Specific Patterns Against Reference Sequences

Marie-Pierre Béal[1]
Univ. Gustave Eiffel, CNRS, LIGM, Champs-sur-Marne, France

Maxime Crochemore
Univ. Gustave Eiffel, CNRS, LIGM, Champs-sur-Marne, France
King's College London, UK

Abstract

We design alignment-free techniques for comparing a set of sequences or just a word, called a target, against another set of words, called a reference. This is done with the detection of factor patterns that distinguish the target from the reference. A target-specific factor of a target T against a reference R is then a factor w of a word in T that is not a factor of a word in R but whose proper factors of w are factors of a word in R. The strategy is based on the notion of minimal absent/forbidden words.

We first address the computation of the set of target-specific factors of a target T against a reference R, where T and R are finite sets of sequences. The result is the construction of an automaton accepting the set of all considered target-specific factors. The construction algorithm runs in linear time according to the size of $T \cup R$.

The second result is the design of an algorithm to compute all the occurrences in a single sequence T of its target-specific factors against a reference R. The algorithm runs in real-time on the target sequence, independently of the number of occurrences of target-specific factors.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Pattern matching

Keywords and phrases Specific pattern, Minimal absent word, Minimal forbidden word, Directed Acyclic Word Graph (DAWG), Suffix automaton

Digital Object Identifier 10.4230/OASIcs.Grossi.2025.14

Category Research

1 Introduction

The goal of this article is to design an alignment-free technique for comparing a set of sequences or words, called a target, against a set of words, called a reference.

The motivation comes from the analysis of genomic sequences as done, for example, by Khorsand et al. in [23] in which the authors introduce the notion of sample-specific strings. To avoid alignments but to extract interesting elements that differentiate the target from the reference, or in general two words, the chosen specific fragments are minimal absent words, also called minimal forbidden words. Target-specific words are factors of the target that are minimal absent words of the reference.

These types of factors associated with absent patterns are rather commonly used to efficiently compare sequences by avoiding complete alignments of full sequences, see, for example, [11] and references therein. In bioinformatics, target-specific words serve as signatures for newly sequenced biological molecules, helping to identify their characteristics. In the domain of molecular biology they allow the discovery of remarkable patterns in some genomic sequences, such as persitent patterns in the analysis of SARS-CoV-2 genomes that are absent in the human genome [31] and minimal sequences in Ebola virus also absent in the human genome [33]), or to build phylogenies of biological molecular sequences using a

[1] Corresponding Author

distance based on absent words (see [10, 32]). These patterns are also helpful to improve both pattern matching methods (see [15]) and text compression with the concept of antidictionaries (see for example [17, 1]). They are also central in some approaches related to the sanitisation of data, that is, the process of hiding confidential information [5, 6], to quote only a few applications.

The notion of a minimal absent word was introduced by Mignosi et al. [27] (see also [4]) in relation to combinatorial aspects of certain sequences. The first linear-time computation of the minimal absent words of the factors of a single sequence is described in [16] (see also [14]). Its time complexity is $O(n)$ on a fixed-size alphabet for a sequence of size n. The algorithm uses the computation of the directed acyclic word graph (DAWG), also called suffix automaton (see [8, 14]), of a single sequence. The same time complexity holds for an integer alphabet of polynomial size (see [19], [20]). This result is obtained using the $O(n)$ time complexity for computing the DAWG of a single sequence, in the case of an integer alphabet of polynomial size. For a general ordered alphabet A, the running time becomes $O(n \log |A|)$.

These algorithms extend to the computation of the minimal absent words of the set of factors of a finite set of sequences of total size n. This is done in [3] in $O(n)$ for a fixed-size alphabet. It becomes $O(n \log |A|)$ for a general ordered alphabet. It is done in $O(n)$ in [29] for an integer alphabet of polynomial size, extending the linear time computation of the DAWG in [19] and [20] to a finite set of sequences. It is also mentioned in [2] that the DAWG of a finite set of sequences can be computed in $O(n)$ time using sparse matrices for an integer alphabet of polynomial size.

Due to the significant role of the notion, the efficient computation of minimal absent words has attracted considerable attention (see, for example, [30] and references therein).

In this article, we continue exploring the approach of target-specific words as in [23] by introducing new algorithmic techniques to detect them. A more general view of the usefulness of formal languages in analyzing a series of genomes using pangenomic graphs is described by Bonizzoni et al. in [9].

A preliminary version of this paper was presented at the DLT 2023 conference [2]. In the present article, the motivation is strengthened due to additional references, the presentation of algorithms has been improved and complete proofs have been added. In addition, Section 5 reveals a surprising phenomenon on the tight link between DAWG matching and the search for minimal absent words, a technique at the core of our solution in Section 4.

