Theoretical Informatics and Applications
Theoret. Informatics Appl. **35** (2001) 565–577

FORBIDDEN FACTORS AND FRAGMENT ASSEMBLY *

F. Mignosi[1], A. Restivo[1] and M. Sciortino[1]

Abstract. In this paper methods and results related to the notion of minimal forbidden words are applied to the fragment assembly problem. The fragment assembly problem can be formulated, in its simplest form, as follows: reconstruct a word w from a given set I of substrings (*fragments*) of a word w. We introduce an hypothesis involving the set of fragments I and the maximal length $m(w)$ of the minimal forbidden factors of w. Such hypothesis allows us to reconstruct uniquely the word w from the set I in linear time. We prove also that, if w is a word randomly generated by a memoryless source with identical symbol probabilities, $m(w)$ is logarithmic with respect to the size of w. This result shows that our reconstruction algorithm is suited to approach the above problem in several practical applications *e.g.* in the case of DNA sequences.

Mathematics Subject Classification. 68Q45, 68R15.

Introduction

Let w be a word over the alphabet A and let $L(w)$ denote the set of factors of w. A word v is a *minimal forbidden factor* of w if $v \notin L(w)$ and all proper factors of v belong to $L(w)$. Denote by $M(w)$ the set of minimal forbidden factors of the word w. The notion of minimal forbidden factors is a very basic one in combinatorics on words: some relevant information on the structure of a word w can be detected by looking at the set $M(w)$ (*cf.* [6,13]). This leads to some important applications in Data Compression (*cf.* [7]) and Symbolic Dynamics (*cf.* [2]).

Keywords and phrases: Factor automaton, minimal forbidden factor, fragment assembly.

* *Partially supported by MURST projects: "Modelli di calcolo innovativi: metodi sintattici e combinatori" and "Bioinformatica e Ricerca Genomica".*

[1] University of Palermo, Dipartimento di Matematica ed Applicazioni, Via Archirafi 34, 90123 Palermo, Italy; e-mail: {mignosi,restivo,mari}@math.unipa.it

In a previous paper (*cf.* [13]) we derived sharp upper and lower bounds for two parameters related to the size of $M(w)$: $c(w)$ that counts the minimal forbidden factors of w and $m(w)$ that gives the maximal length of minimal forbidden factors of w. Moreover we described a linear algorithm that reconstructs the word w from $M(w)$. Such an algorithm uses a close relation between $M(w)$ and the factor automaton of w, *i.e.* the minimal deterministic automaton recognizing $L(w)$.

Now we show that, in general, the size of words in $M(w)$ is "very small" with respect to the size of w: we prove that, for a word w randomly generated by a memoryless source with identical symbol probabilities, the maximal length $m(w)$ of words in $M(w)$ is $O(\log_d(|w|))$ where d is the cardinality of the alphabet.

These results lead to an interesting application to the fragment assembly problem (*cf.* [16,21]). In Section 5 we introduce the general problem and then consider a simplified version of the *fragment assembly problem* that can be formulated as follows: given a set I of substrings (*fragments*) of a word w, *i.e.* $I \subseteq L(w)$, reconstruct w from I. It is obvious that, without any additive hypothesis on I and w, it is not in general possible to infer w from I, indeed in general, given I, there exist several different words w compatible with I, *i.e.* such that $I \subseteq L(w)$. We further introduce an hypothesis concerning the set of fragments I and the size of the set of minimal forbidden words $M(w)$. Such hypothesis allows us to reconstruct uniquely the word w from the set I in linear time. The basic idea is the following. From the results on the size of $M(w)$, it is very reasonable to suppose that, in several practical applications (as for instance in the case of DNA sequences) the set of fragments I is such that any factor of w of length $m(w)$ is "covered" by at least one element of I. Under such hypothesis, one can detect the minimal forbidden factors of the whole word w only by looking at its "fragments" in I. By using the reconstruction algorithm one can recover in linear time the word w from its fragments. Further details on the assembly algorithm here described and a discussion on its adequacy to biological sequences can be found in [14], which also contains another linear-time assembly algorithm based on a more classic approach. In [4], by using the notions of special and univalent factor, an assembly algorithm is proposed, which presents some analogies with that ones of the present paper. A preliminary version of this paper without proofs has been presented in DLT'01 (*cf.* [12]).

