



Constructing Antidictionaries of Long Texts in Output-Sensitive Space

Lorraine A.K. Ayad¹ · Golnaz Badkobeh² · Gabriele Fici³  · Alice Héliou⁴ ·
Solon P. Pissis^{5,6}

Accepted: 6 November 2020 / Published online: 14 December 2020
© The Author(s) 2020

Abstract

A word x that is absent from a word y is called *minimal* if all its proper factors occur in y . Given a collection of k words y_1, \dots, y_k over an alphabet Σ , we are asked to compute the set $M_{\{y_1, \dots, y_k\}}^\ell$ of minimal absent words of length at most ℓ of the collection $\{y_1, \dots, y_k\}$. The set $M_{\{y_1, \dots, y_k\}}^\ell$ contains all the words x such that x is absent from all the words of the collection while there exist i, j , such that the maximal proper suffix of x is a factor of y_i and the maximal proper prefix of x is a factor of y_j . In data compression, this corresponds to computing the antidictionary of k documents. In bioinformatics, it corresponds to computing words that are absent from a genome of k chromosomes. Indeed, the set M_y^ℓ of minimal absent words of a word y is equal to $M_{\{y_1, \dots, y_k\}}^\ell$ for any decomposition of y into a collection of words y_1, \dots, y_k such that there is an overlap of length at least $\ell - 1$ between any two consecutive words in the collection. This computation generally requires $\Omega(n)$ space for $n = |y|$ using any of the plenty available $\mathcal{O}(n)$ -time algorithms. This is because an $\Omega(n)$ -sized text index is constructed over y which can be impractical for large n . We do the identical computation incrementally using output-sensitive space. This goal is reasonable when $\|M_{\{y_1, \dots, y_N\}}^\ell\| = o(n)$, for all $N \in [1, k]$, where $\|S\|$ denotes the sum of the lengths of words in set S . For instance, in the human genome, $n \approx 3 \times 10^9$ but $\|M_{\{y_1, \dots, y_k\}}^{12}\| \approx 10^6$. We consider a constant-sized alphabet for stating our results. We show that all $M_{y_1}^\ell, \dots, M_{y_k}^\ell$ can be computed in $\mathcal{O}(kn + \sum_{N=1}^k \|M_{\{y_1, \dots, y_N\}}^\ell\|)$ total time using $\mathcal{O}(\text{MAXIN} + \text{MAXOUT})$ space, where MAXIN is the length of the longest word in $\{y_1, \dots, y_k\}$ and $\text{MAXOUT} = \max\{\|M_{\{y_1, \dots, y_N\}}^\ell\| : N \in [1, k]\}$. Proof-of-concept experimental results are also provided confirming our theoretical findings and justifying our contribution.

A preliminary version of this paper was presented at the Data Compression Conference 2019 [1].
Gabriele Fici is supported by MIUR-PRIN 2017 project “Algorithms, Data Structures and Combinatorics for Machine Learning”.

Extended author information available on the last page of the article.

Keywords Absent word · Antidictionary · String algorithm · Output sensitive algorithm · Data compression

1 Introduction

The word x is an *absent word* of the word y if it does not occur in y . The absent word x of y is called *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 M_y . The set of all minimal absent words of length at most ℓ of a word y is denoted by M_y^ℓ . For example, if $y = \text{abaab}$, then $M_y = \{\text{aaa}, \text{aaba}, \text{bab}, \text{bb}\}$ and $M_y^3 = \{\text{aaa}, \text{bab}, \text{bb}\}$.

The upper bound on the number of minimal absent words is $\mathcal{O}(\sigma n)$ [2], where σ is the size of the alphabet and n is the length of y , and this bound is tight for integer alphabets [3]; in fact, for large alphabets, such as when $\sigma \geq \sqrt{n}$, this bound is tight even for minimal absent words having the same length [4, 5].

State-of-the-art algorithms compute all minimal absent words of y in $\mathcal{O}(\sigma n)$ time [2, 6, 7] or in $\mathcal{O}(n + \|M_y\|)$ time [8, 9] for integer alphabets. There also exist space-efficient data structures based on the Burrows-Wheeler transform of y that can be applied for this computation [10, 11]. In many real-world applications of minimal absent words, such as in data compression [12–15], in sequence comparison [3, 9], in on-line pattern matching [16], or in identifying pathogen-specific signatures [17], only a subset of minimal absent words may be considered, and, in particular, the minimal absent words of length (at most) ℓ . Since, in the worst case, the number of minimal absent words of y is $\Theta(\sigma n)$, $\Omega(\sigma n)$ space is required to represent them explicitly. In [9], the authors presented an $\mathcal{O}(n)$ -sized data structure for outputting minimal absent words of a specific length in optimal time for integer alphabets.

The problem with existing algorithms for computing minimal absent words is that they make use of $\Omega(n)$ space and the same amount is required even if one is merely interested in the minimal absent words of length at most ℓ . This is because all of these algorithms must construct global data structures, such as the suffix array [6, 7], over the whole input. In theory, this problem can be addressed by using the external memory algorithm for computing minimal absent words presented in [18]. The I/O-optimal version of this algorithm, however, requires a large amount of external memory to build the global data structures for the input [19]. One could also use the algorithm of [20] that computes M_y^ℓ in $\mathcal{O}(n + \|M_y^\ell\|)$ time using $\mathcal{O}(\min\{n, \ell z\})$ space, where z is the size of the LZ77 factorization of y . This algorithm also requires constructing the truncated DAWG, a type of global data structure which could take space $\Omega(n)$. Thus, in this paper, we investigate whether M_y^ℓ can be computed efficiently in output-sensitive space.

Our approach consists in decomposing y into a collection of k words, with a suitable overlap of length $\ell - 1$ between any two consecutive words in the collection.

In fact, the definition of minimal absent word was originally given for languages (sets of words) closed under taking factors (called *factorial languages*). A word $x = \text{aub}$, with $a, b \in \Sigma$, is a minimal absent word of a given factorial language L over the alphabet Σ if x does not occur in any of the words of L but there exist $y_i, y_j \in L$

such that au is a factor of y_i and ub is a factor of y_j . The set of minimal absent words of a word y is precisely the set of minimal absent words of the language $\text{Fact}(y)$ of factors of y . That is, $M_y = M_{\text{Fact}(y)}$.

More generally, if $\{y_1, \dots, y_k\}$ is a collection of k words, $M_{\{y_1, \dots, y_k\}}$ is defined as the set of minimal absent words of $\text{Fact}(\{y_1, \dots, y_k\}) = \cup_{i=1}^k \text{Fact}(y_i)$. So, a word $x = aub$, with $a, b \in \Sigma$, is a minimal absent word of $\{y_1, \dots, y_k\}$ if and only if x is not a factor of any of the words in the collection but there exist i, j such that au is a factor of y_i and ub is a factor of y_j .

We have the following lemma:

Lemma 1 *Let y be a word of length n and let $\ell > 1$. Let $\{y_1, \dots, y_k\}$ be any decomposition of y with overlap $\ell - 1$ between any two consecutive words y_i, y_{i+1} . Then $M_y^\ell = M_{\{y_1, \dots, y_k\}}^\ell$.*

Proof By double inclusion. Let $x = aub$, with $a, b \in \Sigma$, be a minimal absent word of length $\leq \ell$ of y (if $|x| < 2$ the statement is trivial). Then x is not a factor of y and therefore cannot be a factor of $\{y_1, \dots, y_k\}$. Still, since y and $\{y_1, \dots, y_k\}$ have the same factors up to length $\ell - 1$, we have that au and ub are factors of $\{y_1, \dots, y_k\}$, hence x is a minimal absent word of $\{y_1, \dots, y_k\}$.

Conversely, let $x = aub$, with $a, b \in \Sigma$, be a minimal absent word of length $\leq \ell$ of $\{y_1, \dots, y_k\}$. If x was a factor of y , then since x is not a factor of any of the words in the collection, one has that au is a suffix of some y_i , but then because y_i and y_{i+1} have an overlap of length $\ell - 1 \geq |au|$, we would have that au is a prefix of y_{i+1} and it cannot be followed by b since aub does not belong to the set of factors of $\{y_1, \dots, y_k\}$. Contradiction. \square

By Lemma 1, we can state our problem as follows:

Problem 1 Given a collection of k words $\{y_1, \dots, y_k\}$ over an alphabet Σ and an integer $\ell > 1$, compute the set $M_{\{y_1, \dots, y_k\}}^\ell$ of all the words x of length at most ℓ , such that x is absent from all the words of the collection while there exists $1 \leq i, j \leq k$, such that the maximal proper suffix of x is a factor of y_i and the maximal proper prefix of x is a factor of y_j .

In data compression, this scenario corresponds to computing the antidictionary of k documents [12, 13]. In bioinformatics, it corresponds to computing words that are absent from a genome of k chromosomes. As discussed above, this computation generally requires $\Omega(n)$ space for $n = \sum_{N=1}^k |y_N|$. We do the same computation incrementally using output-sensitive space. This goal is reasonable when $\|M_{\{y_1, \dots, y_N\}}^\ell\| = o(n)$, for all $N \in [1, k]$, where $\|S\|$ denotes the sum of the lengths of words in set S . In the human genome, $n \approx 3 \times 10^9$ but $\|M_{\{y_1, \dots, y_k\}}^{12}\| \approx 10^6$, where k is the total number of chromosomes.

