On the number of elements to reorder when updating a suffix array

Martine Léonard, Laurent Mouchard, Mikaël Salson

On the Number of Elements to Reorder When Updating a Suffix Array

M. Léonard [a] L. Mouchard [a,b] M. Salson [c,*]

[a] *Université de Rouen, Department of Computer Science LITIS EA 4108, 76821 Mont Saint Aignan, France*

[b] *King's College London, Department of Informatics, Algorithm Design Group, Strand, London WC2R 2LS, United Kingdom*

[c] *LIFL (UMR 8022 CNRS – Université Lille 1), INRIA Lille-Nord Europe, France*

Abstract

Recently new algorithms appeared for updating the Burrows-Wheeler transform or the suffix array, when the text they index is modified. These algorithms proceed by reordering entries and the number of such reordered entries may be as high as the length of the text. However, in practice, these algorithms are faster for updating the Burrows-Wheeler transform or the suffix array than the fastest reconstruction algorithms.

In this article we focus on the number of elements to be reordered for real-life texts. We show that this number is related to LCP values and that, on average, L_{ave} entries are reordered, where L_{ave} denotes the average LCP value, defined as the average length of the longest common prefix between two consecutive sorted suffixes. Since we know little about the LCP distribution for real-life texts, we conduct experiments on a corpus that consists of DNA sequences and natural language texts. The results show that apart from texts containing large repetitions, the average LCP value is close to the one expected on a random text.

Key words: FM-index, Burrows-Wheeler Transform, Suffix Array, Longest Common Prefix, Dynamic Data Structure, Self-Index, Complexity

* Corresponding author: Mikael.Salson@lifl.fr

1 Introduction

The Burrows-Wheeler Transform [1,7], BWT, is a very interesting block-sorting algorithm that reorders the letters of a text T of length n over an alphabet of size σ for easing its compression. It is used as a preprocessor by some of the most popular lossless text compression tools (such as bzip) that chain it to Run-Length Encoding, entropy encoding or Prediction by Partial Matching methods [3,2].

Conceptually speaking, the suffix array [17,9] is very close to the text produced by the BWT. Due to its intrinsic structure and its similarity with the suffix array, it has also been used for advanced compressed index structures [5,6], such as FM-index, that authorize approximate pattern matching [14,15], and therefore can be used by search engines.

The FM-index is based on the close relationship between the Burrows-Wheeler Transform and the suffix array. It is therefore a sort of compressed suffix array that takes advantage of the compressibility of the indexed text in order to reduce space occupancy with respect to the entropy of the text.

Recently algorithms that update BWT [21], corresponding enhanced/extended suffix array [8] or FM-index [22] appeared when edit operations transform the text T into a text T' of length n'. At first sight, the overall complexity of such algorithms is bounded by $O(n \log n \log \sigma)$ for [21,22], and $O(n^2)$ for [8] making them slower than known linear-time suffix array construction algorithms [11,13].

Nevertheless, practical experiments conducted in the past (see [8,21,22]) have shown that they globally outperform the quickest known suffix array construction algorithm [20] for a reasonable number of edit operations.

The time consuming part when one wants to update a suffix array-related structure is to reorder suffixes. In the worst case a linear number of suffixes can be reordered. In this article, we show that the average time complexity is strongly connected to the average LCP, the longest common prefix. For various texts, such as repetitive genomic sequences or natural language texts, these values are surprisingly small.

In section 2 we briefly sketch properties of the suffix array and the Burrows-Wheeler Transform, we identify four types of suffixes (or cyclic shifts) and, based on that, characterize the number of elements to reorder on average. In section 3 we present the sets of data that have been used for the tests, explain their relevance. In section 4 we present practical results together with discussions and finally we conclude in section 5.

2 Preliminaries and Algorithms

We consider the text $T = T[0\,..\,n]$, a word of length $n+1$ over Σ, a finite ordered alphabet of size σ. The last letter of T is a sentinel letter \$, that has been added to the alphabet Σ and is smaller than any other letter of Σ. A factor starting at position i and ending at position j is denoted by $T[i\,..\,j]$ and a single letter is denoted by $T[i]$ (or T_i to facilitate the reading). We add that when $i > j$, $T[i\,..\,j]$ is the empty word. A factor starting at position 0 is a prefix of T while a factor ending at position n is a suffix of T.

