Theoretical Computer Science 462 (2012) 1–11

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Variable length local decoding and alignment-free sequence comparison

Gilles Didier [a,*], Eduardo Corel [b], Ivan Laprevotte [c], Alex Grossmann [c], Claudine Landès-Devauchelle [c]

[a] Institut de Mathématiques de Luminy, CNRS FRE 3529, Aix-Marseille Université 13288 Marseille Cedex 9, France

[b] Institut für Mikrobiologie und Genetik, Georg-August Universität, 37077 Göttingen, Germany

[c] Laboratoire Statistique et Génome, CNRS UMR 8071, Université d'Evry-Val-d'Essonne, 91037 Evry, France

ARTICLE INFO

Article history:
Received 26 January 2012
Received in revised form 27 July 2012
Accepted 12 August 2012
Communicated by M. Crochemore

Keywords:
Coding
Prefix code
Algorithm
Genetic sequences comparison

ABSTRACT

We present the variable length local decoding, a method which augments the alphabet of a sequence or a set of sequences. Roughly speaking, the approach distinguishes several types of symbols/nucleotides according to their contexts in the sequences. These contexts have variable lengths and are defined from a prefix code.

We first give an original algorithm computing the decoding with a complexity linear both in time and memory space. Next, the approach is applied to alignment-free sequence comparison. We give a heuristic way to select context lengths relevant to this question. The comparison of sequences itself is based on the composition in "augmented" symbols of their variable length local decodings. The results of this comparison are illustrated on a biological alignment.

1. Introduction

In [3,5], we introduced and studied the N-local decoding, which has the effect of turning a sequence (or a set of sequences) into a new sequence over an alphabet with a greater variety of symbols. Briefly summarized, if one associates an identifier to each subword of length N occurring in a sequence s, then an *N-block coding* of s is simply the sequence of identifiers of the successive overlapping subwords of length N running along s. In general, two sequences can differ, even up to a letter-to-letter bijection, and yet have a same N-block coding. However we showed that, given an N-block coding, there is a "maximal antecedent", from which any sequence having the same N-block coding can be obtained by a letter-to-letter map, in other words by identifying some of its symbols. The *N-local decoding* of s is defined to be the maximal antecedent of the N-block coding of s. In [5], we gave an algorithm of linear complexity depending only on the length of the sequence (not on the block length N) to compute it. The decoding was next applied to genetic sequence analysis, notably to alignment-free comparison [4].

The variable length local decoding, first introduced in [1] where it is called "C-escritura" (*C-writing* in Spanish), is a natural extension of the N-local decoding. "Variable length" stands for that, rather than considering the N-block coding of s, we start from a prefix code $\mathcal{P}$ (a set of words in which no element is prefix of another) and define the $\mathcal{P}$-coding of s as the sequence of identifiers of the successive overlapping elements of $\mathcal{P}$ running along s. In [1,2], it is shown that, given a $\mathcal{P}$-coding c and the lengths of the elements of $\mathcal{P}$ corresponding to all the identifiers, there exists again a maximal antecedent, that similarly generates, by letter-to-letter maps, all the sequences that have c as a $\mathcal{P}'$-coding, for any prefix code $\mathcal{P}'$ whose identifiers/elements have lengths which are consistent with those of $\mathcal{P}$. The variable length local decoding of a sequence s relatively to a prefix code $\mathcal{P}$ is then similarly defined as the maximal antecedent of its $\mathcal{P}$-coding.

* Corresponding author. Tel.: +33 4 91 26 96 14; fax: +33 4 91 26 96 55.
E-mail addresses: gilles.didier@univ-amu.fr (G. Didier), eduardo.corel@genopole.cnrs.fr (E. Corel), ivan.laprevotte@genopole.cnrs.fr (I. Laprevotte), alex.grossmann@genopole.cnrs.fr (A. Grossmann), claudine.devauchelle@genopole.cnrs.fr (C. Landès-Devauchelle).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.08.005

In the first part of the present work, we present an original algorithm computing the variable length local decoding of a (set of) sequence(s) in a time and using a memory space both linear with its length. This algorithm can be seen as a generalization of the one introduced in [5].

The second part is devoted to the application of the variable length decoding to the sequence comparison problem [12]. A preliminary step is the development of a heuristic selection of a prefix code adapted to the sequence comparison problem. The resulting variable length decoding is then used to compute a basic dissimilarity based on the composition of sequences in terms of local decoded symbols.

This approach is illustrated on the reference alignment of the Hepatitis C Virus Database Project [8]. We first show that our dissimilarity is well correlated with the identity percentage computed from the alignment and, next, that it is perfectly consistent with the known virus typing and subtyping.

The paper is organized as follows. In Section 2 are presented notations, definitions and results needed by the computation of the variable length decoding. Section 3 is for the formal presentation of the algorithm, its proof and its complexity. In Section 4, we show how to apply the decoding to alignment-free sequences comparison and illustrate the accuracy of the results obtained on a biological alignment. Finally, we discuss the method and present some directions of work in Section 5.

2. Notations, definitions and preliminary results

For a set $\mathcal{A}$, the notation $\#\mathcal{A}$ stands for the cardinality of $\mathcal{A}$.

A sequence s is indexed from 0 to $|s|-1$, where $|s|$ denotes the length of s. We denote $s_{[i,j]}$ the sequence $s_i s_{i+1} \dots s_j$ if $i \leq j$ and the empty word ϵ otherwise.

Prefix has the usual meaning. A prefix of a sequence s is said to be *proper* if it is different of s.

A *prefix code* $\mathcal{P}$ is a set of words satisfying the "prefix property", that is: no word of $\mathcal{P}$ is a (proper) prefix of another word of $\mathcal{P}$. Let s be a sequence which we assume ending with a special symbol not occurring elsewhere in s. We say that a prefix code $\mathcal{P}$ is *s-compliant* if, for all positions i of s, there exists an element w of $\mathcal{P}$ occurring at i. In what follows, s denotes a sequence ending with a special character and $\mathcal{P}$ an s-compliant prefix code. Coding the sequence s by the s-compliant prefix code $\mathcal{P}$ means to associate to each position i of s, the unique (thanks to the prefix property) element of $\mathcal{P}$ occurring at i, which is written $\mathbf{c}(i)$.

Definition 1. The relation $\sim$ is defined for all pairs (i, j) of positions of s by $i \sim j$ if there exists an integer $\ell \geq 0$ such that $\mathbf{c}(i-\ell) = \mathbf{c}(j-\ell)$ and $|\mathbf{c}(i-\ell)| > \ell$.

We write $\approx$ for the transitive closure of $\sim$.

The relation $\approx$ is basically an equivalence relation (under the assumption that $\mathcal{P}$ is s-compliant) and we put Δ for the corresponding partition of positions of s.

The variable length local decoding with regard to $\mathcal{P}$ consists of assigning a same symbol to two positions i and j if and only if they are in relation by $\approx$. In other words, the variable length decoding d of s is, up to a letter-to-letter bijection, the sequence defined by $d_i = d_j \Leftrightarrow i \approx j$.

To informally explain why the sequence d is also called "maximal antecedent" (more formal statements can be found in [2]), let us first note that we have $i \approx j \Rightarrow s_i = s_j$, which means that $d_i = d_j \Rightarrow s_i = s_j$. It follows first that, since $\mathcal{P}$ is a prefix code, the set of subwords $\mathcal{P}' = \{d_{[i,i+\mathbf{c}(i)-1]} \mid 0 \leq i < |s|\}$ of d is also a prefix code. Moreover, the definition of d ensures that for all elements w of $\mathcal{P}$ and all pairs (i, j) of positions of occurrences of w in s, we have $d_{i+k} = d_{j+k}$ for all $0 \leq k < |w|$. This means that the sequence that one obtains by coding d following $\mathcal{P}'$ is, up to a letter-to-letter bijection, the coding of s following $\mathcal{P}$. This makes the variable length local decoding an antecedent of this last coding. Finally, the fact that $d_i = d_j \Rightarrow s_i = s_j$ implies the existence of a letter-to-letter map from d to s, and thus gives sense to the adjective "maximal".