Our Results Antidictionary-based compressors work on $\Sigma = \{0, 1\}$ and in bioinformatics we have $\Sigma = \{A, C, G, T\}$; we thus consider a constant-sized alphabet for

stating our results. We consider the word RAM model with w -bit machine words, where $w = \Omega(\log n)$. We analyze algorithms in the worst case and measure space in terms of machine words.

We show that *all* $M_{y_1}^\ell, \dots, M_{\{y_1, \dots, y_k\}}^\ell$ can be computed in $\mathcal{O}(kn + \sum_{N=1}^k \|M_{\{y_1, \dots, y_N\}}^\ell\|)$ total time using $\mathcal{O}(\text{MAXIN} + \text{MAXOUT})$ space, where MAXIN is the length of the longest word in $\{y_1, \dots, y_k\}$ and $\text{MAXOUT} = \max\{\|M_{\{y_1, \dots, y_N\}}^\ell\| : N \in [1, k]\}$. Proof-of-concept experimental results are also provided confirming our theoretical findings and justifying our contribution.

Paper Organization Section 2 provides the necessary definitions and notation used throughout the paper. In Section 3, we prove several combinatorial properties, which form the basis of our new technique. In Sections 4 and 5, we present our main results. Our experimental results are presented in Section 6. Section 7 concludes the paper with some remarks for further investigation.

A preliminary version of this paper appeared as [1]. Compared to the preliminary version, we have extended the work by adding a simplified space-efficient version of the algorithm (see Section 4). We have also added additional experimental results using real-world datasets, which further substantiate our contribution.

2 Preliminaries

We generally follow [21]. An *alphabet* Σ is a finite ordered non-empty set of elements called *letters*. A *word* is a sequence of elements of Σ . The set of all words over Σ of length at most ℓ is denoted by $\Sigma^{\leq \ell}$. We fix a constant-sized alphabet Σ , i.e., $|\Sigma| = \mathcal{O}(1)$. Given a word $y = uxv$ over Σ , we say that u is a *prefix* of y , x is a *factor* (or subword) of y , and v is a *suffix* of y . We also say that y is a *superword* of x . A factor x of y is called *proper* if $x \neq y$. We let $\text{Fact}(y)$ denote the set of factors of word y .

Given a word y over Σ , the set M_y of *minimal absent words* (MAWs) of y is defined as

$$\{aub : a, b \in \Sigma, au \text{ and } ub \text{ are factors of } y \text{ but } aub \text{ is not}\} \\ \cup \{c \in \Sigma : c \text{ does not occur in } y\}.$$

Given a collection of k words y_1, \dots, y_k over Σ , the set $M_{\{y_1, \dots, y_k\}}$ of *minimal absent words of the collection* $\{y_1, \dots, y_k\}$ is defined as

$$\{aub : a, b \in \Sigma, \exists 1 \leq i, j \leq k, au \in \text{Fact}(y_i), ub \in \text{Fact}(y_j), aub \notin \bigcup_{N=1}^k \text{Fact}(y_N)\} \\ \cup \{c \in \Sigma : c \text{ does not occur in any of the words } y_i\}.$$

MAWs of length 1 of y can be easily found with a linear-time constant-space scan, hence in what follows we will only focus on the computation of MAWs of length at least 2.

The *suffix tree* $\mathcal{T}(y)$ of a non-empty word y of length n is the compact trie representing all suffixes of y [21]. The *branching* nodes of the trie as well as the *terminal* nodes, that correspond to non-empty suffixes of y , become *explicit* nodes of the suffix tree, while the other nodes are *implicit*. We let $\mathcal{L}(v)$ denote the *path-label* from the root node to node v . We say that node v is path-labeled $\mathcal{L}(v)$; i.e., the concatenation of the edge labels along the path from the root node to v . Additionally, $\mathcal{D}(v) = |\mathcal{L}(v)|$ is used to denote the *word-depth* of node v . A node v such that the path-label $\mathcal{L}(v) = y[i..n - 1]$, for some $0 \leq i \leq n - 1$, is *terminal* and is also labeled with index i . Each factor of y is uniquely represented by either an explicit or an implicit node of $\mathcal{T}(y)$ called its *locus*. The *suffix-link* of a node v with path-label $\mathcal{L}(v) = aw$ is a pointer to the node path-labeled w , where $a \in \Sigma$ is a single letter and w is a word. The suffix-link of v exists by construction if v is a non-root branching node of $\mathcal{T}(y)$.

The *generalized suffix tree* of a collection of words $\{y_1, \dots, y_N\}$, denoted by $\mathcal{T}(y_1, \dots, y_N)$, is the suffix tree of word $y_1\#_1 \dots y_N\#_N$, where $\#_1, \dots, \#_N$ are distinct end-markers not belonging to Σ [22].

The *matching statistics* of a word $x[0..|x| - 1]$ with respect to word y is an array $MS_x[0..|x| - 1]$, where $MS_x[i]$ is a pair (f_i, p_i) such that (i) $x[i..i + f_i - 1]$ is the longest prefix of $x[i..|x| - 1]$ that is a factor of y ; and (ii) $y[p_i..p_i + f_i - 1] = x[i..i + f_i - 1]$ [23]. $\mathcal{T}(y)$ can be constructed in time $\mathcal{O}(n)$, and, given $\mathcal{T}(y)$, we can compute MS_x in time $\mathcal{O}(|x|)$ [23].

3 Combinatorial Properties

For convenience, we consider the following setting. Let y_1, y_2 be two words over the alphabet Σ . Let ℓ be a positive integer and set $M_{y_1}^\ell = M_{y_1} \cap \Sigma^{\leq \ell}$ and $M_{y_2}^\ell = M_{y_2} \cap \Sigma^{\leq \ell}$. We want to construct $M_{\{y_1, y_2\}}^\ell = M_{\{y_1, y_2\}} \cap \Sigma^{\leq \ell}$. Let $x \in M_{\{y_1, y_2\}}^\ell$. We have two cases:

Case 1 : $x \in M_{y_1}^\ell \cup M_{y_2}^\ell$;

Case 2 : $x \notin M_{y_1}^\ell \cup M_{y_2}^\ell$.

The following auxiliary fact follows directly from the minimality property.

Fact 1 Word x is absent from word y if and only if x is a superword of a MAW of y .

For Case 1, we prove the following lemma.

Lemma 2 (Case 1) A word $x \in M_{y_1}^\ell$ (resp. $x \in M_{y_2}^\ell$) belongs to $M_{\{y_1, y_2\}}^\ell$ if and only if x is a superword of a word in $M_{y_2}^\ell$ (resp. in $M_{y_1}^\ell$).

Proof Let $x \in M_{y_1}^\ell$ (the case $x \in M_{y_2}^\ell$ is symmetric). Suppose first that x is a superword of a word in $M_{y_2}^\ell$, that is, there exists $v \in M_{y_2}^\ell$ such that v is a factor of x . If $v = x$, then $x \in M_{y_1}^\ell \cap M_{y_2}^\ell$ and therefore, using the definition of MAW, $x \in M_{\{y_1, y_2\}}^\ell$.

If v is a proper factor of x , then x is an absent word of y_2 and again, by definition of MAW, $x \in M_{\{y_1, y_2\}}^\ell$.

Suppose now that x is not a superword of any word in $M_{y_2}^\ell$. Then x is not absent in y_2 by Fact 1, and hence in y_3 , thus x cannot belong to $M_{\{y_1, y_2\}}^\ell$. \square

It should be clear that the statement of Lemma 2 implies, in particular, that all words in $M_{y_1}^\ell \cap M_{y_2}^\ell$ belong to $M_{\{y_1, y_2\}}^\ell$. Furthermore, Lemma 2 motivates us to introduce the *reduced set of MAWs* of y_1 with respect to y_2 as the set $R_{y_1}^\ell$ obtained from $M_{y_1}^\ell$ after removing those words that are superwords of words in $M_{y_2}^\ell$. The set $R_{y_2}^\ell$ is defined analogously.

Example 1 Let $y_1 = \text{abaab}$, $y_2 = \text{bbaaab}$ and $\ell = 5$. We have $M_{y_1}^\ell = \{\text{bb}, \text{aaa}, \text{bab}, \text{aaba}\}$ and $M_{y_2}^\ell = \{\text{bbb}, \text{aaaa}, \text{baab}, \text{aba}, \text{bab}, \text{abb}\}$.

The word bab is contained in $M_{y_1}^\ell \cap M_{y_2}^\ell$ so it belongs to $M_{\{y_1, y_2\}}^\ell$. The word $\text{aaba} \in M_{y_1}^\ell$ is a superword of $\text{aba} \in M_{y_2}^\ell$ hence $\text{aaba} \in M_{\{y_1, y_2\}}^\ell$. On the other hand, the words bbb , aaaa and abb are superwords of words in $M_{y_1}^\ell$, hence they belong to $M_{\{y_1, y_2\}}^\ell$. The remaining MAWs are not superwords of MAWs of the other word. The reduced sets are therefore $R_{y_1}^\ell = \{\text{bb}, \text{aaa}\}$ and $R_{y_2}^\ell = \{\text{baab}, \text{aba}\}$. In conclusion, we have for Case 1 that

$$M_{\{y_1, y_2\}}^\ell \cap (M_{y_1}^\ell \cup M_{y_2}^\ell) = \{\text{aaaa}, \text{bab}, \text{aaba}, \text{abb}, \text{bbb}\}.$$

We now investigate the set $M_{\{y_1, y_2\}}^\ell \setminus (M_{y_1}^\ell \cup M_{y_2}^\ell)$ (Case 2).

