Space-efficient detection of unusual words

Djamal Belazzougui\textsuperscript{1,2} and Fabio Cunial\textsuperscript{3}

\textsuperscript{1}Department of Computer Science, University of Helsinki, Finland.
\textsuperscript{2}Helsinki Institute for Information Technology, Finland.
\textsuperscript{3}Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany.

August 13, 2015

Abstract

Detecting all the strings that occur in a text more frequently or less frequently than expected according to an IID or a Markov model is a basic problem in string mining, yet current algorithms are based on data structures that are either space-inefficient or incur large slowdowns, and current implementations cannot scale to genomes or metagenomes in practice. In this paper we engineer an algorithm based on the suffix tree of a string to use just a small data structure built on the Burrows-Wheeler transform, and a stack of $O(\sigma^2 \log^2 n)$ bits, where $n$ is the length of the string and $\sigma$ is the size of the alphabet. The size of the stack is $o(n)$ except for very large values of $\sigma$. We further improve the algorithm by removing its time dependency on $\sigma$, by reporting only a subset of the maximal repeats and of the minimal rare words of the string, and by detecting and scoring candidate under-represented strings that do not occur in the string. Our algorithms are practical and work directly on the BWT, thus they can be immediately applied to a number of existing datasets that are available in this form, returning this string mining problem to a manageable scale.

1 Introduction

Detecting all the patterns of a string whose number of occurrences matches some notion of statistical surprise is a fundamental requirement of the post-genome era, in which textual datasets grow faster than the ability to understand them, and in which over- and under-representation with respect to a statistical model is often an indicator of structure or function. The sheer volume of the available

\textsuperscript{*}This work was partially supported by Academy of Finland under grant 284598 (Center of Excellence in Cancer Genetics Research).
datasets makes even simple models of patterns and simple measures of statistical surprise useful in practice, if their detection scales to extremely long strings in reasonable time and space. In this paper we focus on the simplest possible model of a pattern – a string $W$, of any length, that occurs without mismatches $f_T(W)$ times in a text $T$ of length $n$ – and we consider measures of statistical surprise that score $W$ according to the expected number $E[f_T(W)]$ and to the variance $\mathbb{V}[f_T(W)]$ of the number of its occurrences in a random text of length $|T|$ (see e.g. [1] and references therein). We assume that the random source is a given Markov chain, and for concreteness we set its order to zero, since this simple case already captures the computational structure of the problem [2]. Moreover, we focus on computing $\mathbb{V}[f_T(W)]$, since computing expectations with respect to a Markov chain of order zero is trivial (see e.g. [3]).

$\mathbb{V}[f_T(W)]$ enjoys the remarkable property that its computation can be carried out by iterating over all the proper borders of $W$, i.e. over all the nonempty substrings of $W$ shorter than $W$ that are at the same time prefix and suffix of $W$: see [3] for a detailed derivation. To make the paper self-contained, here we just recall that $\mathbb{V}(f_T(W))$ can be computed in constant time from the functions $\phi(W)$ and $\gamma(W)$ defined below:

$$
\phi(W) = \sum_{b \in \mathcal{B}(W)} (n - 2|W| + b + 1) \cdot \pi(W[b..|W| - 1]) \\
\gamma(W) = \sum_{b \in \mathcal{B}(W)} \pi(W[b..|W| - 1])
$$

where strings are indexed from zero, $\pi(W) = \prod_{i=0}^{|W|-1} P[W_i]$, $W_i$ is the $i$th character of string $W$, $P[c]$ is the probability of character $c$ according to the given zero-order Markov chain, and $\mathcal{B}(W)$ is the set of all border lengths of $W$. Removing the components of $\mathbb{V}(f_T(W))$ that depend on borders can cause large relative errors in practice [2], so we focus on the exact computation of $\mathbb{V}(f_T(W))$. It is well known that borders have a recursive structure, in the sense that the set of borders of $W$ consists of the longest border $V$ of $W$, and of all the borders of $V$. This observation enables one to map the computation of $\mathbb{V}(f_T(W))$ for a given $W$ onto the Morris-Pratt algorithm [17], thus achieving time $O(|W|)$ in the worst case [3]. Specifically:

$$
\phi(W) = \delta(W) \cdot (\phi(B) - 2(|W| - |B|)\gamma(B) + n - 2|W| + |B| + 1) \\
\gamma(W) = \delta(W) \cdot (1 + \gamma(B))
$$

where $B$ is the longest border of $W$, and $\delta(W) = \pi(W[|B|..|W| - 1])$. In practice, however, we are interested in extracting from a string $T$ all its substrings $W$, of any length, such that a user-specified measure of surprise computed on $E[f_T(W)]$ and $\mathbb{V}[f_T(W)]$ is, say, greater than a threshold. Even though computing $\mathbb{V}(f_T(W))$ takes $O(|W|)$ time for a given $W$, enumerating and scoring in this way all substrings of a text $T$ of length $n$ takes $O(n^2)$ time.