We briefly recall well-known definitions that will be used throughout the document.

The cyclic shift of order i of T is $T^{[i]} = T[i\,..\,n]T[0\,..\,i-1]$ for $0 \leq i \leq n$. The Burrows-Wheeler Transform of T, denoted $BWT(T)$, is the text of length $n+1$ corresponding to the last column L of the conceptual matrix whose rows are the lexicographically sorted cyclic shifts $T^{[i]}$ (see Fig. 1(b)). The suffix array of T, denoted by $SA(T)$, or simply SA, is the list of starting positions of lexicographically sorted suffixes of T (or cyclic shifts of T). That is $SA[j] = i$ if and only if $T[i\,..\,n]$ is the $(j+1)^{\text{th}}$ suffix of T in ascending order. The inverse of SA, denoted by $ISA(T)$, indicates for each suffix of T its corresponding position in SA. For each $i \in [0, n-1]$, $LCP[i]$ is the length of the Longest Common Prefix between suffixes starting at positions $SA[i]$ and $SA[i+1]$ in the text.

$$T = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}{\text{C T A G T T A G \$}}$$

$T^{[0]}$	C	T	A	G	T	T	A	G	\$
$T^{[1]}$	T	A	G	T	T	A	G	\$	C
$T^{[2]}$	A	G	T	T	A	G	\$	C	T
$T^{[3]}$	G	T	T	A	G	\$	C	T	A
$T^{[4]}$	T	T	A	G	\$	C	T	A	G
$T^{[5]}$	T	A	G	\$	C	T	A	G	T
$T^{[6]}$	A	G	\$	C	T	A	G	T	T
$T^{[7]}$	G	\$	C	T	A	G	T	T	A
$T^{[8]}$	\$	C	T	A	G	T	T	A	G

(a) Cyclic shifts

	F								L
$T^{[8]}$	\$	C	T	A	G	T	T	A	G
$T^{[6]}$	A	G	\$	C	T	A	G	T	T
$T^{[2]}$	A	G	T	T	A	G	\$	C	T
$T^{[0]}$	C	T	A	G	T	T	A	G	\$
$T^{[7]}$	G	\$	C	T	A	G	T	T	A
$T^{[3]}$	G	T	T	A	G	\$	C	T	A
$T^{[5]}$	T	A	G	\$	C	T	A	G	T
$T^{[1]}$	T	A	G	T	T	A	G	\$	C
$T^{[4]}$	T	T	A	G	\$	C	T	A	G

(b) Sorted cyclic shifts

	SA	ISA	LCP
0	8	3	0
1	6	7	2
2	2	2	0
3	0	5	0
4	7	8	1
5	3	6	0
6	5	1	3
7	1	4	1
8	4	0	

(c) SA, ISA, LCP

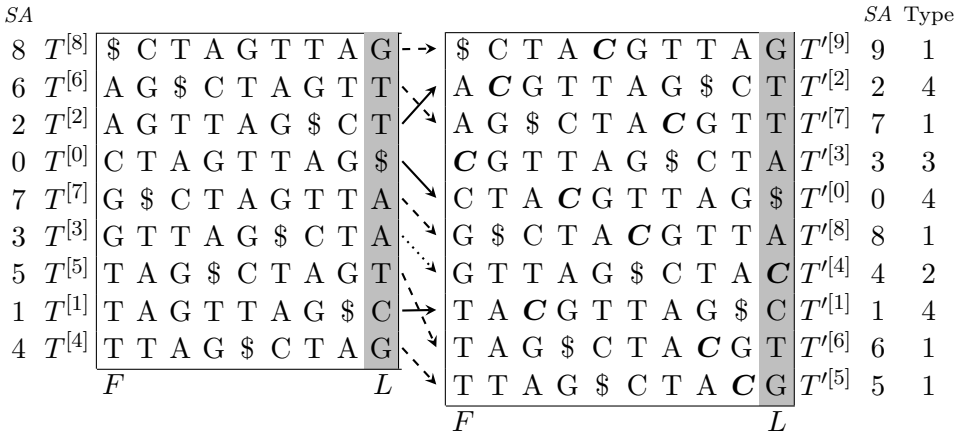
Fig. 1. BWT(CTAGTTAG\$)=GTT\$AATCG and corresponding SA, ISA, LCP.