The results

We design two algorithms that use intensively the notion of suffix links of indexing data structures, such as suffix trees (see [21, 14]) and DAWGs. The links can also be simulated using suffix arrays [26] and their implementations, such as the FM-index [18]. The algorithm in [23] uses the FMD index of Li [25]. All these data structures can accommodate the sequences and their reverse complements.

First, we address the computation of the set of target-specific factors of a target T against a reference R, where T and R are finite sets of sequences over an alphabet A. The result is the construction of an automaton that accepts the set of all the target-specific factors considered. This automaton is a digital tree whose leaves correspond to the specific words. The construction algorithm runs in linear time according to the size n of $T \cup R$, when A has a fixed size. The time complexity is $O(n \log |A|)$ when A is an ordered alphabet. Further, using the result of [29], the running time becomes $O(n)$ when A is an integer alphabet of

polynomial size according to n. Our algorithm uses a marking technique of the DAWG of a finite set of sequences, very close to the skip links used in [29], to compute minimal absent words using a DAWG.

The second result shows the design of an algorithm to compute all the occurrences in a single sequence T of its target-specific factors against a reference R. The algorithm runs in real-time on the target sequence over a fixed-size alphabet and independently of the number of occurrences of target-specific factors. This is obtained after standard processing of the DAWG of the reference, similarly as above. This improves on the result in [23], where the running time of the main algorithm depends on the number of occurrences of sought factors.

Definitions

Let A be a finite alphabet and A^* be the set of finite words drawn from the alphabet A, including the empty word ε. A *language* is a set of finite words. The concatenation of two words u, v is denoted by uv, and, if $x = uv$, v is also denoted by $u^{-1}x$. A *factor* of a word $x \in A^*$ is a word $v \in A^*$ that satisfies $x = uvw$ for some words $u, w \in A^*$. A *proper factor* of x is a factor distinct from the whole word. If P is a set of words, we denote by $\mathrm{Fact}(P)$ the set of factors of words in P, and, if P is finite, $\mathrm{size}(P)$ denotes the sum of lengths of the words in P. A language L is *factorial* is each factor of a word of L belongs to L.

A *minimal absent word* (also called a minimal forbidden word) for a given factorial language $L \subseteq A^*$ with respect to a given alphabet B containing A is a word of B^* that does not belong to L but whose all proper factors do.

Let R, T be two sets of finite words. A *T-specific word with respect to R* is a word u such that: u is a factor of a word in T, u is not a factor of a word in R and any proper factor of u is a factor of a word in R. The set R is called the *reference* and T the *target* of the problem.

Note that a word is a T-specific word with respect to R if and only if it is a minimal absent word of $\mathrm{Fact}(R)$, with respect to the alphabet of letters occurring in $R \cup T$, and is also in $\mathrm{Fact}(T)$. As a consequence, the set of T-specific words with respect to R is both prefix-free (i.e., no word of the set is a prefix of another word of the set) and suffix-free (i.e., no word of the set is a suffix of another word of the set).

It follows from the definition that the set S of T-specific words with respect to R is:

$$\mathrm{Fact}(T) \cap (A^* - \mathrm{Fact}(R)) \cap A\,\mathrm{Fact}(R) \cap \mathrm{Fact}(R)A,$$

where A is the alphabet of letters of words R and T. It is thus a regular language when R and T are regular, in particular, when R and T are finite.

A *finite deterministic automaton* is denoted by $\mathcal{A} = (Q, A, i, F, \delta)$, where A is a finite alphabet, Q is its finite set of states, $i \in Q$ is the unique initial state, $F \subseteq Q$ is the set of final states and δ is the partial function from $Q \times A$ to Q representing the transitions of the automaton. The partial function δ extends naturally to $Q \times A^*$ and a word u is accepted by $\mathcal{A}$ if and only if $\delta(i, u)$ is defined and belongs to F.

2 Background: directed acyclic word graph

In this section, we recall the definition and sketch the construction of the directed acyclic word graph of a finite set of words. This description already appears in [3].

Let $P = \{x_1, x_2, \ldots, x_r\}$ be a finite set of r finite words. A linear-time construction of a deterministic finite state automaton recognizing $\mathrm{Fact}(P)$ has been obtained by Blumer *et al.* in [8, 7] (see also [28]). Their construction is an extension of the well-known incremental

construction of the suffix automaton of a single word (see for instance [12, 14]). The words are added one by one to the automaton. In the sequel, we call this algorithm the DAWG algorithm since it outputs a deterministic automaton called a *directed acyclic word graph*. Let us denote it by $DAWG(P) = (Q, A, i, Q, \delta)$. Let $Suff(v)$ denote the set of suffixes of a word v and $Suff(P)$ the union of all $Suff(v)$ for $v \in P$.

