $\delta^*(q_0, w) = q$ for initial state q_0. Deterministic automaton $M^k_{S^D}(T) = (Q, A_T, \delta, q_0, F)$ accepting the same language as nondeterministic automaton $M^k_{S^N}(T) = (Q_N, A_T, \delta_N, q_0^N, F_N)$ may be created using *subset construction*:

1. Set $Q = \{\{q_0\}\}$ will be defined, state $q_0 = \{q_0^N\}$ will be treated as unmarked.
2. If each state in Q is marked, continue with step 4.
3. Unmarked state q will be chosen from Q and the following operations will be executed:
   (a) $\delta(q, a) = \bigcup \delta_N(r, a)$ for $r \in q$ and for all $a \in A_T$,
   (b) $Q = Q \cup \delta(q, a)$ for all $a \in A_T$,
   (c) state $q \in Q$ will be marked,
   (d) continue with step 2.
4. $F = \{q : q \in Q, r \cap F_N \neq \emptyset, r \in q\}$.

Using subset construction of DFA $M^k_{S^D}(T)$ equivalent to NFA $M^k_{S^N}(T)$, every state $q_D \in Q$ corresponds to some subset of Q_N. This subset is called a *d–subset* (abbreviation of deterministic subset), denoted by $d(q_D)$. Each element of the d–subset corresponds to some state of Q_N. Where no confusion arises, depth of a state corresponding to an element $r_j \in d(q_D)$ of d–subset $d(q_D)$ is simply denoted by r_j. Note that considering only depths of elements, any d–subset $d(q)$ is equal to the end-set of all strings from left language of q. In the algorithms below, d–subset is supposed to be implemented as a list, preserving order of its elements. An element of the d–subset is denoted by r_i, where the subscript i means an index (order) of the element r_i within the d–subset. In figures, states of nondeterministic automata and elements of d–subsets of deterministic automata are denoted by their depths and levels, e.g. $3''$ means state or element with depth 3 and level 2.

Problem formulation (All restricted seeds with the smallest Hamming distance)**.** Given string T and maximum Hamming distance k, find all restricted approximate seeds of T with respect to k and compute their smallest distances.

The algorithm for searching (exact) seeds in generalized strings presented in [2] obviously works for (non-generalized) strings as well, because string is special case of generalized string. It is based on the following idea. First, $M_{S^N}(T)$ is constructed. Equivalent deterministic automaton $M_{S^D}(T) = (Q, A_T, \delta, q_0, F)$ is computed using subset construction. One of conditions for any factor to be a seed of string T is its length. Seed w must cover central part of T (i.e. the part of T between the leftmost and the rightmost position of w within T), and it must cover the uncovered suffix of T and the uncovered prefix of T. All sufficiently long factors are then checked whether they cover the uncovered suffix, prefix of T, respectively. If $M_{S^D}(T)$ accepts some prefix of factor w then w covers uncovered suffix of T. If a suffix automaton $M_{S^D}(T^R)$ for reversed string T accepts some prefix of reversed factor w then w covers uncovered prefix of T. When w satisfies all the conditions, w is a seed of T.

Computation of the smallest Hamming distance of a cover (presented in [6]) is based on the following idea: when the maximum approximation of the first and the last position of cover w in T is $l_{\min}$, for its smallest distance l_m holds $l_m \geq l_{\min}$, because cover is an approximate prefix and suffix of T and thus it cannot cover T without its first and last position. When cover w of T has positions with approximation at most l, for its smallest distance l_m clearly holds $l_m \leq l$. When the positions of w with the maximum approximation equal to l are no longer considered (the first and the last position must be still considered) and w is still cover of T, then for l_m holds $l_m \leq l - 1$. l_m is decremented till w still covers T. This may be used with modifications for computation of seeds.

The algorithm for searching exact seeds from [2] uses two phases: first, deterministic suffix automaton is constructed and then d–subsets are analyzed and seeds are computed. This means that complete automaton or at least all the d–subsets to be analyzed need to be stored in memory at a time. By contrast, the algorithm for searching covers from [6] uses merge of the phases, each d–subset is analyzed just after its construction. A depth-first search like algorithm is used and the states that are no longer needed are removed. For approximate seeds searching, there is also no need to store all elements of d–subsets of the automaton in memory at a time.

3 Problem solution

Some properties are common for exact and approximate seeds with Hamming distance. Hence the algorithm presented in [2] is used as a base of algorithm for the problem studied in this paper, using some (but not all) techniques for searching covers in [6] for further improvements.

Every approximate restricted seed of string T is necessarily an exact factor of T with other possible approximate occurrences. Suffix automaton constructed for T and maximum Hamming distance k has extended transitions defined for all factors of T with respect to k. When $M^k_{S^D}(T)$ is constructed using subset construction from

$M^k_{S^N}(T)$, each element of any d–subset of any state of $M^k_{S^D}(T)$ contains information not only about position (depth within $M^k_{S^N}(T)$) but also about approximation (level within $M^k_{S^N}(T)$). Therefore, it may be easily determined whether string from left language of any state of $M^k_{S^D}(T)$ is an exact factor of T.