Fact 2 Let $x = \text{aub}$, $a, b \in \Sigma$, be such that $x \in M_{\{y_1, y_2\}}^\ell$ and $x \notin M_{y_1}^\ell \cup M_{y_2}^\ell$. Then au occurs in y_1 but not in y_2 and ub occurs in y_2 but not in y_1 , or vice versa.

The rationale for generating the reduced sets should become clear with the next lemma.

Lemma 3 (Case 2) *Let $x \in M_{\{y_1, y_2\}}^\ell \setminus (M_{y_1}^\ell \cup M_{y_2}^\ell)$. Then x has a prefix x_i in $R_{y_i}^\ell$ and a suffix x_j in $R_{y_j}^\ell$, for i, j such that $\{i, j\} = \{1, 2\}$.*

Proof Let $x = \text{aub}$, $a, b \in \Sigma$, be a word in $M_{\{y_1, y_2\}}^\ell \setminus (M_{y_1}^\ell \cup M_{y_2}^\ell)$. By Fact 2, au occurs in y_1 but not in y_2 and ub occurs in y_2 but not in y_1 , or vice versa. Let us assume the first case holds (the other case is symmetric). Since au does not occur in y_2 , we have by Fact 1 that there is a MAW $x_2 \in M_{y_2}^\ell$ that is a factor of au . Since ub occurs in y_2 , x_2 is not a factor of ub . Consequently, x_2 is a prefix of au .

Analogously, there is an $x_1 \in M_{y_1}^\ell$ that is a suffix of ub . Furthermore, x_1 and x_2 cannot be factors one of another. \square

Inspect Fig. 1 in this regard.

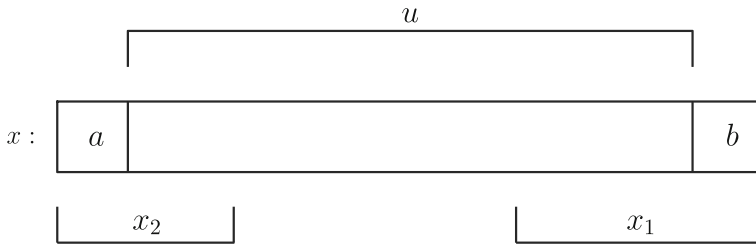


Fig. 1 x_2 occurs in y_1 but not in y_2 ; x_1 occurs in y_2 but not in y_1 ; therefore aub does not occur in $y_1\#y_2$. By construction, au occurs in y_1 and ub occurs in y_2 ; therefore aub is a Case 2 MAW

Example 2 Let $y_1 = abaab, y_2 = bbaaab$ and $\ell = 5$. Consider $x = abaaa \in M_{\{y_1, y_2\}}^\ell \setminus (M_{y_1}^\ell \cup M_{y_2}^\ell)$ (Case 2 MAW). We have that $abaa$ occurs in y_1 but not in y_2 and $baaa$ occurs in y_2 but not in y_1 . Since $abaa$ does not occur in y_2 , there is a MAW $x_2 \in R_{y_2}^\ell$ that is a factor of $abaa$. Since $baaa$ occurs in y_2, x_2 is not a factor of $baaa$. So x_2 is a prefix of $abaa$ and this is aba . Analogously, there is a MAW $x_1 \in R_{y_1}^\ell$ that is a suffix of $abaaa$ and this is aaa .

As a consequence of Lemma 3, in order to construct the set $M_{\{y_1, y_2\}}^\ell \setminus (M_{y_1}^\ell \cup M_{y_2}^\ell)$, we should consider all pairs (x_i, x_j) with x_i in $R_{y_i}^\ell$ and x_j in $R_{y_j}^\ell, \{i, j\} = \{1, 2\}$. In order to construct the final set $M_{\{y_1, \dots, y_k\}}^\ell$, we use incrementally Lemmas 2 and 3. We summarize the whole approach in the following general theorem, which forms the theoretical basis of our technique.

Theorem 1 Let $N > 1$, and let $x \in M_{\{y_1, \dots, y_N\}}^\ell$. Then, either $x \in M_{\{y_1, \dots, y_{N-1}\}}^\ell \cup M_{y_N}^\ell$ (Case 1 MAWs) or, otherwise, $x \in M_{y_i, y_N}^\ell \setminus (M_{y_i}^\ell \cup M_{y_N}^\ell)$ for some i . Moreover, in this latter case, x has a prefix in $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ and a suffix in $R_{y_N}^\ell$, or the converse, i.e., x has a prefix in $R_{y_N}^\ell$ and a suffix in $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ (Case 2 MAWs).

Proof Let $x \in M_{\{y_1, \dots, y_N\}}^\ell, |x| = m$ and $x \notin M_{\{y_1, \dots, y_{N-1}\}}^\ell \cup M_{y_N}^\ell$. Then,

1. $x \notin M_{\{y_1, \dots, y_{N-1}\}}^\ell$ and
2. $x \notin M_{y_N}^\ell$.

By the definition of MAW, $x[0..m - 2]$ and $x[1..m - 1]$ must be factors of some words in $\{y_1, \dots, y_N\}$. However, both cannot be factors of a word in $\{y_1, \dots, y_{N-1}\}$ and both cannot be factors of y_N . Therefore, we have one of the two cases:

- Case A: $x[0..m - 2]$ is factor of a word in $\{y_1, \dots, y_{N-1}\}$ but not of y_N and $x[1..m - 1]$ is a factor of y_N but not of any word in $\{y_1, \dots, y_{N-1}\}$.
- Case B: $x[0..m - 2]$ is factor of y_N but not of any word of the set $\{y_1, \dots, y_{N-1}\}$ and $x[1..m - 1]$ is a factor of a word in $\{y_1, \dots, y_{N-1}\}$ but not of y_N .

These two cases are symmetric, thus only proof of Case A will be presented here. If $x[0]$ does not occur in y_N , then $x[0] \in R_{y_N}^\ell$. Otherwise, let $x[0..t]$ be the longest prefix of $x[0..m-2]$ that is a factor of y_N .

Because $0 \leq t < m-1$, we have that $x[1..t+1]$ is a factor of y_N . Therefore, $x[0..t+1] \in M_{y_N}^\ell$. In addition, all factors of $x[0..t+1]$ occur in $\{y_1, \dots, y_{N-1}\}$, so $x[0..t+1] \in R_{y_N}^\ell$.

Now, $x[1..m-1]$ does not occur in any word in $\{y_1, \dots, y_{N-1}\}$, so either $x[m-1]$ does not occur in any word in $\{y_1, \dots, y_{N-1}\}$, which means that $x[m-1] \in R_{\{y_1, \dots, y_{N-1}\}}^\ell$, or let $x[p..m-1]$ be the longest suffix of $x[1..m-1]$ that occurs in a word in $\{y_1, \dots, y_{N-1}\}$.

Because $0 < p \leq m-1$, we have that $x[p-1..m-2]$ occurs in $\{y_1, \dots, y_{N-1}\}$, therefore $x[p-1..m-1] \in M_{\{y_1, \dots, y_{N-1}\}}^\ell$. Since all factors of $x[p-1..m-1]$ occur in y_N , we have $x[p-1..m-1] \in R_{\{y_1, \dots, y_{N-1}\}}^\ell$. □

4 Space-Efficient Algorithm

In this section, we describe how to transform the combinatorial properties of Section 3 to a space-efficient algorithm for computing the MAWs of a collection of words. Note that we do not analyze the time complexity of this algorithm.

In what follows, we say that word aub , where a and b are letters, is the *overlap* of the words au and ub .

Consider we have only two words y_1 and y_2 . We apply Lemma 2 to construct the reduced sets $R_{y_1}^\ell$ from $M_{y_1}^\ell$ and $R_{y_2}^\ell$ from $M_{y_2}^\ell$. Recall that words in $R_{y_1}^\ell$ are factors of y_2 and words in $R_{y_2}^\ell$ are factors of y_1 .

Definition 1 ($\ell - 1$ extensions) Let $\{i, j\} = \{1, 2\}$. For every word in $R_{y_i}^\ell$, we consider all its occurrences in y_j and take all the factors obtained by extending these occurrences to the left up to length $\ell - 1$. We then obtain the set $\overleftarrow{R}_{y_i}^\ell$ of $(\ell - 1)$ -left extensions of words of $R_{y_i}^\ell$ in y_j . Similarly, we define the set $\overrightarrow{R}_{y_i}^\ell$ of $(\ell - 1)$ -right extensions of words of $R_{y_i}^\ell$ in y_j .

Formally, $\overleftarrow{R}_{y_i}^\ell = \{wv \in \text{Fact}(y_j) : v \in R_{y_i}^\ell \text{ and } |wv| \leq \ell - 1\}$ and $\overrightarrow{R}_{y_i}^\ell = \{vw \in \text{Fact}(y_j) : v \in R_{y_i}^\ell \text{ and } |vw| \leq \ell - 1\}$.

