Optimal Computation of Avoided Words

Yannis Almirantis[1], Panagiotis Charalampopoulos[2], Jia Gao[2], Costas S. Iliopoulos[2], Manal Mohamed[2], Solon P. Pissis[2(✉)], and Dimitris Polychronopoulos[3]

[1] National Center for Scientific Research Demokritos, Athens, Greece
yalmir@bio.demokritos.gr
[2] Department of Informatics, King's College London, London, UK
{panagiotis.charalampopoulos,jia.gao,costas.iliopoulos,
manal.mohamed,solon.pissis}@kcl.ac.uk
[3] Computational Regulatory Genomics, MRC Clinical Sciences Centre,
Imperial College London, London W12 0NN, UK
d.polychronopoulos@csc.mrc.ac.uk

Abstract. The deviation of the observed frequency of a word w from its expected frequency in a given sequence x is used to determine whether or not the word is *avoided*. This concept is particularly useful in DNA linguistic analysis. The value of the standard deviation of w, denoted by $std(w)$, effectively characterises the extent of a word by its edge contrast in the context in which it occurs. A word w of length $k > 2$ is a ρ-avoided word in x if $std(w) \leq \rho$, for a given threshold $\rho < 0$. Notice that such a word may be completely *absent* from x. Hence computing all such words naïvely can be a very time-consuming procedure, in particular for large k. In this article, we propose an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm to compute all ρ-avoided words of length k in a given sequence x of length n over a fixed-sized alphabet. We also present a time-optimal $\mathcal{O}(\sigma n)$-time algorithm to compute all ρ-avoided words (of any length) in a sequence of length n over an integer alphabet of size σ. We provide a tight asymptotic upper bound for the number of ρ-avoided words over an integer alphabet and the expected length of the longest one. We make available an implementation of our algorithm. Experimental results, using both real and synthetic data, show the efficiency of our implementation.

1 Introduction

The one-to-one mapping of a DNA molecule to a sequence of letters suggests that DNA analysis can be modelled within the framework of formal language theory [13]. For example, a region within a DNA sequence can be considered as a "word" on a fixed-sized alphabet in which some of its natural aspects can be described by means of certain types of automata or grammars. However, a linguistic analysis of the DNA needs to take into account many distinctive physical and biological characteristics of such sequences: DNA contains coding regions

This research was partially supported by the Leverhulme Trust.

M. Frith and C.N.S. Pedersen (Eds.): WABI 2016, LNBI 9838, pp. 1–13, 2016.
DOI: 10.1007/978-3-319-43681-4_1

that encode for polypeptide chains associated with biological functions; and non-coding regions, most of which are not linked to any particular function. Both appear to have many statistical features in common with natural languages [10].

A computational tool oriented towards the systematic search for avoided words is particularly useful for *in silico* genomic research analyses. The search for *absent words* is already undertaken in the recent past and several results exist [1]. However, words which may be present in a genome or in genomic sequences of a specific role (e.g., protein coding segments, regulatory elements, conserved non-coding elements etc.) but they are strongly underrepresented—as we can estimate on the basis of the frequency of occurrence of their longest proper factors—may be of particular importance. They can be words of nucleotides which are hardly tolerated because they negatively influence the stability of the chromatin or, more generally, the functional genomic conformation; they can represent targets of restriction endonucleases which may be found in bacterial and viral genomes; or, more generally, they may be short genomic regions whose presence in wide parts of the genome are not tolerated for less known reasons. The understanding of such avoidances is becoming an interesting line of research (for recent studies, see [5,12]).

On the other hand, short words of nucleotides may be systematically avoided in large genomic regions or whole genomes for entirely different reasons: just because they play important signaling roles which restrict their appearance only in specific positions: consensus sequences for the initiation of gene transcription and of DNA replication are well-known such oligonucleotides. Other such cases may be insulators, sequences anchoring the chromatin on the nuclear envelope like lamina-associated domains, short sequences like dinucleotide repeat motifs with enhancer activity, and several other cases. Again, we cannot exclude that this area of research could lead to the identification of short sequences of regulatory activities still unknown.

Brendel et al. in [6] initiated research into the linguistics of nucleotide sequences that focuses on the concept of words in continuous languages—languages devoid of blanks—and introduced an operational definition of words. The authors suggested a method to measure, for each possible word w of length k, the deviation of its observed frequency from the expected frequency in a given sequence. The values of the standard deviation, denoted by $std(w)$, were then used to identify words that are avoided among all possible words of length k. The typical length of avoided (or of overabundant) words of the nucleotide language was found to range from 3 to 5 (tri- to pentamers). The statistical significance of the avoided words was shown to reflect their biological importance. This work, however, was based on the very limited sequence data available at the time: only DNA sequences from two viral and one bacterial genomes were considered. Also note that k might change when considering eukaryotic genomes, the complex dynamics and function of which might impose a more demanding analysis.