It is clear that BWT, SA and ISA are strongly related, even if BWT is made of letters, with potential multiplicity, while SA and ISA are made of unique consecutive integers.
In [21,22] we explained how the FM-index, which is based on these three

data structures, can be updated rather than entirely reconstructed when the text T is edited (insertion, deletion or substitution of a letter or a factor). Gallé *et al.* [8] explained how a suffix array can be updated when indexing for grammatical inference purposes.

In the following we consider indifferently cyclic shifts or suffixes. When a text is updated, in both cases, some suffixes (or cyclic shifts) have to be moved to take into account their new lexicographical order. We will use the word "reordering" whenever a new order has to be assigned (insertion of a new position that shifts everything by one position, rotation, ...).

In this article, we focus on the number of elements that have to be reordered when one wants to update a Burrows-Wheeler Transform or a suffix array. This analysis mainly applies to [21,22] but are of interest for any algorithm that updates dynamically a BWT, a suffix array [8] or a more complex stucture such as FM-indexes. Moreover, it greatly helps in understanding the behavior of several SACA, as presented in [20].



(a) BWT(CTAGTTAG\$) (b) BWT(CTA***C***GTTAG\$)

Fig. 2. T, T' and their respective SA and BWT

Without loss of generality we will consider the insertion of a single letter in T at position i. Let T' be the final text obtained after modifying T.

For updating the suffix array or the Burrows-Wheeler Transform, we need to differentiate four types of rows in the conceptual matrix. They correspond to the position of the inserted letter in the cyclic shifts.

During the updating phase, the cyclic shifts are processed from Type 1 to Type 4.

Type 1 the inserted letter appears after \$ and before L. The number of these rows is variable: the closer to the beginning of the text the edit operation

occurs, the larger this number is. We showed in [21,22] that the respective order of these rows is preserved, they are only subject to implicit reordering. See dashed arrows in Fig. 2.
It corresponds to all cyclic shifts $T'^{[n+1]}$ to $T'^{[i+2]}$ in decreasing order.

Type 2 the inserted letter appears in L. An existing row is modified, the letter appearing in L is stored (in our example A) and replaced by the new letter (in our example $\boldsymbol{C}$). See dotted arrow in Fig. 2.
It corresponds to the unique cyclic shift $T'^{[i+1]}$.

Type 3 the inserted letter appears in F. A new row is inserted, the new letter is inserted in column F (in our example $\boldsymbol{C}$) and previously stored letter in column L (in our example A). Since a new row is inserted, indices of all shifted rows are incremented.
It corresponds to the unique cyclic shift $T'^{[i]}$.

Type 4 the inserted letter appears after F and before \$. Each row has a potential impact on the respective order of the other cyclic shifts. It induces a partial reordering of these cyclic shifts that should be performed if needed. See plain arrows in Fig. 2.
It corresponds to all cyclic shifts $T'^{[i-1]}$ to $T'^{[0]}$ in decreasing order.

We already proved ([21]) that only rows of Type 4 are subject to direct reordering, playing a central role in the overall time complexity. The main reason is that the modification appears before \$ in those cyclic shifts. Hence that may modify their lexicographical ranking. Since we focus on the number of reorderings, we restrict our study to Type 4 cyclic shifts.

For a given modification, directly reordered cyclic shifts are either moved up or down. Without loss of generality, we consider the case where cyclic shifts are moved up.

We consider the j-th cyclic shift of Type 4, that is $T'^{[i-j]}$. We denote by k the position of this cyclic shift in the conceptual matrix and ℓ the corresponding LCP value, that is $\ell = LCP[k-1]$.

Lemma 1 *When $j > \ell$, $T'^{[i-j]}$ is not directly reordered and the reordering stage ends.*

PROOF. When we are processing the j-th cyclic shift of Type 4, $T'^{[i-j]}$, all the cyclic shifts $T'^{[r]}$ such that $r > i-j$ have already been ordered. Suppose that $j > \ell$.

