Information and Computation 262 (2018) 57–68

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Alignment-free sequence comparison using absent words ☆

Panagiotis Charalampopoulos [a], Maxime Crochemore [a], Gabriele Fici [b], Robert Mercaş [c], Solon P. Pissis [a,*]

[a] Department of Informatics, King's College London, London, UK
[b] Dipartimento di Matematica e Informatica, Università di Palermo, Palermo, Italy
[c] Department of Computer Science, Loughborough University, UK

ARTICLE INFO

Article history:
Received 23 December 2016
Received in revised form 22 December 2017
Available online 22 June 2018

Keywords:
Sequence comparison
Absent words
Forbidden words
Circular words
q-grams

ABSTRACT

Sequence comparison is a prerequisite to virtually all comparative genomic analyses. It is often realised by sequence alignment techniques, which are computationally expensive. This has led to increased research into alignment-free techniques, which are based on measures referring to the composition of sequences in terms of their constituent patterns. These measures, such as q-gram distance, are usually computed in time linear with respect to the length of the sequences. In this paper, we focus on the complementary idea: how two sequences can be efficiently compared based on information that does not occur in the sequences. A word is an *absent word* of some sequence if it does not occur in the sequence. An absent word is *minimal* if all its proper factors occur in the sequence. Here we present the first linear-time and linear-space algorithm to compare two sequences by considering *all* their minimal absent words. In the process, we present results of combinatorial interest, and also extend the proposed techniques to compare circular sequences. We also present an algorithm that, given a word x of length n, computes the largest integer for which all factors of x of that length occur in some minimal absent word of x in time and space $\mathcal{O}(n)$. Finally, we show that the known asymptotic upper bound on the number of minimal absent words of a word is tight.

1. Introduction

Sequence comparison is an important step in many basic tasks in bioinformatics, from phylogeny reconstruction to genome assembly. It is often realised by sequence alignment techniques, which are computationally expensive, often requiring quadratic time in the length of the sequences. This has led to increased research into *alignment-free* techniques [2]. Hence standard notions for sequence comparison are gradually being complemented and in some cases replaced by alternative ones [3,4]. One such notion is based on comparing the words that are absent in each sequence [5]. A word is an *absent word* (or a forbidden word) of some sequence if it does not occur in the sequence. Absent words represent a type of *negative information*: information about what does not occur in the sequence.

☆ A preliminary version of this paper, without the first author, was presented at the 12th Latin American Theoretical Informatics Symposium (LATIN 2016) [1].

* Corresponding author.
E-mail addresses: panagiotis.charalampopoulos@kcl.ac.uk (P. Charalampopoulos), maxime.crochemore@kcl.ac.uk (M. Crochemore), gabriele.fici@unipa.it (G. Fici), R.G.Mercas@lboro.ac.uk (R. Mercaş), solon.pissis@kcl.ac.uk (S.P. Pissis).

https://doi.org/10.1016/j.ic.2018.06.002
0890-5401/

Given a sequence of length n, the number of absent words of length at most n is exponential in n. However, the number of certain classes of absent words is only linear in n. This is the case for *minimal absent words*, that is, absent words in the sequence for which all proper factors occur in the sequence [6]. An upper bound on the number of minimal absent words of a word of length n over an alphabet Σ of size σ is known to be σn [7,8]. Hence it may be possible to compare sequences in time proportional to their lengths, for a fixed-sized alphabet, instead of proportional to the product of their lengths. In what follows, we mainly consider sequences over a *fixed-sized alphabet* since the most commonly studied alphabet in this context is {A, C, G, T}.

An $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing all minimal absent words of a sequence of length n over a fixed-sized alphabet based on the construction of suffix automata was presented in [7]. The computation of minimal absent words based on the construction of suffix arrays was considered in [9]; although this algorithm has a linear-time performance in practice, the worst-case time complexity is $\mathcal{O}(n^2)$. New $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space suffix-array-based algorithms were presented in [10–12] to bridge this unpleasant gap. An implementation of the algorithm presented in [11] is currently, to the best of our knowledge, the fastest available for the computation of minimal absent words. A more space-efficient solution to compute all minimal absent words in time $\mathcal{O}(n)$ was also presented in [13] and an external-memory algorithm in [14].

In this paper, we consider the problem of comparing two sequences x and y of respective lengths m and n, using their sets of minimal absent words. In [15], Chairungsee and Crochemore introduced a measure of similarity between two sequences based on the notion of minimal absent words. They made use of a length-weighted index to provide a measure of similarity between two sequences, using sample sets of their minimal absent words, by considering the length of each member in the symmetric difference of these sample sets. This measure can be trivially computed in time and space $\mathcal{O}(m+n)$ provided that these sample sets contain minimal absent words of some bounded length ℓ. For unbounded length, the same measure can be trivially computed in time $\mathcal{O}(m^2+n^2)$: for a given sequence, the cumulative length of all its minimal absent words can grow *quadratically* with respect to the length of the sequence. This length-weighted index forms the basis of a fundamentally new, recently introduced, algorithm for on-line pattern matching [16].

The same problem can be considered for two *circular* sequences. The measure of similarity of Chairungsee and Crochemore can be used in this setting provided that one extends the definition of minimal absent words to circular sequences. In Section 5, we give a definition of minimal absent words for a circular sequence from the Formal Language Theory point of view. We believe that this definition may also be of interest from the point of view of Symbolic Dynamics, which is the original context in which minimal absent words have been introduced [6].

We also find a connection between the information provided by minimal absent words and the information provided by the set of q-grams. The former (absent words) can be seen as some kind of *negative* information, while the latter (q-grams) as *positive* information.

Our contributions. Here we make the following contributions:

a) We first show that the upper bound $\mathcal{O}(\sigma n)$ on the number of minimal absent words of a word of length n over an alphabet of size σ is tight if $2 \leq \sigma \leq n$ (Section 3).
b) We present an $\mathcal{O}(m+n)$-time and $\mathcal{O}(m+n)$-space algorithm to compute the similarity measure introduced by Chairungsee and Crochemore by considering *all* minimal absent words of two sequences x and y of lengths m and n, respectively, over a fixed-sized alphabet; thereby showing that it is indeed possible to compare two sequences in time proportional to their lengths (Section 4).
c) We show how this algorithm can be applied to compute this similarity measure for two circular sequences x and y of lengths m and n, respectively, in the same time and space complexity as a result of the extension of the definition of minimal absent words to circular sequences (Section 5).
d) We then present an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm that given a word x of length n over an integer alphabet computes the largest integer $q(x)$ for which each $q(x)$-gram of x is a $q(x)$-gram of some minimal absent word of x (Section 6).
e) Finally, we provide an open-source code implementation of our algorithms for sequence comparison using minimal absent words and investigate potential applications of our theoretical findings (Section 7).

2. Preliminaries

We begin with basic definitions and notation. Let $y = y[0]y[1]\,..\,y[n-1]$ be a *word* (or *string*) of length $|y| = n$ over a finite ordered alphabet Σ of size $|\Sigma| = \sigma = \mathcal{O}(1)$. We also consider the case of words over an *integer alphabet*, where each letter is replaced by its rank in such a way that the resulting word consists of integers in the range $\{1, \ldots, n\}$.

For two positions i and j on y, we denote by $y[i\,..\,j] = y[i]\,..\,y[j]$ the *factor* (sometimes called *substring*) of y that starts at position i and ends at position j (it is of length 0 if $j < i$), and by ε the *empty word*, word of length 0. We recall that a prefix of y is a factor that starts at position 0 ($y[0\,..\,j]$), a suffix is a factor that ends at position $n-1$ ($y[i\,..\,n-1]$), and that a factor of y is a *proper* factor if it is not y itself. A proper factor of y that is neither a prefix nor a suffix of y is called an *infix* of y. The set of all factors of the word y is denoted by $\mathcal{F}_y$. Any factor of length $q \geq 1$ of y is called a *q-gram* (or *q-mer*) of y. The *q-gram set* of y is the set of all factors of length q of y. We denote the reverse word of y by $\mathsf{rev}(y)$, i.e. $\mathsf{rev}(y) = y[n-1]y[n-2]\ldots y[1]y[0]$. We say that a word x is a *power* of a word y if there exists a positive integer k, $k > 1$, such that x is expressed as k consecutive concatenations of y, denoted by $x = y^k$.