Example 3 (Two sequences) Let $y_1 = \text{abaab}$, $y_2 = \text{bbaaab}$ and $\ell = 5$. We have $R_{y_1}^\ell = \{\text{aaa}, \text{bb}\}$ and $R_{y_2}^\ell = \{\text{aba}, \text{baab}\}$.

We take aaa from $R_{y_1}^\ell$, search for it in y_2 , and extend it to the left to get $\underline{\text{b}}\text{aaa}$ (we underline the letters added in the extension). The word bb cannot be extended further to the left as it appears only at the beginning of y_2 . The set of left extensions of words in $R_{y_1}^\ell$ is therefore $\overleftarrow{R}_{y_1}^\ell = \{\underline{\text{b}}\text{aaa}, \text{aaa}, \text{bb}\}$. The words of $R_{y_2}^\ell$ cannot be $(\ell - 1)$ -left-extended further in y_1 , hence $\overleftarrow{R}_{y_2}^\ell = R_{y_2}^\ell = \{\text{aba}, \text{baab}\}$. Similarly, the $(\ell - 1)$ -right

extensions of words of $R_{y_2}^\ell$ in y_1 are $\overrightarrow{R_{y_2}^\ell} = \{aba, abaa, baab\}$, and the $(\ell - 1)$ -right extensions of words of $R_{y_1}^\ell$ in y_2 are $\overleftarrow{R_{y_1}^\ell} = \{aaa, aaab, bb, bba, bbaa\}$.

Lemma 4 Let $\{i, j\} = \{1, 2\}$. A word aub , where a and b are letters, belongs to $M_{\{y_1, y_2\}}^\ell \setminus (M_{y_i}^\ell \cup M_{y_j}^\ell)$ if and only if there exists a pair $(x_i, x_j) \in R_{y_i}^\ell \times R_{y_j}^\ell$ such that:

1. au is an $(\ell - 1)$ -right extension of x_i ;
2. ub is an $(\ell - 1)$ -left extension of x_j .

Proof It follows directly from Theorem 1. □

As a consequence, in order to find the words in $M_{\{y_i, y_j\}}^\ell \setminus (M_{y_i}^\ell \cup M_{y_j}^\ell)$, we take all the possible words that are obtained as the overlap of an $(\ell - 1)$ -right extension of a word in $R_{y_i}^\ell$ with an $(\ell - 1)$ -left extension of a word in $R_{y_j}^\ell$.

Example 4 (Two sequences, continued) Let $y_1 = abaab, y_2 = bbaaab$ and $\ell = 5$. We start from $\overleftarrow{R_{y_1}^\ell} = \{\underline{baaa}, aaa, bb\}$ and $\overrightarrow{R_{y_2}^\ell} = \{aba, abaa, baab\}$. We take $abaa$ from $\overrightarrow{R_{y_2}^\ell}$ and \underline{baaa} from $\overleftarrow{R_{y_1}^\ell}$. They overlap forming the word $abaaa$, which belongs to $M_{\{y_1, y_2\}}^\ell$. Next, we consider $\overrightarrow{R_{y_1}^\ell} = \{aaa, aaab, bb, bba, bbaa\}$ and $\overleftarrow{R_{y_2}^\ell} = \{aba, baab\}$. We take $bbaa$ from $\overrightarrow{R_{y_1}^\ell}$ and $baab$ from $\overleftarrow{R_{y_2}^\ell}$. They overlap forming the word $bbaab$, which belongs to $M_{\{y_1, y_2\}}^\ell$. This completes $M_{\{y_1, y_2\}}^\ell$, as there are no other overlaps.

It should now be clear how this approach can be generalized to any number of words. We show this in the next example.

Example 5 (Three sequences) Let $y_1 = abaab, y_2 = bbaaab, y_3 = bababaaa$ and $\ell = 5$. We have that $M_{\{y_1, y_2\}}^\ell = \{aaaa, bab, aaba, abaaa, bbaab, abb, bbb\}$ and $M_{y_3}^\ell = \{aaa, bb, aab\}$.

We want to compute $M_{\{y_1, y_2, y_3\}}^\ell = \{aaaa, aaba, abaaa, bbaab, bbab, abb, bbb\}$. We have $M_{\{y_1, y_2\}}^\ell \cap M_{y_3}^\ell = \emptyset$. Next, we examine $(M_{\{y_1, y_2\}}^\ell \cup M_{y_3}^\ell) \setminus (M_{\{y_1, y_2\}}^\ell \cap M_{y_3}^\ell)$. By applying Lemma 2 we can infer that $abaaa, bbaab, bbb, aaaa, aaba, abb \in M_{\{y_1, y_2, y_3\}}^\ell$. Thus, we have $R_{\{y_1, y_2\}}^\ell = \{bab\}$ and $R_{y_3}^\ell = \{bb, aaa, aab\}$. Finally, to obtain $bbab$ we apply Lemma 4 as follows. We build the sets of $(\ell - 1)$ -extensions $\overleftarrow{R_{\{y_1, y_2\}}^\ell} = \{bab, \underline{abab}\}$, $\overleftarrow{R_{y_3}^\ell} = \{bb, aaa, \underline{baaa}, aab, \underline{aaab}, \underline{baab}\}$, $\overrightarrow{R_{\{y_1, y_2\}}^\ell} = \{bab, baba\}$ and $\overrightarrow{R_{y_3}^\ell} = \{bb, bba, bbaa, aaa, aaab, aab\}$. Computing all the possible overlaps of a word in $\overrightarrow{R_{\{y_1, y_2\}}^\ell}$ with a word in $\overleftarrow{R_{y_3}^\ell}$ and all possible overlaps of a word in $\overrightarrow{R_{y_3}^\ell}$ with a word in $\overleftarrow{R_{\{y_1, y_2\}}^\ell}$ we get the word $bbab$, which is the overlap of $bba \in \overleftarrow{R_{y_3}^\ell}$ with $bab \in \overleftarrow{R_{\{y_1, y_2\}}^\ell}$. This completes $M_{\{y_1, y_2, y_3\}}^\ell$, as there are no other overlaps.

For the space-efficient algorithm, we will need to consider $(\ell - 1)$ -extensions of words in the reduced sets one at a time. For this, we define the set of $(\ell - 1)$ -extensions of a single reduced word. Formally, for a word $x \in R_{y_i}^\ell$, we define $\overleftarrow{X} = \{wx \in \text{Fact}(y_j) : |wx| \leq \ell - 1\}$ and $\overrightarrow{X} = \{xw \in \text{Fact}(y_j) : |xw| \leq \ell - 1\}$. Note that $x \in \overleftarrow{X}$ and $x \in \overrightarrow{X}$ and that $\overleftarrow{R}_{y_i}^\ell = \bigcup_{x \in R_{y_i}^\ell} \overleftarrow{X}$ and $\overrightarrow{R}_{y_i}^\ell = \bigcup_{x \in R_{y_i}^\ell} \overrightarrow{X}$. For example, for the word $x = \text{aab} \in R_{y_3}^\ell$ in the previous example, we have $\overleftarrow{X} = \{\text{aab}, \underline{\text{a}}\text{aab}, \underline{\text{b}}\text{aab}\}$.

We now describe our algorithm. Let $\mathcal{T}(x)$ be the suffix tree of word x and $\overline{\mathcal{T}}(X)$ be the generalized suffix tree of the words in set X . Let us further define the following operation over $\mathcal{T}(x)$: $\text{occ}(x, u)$ returns the starting positions of all occurrences of word u in word x . Let y_{\max_N} denote the longest word in the collection $\{y_1, \dots, y_N\}$.

Let $N = 1$. We read y_1 from memory, construct $\mathcal{T}(y_1)$ [21], compute $M_{y_1}^\ell$ [7], and construct $\overline{\mathcal{T}}(M_{y_1}^\ell)$. We report $M_{y_1}^\ell$ as our first output. The space used thus far is bounded by $\mathcal{O}(|y_1| + \|M_{y_1}^\ell\|) = \mathcal{O}(|y_{\max_1}| + \|M_{y_1}^\ell\|)$.

At the N th step, we already have $\overline{\mathcal{T}}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$ in memory from the $(N - 1)$ th step. We read y_N from memory and construct $\overline{\mathcal{T}}(y_N)$. The incremental step, for all $N \in [2, k]$, works as follows.

