Searching All Approximate Covers and Their Distance using Finite Automata

Ondřej Guth, Bořivoj Melichar, and Miroslav Balík

České vysoké učení technické v Praze, Praha, CZ,
{gutho1,melichar,balikm}@fel.cvut.cz

Abstract. *Cover is a type of a regularity of strings. A restricted approximate cover w of string T is a factor of T such that every position of T lies within some approximate occurrence of w in T. In this paper, the problem of* all restricted smallest distance approximate covers of a string *is studied and a polynomial time and space algorithm for solving the problem is presented. It searches for all restricted approximate covers of a string with given limited approximation using Hamming distance and it computes the smallest distance for each found cover. The solution is based on a finite 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.*

1 Introduction

Searching regularities of strings is used in a wide area of applications like molecular biology and computer–assisted music analysis. One of typical regularities is cover.

Finding exact covers is not sufficient in some applications, thus approximate covers have to be computed. In this paper, the Hamming distance is considered.

Exact covers were introduced in [1], an algorithm for computation of all exact covers in linear time was presented in [4]. An algorithm using finite automata approach to computation of all exact covers was introduced in [5].

The algorithm presented in [2] searches for one restricted smallest approximate cover (i.e. cover with the smallest distance), using dynamic programming. An algorithm using finite automata approach to computation all restricted approximate covers for Hamming, Levenshtein, and Damerau distance was introduced in [3].

This paper is organized as follows. In Section 2, some notations and definitions used in this paper are described. In Section 3, the algorithm for the problem is presented. In Section 4, the complexities of the algorithm are proven. In Section 5, experimental results are shown.

2 Preliminaries

An *alphabet* is a nonempty finite set of symbols, denoted by A. A *string* over an alphabet is a finite sequence of symbols of the alphabet. Empty string is an empty sequence of symbols, denoted by ε. An *effective alphabet* of a string T is a set of symbols that really occur in 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 $w[i]$. An operation *concatenation* is defined in this way: $x, y \in A^*$, concatenation of x and y is xy, 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. Suppose $u, w, x, T \in A^*$. w is a *prefix* of T if $T = wu$, w is a *suffix* of T if $T = uw$, and w is a *factor* (also called a substring) of T if $T = uwx$. A set of all prefixes of T is denoted by $Pref(T)$, a set of all suffixes of T is denoted by $Suff(T)$, and a set of all factors of T is denoted by $Fact(T)$.

A *deterministic finite automaton* (also called a deterministic finite state machine, denoted by DFA) is a quintuple (Q, A, δ, q_0, F), where Q is a nonempty finite set of states, A is an input alphabet, δ is a transition function, $\delta : Q \times A \mapsto Q$, $q_0 \in Q$ is an initial state and $F \subseteq Q$ is a set of final states.

A *nondeterministic finite automaton* without ε–transitions is a quintuple (Q, A, δ, q_0, F), where Q is a nonempty finite set of states, A is an input alphabet, δ is a transition function, where $\delta : Q \times A \mapsto \mathcal{P}(Q)$, $q_0 \in Q$ is an initial state and $F \subseteq Q$ is a set of final states. It is denoted by NFA.

A state q is a *successor* of state p of a deterministic finite automaton (Q, A, δ, q_0, F) if $q = \delta(p, a)$ for some $a \in A$. A state q_N is a successor of a state p_N of a NFA $(Q_N, A, \delta_N, q_{0N}, F_N)$ if $q \in \delta_N(p_N, a)$.

String $w = a_1 a_2 \dots a_{|w|}$ is said to be *accepted by a DFA* (Q, A, δ, q_0, F) if there exists a sequence $\delta(q_0, a_1) = q_1, \delta(q_1, a_2) = q_2, \dots, \delta(q_{|w|-1}, a_{|w|}) \in F$. String $w = a_1 a_2 \dots a_{|w|}$ is said to be *accepted by a NFA* (Q, A, δ, q_0, F) if there exists a sequence $\delta(q_0, a_1) = Q_1, \delta(q_1, a_2) = Q_2, \dots, \delta(q_{|w|-1}, a_{|w|}) \subseteq F$ for some $q_1 \in Q_1, \dots, q_{|w|-1} \in Q_{|w|-1}$. A language accepted by a finite automaton M is denoted by $L(M)$.