Definition 2. For all positive integers k, the binary relation $\overset{k}{\smile}$ on the positions of s is defined for all pairs (i, j) of positions of s by $i \overset{k}{\smile} j$ if at least one of the following assertions holds:

1. $i = j$,
2. there exists an integer $\ell \geq 0$ such that $\mathbf{c}(i-\ell) = \mathbf{c}(j-\ell)$ and $|\mathbf{c}(i-\ell)| - \ell \geq k$.

We write $\overset{k}{\backsim}$ for the transitive closure of $\overset{k}{\smile}$.

The following remark follows from the transitivity or by direct application of Definition 2.

Remark 1. Let (i, j) be a pair of positions of s and k a positive integer.

1. $i \approx j$ if and only if $i \overset{1}{\backsim} j$;
2. if $k > \max_{w \in \mathcal{P}} |w|$ then we have $i \overset{k}{\backsim} j \Leftrightarrow i = j$;
3. if $i \overset{k+1}{\backsim} j$ then $i \overset{k}{\backsim} j$;
4. if $i \overset{k+1}{\backsim} j$ then $i + 1 \overset{k}{\backsim} j + 1$;

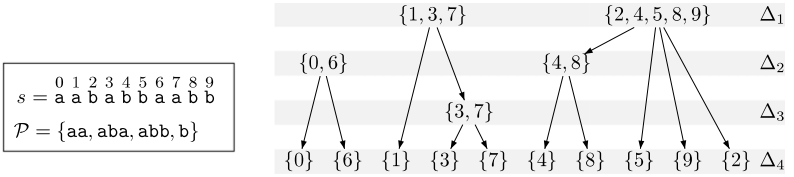

Fig. 1. A sequence s with an s-compliant prefix code $\mathcal{P}$ (left) and the corresponding tree representation of $\bigcup_{k>0} \Delta_k$ (right). Except for singletons, each atom/node is displayed at the row corresponding to the partition of largest index where it occurs. An atom/node also belongs to the partitions of smaller indices until it coalesces with another atom/node. For instance, {0, 6} belongs to both Δ_2 and Δ_1, {3, 7} belongs to both Δ_3 and Δ_2 and {5} belongs to Δ_k for all $k \geq 2$.

5. if $i \overset{k}{\smile} j$ then $s_{[i,i+k-1]} = s_{[j,j+k-1]}$;
6. $i \overset{|\mathbf{c}(i)|}{\smile} j$ if and only if $\mathbf{c}(j) = \mathbf{c}(i)$.

For all positive integers k, the relation $\overset{k}{\smile}$ is an equivalence relation. We put Δ_k for the corresponding partition of positions of s.

With Definition 2 and Item 3 of Remark 1, we get that if $k < k'$ then Δ_k is coarser than $\Delta_{k'}$. In other words, all the partitions Δ_k are nested and can be represented as a set of trees where nodes are all the atoms in $\bigcup_{k>0} \Delta_k$, an atom/node α being a child of an atom/node γ if $\gamma \neq \alpha$ is the smallest atom such that $\alpha \subset \gamma$. In particular, the leaves are the atoms of $\Delta_{\max_{w\in\mathcal{P}}|w|+1}$ (singletons of positions of s) and the roots are the atoms of Δ_1 (Fig. 1). Basically, $\bigcup_{-1\leq k<N} \Delta_k$ and the corresponding tree structures have less than $2|s|$ elements.

We distinguish the subset of *k-special atoms* $\nabla_k \subset \Delta_k$ defined by $\delta \in \nabla_k$ if there exist two positions i and j in δ such that $(i+1) \overset{k}{\not\smile} (j+1)$. For the example of Fig. 1, we have $\nabla_3 = \{\{3, 7\}\}$, $\nabla_2 = \{\{0, 6\}, \{4, 8\}\}$ and $\nabla_1 = \{\{2, 4, 5, 8, 9\}\}$.

We also distinguish the subset $\Gamma_k \subseteq \Delta_k$ of *k-code atoms* defined by $\delta \in \Gamma_k$ if there exists an element w of $\mathcal{P}$ such that $|w| = k$ and $\delta = \{i \mid \mathbf{c}(i) = w\}$. If an element w of $\mathcal{P}$ occurs in s, then there exists an atom $\delta \in \Delta_{|w|}$ such that $\delta = \{i \mid \mathbf{c}(i) = w\}$ (Items 5 and 6 of Remark 1). In Fig. 1, we have $\Gamma_3 = \{\{1\}, \{3, 7\}\}$, $\Gamma_2 = \{\{0, 6\}\}$, $\Gamma_1 = \{\{2, 4, 5, 8, 9\}\}$

An atom γ of Δ_k is called *k-specific* if it does not belong to Δ_{k+1}. The set of k-specific atoms is noted Θ_k. Note that a singleton cannot be k-specific, whatever the value of k. For the example of Fig. 1, we have $\Theta_3 = \{\{3, 7\}\}$, $\Theta_2 = \{\{0, 6\}, \{4, 8\}\}$ and $\Theta_1 = \{\{1, 3, 7\}, \{2, 4, 5, 8, 9\}\}$.

For all positive integers k and all atoms $\gamma \in \Delta_{k+1}$, there exists a unique $\delta \in \Delta_k$ such that $i + 1 \in \delta$ holds for all positions $i \in \gamma$ (*cf.* Item 4 of Remark 1). This atom δ is called the *k-follower* of γ and denoted $\mathrm{F}_k(\gamma)$. Note that F_k is a map from Δ_{k+1} to Δ_k.

Remark 2. Let k be a positive integer. If β is an atom of Δ_{k+2} included in an atom α of Δ_{k+1}, then the atom $\mathrm{F}_{k+1}(\beta)$ is included in $\mathrm{F}_k(\alpha)$. Conversely, if δ is an atom of Δ_{k+1} included in an atom γ of Δ_k then all atoms α of Δ_{k+1} containing an atom of $\mathrm{F}_{k+1}^{-1}(\delta)$ satisfy $\mathrm{F}_k(\alpha) = \gamma$.

Remark 3. Let k be a positive integer. If an atom δ is k-special then it is k-specific.

Proof. If $\delta \in \Delta_k$ is not k-specific then we have $i \overset{k+1}{\smile} j$ for all pairs (i, j) of δ. With Item 4 of Remark 1, it follows that $i+1 \overset{k}{\smile} j+1$ for all pairs (i, j) of atom δ which, by definition, cannot be k-special. □

Proposition 1. *Let k be a positive integer. If an atom δ is $(k+1)$-special then $\mathrm{F}_k(\delta)$ is k-specific. Conversely, if an atom γ is k-specific then we have $\gamma \in \Gamma_k$ or $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1} \neq \emptyset$.*

Proof. The first assertion follows from the definitions of $(k+1)$-special and k-specific atoms.

Conversely, since γ is k-specific, there exist two positions i and j in $\gamma \in \Delta_k$ (*i.e.* $i \overset{k}{\smile} j$) such that $i \overset{k+1}{\not\smile} j$. From $i \overset{k}{\smile} j$, there exists a sequence of positions of $\gamma, x_1, x_2, \ldots, x_n$ such that $x_1 = i, x_n = j$ and $x_m \overset{k}{\smile} x_{m+1}$ for all $1 \leq m < n$. But for $i \overset{k+1}{\not\smile} j$, there is at least an integer $1 \leq o < n$ such that $x_o \overset{k+1}{\not\smile} x_{o+1}$. Here are two possibilities: either $(x_o - 1) \overset{k+1}{\smile} (x_{o+1} - 1)$ or not. In the first case and by definition, the atom of Δ_{k+1} containing $(x_o - 1)$ and $(x_{o+1} - 1)$ both belong to $\mathrm{F}_k^{-1}(\gamma)$ and to ∇_{k+1}. In the second case, we have $(x_o - 1) \overset{k+1}{\not\smile} (x_{o+1} - 1)$, $x_o \overset{k+1}{\not\smile} x_{o+1}$ and $x_o \overset{k}{\smile} x_{o+1}$. By Definition 2, this is possible only if $\mathbf{c}(x_o) = \mathbf{c}(x_{o+1})$ and $|\mathbf{c}(x_o)| = k$. Items 5 and 6 of Remark 1 then ensure that $\gamma \in \Gamma_k$. □