Note 1. For string T, string v such that v is not a factor of T, and any string u being a superstring of v holds:

$$\forall T, u, v \in A^*, v \in \mathit{Fact}(u) : v \notin \mathit{Fact}(T) \Rightarrow u \notin \mathit{Fact}(T)$$

Lemma 1. *For DFA $M^k_{S^D}(T) = (Q_D, A_T, \delta_D, q^D_0, F_D)$ created using subset construction from $M^k_{S^N}(T)$ and for its state $q_i \in Q_D$ with d–subset $d(q_i)$ such that $\forall r \in d(q_i)$: $level(r) > 0$ holds that any successor of q_i cannot contain element r^j such that $level(r^j) = 0$.*

Proof. It holds from the following property of transitions of $M^k_{S^N}(T) = (Q_N, A_T, \delta_N, q^N_0, F_N)$: for all successors $r^j \in Q_N$ of $r^i \in Q_N$ holds that $level(r^i) \leq level(r^j)$. □

Corollary. *As only exact factor of T may be a restricted seed of T, there is no need to construct any state of $M^k_{S^D}(T)$ having only non-zero-level elements in its d–subset, as such state contains no exact factor of T in its left language. Therefore, when such state is created during construction, it may be removed and any of its successors need not be constructed. Such deterministic suffix automaton that contains only states having at least one zero-level element in its d–subset is denoted by $\tilde{M}^k_{S^D}(T)$.*

Note 2. Special type of deterministic suffix automaton, suffix trie, is considered in this paper. Construction of the trie and left language extraction is simpler than for general suffix automaton. As left language of any state of the trie contains exactly one string, extraction of left language of any state takes linear time with respect to length of the string (e.g. using inverted transition function). See Figure 5 for example of suffix trie.

The relation between length and positions of any seed (presented in [2]) holds also for approximate positions with Hamming distance, as the distance is defined for strings of equal lengths only.

Note 3. When searching for covers with Hamming distance [6], it is possible to remove all states q of deterministic suffix trie that do not represent prefix, i.e. such q that $|factor(q)| < depth(r_1)$, where $d(q) = r_1, \ldots, r_{|d(q)|}$. Similar property between the first position and length of a factor is used for searching seeds: $|factor(q)| \leq \frac{depth(r_1)}{2}$. Unlike in computing covers, this condition cannot be used for removing states q and their successors.

Example 1. Let us consider suffix trie for string $T = bbbbbaaabb$ and maximum Hamming distance $k = 2$. Factor aaa cannot be a seed of T as its first approximate position within T is 6. Factor $aaabb$ is a seed of T with respect to k. It is obvious that for states q_1, q_2 of the trie, where $factor(q_1) = aaa$ and $factor(q_2) = aaabb$, holds: q_2 is a successor of a state that is a successor of q_1. Therefore, q_1 must not be removed to be able to find $aaabb$.

```
bbbbbaaa
bbba
 bbba
  bbba
   bbba
    bbba
```

Figure 1: Possible covering of string *bbbbbaaa* with string *bbba* and Hamming distance 2 from Example 2

```
bbbbbaaa
bbba
 bbba
  bbba
   bbba
       b
```

Figure 2: Possible covering of a superstring of *bbbbbaaa* with *bbba* and Hamming distance 1 from Example 2

For computation of the smallest distance l_m of each seed, the idea used for searching covers ([6]) may be used for searching seeds. Unlike searching covers, any position may be removed, including the first and the last, thus the only lower bound of l_m is 0. Determination whether continue to decrement l is for seeds more complex than for covers, as computation of covering of central part of T is not sufficient condition for seeds. For factor w of T, not only positions and their approximation need to be considered, but also distance of the uncovered prefix, suffix, of T, and some suffix, prefix, of w, respectively. See Algorithm 3 for further information.

Example 2. Let us have string $T = bbbbbaaa$ and maximum Hamming distance $k = 2$. One seed of T with respect to k is *bbba*. It may be seed of T with Hamming distance 2, because its positions in T are 4, 5, 6, 7, and 8 with maximum approximation 2 (see Figures 1, 5). When the position 8 with approximation 2 is removed, *bbba* is still seed of T with positions 4, 5, 6, and 7, all with approximation at most 1 (see Figure 2).

The deterministic suffix trie is needed not only to determine positions of each factor w of T, but also for checking whether w is able to cover uncovered prefix and suffix of T (see Algorithm 4). Thus, the trie must be able to accept strings of length at least $|w| - 1$. Therefore, the depth-first search with removing states from [6] cannot be used. By contrast, only elements of d–subset $d(q)$ may be removed after construction of all successors of q, transitions must be preserved. Thus, breadth-first search in the automaton is used (see Algorithm 1 and usage of queues

L, L^R). As no state of trie $\tilde{M}^k_{SD}(T)$ is removed and the last element of each d–subset is preserved, it is possible to recognize all approximate suffixes of T of length at least $|w| - 1$ and their distance.

Like an exact seed, an approximate one must also cover the uncovered prefix and suffix of T (i.e. some prefix of seed w must be an approximate suffix of T and some suffix of w must be an approximate prefix of T). Similar technique as for exact seeds ([2]) is used (Algorithm 4), but with tries $\tilde{M}^k_{SD}(T)$ and $\tilde{M}^k_{SD}(T^R)$. When some suffix of seed w of T (i.e. some prefix of reversed w) is accepted by $\tilde{M}^k_{SD}(T^R)$, i.e. $w = factor(q)$ for some final state q of $\tilde{M}^k_{SD}(T^R)$, w covers the uncovered prefix of T with approximation equal to level of the last element $r_{|d(q)|}$ of $d(q)$. Similarly for a prefix of w, the uncovered suffix of T and $\tilde{M}^k_{SD}(T)$.