A *left language of a state* q of a nondeterministic finite automaton (Q, A, δ, q_0, F) is a set of strings $w = a_1 a_2 \dots a_{|w|}$, where for each w exists a sequence $\delta(q_0, a_1) = Q_1, \delta(q_1, a_2) = Q_2, \dots, \delta(q_{|w|-1}, a_{|w|}) = Q_{|w|}, q \in Q_{|w|}$ for some $q_1 \in Q_1, \dots, q_{|w|-1} \in Q_{|w|-1}$. A *left language of a state* q of a DFA (Q, A, δ, q_0, F) is a set of strings $w = a_1 a_2 \dots a_{|w|}$, where for each w exists a sequence
$\delta(q_0, a_1) = q_1, \delta(q_1, a_2) = q_2, \dots, \delta(q_{|w|-1}, a_{|w|}) = q$.

A *maxfactor of a state* q of a DFA (Q, A, δ, q_0, F) is the longest string of left language of q, denoted by *maxfactor*(q). A *depth of a state* q of a DFA is the length of *maxfactor*(q), denoted by *depth*(q).

A DFA $M_D = (Q, A, \delta, q_0, F)$ is *equivalent* to a NFA $M_N = (Q_N, A, \delta_N, q_{0N}, F_N)$ if $L(M_N) = L(M_D)$. Subset construction may be used:

1. Set $Q = \{\{q_0\}\}$ will be defined, state $q_0 = \{q_{0N}\}$ will be treated as unmarked.
2. If each state in Q is marked then continue with step 4.
3. Unmarked state q will be chosen from Q and the following operations will be executed:
   (a) $\delta(q, a) = \bigcup \delta_N(p_N, a)$ for $p_N \in q$ and for all $a \in A$,
   (b) $Q = Q \cup \delta(q, a)$ for all $a \in A$,
   (c) state $q \in Q$ will be marked,
   (d) continue with step 2.
4. $F = \{q : q \in Q, p_N \cap F_N \neq \emptyset, p_N \in q\}$.

Using subset construction of M_D equivalent to M_N, every state $q_D \in Q$ corresponds to some subset of Q_N. This subset is called a *d–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, as numeric representation of r_j corresponds to the depth. 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.

A *distance* is the minimum number of editing operations that are necessary to convert a string x into a string y. The maximum allowed distance is denoted by k.

The *Hamming distance* between strings x and y is equal to the minimum number of editing operations replace that are necessary to convert x into y. The Hamming distance function is denoted by D_H.

String $w \in A^*$ is an *approximate prefix* of a string $T \in A^*$ with the maximum Hamming distance k if there exists string $p \in \mathit{Pref}(T)$ such that $D_H(w, p) \leq k$. String w is an *approximate suffix* of the string T if there exists string $s \in \mathit{Suff}(T)$ such that $D_H(w, s) \leq k$.

A *nondeterministic Hamming suffix automaton* M for a string T and distance k is such nondeterministic finite automaton without ε–transitions, that $L(M) = \{w : D_H(w, s) \leq k, s \in \mathit{Suff}(T)\}$. Such an automaton $M = (Q, A, \delta, q_0, F)$ may be constructed in this way:

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

For example of a transition diagram of a nondeterministic Hamming suffix automaton see Fig. 1.

A *level* of a state of a nondeterministic Hamming suffix automaton corresponds to the number of errors, a *depth* of a state of this automaton is equal to the corresponding position in T.

Definition 1 (Restricted approximate cover). *Let T and w be strings. We say, that w is a* restricted approximate cover *of T with Hamming distance k if w is a factor of T and there exist strings $s_1, s_2, \dots, s_r$ (all some substrings of 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 $s_1, s_2, \dots, s_r$.*

Note 1. An approximate cover is more general regularity than restricted approximate cover, because (unrestricted) approximate cover of T needs not be a factor of T. In this paper, only restricted approximate cover is considered.