Luckily, a number of statistical scores $z(W)$ enjoy the additional property that $z(XWY) \geq z(W)$ if $f_T(XWY) = f_T(W)$, where $X$ and $Y$ are strings
[1]. Consider then a set $\mathcal{A}$ of substrings of $T$ such that all substrings in the set have the same number of occurrences, and consider the partial order $\preceq$ on $\mathcal{A}$ such that $V \preceq W$ if $W = XVY$ for (possibly empty) strings $X$ and $Y$. If we display to the user just the maximal elements of $\mathcal{A}$ with respect to $\preceq$, we guarantee that every over-represented string $W \in \mathcal{A}$ that we do not output is a substring of a string $XWY \in \mathcal{A}$ in the output which has at least the same score. Symmetrically, if we display just the minimal elements of $\mathcal{A}$ with respect to $\preceq$, we guarantee that every under-represented string $XWY \in \mathcal{A}$ that we do not output is a superstring of a string $W \in \mathcal{A}$ in the output which has at most the same score. A possible choice for $\mathcal{A}$ is the set of all substrings that start at exactly the same positions in $T$: this class has a unique maximal element, which corresponds to a node of the suffix tree of $T$, and a unique minimal element, which corresponds to the right extension by a single character of a node of the suffix tree of $T$. Since the number of all such classes is $O(|T|)$, and since all such classes are connected to one another by a trie, known as the suffix-link tree, it is possible to devise an algorithm that computes $\forall f_T(W)$ for all the maximal and minimal elements of all such classes, in a total amount of time that grows linearly in $|T|$. [2].

In this paper we engineer the algorithm described in [2] to use as its substructure the Burrows-Wheeler transform of $T$, rather than space-inefficient data structures like the (truncated) suffix tree of $T$, or space-efficient simulations of the suffix tree with $O(\log^2 n)$ slowdown, like compressed suffix trees (see e.g. [11] and references therein). We also observe that the time complexity of the algorithm described in [2] depends on the cardinality of the alphabet, and we remove this dependency. Moreover, we adapt the algorithm to work on the smallest possible set of equivalence classes, thus reducing time and output size in practice. Assuming an alphabet of size $\sigma \in o(\sqrt{n}/\log n)$, we can thus perform all the computations described in [2] in $O(n)$ time and in $o(n+\lambda\sqrt{n})$ bits of space, given the BWT of $T$ and few additional data structures [7], where $\lambda$ is the length of a longest repeat of $T$. For statistical reasons, the maximum length of a string to be reported is often $O(\log^2 \sigma)$, thus our algorithm uses effectively $o(n)$ bits of space in addition to the input. Concatenating this setup to the BWT construction algorithm described in [5], we can discover all the over- and under-represented substrings of $T$, directly from $T$ itself, in randomized $O(n)$ time and in $O(n\log \sigma)$ bits of space in addition to $T$ itself. Finally, we extend the algorithm in [2] to consider potentially under-represented strings that do not occur in $T$, thus providing for the first time a way to score and rank the minimal absent words of $T$.

2 Preliminaries

2.1 Strings

Let $\Sigma = [1..\sigma]$ be an integer alphabet, let $\# = 0$ be a separator not in $\Sigma$, let $T = [1..\sigma]^{n-1}\#$ be a string, and let $\varepsilon$ be the empty string. For reasons that
will become clear in Section 2.2, we assume $\sigma \in o(\sqrt{n}/\log n)$ throughout the paper. We denote by $f_T(W)$ the number of (possibly overlapping) occurrences of a string $W$ in the circular version of $T$. A repeat $W$ is a string that satisfies $f_T(W) > 1$. We denote by $\Sigma_T^r(W)$ the set of characters $\{ a \in [0..\sigma] : f_T(aW) > 0 \}$ and by $\Sigma_T^f(W)$ the set of characters $\{ b \in [0..\sigma] : f_T(Wb) > 0 \}$. A repeat $W$ is right-maximal (respectively, left-maximal) iff $|\Sigma_T^r(W)| > 1$ (respectively, iff $|\Sigma_T^f(W)| > 1$). It is well known that $T$ can have at most $n - 1$ right-maximal substrings and at most $n - 1$ left-maximal substrings. A maximal repeat of $T$ is a repeat that is both left- and right-maximal. Clearly a maximal repeat $W$ of $T$ satisfies $f_T(aW) < f_T(W)$ and $f_T(Wb) < f_T(W)$ for any characters $a$ and $b$ in $\Sigma$. A repeat is supermaximal if it is not a proper substring of any other repeat. A minimal rare word of $T$ is a string $W$ that satisfies $f_T(W) < f_T(V)$ for every proper substring $V$ of $W$. Clearly $W$ must have the form $aXb$, where $a$ and $b$ are characters and $X$ is a maximal repeat of $T$. If $f_T(W) > 0$, then $aX$ is a right-maximal substring of $T$ and $Xb$ is a left-maximal substring of $T$. If $f_T(W) = 0$, then $aX$ (respectively, $Xb$) must occur in $T$, but it is not necessarily right-maximal (respectively, left-maximal). A minimal rare word of $T$ that does not occur in $T$ is called minimal absent word (see e.g. [8, 12] and references therein): the total number of such strings can be $\Theta(\sigma n)$ [9]. A minimal rare word of $T$ that occurs exactly once in $T$ is called minimal unique substring (see e.g. [13] and references therein). It is clear that the total number of minimal rare words of $T$ that occur at least once in $T$ is $O(n)$.