The states of $DAWG(P)$ are the equivalence classes of the right invariant equivalence $\equiv_{Suff(P)}$ defined as follows. For $u, v \in \mathrm{Fact}(P)$,

$$u \equiv_{Suff(P)} v \text{ iff } \forall i \in [1, r] \quad u^{-1}Suff(x_i) = v^{-1}Suff(x_i).$$

There is a transition labelled by a from the class of a word u to the class of ua. The automaton $DAWG(P)$ has a unique initial state, which is the class of the empty word, and all its states are final. Note that the (syntactic) congruence $\sim$ defining the minimal automaton of the language is

$$u \sim v \text{ iff } \bigcup_{i=1}^{r} u^{-1}Suff(x_i) = \bigcup_{i=1}^{r} v^{-1}Suff(x_i),$$

and is not the same as the below equivalence. In other words, $DAWG(P)$ is not always a minimal automaton.

The construction of $DAWG(P)$ is performed in time $O(\mathrm{size}(P) \times \log |A|)$ in [8]. A time complexity of $O(\mathrm{size}(P))$ can be obtained for an integer alphabet of polynomial size in [19] and [20]. It can also be obtained with an implementation of automata with sparse matrices (see [14, Exercise 1.15]).

An essential element of the efficient construction is the notion of suffix links between states, denoted by s. We first define the function s' from $\mathrm{Fact}(P) \setminus \{\varepsilon\}$ to $\mathrm{Fact}(P)$ as follows: for any $v \in \mathrm{Fact}(P) \setminus \{\varepsilon\}$,

$$s'(v) \text{ is the longest suffix of } v \text{ that satisfies } u \not\equiv_{Suff(P)} v.$$

Then, we define the partial function s from Q to Q as follows. When $p = \delta(i, v)$ for $v \neq \varepsilon$, $s(p)$ is the state $\delta(i, s'(v))$. The function s is not defined on the initial state i.

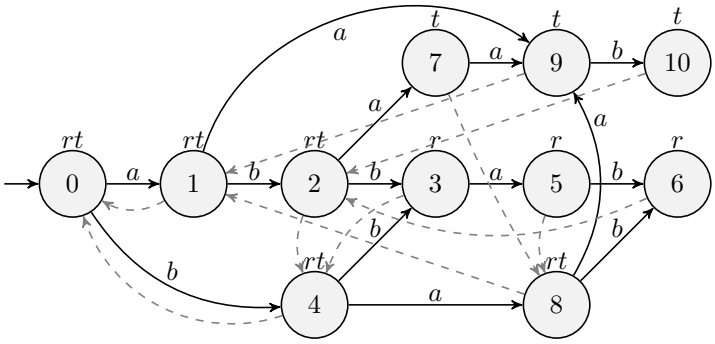

Figure 1 Automaton $DAWG(P)$ for $P = \{abbab, abaab\}$.

▶ **Example 1.** The DAWG obtained with the DAWG algorithm applied to $P = \{abbab, abaab\}$ is displayed in Figure 1. Dashed edges represent the suffix links and marks r, t on states relate to the reference word *abbab* and the target *abaab*. Note that this deterministic automaton is not minimal, as the states 6 and 10, 5 and 9, and 3 and 7, respectively, can be merged pairwise.

3 Computing the set of T-specific words

In this section, we assume that the reference R and the target T are two finite sets of words and our goal is to compute the set of T-specific factors of T against R. To do so, we first compute the directed acyclic word graph $DAWG(R \cup T) = (Q, A, i, Q, \delta)$ of $R \cup T$. Further, we compute a table, called mark, indexed by the set of states Q and that satisfies: for each state p in Q, mark$[p]$ is one of the three values r, t or both r, t according to the fact that each word labeling a path from i to p is a factor of some word in R, in T or in both. This information can be obtained during the construction of the directed acyclic word graph without increasing the running time.

The following algorithm outputs the set of T-specific words occurring in T with respect to R in the form of a trie (digital tree, see [14]).

SPECIFIC-TRIE((Q, A, i, Q, δ) DAWG of $(R \cup T)$, s its suffix link)

```
1  for each p ∈ Q with mark[p] = r, t in width-first search from i do
2      for each a ∈ A do
3          if δ(p, a) = q with mark[q] = r, t not reached yet then
4              δ'(p, a) ← q
5          elseif (δ(p, a) defined with mark[δ(p, a)] = t) and
               (p = i or δ(s(p), a) defined with mark[δ(s(p), a)] = r or r, t) then
6              δ'(p, a) ← new sink
7  return (Q, A, i, {sinks}, δ')
```

▶ **Proposition 2.** *Let $DAWG(R \cup T)$ be the output of Algorithm* DAWG *on the finite set of words $R \cup T$, let s be its suffix function, and let* mark *be the table defined as above. Algorithm* SPECIFIC-TRIE *builds the trie recognizing the set of T-specific words with respect to R.*

Proof. Let S be the set of T-specific words with respect to R.