Proposition 2. *Let k be a positive integer. If an atom γ is k-specific without being a k-code atom then, for all $\delta \in \Delta_{k+1}$ subset of γ, there exist $\beta \in \mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1}$ and $\alpha \in \Delta_{k+2}$ subset of β, such that $\mathrm{F}_{k+1}(\alpha) = \delta$.*

Proof. The fact that γ is k-specific and $\delta \in \Delta_{k+1}$ is a subset of γ implies the existence of a position $i \in \delta$ and position $j \in \gamma$ such that $i \overset{k}{\smile} j$ and $i \overset{k+1}{\not\smile} j$. In the same way as in the proof of Proposition 1, there exists a sequence of positions of $\gamma, x_1, x_2, \ldots, x_n$ such that $x_1 = i, x_n = j$ and $x_m \overset{k}{\smile} x_{m+1}$ for all $1 \leq m < n$. Let p be the smallest integer $1 \leq p < n$ such that $x_p \overset{k+1}{\not\smile} x_{p+1}$ (the existence of p follows from $i \overset{k+1}{\not\smile} j$). We then have $x_p \in \delta$ and $x_{p+1} \notin \delta$.

Since both x_p and x_{p+1} belong to γ, which is not a k-code atom, this is only possible if $(x_p - 1) \overset{k+1}{\smile} (x_{p+1} - 1)$. Therefore the atom β of Δ_{k+1} containing $(x_o - 1)$ and $(x_{o+1} - 1)$ belongs both to $\mathrm{F}_k^{-1}(\gamma)$ and to ∇_{k+1}. Remark 3 ensures that β is $(k+1)$-specific. If α is the atom of Δ_{k+2} which contains the position $(x_o - 1)$, we have $\alpha \subset \beta$ and $\mathrm{F}_{k+1}(\alpha) = \delta$. □

Remark 4. Let k be a positive integer and i be a position of s. If $\{i\} \notin \Delta_k$ then there exists a position $q \leq i$ and a position $p \neq q$ such that $|\mathbf{c}(q)| \geq i - q + k$ and $q \overset{\mathbf{c}(q)}{\smile} p$.

Proof. The singleton $\{i\}$ does not belong to Δ_k if and only if there exists a position $h \neq i$ of s such that $i \overset{k}{\smile} h$. Definition 2 gives us the result. □

Lemma 1. *Let i be a position of s with $i < |s| - 1$. If there exists a positive integer k such that $\{i\} \in \Delta_{k+1}$ and $\{i+1\} \notin \Delta_k$ then, for all $1 \leq \ell \leq k$, there exists a position $j > i$ such that $\{j\} \notin \Delta_\ell$, $\{j\} \in \Delta_{\ell+1}$ and $\{p\} \notin \Delta_{\ell+1}$ for all positions $i < p < j$.*

Proof. Let us assume that we have $\{i\} \in \Delta_{k+1}$ and $\{i+1\} \notin \Delta_k$. From Items 2 and 3 of Remark 1, the fact that $\{i+1\} \notin \Delta_k$ implies that there exists a position $h \neq i+1$ such that $i+1 \overset{k}{\smile} h$. Since $\{i\} \in \Delta_{k+1}$, we have $i \overset{k+1}{\not\smile} h - 1$. By Definition 2, the only possibility is that $\mathbf{c}(i+1) = \mathbf{c}(h)$ and $|\mathbf{c}(i+1)| \geq k$.

Let ℓ be a positive integer smaller than k. In addition to the fact that $|\mathbf{c}(i+1)| \geq k$, this ensures that we have $i < i + |\mathbf{c}(i+1)| - \ell + 1 < |s|$. In other words, $(i + |\mathbf{c}(i+1)| - \ell + 1)$ is indeed a position of $|s|$ strictly greater than i. From $\mathbf{c}(i+1) = \mathbf{c}(h)$ with $|\mathbf{c}(i+1)| \geq k$ and by iterating Items 3 and 4 of Remark 1, we get that $\{p\} \notin \Delta_{\ell+1}$ for all $i < p < i + |\mathbf{c}(i+1)| - \ell + 1$ and $\{i + |\mathbf{c}(i+1)| - \ell + 1\} \notin \Delta_\ell$. We now put $j = i + |\mathbf{c}(i+1)| - \ell + 1$. If $\{j\} \in \Delta_{\ell+1}$, the lemma is proved. Otherwise, according to Remark 4, there exists a position $q \leq j$ such that $|\mathbf{c}(q)| \geq j - q - \ell + 1$ (for $\{i\} \in \Delta_{k+1}$, we have also $i < q$). With the same argument as above, we get that $\{p\} \notin \Delta_{\ell+1}$ for all $q \leq p \leq q + \mathbf{c}(q) - \ell - 1$ and $\{q + \mathbf{c}(q) - \ell\} \notin \Delta_\ell$. Since $\ell > 0$, $q + \mathbf{c}(q) - \ell$ is indeed a position of s. Moreover we have $q + \mathbf{c}(q) - \ell > j$ and $\{p\} \notin \Delta_{\ell+1}$ for all $i < p \leq q + \mathbf{c}(q) - \ell - 1$. Let us replace j by $q + \mathbf{c}(q) - \ell$. Once again, if $\{j\} \in \Delta_{\ell+1}$, we are done. Otherwise, we use Remark 4 to iterate the same steps and replace j with a greater position while keeping the property that $\{j\} \notin \Delta_\ell$ and $\{p\} \notin \Delta_{\ell+1}$ for all $i < p < j$. Since the sequence s is finite, we eventually obtain a position $j > i$ such that $\{j\} \in \Delta_{\ell+1}$, $\{j\} \notin \Delta_\ell$ and $\{p\} \notin \Delta_{\ell+1}$ for all $i < p < j$. □

Proposition 3. *The number of pairs (i, k), where i is a position of s and k a positive integer, such that $\{i\} \in \Delta_{k+1}$ and $\mathrm{F}_k(\{i\})$ contains more than one position, is smaller than $|s|$.*

Proof. Let (i, k) be such that $\{i\} \in \Delta_{k+1}$ and $\mathrm{F}_k(\{i\})$ contains more than one position, which implies that $\{i+1\} \notin \Delta_k$. Lemma 1 gives us the existence of a position $j > i$ such that $\{j\} \notin \Delta_k$, $\{j\} \in \Delta_{k+1}$ and $\{p\} \notin \Delta_{k+1}$ for all positions $i < p \leq j$. Let (i', k') be a pair position-integer such that $i' < j$, $\{i'\} \in \Delta_{k'}$, $\{j\} \in \Delta_{k'+1} \backslash \Delta_{k'}$ and $\{p\} \notin \Delta_{k'+1}$ for all positions $i' < p < j$. By Item 3 of Remark 1, the fact that $\{j\} \in \Delta_{k+1} \backslash \Delta_k$ and $\{j\} \in \Delta_{k'+1} \backslash \Delta_{k'}$ implies that $k' = k$. The positions i and i' are now both defined as the greatest position h smaller than j such that $\{h\} \in \Delta_{k+1}$. It follows that $i = i'$. Finally, we associate in an injective manner, a position j of s to all pairs (i, k) such that $\{i\} \in \Delta_k$ and $\mathrm{F}_k(\{i\})$ contains more than one position. This proves that the number of such pairs is smaller than $|s|$. □

Corollary 1. *The number of pairs (i, k), where i is a position of s and k a positive integer, satisfying $\{i\} \in \Delta_{k+1}$ and $\mathrm{F}_k(\{i\}) \neq \mathrm{F}_{k+1}(\{i\})$, is smaller than $|s|$.*

Proof. It suffices to note that if $\mathrm{F}_k(\{i\}) \neq \mathrm{F}_{k+1}(\{i\})$ then $\mathrm{F}_k(\{i\})$ contains more than one position and to apply Proposition 3. □