String $V \neq \varepsilon$ is a proper border of string $W$ if $W = VXW$ and $W = YV$ for nonempty strings $X$ and $Y$. A string $W$ can have zero, one, or multiple proper borders: we denote by $\text{bord}(W)$ the length of the longest border of $W$. Each border of $W$ is followed by a character when it occurs as a prefix, and it is preceded by a character when it occurs as a suffix: we use $a|W$ to denote the length of the longest border of $W$ that is preceded by character $a$ when it occurs as a suffix, and we use $W|a$ to denote the length of the longest border of $W$ that is followed by character $a$ when it occurs as a prefix. Clearly both $a|W$ and $W|a$ can be zero. We denote by $B(W)$ the set of lengths of all borders of $W$, by $B'(W)$ the set of pairs $\{(a, a|W) : a \in \sigma, a|W \neq 0 \}$, and by $B'(W)$ the set of pairs $\{(a, W|a) : a \in \sigma, W|a \neq 0 \}$. It is well known that $B(W) = \{ \text{bord}(W) \} \cup B(V)$, where $V$ is the longest border of $W$: see e.g. [10] and references therein. In this paper we will also use $\text{left}_W$ to denote an array of size $|\Sigma_T^r(W)|$, indexed by the characters in $\Sigma_T^r(W)$ in lexicographic order, such that $\text{left}_W[c] = W[c]$. Similarly, we will use $\text{right}_W$ to denote an array of size $|\Sigma_T^f(W)|$, indexed by the characters in $\Sigma_T^f(W)$ in lexicographic order, such that $\text{right}_W[c] = c|W$. Set $B(W)$ determines all possible ways in which $W$ can overlap with itself: specifically, the maximum possible number of occurrences of $W$ in a string of length $n$ is $f^*(W, n) = \lfloor (n - |W| + 1)/\text{period}(W) \rfloor$, where $\text{period}(W) = |W| - \text{bord}(W)$. When $f_T(W)$ needs to be compared to $f_T(W)$, where $|T| \neq |T|$, it is customary to divide $f_T(W)$ by $f^*(W, |T|)$. It is easy to see that the longest border of a random string of length $n$ generated by an IID source is expected to tend to a constant as $n$ tends to infinity.

For reasons of space we assume the reader to be familiar with the notion of
suffix tree $\mathbf{ST}_T$ of a string $T$, which we do not define here. We denote by $\ell(v)$ the string label of a node $v$ in a suffix tree. It is well known that a substring $W$ of $T$ is right-maximal iff $W = \ell(v)$ for some internal node $v$ of $\mathbf{ST}_T$. We assume the reader to be familiar with the notion of suffix link connecting a node $v$ with $\ell(v) = aW$ for some $a \in [0..\sigma]$ to a node $w$ with $\ell(w) = W$: we say that $w = \text{suffixLink}(v)$ in this case. Here we just recall that suffix links and internal nodes of $\mathbf{ST}_T$ form a tree, called the suffix-link tree of $T$ and denoted by $\mathbf{SLT}_T$, and that inverting the direction of all suffix links yields the so-called explicit Weiner links. Given an internal node $v$ and a symbol $a \in [0..\sigma]$, it might happen that string $a\ell(v)$ does occur in $T$, but that it is not right-maximal, i.e. it is not the label of any internal node of $\mathbf{ST}_T$: all such left extensions of internal nodes that end in the middle of an edge are called implicit Weiner links. An internal node $v$ of $\mathbf{ST}_T$ can have more than one outgoing Weiner link, and all such Weiner links have distinct labels: in this case, $\ell(v)$ is a maximal repeat. It is known that the number of suffix links (or, equivalently, of explicit Weiner links) is upper-bounded by $2n-2$, and that the number of implicit Weiner links can be upper-bounded by $2n-2$ as well.

If $V$ is a nonempty proper border of $W$, then $\Sigma_r^f(W) \subseteq \Sigma_r^f(V)$ and $\Sigma_r(W) \subseteq \Sigma_r(V)$. Thus, if $W$ is right-maximal (respectively, left-maximal) then $V$ is right-maximal (respectively, left-maximal); if $W$ is a maximal repeat, then $V$ is a maximal repeat; and if $W = aX$ where $a \in \Sigma$ and $X$ is a maximal repeat, then $V = aY$ where $Y$ is a maximal repeat.\footnote{Thus, maximal repeats connected by longest border relationships form a tree rooted at the empty string: the path from the root to a maximal repeat lists all its borders, and the internal nodes of this tree cannot be supermaximal repeats. Similarly, longest border relationships and strings $aW$ (respectively, $Wa$) where $W$ is a maximal repeat, form a tree rooted at the empty string.}

### 2.2 Enumerating maximal repeats and minimal rare words

For reasons of space we assume the reader to be familiar with the notion and uses of the Burrows-Wheeler transform of $T$, including the $C$ array, the rank function, and backward searching. In this paper we use $\mathbf{BWT}_T$ to denote the BWT of $T$, we use $\text{range}(W) = [\text{sp}(W)..<\text{ep}(W)]$ to denote the lexicographic interval of a string $W$ in a BWT that is implicit from the context, and we use $\Sigma_{i,j}$ to denote the set of distinct characters that occur inside interval $[i..j]$ of a string that is implicit from the context. We also denote by $\text{rangeDistinct}(i, j)$ the function that returns the set of tuples $\{(c, \text{rank}(c, p_c), \text{rank}(c, q_c)) : c \in \Sigma_{i,j}\}$, in any order, where $p_c$ and $q_c$ are the first and the last occurrence of character $c$ inside interval $[i..j]$, respectively. Here we focus on a specific application of $\mathbf{BWT}_T$: enumerating all the right-maximal substrings of $T$, or equivalently all the internal nodes of $\mathbf{ST}_T$. In particular, we use the algorithm described in [5] (Section 4.1), which we sketch here for completeness.