Case 1 : We want to check all pairs $(x_1, x_2) \in M_{\{y_1, \dots, y_{N-1}\}}^\ell \times M_{y_N}^\ell$, applying Lemma 2, to construct the set

$$M = \{w \in M_{\{y_1, \dots, y_N\}}^\ell : w \in M_{\{y_1, \dots, y_{N-1}\}}^\ell \cup M_{y_N}^\ell\}$$

and the sets $R_{\{y_1, \dots, y_{N-1}\}}^\ell, R_{y_N}^\ell$. We proceed as follows. We first compute $M_{y_N}^\ell$ using $\overline{\mathcal{T}}(y_N)$. At this point note that we cannot store $M_{y_N}^\ell$ explicitly since it could be the case that $\|M_{y_N}^\ell\| = \omega(\|M_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$. Instead, we output the words in the following constant-space form: $\langle i_1, i_2, \alpha \rangle$ per word [7]; such that $y_N[i_1..i_2] \cdot \alpha \in M_{y_N}^\ell$, where $\alpha \in \Sigma$. In this case, the space used is bounded by $\mathcal{O}(|y_{\max_N}| + \|M_{\{y_1, \dots, y_N\}}^\ell\|)$. We perform the following:

1. We first want to check if the elements of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ are superwords of any element of $M_{y_N}^\ell$. We search for $x_2 \in M_{y_N}^\ell$ in $\overline{\mathcal{T}}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$, one element from $M_{y_N}^\ell$ at a time. By going through all the occurrences of x_2 using $\overline{\mathcal{T}}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$, we find all elements in $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ that are superwords of some x_2 .
2. We next want to check if $x_2 \in M_{y_N}^\ell$ is a superword of any element of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$. We use the classic matching statistics algorithm (see Section 7.8 of [23]), on $\overline{\mathcal{T}}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$. This algorithm finds the longest prefix of $x_2[i..|x_2| - 1]$ that matches any factor of the elements in $M_{\{y_1, \dots, y_{N-1}\}}^\ell$, for all $i \in [0, |x_2| - 1]$, using $\overline{\mathcal{T}}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$. By definition, no element in $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ is a factor of another element in the same set. Thus, if a longest match corresponds to an element in $M_{\{y_1, \dots, y_{N-1}\}}^\ell$, this can be found using $\overline{\mathcal{T}}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$.

We create set $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ explicitly since it is a subset of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$. Set $R_{y_N}^\ell$ is created implicitly: every element $x_2 \in R_{y_N}^\ell$ is stored as a tuple $\langle i_1, i_2, \alpha \rangle$ such that $x_2 = y_N[i_1..i_2] \cdot \alpha$. (Recall that y_N is currently stored in memory.) We can thus store every element of $\{x_2 : x_2 \in M \cap M_{y_N}^\ell\}$ with the same representation. All other elements of M can be stored explicitly as they are elements of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$. The space used thus far is thus bounded by $\mathcal{O}(|y_{\max_N}| + \|M_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$.

Case 2 : We want to compute $\{w : w \in M_{\{y_1, \dots, y_N\}}^\ell, w \notin M_{\{y_1, \dots, y_{N-1}\}}^\ell \cup M_{y_N}^\ell\}$. To this end, we consider all pairs $(x_1, x_2) \in R_{\{y_1, \dots, y_{N-1}\}}^\ell \times R_{y_N}^\ell$. (We symmetrically consider all pairs $(x_1, x_2) \in R_{y_N}^\ell \times R_{\{y_1, \dots, y_{N-1}\}}^\ell$.)

We read y_N to construct $\mathcal{T}(y_N)$.

Then we consider one by one each pair (x_1, x_2) of $R_{\{y_1, \dots, y_{N-1}\}}^\ell \times R_{y_N}^\ell$.

Using $\mathcal{T}(y_N)$ we compute $P_N = \text{occ}(y_N, x_1)$ and

$\overleftarrow{X}_1 = \{wx_1 \in \text{Fact}(y_N) : |wx_1| \leq \ell - 1\}$.

For all $i \in [1, N - 1]$ we perform the following:

1. We read y_i from memory and construct $\mathcal{T}(y_i)$
2. We compute $P_i = \text{occ}(y_i, x_2)$ and the set $\overrightarrow{X}_2 \cap \text{Fact}(y_i) = \{x_2w \in \text{Fact}(y_i) : |x_2w| \leq \ell - 1\}$
3. For each $(\overleftarrow{x}_1, \overrightarrow{x}_2) \in (\overleftarrow{X}_1, \overrightarrow{X}_2 \cap \text{Fact}(y_i))$ with $|\overleftarrow{x}_1| = |\overrightarrow{x}_2|$:
 - (a) We check the equality of the longest proper suffix of \overrightarrow{x}_2 and the longest proper prefix of \overleftarrow{x}_1 . If there is equality we note $u = \overrightarrow{x}_2[1 \dots |\overrightarrow{x}_2| - 1] = \overleftarrow{x}_1[0 \dots |\overleftarrow{x}_1| - 2]$.
 - (b) By Lemma 4, $x_2[0] \cdot u \cdot x_1[|x_1| - 1]$ is a MAW of the set of words $\{y_1, \dots, y_N\}$ and so we add element $x_2[0] \cdot u \cdot x_1[|x_1| - 1]$, expressed implicitly, to set M .

Note that $|P_N| = \mathcal{O}(|y_N|)$ for all x_1 and $|P_i| = \mathcal{O}(|y_i|)$ for all x_2 . If one pair of extensions is handled at a time, the space used is bounded by $\mathcal{O}(|y_{\max_N}| + \|M_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$. We repeat the same process with word y_i , for all $i \in [2, N - 1]$.

Finally, we delete the set $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ and $\mathcal{T}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$, we set $M_{\{y_1, \dots, y_N\}}^\ell = M$, where every element is now safely expressed explicitly, and construct $\mathcal{T}(M_{\{y_1, \dots, y_N\}}^\ell)$. We are now ready for step $N + 1$.

We arrive at the following result.

Theorem 2 *If $\mathcal{O}(|y_{\max_{N-1}}| + \|M_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$ space is made available at step $N - 1$, we can compute $M_{\{y_1, \dots, y_N\}}^\ell$ from $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ using $\mathcal{O}(|y_{\max_N}| + \|M_{\{y_1, \dots, y_N\}}^\ell\|)$ space.*

Proof The space complexity follows from the above discussion. The correctness follows from Lemmas 2 and 4. □

We obtain immediately the following corollary.

Corollary 1 *If $\|M_{\{y_1, \dots, y_N\}}^\ell\| = \mathcal{O}(\|M_{\{y_1, \dots, y_k\}}^\ell\|)$, for all $1 \leq N < k$, we can compute $M_{\{y_1, \dots, y_k\}}^\ell$ using $\mathcal{O}(|y_{\max k}| + \|M_{\{y_1, \dots, y_k\}}^\ell\|)$ space.*

5 Time-Efficient Algorithm

In this section, we provide a time-efficient implementation of the algorithm presented in Section 4. Let us first introduce an algorithmic tool. In the *weighted ancestor* problem, introduced in [24], we consider a rooted tree T with an integer weight function μ defined on the nodes. We require that the weight of the root is zero and the weight of every non-root node is strictly larger than the weight of its parent. A weighted ancestor query, given a node v and an integer value $w \leq \mu(v)$, asks for the highest ancestor u of v such that $\mu(u) \geq w$, i.e., such an ancestor u has the property that that $\mu(u) \geq w$ and $\mu(u)$ is the smallest possible. When T is the suffix tree of a word y of length n , we can locate the locus of any factor of $y[i..j]$ using a weighted ancestor query. We define the weight of a node of the suffix tree as the length of the word it represents. Thus a weighted ancestor query can be used for the terminal node labeled with i to create (if necessary) and mark the node that corresponds to $y[i..j]$.

Theorem 3 ([25]) *Given a collection Q of weighted ancestor queries on a weighted tree T on n nodes with integer weights up to $n^{\mathcal{O}(1)}$, all the queries in Q can be answered off-line in $\mathcal{O}(n + |Q|)$ time.*

5.1 The Algorithm

At the N th step, we have in memory the set $M_{\{y_1, \dots, y_{N-1}\}}^\ell$. Our time-efficient algorithm works as follows:

1. We read word y_N from memory and compute $M_{y_N}^\ell$ in time $\mathcal{O}(|y_N|)$. We output the words in the previously described constant-space form $\langle i_1, i_2, \alpha \rangle$ such that $y_N[i_1..i_2] \cdot \alpha \in M_{y_N}^\ell$.
2. Here we compute Case 1 MAWs. We apply Lemma 2 to construct set

$$M = \{w : w \in M_{\{y_1, \dots, y_N\}}^\ell, w \in M_{\{y_1, \dots, y_{N-1}\}}^\ell \cup M_{y_N}^\ell\}$$

and the sets $R_{\{y_1, \dots, y_{N-1}\}}^\ell, R_{y_N}^\ell$ as follows.

- (a) We first want to find the elements of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ that are superwords of at least one word $y_N[i_1..i_2] \cdot \alpha$. We build the generalized suffix tree $T_1 = \mathcal{T}(M_{\{y_1, \dots, y_{N-1}\}}^\ell \cup \{y_N\})$ [23]. We find the locus of the longest proper prefix $y_N[i_1..i_2]$ of each element of $M_{y_N}^\ell$ in T_1 via answering off-line a batch of weighted ancestor queries (Theorem 3). From there on, we spell α and mark the corresponding node on T_1 , if any. After processing all $\langle i_1, i_2, \alpha \rangle$ in the same manner, we traverse T_1 to gather all occurrences (starting positions) of words $y_N[i_1..i_2] \cdot \alpha$ in the elements of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$, thus finding the elements of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ that

are superwords of any $y_N[i_1..i_2] \cdot \alpha$. By definition, no MAW $y_N[i_1..i_2] \cdot \alpha$ is a prefix of another MAW $y_N[i'_1..i'_2] \cdot \alpha'$, thus the marked nodes form pairwise disjoint subtrees, and the whole process takes time $\mathcal{O}(|y_N| + \|M_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$, the size of T_1 .