For complete solution of the problem see Algorithm 1.

Algorithm 2 Compute state of a deterministic suffix trie $\tilde{M} = Q_D, A_T, \delta_D, q_0^D, F_D$.

Input: NSA $(Q_N, A_T, \delta_N, q_0^N, F_N)$, state $q^t \in Q_D$, symbol $a \in A_T$, queue L of states.
Output: Modified $\tilde{M}$ with possibly added successor q^u of state q^t for symbol a, modified queue L.

create new state q^u
define $depth(q^u) = depth(q^t) + 1$
for all $r^i \in d(q^t)$ (in order as stored in $d(q^t)$) **do**
  append all $r^j \in \delta_N(r^i, a)$ to $d(q^u)$ in ascending order by $depth(r^j)$
end for
if exists $r \in d(q^u)$ where $level(r) = 0$ **then**
  $Q_D \leftarrow Q_D \cup \{q^u\}$
  enqueue(L, q^u)
  if $r^u_{|d(q^u)|} \in F_N, d(q^u) = r^u_1, \dots r^u_{|d(q^u)|}$ **then**
    $F_D \leftarrow F_D \cup \{q^u\}$
  end if
end if

Algorithm 3 The smallest distance of a seed of T.

Input: d–subset $d(q) = r_1, r_2, \dots, r_{|d(q)|}$ representing seed w of T.
Output: The smallest distance l_m of w.

$t \leftarrow d(q)$
$l_{\max} \leftarrow \max_{r \in t}\{level(r)\}$
$l \leftarrow l_{\max}$
repeat
  for all $r \in t : level(r) = l$ **do**
    remove r from t
  end for
  $l \leftarrow l - 1$
until w is a seed of T using positions determined by t with respect to l (Algorithm 4)
$l_m \leftarrow l + 1$.

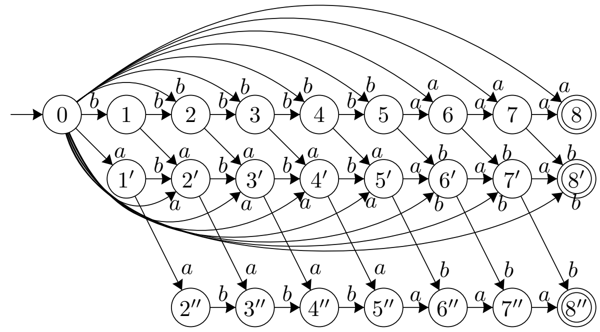

Figure 3: Transition diagram of the nondeterministic approximate suffix automaton $M^k_{SN}(T)$ for string $T = bbbbbaaa$ and maximum Hamming distance $k = 2$ from Example 3

Example 3. Let us compute set of all seeds with maximum Hamming distance $k = 2$ for string $T = bbbbbaaa$. Nondeterministic suffix automata $M^k_{SN}(T)$ (see Figure 3) and $M^k_{SN}(T^R)$ (see Figure 4) are constructed. Next, subset construction of deterministic suffix trie $M^k_{SD}(T)$ from $M^k_{SN}(T)$ starts state-by-state (see transition diagram of $M^k_{SD}(T)$ with all states, that need to be constructed, at Figure 5), the same is done with trie $M^k_{SD}(T^R)$ from $M^k_{SN}(T^R)$. Some states may have only elements with non-zero level in its d–subset (e.g. $7''8'$). Such states are removed and their successors are not constructed as strings from their left languages (e.g. $aaaa$) are not factors of T (follows by Corollary of Lemma 1).

All other states need to be checked whether their left languages contain some seeds. For example, state with d–subset $6''7'8$ contains string aaa in its left language. The string occurs approximately at positions 6, 7 in T and exactly at position 8 in T, thus it cannot be seed of T, as its leftmost occurrence within T ends at position 6 and its length is 3 (i.e. the occurrence starts at position 4), so any proper suffix of aaa cannot cover the uncovered prefix (positions 1 to 3) of T.

Other example is state with d–subset $4'5'67'8''$, which contains string $bbba$ in its left language. This string covers T with Hamming distance 2, and therefore it is seed of T (see Figure 1). When all positions with the maximum distance (i.e. 8) are not considered, $bbba$ is still seed of T, as proper prefix b of $bbba$ covers uncovered suffix a of T with Hamming distance 1 (see Figure 2). See resulting table of all seeds and their distances in Table 1.

4 Time and space complexities

Note 4. As parts of $M^k_{SD}(T)$ and $M^k_{SD}(T^R)$ are constructed the same way in Algorithm 1, the time and space complexities of their construction are the same.

Lemma 2. *Left languages of states of deterministic suffix trie* $\tilde{M}^k_{SD}(T) = (Q_D, A_T, \delta_D, q_0^D, F_D)$ *are distinct, i.e.*

$$q_1, q_2 \in Q_D; q_1 \neq q_2 \Rightarrow factor(q_1) \neq factor(q_2)$$

Algorithm 1 Compute set of seeds of T with the smallest Hamming distances.

Input: String T, maximum Hamming distance k.
Output: Set $hseeds^k(T)$ of all seeds of T.