Given a substring $W$ of $T$, let $b_1 < b_2 < \cdots < b_k$ be the sorted sequence of all the distinct characters in $\Sigma_r(W)$, and let $a_1, a_2, \ldots, a_k$ be the sequence of all the characters in $\Sigma_r(W)$, not necessarily sorted. Assume that we represent...
a substring $W$ of $T$ as a pair $\text{repr}(W) = (\text{chars}[1..k], \text{first}[1..k+1])$, where $\text{chars}[i] = b_i$, $\text{range}(W_{b_i}) = [\text{first}[i]..\text{first}[i+1]-1]$ for $i \in [1..k]$, and function $\text{range}()$ refers to $\text{BWT}_T$. Note that $\text{range}(W) = [\text{first}[1]..\text{first}[k+1]-1]$, since it coincides with the concatenation of the intervals of the right extensions of $W$ in lexicographic order. If $W$ is not right-maximal, array $\text{chars}$ in $\text{repr}(W)$ has length one. Given a data structure that supports $\text{rangeDistinct}$ queries on $\text{BWT}_T$, and given the $C$ array of $T$, there is an algorithm that converts $\text{repr}(W)$ into the sequence $a_1, \ldots, a_h$ and into the corresponding sequence $\text{repr}(a_1W), \ldots, \text{repr}(a_hW)$, in $O(de)$ time and $O(\sigma^2 \log n)$ bits of space in addition to the input and the output [5], where $d$ is the time taken by the $\text{rangeDistinct}$ operation per element in its output, and $e$ is the number of distinct strings $a_iWb_j$ that occur in the circular version of $T$, where $i \in [1..h]$ and $j \in [1..k]$. We encapsulate this algorithm into a function that we call $\text{extendLeft}$.

If $a_iW$ is right-maximal, i.e. if array $\text{chars}$ in $\text{repr}(a_iW)$ has length greater than one, we push pair $(\text{repr}(a_iW), |W|+1)$ onto a stack $S$. In the next iteration we pop the representation of a string from the stack and we repeat the process, until the stack becomes empty. This process is equivalent to following all the explicit Weiner links from the node $v$ of $\text{ST}_T$ with $\ell(v) = W$, not necessarily in lexicographic order. Thus, running the algorithm from a stack initialized with $\text{repr}(\varepsilon)$ is equivalent to performing a depth-first preorder traversal of the suffix-link tree of $T$ (but with an arbitrary exploration order on the children of each node), which guarantees to enumerate all the right-maximal substrings of $T$. Every operation performed by the algorithm can be charged to a distinct node or Weiner link of $\text{ST}_T$, thus the algorithm runs in $O(nd)$ time. We keep the depth of the stack to $O(\log n)$ rather than to $O(n)$ by using the folklore trick of pushing at every iteration the pair $(\text{repr}(a_iW), |a_iW|)$ with largest $\text{range}(a_iW)$ first (see e.g. [13]). Every suffix-link tree level in the stack contains at most $\sigma$ pairs, and each pair takes at most $\sigma \log n$ bits of space, thus the total space used by the stack is $O(\sigma^2 \log^2 n)$ bits. The following theorem follows from our assumption that $\sigma \in o(\sqrt{n}/\log n)$:

**Theorem 1 ([5]).** Let $T \in [1..\sigma]^{n-1}\#$ be a string. Given a data structure that supports $\text{rangeDistinct}$ queries on $\text{BWT}_T$, we can enumerate all the right-maximal substrings $W$ of $T$, and for each of them we can return $|W|$, $\text{repr}(W)$, the sequence $a_1, a_2, \ldots, a_h$ of all characters in $\Sigma_T^\ell(W)$ (not necessarily sorted), and the sequence $\text{repr}(a_1W), \ldots, \text{repr}(a_hW)$, in $O(nd)$ time and in $O(\sigma^2 \log^2 n)$ bits of space in addition to the input and the output, where $d$ is the time taken by the $\text{rangeDistinct}$ operation per element in its output.

Theorem 1 does not specify the order in which the right-maximal substrings must be enumerated, nor the order in which the left extensions of a right-maximal substring must be returned. The algorithm we just described can be adapted to return all the maximal repeats of $T$, within the same bounds, by outputting a right-maximal string $W$ iff $|\text{rangeDistinct}(\text{sp}(W), \text{ep}(W))| > 1$. Computing the minimal rare words that occur in $T$ is also easy:
Lemma 1. Let $T \in [1..\sigma]^{n-1}$ be a string. Given a data structure that supports rangeDistinct queries on $BWT_T$, we can enumerate all the minimal rare words $W$ of $T$ that occur at least once in $T$, and for each of them we can return $|W|$ and range$(W)$, in $O(nd)$ time and in $O(\sigma^2 \log^2 n) = o(n)$ bits of space in addition to the input and the output, where $d$ is the time taken by the rangeDistinct operation per element in its output.

Proof. We use a technique similar to the one described in [6]. Specifically, we adapt Theorem 1 to iterate over all maximal repeats of $T$, and we allocate a temporary array $freq[0..\sigma]$, indexed by all characters in the alphabet. After having enumerated a maximal repeat $W$, we scan $\text{repr}(W)$, we compute the number of occurrences of every right extension $Wb$ of $W$ using array $\text{first}$, and we write $f_T(Wb)$ in $freq[b]$. Then, for every $i \in [1..h]$, we check whether $\text{repr}(a_iW)$ contains more than one character: if this is the case, then $f_T(a_iW) > f_T(a_iWb)$ for every $b \in [0..\sigma]$. Thus, we scan $\text{repr}(a_iW)$ and for every $b$ in its array $\text{chars}$ we check whether $f_T(a_iWb) < f_T(Wb)$, by accessing $freq[b]$. If this is the case, then $a_iWb$ is a minimal rare word, and its interval in $BWT_T$ can be derived in constant time from the array $\text{first}$ of $\text{repr}(a_iW)$. At the end of this process, we reset array $freq$ to its initial state by scanning $\text{repr}(W)$ again. □

Minimal rare words that do not occur in $T$ can be enumerated using a slight variation of Lemma 1 as described in [6].