Definition 2 (Restricted smallest distance approximate cover). *Let T and w be strings. We say, that w is a* restricted smallest distance approximate cover *of T with distance k if w is a restricted approximate cover of T with the distance k and there exists no $l < k$ such that w is a restricted approximate cover of T with the distance l.*

Problem 1 (All restricted smallest distance approximate covers of a string). Given string T over alphabet A, Hamming distance function D_H and distance k, find all restricted approximate covers of T and their smallest distances. A set of all restricted smallest distance approximate covers of string T under Hamming distance k is denoted by $covers_{H^k}(T)$.

As any approximate cover of a string T under Hamming distance is an approximate prefix and an approximate suffix of T (proven in [3]), an automaton accepting only such strings can be used.

Definition 3 (Approximate cover candidate automaton). *An* approximate cover candidate automaton (Q, A, δ, q_0, F) *for string* $T \in A^*$*, Hamming distance function* D_H *and the maximum distance* k *accepts set* $W = \{w_1, w_2, \ldots, w_l\}$ *of factors of* T*, where for each* $w_i \in W$ *holds:*

1. *there exists* $p \in Pref(T)$ *such that* $D_H(p, w_i) \leq k$*, and*
2. *there exists* $s \in Suff(T)$ *such that* $D_H(s, w_i) \leq k$*.*

In [3], a construction of an automaton accepting intersection of approximate prefixes and approximate suffixes is used for construction of a deterministic approximate cover candidate automaton. Although this is a straightforward idea, specialized method (more effective) is presented for Hamming distance in the following section.

3 Problem solution

The principle of the solution is following: first, we perform a subset construction of a deterministic cover candidate automaton from a nondeterministic Hamming suffix automaton for string T and k, as every $d(q)$ represents a set of positions of $w = maxfactor(q)$ within T. If we treat with $d(q)$ as with a sorted list (ordered by depths of its elements), each pair of subsequent elements represents positions of subsequent occurrences of w within T. When for such positions $i, j, i < j$ holds $j - i > |w|$, we know that w cannot be a cover of T. The distance of w is the minimum l such that it is possible to remove all elements $r \in d(q)$ having $level(r) > l$ and the previous condition holds.

In fact, it is not necessary to save complete deterministic automaton. Unlike in [3], we do not make construction of the deterministic cover candidate automaton and subsequent computation of covering. A depth–first search algorithm is used to perform subset construction and computation of covering and of the distance of each cover: in Algorithm 2, for each state and symbol, a successor q is generated, it is determined whether it represents a cover and the distance is computed. When q represents an approximate prefix, its successors are recursively generated and processed. Note that the set of final states of the deterministic approximate cover candidate automaton is not needed (it would contain all states having d–subsets containing element corresponding to some final state of the nondeterministic Hamming suffix automaton).

Distance l of each cover $w = maxfactor(q)$ may vary between 0 and k. Moreover, it cannot be less than level of the first or the last element of $d(q)$, because each cover must be an approximate prefix and suffix. Of course, it cannot be more than the maximum level of elements of $d(q)$. The Algorithm 1 removes all the elements having the maximum level but the first and the last element of $d(q)$, and tries whether w covers T without those removed positions.

Algorithm 1 Smallest distance of a cover of T.

Input: d–subset $d(q)$ representing a cover w of T.
Output: The smallest distance l of w.

```
1: l_min ← max{level(r_1), level(r_|d(q)|)}
2: l_max ← max_{r∈d(q)}{level(r)}
3: l ← l_max
4: repeat
5:    for all r ∈ d(q) \ {r_1, r_|d(q)|} : level(r) = l do
6:       remove r from d(q)
7:    end for
8:    l ← l − 1
9: until l ≥ l_min and for all i = 2, 3, ..., |d(q)| : r_i − r_{i−1} ≤ depth(q)
10: l ← l + 1.
```

Example 1. Let us have a string $T = aabccccb$ over alphabet $A = \{a, b, c\}$ and let us compute a set of all restricted smallest distance approximate covers of T under Hamming distance $k = 2$ using Algorithm 3.

Because of the distance 2, we are interested in covers of length at least 3 or having distance less than 2. We construct a nondeterministic Hamming suffix automaton M_S (see Fig. 1), then an approximate cover candidate automaton M is analysed (see Fig. 2).