$hseeds^k(T) \leftarrow \emptyset$
construct $M^k_{SN}(T) = (Q_N, A_T, \delta_N, q_0^N, F_N)$
construct $M^k_{SN}(T^R) = (Q_N^R, A_T, \delta_N^R, q_0^{NR}, F_N^R)$
create new state q_0^D as the initial one of the deterministic suffix trie $M^k_{SD}(T) = (Q_D, A_T, \delta_D, q_0^D, F_D)$
create new state q_0^{DR} as the initial one of the deterministic suffix trie $M^k_{SD}(T^R) = (Q_D^R, A_T, \delta_D^R, q_0^{DR}, F_D^R)$
define $factor(q_0^D) = \varepsilon$, $depth(q_0^D) = 0$, $depth(q_0^{DR}) = 0$
create L, L^R new empty queues of states
enqueue(L, q_0^D), enqueue(L^R, q_0^{DR})
while L^R is not empty {construct complete $\tilde{M}^k_{SD}(T^R)$ in this loop} **do**
  $q^{tR} \leftarrow$ dequeue(L^R)
  for all $a \in A_T$ **do**
    compute new state q^{uR} as a successor of state q^{tR} for symbol a using Algorithm 2
    discard all elements of $d(q^{tR})$ but the last one {all successors of $d(q^{tR})$ have just been computed}
  end for
end while
while L is not empty {construct $\tilde{M}^k_{SD}(T)$ and compute seeds in this loop} **do**
  $q^t \leftarrow$ dequeue(L)
  for all $a \in A_T$ **do**
    compute new state q^u as a successor of state q^t for symbol a using Algorithm 2
    if exists $r \in d(q^u)$ where $level(r) = 0$ {only state q^u that is part of $\tilde{M}^k_{SD}(T)$ is further processed} **then**
      define $w = factor(q^u) = factor(q^t).a$
      if w is a seed of T using positions determined by $d(q^u)$ (Algorithm 4) **then**
        compute the smallest distance l_m of w (Algorithm 3)
        if $|w| > k$ or $l_m < |w|$ {all strings of length less or equal to l_m are seeds} **then**
          $hseeds^k(T) \leftarrow hseeds^k(T) \cup \{(w, l_m)\}$
        end if
      end if
    end if
  end for
  discard all elements of $d(q^t)$ but the last one
end while

Algorithm 4 Determine whether string w is a seed of T with maximum Hamming distance l.

Input: Already constructed parts of deterministic suffix automata $M^k_{SD}(T) = (Q, A_T, \delta, q_0, F)$ and $M^k_{SD}(T^R) = (Q^R, A_T, \delta_R, q_0^R, F^R)$, d–subset $t = r_1, r_2, \ldots, r_{|t|}$ for $q \in Q$ and $w \in factor(q)$, maximum Hamming distance l.
Output: Resolution whether w is a seed of T with respect to l and t.

if for all $i = 2, 3, \ldots, |t| : r_i - r_{i-1} \leq |w|$
and $\exists p \in Pref^0(w), |p| \geq |T| - r_{|t|} : \delta^*(q_0, p) = q^1, q^1 \in F \wedge level(r^{q^1}_{|d(q^1)|}) \leq l$
and $\exists s \in Suff^0(w), |s| \geq r_1 - |w| : \delta^*_R(q_0^R, s) = q^2, q^2 \in F^R \wedge level(r^{q^2}_{|d(q^2)|}) \leq l$ **then**
  return true
else
  return false
end if