Let $p = SA[k-1]$, we also know that $SA[k] = i-j$. For the sake of clarity, we also note $c = T'^{[p]}[\ell]$ and $c' = T'^{[i-j]}[\ell]$.
The modification appears at position j in $T'^{[i-j]}$, hence:

$$T'^{[i-j]}[0\,..\,\ell] = T^{[i-j]}[0\,..\,\ell], \text{ since } j > \ell \quad (1)$$

Following the definitions of LCP and BWT, $T^{[i-j]}$ and $T'^{[p]}$ share a common prefix of length ℓ and $T^{[i-j]}[\ell] > c$. Using (1), we also know that this relation still holds for $T'^{[i-j]}$: prefixes of length ℓ of $T'^{[i-j]}$ and $T'^{[p]}$ are equal and $c' > c$ (see Fig. 3).

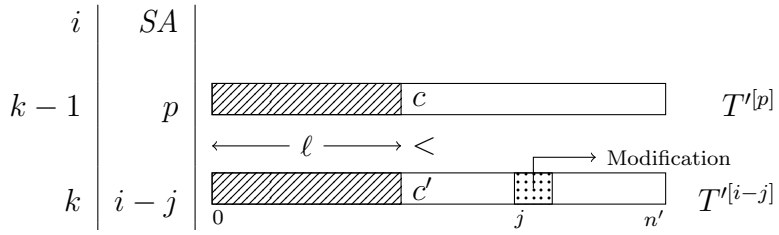


Fig. 3. $T'^{[i-j]}$ does not have to be reordered.

We showed that $T'^{[i-j]}$ is correctly ordered and has not to be reordered. We proved [21, Lemma 3] that if $T'^{[i-j]}$ is correctly ordered then all the cyclic shifts $T'^{[q]}$, $q < i - j$ are correctly ordered as well. Finally, if $j > LCP[k-1]$ then the reordering ends.

□

It gives us a very interesting upper bound on the number of reorderings that should be performed after a given edit operation.

Lemma 2 *The average number of elements to be reordered, after one given edit operation, is at most equal to L_{ave}, the average LCP value.*

PROOF. For a modification occurring at position i in T, we denote by r_i the number of elements to be reordered and by r the array of all r_i, for $0 \leq i \leq n$. We also denote by $PLCP$ the array which is the permuted LCP array where values appear in *position order* rather than *lexicographical order*. $PLCP$ array has the following property [12]: $PLCP[i-1] - 1 \leq PLCP[i]$, $\forall i \in [0\,..\,n-1]$. We insert a 0 at the beginning of the array such that $PLCP[-1] = 0$. By definition $PLCP[n-1] = 0$. We will show that r is a permutation of $PLCP$.

From the previous lemma, $PLCP[i] = 0 \iff r_{i+1} = 0$. Let us consider $PLCP[j\,..\,j']$, where for any k such that $j \leq k \leq j'$, we have $PLCP[k] > 0$, and $PLCP[j-1] = 0$, $PLCP[j'+1] = 0$.

Let $PLCP[j] = L_1$, $L_1 > 0$ and the next values in the $PLCP$ array decrease one by one until a value ℓ_1 at position j_1. Then we have $PLCP[j_1 + 1] = L_2$, with $L_2 \geq \ell_1$, and again the values decrease, one by one, until a value $\ell_2 \leq L_2$. And so forth until $PLCP[j'] = 1$.

Finally, we have $PLCP[j\,..\,j'] = [L_1, L_1-1, .., \ell_1, L_2, L_2-1, .., \ell_{p-1}, L_p, .., 1]$.