Proposition 4. *The number of pairs (α, k), where α is an atom of $\cup_{\ell>0}\Delta_\ell$ and k a positive integer, such that α contains more than one position, $\alpha \in \Delta_{k+1} \cap \Delta_k$ and $\mathrm{F}_k(\alpha) \neq \mathrm{F}_{k+1}(\alpha)$, is smaller than $2|s|$.*

Proof. Let (α, k) be such that $\#\alpha > 1$, $\alpha \in \Delta_{k+1} \cap \Delta_k$ and $\mathrm{F}_k(\{\alpha\}) \neq \mathrm{F}_{k+1}(\alpha)$. We associate to the pair (α, k), a finite sequence $(\beta_p^\alpha)_{0 \leq p \leq m_\alpha}$ of atoms of $\cup_{\ell>0}\Delta_\ell$ constructed by setting $\beta_0^\alpha = \alpha$ and by iteratively determining the next terms of the sequence in the following way. Let us assume that β_p^α belongs to $\Delta_{k+p+1} \cap \Delta_{k+p}$ and that $\#\beta_p^\alpha > 1$ (these assumptions are granted for $\beta_0^\alpha = \alpha$). We pick an atom β_{p+1}^α in $\mathrm{F}_{k+p}^{-1}(\beta_p^\alpha) \subset \Delta_{k+p+1}$. The assumption that β_p^α contains more than one position ensures that $\mathrm{F}_{k+p}^{-1}(\beta_p^\alpha) \neq \emptyset$: the only atom whose inverse image by F_{k+p} (if defined) could be empty, would be $\{0\}$. Here are three mutually exclusive possibilities:

1. $\beta_{p+1}^\alpha \in \Delta_{k+p+2}$ and $\#\beta_{p+1}^\alpha > 1$,
2. $\beta_{p+1}^\alpha \in \Delta_{k+p+2}$ and $\#\beta_{p+1}^\alpha = 1$,
3. $\beta_{p+1}^\alpha \notin \Delta_{k+p+2}(\alpha)$.

In the first case, we increment p and iterate. In the second and third cases, we stop, set $m_\alpha = p + 1$ and associate the pair $(\beta_{m_\alpha}^\alpha, k + m_\alpha)$ to the pair (α, k). Note that in the first case, the assumptions $\beta_p^\alpha \in \Delta_{k+p+1} \cap \Delta_{k+p}$ and $\#\beta_p^\alpha > 1$ hold for the incremented p. Since p increases at each step, the iteration eventually stops (Item 2 of Remark 1).

The process above allows us to associate to all pairs (α, k) such that α is an atom of $\cup_{\ell>0}\Delta_\ell$ and k a positive integer, satisfying $\#\alpha > 1$, $\alpha \in \Delta_{k+1} \cap \Delta_k$ and $\mathrm{F}_k(\alpha) \neq \mathrm{F}_{k+1}(\alpha)$, a pair $(\beta^\alpha_{m_\alpha}, k + m_\alpha)$ satisfying exactly one of the following properties:

- $\beta^\alpha_{m_\alpha} \in \Delta_{k+m_\alpha}(\alpha)$, $\#\beta^\alpha_{m_\alpha+1} = 1$ and $\mathrm{F}_{k+m_\alpha-1}(\beta^\alpha_{m_\alpha}) = \beta^\alpha_{m_\alpha-1}$ contains more than one position (Type 1 pairs),
- $\beta^\alpha_{m_\alpha}$ is $(k + m_\alpha)$-specific (Type 2 pairs).

With Proposition 3, we get that the number of Type 1 pairs is smaller than $|s|$. Item 3 of Remark 1 makes sure that to each atom γ is corresponding (at most) a unique integer ℓ such that γ is ℓ-specific. Type 2 pairs are thus bounded by the number of atoms of $\cup_{\ell>0}\Delta_\ell$, which is smaller than $|s|$.

Let us now consider a pair (δ, ℓ) – of any type – obtained by the previous process. This means that there exists (at least) a pair (α, k) such that $\#\alpha > 1$, $\alpha \in \Delta_{k+1} \cap \Delta_k$ and $\mathrm{F}_k(\alpha) \neq \mathrm{F}_{k+1}(\alpha)$, from which we can construct a sequence $(\beta^\alpha_p)_{0\leq p\leq m_\alpha}$ with $\delta = \beta^\alpha_p$ and $\ell = k + m_\alpha$. The way in which this sequence is defined ensures that we have $\beta^\alpha_p = \mathrm{F}_{k+p}(\beta^\alpha_{p+1})$, $\beta^\alpha_p \in \Delta_{k+p+2} \cap \Delta_{k+p+1}$ and $\#\beta^\alpha_p > 1$ for all $0 \leq p < m_\alpha$. Define the *followers path* $(\gamma_p)_{0\leq p<\ell}$ of (δ, ℓ) by $\gamma_0 = \delta$ and $\gamma_p = \mathrm{F}_{\ell-p}(\gamma_{p-1})$ for all $0 < p < \ell$. We have $\gamma_p \in \Delta_{\ell-p} \cap \Delta_{\ell-p}$ for all $0 < p \leq m_\alpha$, and m_α is the smallest integer p satisfying $\mathrm{F}_p(\gamma_p) \neq \mathrm{F}_{p+1}(\gamma_p)$. It follows that (α, k) is the unique pair to which can be associated (δ, ℓ) by the process above.

In conclusion, we injectively associate to all pairs satisfying the assumptions of the proposition, a pair which is taken among a set containing less than $2|s|$ elements. This proves our bound. □

3. Algorithm

The main algorithm, Algorithm 1, iteratively constructs the partitions Δ_k, starting from $\Delta_{\max_{w\in\mathcal{P}}|w|+1}$, which is made of singletons corresponding to all the positions of s, to the partition $\Delta_1 = \Delta$, in which each atom contains the set of occurrence positions of a symbol in the local decoding.

It takes as inputs:

- a table W containing the elements of $\mathcal{P}$ sorted following the decreasing order of their lengths (in particular $|\mathrm{W}[0]| = \max_{w\in\mathcal{P}}|w|$),
- the sets $(\mathcal{O}_w)_{w\in\mathcal{P}}$, where $\mathcal{O}_w$ contains all the occurrence positions of w.

Under our basic assumptions on $\mathcal{P}$ (for instance that all its elements occur in s), these inputs occupy a memory space, and can be constructed in a time, which are both linear with $|s|$. Indeed, the elements of $\mathcal{P}$ can be ordered using a counting sort and sets $(\mathcal{O}_w)_{w\in\mathcal{P}}$ can be obtained from a suffix tree.

Our algorithm benefits from the fact that partitions are nested, which allows us to represent all atoms α in $\cup_{k>0}\Delta_k$ as a node belonging to a family of trees, where an atom α is the direct ancestor of an atom β if it is the smallest atom containing β. The atoms/nodes of the subtree rooted at an atom/node α are exactly those included in α. We introduce the notations used to deal with these tree structures. For a node α, $\mathrm{P}(\alpha)$ denotes its direct ancestor or parent (`Null` if it doesn't have one) and $\mathrm{C}(\alpha)$, the set of children of α. Note that each time the parent of α is set to a node β (*i.e.* $\mathrm{P}(\alpha) \leftarrow \beta$), α is implicitly added to the set of children of β. This means that $\mathrm{C}(\beta) \leftarrow \mathrm{C}(\beta) \cup \{\alpha\}$ is performed but not explicit in Algorithm 1. The function `new_node` is a basic primitive which allocates and returns a new node with no child and a `Null` parent.

Algorithm 1 also deals with the sets of atoms defined in the preceding section: Θ_k and ∇_k (k-specific and k-special atoms). We also consider the set $\overline{\nabla}_k$ of k-specific atoms which are not k-special, that is $\overline{\nabla}_k = \Theta_k \setminus \nabla_k$. Beside their inclusion tree structure, atoms/nodes α are also linked by their k-followers $\mathrm{F}_k(\alpha)$ (the follower of a node returned by `new_node` is initialized to `Null`). Again, setting the k-follower of a node α to a node β (*i.e.* $\mathrm{F}_k(\alpha) \leftarrow \beta$) implicitly adds α to the set $\mathrm{F}_k^{-1}(\beta)$.