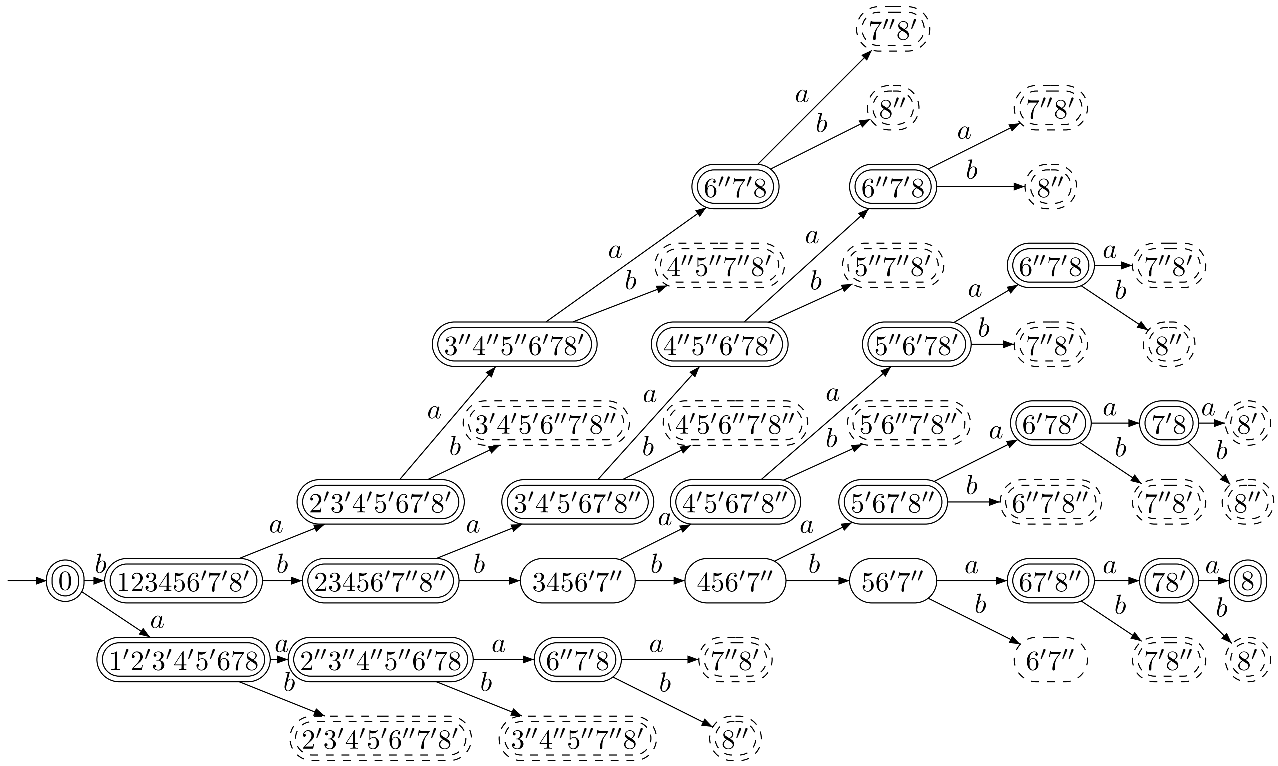

Figure 5: Transition diagram of the constructed part of the suffix trie $M^k_{SD}(T)$ for string $T = bbbbbaaa$ and maximum Hamming distance $k = 2$ from Example 3; dashed states are removed as their left language do not contain exact factor of T and thus they are not states of $\tilde{M}^k_{SD}(T)$

Table 1: All seeds of string $T = bbbbbaaa$ with maximum Hamming distance $k = 2$ and their smallest distances l_m; p is used prefix of a seed, s is used suffix (both computed by Algorithm 4); see Example 3

seed	d–subset	l_m	occurrences	p	s
ba	$2'3'4'5'67'8'$	1	2,3,4,5,6,7,8	ε	ε
baa	$3''4''5''6'78'$	2	3,4,5,6,7,8	ε	ε
bba	$3'4'5'67'8''$	1	3,4,5,6,7	b	ε
bbb	$3456'7''$	2	3,4,5,6,7	b	ε
$baaa$	$6''7'8$	2	6,7,8	ε	aa
$bbaa$	$4''5''6'78'$	2	4,5,6,7,8	ε	aa
$bbba$	$4'5'67'8''$	1	4,5,6,7	b	ε
$bbbb$	$456'7''$	2	4,5,6,7	b	ε
$bbaaa$	$6''7'8$	2	6,7,8	ε	a
$bbbaa$	$5''6'78'$	1	6,7,8	ε	a
$bbbba$	$5'67'8''$	1	5,6,7	b	ε
$bbbbb$	$56'7''$	2	5,6,7	b	ε
$bbbaaa$	$6''7'8$	1	7,8	ε	a
$bbbbaa$	$6'78'$	1	6,7,8	ε	ε
$bbbbba$	$67'8''$	1	6,7	b	ε
$bbbbaaa$	$7'8$	1	7,8	ε	ε
$bbbbbaa$	$78'$	1	7,8	ε	ε
$bbbbbaaa$	8	0	8	ε	ε

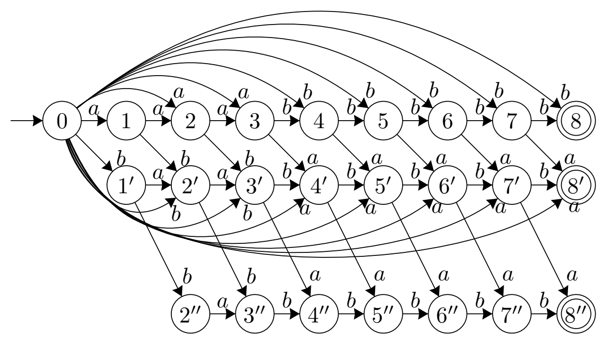

Figure 4: Transition diagram of the nondeterministic approximate suffix automaton $M^k_{SN}(T^R)$ for reversion of string T, i.e. $T^R = aaabbbbb$, and maximum Hamming distance $k = 2$ from Example 3

Proof by contradiction. Let us have following consideration: if there existed two states $q_1, q_2 \in Q_D, q_1 \neq q_2$ and $factor(q_1) = factor(q_2) = w$, it would mean existence of two distint sequences of transitions: $\delta^*_D(q^D_0, w) = q_1$ and $\delta^*_D(q^D_0, w) = q_2$. As Algorithm 1 creates new state for every $a \in A_T$, the resulting automaton $\tilde{M}^k_{SD}(T)$ is deterministic, so such distinct sequences of transitions for the same string are not possible, thus either $q_1 = q_2$ or $factor(q_1) \neq factor(q_2)$. $\square$