Lemma 1 can be rewritten as follows:

$$r_i = \begin{cases} 0 & \text{if } PLCP[i-1] = 0 \\ d & \text{otherwise, with } 1 \le k \le d,\ PLCP[i-k] \ge k \\ & \text{and } PLCP[i-(d+1)] < d+1 \end{cases} \quad (2)$$

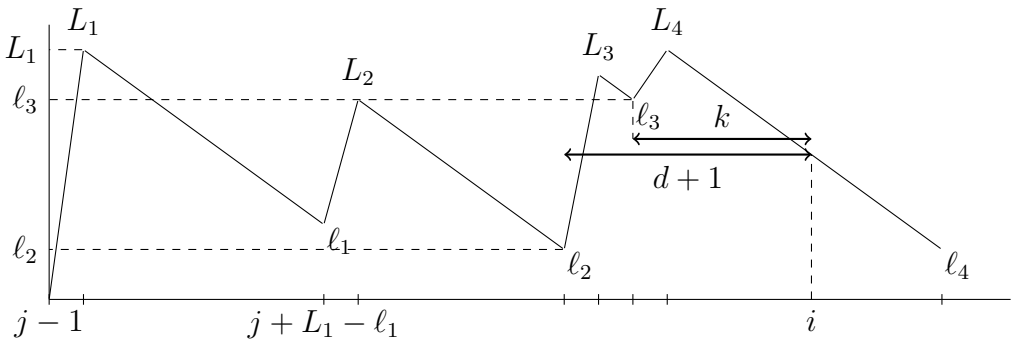


Fig. 4. Computation of $r_i = d$ using the $PLCP$ array, we have $\ell_3 \ge k$ and $\ell_2 < d+1$.

Indeed, a suffix might be reordered whenever the modification belongs to a prefix which appears more than once. When it is not the case anymore (*ie.* $PLCP[i-(d+1)] < d+1$, see Fig. 4), the corresponding suffix does not have to be reordered.

Since $PLCP[j-1] = 0$, $r_j = 0$. By definition $r_{i+1} \le r_i + 1$, hence $r_{j+1} = 1$, $r_{j+2} = 2$ and so forth until position $j + L_1$, where $r_{j+L_1} = L_1$.

The next value r_{j+L_1+1} is ℓ_1 (if $\ell_1 \neq L_1$) since $PLCP[j+L_1+1-(\ell_1+1)] = PLCP[j+L_1-\ell_1] = \ell_1 < \ell_1+1$ (see equation (2)). Then the sequence increases again until reaching L_2 and so forth.

In the special case where $\ell_i = L_i$ (the decreasing sequence is therefore of length 1), with $1 \le i < p$, let $r_k = L_i$, then $r_{k+1} = L_i + 1$ and the sequence keeps increasing until L_{i+1}.

Finally $r[j+1\,..\,j'+1] = [1, .., L_1, \ell_1, .., L_2, \ell_2, .., L_p]$ and it is a permutation of $PLCP[j\,..\,j']$. We have shown this, considering a 0-free factor of $PLCP$. Therefore this is true for each such factor. Moreover using the definition of r_i, we know that we have as many 0's in r than in $PLCP$. Hence, r is a permutation of $PLCP$ and the average number of elements to be reordered is equal to L_{ave}.

□

Hence, when modifying T at position i, we have at most k reordered elements if $PLCP[i-(k+1)] < k+1$. Therefore LCP values can be used for determining

an upper bound of the number of elements to directly reorder for updating a suffix array.

3 Materials and methods: experimental data and their relevance

Based on Lemma 2, we conduct experiments on various texts. Our purpose is to determine the distribution of LCP values and give a better insight of the average number of reorderings when one of the studied texts is updated.

In what follows:

- the percentile LCP value, denoted by L_{perc}, is an upper bound of the number of elements to reorder in 99% of the cases;
- the average LCP value, denoted by L_{ave}, is an upper bound of the number of elements to reorder on average;
- the maximal LCP value, denoted by L_{max}, is an upper bound of the number of elements to reorder, in any case.

Note that these values are upper-bounds since the number of elements to reorder also depends on the inserted (or deleted) letters. For instance, let us consider $T = \overset{0\ 1\ 2\ 3\ 4}{\mathrm{AAAA\$}}$, we have $SA(T) = 4\ 3\ 2\ 1\ 0$. If one inserts a A at position 4 in T, the resulting suffix array is 5 4 3 2 1 0, no element was reordered. On the opposite if one inserts a G at position 4, the resulting suffix array is 5 0 1 2 3 4, which denotes that many elements were reordered.