Example 6 Let $\ell = 3$, $M_{\{y_1, \dots, y_{N-1}\}}^\ell = \{aaa, bb\}$ and $y_N = abba$, so that $M_{y_N}^\ell = \{aa, bab, aba, bbb\}$. We build the suffix tree of $aaa\#_1bb\#_2abba\#_3$. From $M_{y_N}^\ell$ we only find the path-label aa in the suffix tree (see the node in red in Fig. 2). By gathering all occurrences of aa in the subtree rooted at that node (terminal nodes), we conclude that a word from $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ (aaa) is a superword of aa . We use identifiers in the form $\#_i$ where $i \in \{1, \dots, |M_{\{y_1, \dots, y_{N-1}\}}^\ell| + |M_{y_N}^\ell|\}$.

- (b) We next want to check if the words $y_N[i_1..i_2] \cdot \alpha$ are superwords of at least one element of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$. Recall that equality of words has already been checked in step 2(a). So we are just left with checking whether any element, say u , of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ is a proper factor of $y_N[i_1..i_2] \cdot \alpha$. We rely on the following simple fact. If u is a proper factor of $y_N[i_1..i_2] \cdot \alpha$, then it is either a factor of the longest proper prefix of $y_N[i_1..i_2] \cdot \alpha$ or a factor of the longest proper suffix of $y_N[i_1..i_2] \cdot \alpha$ or of both. By definition of MAWs, $y_N[i_1..i_2] \cdot \alpha$ can be represented by $\langle i_1, i_2, \alpha \rangle$ or, equivalently, by $\langle \beta, j_1, j_2 \rangle$, where α, β are letters, $[i_1, i_2], [j_1, j_2]$ are intervals over y_N and $y_N[i_1..i_2] \cdot \alpha = \beta \cdot y_N[j_1..j_2]$. The latter of the two representations can be obtained from the former using a batch of weighted ancestor queries similar to step 2(a) in linear time for all words in $M_{y_N}^\ell$. We use

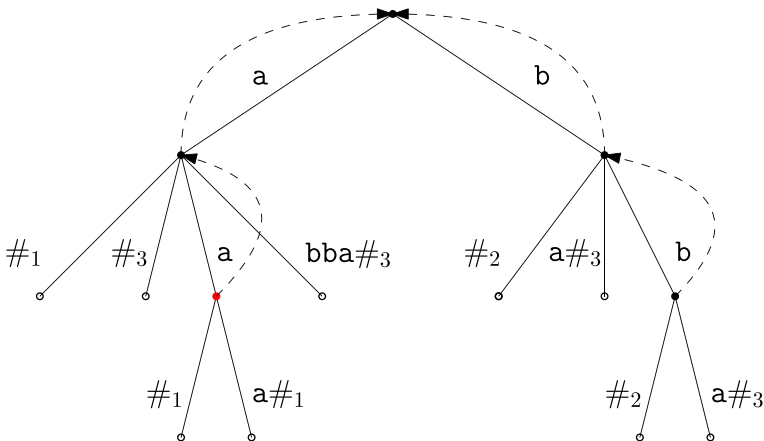


Fig. 2 The red node in the tree marks the path-label aa where $aa \in M_{y_N}^\ell$. By traversing the leaves of this node, it is clear that aa is a proper factor of the first element aaa in $M_{\{y_1, \dots, y_{N-1}\}}^\ell$

radixsort to sort all intervals $[i_1, i_2]$ and run the matching statistics algorithm for y_N with respect to $\mathcal{T}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$. By definition, no element in $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ is a factor of another element in the same set. Thus, if a factor of $y_N[i_1..i_2]$ corresponds to an element in $M_{\{y_1, \dots, y_{N-1}\}}^\ell$, this can be located in $\mathcal{T}(M_{\{y_1, \dots, y_{N-1}\}}^\ell)$ while running the matching statistics algorithm. The whole process takes $\mathcal{O}(|y_N| + \|M_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$ time: $\mathcal{O}(\|M_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$ time to construct the suffix tree and a further $\mathcal{O}(|y_N|)$ time for the matching statistics algorithm and for processing the $\mathcal{O}(|y_N|)$ tuples. We treat the case $[j_1, j_2]$ analogously.

Example 7 Let $\ell = 3$, $M_{\{y_1, \dots, y_{N-1}\}}^\ell = \{aaa, bb\}$ and $y_N = abba$, so that $M_{y_N}^\ell = \{aa, bab, aba, bbb\}$. We build the suffix tree of $aaa\#_1bb\#_2$ (see Fig. 3). We use identifiers in the form $\#_j$ where $j \in \{1, \dots, |M_{\{y_1, \dots, y_{N-1}\}}^\ell|\}$. The sorted list of unique tuples from $M_{y_N}^\ell$ is $\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle$. The longest match at position 0 of $y_N = abba$ is a and there is no full match of a word from $M_{\{y_1, \dots, y_{N-1}\}}^\ell$. We follow the suffix-link of the node, which takes us to the root. The longest match at position 1 is bb . This takes us to a full match of a word from $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ (bb), which means that bbb , whose prefix and suffix is represented by the single tuple $\langle 1, 2 \rangle$, is a superword of word bb from $M_{\{y_1, \dots, y_{N-1}\}}^\ell$.

We create set $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ explicitly since it is a subset of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$. We create set $R_{y_N}^\ell$ implicitly: every element $x \in R_{y_N}^\ell$ is stored as a tuple $\langle i_1, i_2, \alpha \rangle$ such

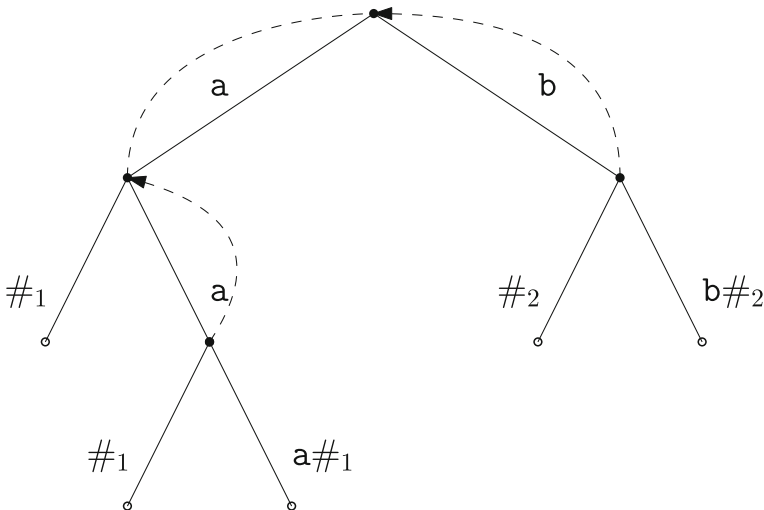


Fig. 3 Starting from the root, we find a path-label bb . The second element of $M_{\{y_1, \dots, y_{N-1}\}}^\ell$ is a factor of word bbb in $M_{y_N}^\ell$, whose prefix and suffix corresponds to the tuple $\langle 1, 2 \rangle$

- that $x = y_N[i_1..i_2] \cdot \alpha$. We store every element of $\{x_2 : x_2 \in M \cap M_{y_N}^\ell\}$ with the same representation. All other elements of M are stored explicitly.
3. Construct the suffix tree of y_N and use it to locate all occurrences of words in $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ in y_N and store the occurrences as pairs (starting position, ending position). This step can be carried out in time $\mathcal{O}(|y_N| + \|R_{\{y_1, \dots, y_{N-1}\}}^\ell\|)$. This claim is due to the following fact: no element in $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ is a prefix of another element in $R_{\{y_1, \dots, y_{N-1}\}}^\ell$.
 4. For every $i \in [1, N - 1]$, we perform the following to compute Case 2 MAWs:
 - (a) Read word y_i from memory. Construct the suffix tree T_x of word $x = y_i \# y_N$ in time $\mathcal{O}(|y_i| + |y_N|)$. Use T_x to locate all occurrences of elements of $R_{y_N}^\ell$ in y_i and store the occurrences as pairs (starting position, ending position). This step can be completed in time $\mathcal{O}(|y_i| + |y_N|)$ similar to 2. This claim is due to the fact that no element in $R_{y_N}^\ell$ is a prefix of another element in $R_{y_N}^\ell$.
 - (b) During a bottom-up DFS traversal of T_x , we mark, at each explicit node of T_x , the smallest starting position of the word represented by that node, and the largest starting position of the same word. This can be done in time $\mathcal{O}(|y_i| + |y_N|)$ by propagating upwards the labels of the terminal nodes (starting positions of suffixes) and updating the smallest and largest positions analogously.
 - (c) Compute the set $M_{y_i \# y_N}^\ell$ using [7], and output the words in the following constant-space form: $\langle a, i_1, i_2, b \rangle$ per word; such that $a \cdot x[i_1..i_2] \cdot b$ is a MAW. This can be carried out in time $\mathcal{O}(|y_i| + |y_N|)$.
 - (d) For each element of $M_{y_i \# y_N}^\ell$, we need to locate the node representing word $ax[i_1..i_2] = au$ and the node representing word $x[i_1..i_2]b = ub$. This can be done in time $\mathcal{O}(|y_i| + |y_N|)$ via answering off-line a batch of weighted ancestor queries (Theorem 3). At this point, we have located the two nodes on T_x . We assign a pointer from the stored starting position g of au to the ending position f of ub (see Fig. 4), only if g is before $\#$ and f is after $\#$ (f can be computed using the stored starting position of ub and the length of ub). Conversely, we assign a pointer from the ending position f of ub to the stored starting position g of au , only if f is before $\#$ and g is after $\#$.
 - (e) Suppose au occurs in y_i and ub in y_N . We make use of the pointers as follows. Recall steps 3 and 4(a) and check whether au starts where a word r_1 of $R_{y_N}^\ell$ starts and ub ends where a word r_2 of $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ ends. If this is