1. Preliminaries

Let A be a finite alphabet and let A^* be the set of finite words over the alphabet A, the empty word ϵ included. Let $L \subseteq A^*$ be a *factorial language*, *i.e.* a language satisfying $\forall u, v \in A^*,\ uv \in L \Rightarrow u, v \in L$. From an algebraic point of view we observe that the complement language $L^c = A^* \setminus L$ is a two-sided ideal of the free monoid A^*. Denote by $M(L)$ the base of this ideal, *i.e.* $L^c = A^*M(L)A^*$. The set $M(L)$ is called the set of *minimal forbidden words* for L. A word $v \in A^*$ is *forbidden* for the factorial language L if $v \notin L$, which is equivalent to say that v occurs in no word of L. In addition, v is *minimal* if it has no proper factor that is

forbidden. From the minimality of its words follows that $M(L)$ is an *antifactorial language*, *i.e.* $\forall u, v \in M(L), u \neq v \Rightarrow u$ is not a factor of v.

Remark 1.1. One can note that the set $M(L)$ uniquely characterizes L, just because

$$L = A^* \setminus A^* M(L) A^*. \tag{1}$$

Conversely the following remark shows that L also uniquely characterizes $M(L)$.

Remark 1.2. A word $v = a_1 a_2 \cdots a_n$ belongs to $M(L)$ if and only if the two conditions hold:

- v is forbidden, (*i.e.* $v \notin L$);
- both $a_1 a_2 \cdots a_{n-1} \in L$ and $a_2 a_3 \cdots a_n \in L$ (the prefix and the suffix of v of length $n-1$ belong to L).

Hence we have that

$$M(L) = AL \cap LA \cap (A^* \setminus L). \tag{2}$$

From the equalities (1) and (2) it follows that $M(L)$ uniquely characterizes L and L uniquely characterizes $M(L)$ respectively. Recall that a language $L \subset A^*$ is *rational* if it is recognized by a finite state automaton. From the equalities (1) and (2), one also derives that L and $M(L)$ are simultaneously rational, *i.e.* L is rational if and only if $M(L)$ is a rational language. By using the classical algorithms of automata theory, in the case L is rational, one can effectively construct $M(L)$ from L, and conversely.

In this paper we are interested in the language $L(w)$ of factors of a single word w and consequently on the set $M(L(w))$, which is here denoted simply by $M(w)$ and is called the set of *minimal forbidden factors* of the word w.

Example 1.3. Let us consider the word $w = acbcabcbc$ on the alphabet $\{a, b, c\}$. One has that

$$M(w) = \{aa, ba, bb, cc, aca, cac, cbcb, abca, bcbca\}.$$

It is obvious that $M(w)$ is a finite set uniquely characterizing the word w.

2. $M(w)$ and Factor automaton

In this section we report some results of [6] showing that there is a close relation between the set $M(w)$ and the *factor automaton* of the word w, *i.e.* the minimal deterministic automaton recognizing $L(w)$.

Given the antifactorial set $M(w)$, the finite automaton $\mathcal{A}(w) = (Q, A, i, T, F)$ is defined, where

- the set Q of states is $\{v \mid v \text{ is a prefix of a word in } M(w)\}$;
- A is the current alphabet;

- the initial state i is the empty word ϵ;
- the set T of terminal states is $Q \setminus M(w)$.

States of $\mathcal{A}(w)$ that are words of $M(w)$ are sink states. The set F of transitions is partitioned into the three (pairwise disjoint) sets F_1, F_2, and F_3 defined by:

- $F_1 = \{(u, a, ua) \mid ua \in Q, a \in A\}$ (forward edges or tree edges);
- $F_2 = \{(u, a, v) \mid u \in Q \setminus M(w), a \in A, ua \notin Q, v$ longest suffix of ua in $Q\}$ (backward edges);
- $F_3 = \{(u, a, u) \mid u \in M(w), a \in A\}$ (loops on sink states).

Theorem 2.1. *For any $w \in A^*$, the automaton obtained from $\mathcal{A}(w)$ by removing its sink states is the factor automaton of w (i.e. the minimal deterministic finite automaton $\mathcal{F}(w)$ accepting the language $L(w)$).*

Note that, as it is showed in [6], the previous construction is possible even for an antifactorial language M and in this case it provides an automaton $\mathcal{A}(M)$ recognizing the corresponding language L. However, in the general case, $\mathcal{A}(M)$ is not minimal. In [6] the above definition of $\mathcal{A}(M)$ is turned into an algorithm, called L-AUTOMATON that builds the automaton from a finite antifactorial set of words. The input is the trie $\mathcal{T}$ representing M. The procedure can be adapted to test whether $\mathcal{T}$ represents an antifactorial set, or even to generate the trie of the antifactorial language associated with a set of words. In [6] it is proved that this algorithm runs in time $O(|Q| \times |A|)$, where Q and A are respectively the set of states and the alphabet of the input trie where the transition functions are implemented by transition matrices.