Our Contribution. The computational problem can be described as follows. Given a sequence x of length n, an integer k, and a real number $\rho < 0$, compute the set of ρ-avoided words of length k, i.e. all words w of length k for which

$std(w) \leq \rho$. We call this set the *ρ-avoided words* of length k in x. Brendel et al. did not provide an efficient solution for this computation [6]. Notice that such a word may be completely absent from x. Hence the set of ρ-avoided words can be naïvely computed by considering all possible σ^k words, where σ is the size of the alphabet. Here we present an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing all ρ-avoided words of length k in a sequence x of length n over a fixed-sized alphabet. We also present a time-optimal $\mathcal{O}(\sigma n)$-time algorithm to compute all ρ-avoided words (of any length) in a sequence of length n over an integer alphabet of size σ. We provide a tight asymptotic upper bound for the number of ρ-avoided words over an integer alphabet and the expected length of the longest one. We make available an open-source implementation of our algorithm. Experimental results, using both real and synthetic data, show its efficiency and applicability. Specifically, using our method we confirm that restriction endonucleases which target self-complementary sites are not found in eukaryotic sequences [12].

2 Terminology and Technical Background

Definitions and Notation. We begin with basic definitions and notation generally following [7]. Let $x = x[0]x[1]\dots x[n-1]$ be a *word* of *length* $n = |x|$ over a finite ordered *alphabet* Σ of fixed size, i.e. $\sigma = |\Sigma| = \mathcal{O}(1)$. We also consider the case of an *integer alphabet*; in this case each letter is replaced by its rank such that the resulting string consists of integers in the range $\{1, \dots, n\}$. For two positions i and j on x, we denote by $x[i\,..\,j] = x[i]\,..\,x[j]$ the *factor* (sometimes called *subword*) of x that starts at position i and ends at position j (it is empty if $j < i$), and by ε the *empty word*, word of length 0. We recall that a prefix of x is a factor that starts at position 0 ($x[0\,..\,j]$) and a suffix is a factor that ends at position $n-1$ ($x[i\,..\,n-1]$), and that a factor of x is a *proper* factor if it is not x itself. A factor of x that is neither a prefix nor a suffix of x is called an *infix* of x.

Let $w = w[0]w[1]\,..\,w[m-1]$ be a word, $0 < m \leq n$. We say that there exists an *occurrence* of w in x, or, more simply, that w *occurs in* x, when w is a factor of x. Every occurrence of w can be characterised by a starting position in x. Thus we say that w occurs at the *starting position* i in x when $w = x[i\,..\,i+m-1]$. Further let $f(w)$ denote the *observed frequency*, that is, the number of occurrences of a non-empty word w in word x. If $f(w) = 0$ for some word w, then w is called *absent*, otherwise, w is called *occurring*.

By $f(w_p)$, $f(w_s)$, and $f(w_i)$ we denote the observed frequency of the longest proper prefix w_p, suffix w_s, and infix w_i of w in x, respectively. We can now define the *expected frequency* of word w, $|w| > 2$, in x as in Brendel et al. [6]:

$$E(w) = \frac{f(w_p) \times f(w_s)}{f(w_i)}, \text{ if } f(w_i) > 0; \text{ else } E(w) = 0. \tag{1}$$

The above definition can be explained intuitively as follows. Suppose we are given $f(w_p)$, $f(w_s)$, and $f(w_i)$. Given an occurrence of w_i in x, the probability of it being preceded by $w[0]$ is $\frac{f(w_p)}{f(w_i)}$ as $w[0]$ precedes exactly $f(w_p)$ of the $f(w_i)$

occurrences of w_i. Similarly, this occurrence of w_i is also an occurrence of w_s with probability $\frac{f(w_s)}{f(w_i)}$. Although these two events are not always independent, the product $\frac{f(w_p)}{f(w_i)} \times \frac{f(w_s)}{f(w_i)}$ gives a good approximation of the probability that an occurrence of w_i at position j implies an occurrence of w at position $j-1$. It can be seen then that by multiplying this product by the number of occurrences of w_i we get the above formula for the expected frequency of w.
Moreover, to measure the deviation of the observed frequency of a word w from its expected frequency in x, we define the *standard deviation* (χ^2 test) of w as:

$$std(w) = \frac{f(w) - E(w)}{\max\{\sqrt{E(w)}, 1\}}. \tag{2}$$

For more details on the *biological* justification of these definitions see [6].

Using the above definitions and a given threshold, we are in a position to classify a word w as either *avoided* or *common* in x. In particular, for a given threshold $\rho < 0$, a word w is called ρ-*avoided* if $std(w) \leq \rho$. In this article, we consider the following computational problem.