Consider a word ua ($a \in A$) accepted by the automaton $\mathcal{A} = (Q, A, i, \{\text{sinks}\}, \delta')$ returned by the algorithm. Note that $\mathcal{A}$ accepts only nonempty words. Let $p = \delta'(i, u)$. Since the *DAWG* automaton is processed with a width-first search, u is the shortest word for which $\delta(i, u) = p$. Therefore, if $u = bv$ with $b \in A$, we have $\delta(i, v) = s(p)$ by definition of the suffix function s. When the test "($\delta(p, a)$ defined and mark$[\delta(p, a)] = t$) and ($\delta(s(p), a)$ defined and mark$[\delta(s(p), a)] = r$ or r, t)" is satisfied, this implies that $va \in \text{Fact}(R)$. Thus, $bva \notin \text{Fact}(R)$, while $bv, va \in \text{Fact}(R)$ and $bva \in \text{Fact}(T)$. So, ua is a T-specific word with respect to R. If u is the empty word, then $p = i$. The transition from i to the sink labeled by a is created under the condition "$\delta(p, a)$ defined and mark$[\delta(p, a)] = t$", which means that $a \in \text{Fact}(T)$. The word a is again a T-specific word with respect to R. Thus the words accepted by $\mathcal{A}$ are T-specific words with respect to R.

Conversely, let $ua \in S$. If u is the empty word, this means that a does not occur in $\text{Fact}(R)$ and occurs in $\text{Fact}(T)$ therefore there is a transition labeled by a from i in $DAWG(R \cup T)$ to a state marked t. Thus, a transition from i to a sink state in $\mathcal{A}$ is created in line 6, and a is accepted by $\mathcal{A}$. Now assume that $u = bv$. The word u is in $\text{Fact}(R)$. So let $p = \delta(i, u)$. Note that u is the shortest word for which $p = \delta(i, u)$, because all such words are suffixes of each other in the *DAWG* automaton. The word ua is not in $\text{Fact}(R)$ and is in $\text{Fact}(T)$, so the condition "$\delta(p, a)$ defined and mark$[p, a] = t$" is satisfied. Let $q = s(p)$. We have $q = \delta(i, v)$ because of the minimality of the length of u and the definition of s. Since va is in $\text{Fact}(R)$, the condition "$\delta(s(p), a)$ defined and mark$[\delta(s(p), a)] = r$ or r, t" at line 5 is satisfied. This results in the creation of a transition at line 6, enabling $\mathcal{A}$ to accept ua as desired. ◀

▶ **Example 3.** The automaton $DAWG(R \cup T)$, where $R = \{abbab\}$ and $T = \{abaab\}$ is shown in Figure 1. The output of Algorithm Specific-trie applied to it is shown in Figure 2, where the black squares are the accepting sink states of the trie. The set of T-specific words with respect to R is $\{aa, aba\}$.

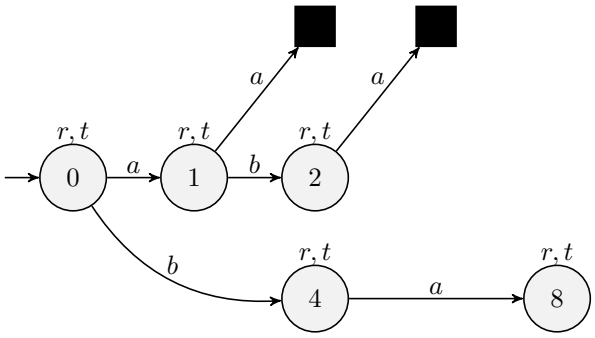

Figure 2 The trie of T-specific words with respect to R.

A main point in algorithm Specific-trie is that it uses the function s defined on states of the input DAWG. It is not possible to proceed similarly when considering the minimal factor automaton of $\mathrm{Fact}(R \cup T)$ because there is no analogue function s. However, it is possible to reduce the automaton $DAWG(R \cup T)$ by merging states having the same future (right context) and the same image by s. For example, on the DAWG of Figure 1, states 6 and 10 can be merged because $s(6) = s(10) = 2$. States 3 and 7, nor states 5 and 9 cannot be merged with the same argument.