Recall also that there are some interesting results (*cf.* [3, 5]) about the size of factor automaton of a word w. Indeed, by denoting with Q and E the set of states and edges respectively, it is proved that the size of $\mathcal{F}(w)$ is linear with the length of the word w. In particular, if $|w| \leq 2$ then $|Q| = |w|+1$ and $|w| \leq |E| \leq 2|w|-1$. If $|w| \geq 3$ then $|w| + 1 \leq |Q| \leq 2|w| - 2$ and $|w| \leq |E| \leq 3|w| - 4$. These bounds will be useful in next sections.

3. On the size of $M(w)$

In this section we are interested in a valuation of the size of $M(w)$. Given a finite word w, we consider the following parameters:

$$c(w) = \mathrm{Card}(M(w))$$

$$m(w) = \max\{|v|, v \in M(w)\}\cdot$$

Example 1.3. (*continued*) Let $w = acbcabcbc$. We have $c(w) = 9$, $m(w) = 5$.

We report (*cf.* [13]) two results on the bounds for these parameters. The first one states in particular an upper bound of $c(w)$, which linearly depends on the length of the word w. Remark that the cardinality of $L(w)$, the set of factors of w, is $O(|w|^2)$. Denote by d the cardinality of the alphabet A and by $d(w)$

the number of the letters of A occurring in w, *i.e.* $d(w) = \mathrm{Card}(\mathrm{alph}(w))$, where $\mathrm{alph}(w)$ denotes the set of letters occurring in w.

Theorem 3.1. *Let $w = a_1 \dots a_n$ be a finite word over the alphabet A. The following inequalities hold:*

$$d \leq c(w) \leq (d(w) - 1)(|w| - 1) + d.$$

Moreover previous inequalities are sharp.

The next theorem gives lower and upper bounds on the parameter $m(w)$. We consider $d > 1$, because if A has just one element, it is trivial that $m(w) = |w| + 1$. Recall that, for any real number α, $\lceil \alpha \rceil$ denotes the smallest integer greater than or equal to α.

Theorem 3.2. *Let w be a finite word over the alphabet A with at least two elements. The following inequalities hold:*

$$\lceil \log_d(|w| + 1) \rceil \leq m(w) \leq |w| + 1.$$

Furthermore the bounds are actually attained.

We now evaluate this parameter for a word w which is randomly generated by a memoryless source with identical symbol probabilities. For simplicity we consider sources over a binary alphabet. All the results presented here can be easily generalized to alphabets of more letters.

Remark 3.3. It is easy to prove that $k < m(w) - 1$ if and only if there exists a factor of the word w of length k having at least two occurrences.

Theorem 3.4. *Let w be a word over a binary alphabet which is randomly generated by a memoryless source with identical symbol probabilities. The probability that there exists a word v of length k that appears at least twice as factor of w is smaller than $(n-k+1)(k-1)2^{-k} + (n-2k+1)^2 2^{-k}$, where n is the length of w.*

Proof. Let us evaluate the probability that a word $v = a_1 a_2 \dots a_k$ of length k appears at least twice as factor of w. We consider two cases. Case 1: the position of the second occurrence of v in w appears at a distance smaller than k from the position of the first occurrence. Case 2: the position of the second occurrence of v in w appears at a distance greater than or equal to k from the position of the first occurrence. The reason of considering the second case is evident. Since the source is memoryless, the probability of the event of a second occurrence of v in Case 2, is independent of the fact that there was a first occurrence of v, and this allows an easy computation of the probability. In the first case instead, the last letters of v must coincide with the first letters of v, *i.e.* the word v self-overlaps and must be periodic with period smaller than or equal to k (*cf.* [11]). Let us analyze now the first case. Suppose now that a word v of length k appears at position i and that it has its second occurrence at position $i + 1$. Therefore it must be either the word 1^k or the word 0^k, *i.e.* just one symbol repeated k times. The probability