AvoidedWordsComputation
Input: A word x of length n, an integer $k > 2$, and a real number $\rho < 0$
Output: All ρ-avoided words of length k in x

Suffix Trees. In our algorithms, suffix trees are used extensively as computational tools. For a general introduction to suffix trees, see [7].

The *suffix tree* $\mathcal{T}(x)$ of a non-empty word x of length n is a compact trie representing all suffixes of x, the nodes of the trie which become nodes of the suffix tree are called *explicit* nodes, while the other nodes are called *implicit*. Each edge of the suffix tree can be viewed as an upward maximal path of implicit nodes starting with an explicit node. Moreover, each node belongs to a unique path of that kind. Then, each node of the trie can be represented in the suffix tree by the edge it belongs to and an index within the corresponding path.

We use $\mathcal{L}(v)$ to denote the *path-label* of a node v, i.e., the concatenation of the edge labels along the path from the root to v. We say that v is path-labelled $\mathcal{L}(v)$. Additionally, $\mathcal{D}(v) = |\mathcal{L}(v)|$ is used to denote the *word-depth* of node v. Node v is a *terminal* node, if and only if, $\mathcal{L}(v) = x[i \,..\, n-1]$, $0 \leq i < n$; here v is also labelled with index i. It should be clear that each occurring word w in x is uniquely represented by either an explicit or an implicit node of $\mathcal{T}(x)$. The *suffix-link* of a node v with path-label $\mathcal{L}(v) = \alpha y$ is a pointer to the node path-labelled y, where $\alpha \in \Sigma$ is a single letter and y is a word. The suffix-link of v exists if v is a non-root internal node of $\mathcal{T}(x)$.

In any standard implementation of the suffix tree, we assume that each node of the suffix tree is able to access its parent. Note that once $\mathcal{T}(x)$ is constructed, it can be traversed in a depth-first manner to compute the word-depth $\mathcal{D}(v)$ for each node v. Let u be the parent of v. Then the word-depth $\mathcal{D}(v)$ is computed by adding $\mathcal{D}(u)$ to the length of the label of edge (u, v). If v is the root then

$\mathcal{D}(v) = 0$. Additionally, a depth-first traversal of $\mathcal{T}(x)$ allows us to count, for each node v, the number of terminal nodes in the subtree rooted at v, denoted by $\mathcal{C}(v)$, as follows. When internal node v is visited, $\mathcal{C}(v)$ is computed by adding up $\mathcal{C}(u)$ of all the nodes u, such that u is a child of v, and then $\mathcal{C}(v)$ is incremented by 1 if v itself is a terminal node. If a node v is a leaf then $\mathcal{C}(v) = 1$.

3 Useful Properties

In this section, we provide some useful insights of combinatorial nature which were not considered by Brendel et al. [6]. By the definition of ρ-avoided words it follows that a word w may be ρ-avoided even if it is absent from x. In other words, $std(w) \leq \rho$ may hold for either $f(w) > 0$ (occurring) or $f(w) = 0$ (absent).

This means that a naïve computation should consider *all* possible σ^k words. Then for each possible word w, the value of $std(w)$ can be computed via pattern matching on the suffix tree of x. In particular we can search for the occurrences of w, w_p, w_s, and w_i in x in time $\mathcal{O}(k)$ [7]. In order to avoid this inefficient computation, we exploit the following crucial lemmas.

Definition 1 [3]. *An absent word w of x is* minimal *if and only if all its proper factors occur in x.*

Lemma 1. *Any absent ρ-avoided word w in x is a minimal absent word of x.*

Proof. For w to be a ρ-avoided word it must hold that

$$std(w) = \frac{f(w) - E(w)}{\max\{\sqrt{E(w)}, 1\}} \leq \rho < 0.$$

This implies that $f(w) - E(w) < 0$, which in turn implies that $E(w) > 0$ since $f(w) = 0$. From $E(w) = \frac{f(w_p) \times f(w_s)}{f(w_i)} > 0$, we conclude that $f(w_p) > 0$ and $f(w_s) > 0$ must hold. Since $f(w) = 0$, $f(w_p) > 0$, and $f(w_s) > 0$, w is a minimal absent word of x: all proper factors of w occur in x. □

Lemma 2. *Let w be a word occurring in x and $\mathcal{T}(x)$ be the suffix tree of x. Then, if w_p is a path-label of an implicit node of $\mathcal{T}(x)$, $std(w) \geq 0$.*