▶ **Proposition 4.** *Algorithms* Dawg *and* Specific-trie *together run in time* $O(\mathrm{size}(R \cup T) \times |A|)$ *when applied to reference* R *and target* T*, two finite sets of words, if the transition functions are implemented by transition matrices. This complexity is* $O(\mathrm{size}(R \cup T) \times \log |A|)$ *for an ordered alphabet, and* $O(\mathrm{size}(R \cup T))$ *for an integer alphabet of polynomial size.*

Proof. The running time depends on the time for computing the DAWG of $R \cup T$. The relies on the time for computing the transitions of the DAWG, that is $\delta(q, a)$, which is constant for an integer alphabet of polynomial size, due to [29]. ◀

For P a set of words, we denote by A_P the set of letters occurring in P.

▶ **Proposition 5.** *Let* R, T *be two finite sets of words. The number of* T*-specific words with respect to* R *is no more than* $(2\,\mathrm{size}(R) - 2)(|A_R| - 1) + |A_T \setminus A_R| - |A_R| + m$*, if* $\mathrm{size}(R) > 1$*, where* m *the number of words in* R*. The bound becomes* $|A_T \setminus A_R|$ *when* $\mathrm{size}(R) \leq 1$*.*

Proof. We let S denote the set of T-specific words with respect to R. Since S is included in the set of minimal absent words of $\mathrm{Fact}(R)$ with respect to the alphabet $A = A_R \cup A_T$, the bound comes from [3, Corollary 4.1]. ◀

In conclusion, the algorithm generates a trie of minimal absent words, which represent potentially interesting patterns. This trie can be used to explore the set of patterns that align with the application's objectives.

4 Computing occurrences of target-specific factors: the T-specific table

In this section, we consider that T is a single word and R is a finite set of words as before. The goal of the section is to design an algorithm that computes all the occurrences in T of words that are T-specific with respect to R. To do so, we define the T-specific table associated with the pair R, T of words of the problem.

A letter of T at position k is denoted by $T[k]$ and $T[i\,..\,j]$ denotes the factor $T[i]T[i+1]\cdots T[j]$ of T. Then, the T-specific table Ts is defined, for $i = 0, \ldots, |T|-1$, by

$$Ts[i] = \begin{cases} j, & \text{if } T[i\,..\,j] \text{ is } T\text{-specific}, i \le j, \\ -1, & \text{else.} \end{cases}$$

Note that the set of T-specific factors is prefix-free, that is, no proper prefix of an element of the set is also in the set. (The set of T-specific factors is also suffix-free.) Therefore, for each position k on T, there is at most one T-specific factor of T starting at k (and for each position j on T there is at most one T-specific factor of T ending at j).

Instead of computing the T-specific table Ts, in a straightforward way, the algorithm below can be transformed to compute the list of pairs (i, j) of positions on T for which $Ts[i] = j$ and $j \neq -1$.

To compute the table we use $\mathcal{R}$, the Suffix automaton or rather the DAWG of R. The former is the minimal automaton accepting the suffixes of R (see [14, Section[5.4]) and the latter has the same structure but with all states as terminal states instead (see [8]). As such, it accepts $\mathrm{Fact}(R)$ the set of factors of R.

The automaton is given with its transition function δ, its initial state i and and is equipped with both the suffix link s (used here as a failure link) and the length function ℓ defined on states. The function ℓ is defined by: $\ell[p] = \max\{|z| \in A^* \mid \delta(i, z) = p\}$. The functions s and ℓ transform the automaton into a search machine (see [14, Section 6.6]).

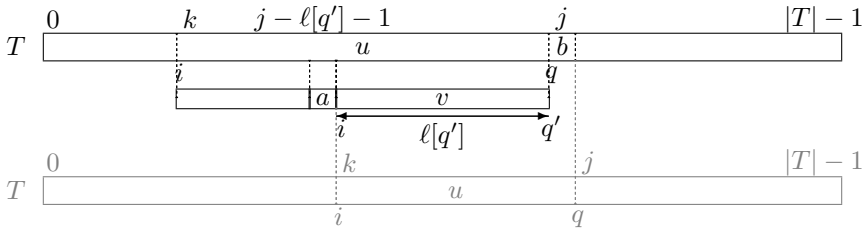

Figure 3 A T-specific word found: when $u \in \mathrm{Fact}(R)$ and $ub \notin \mathrm{Fact}(R)$, either avb or b is a T-specific factor with respect to R (a, b are letters). The gray bottom displays the situation, and specifically the variables q and j used in Algorithm TsTable, after processing letter b.