Let x be a word of length m with $0 < m \leq n$. We say that there exists an *occurrence* of x in y, or, more simply, that x *occurs in* y, when x is a factor of y. Every occurrence of x can be characterised by a starting position in y. We thus say that x occurs at the *starting position* i in y when $x = y[i\,..\,i+m-1]$. Opposingly, we say that the word x is an *absent word* of y if it does not occur in y. The absent word x of y is *minimal* if and only if all its proper factors occur in y. The set of all minimal absent words for a word y is denoted by $\mathcal{M}_y$. For example, if $y = \mathtt{abaab}$, then $\mathcal{M}_y = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}\}$. In general, if we suppose that all the letters of the alphabet appear in y which has length n, the length of a minimal absent word of y lies between 2 and $n+1$. It is equal to $n+1$ if and only if y is of the form a^n for some letter a. So, if y contains occurrences of at least two different letters, the length of any minimal absent word of y is upper bounded by n.

We now recall some basic facts about minimal absent words in Formal Language Theory. For further details and references the reader is recommended [17]. A *language* over the alphabet Σ is a set of finite words over Σ. A language is *regular* if it is recognised by a finite state automaton. A language is called *factorial* if it contains all the factors of its words, while it is called *antifactorial* if no word in the language is a proper factor of another word in the language. Given a word x, the language *generated* by x is the language $x^* = \{x^k \mid k \geq 0\} = \{\varepsilon, x, xx, xxx, \ldots\}$. The *factorial closure* of a language L is the language consisting of all factors of the words in L, that is, the language $\mathcal{F}_L = \cup_{y \in L} \mathcal{F}_y$. Given a factorial language L, one can define the (antifactorial) language of minimal absent words for L as

$$\mathcal{M}_L = \{aub \mid a, b \in \Sigma, u \in \Sigma^*, aub \notin L, au, ub \in L\}.$$

Notice that $\mathcal{M}_L$ is not the same language as the union of $\mathcal{M}_x$ for $x \in L$. Every factorial language L is uniquely determined by its (antifactorial) language of minimal absent words $\mathcal{M}_L$, through the equation

$$L = \Sigma^* \setminus \Sigma^* \mathcal{M}_L \Sigma^*. \tag{1}$$

The converse is also true, since by the definition of a minimal absent word we have

$$\mathcal{M}_L = \Sigma L \cap L \Sigma \cap (\Sigma^* \setminus L). \tag{2}$$

The previous equations define a bijection between factorial and antifactorial languages. Moreover, this bijection preserves regularity. In the case of a single word x, the set of minimal absent words for x is indeed the antifactorial language $\mathcal{M}_{\mathcal{F}_x}$. Thus, applying (1) and (2) to the language of factors of a single word, we have the following lemma.

Lemma 1. *Given two words x and y, $x = y$ if and only if $\mathcal{M}_x = \mathcal{M}_y$.*

Given a word x of length n over a fixed-sized alphabet, it is possible to compute a trie storing all the minimal absent words of x in time and space linear in n. The size (number of nodes) of this trie is linear in n. Furthermore, we can retrieve x from its set of minimal absent words in time and space linear in the size of the input trie representing the minimal absent words of x. Indeed, the algorithm MF-TRIE, introduced in [7], builds the tree-like deterministic automaton accepting the set of minimal absent words for a word x taking as input the factor automaton of x, that is the minimal deterministic automaton recognising the set of factors of x. The leaves of the trie correspond to the minimal absent words for x, while the internal states are those of the factor automaton. Since the factor automaton of a word x has less than $2|x|$ states (for details, see [18]), this provides a representation of the minimal absent words of a word of length n in space $\mathcal{O}(n)$.

2.1. Suffix array and suffix tree

We denote by SA the *suffix array* of a non-empty word y of length n. SA is an integer array of size n storing the starting positions of all (lexicographically) sorted non-empty suffixes of y, i.e. for all $1 \leq r < n$ we have $y[\mathsf{SA}[r-1]\,..\,n-1] < y[\mathsf{SA}[r]\,..\,n-1]$ [19]. Let $\mathsf{lcp}(r, s)$ denote the length of the longest common prefix between $y[\mathsf{SA}[r]\,..\,n-1]$ and $y[\mathsf{SA}[s]\,..\,n-1]$ for all positions r, s on y, and 0 otherwise. We denote by LCP the *longest common prefix* array of y defined by $\mathsf{LCP}[r] = \mathsf{lcp}(r-1, r)$ for all $1 \leq r < n$, and $\mathsf{LCP}[0] = 0$. The inverse iSA of the array SA is defined by $\mathsf{iSA}[\mathsf{SA}[r]] = r$, for all $0 \leq r < n$. It is known that SA [20], iSA, and LCP [21] of a word of length n, over an integer alphabet, can be computed in time and space $\mathcal{O}(n)$.

The *suffix tree* $\mathcal{T}(y)$ of a non-empty word y of length n is a compact trie representing all suffixes of y. The nodes of the trie which become nodes of the suffix tree are called *explicit* nodes, while the other nodes are called *implicit*. Each edge of the suffix tree can be viewed as an upward maximal path of implicit nodes starting with an explicit node. Moreover, each node belongs to a unique path of that kind. Thus, each node of the trie can be represented in the suffix tree by the edge it belongs to and an index within the corresponding path. We let $\mathcal{L}(v)$ denote the *path-label* of a node v, i.e., the concatenation of the edge labels along the path from the root to v. We say that v is path-labelled $\mathcal{L}(v)$. Additionally, $\mathcal{D}(v) = |\mathcal{L}(v)|$ is used to denote the *string-depth* of node v. Node v is a *terminal* node if its path label is a suffix of y, that is, $\mathcal{L}(v) = y[i\,..\,n-1]$ for some $0 \leq i < n$; here v is also labelled with index i. It should be clear that each factor of y is uniquely represented by either an explicit or an implicit node of $\mathcal{T}(y)$. The *suffix-link* of a node v with path-label $\mathcal{L}(v) = \alpha w$ is a pointer to the node path-labelled w, where $\alpha \in \Sigma$ is a single letter and w is a word. The suffix-link of v is defined if v is an explicit node of $\mathcal{T}(y)$, different from the root. In any standard implementation of the suffix tree, we assume that each node of the suffix tree is able to access its parent. Note that once $\mathcal{T}(y)$ is constructed, it can be traversed

in a depth-first manner to compute the string-depth $\mathcal{D}(v)$ for each node v. Let u be the parent of v. Then the string-depth $\mathcal{D}(v)$ is computed by adding $\mathcal{D}(u)$ to the length of the label of edge (u, v). If v is the root, then $\mathcal{D}(v) = 0$. It is known that the suffix tree of a word of length n, over an integer alphabet, can be computed in time and space $\mathcal{O}(n)$ [22].

2.2. A measure of similarity between words based on minimal absent words

In what follows, as already proposed in [11], for every word y, the set of minimal absent words associated with y, denoted by $\mathcal{M}_y$, is represented as a set of tuples $\langle a, i, j\rangle$, where the corresponding minimal absent word x of y is defined by $x[0] = a$, $a \in \Sigma$, and $x[1\,..\,m-1] = y[i\,..\,j]$, where $j - i + 1 = m \geq 2$. It is known that if $|y| = n$ and $|\Sigma| = \sigma$, then $|\mathcal{M}_y| \leq \sigma n$ [8].