Lemma 2 ([6]). Let $T \in [1..\sigma]^{n-1}$ be a string. Given a data structure that supports rangeDistinct queries on $BWT_T$, we can enumerate all the minimal rare words $aWb$ of $T$ that do not occur in $T$, where $a$ and $b$ are characters and $W \in [1..\sigma]^*$, and for each of them we can return $a$, $b$, $|W|$, and range$(W)$, in $O(nd + \text{occ})$ time and in $O(\sigma^2 \log^2 n) = o(n)$ bits of space in addition to the input and the output, where $d$ is the time taken by the rangeDistinct operation per element in its output, and $\text{occ}$ is the output size.

For reasons of space, we assume throughout the paper that $d$ is the time per element in the output of a rangeDistinct data structure that is implicit from the context.

3 Computing the border of all right-maximal substrings

As mentioned in the introduction, computing the exact variance of the number of occurrences of a string $W$ in a random text of length $n$ can be mapped to the computation of the longest border of all suffixes of $W$ [3], and thus takes $O(|W|)$ time using the Morris-Pratt algorithm [17]. To compute the longest border of all right-maximal substrings of a text $T$, as well as of all substrings $Wb$ of $T$ such that $b \in [0..\sigma]$ and $W$ is right-maximal, in overall linear time on
[T], we need the following algorithm described in [2], which we sketch here for completeness:

Theorem 2 ([2]). Let $T \in [1..\sigma]^{n}$ be a string. There is an algorithm that computes \emph{bord}(W) for all right-maximal substrings $W$ of $T$, and for all substrings $W = Vb$ of $T$ where $b \in [1..\sigma]$ and $V$ is right-maximal, in $O(n)$ words of space. The running time of this algorithm is linear in $n$ and depends on $\sigma$.

Proof sketch. We build the suffix tree $ST_T$ of $T$, and we assume that every node $v$ of $ST_T$ stores sets $\Sigma_T^l(\ell(v))$ and $\Sigma_T^r(\ell(v))$ as lexicographically sorted lists. We perform a depth-first traversal of the suffix-link tree of $T$, and we store in each node $v$ with label $\ell(v) = W$ the arrays $\text{right}_v$ and $\text{left}_v$ described in Section 2.1, recall that $\text{right}_v$ (respectively, $\text{left}_v$) is indexed by all characters $a \in \Sigma_T(W)$ (respectively, $b \in \Sigma_T(W)$), and it stores value $a|W$ at position $\text{right}_v[a]$ (respectively, value $W|b$ at position $\text{left}_v[b]$). Clearly, if $\text{right}_v[a] > 0$, then $\text{bord}(aW) = \text{right}_v[a] + 1$. If $\text{right}_v[a] = 0$, then $\text{bord}(aW)$ is either one (if $a$ matches the last character of $W$) or zero. Once we know $\text{bord}(aW)$, we compute array $\text{right}_aW$ by exploiting the identity $B(aW) = \{\text{bord}(aW)\} \cup B(V)$, where $V$ is the longest border of $aW$ (see Section 2.1): specifically, for every character $c \in \Sigma^l_T(aW)$, we know that $c$ belongs also to $\Sigma^l_T(V)$, thus we set $\text{right}_aW[c] = \text{right}_V[c]$ if $c \neq d$, and we set $\text{right}_aW[d] = \text{bord}(aW)$, where $d$ is the character that precedes the suffix of $aW$ of length $\text{bord}(aW)$. Since we know that every character $c \in \Sigma_T(aW)$ also belongs to $\Sigma_T(V)$, we can compute array $\text{left}_aW$ from $\text{left}_V$ in the same way.

Every cell of every array $\text{right}_v$ can be charged to a (possibly implicit) Weiner link of $ST_T$, and every cell of every array $\text{left}_v$ can be charged to an edge of $ST_T$, thus the algorithm uses $O(n)$ words of space overall in addition to $ST_T$. However, copying $\text{right}_aW[c]$ from $\text{right}_V[c]$, where $V$ is the longest border of $aW$ (respectively, $\text{left}_aW[c]$ from $\text{left}_V[c]$) requires retrieving $c$ from the list of left extensions (respectively, right extensions) of $V$, or merging such list with the corresponding list of $aW$, which introduces a dependency on $\sigma$. □

A dependency on $\sigma$ can be problematic in data mining applications, where the alphabet could be the result of a dense discretization of a continuous range (see e.g. [15, 16] and references therein). To make Theorem 2 independent of $\sigma$, we start from generalizing the algorithm described in [15] to a trie, counting return arcs that do not point to the root:

Lemma 3. Let $T = (V,E,\sigma)$ be a trie on alphabet $[1..\sigma]$, let $\ell(v) = \ell(e_k) \cdot \ell(e_{k-1}) \cdots \ell(e_1)$ be the label of a node $v \in V$ that is reachable from the root with path $e_1, e_2, \ldots, e_k$, where $e_i \in E$ for all $i \in [1..k]$, and let $a|v = w \in V : \ell(w) = a\ell(v)$. The set of return arcs $E' = \{(v,a|v) : v \in V, a \in [1..\sigma], a|v \neq 0\}$ satisfies $|E'| \leq 2|V|$.