In order to make the algorithm clearer and consistent with the notations of the preceding section, we keep indexing by k or $k + 1$ the sets of atoms as well as the follower links. Actually, the algorithms are still accurate if one removes all the indices. This is clear for the sets ∇_k, $\overline{\nabla}_k$ and Θ_k : ∇_k and $\overline{\nabla}_k$ are computed from scratch at the end of the kth iteration in order to be used as ∇_{k+1} and $\overline{\nabla}_{k+1}$ in the next iteration, while Θ_k is always empty when an iteration starts. The case of the follower links is discussed below.

Let us start with Algorithm 1, our main algorithm. The initialization consists in creating all the atoms/nodes of $\Delta_{|\mathrm{W}[0]|+2} = \Delta_{|\mathrm{W}[0]|+1}$ and setting their $(|\mathrm{W}[0]| + 1)$-followers, which, by Item 2 of Remark 1, are both trivial: atoms are singletons $(\{p\})_{0\leq p<|s|}$ and $\mathrm{F}_{|\mathrm{W}[0]|+1}(\{p-1\}) = \{p\}$ (loops at Lines 1 and 2). The variable c used to parse the sorted table W of elements of $\mathcal{P}$ and the sets listed above are also initialized on Line 3.

Line 4 of Algorithm 1 starts the main loop. The kth iteration, k taking its values in the set of lengths of the codewords of $\mathcal{P}$ in decreasing order, computes first all the k-specific atoms, so that, at the end of this iteration, the atoms of Δ_k correspond exactly to the root nodes of the tree structures, and, secondly, performs all the necessary updates for the next iteration, that is, computes ∇_k and $\overline{\nabla}_k$ and the k-follower links. Remark that during the k^{th} iteration, a root node belongs to Δ_{k+1} or Δ_k (possibly to both). More specifically, a node belongs to Δ_{k+1} if and only if it is either a root node not belonging to Θ_k or the direct child of a root node belonging to Θ_k.

Proposition 1 says that an atom α is k-specific only if it is a k-code atom or if $\mathrm{F}_k^{-1}(\alpha) \cap \nabla_{k+1} \neq \emptyset$. This is why the main loop contains two internal loops. The first one (Lines 6–9) runs over the elements of $\mathcal{P}$ with length k and creates the

corresponding k-code atoms. The second loop (Lines 11–21) constructs the k-specific atoms α such that $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1} \neq \emptyset$. It runs over the k between two successive (different) lengths of elements of $\mathcal{P}$ (there is not necessarily an element of length k in $\mathcal{P}$ for any k).

Algorithm 1: `compute` Δ_1

```
1  for p ← 0 to |s| − 1 do {p} ← new_node;
2  for p ← 1 to |s| − 1 do F_{|W[0]|+1}({p − 1}) ← {p};
3  c ← 0; ∇_{|W[0]|+1} ← ∅; ∇̄_{|W[0]|+1} ← ∅; Θ_{|W[0]|} ← ∅;
4  repeat
5      k ← |W[c]|;
6      repeat
7          Compute_Atom_Code(W[c]);
8          c ← c + 1;
9      until c ≥ #P or |W[c − 1]| > |W[c]|;
10     if c < #P then m ← |W[c]| else m ← 0;
11     repeat
12         forall δ ∈ ∇_{k+1} do
13             if F_k(δ) = Null then
14                 Compute_Follower(δ);
15         forall δ ∈ ∇̄_{k+1} do F_k(δ) ← F_{k+1}(α);        /* where α ∈ C(δ) */
16         ∇_k ← ∅; ∇̄_k ← ∅;
17         forall γ ∈ Θ_k do
18             if ∃(α, δ) ∈ C(γ)^2 such that F_k(α) ≠ F_k(δ) then ∇_k ← ∇_k ∪ {γ};
19             else ∇̄_k ← ∇̄_k ∪ {γ}
20         k ← k − 1; Θ_k ← ∅;
21     until k ≤ m or ∇_{k+1} = ∅;
22 until c ≥ #P;
```

Code atoms

For all the elements w of length k in $\mathcal{P}$, Loop Lines 6–9 calls Procedure `Compute_Atom_Code`, which creates the corresponding k-code atoms. In order to get all the children of the k-code atom corresponding to an element $w \in \mathcal{P}$ with $|w| = k$, this procedure climbs up from the singleton atoms corresponding to occurrence positions of w, to a root node (at this point, a root node corresponds to an atom of Δ_{k+1}). It uses two temporary sets of atoms: Λ and Φ. Each node parsed during the climbing is added to the set Λ in order to be avoided by the next iterations. The set Φ stores the nodes of Δ_{k+1} (the root nodes) which are included in the current k-code atom. Note that a k-code atom γ is not always k-specific. This corresponds to the case when $\#\Phi \leq 1$ and there is nothing to do here (Line 8). If $\#\Phi > 1$ (this atom is then k-specific), the k-code atom γ is created and is added to Θ_k. Next, its set of children is set to Φ and the follower link of all atoms of Δ_{k+1} having this new atom as k-follower has to be updated (Lines 12–16). From Remark 2, these atoms are exactly the ones of Δ_{k+1} which contain an atom having a child of γ as $(k+1)$-follower link. Let α be an atom of $\mathrm{C}(\gamma)$ and ζ be an atom of $\mathrm{F}_{k+1}^{-1}(\alpha) \subset \Delta_{k+2}$. If ζ itself belongs to Δ_{k+1} (which is equivalent to being either a root node or the direct child of a node of Θ_k – condition Line 13), we set $\mathrm{F}_k(\zeta)$ to γ. Otherwise, we set the k-follower link of $\mathrm{P}(\zeta)$ to γ. According to Remark 2, performing this for all $\alpha \in \mathrm{C}(\gamma)$ and all $\zeta \in \mathrm{F}_{k+1}^{-1}(\alpha)$ ensures us that we parse all atoms having γ as k-follower link.

Other specific atoms

We also have to compute the k-specific atoms γ which are not k-code atoms, that are such that $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1} \neq \emptyset$. These atoms are computed by the loop Lines 11–21 of Algorithm 1 (this loop runs on k but its bounds ensure that there are no k-code atoms between them). To this end, we parse the atoms δ of ∇_{k+1}, and call Procedure `Compute_Follower` (δ) to compute the atom of Δ_k which is the k-follower of δ.

Remark 5. Let k be a positive integer and γ be an atom of Δ_k which is k-specific without being a k-code atom.

1. Let $\delta \in \nabla_{k+1}$. We have $\mathrm{F}_k(\delta) = \gamma$ if and only if $\mathrm{F}_{k+1}(\alpha) \subset \gamma$ for all atoms $\alpha \in \Delta_{k+2}$ included in δ (an atom of ∇_{k+1} is $(k+1)$-specific, thus does not belong to Δ_{k+2}). In terms of trees structure, it says that $\mathrm{F}_k(\delta) = \gamma$ if and only if all the $(k+1)$-followers of the children nodes of δ have γ as parent.
2. Two atoms δ and δ' belong to $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1}$ if and only if there exists a sequence $(\delta_i)_{1 \leq i \leq p}$ of atoms of $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1}$ satisfying $\delta_1 = \delta$, $\delta_p = \delta'$ and, for all $1 \leq i < p$, that there exist $\alpha \subseteq \delta_i$ and $\beta \subseteq \delta_{i+1}$, both belonging to Δ_{k+2}, such that $\mathrm{F}_{k+1}(\alpha) = \mathrm{F}_{k+1}(\beta)$.
3. We have $\gamma = \cup_{\alpha \in \Psi} \mathrm{F}_{k+1}(\alpha)$, where Ψ contains all the atoms α of Δ_{k+1} which are subsets of an atom of $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1}$.