In [15], Chairungsee and Crochemore introduced a measure of similarity between two words x and y based on the notion of minimal absent words. Let $\mathcal{M}_x^\ell$ (respectively $\mathcal{M}_y^\ell$) denote the set of minimal absent words of length at most ℓ of x (respectively y). The authors made use of a length-weighted index to provide a measure of similarity between x and y, using their sample sets $\mathcal{M}_x^\ell$ and $\mathcal{M}_y^\ell$, by considering the length of each member in the symmetric difference $(\mathcal{M}_x^\ell \triangle \mathcal{M}_y^\ell)$ of the sample sets. For sample sets $\mathcal{M}_x^\ell$ and $\mathcal{M}_y^\ell$, they defined this index to be

$$\mathsf{LW}_\ell(x, y) = \sum_{w \in \mathcal{M}_x^\ell \triangle \mathcal{M}_y^\ell} \frac{1}{|w|^2}.$$

In this paper we consider a more general measure of similarity for two words x and y. It is based on the set $\mathcal{M}_x \triangle \mathcal{M}_y$, and is defined by

$$\mathsf{LW}(x, y) = \sum_{w \in \mathcal{M}_x \triangle \mathcal{M}_y} \frac{1}{|w|^2},$$

so without any restriction on the lengths of minimal absent words. The smaller the value of $\mathsf{LW}(x, y)$, the more similar we assume x and y to be. Note that $\mathsf{LW}(x, y)$ is affected by both the cardinality of $\mathcal{M}_x \triangle \mathcal{M}_y$ and the lengths of its elements; longer words in $\mathcal{M}_x \triangle \mathcal{M}_y$ contribute less in the value of $\mathsf{LW}(x, y)$ than shorter ones. Hence, intuitively, the shorter the words in $\mathcal{M}_x \triangle \mathcal{M}_y$, the more dissimilar x and y are.

We provide the following examples for illustration. Let $x = \mathtt{abaab}$ and $y = \mathtt{aabbbaa}$. We have $\mathcal{M}_x = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}\}$ and $\mathcal{M}_y = \{\mathtt{aaa}, \mathtt{bbbb}, \mathtt{aba}, \mathtt{abba}, \mathtt{bab}, \mathtt{baab}\}$. Thus,

$$\mathcal{M}_x \triangle \mathcal{M}_y = \{\mathtt{aaba}, \mathtt{aba}, \mathtt{abba}, \mathtt{baab}, \mathtt{bb}, \mathtt{bbbb}\},$$

so that

$$\mathsf{LW}(\mathtt{abaab}, \mathtt{aabbbaa}) = 4 \cdot \frac{1}{4^2} + \frac{1}{3^2} + \frac{1}{2^2} = \frac{11}{18}.$$

Similarly,

$$\mathsf{LW}(\mathtt{aaa}, \mathtt{bbb}) = \frac{17}{8}$$

and

$$\mathsf{LW}(\mathtt{aaa}, \mathtt{aaaa}) = \frac{41}{400}.$$

This measure of similarity aims at quantifying the distance between two words by means of their minimal absent words. In fact, we show here that this measure is consistent with the notion of *distance* (metric) in the mathematical sense.

Lemma 2. *$LW(x, y)$ is a metric on Σ^*.*

Proof. It is clear that *non-negativity*, $\mathsf{LW}(x, y) \geq 0$ for any $x, y \in \Sigma^*$, and *symmetry*, $\mathsf{LW}(x, y) = \mathsf{LW}(y, x)$ for any $x, y \in \Sigma^*$, are satisfied. In addition, we have from Lemma 1 that $\mathsf{LW}(x, y) = 0$ if and only if $x = y$, hence *identity* is also satisfied. Furthermore, given three sets A, B and C, we have by the properties of the symmetric difference that $A \triangle B \subseteq (A \triangle C) \cup (C \triangle B)$. Thus, given three words x, y and z, for every $w \in \mathcal{M}_x \triangle \mathcal{M}_y$ we have that $w \in \mathcal{M}_x \triangle \mathcal{M}_z$ or $w \in \mathcal{M}_z \triangle \mathcal{M}_y$ and therefore w contributes by $1/|w|^2$ to $\mathsf{LW}(x, y)$ and to one of $\mathsf{LW}(x, z)$ and $\mathsf{LW}(z, y)$. This shows that $\mathsf{LW}(x, y) \leq \mathsf{LW}(x, z) + \mathsf{LW}(z, y)$ for any $x, y, z \in \Sigma^*$ and so *triangle inequality* is also satisfied. □

Based on this similarity measure we consider the following problem:

MAW-SequenceComparison
Input: a word x of length m and a word y of length n
Output: $\mathsf{LW}(x, y)$.

In Section 4, we show that this problem can be solved in $\mathcal{O}(m + n)$-time and space.

2.3. Extension to circular words

We also consider the aforementioned problem for two circular words. A circular word of length m can be viewed as a traditional linear word which has the left- and right-most letters wrapped around. Under this notion, the same circular word can be seen as m different linear words, which would all be considered equivalent. More formally, given a word x of length m, we denote by $x^{\langle i \rangle} = x[i\,..\,m-1]x[0\,..\,i-1]$, $0 \leq i < m$, the i-th *rotation* of x, where $x^{\langle 0 \rangle} = x$. Given two words x and y, we define $x \sim y$ if there exists i, $0 \leq i < |x|$, such that $y = x^{\langle i \rangle}$. A *circular word* $\tilde{x}$ is a conjugacy class of the equivalence relation $\sim$. Given a circular word $\tilde{x}$, any (linear) word x in the equivalence class $\tilde{x}$ is called a *linearisation* of the circular word $\tilde{x}$. Conversely, given a linear word x, we say that $\tilde{x}$ is a *circularisation* of x if x is a linearisation of $\tilde{x}$.

The factorial closures of the languages generated by two rotations of the same word $x^{\langle i \rangle}$ and $x^{\langle j \rangle}$, i.e. the languages $\mathcal{F}_{(x^{\langle i \rangle})^*}$ and $\mathcal{F}_{(x^{\langle j \rangle})^*}$, coincide, so one can unambiguously define the (infinite) language $\mathcal{F}_{\tilde{x}}$ of factors of the circular word $\tilde{x}$ as the language $\mathcal{F}_{x^*}$, where x is any linearisation of $\tilde{x}$. This is coherent with the fact that a circular word can be seen as a word drawn on a circle, where there is no beginning and no end.

In Section 5, we give the definition of the set $\mathcal{M}_{\tilde{x}}$ of minimal absent words for a circular word $\tilde{x}$. We prove that the following problem can be solved within the same time and space complexity as its counterpart in the linear case.

MAW-CircularSequenceComparison
Input: a word x of length m and a word y of length n
Output: $\mathsf{LW}(\tilde{x}, \tilde{y})$, where $\tilde{x}$ and $\tilde{y}$ are circularisations of x and y, respectively.

2.4. Minimal absent words and q-grams

In Section 6, we present an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm that given a word x of length n computes $q(x)$, the largest integer for which each $q(x)$-gram of x is a $q(x)$-gram of some minimal absent word of x.

MAW-Qgrams
Input: a word x of length n
Output: $q(x)$

3. Tight asymptotic bound on the number of minimal absent words

An important property of the minimal absent words of a word x, that is at the basis of the algorithms presented in next sections, is that their number is linear in the size of x. Let x be a word of length n over an alphabet of size σ. In [8] it is shown that the total number of minimal absent words of x is smaller than or equal to σn. In the following lemma we show that $\mathcal{O}(\sigma n)$ is a tight asymptotic bound for the number of minimal absent words of x whenever $2 \leq \sigma \leq n$.

Lemma 3. *The upper bound $\mathcal{O}(\sigma n)$ on the number of minimal absent words of a word x of length n over an alphabet of size σ is tight if $2 \leq \sigma \leq n$.*

Proof. The total number of minimal absent words of x is smaller than or equal to σn [8]. Hence $\mathcal{O}(\sigma n)$ is an asymptotic upper bound for the number of minimal absent words of x. In what follows we provide examples to show that this bound is tight if $2 \leq \sigma \leq n$.

Let $\Sigma = \{a_1, a_2\}$, i.e. $\sigma = 2$, and consider the word $x = a_2 a_1^{n-2} a_2$ of length n. All words of the form $a_2 a_1^k a_2$, for $0 \leq k \leq n-3$, are minimal absent words of x. Hence x has at least $n-2 = \Omega(n)$ minimal absent words.