Looking at the d–subset $\{3, 4'', 8''\}$, it represents an approximate prefix and suffix aab of length 3, but for its positions holds $8 - 4 \not\leq 3$, thus the factor aab is not an approximate cover of T with Hamming distance 2. Looking at the other d–subset $\{3'', 5'', 6', 7', 8\}$, it represents factor ccb, that covers T with Hamming distance 2. It is checked whether it covers T with distance 1 (Alg. 1). As the first element of the d–subset has level equal to 2, $l_{\min}$ is equal to 2. The resulting set of the covers is $covers_{aabccccb^H}(2) = \{(ccb, 2), (aabccccb, 0)\}$.

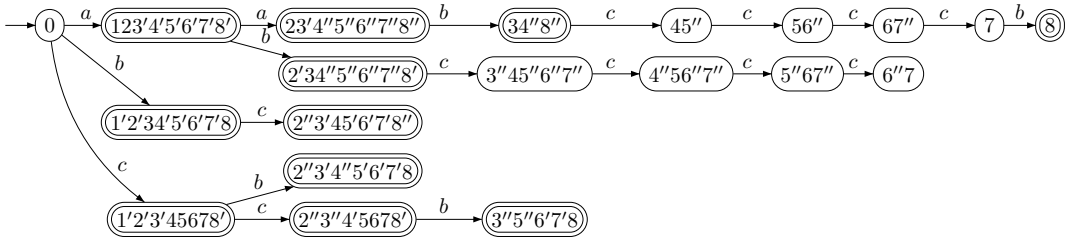

Fig. 2. Transition diagram of complete deterministic approximate cover candidate automaton for string $T = aabccccb$ and the maximum Hamming distance 2

Algorithm 2 Process state of a deterministic approximate cover candidate automaton $M = (Q, A, \delta, q_0, F)$ constructed for string T and the maximum distance k from a nondeterministic Hamming suffix automaton $M_S = (Q_S, A, \delta_S, q_{0S}, F_S)$.

Input: State q_i having depth i and the d–subset $d(q_i)$.
Output: The temporary set of restricted smallest distance approximate covers c.

$c \leftarrow \emptyset$
for all $a \in A$ **do**
  create new state q, define $depth(q) = depth(q_i) + 1$
  for all r_s in $d(q_i)$ (in order as stored in $d(q_i)$) **do**
    append all $r_i \in \delta_S(r_s, a)$ to $d(q)$ in ascending order by $depth(r_i)$
  end for
  if for the first $r_1 \in d(q)$ holds $r_1 \leq depth(q_i)$ **then**
    if exists $r \in d(q)$ where $level(r) = 0$ within M_S **then**
      define $w = maxfactor(q) = maxfactor(q_i).a$
      if $r_{|d(q)|} \in F_S$ **then**
        if for all $i = 2, 3, \ldots, |d(q)|$: $depth(r_i) - depth(r_{i-1}) \leq depth(q)$ **then**
          define l the smallest distance of w (Alg. 1)
          if $|w| > k$ or $l < |w|$ **then**
            $c \leftarrow c \cup (w, l)$
          end if
        end if
      end if
      process state q (this algorithm), c' is result
      $c \leftarrow c \cup c'$
    end if
  end if
end for

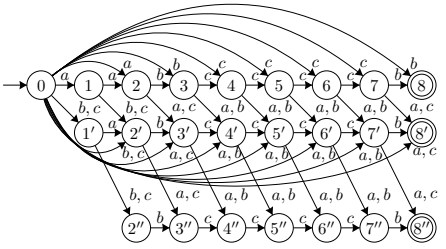

Fig. 1. Transition diagram of nondeterministic Hamming suffix automaton for string $aabccccb$ and the distance 2

Algorithm 3 Computation of a set of all restricted smallest distance approximate covers for string T and the Hamming distance k.

Input: String $T = a_1 a_2 \ldots a_n$, the Hamming distance k.
Output: Set of all restricted smallest distance approximate covers $covers_{H^k}(T)$ of string T using the Hamming distance function D_H and the distance k.

1: $covers_{H^k}(T) \leftarrow \{(T, 0)\}$.
2: Construct nondeterministic Hamming suffix automaton $M_S = (Q_S, A, \delta_S, q_{0S}, F_S)$ for T and k.
3: Create state q_0 of the deterministic approximate cover candidate automaton $M(T) = (Q, A, \delta, q_0, F)$.
4: Define $maxfactor(q_0) = \varepsilon$.
5: Process state q_0 using Algorithm 2.
6: $covers_{H^k}(T)$ is the resulting set from the previous step.