Procedure `Compute_Atom_code(`w`)`

```
1  Λ ← ∅; Φ ← ∅;
2  forall p ∈ O_w do
3      δ ← {p};
4      while P(δ) ≠ Null and δ ∉ Λ do
5          Λ ← Λ ∪ {δ}; δ ← P(δ);
6      if P(δ) = Null and δ ∉ Λ then
7          Λ ← Λ ∪ {δ}; Φ ← Φ ∪ {δ};
8  if #Φ > 1 then
9      γ ← new_node; Θ_k ← Θ_k ∪ {γ};
10     forall α ∈ Φ do
11         P(α) ← γ;
12         forall ζ ∈ F_{k+1}^{-1}(α) do
13             if P(ζ) = Null or P(ζ) ∈ Θ_k (i.e. ζ ∈ Δ_{k+1}) then
14                 F_k(ζ) ← γ;
15             else if F_k(P(ζ)) = Null then
16                 F_k(P(ζ)) ← γ;
```

Proof. Items 1 and 3 are straightforward from definitions and basic properties, therefore we prove only Item 2. Let a and a' be two positions of δ and δ', respectively. Since $\mathrm{F}_k(\delta) = \mathrm{F}_k(\delta') = \gamma$, we have $a+1 \overset{k}{\smile} a'+1$. By Definition 2, there exists a sequence of positions $x_1, x_2, \ldots, x_n$ such that $x_1 = a+1$, $x_n = a'+1$ and $x_m \overset{k}{\smile} x_{m+1}$ for all $1 \le m < n$. Moreover, assuming that γ is not a k-code atom then implies that $x_m \overset{k+1}{\not\smile} x_{m+1} \Rightarrow x_m - 1 \overset{k+1}{\smile} x_{m+1} - 1$. Let $j[\ell]$ be the ℓth index such that $x_{j[\ell]} \overset{k+1}{\not\smile} x_{j[\ell]+1}$. With what precedes, there exists an atom $\delta_\ell \in \nabla_{k+1}$ containing both $x_{j[\ell]} - 1$ and $x_{j[\ell]+1} - 1$. For all such ℓ but the last one, all positions x_m with $j[\ell] < m \le j[\ell+1]$ belong to a same atom of Δ_{k+1}. It follows that the atom $\alpha \subseteq \delta_\ell$ containing $x_{j[\ell]+1}$ and the atom $\beta \subseteq \delta_{\ell+1}$ containing $x_{j[\ell+1]}$, verify $\mathrm{F}_{k+1}(\alpha) = \mathrm{F}_{k+1}(\beta)$. The converse is a direct consequence of the definitions. □

The preceding remark is used by Procedure `Compute_Follower` to compute the k-specific atom γ corresponding to a $(k+1)$-special atom δ (*i.e.* such that $\mathrm{F}_k(\delta) = \gamma$). The set Ω, initialized to $\{\delta\}$, is devoted to temporarily storing the atoms of $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1}$ and grows each time a node belonging to $\mathrm{F}_k^{-1}(\gamma) \cap \nabla_{k+1}$ is encountered (Line 11). The loop starting at line 4 first picks an atom α in Ω and considers all its children β. In the case that $\mathrm{F}_{k+1}(\beta)$ was not encountered before (*i.e.* is not a child of γ), γ becomes its parent. Next, all atoms ζ having the same $(k+1)$-follower link as β are taken into account (Lines 6–11). This has a twofold purpose. First, we have to set or update the k-follower link of the atoms of Δ_{k+1} to which they belong. A second purpose is to check if these atoms of Δ_{k+1} are "new" $(k+1)$-special atoms belonging to $\mathrm{F}_k^{-1}(\gamma)$. In this case, they are added to Ω (Line 11) in order to be processed during a next iteration of the main loop of the procedure.

Procedure `Compute_Follower(`δ`)`

```
1  γ ← new_node; Θ_k ← Θ_k ∪ {γ}; Ω ← {δ}; F_k(δ) ← γ;
2  while Ω ≠ ∅ do
3      α ← pop(Ω);
       /* pop(Ω) picks an element ω in Ω, removes it from Ω and finally returns ω.   */
4      forall β ∈ C(α) such that F_{k+1}(β) ∉ C(γ) do
5          P(F_{k+1}(β)) ← γ;
6          forall ζ ∈ F_{k+1}^{-1}(F_{k+1}(β)) do
7              if P(ζ) = Null or P(ζ) ∈ Θ_k then
8                  F_k(ζ) ← γ;
9              else if F_k(P(ζ)) = Null then
10                 F_k(P(ζ)) ← γ;
11                 if P(ζ) ∈ ∇_{k+1} then Ω ← Ω ∪ {P(ζ)};
```

Follower links

At each iteration of the main loop of Algorithm 1, we need the $(k+1)$-follower links of all nodes in Δ_{k+2}, in order to make sure that we have set the k-follower links of all nodes in Δ_{k+1} at the end of this loop. Let δ be a node of Δ_{k+1}. We have to face the following mutually exclusive situations:

1. $\delta \notin \Theta_{k+1}$ and $F_k(\delta) \notin \Theta_k$: in this case, we have $F_k(\delta) = F_{k+1}(\delta)$ and the follower link does not have to be updated.
2. $\delta \notin \Theta_{k+1}$ and $F_k(\delta) \in \Theta_k$: $F_k(\delta)$ has to be updated at the creation of $F_k(\delta)$. Since $\delta \notin \Theta_{k+1}$, there exists an atom $\beta \in C(F_k(\delta))$ such that $\delta \in F_{k+1}^{-1}(\beta)$.
3. $\delta \in \Theta_{k+1}$ and $F_k(\delta) \in \Theta_k$: $F_k(\delta)$ has to be updated at the creation of $F_k(\delta)$. Since $\delta \in \Theta_{k+1}$, there exists an atom $\beta \in C(F_k(\delta))$ such that $C(\delta) \cap F_{k+1}^{-1}(\beta) \neq \emptyset$.
4. $\delta \in \Theta_{k+1}$ and $F_k(\delta) \notin \Theta_k$: $F_k(\delta)$ is `Null` and has to be set to $F_{k+1}(\beta)$, where $\beta \in C(\delta)$.

There is nothing to do in Case 1. Cases 2 and 3 are handled by the loops Lines 12–16 of Procedure `Compute_Atom_Code` and Lines 6–11 of Procedure `Compute_Follower`. It remains to deal with atoms of Case 4, which are exactly the ones belonging to $\overline{\nabla}_{k+1}$. This is done in Loop Line 15 of Algorithm 1.

Let us explain why in practice each atom/node has just one follower link, which is updated when needed. It is enough to remark that we just need the $(k+1)$-follower links, which go from Δ_{k+2} to Δ_{k+1}. By induction, we can prove that all the $(k+1)$-follower links are valid when an iteration of the main loop starts. The only issue which could arise might be if we use an updated k-follower link as a $(k+1)$-follower link. One can see that no such confusion is possible. In the Procedure `Compute_Atom_Code`, all the elements of Φ are in Δ_{k+1} and by construction (subsets of occurrence positions of words of $\mathcal{P}$ partitions the whole set of positions) cannot be involved in another call of `Compute_Atom_Code`. In Procedure `Compute_Follower`, we consider first follower links of atoms β, which belong to Δ_{k+2} and not to Δ_{k+1} (as children of an atom of this set). The other follower links used go to $F_{k+1}(\beta)$ which do not belong to Δ_k.

End of main loop

At this point (Line 16 of Algorithm 1), we have computed all the k-specific atoms and updated the follower function to F_k for all atoms of Δ_{k+1}. It just remains to determine the sets ∇_k and $\overline{\nabla}_k$ before starting the next iteration. This is done in Loop Lines 17–18, running over atoms of Θ_k, which is split into ∇_k and $\overline{\nabla}_k$ following the remark just below (a direct consequence of Remark 3).

Remark 6. Let k be a positive integer. An atom γ is k-special if and only if it is k-specific and it contains two atoms α and β of Δ_{k+1} satisfying $F_k(\alpha) \neq F_k(\beta)$.

Complexity

Proposition 5. *Algorithm 1 runs in* $O(|s|)$ *times using* $O(|s|)$ *memory space.*

Proof. The memory space used by the algorithm is linear with $|s|$. It uses first a set of trees with $|s|$ leaves, each internal node having at least two children. Moreover, the number of follower links, the other structure needed, is $O(|s|)$ (Corollary 1 and Proposition 4).