Proof. We say that a return arc $(v,a|v) \in E'$ is of type 1 if there is an edge $v = (v,w) \in E$ with $\ell(e) = a$, and we say that it is of type 2 otherwise. The total number of type-1 return arcs is at most $|V|$, thus we focus on type-2 return arcs. Let $A = \ell(v)$ and let $B = A[1..a|\ell(v)|]$. We charge a type-2 return arc
(v, a|v) to the vertex w that satisfies A = B · ℓ(w). Assume that two distinct type-2 return arcs \((u_1, v_1), (u_2, v_2)\) are charged to the same vertex w. Clearly it must be \(u_1 \neq u_2\). If \(u_1\) and \(u_2\) do not lie on the same path of \(T\), then w must be the lowest common ancestor of \(u_1\) and \(u_2\), or one of its ancestors (excluding the root). Let \(W\) be the label of the path from w to the lowest common ancestor of \(u_1\) and \(u_2\): clearly \(ℓ(u_1) = X_1 · a · W · ℓ(w)\) and \(ℓ(u_2) = X_2 · b · W · ℓ(w)\), where a and b are distinct characters and \(X_1\) and \(X_2\) are strings of the same length, but at the same time it must be \(X_1 · a · W = X_2 · b · W\), a contradiction. Assume thus that \(u_1\) and \(u_2\) lie on the same path of \(T\): without loss of generality, let \(u_1\) be an ancestor of \(u_2\). Further assume that there is an edge \(e = (u_1, v) \in E\) with \(ℓ(e) = a\). Then it must be that \(ℓ(u_1) = Y_1 · ℓ(w) = X_1 · b · Y_1\) and \(ℓ(u_2) = Y_2 · a · Y_1 · ℓ(w) = X_2 · Y_2 · a · Y_1\) and \(a \neq b\), but at the same time it must be that \(a · Y_1 = b · Y_1\), a contradiction. □

□

**Lemma 4.** Let \(T \in [1..σ]^n\) be a string. There is an algorithm that computes bord\((W)\) for all right-maximal substrings \(W\) of \(T\) in \(O(n)\) time and words of space.

**Proof.** We proceed as in Theorem 2 but at every node \(v\) of ST\(_T\) with \(ℓ(v) = W\), we store set \(B^r(W)\), sorted in lexicographic order, rather than right\(_W\). Recall that \(B^r(W)\) is the set of pairs \(\{(a, a|W) : a \in [1..σ], a|W \neq 0\}\), i.e. a representation of the return arcs of Lemma 3. At every node \(v\) we merge \(B^r(W)\) with the lexicographically sorted list of characters in \(Σ_f(T)\), setting bord\((aW)\) = \(a|W\) (which might be zero) for every left extension \(aW\). Once we know bord\((aW)\), we build \(B^r(aW)\) by copying the entire \(B^r(V)\), where \(V\) is the longest border of \(aW\), and by updating or inserting pair \((d, d|aW)\) using a linear scan of \(B^r(V)\), where \(d\) is the character that precedes the suffix of \(aW\) of length bord\((aW)\). This process touches every return arc of Lemma 3 a constant number of times. □

□

Lemma 4 can be clearly applied to any trie \(T\) of size \(n\) on an alphabet of size \(σ\), in which every node stores its children in lexicographic order: the space used by such algorithm in addition to the trie is \(O(\min\{n, λσ\})\), where \(λ\) is the length of a longest path in \(T\). However, Lemma 3 does not generalize to radix trees, thus we cannot use it to bound the construction time of left arrays in Theorem 2. To achieve this, it suffices to replace left arrays with suitable stacks:

**Lemma 5.** Let \(T \in [1..σ]^n\) be a string. There is an algorithm that computes bord\((W)\) for all right-maximal substrings \(W\) of \(T\), and for all substrings \(W = Vb\) of \(T\) where \(b \in [0..σ]\) and \(V\) is right-maximal, in \(O(n)\) time and words of space.

**Proof.** We run the algorithm in Lemma 4 keeping \(σ\) stacks \(S_1, S_2, \ldots, S_σ\). Assume that, when we visit the node \(v\) of ST\(_T\) with \(ℓ(v) = W\), the top of stack \(S_b\) for all \(b \in Σ_f(T)\) stores value \(W|b\) (which might be zero). Assume that, in the depth-first traversal of the suffix-link tree of \(T\), we choose to visit the
When we visit the node with \( \ell(w) = aW \) next: then, we iterate over all characters \( b \in \Sigma_T(aW) \), we compute \( aWb \) by accessing position \( \text{bord}(aW) \) of stack \( S_b \), and we push \( aWb \) on \( S_b \). This works since, if \( aW \) can be extended to the right by character \( b \), then every suffix of \( aW \) can be extended to the right with character \( b \) as well. This process takes overall \( O(n) \) time, and the size of all stacks \( S_1, \ldots, S_\sigma \) is \( O(\min\{n, \lambda\sigma\}) \), where \( \lambda \) is the length of a longest repeat of \( T \), since every element in every stack can be charged to an edge of \( ST_T \). □

The information stored by Lemma 5 is enough to compute in \( O(n) \) time and space the longest border of all minimal rare words that occur at least once in \( T \). Adapting Lemma 5 to compute the longest border of all minimal \emph{absent} words of \( T \) is also easy:

**Lemma 6.** Let \( T \in [1..\sigma]^n \) be a string. There is an algorithm that computes \( \text{bord}(W) \) for all minimal absent words \( W \) of \( T \) in \( O(n + \text{occ}) \) time and words of space, where \( \text{occ} \) is the size of the output.