Proof. For any w that occurs in x it holds that $f(w_i) \geq f(w_s)$, which implies that $f(w_p) \geq \frac{f(w_p) \times f(w_s)}{f(w_i)} = E(w)$. Furthermore, by the definition of the suffix tree, if w occurs in x and w_p is a path-label of an implicit node then $f(w_p) = f(w)$. It thus follows that $f(w) - E(w) = f(w_p) - E(w) \geq 0$, and since $\max\{1, \sqrt{E(w)}\} > 0$, the claim holds. □

Lemma 3. *The number of ρ-avoided words of length $k > 2$ in a word of length n over an alphabet of size σ is $\mathcal{O}(\sigma n)$; in particular, this number is no more than $(\sigma + 1)n - k + 1$.*

Proof. By Lemma 1, every ρ-avoided word is either occurring or a minimal absent word. It is known that the number of minimal absent words in a word of length n is smaller than or equal to σn [11]. Clearly, the occurring ρ-avoided words in a word of length n are at most $n - k + 1$. Therefore the lemma holds. □

4 Avoided Words Algorithm

In this section, we present Algorithm AvoidedWords for computing all ρ-avoided words of length k in a given word x. The algorithm builds the suffix tree $\mathcal{T}(x)$ for word x, and then prepares $\mathcal{T}(x)$ to allow constant-time observed frequency queries. This is mainly achieved by counting the terminal nodes in the subtree rooted at node v for every node v of $\mathcal{T}(x)$. Additionally during this preprocessing, the algorithm computes the word-depth of v for every node v of $\mathcal{T}(x)$. By Lemma 1, ρ-avoided words are classified as either occurring or (minimal) absent, therefore Algorithm AvoidedWords calls Routines AbsentAvoidedWords and OccurringAvoidedWords to compute both classes of ρ-avoided words in x. The outline of Algorithm AvoidedWords is as follows.

AvoidedWords(x, k, ρ)
1 $\mathcal{T}(x) \leftarrow$ SuffixTree(x)
2 **for** each node $v \in \mathcal{T}(x)$ **do**
3 $\quad\mathcal{D}(v) \leftarrow$ word-depth of v
4 $\quad\mathcal{C}(v) \leftarrow$ number of terminal nodes in the subtree rooted at v
5 AbsentAvoidedWords(x, k, ρ)
6 OccurringAvoidedWords(x, k, ρ)

4.1 Computing Absent Avoided Words

In Lemma 1, we showed that each absent ρ-avoided word is a minimal absent word. Thus, Routine AbsentAvoidedWords starts by computing all minimal absent words in x; this can be done in time and space $\mathcal{O}(n)$ for a fixed-sized alphabet or in time $\mathcal{O}(\sigma n)$ for integer alphabets [3,4]. Let $<(i, j), \alpha>$ be a tuple representing a minimal absent word in x, where for some minimal absent word w of length $|w| > 2$, $w = x[i\,..\,j]\alpha$, $\alpha \in \Sigma$; this representation is clearly unique.

AbsentAvoidedWords(x, k, ρ)
1 $\mathcal{A} \leftarrow$ MinimalAbsentWords(x)
2 **for** each tuple $< (i, j), \alpha > \in \mathcal{A}$ such that $k = j - i + 2$ **do**
3 $\quad u_p \leftarrow$ Node(i, j)
4 $\quad$ **if** IsImplicit(u_p) **then**
5 $\quad\quad(u, v) \leftarrow$ Edge(u_p)
6 $\quad\quad f_p \leftarrow \mathcal{C}(v)$
7 $\quad$ **else** $f_p \leftarrow \mathcal{C}(u_p)$
8 $\quad u_i \leftarrow$ Node$(i + 1, j)$
9 $\quad$ **if** IsImplicit(u_i) **then**
10 $\quad\quad(u, v) \leftarrow$ Edge(u_i)
11 $\quad\quad f_i \leftarrow f_s \leftarrow \mathcal{C}(v)$
12 $\quad$ **else** $f_i \leftarrow \mathcal{C}(u_i)$
13 $\quad\quad u_s \leftarrow$ Child(u_i, α)
14 $\quad\quad f_s \leftarrow \mathcal{C}(u_s)$
15 $\quad E \leftarrow f_p \times f_s / f_i$
16 $\quad$ **if** $(0 - E)/(\max\{1, \sqrt{E}\}) \leq \rho$ **then**
17 $\quad\quad$ Report$(x[i\,..\,j]\alpha)$

Intuitively, the idea is to check the length of every minimal absent word. If a tuple $<(i,j),\alpha>$ represents a minimal absent word w of length $k = j-i+2$, then the value of $std(w)$ is computed to determine whether w is an absent ρ-avoided word. Note that, if $w = x[i\,..\,j]\alpha$ is a minimal absent word, then $w_p = x[i\,..\,j]$, $w_i = x[i+1\,..\,j]$, and $w_s = x[i+1\,..\,j]\alpha$ occur in x by Definition 1. Thus, there are three (implicit or explicit) nodes in $\mathcal{T}(x)$ path-labelled w_p, w_i, and w_s, respectively. The observed frequencies of w_p, w_i, and w_s are already computed during the preprocessing of $\mathcal{C}$, which stores the number of terminal nodes in the subtree rooted at v, for each node v.