that this event happens at position i is the sum of the probability that in position i there are $k+1$ consecutive zeroes or ones, *i.e.* $2^{-(k+1)} + 2^{-(k+1)} = 2^{-k}$. Suppose now that a word v of length k appears at position i and that it has its second occurrence at position $i+2$. Word v cannot be the word 1^k nor the word 0^k, because in this case the second occurrence would have appeared earlier. Therefore it must be either the word $(10)^{k/2}$ or the word $(01)^{k/2}$. The probability that this event happens at position i is the sum of the probabilities that at position i there are $k+2$ consecutive letters from the sequence $101010\ldots$, *i.e.* the probability is $2^{-(k+2)} + 2^{-(k+2)} = 2^{-(k+1)}$. If we ask the same as above at position i and $i+3$ we get 6 words, each of them has probability 2^{-k}, and the event has probability for each of them $2^{-(k+3)}$. For positions i and $i+j$ with $j \leq k-1$ we have in general that v could be one word among at most 2^j words (the one having as smallest period the number j), and for each of them the event has probability $2^{-(k+j)}$, for a global probability of at most 2^{-k}. Adding up all those probabilities for j running from 1 to $k-1$ we obtain that Case 1 happens when the first position of v is i with probability smaller than $(k-1)2^{-k}$. The overall probability of Case 1 is smaller than $(n-k+1)(k-1)2^{-k}$, where n is the number of symbols in the string w. A simple exercise shows that the overall probability of Case 2 is smaller than $(n-2k+1)^2 2^{-k}$, and this concludes the proof. □

From Remark 3.3 and previous theorem we have the following:

Corollary 3.5. *The probability that $m(w)$ is no more than $3\log_2 n + 1$ is about 1 for large enough values of n.*

For a memoryless source where the symbols have fixed probabilities different from 1/2, analogous computation can be done. From previous corollary easily follows that, for a word w randomly generated by a memoryless source, the parameter $m(w)$ is $O(\log_d(n))$ where n is the length of w and d is the cardinality of the alphabet.

4. Reconstruction algorithm

In a previous paper (*cf.* [13]) a linear algorithm that reconstructs the word w from the set $M(w)$ is proposed. Such an algorithm uses a close relation between $M(w)$ and factor automaton $\mathcal{F}(w)$ of w (*cf.* [6] and also Sect. 2) and the linear size of $\mathcal{F}(w)$ (*cf.* [5] and also Sect. 2). Now we report this algorithm because it represents the core of the Assembly algorithm described in Section 5. The algorithm involves, besides factor automaton, another known construction that is topological sort of a directed acyclic graph (*cf.* [1]). The main procedures used in the algorithm are L-automaton (*cf.* [6] and see also Sect. 2), TopologicalSort (*cf.* [1]) and Buildword. We will give a description of the algorithm, dwelling upon the procedure Buildword.

Procedure L-AUTOMATON takes as input the trie of the set $M(w)$ and returns a complete deterministic automaton accepting $L(w)$. Recall that from this automaton it is possible, by removing its sink states, to obtain the minimal deterministic automaton $\mathcal{F}(w)$ (see Th. 2.1).

Procedure BUILDWORD, that is related to the search for the longest path in a directed acyclic graph, works as follows: it first calls Procedure TOPOLOGICALSORT that produces a linear ordering of all vertices of the transition graph $\mathcal{G}(w)$ of the factor automaton $\mathcal{F}(w)$ by using a depth first search procedure. If $\mathcal{G}(w)$ contains an edge (q, p), then q appears before p in the ordering. Recall that the transition graph of a factor automaton is a directed acyclic graph (if the graph is not acyclic, then no linear ordering is possible).

Then in Procedure BUILDWORD we create the *precedence-lists* B of the factor automaton $\mathcal{F}(w) = (Q, A, q_0, T, \delta)$. If n is the number of states of $\mathcal{F}(w)$ then B consists of an array of n lists, one for each state. For each state q the *precedence-list* $B(q)$ contains (pointers to) all states s such that $\delta(s, a) = q$ for some $a \in A$:

$$B(q) = \{s|\ \exists a \in A \text{ such that } \delta(s, a) = q\}\cdot$$

Moreover we define a function $\pi : Q \mapsto \mathbb{N}$ such that, for each state q, $\pi(q)$ is the length of the longest path from q_0 to q in the transition graph $\mathcal{G}(w)$ of the factor automaton $\mathcal{F}(w)$. A recursive definition of the function π is the following:

$$\begin{aligned} \pi(q_0) &= 0 \\ \pi(q) &= \max\{\pi(s)|s \in B(q)\} + 1 \end{aligned}$$

where q_0 is the initial state. Remark that, if $s \in B(q)$ then $s < q$ in the topological sort.