Let $\Sigma = \{a_1, a_2, a_3, \ldots, a_\sigma\}$ with $3 \leq \sigma \leq n$ and consider the word $x = a_2 a_1^k a_3 a_1^k \ldots a_i a_1^k a_{i+1} \ldots a_\sigma a_1^k a_1^m$, where $k = \lfloor \frac{n}{\sigma-1} \rfloor - 1$ and $m = n - (\sigma-1)(k+1)$. Note that $|x| = n$. Further note that $a_i a_1^j$ is a factor of x, for all $2 \leq i \leq \sigma$ and $0 \leq j \leq k$. Similarly, $a_1^j a_l$ is a factor of x, for all $3 \leq l \leq \sigma$ and $0 \leq j \leq k$. Thus, all proper factors of all the words in the set $S = \{a_i a_1^j a_l \mid 0 \leq j \leq k,\ 2 \leq i \leq \sigma,\ 3 \leq l \leq \sigma\}$ occur in x. However, the only words in S that occur in x are the ones of the form $a_i a_1^k a_{i+1}$, for $2 \leq i < \sigma$. Hence x has at least $(\sigma-1)(\sigma-2)(k+1) - (\sigma-2) = (\sigma-1)(\sigma-2)\lfloor \frac{n}{\sigma-1} \rfloor - (\sigma-2) = \Omega(\sigma n)$ minimal absent words. □

4. Sequence comparison using minimal absent words

The goal of this section is to provide the first linear-time and linear-space algorithm for computing the similarity measure (see Section 2) between two words defined over a fixed-sized alphabet. To this end, we consider two words x and y of lengths m and n, respectively, and their associated sets of minimal absent words, $\mathcal{M}_x$ and $\mathcal{M}_y$, respectively. Next, we give a linear-time and linear-space solution for the MAW-SequenceComparison problem.

It is known from [7] and [11] that we can compute the sets $\mathcal{M}_x$ and $\mathcal{M}_y$ in linear time and space relative to the two lengths m and n, respectively. The idea of our strategy consists of a merge sort on the sets $\mathcal{M}_x$ and $\mathcal{M}_y$, after they

have been ordered with the help of suffix arrays. To this end, we construct the suffix array associated to the word $w = xy$, together with the implicit LCP array corresponding to it. All of these structures can be constructed in time and space $\mathcal{O}(m+n)$, as mentioned earlier. Furthermore, we can preprocess the array LCP for range minimum queries, which we denote by $\mathsf{RMQ}_{\mathsf{LCP}}$ [23]. With the preprocessing complete, the longest common prefix LCE of two suffixes of w starting at positions p and q can be computed in constant time [24], using the formula

$$\mathsf{LCE}(w, p, q) = \mathsf{LCP}[\mathsf{RMQ}_{\mathsf{LCP}}(\mathsf{iSA}[p]+1, \mathsf{iSA}[q])].$$

Using these data structures, it is straightforward to sort the tuples in the sets $\mathcal{M}_x$ and $\mathcal{M}_y$ lexicographically. That is, two tuples, x_1 and x_2, are ordered such that the one being the prefix of the other comes first, or according to the letter following their longest common prefix, when the former is not the case. In our setting, the latter is always the case since $\mathcal{M}_x$ is prefix-free by the definition of minimal absent words. To do this, we simply go once through the suffix array associated with w and assign to each tuple in $\mathcal{M}_x$, respectively $\mathcal{M}_y$, the rank of the suffix starting at the position indicated by its second component, in the suffix array. Since sorting an array of n distinct integers, such that each is in $[0, n-1]$, can be done in time $\mathcal{O}(n)$ (using for example bucket sort) we can sort each of the sets of minimal absent words, taking into consideration the letter on the first position and these ranks. Thus, from now on, we assume that $\mathcal{M}_x = \{x_0, x_1, \ldots, x_k\}$ where x_i is lexicographically smaller than x_{i+1}, for $0 \le i < k \le \sigma m$, and $\mathcal{M}_y = \{y_0, y_1, \ldots, y_\ell\}$, where y_j is lexicographically smaller than y_{j+1}, for $0 \le j < \ell \le \sigma n$.

We now proceed with the merge. Thus, considering that we are analysing x_i from $\mathcal{M}_x$ and y_j from $\mathcal{M}_y$, we note that the two are equal if and only if $x_i[0] = y_j[0]$ and

$$\mathsf{LCE}(w, x_i[1], |x| + y_j[1]) \ge \ell, \text{ where } \ell = x_i[2] - x_i[1] = y_j[2] - y_j[1].$$

In other words, the two minimal absent words are equal if and only if their first letters coincide, they have equal length $\ell + 1$, and the longest common prefix of the suffixes of w starting at the positions indicated by the second components of the tuples has length at least ℓ.

Such a strategy will empower us with the means for constructing a new set $\mathcal{M}_{x,y} = \mathcal{M}_x \cup \mathcal{M}_y$. At each step, when analysing tuples x_i and y_j, we proceed as follows:

$$\mathcal{M}_{x,y} = \begin{cases} \mathcal{M}_{x,y} \cup \{x_i\}, & \text{and increment } i, & \text{if } x_i < y_j; \\ \mathcal{M}_{x,y} \cup \{y_j\}, & \text{and increment } j, & \text{if } x_i > y_j; \\ \mathcal{M}_{x,y} \cup \{x_i = y_j\}, & \text{and increment both } i \text{ and } j, & \text{if } x_i = y_j. \end{cases}$$

Observe that the last condition is saying that basically each common tuple is added only once to their union.

Furthermore, simultaneously with this construction we can also calculate the similarity between the words, given by $\mathsf{LW}(x, y)$, which is initially set to 0. Thus, at each step, when comparing the tuples x_i and y_j, we update

$$\mathsf{LW}(x, y) = \begin{cases} \mathsf{LW}(x, y) + \frac{1}{|x_i|^2}, & \text{and increment } i, & \text{if } x_i < y_j; \\ \mathsf{LW}(x, y) + \frac{1}{|y_j|^2}, & \text{and increment } j, & \text{if } x_i > y_j; \\ \mathsf{LW}(x, y), & \text{and increment both } i \text{ and } j, & \text{if } x_i = y_j. \end{cases}$$

We impose the increment of both i and j in the case of equality as in this case we only look at the symmetric difference between the sets of minimal absent words.

As all these operations take constant time and we perform them once per each tuple in $\mathcal{M}_x$ and $\mathcal{M}_y$, it is easily concluded that the whole operation takes, in the case of a fixed-sized alphabet, time and space $\mathcal{O}(m+n)$. Thus, we can compute the symmetric difference between the *complete* sets of minimal absent words, as opposed to [15], of two words defined over a fixed-sized alphabet, in linear time and space with respect to the lengths of the two words. We hence obtain the following result:

Theorem 4. *Problem* MAW-SequenceComparison *can be solved in time and space* $\mathcal{O}(m+n)$.

5. Circular sequence comparison using minimal absent words

In this section we extend the notion of minimal absent words to circular words. Recall from Section 2 that, given a circular word $\tilde{x}$, the set $\mathcal{F}_{\tilde{x}}$ of factors of $\tilde{x}$ is defined as the (infinite) set $\mathcal{F}_{x^*}$, where x is any linearisation of x. We therefore define the set $\mathcal{M}_{\tilde{x}}$ of minimal absent words of the circular word $\tilde{x}$ as the set of minimal absent words of the language $\mathcal{F}_{x^*}$, where x is any linearisation of x.

For instance, let $x = \mathtt{aabbabb}$. Then we have

$$\mathcal{M}_{\tilde{x}} = \{\mathtt{aaa}, \mathtt{aba}, \mathtt{bbb}, \mathtt{aabbaa}, \mathtt{babbab}\}.$$

Although $\mathcal{F}_{x^*}$ is an infinite language, the set $\mathcal{M}_{\tilde{x}} = \mathcal{M}_{\mathcal{F}_{x^*}}$ of minimal absent words of $\tilde{x}$ is always finite. More precisely, we have the following structural lemma (see also [25]).