To evaluate the total time complexity, let us first consider the total time required to create k-code atoms (Lines 2–7 of Procedure `Compute_Atom_Code`). Since each node parsed is added to a set Λ and avoided in the next iterations, the total number of iterations is smaller than the number of edges "parent/child", thus than $|s|$. Since its destination becomes a child of a k-specific atom, each follower link is processed at most once by Loop starting Lines 12 of Procedure `Compute_Atom_Code`. The total number of iterations is then smaller than the number of follower links, thus $O(|s|)$ (Corollary 1 and Proposition 4).

Let us split the total time complexity consumed by Procedure `Compute_Follower` into two parts: the first one for the time spent in Loop starting Line 6, and the second one for gathering what remains. With the same argument as before, the total number of iterations of Loop starting at Line 6 is $O(|s|)$. The total time spent during the second part is also linear with $|s|$: at each iteration of Loop starting at Line 4, a parent–child edge is created and a node is processed at most once by the main loop.

Each atom/node appears as an element of Θ_k, thus as an element of ∇_k or $\overline{\nabla}_k$, for at most one value of k. It implies that the total number of iterations of both Loop Line 15 and Loop Lines 17–21 is $O(|s|)$. □

4. Alignment-free sequences comparison

In this section, we are concerned with a set of sequences $S = (S^{(i)})_i$. Note that all definitions, results and algorithms of the preceding section, established for a single sequence, extend naturally to sets of sequences (one just needs to replace "position" by pair "index of sequence/position in the sequence"). Our strategy is to define, and compute, a variable length local decoding of the set of sequences, with a prefix code adapted to the sequences of the set, and then to compare the compositions of sequences of this decoding.

4.1. Building a prefix code

Building a prefix code which is compliant with a given set of sequences can always be done by pruning the suffix trie or tree of this set, somehow in the same way as the context algorithm used in learning variable lengths Markov chains does (except that this last one deals with suffix codes and prefix tries [9]). To decide which node to prune, we need an adequate criterion. The one we choose is based upon the occurrence probabilities of the corresponding words.

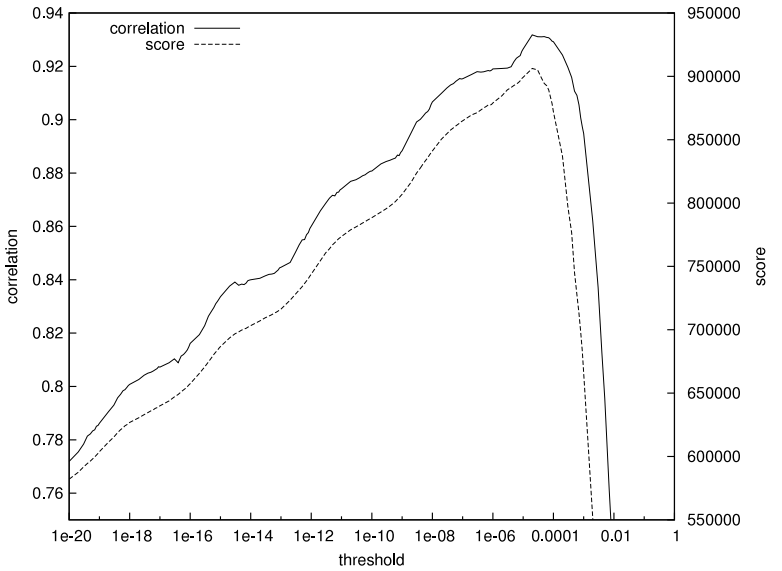

Fig. 2. Score of the local decoding (dashed) and the Pearson coefficient of the related dissimilarity with the reference identity percentage (plain) *versus* the threshold used to build the corresponding prefix code.

More specifically, to construct our prefix code, we start by fixing a probability threshold t. Next, we parse the suffix tree of S from the root node and prune at the shallowest nodes of the suffix tree which are associated to words having a probability smaller than t to occur more than once in the whole set of sequences. These probabilities are computed under an order 1 Markov model estimated from the set of sequences, and are in practice approximated from a binomial distribution like in [11], thus using a constant computation time per node.

4.2. A heuristic to select a threshold

Fixing an arbitrary threshold might sound a good way to select a prefix code well suited to the composition and the total length of the set of sequences. Unfortunately, a multiple testing issue arises here. While the length of the set of sequences grows, one has to consider longer words – deeper nodes – for a same probability threshold t. The longer the words we consider, the greater their number, and so is the probability that some of them occur more than once.

To tackle this issue, we define the score of a local decoding $L = (L^{(i)})_i$ of S (L is itself a set of sequences) in the following way. For a decoded symbol $\mathtt{x}$, we put $\mathbf{occ}(\mathtt{x}, L)$ for the number of occurrences of $\mathtt{x}$ in L and $\mathbf{pres}(\mathtt{x}, L)$ for the number of sequences of L in which $\mathtt{x}$ occurs. The score of L is now defined as:

$$\mathbf{score}(L) = \sum_{\substack{\mathtt{x} \text{ such that} \\ \mathbf{pres}(\mathtt{x},L)>1 \\ \mathbf{occ}(\mathtt{x},L)<2.\mathbf{pres}(\mathtt{x},L)}} \mathbf{pres}(\mathtt{x}, L).$$

The variable length local decoding that we choose is the one that we obtain from the prefix code corresponding to the threshold achieving the greatest score. The plot of the score of a local decoding associated to a threshold t, *versus* t is roughly unimodal. For small values of t, there are a lot of symbols $\mathtt{x}$ such that $\mathbf{pres}(\mathtt{x}, L) = 1$, while, as t goes to 1, there are more and more symbols $\mathtt{x}$ such that $\mathbf{occ}(\mathtt{x}, L) \geq 2.\mathbf{pres}(\mathtt{x}, L)$. In these two cases, a lot of symbols $\mathtt{x}$ do not contribute to the score. Indeed, we observe an intermediate value of t which maximizes the score (dashed plot of Fig. 2).

Note that the threshold selection stage is not very time consuming since we assume that the plot score *versus* threshold is unimodal and perform a dichotomic mode search to find it.

4.3. Dissimilarity

We use the same basic dissimilarity as in [5]. Let $(L^{(i)})_i$ be the variable length decoding of set of sequences $(S^{(i)})_i$ which is the "best" according to the threshold selection of Section 4.1. Let $\mathbf{occ}(\mathtt{x}, L^{(i)})$ be the number of occurrences of a symbol $\mathtt{x}$ in the decoded sequence $L^{(i)}$. The dissimilarity between sequences $S^{(i)}$ and $S^{(j)}$ is then defined in the following way:

$$\mathrm{d}(S^{(i)}, S^{(j)}) = 1 - \frac{\sum_{\mathtt{x}} \min\{\mathbf{occ}(\mathtt{x}, L^{(i)}), \mathbf{occ}(\mathtt{x}, L^{(j)})\}}{\min\{|S^{(i)}|, |S^{(j)}|\}}.$$