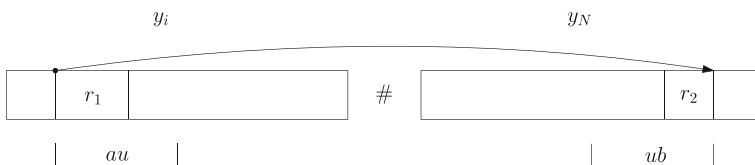


Fig. 4 au starts where a word r_1 of $R_{y_N}^\ell$ starts in y_i and ub ends where a word r_2 of $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ ends in y_N

the case, then by Theorem 1 *aub* is added to our output set M , otherwise we discard it. Inspect Fig. 4 in this regard. Conversely, if *au* occurs in y_N and *ub* in y_i , then we check whether *au* starts where a word r_2 of $R_{\{y_1, \dots, y_{N-1}\}}^\ell$ starts and whether *ub* ends where a word r_1 of $R_{y_N}^\ell$ ends. If this is the case, then *aub* is added to M , otherwise we discard it.

Finally, we set $M_{\{y_1, \dots, y_N\}}^\ell = M$ as the output of the N th step. Let MAXIN be the length of the longest word in $\{y_1, \dots, y_k\}$ and $\text{MAXOUT} = \max\{\|M_{\{y_1, \dots, y_N\}}^\ell\| : N \in [1, k]\}$.

Theorem 4 Given k words y_1, y_2, \dots, y_k and an integer $\ell > 1$, all $M_{y_1}^\ell, \dots, M_{\{y_1, \dots, y_k\}}^\ell$ can be computed in $\mathcal{O}(kn + \sum_{N=1}^k \|M_{\{y_1, \dots, y_N\}}^\ell\|)$ total time using $\mathcal{O}(\text{MAXIN} + \text{MAXOUT})$ space, where $n = \sum_{N=1}^k |y_N|$.

Proof From the above discussion, the time is bounded by $\mathcal{O}(\sum_{N=1}^k \sum_{i=1}^{N-1} (|y_N| + |y_i|) + \sum_{N=1}^k \|M_{\{y_1, \dots, y_N\}}^\ell\|)$. We can bound the first term as follows.

$$\begin{aligned} & \sum_{N=1}^k \sum_{i=1}^{N-1} (|y_N| + |y_i|) \\ & \leq \sum_{N=1}^k \sum_{i=1}^k (|y_N| + |y_i|) \\ & = \sum_{N=1}^k \sum_{i=1}^k |y_N| + \sum_{N=1}^k \sum_{i=1}^k |y_i| \\ & = 2k(|y_1| + \dots + |y_k|). \end{aligned}$$

Therefore, the time is bounded by $\mathcal{O}(kn + \sum_{N=1}^k \|M_{\{y_1, \dots, y_N\}}^\ell\|)$.

The space is bounded by the maximum time spent at a single step; namely, the length of the longest word in the collection plus the maximum total size of set elements across all output sets. Note that the total output size of the algorithm is the sum of all its output sets, that is $\sum_{N=1}^k \|M_{\{y_1, \dots, y_N\}}^\ell\|$, and MAXOUT could come from any intermediate set.

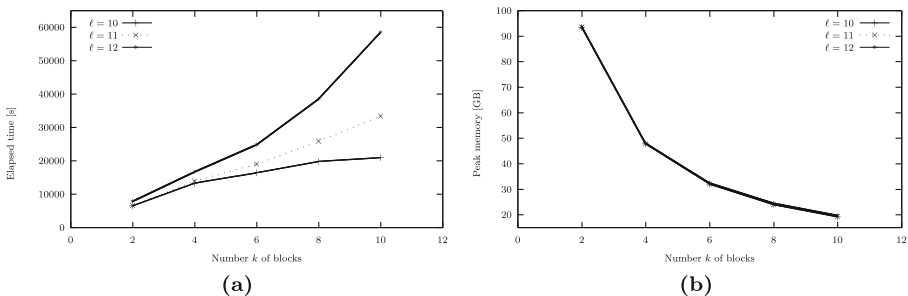


Fig. 5 Elapsed time and peak memory usage using increasing k blocks of the entire human genome (hg38) for $\ell = 10, 11, 12$; notice that the peak memory usage is the same for all values of ℓ

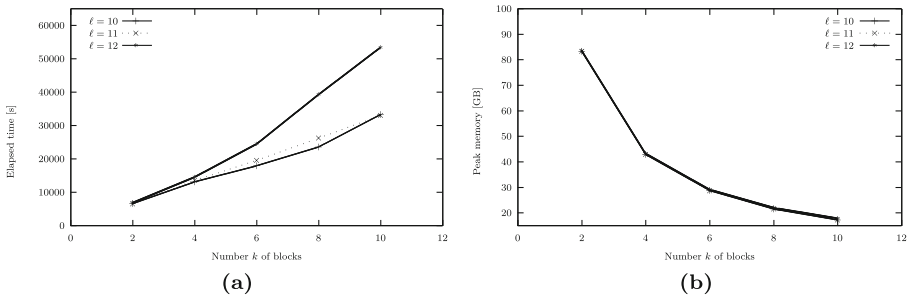


Fig. 6 Elapsed time and peak memory usage using increasing k blocks of the entire mouse genome (mm10) for $\ell = 10, 11, 12$; notice that the peak memory usage is the same for all values of ℓ

The correctness of the algorithm follows from Lemma 2 and Theorem 1. \square

6 Proof-of-Concept Experiments

In this section, we do not directly compare against the fastest internal [7] or external [18] memory implementations because the former assumes that we have the required amount of internal memory, and the latter assumes that we have the required amount of external memory to construct and store the global data structures for a given input dataset. If the memory for constructing and storing the data structures is available, these linear-time algorithms are surely faster than the method proposed here. In what follows, we rather show that our output-sensitive technique offers a space-time tradeoff, which can be usefully exploited for specific values of ℓ , the maximal length of MAWs we wish to compute.

The time-efficient algorithm discussed in Section 5 (with the exception of storing and searching the reduced sets of words explicitly rather than in the constant-space form previously described) has been implemented in the C++ programming language¹. The correctness of our implementation has been confirmed against that of [7]. We have also implemented the algorithm discussed in Section 4 but as it was significantly slower, its results are omitted from here. As input datasets here we used: the entire human genome (version hg38), which has an approximate size of 3.1GB; the entire mouse genome (version mm10), which has an approximate size of 2.6GB; and the entire chimp genome (version panTro6), which has an approximate size of 2.8GB. All datasets were downloaded from the UCSC Genome Browser [26]. The following experiments were conducted on a machine with an Intel Core i5-4690 CPU at 3.50 GHz and 128GB of memory running GNU/Linux. We ran the program by splitting the genomes into $k = 2, 4, 6, 8, 10$ blocks and setting $\ell = 10, 11, 12$. Figures 5, 6 and 7 depict the change in elapsed time and peak memory usage as k and ℓ increase (space-time tradeoff).

¹The implementation can be made available upon request.

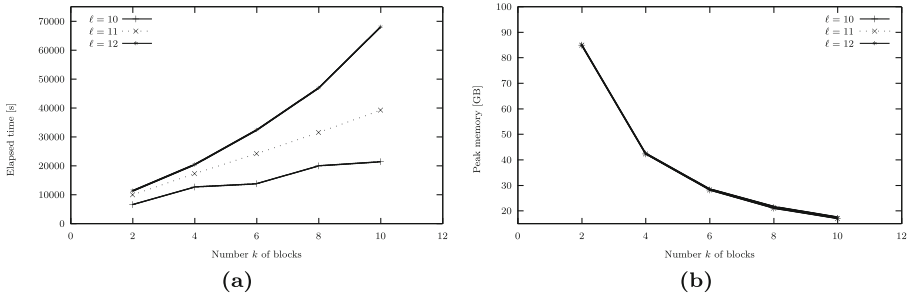


Fig. 7 Elapsed time and peak memory usage using increasing k blocks of the entire chimp genome (panTro6) for $\ell = 10, 11, 12$; notice that the peak memory usage is the same for all values of ℓ

In accordance to Theorem 4: graph (a) in all figures shows an increase of time as k and ℓ increase; and graph (b) in all figures shows a decrease in peak memory as k increases. Notice that the space to construct the block-wise data structures bounds the total space used for the specific ℓ values and that is why the memory peak is essentially the same for the ℓ values used. This can specifically be seen for $\ell = 10$ where all words of length 10 are present in all three genomes. The same datasets were used to run the fastest internal memory implementation for computing MAWs [7] on the same machine for $\ell = 12$. Notice that the algorithm of [7] takes the same time and space irrespective of ℓ . It took only 2934 seconds to process the human genome but with a peak memory usage of 75.40GB (it took 2844 seconds to process the mouse genome with a peak memory usage of 66.54GB; and 2731 seconds to process the chimp genome with a peak memory usage of 69.49GB). These results confirm our theoretical findings and justify our contribution.