**Proof.** We proceed as in Lemma 5, but when we visit the node \( v \) of \( ST_T \) with \( \ell(v) = aW \) for some \( a \in [1..\sigma] \), we assume that the top of stack \( S_b \) for all \( b \in \Sigma_T(aW) \) stores value \( aWb \) (which might be zero). Assume that, in the depth-first traversal of the suffix-link tree of \( T \), we choose to visit the node \( w \) with \( \ell(w) = caW \) next: then, we iterate over all characters \( b \in \Sigma_T(aW) \), we compute \( caWb \) by accessing position \( \text{bord}(caW) \) of stack \( S_b \), and we push \( caWb \) on \( S_b \). This works since, if \( aW \) can be extended to the right by character \( b \), then the proper suffix of the longest border of \( caW \) can be extended to the right with character \( b \) as well. Every element in every stack can be charged either to an edge of \( ST_T \) or to a minimal absent word of \( T \), thus the size of all stacks is \( O(\min\{n + \text{occ}, \lambda\sigma\}) \), where \( \lambda \) is the length of a longest repeat of \( T \). □

4 Detecting unusual words in small space

As mentioned in the introduction, in order to compute the exact variance of a substring \( W \) of a string \( T \), we just need to compute functions \( \phi(W) \) and \( \gamma(W) \). Specifically, we just need to compute such functions on all right-maximal substrings of \( T \), and on all substrings \( Wa \) of \( T \) such that \( W \) is right-maximal \cite{1}. Since \( \phi(W) \) and \( \gamma(W) \) can be computed from \( \phi(V) \) and \( \gamma(V) \), where \( V \) is the longest border of \( W \), it is easy to see that we can adapt Lemma 5 as follows. When we visit the node \( v \) of \( ST_T \) with \( \ell(v) = W \), we store \( \phi(W) \) and \( \gamma(W) \), and the top of stack \( S_b \) for all \( b \in \Sigma_T(W) \) stores the following values (which might be zero) in addition to \( Wb \):

- \( F_b(\mid W\mid) = D_b(\mid W\mid) \cdot (F_b(Wb) - 2(\mid W\mid - Wb) \cdot G_b(Wb) + \mid T\mid - 2\mid W\mid + Wb) \)
- \( G_b(\mid W\mid) = D_b(\mid W\mid) \cdot (1 + G_b(Wb)) \)


where $D_b(|W|) = \pi(W[b+1..|W|-1]) \cdot P[b]$. Note that such values are computed recursively. We derive $\pi(W[b+1..|W|-1])$ by keeping an additional stack that stores $\pi(V)$ for every suffix $V$ of $W$. The entire process can be implemented using just a data structure that supports rangeDistinct queries on $\text{BWT}_T$, by implementing Lemma 5 on top of the iterator described in Theorem 1.

**Theorem 3.** Let $T \in [1..\sigma]^{n-1}$ be a string. Given a data structure that supports rangeDistinct queries on $\text{BWT}_T$, we can compute $\phi(W)$ and $\gamma(W)$ for all right-maximal substrings $W$ of $T$, and for all substrings $W = Vb$ of $T$ such that $b \in [1..\sigma]$ and $V$ is right-maximal, in $O(nd)$ time and in $o(n + \lambda \sqrt{n})$ bits of space in addition to the input and the output, where $d$ is the time taken by the rangeDistinct operation per element in its output and $\lambda$ is the length of a longest repeat of $T$.

**Proof.** Recall that the iterator of Theorem 1 returns, for every right-maximal substring $W$ of $T$, the set of all its right extensions in lexicographic order, but the set of all its left extensions in arbitrary order. To implement Lemma 5 in this case, we just need a temporary array $\text{buffer}[1..\sigma]$ of $\sigma \log n$ bits that is initialized to all zeros at the beginning of the traversal. When the iterator visits substring $W$, we scan $B^r(W)$ and we set $\text{buffer}[a] = a|W$ for all $(a,a|W) \in B^r(W)$. Then, for every left extension $a_i$ of $W$ provided by the iterator, we compute $\text{bord}(a_iW)$ by accessing $\text{buffer}[a_i]$, and we proceed as in Lemma 3. Once we have finished processing substring $W$, we reset $\text{buffer}$ to its previous state by setting $\text{buffer}[a] = 0$ for all $(a,a|W) \in B^r(W)$. The claimed space complexity comes from Theorem 1 and from our assumption that $\sigma = o(\sqrt{n}/\log n)$. \qed

For statistical reasons only substrings of length $O(\log \sigma n)$ are candidates for being over- or under-represented [1], so the space complexity of Theorem 3 is effectively $o(n)$, every surprising substring can be encoded in a constant number of machine words, and thus it can be printed in constant time. The technique described in Theorem 3 can be used to apply Lemma 4 to tries whose nodes do not store the list of their children in lexicographic order. Moreover, it is easy to adapt Theorem 3 to output all candidate over- and under-represented substrings whose statistical score matches a user-specified criterion, by keeping an additional stack of characters of size $\lambda \log \sigma$ and by exploiting the fact that the iterator in Theorem 1 returns the length of every substring it visits.

Reducing the number of patterns displayed by a data mining algorithm is key for making it useful in practice. According to the monotonicity of the scores described in Section 1, we can limit the search for over-represented (respectively, under-represented) substrings to maximal repeats (respectively, to minimal rare words); this observation was already implicit in [1], and Theorem 3 can be easily

---

2We assume the word RAM model of computation with words of size $\Omega(\log n)$ bits, in which all standard operations including multiplication have unit cost.
adapted to consider only such candidates. Moreover, in a practical implementation we can compute and store $\phi(W)$ and $\gamma(W)$ just for maximal repeats, since the longest border of a maximal repeat is itself a maximal repeat. We can also avoid storing numbers $F_b(|W|)$, $G_b(|W|)$ and $W^b$ for all $b \in \Sigma_T^r(W)$ on the stacks of Lemma 5 whenever $W$ is a right-maximal substring of $T$ that cannot be written as $aV$ for a character $a$ and a maximal repeat $V$. Within the working space budget of Theorem 3 but in time $O(nd + \omega c)$, we can also compute the border and the statistical scores of all minimal absent words of $T$, by adapting Lemma 6 to work on Theorem 1: such strings are the only strings that do not occur in $T$ which could be under-represented in $T$, however they were not reported in previous works [1, 2]. The ability to assign a statistical score to minimal absent words could be useful also in other contexts, for example in choosing which minimal absent words should be displayed to the user, since their total number is $\Theta(n\sigma)$ in the worst case.