Lemma 5. *Let $\tilde{x}$ be a circular word and x any linearisation of $\tilde{x}$. Then*

$$\mathcal{M}_{\tilde{x}} = \mathcal{M}_{xx}^{|x|}. \tag{3}$$

That is, the minimal absent words of the circular word $\tilde{x}$ are precisely the minimal absent words of the (linear) word xx whose length is not greater than the length of x, where x is any linearisation of $\tilde{x}$.

Proof. If aub, with $a, b \in \Sigma$ and $u \in \Sigma^*$, is an element in $\mathcal{M}_{xx}^{|x|}$, then clearly $aub \in \mathcal{M}_{\mathcal{F}_{x^*}} = \mathcal{M}_{\tilde{x}}$.

Conversely, let aub, with $a, b \in \Sigma$ and $u \in \Sigma^*$, be an element in $\mathcal{M}_{\tilde{x}} = \mathcal{M}_{\mathcal{F}_{x^*}}$. Then $aub \notin \mathcal{F}_{x^*}$, while $au, ub \in \mathcal{F}_{x^*}$. So, there exists a letter $\bar{b}$ different from b such that $au\bar{b} \in \mathcal{F}_{x^*}$ and a letter $\bar{a}$ different from a such that $\bar{a}ub \in \mathcal{F}_{x^*}$. Therefore, $au, \bar{a}u, ub, u\bar{b} \in \mathcal{F}_{x^*}$. Any word of length at least $|x| - 1$ cannot be extended to the right nor to the left by different letters in $\mathcal{F}_{x^*}$ as such factors would yield two rotations of x with different letter multiplicities. Hence $|aub| \leq |x|$. Since au and ub are factors of some rotation of x, we have $au, ub \in \mathcal{F}_{xx}$, whence $aub \in \mathcal{M}_{xx}$. □

The equality (3) was first introduced as the definition of the set of minimal absent words of a circular word in [26].

Recall that a word x is *a power* of a word y if there exists a positive integer k, $k > 1$, such that x is expressed as k consecutive concatenations of y, denoted by $x = y^k$. Conversely, a word x is *primitive* if $x = y^k$ implies $k = 1$. Notice that a word is primitive if and only if any of its rotations is. We can therefore extend the definition of primitivity to circular words. The definition of $\mathcal{M}_{\tilde{x}}$ does not allow one to uniquely reconstruct $\tilde{x}$ from $\mathcal{M}_{\tilde{x}}$, unless $\tilde{x}$ is known to be primitive, since it is readily verified that $\mathcal{F}_{x^*} = \mathcal{F}_{xx^*}$ and therefore also the minimal absent words of these two languages coincide. However, from the algorithmic point of view, this issue can be easily managed by storing the length $|x|$ of a linearisation x of $\tilde{x}$ together with the set $\mathcal{M}_{\mathcal{F}_{x^*}}$. Moreover, in most practical scenarios, for example when dealing with biological sequences, it is highly unlikely that the input circular word is not primitive.

Using the result of Lemma 5, we can easily extend the algorithm described in the previous section to the case of circular words. That is, given two circular words $\tilde{x}$ of length m and $\tilde{y}$ of length n, we can compute in time and space $\mathcal{O}(m+n)$ the distance $\mathsf{LW}(\tilde{x}, \tilde{y})$. We hence obtain the following result.

Theorem 6. *Problem* MAW-CircularSequenceComparison *can be solved in time and space $\mathcal{O}(m+n)$.*

6. From minimal absent words to q-grams

In this section we consider a word x of length n over an integer alphabet. Our aim is to provide a measure of the extent to which some *positive* information about x, the q-gram sets of x, exist unaltered in the set of minimal absent words of x, which can be seen as *negative* information about x. More specifically, we define $q(x)$ as the largest integer for which each $q(x)$-gram of x is a $q(x)$-gram of some minimal absent word of x. Note that, for instance, the set of q-grams of x is used in molecular biology applications such as genome assembly [27].

Example 7. Consider the word $x = \mathtt{abaab}$ over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Its set of minimal absent words is $\mathcal{M}_x = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}\}$. The set of 2-grams of x is $\{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}\}$ and, as can be easily seen, each of them is a factor of some word in $\mathcal{M}_x$. The set of 3-grams of x is $\{\mathtt{aab}, \mathtt{aba}, \mathtt{baa}\}$ and we observe that $\mathtt{baa}$ is not a factor of any of the words in $\mathcal{M}_x$. We can hence conclude that in this case $q(x) = 2$.

We present a non-trivial $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm to compute $q(x)$.

6.1. Useful properties

Let $h(x)$ be the length of a shortest factor of x that occurs only once in x. In addition, let $t(x)$ be the length of a shortest infix (factor that is not a prefix nor a suffix) of x that occurs only once in x.

Following the proof of [28, Proposition 10], any factor of a word x that occurs more than once in x is a factor of some minimal absent word of x and hence $q(x) \geq h(x) - 1$.

Lemma 8. *For any word x it holds that $q(x) \leq t(x) + 1$.*

Proof. Consider any non-empty infix u of x that occurs only once and suppose it is preceded by letter a and followed by letter b. Then aub can not be a factor of any of the minimal absent words of x as the largest infix of any minimal absent word of x must occur at least twice in x, once in an occurrence of the largest proper prefix of this minimal absent word in x and once in an occurrence of its larger proper suffix in x. Note that $au \neq ub$, since otherwise $aub = a^{2+|u|}$ and then u does not occur only once in x. It thus follows that $q(x) \leq t(x) + 1$. □

Fact 9. *We can compute $h(x)$ and $t(x)$ — and hence obtain the relevant bounds for $q(x)$ — in time $\mathcal{O}(n)$ for a word x of length n.*

Remark 10. The relation between minimum unique substrings and maximum repeats has been investigated in [29].

Note that all 1-grams a_i that occur in x are trivially contained in some minimal absent word of the form a_i^k for some k, so in what follows we assume that the factors of x, for which we want to examine when they are factors of some minimal absent word of x, are of the form aub, where a and b are (not necessarily distinct) letters and u a (possibly empty) word. It is clear that any such factor aub of x occurring only once can not be a minimal absent word itself. In addition, following the proof of Lemma 8, it can not be an infix of a minimal absent word. In the following lemma we provide a necessary and sufficient condition for aub to occur as a prefix of some minimal absent word of x.

Lemma 11. *Let aub, with $a, b \in \Sigma$ and u a word, be a factor occurring only once in a word x. The two following statements are equivalent:*

1. *aub is a prefix of some minimal absent word of x;*
2. *ub occurs at least twice in x and if $j_1 < j_2 < \ldots < j_k$ are the starting positions of its occurrences, with $j_m - 1$ being the starting position of the occurrence of aub, at least one of the following holds: (i) $x[j_m + k] \neq x[j_i + k]$ for some i, k such that $j_m + k \leq n - 1$ and $j_i + k \leq n - 1$; (ii) $m \neq 1$.*

Proof. (1. $\Rightarrow$ 2.): Consider a word aub occurring just once in x and appearing as a prefix of some minimal absent word. Firstly aub is not itself a minimal absent word, so any minimal absent word that has aub as a prefix must be of the form $aubvd$, where v is a possibly empty word and d a letter. The existence of this minimal absent word means that $ubvd$ occurs in x and it is not preceded by a (so ub occurs at least twice in x) and that $aubv$ occurs in x and either:

(i) it is followed by a letter $c \neq d$, or
(ii) $aubv$ is a suffix of x.

(1. $\Leftarrow$ 2.): If (i) holds, then for any minimal such k we have that $x[j_m - 1 \,..\, j_m + k - 1]x[j_i + k]$ is a minimal absent word. If (i) does not hold, but (ii) holds, then $x[j_m \,..\, n - 1]x[j_1 + n - j_m]$ is a minimal absent word. □

Similarly, whether aub is a suffix of some minimal absent word of x depends on the extensions of $\mathsf{rev}(u)a$ in $\mathsf{rev}(x)$.