We also remark that the factor automaton $\mathcal{F}(w)$ satisfies the property that if x and y are respectively the label of two paths from state q to the state p, then $|x| \neq |y|$. From this property it follows that, for any $q \in Q \setminus \{q_0\}$, there exists a unique state $s_q \in B(q)$ such that $\pi(s_q) = \max\{\pi(s)|s \in B(q)\}$. Moreover, according to the previous property, we can define the following partial function $l : Q \times Q \mapsto A$ such that

$$l(p, q) = a \quad \text{if} \quad \delta(p, a) = q.$$

Finally, if $q = q_{n-1}$ is the last state in the linear ordering produced by Procedure TOPOLOGICALSORT, the word is built by taking the letter $l(s_q, q)$, by setting q equal to s_q and updating it by concatenating on the left the letter $l(s_q, q)$. This cycle has to be carried out while q is different from the initial state.

```
BUILDWORD (factor automaton F(w) = (Q, A, i, T, δ))
1.  {q_0, q_1, ..., q_{n-1}} ← TOPOLOGICALSORT(G(w));
2.  for each state q_i, 0 ≤ i ≤ n − 1, create the precedence-list B(q_i);
3.  π(q_0) ← 0;
        for each state q_i, 1 ≤ i ≤ n − 1
            Find the maximum m_i of the set {π(s), s ∈ B(q_i)};
            return the unique state s_{q_i} such that π(s_{q_i}) = m_i;
            π(q_i) ← m_i + 1;
4.  w ← ε;
5.  q ← q_{n-1};
6.  while q ≠ q_0 do
        a ← l(s_q, q);
        w ← aw;
        q ← s_q;
7.  return w;
```

The following proposition, by using the fact that $\sum_{i=0}^{n-1} |B(q_i)| = |E|$, establishes an upper bound on the time required for Procedure BUILDWORD.

Proposition 4.1. *The execution time of Procedure* BUILDWORD *is* $O(|Q| + |E|)$, *where* Q *and* E *are respectively the set of the states and the set of the edges of factor automaton* $\mathcal{F}(w)$ *of a word* w *over a fixed alphabet* A.

```
w-RECONSTRUCTION (Trie T(w) representing the set M(w))
1.  A(w) ← L-AUTOMATON(T(w));
2.  F(w) ← removing sink states of A(w);
3.  w ← BUILDWORD(F(w));
4.  return w;
```

From Proposition 4.1, from linear size of factor automaton and since L-AUTOMATON procedure runs in time $O(|Q| \times |A|)$, we can conclude that:

Proposition 4.2. *Given the set* $M(w)$ *of minimal forbidden factors of a word* w *over a fixed alphabet* A, *it is possible to reconstruct* w *in linear time with respect to the size of the trie representing the set* $M(w)$.

5. Applications to fragment assembly

The *fragment assembly problem* or *sequence reconstruction problem* (*cf.* [16,21]) consists in finding, given a set of words $\mathcal{F}$ (called the *fragment set*) and an error rate $\epsilon \in [0, 1)$, a word w such that for all $f_i \in \mathcal{F}$ there is a factor v of w such that

$$\max_i \{\min\{d(v, f_i), d(v, f_i^r)\}\} \leq \epsilon |v|,$$

where d is the edit distance (*cf.* [15]) and f_i^r is the Watson–Crick complement of the fragment f_i (*cf.* [16]).

Classically the set of fragments is obtained by a shotgun sequencing approach, in which a random sampling of short factors of a DNA segment is acquired. This technique must account for the following essential characteristic of the data (*cf.* [16]) that are *incomplete coverage*, *sequencing errors*, *unknown orientation*. Almost all computational architectures for fragment assembly follow a general outline first formalized in [18] and [17] (*cf.* [9], Sect. 16.15). That outline contains three discrete steps: *overlap detection*, *fragment layout* and *deciding the consensus*. The third step is also called *final multiple alignment* (*cf.* [21], Sect. 7.1.3). While shotgun sequencing infers a DNA sequence given the sequences of overlapping fragments, a complementary method, called sequencing by hybridization (SBH), infers a DNA sequence given the set of oligomers that represents all factors of some fixed length k. Such an approach can be recast as an Eulerian path problem (*cf.* [8, 20] and references therein). However, since experimental reality keeps k small and implies that the data will be error prone, most sequencing is done using variations of the method of the shutgun approach. In [10] and [19] are proposed algorithms for sequence assembly that combine techniques of both shotgun and SBH methods.