Figure 3 illustrates the principle of Algorithm TsTable. Let us assume that the factor $u = T[k\,..\,j-1]$ is a factor of R but ub is not for some letter b. Then, let v be the longest suffix of u for which vb is a factor of R. If it exists, then clearly avb, with the letter a preceding v, is T-specific. Indeed, $av, vb \in \mathrm{Fact}(R)$ and $avb \notin \mathrm{Fact}(R)$, which means that avb is a minimal absent word of R while occurring in T. Therefore, setting $q' = \delta(i, v)$ and since $|v| = \ell[q']$, the minimal absent word avb is identified by setting $Ts[j - \ell[q'] - 1] = j$. If there is no suffix of u satisfying the condition, the letter b alone is T-specific and is identified by setting $Ts[j] = j$.

```
TsTable(T target word, R DAWG(R), i initial(R))
 1  (q, j) ← (i, 0)
 2  while j < |T| do
 3        Ts[j] ← −1
 4        if δ(q, T[j]) undefined then
 5              while q ≠ i and δ(q, T[j]) undefined do
 6                    q ← s[q]
 7              if δ(q, T[j]) undefined then      ▷ q = i
 8                    Ts[j] ← j
 9              else Ts[j − ℓ[q] − 1] ← j
10                    q ← δ(q, T[j])
11        else q ← δ(q, T[j])
12        j ← j + 1
13  return Ts
```

▶ **Theorem 6.** *Algorithm* TsTable *computes the T-specific table with respect to R and runs in linear time, that is, $O(|T|)$ on a fixed-size alphabet.*

Proof. The algorithm implements the ideas detailed above and illustrated by Figure 3. The formal proof relies on the following invariant condition of the main while loop: let u be a factor of R and q be a state of $\mathcal{R}$ that satisfies $q = \delta(i, u)$, and let j be the current position on T, then u is the longest factor of R ending at position $j - 1$ on T. Before the first pass in the main while loop starts, u is the empty word, $q = i$, $j = 0$, and the condition is satisfied.

During each pass in the main while loop, by default $Ts[j]$ is first set to -1 (line 3) to cover the possibility that no minimal absent word ends at position j. Let us examine what is done during a pass in the while loop after the instruction at line 3.

- If $\delta(q, T[j])$ is defined at line 4, the next value of q is set at line 11, followed by the increment of j at line 12. Therefore, $uT[j]$ is the longest factor of R ending at $j - 1$, as required for the invariant to hold.
- The case where $\delta(q, T[j])$ is undefined at line 4 is illustrated on Figure 3. Then, the loop at lines 5-6 uses the suffix link s to finds the longest suffix v of u for which $vb = vT[j]$ is a factor of R by the definition of s in the DAWG.
  - If the execution of the loop terminates with $\delta(q, T[j])$ still undefined, this indicates that q is the initial state and that v is the empty word. Consequently, the letter $T[j]$ is the minimal absent word at position j.
  - Otherwise, first note that v is shorter than u because ub is not a factor of R. Thus, there exists a letter a such that av is a suffix of u. Therefore, avb is the minimal absent word ending at position j because $av, vb \in \mathrm{Fact}(R)$ but $avb \notin \mathrm{Fact}(R)$. This is established at line 9 because the position of letter a is $j - \ell[q] - 1$. The subsequent execution of lines 9 and 10 ensures that the invariant condition holds in this case as well.

As for the running time, note that the instructions at lines 3 and 7-12 execute in constant time for each value of j. All the executions of the instruction at line 6 execute in time $O(|T|)$ because the link s reduces strictly the potential length of the T-specific word ending at j, incrementing the starting position $j - \ell[q] - 1$ of v in the picture.

Moreover, each computation of a transition $\delta(q, T[j])$ executes in constant time on a fixed-size alphabet if, for example, the DAWG is implemented with a sparse matrix technique.

Thus, the entire execution is completed in time $O(|T|)$. ◀

Algorithm TsTable can be improved to run in real-time on a fixed-size alphabet. This is done by optimising the suffix link s defined on the automaton $\mathcal{R}$. To do so, let us define, for each state q of $\mathcal{R}$,

$$\mathsf{out}(q) = \{a \mid \delta(q, a) \text{ defined for letter } a\}.$$

Then, the optimised suffix link G is defined by $G[\mathit{initial}(\mathcal{R})] = \mathbf{nil}$ and, for any other state q of $\mathcal{R}$, by

$$G[q] = \begin{cases} s[q], & \text{if } \mathsf{out}(q) \subset \mathsf{out}(s[q]), \\ G[s[q]], & \text{else.} \end{cases}$$

Note that, since we always have $\mathsf{out}(q) \subseteq \mathsf{out}(s[q])$, the definition of G can be reformulated as

$$G[q] = \begin{cases} s[q], & \text{if } \mathsf{deg}(q) < \mathsf{deg}(s[q]), \\ G[s[q]], & \text{else,} \end{cases}$$

where deg is the outgoing degree of a state. Therefore, its computation can be performed in linear time with respect to the number of states of $\mathcal{R}$. After substituting G for s in Algorithm TsTable, when the alphabet is of size α the instruction at line 6 executes no more than α times for each value of q. So the time to process a given state q is constant. This is summarized in the next corollary.