Definition 1. Let us consider string T and maximum Hamming distance k. When a factor w approximately occurs e-times in T with respect to k, we say that *number of repetitions of w in T with respect to k*, denoted by $R_w^k(T)$, is $e-1$. Then *number of repetitions of all factors of T with respect to k*, denoted by $R^k(T)$, is defined as

$$R^k(T) = \sum_{w \in Fact(T)} R_w^k(T)$$

Lemma 3. *Number of states of $\tilde{M}_{SD}^k(T)$ is*

$$\frac{1}{2} \cdot (|T|^2 + |T|) - R^k(T) + 1$$

Proof. Number of exact factors of T is $\frac{1}{2} \cdot (|T|^2 + |T|)$. As left language of each state of $\tilde{M}_{SD}^k(T)$ contains exactly one string, the number of states of $\tilde{M}_{SD}^k(T)$ cannot be greater. By Lemma 2, a factor of T is contained in left language of exactly one state independent of number of its repetitions, therefore $R^k(T)$ is subtracted. □

Note 5. As restricted approximate seeds of string T are exact factors of T, it is meaningful to consider effective alphabet A_T only and $|A_T| \leq |T|$ always holds (recall that effective alphabet A_T consists only of symbols that occur in T). It is also meaningless to consider high k, because every factor of T having length less or equal to k is always approximate seed of T. Thus $k \leq |T|$ always holds. Usually k and $|A_T|$ are independent of $|T|$.

Lemma 4. *Number of states of $M_{SD}^k(T)$ constructed using Algorithm 1 is at most*

$$|A_T| \cdot (\frac{1}{2} \cdot (|T|^2 + |T|) - R^k(T)) + 1$$

Proof. By Lemma 3, number of states of $\tilde{M}_{SD}^k(T) = (\tilde{Q}_D, A_T, \tilde{\delta}_D, \tilde{q}_0^D, \tilde{F}_D)$ is $\frac{1}{2} \cdot (|T|^2 + |T|) - R^k(T) + 1$. But using Algorithm 1 there are also constructed (but not stored) more states that have strings not being factors of T in their left languages. For every state $q \in \tilde{Q}_D$ there is constructed a successor q^j for each $a \in A_T$, but not every q^j is in $\tilde{Q}_D$. Number of such successors varies from 0 to $|A_T|$ for each state of $\tilde{M}_{SD}^k(T)$ (but the initial one, which has successors in $\tilde{Q}_D$ for all $a \in A_T$), thus there could be at most $|A_T| \cdot (\frac{1}{2} \cdot (|T|^2 + |T|) - R^k(T)) + 1$ states constructed. □

Lemma 5. *For every d–subset of $\tilde{M}_{SD}^k(T)$ constructed by Algorithm 1 holds that there are no two elements having the same depth.*

Proof. It holds from properties of transition function of $M_{SN}^k(T) = (Q_N, A_T, \delta_N, q_0^N, F_N)$: for successors of the initial state q_0^N holds:

$$\forall a \in A_T : \forall r_i, r_j \in \delta_N(q_0^N, a) : depth(r_i) \neq depth(r_j)$$

Therefore, d–subsets of successors of initial state of $\tilde{M}_{SD}^k(T) = (\tilde{Q}_D, A_T, \tilde{\delta}_D, \tilde{q}_0^D, \tilde{F}_D)$ contain elements with distinct depths only. For any successors r_j of all states r_i of $M_{SN}^k(T)$ but the initial one holds:

$$\forall a \in A_T, \forall r_i : \forall r_j \in \delta_N(r_j, a) : depth(r_j) = depth(r_i) + 1$$

Let us use induction. Successors of initial state of $\tilde{M}_{SD}^k(T)$ have no elements with the same depth in its d–subset. Let us consider any state $q_i \in \tilde{Q}_D \setminus \{\tilde{q}_0^D\}$ having no elements with the same depth in its d–subset. Any successor q_j of such state q_i cannot have d–subset having some elements with the same depth, as any element r^s of $d(q_j)$ is constructed from element $r^i \in d(q_i)$ this way: $r^s \in \delta_N(r^i, a), a \in A_T$ and thus $depth(r^s) = depth(r^i) + 1$. Therefore, the Lemma holds for all d–subsets of $\tilde{M}_{SD}^k(T)$. □

Lemma 6. *Number of elements of all d–subsets of $\tilde{M}_{SD}^k(T)$ is not greater than*

$$\frac{1}{2} \cdot (|T|^3 + |T|^2) - |T| \cdot R^k(T) + 1$$

Proof. By Lemma 3, number of states of $\tilde{M}_{SD}^k(T)$ is at most $\frac{1}{2} \cdot (|T|^2 + |T|) - R^k(T) + 1$. As for transition function of $M_{SN}^k(T) = (Q_N, A_T, \delta_N, q_0^N, F_N)$ holds:

$$\forall a \in A_T : |\delta_N(q_0^N, a)| = |T|$$

and

$$\forall a \in A_T, \forall r \in Q_N \setminus \{q_0^N\} : |\delta_N(r, a)| \leq 1$$

and by Lemma 5, it is obvious that for all states q of $\tilde{M}_{SD}^k(T)$ holds $|d(q)| \leq |T|$ and moreover for initial state $\tilde{q}_0^D$ of $\tilde{M}_{SD}^k(T)$ holds $|d(\tilde{q}_0^D)| = 1$. Therefore, number of elements of all d–subsets cannot be greater than $|T|$-times number of states but the initial one. □

Lemma 7. *Number of elements of all d–subsets of $M_{SD}^k(T)$ constructed using Algorithm 1 is not greater than*

$$|A_T| \cdot (\frac{1}{2} \cdot (|T|^3 + |T|^2) - |T| \cdot R^k(T)) + 1$$

Proof. Clearly holds by Lemma 4 and 6. □

Lemma 8. *Time complexity of the check whether d–subset $d(q)$ of $\tilde{M}_{SD}^k(T)$ represents a seed $w = factor(q)$ of T (Algorithm 4) is at most*

$$2 \cdot |d(q)| + 2 \cdot |w| - 2$$

that is $\mathcal{O}(|T|)$.