In this paper we restrict our attention to the case where the error rate is equal to zero, the coverage is complete and the orientation of the sequences is known. We feel confident that we will be able to use variations of our algorithm to settle the general case. Therefore in this paper the fragment assembly problem can be formulated as follows: given a set I of substrings (*fragments*) of a word w, *i.e.* $I \subseteq L(w)$, reconstruct w from I. It is obvious that, without any additive hypothesis on I and w, it is not in general possible to infer w from I, indeed in general, given I, there exist several different words w compatible with I, *i.e.* such that $I \subseteq L(w)$. In this paper we introduce an hypothesis concerning the set of fragments I and the size of the set of minimal forbidden words $M(w)$. Such hypothesis allows us to reconstruct uniquely the word w from the set I. The basic idea of the algorithm is the following. In Section 4 we showed that, given the set $M(w)$, it is possible to reconstruct in linear time the word w. We can obtain this set from the set of minimal forbidden factors of another word w_1 obtained by concatenating all fragments in I. Indeed, if the set of fragments is such that any factor of w of length $m(w)$ is "covered" by at least one element of I, we show that one can obtain all minimal forbidden factors of the original word w from those of w_1. Actually the estimate of the parameter $m(w)$ in previous section shows that the minimal forbidden factors are "short enough" with respect to the length of w. So the previous hypothesis on fragments is appropriate in some practical applications. We broach now these arguments more rigorously.

Definition 5.1. A set of substrings I of a word w is a *k-cover* for w if every substring of length k of w is a substring of at least one string in I. The covering index of I, denoted $C(I)$, is the largest value of k such that I is a k-cover for w.

Clearly, I is a k-cover for w for all $k \leq C(I)$. We can now state the following theorem that will be proved in the following. Recall that we use the notation $||I||$ to denote the sum of the lengths of all words in I.

Theorem 5.2. *Let w be a word over a fixed alphabet A and let I be a set of substrings of w such that*

$$m(w) \leq C(I).$$

Then the word w can be uniquely reconstructed by the set I and this reconstruction can be done in linear time $O(||I||)$.

We note that in the previous theorem we do not need to know the length of the string w to recover it. The "linear time" has to be considered under the usual standard assumption that we can store, add and compare integers in constant time. Note also that the condition $m(w) \leq C(I)$ in previous theorem is tight in the sense that, if there exists a substring v that occurs in w at least twice and such that v never appears as a proper factor of any element of I, then the reconstruction can be ambiguous, *i.e.* there could exist several words w compatible with I.

In this section we present an algorithm, called ASSEMBLY, that, under the hypothesis of Theorem 5.2, solves the fragment assembly problem in linear time.

In the first step the following easy CONCAT algorithm is used. Its input is the set of fragments I and a symbol \$ that is not in the alphabet A of all fragments in I. The output is a word w_1 over the alphabet $A \cup \{\$\}$ that is the concatenation of all strings in I interspersed with the symbol \$, *i.e.* between two consecutive strings there is one symbol \$. The set I is structured as a simple stack. The operation of concatenation between words is denoted by a simple dot "." and the empty word is denoted by ε.

```
CONCAT (set I, symbol $)
1.  w1 ← ε;
2.  while I ≠ ∅
3.      extract v from I;
4.      w1 ← w1.v.$;
5.  return (w1);
```