Notice that for an explicit node v path-labelled $w' = x[i'\,..\,j']$, the value $\mathcal{C}(v)$ represents the number of occurrences (observed frequency) of w' in x; whereas for an implicit node along the edge (u,v) path-labelled w'', the number of occurrences of w'' is equal to $\mathcal{C}(v)$ (and not $\mathcal{C}(u)$). The implementation of this procedure is given in Routine AbsentAvoidedWords.

4.2 Computing Occurring Avoided Words

Lemma 2 suggests that for each occurring ρ-avoided word w, w_p is a path-label of an explicit node v of $\mathcal{T}(x)$. Thus, for each internal node v such that $\mathcal{D}(v) = k-1$ and $\mathcal{L}(v) = w_p$, Routine OccurringAvoidedWords computes $std(w)$, where $w = w_p\alpha$, $\alpha \in \Sigma$, is a path-label of a child (explicit or implicit) node of v. Note that if w_p is a path-label of an explicit node v then w_i is a path-label of an explicit node u of $\mathcal{T}(x)$; node u is well-defined and it is the node pointed at by the suffix-link of v. The implementation of this procedure is given in Routine OccurringAvoidedWords.

OccurringAvoidedWords(x, k, ρ)

```
1  N ← an empty stack
2  Push(N, root(T(x)))
3  while N is not empty do
4      u ← Pop(N)
5      for each edge (u, v) of T(x) do
6          if D(v) < k − 1 then
7              Push(N, v)
8          elseif D(v) = k − 1 then
9              f_p ← C(v)
10             f_i ← C(suffix-link[v])
11             for each child v' of v do
12                 f_w ← C(v')
13                 α ← L(v')[k − 1]
14                 f_s ← C(Child(suffix-link[v], α))
15                 E ← f_p × f_s/f_i
16                 if (f_w − E)/(max{1, √E}) ≤ ρ then
17                     Report(L(v')[0 .. k − 1])
```

4.3 Analysis of the Algorithm

Lemma 4. *Given a word x, an integer $k > 2$, and a real number $\rho < 0$, Algorithm* AvoidedWords *computes all ρ-avoided words of length k in x.*

Proof. By definition, a ρ-avoided word w is either an absent ρ-avoided word or an occurring one. Hence, the proof of correctness relies on Lemmas 1 and 2. First, Lemma 1 indicates that an absent ρ-avoided word in x is necessarily a minimal absent word. Routine AbsentAvoidedWords considers each minimal absent word w and verifies if w is a ρ-avoided word of length k.

Second, Lemma 2 indicates that for each occurring ρ-avoided word w, w_p is a path-label of an explicit node v of $\mathcal{T}(x)$. Routine OccurringAvoidedWords considers every child of each such node of word-depth k, and verifies if its path-label is a ρ-avoided word. □

Lemma 5. *Given a word x of length n over a fixed-sized alphabet, an integer $k > 2$, and a real number $\rho < 0$, Algorithm* AvoidedWords *requires time and space $\mathcal{O}(n)$; for integer alphabets, it requires time $\mathcal{O}(\sigma n)$.*

Proof. Constructing the suffix tree $\mathcal{T}(x)$ of the input word x takes time and space $\mathcal{O}(n)$ for a word over a fixed-sized alphabet [7]. Once the suffix tree is constructed, computing arrays $\mathcal{D}$ and $\mathcal{C}$ by traversing $\mathcal{T}(x)$ requires time and space $\mathcal{O}(n)$. Note that the path-labels of the nodes of $\mathcal{T}(x)$ can by implemented in time and space $\mathcal{O}(n)$ as follows: traverse the suffix tree to compute for each node v the smallest index i of the terminal nodes of the subtree rooted at v. Then $\mathcal{L}(v) = x[i \,..\, i + \mathcal{D}(v) - 1]$.

Next, Routine AbsentAvoidedWords requires time $\mathcal{O}(n)$. It starts by computing all minimal absent words of x, which can be achieved in time and space $\mathcal{O}(n)$ over a fixed-sized alphabet [3,4]. The rest of the procedure deals with checking each of the $\mathcal{O}(n)$ minimal absent words of length k. Checking each minimal absent word w to determine whether it is a ρ-avoided word or not requires time $\mathcal{O}(1)$. In particular, an $\mathcal{O}(n)$-time preprocessing of $\mathcal{T}(x)$ allows the retrieval of the (implicit or explicit) node in $\mathcal{T}(x)$ corresponding to the longest proper prefix of w in time $\mathcal{O}(1)$ [9]. Finally, Routine OccurringAvoidedWords requires time $\mathcal{O}(n)$. It traverses the suffix tree $\mathcal{T}(x)$ to consider all explicit nodes of word-depth $k - 1$. Then for each such node, the procedure checks every (explicit or implicit) child of word-depth k. The total number of these children is at most $n - k + 1$. For every child node, the procedure checks whether its path-label is a ρ-avoided word in time $\mathcal{O}(1)$ via the use of suffix-links.