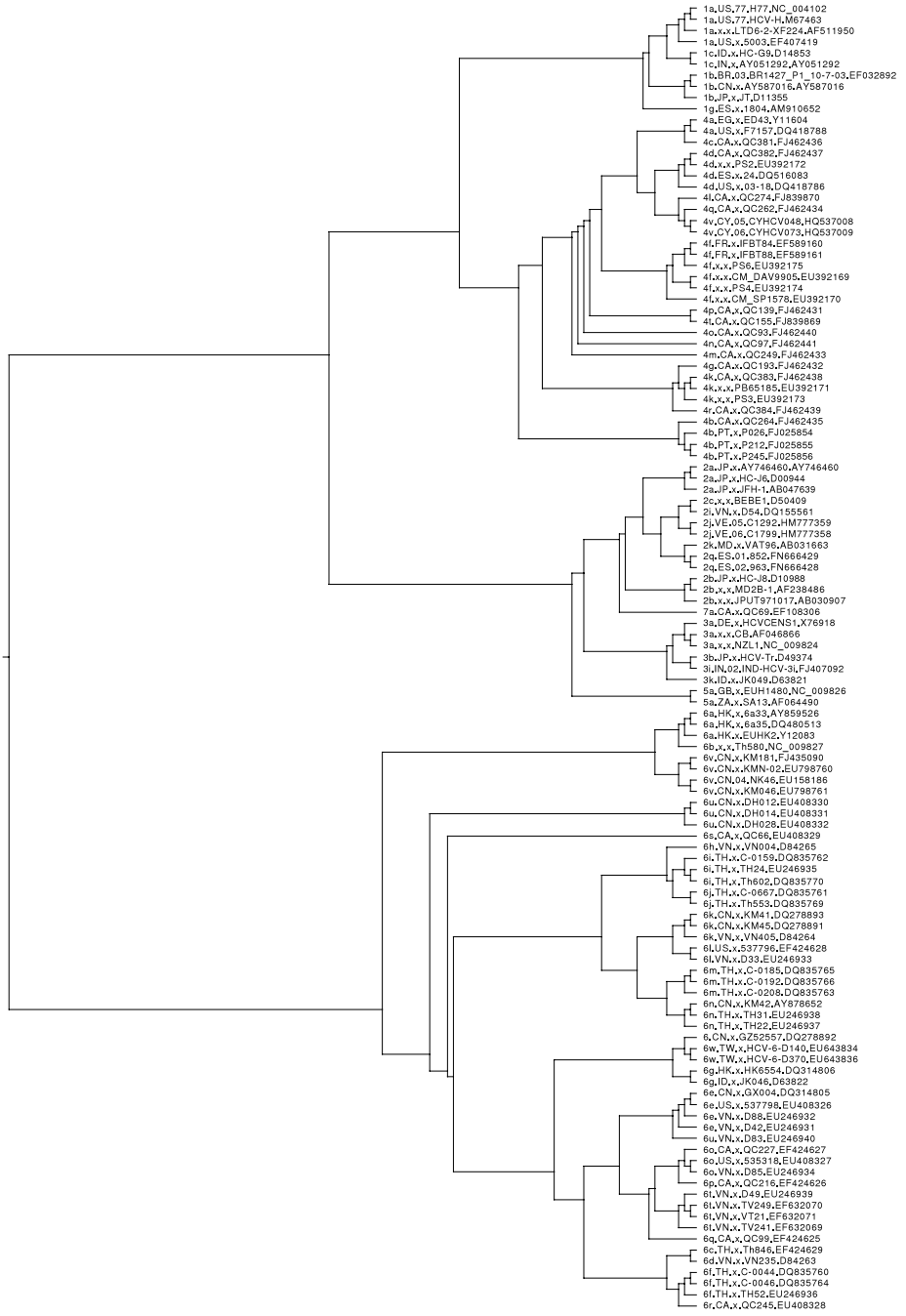

Fig. 3. Neighbor-joining tree computed from the variable length local decoding dissimilarity (drawn using Archaeopteryx [6]).

Note that $\sum_{\mathrm{x}} \min\{\mathbf{occ}(\mathrm{x}, L^{(i)}), \mathbf{occ}(\mathrm{x}, L^{(j)})\}$ is the largest number of matches that can be observed between two sequences having the same symbol compositions as $L^{(i)}$ and $L^{(j)}$, while $\min\{|S^{(i)}|, |S^{(j)}|\}$ is the largest number of matches that can be observed between two sequences having lengths $|S^{(i)}|$ and $|S^{(j)}|$. In particular, we have $\mathrm{d}(S^{(i)}, S^{(j)}) = \mathrm{d}(S^{(j)}, S^{(i)})$ and $\mathrm{d}(S^{(i)}, S^{(i)}) = 0$.

4.4. An application: Hepatitis C virus typing

To evaluate the relevance of the preceding dissimilarity in terms of sequence comparison, we consider the reference alignment of the Hepatitis C Virus Database Project [8], which contains the sequences of complete genomes of 117 Hepatitis C viruses of various types and subtypes. Sequences lengths go from 9112 to 9689.

We start by computing the identity percentages between each pair of lines of the alignment. The identity percentage between two lines is here defined as the number of matches between these lines divided by the length of the shortest sequence. Next, we transform each alignment into a set of sequences (by removing all gaps) and compute the variable length local decoding dissimilarity. A first way to evaluate its accuracy is given by the Pearson correlation coefficient computed over the upper or lower triangular part (excluding diagonal) of the dissimilarity and the corresponding part of the identity percentage taken as reference. Our dissimilarity is correlated with a Pearson coefficient of about 0.93 to the identity percentage in the multiple alignment.

To check how relevant is the heuristic selecting the prefix code, we plot on Fig. 2 the evolution of the Pearson coefficient of dissimilarities arising from local decodings regarding prefix codes computed from the threshold t, as well as the evolution of the score defined in Section 4.2, both *versus* t. We observe that the mode of the score plot corresponds remarkably well to the range of thresholds where the correlation with the identity percentage is maximal.

Finally, we compute the neighbor joining tree of our dissimilarity by using SplitsTree4 [7] (Fig. 3). This tree is perfectly consistent with the typing and subtyping of the viruses. In Fig. 3, the type of each sequence is given by the number starting its identifier and its subtype by the letter just following it. One can observe that, for each type/subtype of viruses, there is a subtree which contains all the corresponding sequences and only these ones.

5. Conclusion

The variable length local decoding can be seen as an alternative to alignment for sequence comparison, while not being time or memory consuming: it has the same order of complexity as what is needed just to read sequences.

We plan to run a more systematic evaluation of the approach over biological dataset, as well as to apply the approach to other related questions, like the selection of multiple alignment anchors to be taken as input by software such as DIALIGN [10].

On a more methodological point of view, there is not much left to do to improve the algorithm, but there is still work to do to improve the way of selecting a "good" prefix code: we select the prefix code which has the best score (in the sense of Section 4.2) only among the set of prefix codes which can be obtained by the threshold pruning process described in Section 4.1. A natural improvement would be to increase the set of prefix codes among which we compute the score, while keeping feasible computation times.

References

[1] V. Acuña, C-writing and classification by context (Spanish). Thesis of M.Sc. in computer science, Universidad de Chile, 2004.
[2] V. Acuña, G. Didier, A. Maass, Coding with variable block maps, Theoretical Computer Science 369 (1–3) (2006) 396–405.
[3] G. Didier, Caractérisation des n-écritures et application à l'étude des suites de complexité ultimement n+ cste, Theoretical Computer Science 215 (1–2) (1999) 31–49.
[4] G. Didier, L. Debomy, M. Pupin, M. Zhang, A. Grossmann, C. Devauchelle, I. Laprevotte, Comparing sequences without using alignments: application to HIV/SIV subtyping, BMC Bioinformatics 8 (1) (2007) 1.
[5] G. Didier, I. Laprevotte, M. Pupin, A. Hénaut, Local decoding of sequences and alignment-free comparison, Journal of Computational Biology 13 (8) (2006) 1465–1476.
[6] M.V. Han, C.M. Zmasek, Phyloxml: XML for evolutionary biology and comparative genomics, BMC Bioinformatics 10 (1) (2009) 356.
[7] D.H. Huson, D. Bryant, Application of phylogenetic networks in evolutionary studies, Molecular Biology and Evolution 23 (2) (2006) 254–267.
[8] C. Kuiken, K. Yusim, L. Boykin, R. Richardson, The Los Alamos Hepatitis C sequence database, Bioinformatics 21 (3) (2005) 379–384.
[9] J. Rissanen, A universal data compression system, IEEE Transactions on Information Theory 29 (5) (1983) 656–664.
[10] A.R. Subramanian, M. Kaufmann, B. Morgenstern, et al., Dialign-tx: greedy and progressive approaches for segment-based multiple sequence alignment, Algorithms Molecular Biology 3 (6) (2008).
[11] J. van Helden, B. André, J. Collado-Vides, Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies 1, Journal of Molecular Biology 281 (5) (1998) 827–842.
[12] S. Vinga, J. Almeida, Alignment-free sequence comparison a review, Bioinformatics 19 (4) (2003) 513–523.