Proof. The check whether w covers central part of T (comparison of each two subsequent elements' depth) takes $2 \cdot |d(q)| - 2$, which is $\mathcal{O}(|T|)$ by Lemma 5. The check of existence of a prefix of w to cover uncovered suffix of T takes $|w|$, that is $\mathcal{O}(|T|)$, as it is found during reading

w as input for constructed part of $\tilde{M}^k_{SD}(T)$. The check of existence of a suffix of w to cover uncovered prefix of T takes also $|w|$, as it is found during reading w backwards as input for constructed part of $\tilde{M}^k_{SD}(T^R)$. □

Lemma 9. *Time complexity of the computation of the smallest distance of seed $w = factor(q)$ of T (Algorithm 3) is at most*

$$|d(q)| + k \cdot (3 \cdot |d(q)| + 2 \cdot |w| - 2)$$

that is $\mathcal{O}(k \cdot |T|)$.

Proof. Retrieving $l_{\max}$ takes $|d(q)|$. Then there are at most $l \leq k$ iterations in Algorithm 3. Each iteration means removal of some elements of a d–subset and check whether w is still seed of T (Lemma 8). □

Note 6. Number of all seeds is $\mathcal{O}(|T|^2)$ (like number of factors). Thus, the sum of their lengths is $\mathcal{O}(|T|^3)$, denoted by $|hseeds^k(T)|$.

Theorem 1. *Time complexity of computation of all seeds with their smallest distance for string T with maximum Hamming distance k (Algorithm 1) is*

$$\mathcal{O}(k \cdot |A_T| \cdot |T|^3)$$

Proof. Construction of nondeterministic suffix automaton $M^k_{SN}(T) = (Q_N, A_T, \delta_N, q_0^N, F_N)$ for T and k takes $\mathcal{O}(k \cdot |A_T| \cdot |T|)$. For each state of $\tilde{M}^k_{SD}(T)$ (Lemma 3) and for each symbol of A_T, new d–subset is constructed. As each element of any d–subset may be constructed in constant time (just using already known δ_N) and the elements are naturally ordered (no need to sort – proven in [6]), all d–subsets are constructed in at most

$$|A_T| \cdot (\frac{1}{2} \cdot (|T|^3 + |T|^2) - |T| \cdot R^k(T)) + 1$$

time. Each d–subset is checked whether it contains element with zero level in linear time. The left language extraction of state takes linear time and by Lemma 8 and 3 the theorem holds. □

Lemma 10. *During construction of $\tilde{M}^k_{SD}(T)$ (Algorithm 1), there are at most $\mathcal{O}(|T|^2)$ elements of d–subsets stored in memory at a time.*

Proof. Number of factors of T of equal length z is at most $\min(|T| - z + 1, |A_T|^z)$. Number of approximate positions of such factor is also at most $|T| - z + 1$. As Alg. 1 uses breadth-first search for the construction, there are sometimes states with equal length z only. In such case, there are $\mathcal{O}(|T|^2)$ elements in L. Otherwise, there are stored states with depths z and $z + 1$, so number of elements in L stored at a time is $\mathcal{O}(|T|^2) + \mathcal{O}(|T|^2) = \mathcal{O}(|T|^2)$. □

Theorem 2. *Space complexity of computation of all seeds is*

$$\mathcal{O}(|T|^2 + |hseeds^k(T)|)$$

Proof. Space complexity of construction of $M^k_{SN}(T)$ is $\mathcal{O}(k \cdot |A| \cdot |T|)$ (proven in [6]). By Lemma 10, number of elements stored in memory at a time is $\mathcal{O}(|T|^2)$, as no more elements of d–subsets than those in L plus $\mathcal{O}(|T|)$ new are in memory at a time. By Lemma 3, number of states of $\tilde{M}^k_{SD}(T)$ is $\mathcal{O}(|T|^2)$ (they all are stored in memory with one element each). As the constructed automaton is trie, number of transitions is also $\mathcal{O}(|T|^2)$. The space complexity also depends on size of result, $|hseeds^k(T)|$. □

5 Experimental results

The algorithm was implemented in C++ using STL and compiled using GNU C++ 3.4.6 with O3 optimizations level. The dataset used to test the algorithm is the nucleotide sequence of Saccharomyces cerevisiae chromosome IV[1]. The string T consists of the first $|T|$ characters of the chromosome.

The first set of tests was run on an AMD Athlon 64 3200+ (2200 MHz) system, with 2.5 GB of RAM, under Gentoo Linux operating system (see Figures 6 and 7).