For integer alphabets, the suffix tree can be constructed in time $\mathcal{O}(n)$ [8] and all minimal absent words can be computed in time $\mathcal{O}(\sigma n)$ [3,4]. The efficiency of Algorithm AvoidedWords is then limited by the total number of words to be considered, which, by Lemma 3, is $\mathcal{O}(\sigma n)$. □

Lemmas 4 and 5 imply the first result of this article.

Theorem 1. *Algorithm* AvoidedWords *solves Problem* AvoidedWordsComputation *in time and space $\mathcal{O}(n)$. For integer alphabets, the algorithm solves the problem in time $\mathcal{O}(\sigma n)$.*

4.4 Optimal Computation of all ρ-Avoided Words

Although the biological motivation is yet to be shown for this, we present here how we can modify Algorithm AvoidedWords so that it computes *all* ρ-avoided words (of all lengths) in a given word x of length n over an integer alphabet of size σ in time $\mathcal{O}(\sigma n)$. We further show that this algorithm is in fact time-optimal. All omitted proofs will be presented in the full version of this article.

Lemma 6. *The upper bound $\mathcal{O}(\sigma n)$ on the number of minimal absent words of a word of length n over an alphabet of size σ is tight if $2 \leq \sigma \leq n$.*

Lemma 7. *The number of ρ-avoided words in a word of length n over an alphabet of size $2 \leq \sigma \leq n$ is $\mathcal{O}(\sigma n)$ and this bound is tight.*

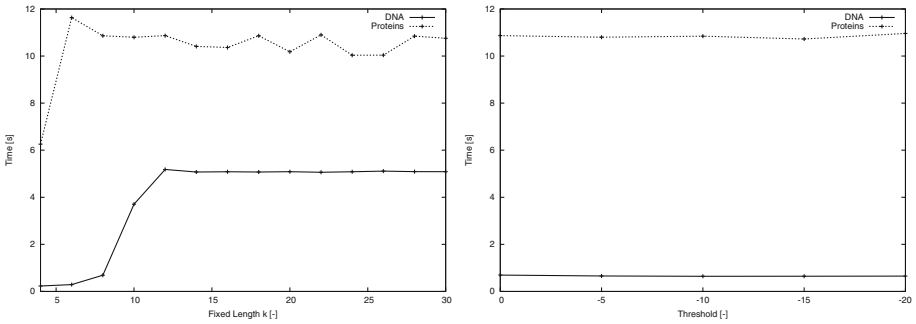
It is clear that if we just remove the condition on the length of each minimal absent word in Line 2 of AbsentAvoidedWords we then compute all absent ρ-avoided words in time $\mathcal{O}(\sigma n)$. In order to compute all occurring ρ-avoided words in x it suffices by Lemma 2 to investigate the children of explicit nodes. We can thus traverse the suffix tree $\mathcal{T}(x)$ and for each explicit internal node, check for all of its children (explicit or implicit) whether their path-label is a ρ-avoided word. We can do this in $\mathcal{O}(1)$ time as described. The total number of these children is at most $2n - 1$, as this is the bound on the number of edges of $\mathcal{T}(x)$ [7]. This modified algorithm is clearly time-optimal for fixed-sized alphabets as it then runs in time $\mathcal{O}(n)$. The time optimality for integer alphabets follows directly from Lemmas 6 and 7. Hence we obtain the second result of this article.

Theorem 2. *Given a word x of length n over an integer alphabet of size σ and a real number $\rho < 0$, all ρ-avoided words in x can be computed in time $\mathcal{O}(\sigma n)$. This is time-optimal if $2 \leq \sigma \leq n$.*

Lemma 8. *The expected length of the longest ρ-avoided word in a word of length n over an alphabet Σ of size $\sigma > 1$ is $\mathcal{O}(\log_\sigma n)$ when the letters are independent and identically distributed random variables uniformly distributed over Σ.*