4 Complexities

Lemma 1. *The nondeterministic Hamming suffix automaton $M_S = (Q, A, \delta, q_0, F)$ for string T and the distance k contains $(|T|+1)\cdot(k+1) - \frac{k^2+k}{2}$ states and $|A| \cdot (|T| \cdot (k+1) - 1 + \frac{k-k^2}{2}) + |T| - k + 1$ transitions.*

Proof. The automaton consists of layers of states $q^{(i)}$ for each level i. The layer of states q^0 contains $|T| + 1$ states. Each layer of states $q^{(i)}$ contains one state less in comparison with layer of states $q^{(i-1)}$, thus it contains $|T| - i + 1$ and layer of states $q^{(k)}$ contains $|T| - k + 1$ states.

The automaton contains $|A|$ transitions from each state, with some exceptions. There are $k+1$ final states having no successor. In the layer of states $q^{(k)}$, each state has only one successor. From the initial state, there are $|T|$ transitions defined to the states $q^{(0)}$ having $level(q^{(0)}) = 0$ and $|T| \cdot (|A| - 1)$ transitions to the states $q^{(1)}$ having $level(q^{(1)}) = 1$. Thus in M_S there are $|Q| \cdot |A| + |T| \cdot |A| - (k+1) \cdot |A| - (|T| - k + 1) \cdot (|A| - 1) = |A| \cdot (|Q| - 2) + |T| - k + 1$ transitions.

Note 2. As restricted approximate covers of string T are exact factors of T, it is meaningful to consider effective alphabet A only, thus $|A| \leq |T|$ always holds.

It is also meaningless to consider large k, because every factor of T having length less or equal to k is always approximate cover of T. Thus $k \leq |T|$ always holds.

Usually, $k \ll |T|$ and $|A| \ll |T|$ (e.g. in DNA analysis, $A = \{a, c, g, t\}$). Therefore k and $|A|$ may be considered as small constants independent of $|T|$.

Lemma 2. *The deterministic approximate cover candidate automaton M for string T and the Hamming distance contains at most $\frac{|T|^2+|T|}{2} + 1$ states.*

Proof. Each d–subset $d(q)$ of M contains at least one r such that $level(r) = 0$, thus $maxfactor(q) \in Fact(T)$. The number of possible factors of length $depth(q)$ is at most $|T| - depth(q) + 1$, thus the maximum number of states of M having equal depth is also $|T| - depth(q) + 1$. The automaton M also contains an initial state. Therefore, the number of states of M is at most $\frac{(|T|-1+1)+(|T|-|T|+1)}{2} \cdot |T| + 1$.

Lemma 3. *During the construction of the deterministic cover candidate automaton M for string T, Algorithms 2, 3 need to hold at most $|T| + 2$ states at a time.*

Proof. Algorithm 2 works as a depth–first search algorithm. For each state and symbol it generates at most one state – possible successor. Thus it holds at most $|T| + 1$ states of M ($|T|$ states having d–subsets representing exact prefixes of T plus initial state) and a state generated for a final state, having empty d–subset.

Lemma 4. *During the construction of the deterministic cover candidate automaton M for string T, Algorithms 2, 3 need to hold at most $\frac{|T|^2+|T|}{2} + 1$ elements of d–subsets at a time.*

Proof. Alg. 2 needs at most $|T|+2$ states in a memory at a time (Lemma 3). The deterministic cover candidate automaton $M = (Q, A, \delta, q_0, F)$ is constructed by subset construction from a nondeterministic Hamming suffix automaton $M_S = (Q_S, A, \delta_S, q_{0S}, F_S)$. In M_S, each state but q_{0S} has at most one successor for each symbol, q_{0S} has $|T|$ successors for each symbol. For each state p_S and its successor q_S in M_S holds: $depth(q_S) > depth(p_S)$. The longest possible d–subset $d(p)$ contains $r_{|T|}$ having $depth(r_{|T|}) = |T|$, and r_1 having $depth(r_1) = 1$. As $|\delta_S(r_1, a)| \leq 1$ and $\delta_S(r_{|T|}, a) = \emptyset$ for every $a \in A$, for state p and its successor q in M holds: $|d(q)| \leq |d(p)|$ for $p \neq q_0$ and $|d(q)| \leq |T|$ for $p = q_0$.