6.2. Computing $q(x)$

In this section we present Algorithm MawToQgrams that, given a word x of length n over an integer alphabet, computes $q(x)$ in time and space $\mathcal{O}(n)$. The algorithm first creates the suffix trees of x and $\mathsf{rev}(x)$ and then preprocesses them in time $\mathcal{O}(n)$. The preprocessing phase for each tree is a depth-first search traversal which allows us to store in each node v a boolean variable $\mathcal{B}(v)$ which indicates if there is any branching in the subtree rooted at v and a variable $\mathcal{S}(v)$ indicating the starting position of the first occurrence of $\mathcal{L}(v)$ in x. The latter can be done in time $\mathcal{O}(n)$ since we store the starting position of the suffix corresponding to each terminal node while constructing the suffix tree. Algorithm MawToQgrams then calls Routines InfixBound, PrefixBound and SuffixBound to compute $q(x)$.

```
MawToQgrams(x)
 1  T(x) ← SuffixTree(x)
 2  T(rev(x)) ← SuffixTree(rev(x))
 3  for each node v ∈ T(x) do
 4       B(v) ← true if there is any branching below v and false otherwise
 5       S(v) ← Starting position of the first occurrence of L(v) in x
 6  for each node v ∈ T(rev(x)) do
 7       B(v) ← true if there is any branching below v and false otherwise
 8       S(v) ← Starting position of the first occurrence of L(v) in x
 9  q ← InfixBound(x)
10  q ← PrefixBound(x, q)
11  q ← SuffixBound(x, q)
12  return q
```

As we have already seen, all the factors of x that occur more than once in x also occur in some minimal absent word of x. Hence our aim is to identify a shortest factor of x that is not a factor of any of the minimal absent words of x.

We first present Routine Test that, given as inputs i and j, tests if the factor $x[i \,..\, j]$ of x that occurs only once in x also occurs in some minimal absent word of x. Let $x[i \,..\, j] = aub$, where $a, b \in \Sigma$ and u is a word. The routine first checks if

$x[i\,..\,j]$ occurs as a prefix of some minimal absent word of x by checking statement (2) of Lemma 11 as follows. It considers the node of $\mathcal{T}(x)$ with path-label $x[i+1\,..\,j] = ub$; note that in the pseudocode this node is denoted by $\textsc{Node}(\mathcal{T}(x))(i+1, j)$. If this node is explicit, then it is named v, while if it is implicit, then the destination of the edge it is on is named v. The routine then checks in time $\mathcal{O}(1)$ if $\mathcal{B}(v)$ is `true` or if $\mathcal{S}(v) \leq i$. If this is the case, then aub is a factor of some minimal absent word and the test returns `true`. Otherwise, the analogous check is performed for $\mathsf{rev}(x)[n-j\,..\,n-i-1] = \mathsf{rev}(u)a$ in $\mathcal{T}(\mathsf{rev}(x))$. If both checks are unsuccessful, then the routine returns `false`. We discuss how to efficiently obtain the desired nodes later in this section.

```
Test(i,j)
 1  v ← Node(T(x))(i + 1, j)
 2  if IsImplicit(v) then
 3       (v1, v2) ← Edge(v)
 4       v ← v2
 5  if B(v) = true or S(v) ≤ i then
 6       return true
 7  v ← Node(T(rev(x)))(n − j, n − i − 1)
 8  if IsImplicit(v) then
 9       (v1, v2) ← Edge(v)
10       v ← v2
11  if B(v) = true or S(v) ≤ n − j − 1 then
12       return true
13  return false
```

Now note that the factors of x that occur only once in x are the labels of the leaves and of the implicit nodes on the edges between internal nodes and leaves in the suffix tree. Hence, if node u is a leaf with $\mathcal{L}(u) = x[i\,..\,n-1]$, then $x[i\,..\,\mathcal{D}(parent(u)) + 1]$ corresponds to the shortest unique factor of x occurring at i. We can thus find $t(x)$ and all the infixes of x of a given length that occur only once in x in time $\mathcal{O}(n)$. We can also obtain the shortest unique prefix and the shortest unique suffix of x in time $\mathcal{O}(1)$.

Routine InfixBound first computes all unique infixes of x of length $t(x)$ and tests if there is any of them that does not occur in any minimal absent word of x, in which case we have that $q(x) \leq t(x) - 1$. If this is not the case, the routine computes all unique infixes of x of length $t(x) + 1$ and tests if there is any of them that does not occur in any minimal absent word of x, in which case we have that $q(x) \leq t(x)$. Otherwise, we use the bound $q(x) \leq t(x) + 1$ shown in Lemma 8, and hence do not have to increment again.

```
InfixBound(x)
 1  ℓ ← t(x)
 2  I ← all (i, j) such that x[i .. j] is a unique infix of length ℓ
 3  for each (i, j) ∈ I do
 4       if Test(i, j) = false then
 5            return ℓ − 1
 6  I′ ← all (i, j) such that x[i .. j] is a unique infix of length ℓ + 1
 7  for each (i, j) ∈ I′ do
 8       if Test(i, j) = false then
 9            return ℓ
10  return ℓ + 1
```

Finally, we also perform the same test for the prefixes and suffixes of x that occur only once and their length is smaller than the bound we have at that point. This is done by Routines PrefixBound and SuffixBound. We can then conclude on the value of $q(x)$.

```
PrefixBound(x,q)
1  p ← Length of shortest unique prefix of x
2  while p ≤ q do
3       if Test(0, p − 1) = false then
4            return p − 1
5       p ← p + 1
6  return q
```

SuffixBound(x,q)
1 $s \leftarrow$ Length of shortest unique suffix of x
2 **while** $s \leq q$ **do**
3 **if** Test$(n-s, n-1) =$ `false` **then**
4 **return** $s-1$
5 $s \leftarrow s+1$
6 **return** q

We now discuss how to answer the queries Node$(\mathcal{T}(x))(i+1, j)$ in line 1 of Test in time $\mathcal{O}(n)$ in total. We first discuss how to answer the queries asked within Routine InfixBound. While computing sets $\mathcal{I}$ and $\mathcal{I}'$, alongside the pair (i, j), we also store a pointer to the deepest explicit ancestor $v_{i,j}$ of the node with path-label $x[i\,..\,j]$. We can do this in time $\mathcal{O}(n)$ due to how we compute $\mathcal{I}$ and $\mathcal{I}'$. We have that $\mathcal{D}(v_{i,j}) = t(x) - 1 = j - i$ for $(i, j) \in \mathcal{I}$ and $\mathcal{D}(v_{i,j}) = t(x) - 1 = j - i - 1$ or $\mathcal{D}(v_{i,j}) = t(x) = j - i$ for $(i, j) \in \mathcal{I}'$. Following the suffix-link from the explicit node $v_{i,j}$ we retrieve the node with path-label $x[i+1\,..\,j-1]$ or the node with path-label $x[i+1\,..\,j-2]$. We then only need to answer at most two child queries for each such node to obtain the node with path-label $x[i+1\,..\,j]$. Answering such queries on-line bears the cost of $\mathcal{O}(\log\sigma)$ per query for integer alphabets or that of non-determinism if we make use of perfect hashing to store the edges at every node of $\mathcal{T}(x)$ [30]. We instead answer these queries off-line: it is well-known that we can answer q child queries off-line during a depth-first traversal of the suffix tree in $\mathcal{O}(n+q)$ deterministic time by first sorting the queries at each node of $\mathcal{T}(x)$. We first answer one child query per pair (i, j) in a batch and then the potential second ones in another batch. The total time required for this is $\mathcal{O}(n)$. Routine PrefixBound only considers nodes with path-labels of the form $x[1\,..\,h]$, which can be found by following the edges upwards from the node with path-label $x[1\,..\,n-1]$. Routine SuffixBound only considers terminal nodes to which we can afford to store pointers while creating $\mathcal{T}(x)$. We answer the respective queries for $\mathcal{T}(\mathsf{rev}(x))$ (line 7 of Test) in a similar fashion. Finally, having the pointers to the required nodes, we perform all the tests off-line.