5 Implementation and Experimental Results

Algorithm AvoidedWords was implemented as a program to compute the ρ-avoided words of length k in one or more input sequences. The program was implemented in the `C++` programming language and developed under GNU/Linux operating system. The input parameters are a (Multi)FASTA file with the input sequences(s), an integer $k > 2$, and a real number $\rho < 0$. The output is a file with the set of ρ-avoided words of length k per input sequence. The implementation is distributed under the GNU General Public License, and it is available at http://github.com/solonas13/aw. The experiments were conducted on a Desktop PC using one core of Intel Core i5-4690 CPU at 3.50 GHz under GNU/Linux. The program was compiled with `g++` version 4.8.4 at optimisation level 3 (-O3). We also implemented a brute-force approach for the computation



(a) Time for $n = 1\text{M}$ and $\rho = -10$ (b) Time for $n = 1\text{M}$ and $k = 8$

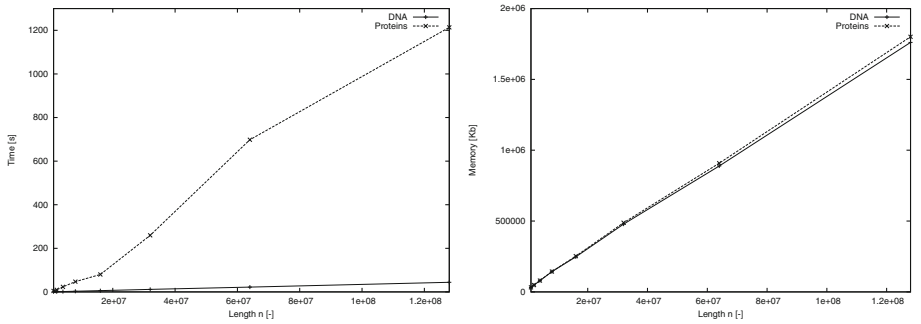
Fig. 1. Elapsed time of Algorithm AvoidedWords using synthetic DNA ($\sigma = 4$) and proteins ($\sigma = 20$) data of length 1M for variable k and variable ρ.

of ρ-avoided words. We mainly used it to confirm the correctness of our implementation. Here we do not plot the results of the brute-force approach as it is easily understood that it is orders of magnitude slower than our approach.

To evaluate the time performance of our implementation, synthetic DNA ($\sigma = 4$) and proteins ($\sigma = 20$) data were used. The input sequences were generated using a randomised script. In the first experiment, our task was to establish that the performance of the program does not essentially depend on k and ρ; i.e., the elapsed time of the program remains unchanged up to some constant with increasing values of k and decreasing values of ρ. As input datasets, for this experiment, we used a DNA and a proteins sequence both of length 1M (1 Million letters). For each sequence we used different values of k and ρ. The results, for elapsed time are plotted in Fig. 1. It becomes evident from the results that the time performance of the program remains unchanged up to some constant. The longer time required for the proteins sequences for some value of k is explained by the increased number of branching nodes in this depth in the corresponding suffix tree due to the size of the alphabet ($\sigma = 20$). To confirm this we counted the number of nodes considered by the algorithm to compute the ρ-avoided words for $k = 4$ and $\rho = -10$ for both sequences. The number of considered nodes for the DNA sequence was 260 whereas for the proteins sequence it was $1,585,510$.

In the second experiment, our task was to establish the fact that the elapsed time and memory usage of the program grow linearly with n, the length of the input sequence. As input datasets, for this experiment, we used synthetic DNA and proteins sequences ranging from 1 to 128 M. For each sequence we used constant values for k and ρ: $k = 8$ and $\rho = -10$. The results, for elapsed time and peak memory usage, are plotted in Fig. 2. It becomes evident from the results that the elapsed time and memory usage of the program grow linearly with n. The longer time required for the proteins sequences compared to the DNA sequences for increasing n is explained by the increased number of branching

nodes in this depth ($k = 8$) in the corresponding suffix tree due to the size of the alphabet ($\sigma = 20$). To confirm this we counted the number of nodes considered by the algorithm to compute the ρ-avoided words for $n = 64$M for both the DNA and the proteins sequence. The number of nodes for the DNA sequence was $69,392$ whereas for the proteins sequence it was $43,423,082$.



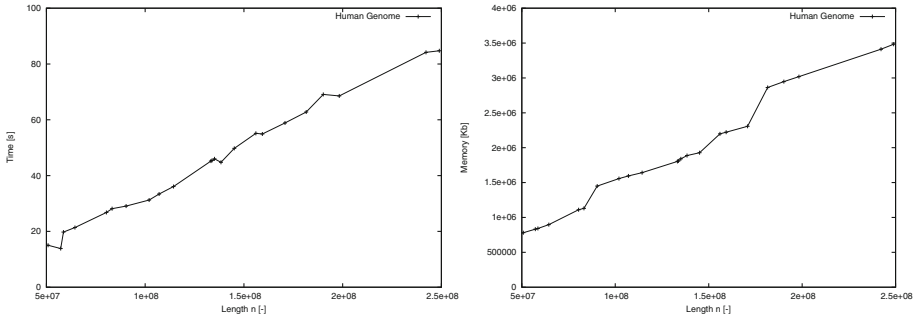
(a) Time for $k = 8$ and $\rho = -10$ (b) Memory for $k = 8$ and $\rho = -10$

Fig. 2. Elapsed time and peak memory usage of Algorithm AvoidedWords using synthetic DNA ($\sigma = 4$) and proteins ($\sigma = 20$) data of length 1M to 128M.