Theorem 1. *Space complexity of Alg. 3 is $\mathcal{O}(|T|^2)$.*

Proof. It clearly holds that for construction of the nondeterministic Hamming suffix automaton $M_S = (Q_S, A, \delta_S, q_{0S}, F_S)$, there is no need for any additional data structures. For the purpose of the construction of the deterministic cover candidate automaton M, only the set of states and transitions from q_{0S} need to be preserved, because the rest may be computed later in $\mathcal{O}(1)$ time and space using knowledge of a depth and a level of a state, k, and T. Thus the space complexity of this construction is $\mathcal{O}((k + |A|) \cdot |T|)$.

During the computation of the smallest distance (Algorithm 1), only $\mathcal{O}(1)$ additional data is needed. During the processing of states of M (Algorithm 2), the needed space is limited by the number of elements of all d–subsets (Lemma 4) preserved in a memory and by the number of all approximate covers (the result, limited by the number of all factors of T – at most $\mathcal{O}(|T|^2)$).

Lemma 5. *Using Algorithms 2, and 3 for construction of a deterministic cover candidate automaton $M = (Q, A, \delta, q_0, F)$ from a nondeterministic Hamming suffix automaton $M_S = (Q_S, A, \delta_S, q_{0S}, F_S)$, all d–subsets are sorted in ascending order by depths within M_S.*

Proof. Having $p, q \in Q \setminus \{q_0\}$ such that q is a successor of p, suppose that $d(p)$ is sorted in order by depths within M_S. It holds that for any $p_S, q_S \in Q_S$ such that q_S is a successor of p_S, $depth(q_S) > depth(p_S)$. Therefore $d(q)$ constructed from already sorted $d(p)$ is also sorted.

For $p = q_0$, it is supposed that $\delta_S(q_{0S}, a)$ is constructed as sorted in order by depths within M_S.

Lemma 6. *Time complexity of Algorithm 1 is $\mathcal{O}(k \cdot |T|)$ for each state.*

Proof. Algorithm 1 may remove some elements of a d–subset in each iteration, thus the iteration may take $\mathcal{O}(|T|)$ time. The number of iterations may be at most k.

Lemma 7. *Time complexity of Algorithm 2 (from the initial state) is $\mathcal{O}((k + |A|) \cdot |T|^3)$.*

Proof. Algorithm 2 constructs for all states q and all $a \in A$ the d–subsets of all possible successors of q. The number of states is $\mathcal{O}(|T|^2)$ (Lemma 2) and the number of elements of each d–subset is $\mathcal{O}(|T|)$. For each state, the computation of covering is performed (it takes $\mathcal{O}(|T|)$), and for each cover (their number is $\mathcal{O}(T^2)$), the computation of the smallest distance is performed (it takes $\mathcal{O}(k \cdot |T|)$ for each cover – Lemma 6).

Theorem 2. *Time complexity of Alg. 3 is $\mathcal{O}((k+|A|) \cdot |T|^3)$.*

Proof. It clearly holds that construction of the nondeterministic Hamming suffix automaton takes $\mathcal{O}((k + |A|) \cdot |T|)$. Construction of the deterministic cover candidate automaton takes $\mathcal{O}((k+|A|) \cdot |T|^3)$ (Lemma 7).

5 Experimental results

The algorithm was implemented in C++ using STL, the program was compiled using the GNU C++ compiler 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 a AMD Athlon 64 3200+ (2200 MHz) system, with 2.5 GB of RAM, under Fedora Linux operating system (see Figs. 3, 4).