▶ **Corollary 7.** *When using the optimised suffix link, Algorithm TsTable runs in real time on a fixed-size alphabet.*

On a more general alphabet of size α, processing a given state of the automaton can be done in time $O(\log \alpha)$.

5 Absent word searching vs string matching

Searching for absent factor occurrences as done in the previous section is based on the DAWG matching technique [13] (see also [14, Section 6.6]) in the domain of string matching algorithms. As such, Algorithm TsTable looks like a side product of string matching.

The goal of string matching algorithms is to locate a given pattern p in a longer text T. Figure 3 displays the generic situation when a part u of the pattern has been found in T and is about to be appended with the letter b. The extension of u is successful if b matches its aligned letter of the pattern, which eventually can lead to the detection of an occurrence of the whole pattern.

The situation in Figure 3 is also the generic situation when searching for minimal absent words. However, in contrast, this is an unsuccessful match of the letter b that immediately yields the detection in T of a minimal absent word of p.

The role of the DAWG matching technique is essential here to detect minimal absent words of p occurring in T because the DAWG of p stores and provides direct access to all factors of p. Instead, if an online string matching algorithm is used, like KMP [24] or Simon-Hancart [34, 22] algorithms (see also [14, Chapter 2]), the algorithm will detect only some absent words. That is, those of the form aub in which au is a factor of p but ub is only a prefix of it. This does not produce all the minimal absent words. However, other types of string matching based on text indexes can certainly be used for the same purpose.

As a conclusion, the algorithm designed in the previous section reveals a surprising phenomenon: the very tight link between the existence of minimal absent words that pop up naturally in the analysis of pattern searching based on a Suffix automaton or a DAWG.

References

1 Lorraine A. K. Ayad, Golnaz Badkobeh, Gabriele Fici, Alice Héliou, and Solon P. Pissis. Constructing antidictionaries of long texts in output-sensitive space. *Theory Comput. Syst.*, 65(5):777–797, 2021. doi:10.1007/S00224-020-10018-5.

2 Marie-Pierre Béal and Maxime Crochemore. Fast detection of specific fragments against a set of sequences. In *27th International Conference on Developments in Language Theory, DLT 2023*, volume 13911 of *Lecture Notes in Computer Science*, pages 51–60, 2023. doi:10.1007/978-3-031-33264-7_5.

3 Marie-Pierre Béal, Maxime Crochemore, Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. Computing forbidden words of regular languages. *Fundam. Informaticae*, 56(1-2):121–135, 2003. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi56-1-2-08.

4 Marie-Pierre Béal, Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. Forbidden words in symbolic dynamics. *Adv. Appl. Math.*, 25(2):163–193, 2000. doi:10.1006/aama.2000.0682.

5 G. Bernardini, C. Liu, G. Loukides, A. Marchetti-Spaccamela, S.P. Pissis, L. Stougie, and M. Sweering. Missing value replacement in strings and applications. *Data Mining and Knowledge Discovery*, 39(12), 2025. doi:10.1007/s10618-024-01074-3.

6 Giulia Bernardini, Alessio Conte, Garance Gourdel, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, Giulia Punzi, Leen Stougie, and Michelle Sweering. Hide and Mine in Strings: Hardness, Algorithms, and Experiments . *IEEE Transactions on Knowledge & Data Engineering*, 35(06):5948–5963, June 2023. doi:10.1109/TKDE.2022.3158063.

7 A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987. doi:10.1145/28869.28873.

8 Anselm Blumer, J. Blumer, Andrzej Ehrenfeucht, David Haussler, and Ross M. McConnell. Building the minimal DFA for the set of all subwords of a word on-line in linear time. In Jan Paredaens, editor, *Automata, Languages and Programming, 11th Colloquium, Antwerp, Belgium, July 16-20, 1984, Proceedings*, volume 172 of *Lecture Notes in Computer Science*, pages 109–118. Springer, 1984. doi:10.1007/3-540-13345-3_9.

9 Paola Bonizzoni, Clelia De Felice, Yuri Pirola, Raffaella Rizzi, Rocco Zaccagnino, and Rosalba Zizza. Can formal languages help pangenomics to represent and analyze multiple genomes? In Volker Diekert and Mikhail V. Volkov, editors, *Developments in Language Theory - 26th International Conference, DLT 2022, Tampa, FL, USA, May 9-13, 2022, Proceedings*, volume 13257 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2022. doi:10.1007/978-3-031-05578-2_1.

10 Supaporn Chairungsee and Maxime Crochemore. Using minimal absent words to build phylogeny. *Theor. Comput. Sci.*, 450:109–116, 2012. doi:10.1016/J.TCS.2012.04.031.