In the next experiment, our task was to evaluate the time and memory performance of our implementation with real data. As input datasets, for this experiment, we used all chromosomes of the human genome. Their lengths range from around 46M (chromosome 21) to around 249M (chromosome 1). For each sequence we used $k = 8$ and $\rho = -10$. The results, for elapsed time and peak memory usage, are plotted in Fig. 3. The results with real data confirm that the elapsed time and memory usage of the program grow linearly with n.

As last experiment, we computed the set of avoided words for $k = 6$ (hexamers) and $\rho = -10$ in the complete genome of *E. coli* and sorted the output in increasing order of their standard deviation. The most avoided words were extremely enriched in self-complementary (palindromic) hexamers. In particular, within the output of 28 avoided words, 23 were self-complementary; and the 17 most avoided ones were *all* self-complementary. For comparison, we computed the set of avoided words for $k = 6$ and $\rho = -10$ from an eukaryotic sequence: a segment of the human chromosome 21 (its leftmost segment devoid of `N`'s) equal to the length of the *E. coli* genome. In the output of 10 avoided words, no self-complementary hexamer was found. Our results confirm that the restriction endonucleases which target self-complementary sites are not found in eukaryotic sequences [12].

Our immediate target is to investigate the avoidance of words in the context of Genomic Regulatory Blocks (GRBs), chromosomal regions spanned by highly conserved non-coding elements (HCNEs), most of which serve as regulatory inputs of one target gene in the region [2].



(a) Time for $k = 8$ and $\rho = -10$ (b) Memory for $k = 8$ and $\rho = -10$

Fig. 3. Elapsed time and peak memory usage of Algorithm AvoidedWords using all chromosomes of the human genome.

References

1. Acquisti, C., Poste, G., Curtiss, D., Kumar, S.: Nullomers: really a matter of natural selection? PLoS ONE **2**(10), e1022 (2007)
2. Akalin, A., Fredman, D., Arner, E., Dong, X., Bryne, J., Suzuki, H., Daub, C., Hayashizaki, Y., Lenhard, B.: Transcriptional features of genomic regulatory blocks. Genome Biol. **10**(4), 1 (2009)
3. Barton, C., Heliou, A., Mouchard, L., Pissis, S.P.: Linear-time computation of minimal absent words using suffix array. BMC Bioinform. **15**(1), 1–10 (2014)
4. Barton, C., Heliou, A., Mouchard, L., Pissis, S.P.: Parallelising the computation of minimal absent words. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) PPAM 2015, Part II. LNCS, vol. 9574, pp. 243–253. Springer, Heidelberg (2016)
5. Belazzougui, D., Cunial, F.: Space-efficient detection of unusual words. In: Iliopoulos, C., Puglisi, S., Yilmaz, E. (eds.) SPIRE 2015. LNCS, vol. 9309, pp. 222–233. Springer, Heidelberg (2015)
6. Brendel, V., Beckmann, J.S., Trifonov, E.N.: Linguistics of nucleotide sequences: morphology and comparison of vocabularies. J. Biomol. Struct. Dyn. **4**(1), 11–21 (1986)
7. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, New York (2007)
8. Farach, M.: Optimal suffix tree construction with large alphabets. In: FOCS, pp. 137–143. IEEE (1997)
9. Gawrychowski, P., Lewenstein, M., Nicholson, P.K.: Weighted ancestors in suffix trees. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 455–466. Springer, Heidelberg (2014)
10. Mantegna, R.N., Buldyrev, S.V., Goldberger, A.L., Havlin, S., Peng, C.K., Simons, M., Stanley, H.E.: Linguistic features of noncoding DNA sequences. Phys. Rev. Lett. **73**, 3169–3172 (1994)

11. Mignosi, F., Restivo, A., Sciortino, M.: Words and forbidden factors. Theoret. Comput. Sci. **273**(1–2), 99–117 (2002)
12. Rusinov, I., Ershova, A., Karyagina, A., Spirin, S., Alexeevski, A.: Lifespan of restriction-modification systems critically affects avoidance of their recognition sites in host genomes. BMC Genom. **16**(1), 1–15 (2015)
13. Searls, D.B.: The linguistics of DNA. Am. Sci. **80**(6), 579–591 (1992)