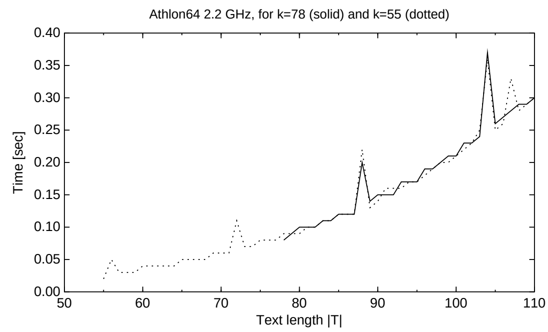

Figure 6: Time consumption of the experimental run on the Athlon64 with respect to the text size (see Section 5)

The second set of tests was run on an AMD Athlon (1400 MHz) system, with 1.2 GB of RAM, under Gentoo Linux operating system (see Figure 8).

Note 7. In comparison to experimental results presented in [3], the algorithm presented in this paper runs a bit faster for the same data, even on a slightly slower computer (1.3 seconds in [3] for text length 100 vs. maximum 0.7 second for text length 113 – see Figure 8).

6 Conclusion

In this paper, we have shown that an algorithm design based on determinization of a suffix automaton is appro-

[1] The Saccharomyces cerevisiae chromosome IV dataset could be downloaded from http://www.genome.jp/.

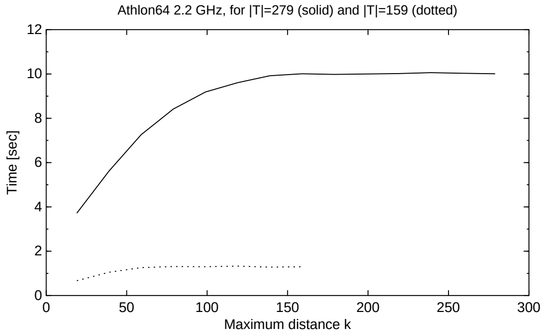

Figure 7: Time consumption of the experimental run on the Athlon64 with respect to the maximum distance (see Section 5)

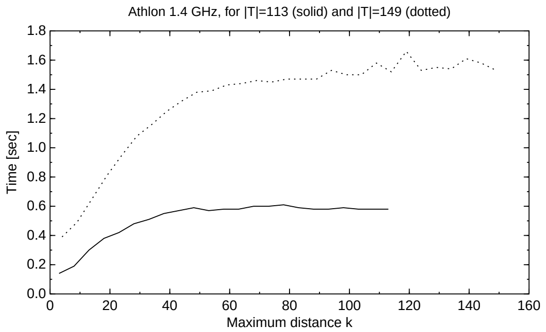

Figure 8: Time consumption of the experimental run on the Athlon with respect to the maximum distance (see Section 5)

priate for computation of all restricted seeds with the smallest Hamming distance. The presented algorithm is straightforward, easy to understand and to implement and its theoretical and experimental time requirements are comparable to the existing approach ([4]).

For the future work, we would like to extend the algorithm for searching seeds to other distances and to utilize similar approach for searching other types of regularities.

Acknowledgments

This research was supported by the Czech Technical University in Prague as grant No. CTU0803113 and as grant No. CTU0915313, by the Ministry of Education, Youth and Sports of the Czech Republic under research program MSM 6840770014, and by the Czech Science Foundation as project No. 201/06/1039 and as project No. 201/09/0807.

References

[1] Iliopoulos, C. S., Moore, D., and Park, K. S., "Covering a String," *CPM '93: Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, Springer-Verlag, London, UK, 1993, pp. 54–62.

[2] Voráček, M. and Melichar, B., "Computing Seeds in Generalized Strings," *Proceedings of Workshop 2006*, Czech Technical University in Prague, 2006, pp. 138–139.

[3] Christodoulakis, M., Iliopoulos, C. S., Park, K. S., and Sim, J. S., "Implementing Approximate Regularities," *Mathematical and Computer Modelling*, Vol. 42, October 2005, pp. 855–866.

[4] Christodoulakis, M., Iliopoulos, C. S., Park, K. S., and Sim, J. S., "Approximate Seeds of Strings," *Journal of Automata, Languages and Combinatorics*, Vol. 10, No. 5/6, 2005, pp. 609–626.

[5] Guth, O., "Searching Approximate Covers of Strings Using Finite Automata," *POSTER 2008*, Czech Technical University in Prague, Faculty of Electrical Engineering, Praha, 2008.

[6] Guth, O., Melichar, B., and Balík, M., "Searching All Approximate Covers and Their Distance using Finite Automata," *Information Technologies – Applications and Theory*, Univerzita P. J. Šafárika, Košice, 2008, pp. 21–26.

[7] Melichar, B., Holub, J., and Polcar, T., "Text Searching Algorithms, Volume I," November 2005, Available at `http://stringology.org/athens`.

[8] Melichar, B., "Text Searching Algorithms, Volume II," March 2006, Available at `http://stringology.org/athens`.

[9] Lahoda, J., Melichar, B., and Žďárek, J., "Pattern Matching in DCA Coded Text," *Proceedings of the 13th International Conference on Implementation and Application of Automata*, Springer, Heidelberg, 2008, pp. 151–160.

[10] Melichar, B. and Polcar, T., "The Longest Common Subsequence Problem – A Finite Automata Approach," *Implementation and Application of Automata*, Springer, New York, 2003, pp. 294–296.

[11] Žďárek, J., "Automata and 2D Pattern Matching," *Advances on Two-dimensional Language Theory*, University of Salerno, Salerno, 2006, p. 15.

[12] Voráček, M., "Computing Covers in Generalized Strings," *POSTER 2005*, Czech Technical University in Prague, Faculty of Electrical Engineering, 2005.