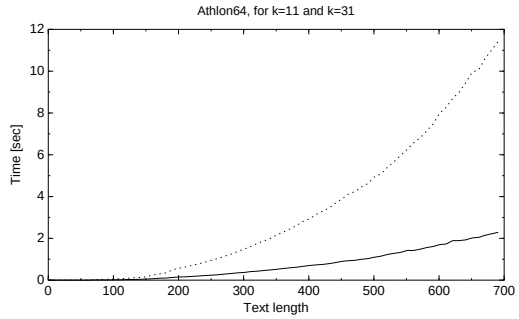

Fig. 3. Time consumption with respect to the text size (solid line for $k = 11$, dotted one for $k = 31$)

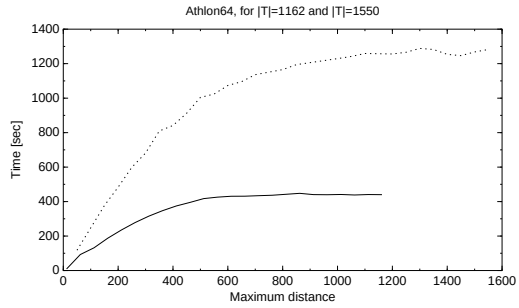

Fig. 4. Time consumption with respect to the distance (solid line for $|T| = 1162$, dotted one for $|T| = 1550$)

The second set of tests was run on a AMD Athlon (1400 MHz) system, with 1.2 GB of RAM, under Gentoo Linux operating system (see Figs. 5, 6).

Note 3. In comparison with experimental results presented in [2], the algorithm presented in this paper runs a bit faster for the same data, even on a slightly slower computer (1.3 seconds in [2] for text length 100 vs. maximum 1.0 second for text length 114 – see Fig. 6).

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

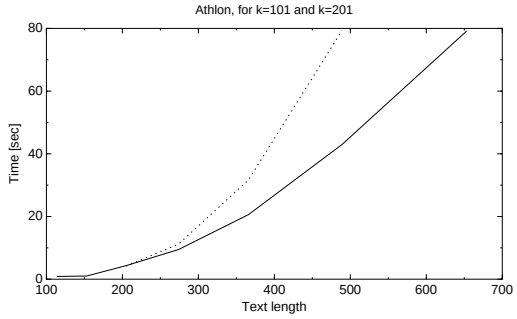

Fig. 5. Time consumption with respect to the text size (solid line for $k = 101$, dotted one for $k = 201$)

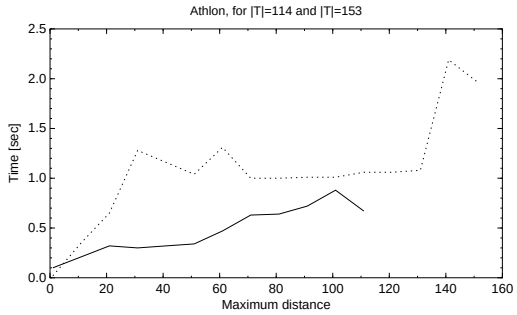

Fig. 6. Time consumption with respect to the maximum distance (solid line for $|T| = 114$, dotted one for $|T| = 153$)

6 Conclusion and future work

In this paper, we have shown that an algorithm design based on a determinisation of a suffix automaton is appropriate for all restricted smallest distance approximate covers of a string problem for 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 ([2]).

The algorithm may be extended to work with other distance functions, possibly using the idea presented in [3]. Theoretical and experimental analysis similar to one presented here may be accomplished. The algorithm may be also extended to use parallelism.

Acknowledgements

This research was partially supported by the Ministry of Education, Youth, and Sport of the Czech Republic under research program MSM 6840770014, by the

Czech Science Foundation as project No. 201/06/1039, and by the Czech Technical University in Prague as project No. CTU0803113.

References

1. Apostolico, A., Farach, M., and Iliopoulos, C. S. Optimal superprimitivity testing for strings. *Inf. Process. Lett. 39*, 1 (1991), 17–20.
2. Christodoulakis, M., Iliopoulos, C. S., Park, K., and Sim, J. S. Implementing approximate regularities. *Mathematical and Computer Modelling 42* (October 2005), 855–866.
3. Guth, O. Searching approximate covers of strings using finite automata. In *Proceedings of POSTER* (2008), Faculty of Electrical Engineering, Czech Technical University in Prague.
4. Smyth, W. F. Approximate periodicity in strings. *Utilitas Mathematica 51* (1997), 125–135.
5. Voráček, M., and Melichar, B. Searchig for regularities in generalized strings using finite automata. In *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics* (2005), WILEY – VCH Verlag, pp. 809–812.