Alternatively, we can obtain a deterministic $\mathcal{O}(n)$-time solution by employing a data structure for a special case of Union-Find [31] — a detailed description of this technique can be found in the appendix of [32].

Theorem 12. *Problem* MAW-Qgrams *can be solved in time and space* $\mathcal{O}(n)$.

Proof. We build and preprocess the suffix trees of x and $\mathsf{rev}(x)$ in time and space $\mathcal{O}(n)$ [22]. Based on Lemma 8 we then have to perform the test for $\mathcal{O}(n)$ factors, which we can find in time $\mathcal{O}(n)$. The tests are performed in total time $\mathcal{O}(n)$ by finding the required nodes and using the preprocessed suffix trees to check statement (2) of Lemma 11. We only need extra space $\mathcal{O}(n)$ to store a representation of the computed factors and perform the tests. □

7. Implementation and applications

We implemented the algorithms presented in Section 4 and Section 5 as programme scMAW to perform pairwise sequence comparison for a set of sequences using minimal absent words. scMAW uses programme MAW [11] for linear-time and linear-space computation of minimal absent words using suffix array. scMAW was implemented in the C programming language and developed under GNU/Linux operating system. It takes, as input argument, a file in MultiFASTA format with the input sequences, and then any of the two methods, for *linear* or *circular* sequence comparison, can be applied. It then produces a file in PHYLIP format with the distance matrix as output. Cell $[x, y]$ of the matrix stores $\mathsf{LW}(x, y)$ (or $\mathsf{LW}(\tilde{x}, \tilde{y})$ for the circular case). The implementation is distributed under the GNU General Public License (GPL), and it is available at http://github.com/solonas13/maw, which is set up for maintaining the source code and the man-page documentation. Notice that *all* input datasets and the produced outputs referred to in this section are publicly maintained at the same web-site.

An important feature of the proposed algorithms is that they require space linear in the length of the sequences (see Theorem 4 and Theorem 6). Hence, we were also able to implement scMAW using the Open Multi-Processing (OpenMP) PI for shared memory multiprocessing programming to distribute the workload across the available processing threads without a large memory footprint.

7.1. Applications

Recently, there has been a number of studies on the biological significance of absent words in various species [5,33,34]. In [33], the authors presented dendrograms from dinucleotide relative abundances in sets of minimal absent words for prokaryotes and eukaryotic genomes. The analyses support the hypothesis that minimal absent words are inherited through a common ancestor, in addition to lineage-specific inheritance, only in vertebrates. Very recently, in [34], it was shown that there exist three minimal words in the Ebola virus genomes which are absent from human genome. The authors suggest that the identification of such species-specific sequences may prove to be useful for the development of both diagnosis and therapeutics.

In this section, we show a potential application of our results for the construction of dendrograms for DNA sequences with circular structure. Circular DNA sequences can be found in viruses, as plasmids in archaea and bacteria, and in the

Table 1
Accuracy measurements based on relative pairwise RF distance.

Dataset $<\alpha, \beta, \gamma, \delta, \epsilon>$	T_1 vs. T_3	T_2 vs. T_3
$<12, 2500, 0.05, 0.06, 0.04>$	100%	100%
$<12, 2500, 0.20, 0.06, 0.04>$	100%	88,88%
$<12, 2500, 0.35, 0.06, 0.04>$	100%	100%
$<25, 2500, 0.05, 0.06, 0.04>$	100%	100%
$<25, 2500, 0.20, 0.06, 0.04>$	100%	100%
$<25, 2500, 0.35, 0.06, 0.04>$	100%	100%
$<50, 2500, 0.05, 0.06, 0.04>$	100%	97,87%
$<50, 2500, 0.20, 0.06, 0.04>$	100%	97,87%
$<50, 2500, 0.35, 0.06, 0.04>$	100%	100%

mitochondria and plastids of eukaryotic cells. Circular sequence comparison thus finds applications in several contexts such as reconstructing phylogenies using viroids RNA [35] or Mitochondrial DNA (MtDNA) [36]. Conventional tools to align circular sequences could yield an incorrectly high genetic distance between closely-related species. Indeed, when sequencing molecules, the position where a circular sequence starts can be totally arbitrary. Due to this *arbitrariness*, a suitable rotation of one sequence would give much better results for a pairwise alignment [37,4]. In what follows, we demonstrate the power of minimal absent words to pave a path to resolve this issue by applying Lemma 5 and Theorem 6. Next we do not claim that a solid phylogenetic analysis is presented but rather an investigation for potential applications of our theoretical findings.

We performed the following experiment with synthetic data. First, we simulated a basic dataset of DNA sequences using INDELible [38]. The number of taxa, denoted by α, was set to 12; the length of the sequence generated at the root of the tree, denoted by β, was set to 2500bp; and the substitution rate, denoted by γ, was set to 0.05. We also used the following parameters: a deletion rate, denoted by δ, of 0.06 *relative* to substitution rate of 1; and an insertion rate, denoted by ϵ, of 0.04 *relative* to substitution rate of 1. The parameters were chosen based on the genetic diversity standard measures observed for sets of MtDNA sequences from primates and mammals [37]. We generated another instance of the basic dataset, containing one *arbitrary* rotation of each of the α sequences from the basic dataset. We then used this randomised dataset as input to scMAW by considering $\mathsf{LW}(\tilde{x}, \tilde{y})$ as the distance metric. The output of scMAW was passed as input to NINJA [39], an efficient implementation of neighbour-joining [40], a well-established hierarchical clustering algorithm for inferring dendrograms (trees). We thus used NINJA to infer the respective tree T_1 under the neighbour-joining criterion. We also inferred the tree T_2 by following the same pipeline, but by considering $\mathsf{LW}(x, y)$ as distance metric, as well as the tree T_3 by using the *basic* dataset as input of this pipeline and $\mathsf{LW}(\tilde{x}, \tilde{y})$ as distance metric. Hence, notice that T_3 represents the original tree. Finally, we computed the pairwise Robinson-Foulds (RF) distance [41] between: T_1 and T_3; and T_2 and T_3.

Let us define *accuracy* as the difference between 1 and the relative pairwise RF distance. We repeated this experiment by simulating different datasets $<\alpha, \beta, \gamma, \delta, \epsilon>$ and measured the corresponding accuracy. The results in Table 1 (see T_1 vs. T_3) suggest that by considering $\mathsf{LW}(\tilde{x}, \tilde{y})$ we can always re-construct the original tree even if the sequences have first been arbitrarily rotated (Lemma 5). This is not the case (see T_2 vs. T_3) if we consider $\mathsf{LW}(x, y)$. Notice that 100% accuracy denotes a (relative) pairwise RF distance of 0.

8. Final remarks

In this paper, complementary to measures that refer to the composition of sequences in terms of their constituent patterns, we considered sequence comparison using minimal absent words, information about what does not occur in the sequences. We presented the first linear-time and linear-space algorithm to compare two sequences by considering *all* their minimal absent words. In the process, we presented some results of combinatorial interest, and also extended the proposed techniques to circular sequences. The power of minimal absent words is highlighted by the fact that they provide a tool for sequence comparison that is as efficient for circular as it is for linear sequences; whereas this is not the case, for instance, using the general edit distance model [42]. In addition, we presented a linear-time and linear-space algorithm that given a word x computes the largest integer $q(x)$ for which each $q(x)$-gram of x is a $q(x)$-gram of some minimal absent word of x. Finally, a preliminary experimental study shows the potential of our theoretical findings with regard to alignment-free sequence comparison using negative information.