The second goal of the ASSEMBLY algorithm consists in the construction of the trie of the minimal forbidden factors of w_1 having length smaller than or equal to $m(w)$ and not containing the symbol \$. We first construct the factor automaton $\mathcal{F}(w_1)$ of the word w_1 by using the FACTORAUTOMATON algorithm given in [5]. Recall that such a construction can be implemented to work on the input word w_1 in time $O(|w_1| \times \log |A|)$ within $O(|w_1|)$ space if one uses adjacency lists. It produces also a *suffix function* that is a failure function defined on the states of the automaton. Then we use the CREATE_TRIE algorithm that is a light variation of the MF-TRIE algorithm described in [6], where the trie of all minimal forbidden factors of a word was obtained by its factor automaton. The input of CREATE_TRIE algorithm is the factor automaton $\mathcal{F}(w_1)$ of word w_1. It includes the suffix function produced by the FACTORAUTOMATON algorithm (*cf.* [5]). The time-complexity of CREATE_TRIE algorithm, as in the case of MF-TRIE, is $O(|w_1| \times |A|)$ on input $\mathcal{F}(w_1)$ if transition functions are implemented by transition matrices. The result is a consequence of the linear size of $\mathcal{F}(w_1)$. Moreover the correctness of the

CREATE_TRIE algorithm is a simple consequence of the correctness of MF-TRIE algorithm, together with the fact, stated in next Proposition 5.3, that the set of minimal forbidden factors of original string w is exactly the set of words that are found by CREATE_TRIE algorithm. Note that the hypothesis of next proposition is essential to its proof.

Proposition 5.3. *Let w be a word over a fixed alphabet A and let I be a set of substrings of w such that*

$$m(w) \leq C(I).$$

Then the set of minimal forbidden factors of the word w is exactly the set of all the minimal forbidden factors of w_1 that do not contain the symbol \$ and that have length smaller than or equal to $m(w)$, i.e.

$$M(w) = M(w_1) \cap A^{\leq m(w)}.$$

Proof. Any minimal forbidden substring $m = avb$ of w_1 of length at most $m(w)$ that does not contain the symbol \$ is also a minimal forbidden substring of w. Indeed av and vb are substrings of w_1 and consequently of w. The word $m = avb$ is not a substring of w_1. Since $m(w) \leq C(I)$, it is a forbidden substring of w too, otherwise it would have appeared in w_1. Hence $m = avb$ is a minimal forbidden substring of w.

Conversely suppose that $m = avb$ is a minimal forbidden substring of w, where a and b are letters. The substrings av and vb have length at most $m(w) - 1$. As $C(I) > m(w) - 1$ each of them must appear in some position of w_1. The substring $m = avb$ cannot be a substring of w_1 and the thesis follows. □

```
CREATETRIE (factor automaton F(w1) = (Q, A ∪ {$}, i, T, δ),
                suffix function h)
1.  for each state p ∈ Q in breadth-first search from i and each a ∈ A
2.      if δ(p, a) undefined and (p = i or δ(h(p), a) defined)
3.          δ'(p, a) ← new sink;
4.      else
5.          if δ(p, a) = q and q is distant from i more than p
6.              δ'(p, a) ← q;
7.  In a depth-first search with respect to δ' prune all branches of
    trie T(w) not ending in a state that is sink and has depth
    smaller than or equal to m(w);
8.  return T(w) = (Q', A, i, {sinks}, δ');
```

Recall that the suffix function h is defined as follows. Let $u \in (A \cup \{\$\})^+$ and $p = \delta(i, u)$. Then $h(p) = \delta(i, u')$ where u' is the longest suffix of u for which $\delta(i, u) \neq \delta(i, u')$.

Note that, at line 6, not all the states of $\mathcal{F}(w_1)$ are reachable starting from the root i, because δ' does not represent edges labeled by \$. All states and edges that are not reachable from i are implicitly pruned in $\mathcal{T}(w)$. The implicit and explicit

pruning operations at line 6 obviously change the set of states, the set of sinks and the function δ'.

Remark 5.4. It is easy to see that the elements of $M(w_1) \cap A^*$ that do not belong to $M(w)$ (*i.e.* having length greater than $m(w)$) are of the form avb, where $av\$$ and $\$vb$ are factors of the word w_1. By using this remark it is possible to modify the CREATE_TRIE algorithm in order to obtain the elements of $M(w)$, without the explicit knowledge of the value of $m(w)$.

The final step of algorithm ASSEMBLY consists in recovering the word w from $\mathcal{T}(w)$. This is done by w-RECONSTRUCTION algorithm showed in Section 4 (*cf.* also [13]).

The overall ASSEMBLY algorithm is thus

```
ASSEMBLY (set of fragments I)
1.  w_1 ← CONCAT (set I, $);
2.  F(w_1) = (Q, A ∪ {$}, i, T, δ) ← FACTORAUTOMATON (w_1);
3.  T(w) = (Q', A, i, {sinks}, δ') ← CREATETRIE (F(w_1), h);
4.  w ← w-RECONSTRUCTION (T(w));
5.  return w;
```

From Propositions 5.3 and from the linear time complexity of all the procedures used in the previous algorithm the proof of Theorem 5.2 follows.