5 Implementation and experiments

The algorithms described in this paper are practical: a prototype implementation that scores all maximal repeats and minimal rare words of a DNA string of length $1.48 \cdot 10^6$ uses on average just $12 \cdot 10^3$ bits for the stack (which can fit in the L1 cache of current processors), with a few sudden bursts that reach up to approximately $6 \cdot 10^5$ bits (which can fit in the L2 cache of current processors): see Figure 1 (left). Since borders are short on average (constant for a random string), border information takes a negligible fraction of the stack, which is approximately equally divided between BWT intervals and quantities related to the variance. Since most borders are short, most recursive accesses of the algorithm are directed to the bottom of the stack (Figure 1, right), so the hardware can automatically move portions of this region to the L1 cache. Moreover, if the user knows that the length of the longest border of every substring of the string under analysis is upper-bounded by a constant $\beta$, the stack can be made even smaller by pushing border and variance information just for string of length at most $\beta$.

Our prototype implementation compares favourably to the previous implementation of Theorem 2 described in [4]: given the BWT represented as a wavelet tree, our implementation scores all maximal repeats and minimal rare substrings in approximately 33 seconds on a 2.50 GHz Intel Xeon E5-2640, while the previous implementation takes approximately 57 seconds and a peak of 6 gigabytes to build a truncated suffix tree and to score only candidate substrings of length at most 12, and 2 (respectively, 4) minutes and a peak of 14 gigabytes to build a truncated suffix tree and to score only candidate substrings of length at most 24 (respectively, 36). Being essentially a tree traversal, our algorithm is intrinsically parallel: we are in the process of developing a shared-memory, multithreaded implementation, and of testing the speedup induced by using more than one core.
Figure 1: A prototype implementation that scores all maximal repeats and minimal rare words of the genome of *Sorangium cellulosum* (*NC_021658* in NCBI). Left panel: number of bits taken by the stack in an entire run, sampled every two second. Top insert: detail of a typical subinterval without spikes. Bottom insert: percent composition of the stack in an entire run. Light gray: information to traverse the suffix-link tree using the BWT. White: border information. Black: variance information. Right panel: number of recursive read requests per string length. Thin line: *NC_021658*, thick line: a random reshuffle of *NC_021658*. The insert details the interval of lengths [1..21]. The peak around length 12 is caused by the retrievals of $\delta(W)$ generated by substrings of expected length.

6 Acknowledgements

We thank Alberto Apostolico for illuminating the details of the algorithms in [2], and Stefano Lonardi for providing the source code of the implementation described in [4].

References

[1] Alberto Apostolico, Mary Ellen Bock, and Stefano Lonardi. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10(3-4):283–311, 2003.

[2] Alberto Apostolico, Mary Ellen Bock, Stefano Lonardi, and Xuyan Xu. Efficient detection of unusual words. *Journal of Computational Biology*, 7(1-2):71–94, 2000.
[3] Alberto Apostolico, Mary Ellen Bock, and Xuyan Xu. Annotated statistical indices for sequence analysis. In Proceedings of Compression and Complexity of Sequences 1997, pages 215–229. IEEE, 1998.

[4] Alberto Apostolico, Fang-Cheng Gong, and Stefano Lonardi. Verbumculus and the discovery of unusual words. Journal of Computer Science and Technology, 19(1):22–41, 2004.

[5] Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In Proceedings of the 46th Annual ACM Symposium on Theory of Computing, STOC ’14, pages 148–193, New York, NY, USA, 2014. ACM.

[6] Djamal Belazzougui and Fabio Cunial. A framework for space-efficient string kernels. In Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings, pages 13–25. Springer International Publishing, 2015.

[7] Djamal Belazzougui, Gonzalo Navarro, and Daniel Valenzuela. Improved compressed indexes for full-text document retrieval. Journal of Discrete Algorithms, 18:3–13, January 2013.

[8] Supaporn Chairungsee and Maxime Crochemore. Using minimal absent words to build phylogeny. Theoretical Computer Science, 450:109–116, 2012.

[9] Maxime Crochemore, Filippo Mignosi, and Antonio Restivo. Automata and forbidden words. Information Processing Letters, 67(3):111–117, 1998.

[10] Maxime Crochemore and Wojciech Rytter. Jewels of stringology. World Scientific, 2002.

[11] Simon Gog. Compressed suffix trees: design, construction, and applications. PhD thesis, University of Ulm, Germany, 2011.

[12] Julia Herold, Stefan Kurtz, and Robert Giegerich. Efficient computation of absent words in genomic sequences. BMC Bioinformatics, 9(1):167, 2008.

[13] Charles AR Hoare. Quicksort. The Computer Journal, 5(1):10–16, 1962.

[14] Atalay Mert Ileri, M. Oguzhan Küleki, and Bojian Xu. A simple yet time-optimal and linear-space algorithm for shortest unique substring queries. Theoretical Computer Science, 562:621–633, 2015.

[15] Eamonn Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. Finding surprising patterns in a time series database in linear time and space. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’02, pages 550–556, New York, NY, USA, 2002. ACM.
[16] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[17] James H. Morris and Vaughan R. Pratt. A linear pattern-matching algorithm. Technical Report 40, University of California, Berkeley, 1970.

[18] Imre Simon. String matching algorithms and automata. In *First South American Workshop on String Processing, Belo Horizonte, Brazil*, pages 151–157, 1993.