7 Final Remarks

We presented a new technique for constructing antidictionaries of long texts in output-sensitive space. Let us conclude with the following remarks:

1. Any space-efficient algorithm designed for global data structures (such as [20]) can be directly applied to the k documents in our technique to further reduce the working space.
2. There is a connection between MAWs and other word regularities [9]. Our technique could potentially be applied to computing these regularities in output-sensitive space.
3. Our technique could serve as a basis for a new parallelization scheme for constructing antidictionaries (see also [27]), in which several documents are processed concurrently.

Funding Open access funding provided by Università degli Studi di Palermo within the CRUI-CARE Agreement.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.


References

1. Ayad, L.A.K., Badkobeh, G., Fici, G., Héliou, A., Pissis, S.P.: Constructing antidictionaries in output-sensitive space. In: Bilgin, A., Marcellin, M.W., Serra-Sagristà, J., Storer, J.A. (eds.) Data Compression Conference, DCC 2019, pp. 538–547. IEEE, Snowbird (2019)
2. Crochemore, M., Mignosi, F., Restivo, A.: Automata and forbidden words. *Inf. Process. Lett.* **67**(3), 111–117 (1998)
3. Charalampopoulos, P., Crochemore, M., Fici, G., Mercas, R., Pissis, S.P.: Alignment-free sequence comparison using absent words. *Inf. Comput.* **262**(1), 57–68 (2018)
4. Almirantis, Y., Charalampopoulos, P., Gao, J., Iliopoulos, C.S., Mohamed, M., Pissis, S.P., Polychronopoulos, D.: On avoided words, absent words, and their application to biological sequence analysis. *Algorithm. Mol. Biol.* **12**(1), 5:1–5:12 (2017)
5. Fici, G., Gawrychowski, P.: Minimal absent words in rooted and unrooted trees. In: String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019. Proceedings, Segovia (2019)
6. Fukae, H., Ota, T., Morita, H.: On fast and memory-efficient construction of an antidictionary array. In: Proceedings of the 2012 IEEE International Symposium on Information Theory, pp. 1092–1096, IEEE (2012)
7. Barton, C., Héliou, A., Mouchard, L., Pissis, S.P.: Linear-time computation of minimal absent words using suffix array. *BMC Bioinform.* **15**, 388 (2014)
8. Fujishige, Y., Tsujimaru, Y., Inenaga, S., Bannai, H., Takeda, M.: Computing DAWGs and minimal absent words in linear time for integer alphabets. In: Faliszewski, P., Muscholl, A., Niedermeier, R. (eds.) 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, LIPIcs, vol. 58, pp. 38:1–38:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Kraków (2016)
9. Charalampopoulos, P., Crochemore, M., Pissis, S.P.: On extended special factors of a word. In: Gagie, T., Moffat, A., Navarro, G., Cuadros-Vargas, E. (eds.) String Processing and Information Retrieval - 25th International Symposium, SPIRE 2018, Lima, Proceedings, Lecture Notes in Computer Science, vol. 11147, pp. 131–138. Springer (2018)
10. Belazzougui, D., Cunial, F., Kärkkäinen, J., Mäkinen, V.: Versatile succinct representations of the bidirectional Burrows-Wheeler transform. In: Bodlaender, H.L., Italiano, G.F. (eds.) Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis. Proceedings, Lecture Notes in Computer Science, vol. 8125, pp. 133–144. Springer (2013)
11. Belazzougui, D., Cunial, F.: A framework for space-efficient string kernels. *Algorithmica* **79**(3), 857–883 (2017)
12. Crochemore, M., Mignosi, F., Restivo, A., Salemi, S.: Data compression using antidictionaries. *Proc. IEEE* **88**(11), 1756–1768 (2000)
13. Crochemore, M., Navarro, G.: Improved antidictionary based compression. In: 22nd International Conference of the Chilean Computer Science Society (SCCC 2002), pp. 7–13, Copiapo (2002)
14. Fiala, M., Holub, J.: DCA using suffix arrays. In: 2008 data compression conference (DCC 2008), pp. 516. IEEE Computer Society, Snowbird (2008)
15. Ota, T., Morita, H.: On the adaptive antidictionary code using minimal forbidden words with constant lengths. In: Proceedings of the International Symposium on Information Theory and its Applications, ISITA 2010, pp. 72–77. IEEE, Taichung (2010)
16. Crochemore, M., Héliou, A., Kucherov, G., Mouchard, L., Pissis, S.P., Ramusat, Y.: Minimal absent words in a sliding window and applications to on-line pattern matching. In: Klasing, R., Zeitoun,

- M. (eds.) *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, Proceedings, Lecture Notes in Computer Science*, vol. 10472, pp. 164–176. Springer (2017)
17. Silva, R.M., Pratas, D., Castro, L., Pinho, A.J., Ferreira, P.J.S.G.: Three minimal sequences found in Ebola virus genomes and absent from human DNA. *Bioinformatics* **31**(15), 2421–2425 (2015)
 18. Héliou, A., Pissis, S.P., Puglisi, S.J.: emMAW: computing minimal absent words in external memory. *Bioinformatics* **33**(17), 2746–2749 (2017)
 19. Kärkkäinen, J., Kempa, D., Puglisi, S.J.: Parallel external memory suffix sorting. In: Cicalese, F., Porat, E., Vaccaro, U. (eds.) *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Proceedings, Lecture Notes in Computer Science*, vol. 9133, pp. 329–342. Springer (2015). https://doi.org/10.1007/978-3-319-19929-0_28
 20. Fujishige, Y., Takagi, T., Hendrian, D.: Truncated DAWGs and their application to minimal absent word problem. In: Gagie, T., Moffat, A., Navarro, G., Cuadros-Vargas, E. (eds.) *String Processing and Information Retrieval - 25th International Symposium, SPIRE 2018, Lima, Proceedings, Lecture Notes in Computer Science*, vol. 11147, pp. 139–152. Springer (2018)
 21. Crochemore, M., Hancart, C., Lecroq, T.: *Algorithms on strings*. Cambridge University Press (2007)
 22. Farach, M.: Optimal suffix tree construction with large alphabets. In: *38th Annual Symposium on Foundations of Computer Science, FOCS '97*, pp. 137–143. IEEE Computer Society, Miami Beach (1997)
 23. Gusfield, D.: *Algorithms on strings, trees, and sequences: Computer science and computational biology*. Cambridge University Press, New York (1997)
 24. Farach, M., Muthukrishnan, S.: Perfect hashing for strings: Formalization and algorithms. In: Hirschberg, D.S., Myers, E.W. (eds.) *Combinatorial Pattern Matching, 7th Annual Symposium, CPM 96, Laguna Beach, Proceedings, Lecture Notes in Computer Science*, vol. 1075, pp. 130–140. Springer (1996)
 25. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Walen, T.: A linear-time algorithm for seeds computation. *ACM Trans. Algorithm.* **16**(2), 27:1–27:23 (2020)
 26. Kent, W.J., Sugnet, C.W., Furey, T.S., Roskin, K.M., Pringle, T.H., Zahler, A.M., Haussler, D.: The human genome browser at UCSC. *Genome Res.* **12**(6), 996–1006 (2002)
 27. Barton, C., Héliou, A., Mouchard, L., Pissis, S.P.: Parallelising the computation of minimal absent words. In: Wyrzykowski, R., Deelman, E., Dongarra, J.J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) *Parallel processing and applied mathematics - 11th international conference, PPAM 2015, Krakow. revised selected papers, part II, Lecture Notes in Computer Science*, vol. 9574, pp. 243–253. Springer (2015)
 28. Gagie, T., Moffat, A., Navarro, G., Cuadros-Vargas, E. (eds.): *String processing and information retrieval - 25th international symposium, SPIRE 2018, Lima, proceedings, Lecture Notes in Computer Science*, vol. 11147. Springer (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Lorraine A.K. Ayad¹ · Golnaz Badkobeh² · Gabriele Fici³  · Alice Héliou⁴ · Solon P. Pissis^{5,6}

✉ Gabriele Fici
gabriele.fici@unipa.it

Lorraine A.K. Ayad
lorraine.ayad@kcl.ac.uk

Golnaz Badkobeh
g.badkobeh@gold.ac.uk

Alice Héliou
alice.heliou@gmail.com

Solon P. Pissis
solon.pissis@cwi.nl

- ¹ Department of Informatics, King's College London, London, UK
- ² Department of Computing, Goldsmiths University of London, London, UK
- ³ Dipartimento di Matematica e Informatica, Università di Palermo, Palermo, Italy
- ⁴ Laboratoire d'Informatique de l'École Polytechnique, École Polytechnique, Palaiseau, France
- ⁵ CWI, Amsterdam, The Netherlands
- ⁶ Vrije Universiteit, Amsterdam, The Netherlands