REFERENCES

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms*. Addison Wesley, Reading, Mass (1983).

[2] M.-P. Béal, F. Mignosi, A. Restivo and M. Sciortino, Forbidden Words in Symbolic Dynamics. *Adv. in Appl. Math.* **25** (2000) 163-193.

[3] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen and J. Seiferas, The smallest automaton recognizing the subwords of a text. *Theoret. Comput. Sci.* **40** (1985) 31-55.

[4] A. Carpi, A. de Luca and S. Varricchio, Words, univalent factors, and boxes. *Acta Inform.* (to appear).

[5] M. Crochemore and C. Hancart, Automata for matching patterns, in *Handbook of Formal Languages*, Vol. 2, Chap. 9, edited by G. Rosenberg and A. Salomaan. Springer (1997) 399-462.

[6] M. Crochemore, F. Mignosi and A. Restivo, Automata and forbidden words. *Inform. Process. Lett.* **67** (1998) 111-117.

[7] M. Crochemore, F. Mignosi, A. Restivo and S. Salemi, Data compression using antidictionaries, in *Proc. of the IEEE, Special Issue on Lossless Data Compression*, Vol. 88, edited by J.A. Storer (2000) 1756-1768.

[8] A. Frieze and B.V. Halldórsson, Optimal Sequencing by Hybridization in Rounds, in *Proc. of RECOMB 2001*, edited by T. Lengauer, D. Sankoff, S. Istrail, P. Pevzner and M. Waterman. ACM Press (2001) 141-148

[9] D. Gusfield, *Algorithms on strings, trees, and sequences: Computer science and computational biology*. Cambridge University Press (1997).

[10] R. Idury and M. Waterman, A new algorithm for DNA sequence assembly. *J. Comput. Biol.* **2** (1995) 291-306.

[11] F. Mignosi and A. Restivo, Periodicity, in *M. Lothaire, Algebraic Combinatorics on Words*, Chap. 8. Cambridge University Press (to appear) 237-274. Also available at url: `http://www-igm.univ-mlv.fr/~berstel/Lothaire/index.html`

[12] F. Mignosi, A. Restivo and M. Sciortino, Forbidden Factors and Fragment Assembly. *Lecture Notes in Comput. Sci.* (2001). Proceedings of DLT'01.

[13] F. Mignosi, A. Restivo and M. Sciortino, Words and Forbidden Factors. *Theoret. Comput. Sci.* **273** (2002) 99-117.

[14] F. Mignosi, A. Restivo, M. Sciortino and J. Storer, *On Sequence Assembly*, Technical Report cs-00-210. Brandeis University (2000).

[15] S. Muthukrishnan and S.C. Sahinalp, Approximate nearest neighbors and sequence comparison with block operations. ACM Press (2000). *Proceedings of STOC 2000*.

[16] G. Myers, Whole-Genome DNA Sequencing, *IEEE Comput. Engrg. Sci.* **3** (1999) 33-43.

[17] H. Peltola, H. Soderlund and E. Ukkonen, SEQAID: A DNA Sequence Assembly Program Based on a Mathematical Model. *Nucl. Acids Res.* **12** (1984) 307-321.

[18] M. Peltola, H. Soderlund, J. Tarhio and E. Ukkonen, Algorithms for some string matching problems arising in molecular genetics, in *Proc. of the 9th IFIP World Computer Congress* (1983) 59-64.

[19] P.A. Pevzner, H. Tang and M. Waterman, A New Approach Fragment Assembly in DNA Sequencing, in *Proc. of RECOMB 2001*, edited by T. Lengauer, D. Sankoff, S. Istrail, P. Pevzner and M. Waterman. ACM Press (2001) 141-148.

[20] R. Shamir and D. Tsur, Large Scale Sequencing by Hybridization, in *Proc. of RECOMB 2001*, edited by T. Lengauer, D. Sankoff, S. Istrail, P. Pevzner and M. Waterman. ACM Press (2001) 269-278.

[21] M.S. Waterman, *Introduction to computational biology: Maps, sequences and genomes*. Chapman & Hall (1995).

Received June 26, 2001. Revised February 21, 2002.