11 Panagiotis Charalampopoulos, Maxime Crochemore, Gabriele Fici, Robert Mercas, and Solon P. Pissis. Alignment-free sequence comparison using absent words. *Inf. Comput.*, 262:57–68, 2018. doi:10.1016/j.ic.2018.06.002.

12 Maxime Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45(1):63–86, 1986. doi:10.1016/0304-3975(86)90041-1.

13 Maxime Crochemore. Longest common factor of two words. In Ehrig, Kowalski, Levi, and Montanari, editors, *TAPSOFT'87 (Pisa, 1987)*, number 249 in LNCS, pages 26–36. Springer-Verlag, 1987. doi:10.1007/3-540-17660-8_45.

14 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007. 392 pages.

15 Maxime Crochemore, Alice Héliou, Gregory Kucherov, Laurent Mouchard, Solon P. Pissis, and Yann Ramusat. Absent words in a sliding window with applications. *Inf. Comput.*, 270, 2020. doi:10.1016/j.ic.2019.104461.

16 Maxime Crochemore, Filippo Mignosi, and Antonio Restivo. Automata and forbidden words. *Inf. Process. Lett.*, 67(3):111–117, 1998. doi:10.1016/S0020-0190(98)00104-5.

17 Maxime Crochemore, Filippo Mignosi, Antonio Restivo, and Sergio Salemi. Data compression using antidictionaries. *Proc. IEEE*, 88(11):1756–1768, 2000. doi:10.1109/5.892711.

18 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 390–398. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892127.

19 Yuta Fujishige, Yuki Tsujimaru, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing dawgs and minimal absent words in linear time for integer alphabets. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 38:1–38:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.MFCS.2016.38.

20 Yuta Fujishige, Yuki Tsujimaru, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Linear-time computation of dawgs, symmetric indexing structures, and maws for integer alphabets. *Theor. Comput. Sci.*, 973:114093, 2023. doi:10.1016/J.TCS.2023.114093.

21 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

22 Christophe Hancart. On simon's string searching algorithm. *Inf. Process. Lett.*, 47(2):95–99, 1993. doi:10.1016/0020-0190(93)90231-W.

23 Parsoa Khorsand, Luca Denti, Human Genome Structural Variant Consortium, Paola Bonizzoni, Rayan Chikhi, and Fereydoun Hormozdiari. Comparative genome analysis using sample-specific string detection in accurate long reads. *Bioinformatics Advances*, 1(1), May 2021. doi:10.1093/bioadv/vbab005.

24 Donald E. Knuth, J. H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

25 Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, May 2012. doi:10.1093/bioinformatics/bts280.

26 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

27 Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. Forbidden factors in finite and infinite words. In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Paun, and Grzegorz Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 339–350. Springer, 1999.

28 Gonzalo Navarro and Mathieu Raffinot. *Flexible pattern matching in strings—practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002. 232 pages. doi:10.1017/CBO9781316135228.

29 Kouta Okabe, Takuya Mieno, Yuto Nakashima, Shunsuke Inenaga, and Hideo Bannai. Linear-time computation of generalized minimal absent words for multiple strings. In Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini, editors, *String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26-28, 2023, Proceedings*, volume 14240 of *Lecture Notes in Computer Science*, pages 331–344. Springer, 2023. doi:10.1007/978-3-031-43980-3_27.

30 Armando J. Pinho, Paulo Jorge S. G. Ferreira, Sara P. Garcia, and João M. O. S. Rodrigues. On finding minimal absent words. *BMC Bioinform.*, 10, 2009. doi:10.1186/1471-2105-10-137.

31 Diogo Pratas and Jorge M Silva. Persistent minimal sequences of sars-cov-2. *Bioinformatics*, 36(21):5129–5132, July 2020. doi:10.1093/bioinformatics/btaa686.

32 Mohammad Saifur Rahman, Ali Alatabbi, Tanver Athar, Maxime Crochemore, and M. Sohel Rahman. Absent words and the (dis)similarity analysis of dna sequences: an experimental study. *BMC Research Notes*, 9(186):1–8, 2016. doi:10.1186/s13104-016-1972-z.

33 Raquel M. Silva, Diogo Pratas, Luísa Castro, Armando J. Pinho, and Paulo Jorge S. G. Ferreira. Three minimal sequences found in ebola virus genomes and absent from human DNA. *Bioinform.*, 31(15):2421–2425, 2015. doi:10.1093/bioinformatics/btv189.

34 Imre Simon. String matching algorithms and automata. In Juhani Karhumäki, Hermann A. Maurer, and Grzegorz Rozenberg, editors, *Results and Trends in Theoretical Computer Science, Colloquium in Honor of Arto Salomaa, Graz, Austria, June 10-11, 1994, Proceedings*, volume 812 of *Lecture Notes in Computer Science*, pages 386–395. Springer, 1994. doi:10.1007/3-540-58131-6_61.