Acknowledgments

We warmly thank Alice Héliou (École Polytechnique) for her inestimable code contribution and Antonio Restivo (Università di Palermo) for useful discussions. We also thank the anonymous reviewers for their constructive comments which greatly improved the presentation of the paper. Gabriele Fici's work was supported by the PRIN 2010/2011 project #"Automi e Linguaggi Formali: Aspetti Matematici e Applicativi" of the Italian Ministry of Education (MIUR) and by the "National Group for Algebraic and Geometric Structures, and their Applications" (GNSAGA – INdAM). Robert Mercaş's work was supported by a #Newton Fellowship of the Royal Society. Solon P. Pissis's work was supported by a Research Grant (#RG130720) awarded by the Royal Society.

References

[1] M. Crochemore, G. Fici, R. Mercaş, S.P. Pissis, Linear-time sequence comparison using minimal absent words & applications, in: LATIN, in: LNCS, vol. 9644, Springer, Berlin Heidelberg, 2016, pp. 334–346.
[2] S. Vinga, J. Almeida, Alignment-free sequence comparison—a review, Bioinformatics 19 (2003) 513–523.
[3] M. Domazet-Lošo, B. Haubold, Efficient estimation of pairwise distances between genomes, Bioinformatics 25 (24) (2009) 3221–3227.
[4] R. Grossi, C.S. Iliopoulos, R. Mercaş, N. Pisanti, S.P. Pissis, A. Retha, F. Vayani, Circular sequence comparison: algorithms and applications, Algorithms Mol. Biol. 11 (2016) 12.
[5] C. Acquisti, G. Poste, D. Curtiss, S. Kumar, Nullomers: really a matter of natural selection?, PLoS ONE 2 (10) (2017).
[6] M. Béal, F. Mignosi, A. Restivo, M. Sciortino, Forbidden words in symbolic dynamics, Adv. Appl. Math. 25 (2) (2000) 163–193.
[7] M. Crochemore, F. Mignosi, A. Restivo, Automata and forbidden words, Inf. Process. Lett. 67 (1998) 111–117.
[8] F. Mignosi, A. Restivo, M. Sciortino, Words and forbidden factors, Theor. Comput. Sci. 273 (1–2) (2002) 99–117.
[9] A.J. Pinho, P.J.S.G. Ferreira, S.P. Garcia, J.M.O.S. Rodrigues, On finding minimal absent words, BMC Bioinform. 10 (1) (2009).
[10] H. Fukae, T. Ota, H. Morita, On fast and memory-efficient construction of an antidictionary array, in: ISIT, IEEE, 2012, pp. 1092–1096.
[11] C. Barton, A. Heliou, L. Mouchard, S.P. Pissis, Linear-time computation of minimal absent words using suffix array, BMC Bioinform. 15 (2014) 388.
[12] C. Barton, A. Heliou, L. Mouchard, S.P. Pissis, Parallelising the computation of minimal absent words, in: PPAM, in: LNCS, vol. 9574, 2015, pp. 243–253.
[13] D. Belazzougui, F. Cunial, J. Kärkkäinen, V. Mäkinen, Versatile succinct representations of the bidirectional Burrows–Wheeler transform, in: ESA, in: LNCS, vol. 8125, 2013, pp. 133–144.
[14] A. Heliou, S.P. Pissis, S.J. Puglisi, emMAW: computing minimal absent words in external memory, Bioinformatics 33 (17) (2017) 2746–2749.
[15] S. Chairungsee, M. Crochemore, Using minimal absent words to build phylogeny, Theor. Comput. Sci. 450 (2012) 109–116.
[16] M. Crochemore, A. Heliou, G. Kucherov, L. Mouchard, S.P. Pissis, Y. Ramusat, Minimal absent words in a sliding window and applications to on-line pattern matching, in: FCT, in: LNCS, vol. 10472, Springer, Berlin Heidelberg, 2017, pp. 164–176.
[17] G. Fici, Minimal Forbidden Words and Applications, Ph.D. thesis, Université de Marne-la-Vallée, 2006.
[18] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, New York, NY, USA, 2007.
[19] U. Manber, E.W. Myers, Suffix arrays: a new method for on-line string searches, SIAM J. Comput. 22 (5) (1993) 935–948.
[20] G. Nong, S. Zhang, W.H. Chan, Linear suffix array construction by almost pure induced-sorting, in: DCC, IEEE, 2009, pp. 193–202.
[21] J. Fischer, Inducing the LCP-array, in: WADS, in: LNCS, vol. 6844, 2011, pp. 374–385.
[22] M. Farach, Optimal suffix tree construction with large alphabets, in: FOCS, 1997, pp. 137–143.
[23] J. Fischer, V. Heun, Space-efficient preprocessing schemes for range minimum queries on static arrays, SIAM J. Comput. 40 (2) (2011) 465–492.
[24] L. Ilie, G. Navarro, L. Tinta, The longest common extension problem revisited and applications to approximate string searching, J. Discret. Algorithms 8 (4) (2010) 418–428.
[25] G. Fici, A. Restivo, L. Rizzo, Minimal forbidden factors of circular words, Theor. Comput. Sci. https://doi.org/10.1016/j.tcs.2018.05.037.
[26] T. Ota, H. Morita, On a universal antidictionary coding for stationary ergodic sources with finite alphabet, in: ISITA, IEEE, 2014, pp. 294–298.
[27] P.A. Pevzner, H. Tang, M.S. Waterman, An Eulerian path approach to DNA fragment assembly, Proc. Natl. Acad. Sci. 98 (17) (2001) 9748–9753.
[28] G. Fici, F. Mignosi, A. Restivo, M. Sciortino, Word assembly through minimal forbidden words, Theor. Comput. Sci. 359 (1) (2006) 214–230.
[29] L. Ilie, W.F. Smyth, Minimum unique substrings and maximum repeats, Fundam. Inform. 110 (1–4) (2011) 183–195.
[30] M.L. Fredman, J. Komlós, E. Szemerédi, Storing a sparse table with O(1) worst case access time, J. ACM 31 (3) (1984) 538–544.
[31] H.N. Gabow, R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. Syst. Sci. 30 (2) (1985) 209–221.
[32] C. Barton, T. Kociumaka, C. Liu, S.P. Pissis, J. Radoszewski, Indexing weighted sequences: neat and efficient, CoRR, arXiv:1704.07625v1.
[33] S.P. Garcia, O.J. Pinho, J.M.O.S. Rodrigues, C.A.C. Bastos, P.J.S.G. Ferreira, Minimal absent words in prokaryotic and eukaryotic genomes, PLoS ONE 6 (2011).
[34] R.M. Silva, D. Pratas, L. Castro, A.J. Pinho, P.J.S.G. Ferreira, Three minimal sequences found in Ebola virus genomes and absent from human DNA, Bioinformatics 31 (15) (2015) 2421–2425.
[35] A. Mosig, I.L. Hofacker, P.F. Stadler, Comparative analysis of cyclic sequences: viroids and other small circular RNAs, in: GCB, in: LNI, vol. 83, 2006, pp. 93–102.
[36] A. Goios, L. Pereira, M. Bogue, V. Macaulay, A. Amorim, mtDNA phylogeny and evolution of laboratory mouse strains, Genome Res. 17 (3) (2007) 293–298.
[37] C. Barton, C.S. Iliopoulos, R. Kundu, S.P. Pissis, A. Retha, F. Vayani, Accurate and efficient methods to improve multiple circular sequence alignment, in: SEA, in: LNCS, vol. 9125, 2015, pp. 247–258.
[38] W. Fletcher, Z. Yang, INDELible: a flexible simulator of biological sequence evolution, Mol. Biol. Evol. 26 (8) (2009) 1879–1888.
[39] T.J. Wheeler, Large-scale neighbor-joining with NINJA, in: WABI, in: LNCS, vol. 5724, 2009, pp. 375–389.
[40] N. Saitou, M. Nei, The neighbor-joining method: a new method for reconstructing phylogenetic trees, Mol. Biol. Evol. 4 (4) (1987) 406–425.
[41] D. Robinson, L. Fould, Comparison of phylogenetic trees, Math. Biosci. 53 (1–2) (1981) 131–147.
[42] M. Maes, On a cyclic string-to-string correction problem, Inf. Process. Lett. 35 (2) (1990) 73–78.