We subsequently define $R_n = L_{ave}/n$ as the ratio between the average LCP value and the length of the text. In the worst case, for T=A^n, L_{ave}=$n/2$ and R_n=0.5.

Remark 3 *This analysis is of interest for all algorithms that update BWT or SA by reordering elements. It is also of importance for understanding the behavior of suffix array construction algorithms as described by Puglisi et al. [20]. For two texts having the same length, Seward's algorithm [23] is slower when processing the text with the largest L_{ave}. For solving the problem that arises while considering large LCP values, Manzini and Ferragina's algorithm [19] performs a special processing for LCP values that are larger than a threshold chosen by the user. In order to determine that appropriate threshold, one needs to use the distribution of LCP values for various texts, with respect to their length.*

In what follows, we consider two different classes of texts: genomic sequences (unstructured texts) and natural language texts (structured texts). Each studied text possesses specific properties such as number and length of repeats,

entropy and alphabet size.

These texts can be classified according to a measure of repetitiveness, such as the one introduced by Haubold and Wiehe [10]. This measure is based on the length of the shortest unique substring. They define their index of repetitiveness, denoted by I_r, as the logarithm of the ratio of the sum of the shortest unique substring lengths at each position, and the theoretical expectation for a random text having the same composition (as described in [10]). We note that I_r can be computed in a simpler way than the one described in their article, based on suffix trees: compute the LCP array for the original text (concatenated to its reverse complement in the case of a DNA sequence); traverse the array top to bottom; for each two consecutive values, pick the largest one and add it to the final result. In a similar manner, compute the sum for the randomly shuffled original text. The logarithm of the ratio of the two sums corresponds to the I_r value. The authors mention they used a suffix tree, which is a very space-consuming structure, we instead consider the natural inherent LCP values extending the traditional suffix array. It permits a faster computation of the I_r values for genome sequences they were not able to handle, due to memory limitations.

In [4], Fayolle and Ward showed that the average LCP value, under a Markovian model of order one, is $(\log n)/H_1(T) + C$, where $H_1(T)$ is the first-order entropy of T and C is a constant. In order to study if our practical values are diverging significantly from the theoretical values, we are defining $R_{\mathrm{FW}} = L_{ave} \times H_1(T)/\log n$. It is clearly a measure of proximity between the number of direct reorderings that should be performed on average and Fayolle and Ward's theoretical value.

We know that L_{ave} ranges from $O(\log n)$ to $n/2$ depending on the texts. On one hand many articles show that on a randomly generated text, the average LCP is $O(\log n)$, on the other hand one can easily build a text whose L_{ave} is $n/2$ (*e.g.* A^n). Since the texts we chose are neither random nor composed of a single letter, we only know that their L_{ave} is between $O(\log n)$ and $n/2$. In this section, we want to determine the LCP distribution and, among that, the average LCP value on different classes of text containing different types of repetition. This will allow us to evaluate how far typical texts are from the two extremes and how many elements have to be reordered at most for updating a suffix array for these classes of text.

3.1 Genomic Sequences

More and more complete genome sequences are available [16], Haubold and Wiehe selected a subset of 336 organisms (330 prokaryotes and 6 eukaryotes)

and computed I_r values for each genome sequence. They ranked the genome sequences, from the most repeated to the least repeated sequence [1].
Following this study, we focus on the five most repeated among the 330 prokaryote genomes (whose sequence lengths are close to 2 Mbp). The corresponding I_r values range from 6.34 to 3.84, and are related to organisms: *M. flagellatus*, *S. agalactiae*, *D. ethenogenes*, *F. tularensis*, *N. meningitidis*.

We extended our study to encompass the six eukaryote organisms considered in [10]: mouse-ear cress (*A. thaliana*), nematode (*C. elegans*), fruit fly (*D. melanogaster*), yeast (*S. cerevisiae*), human (*H. sapiens*) and mouse (*M. musculus*). The genome sequences are longer: the largest chromosome (chr. 1) of *H. sapiens* contains more than 200 Mbp, and are usually sheltering shorter repeats. These eukaryote organisms have been extensively studied, are well documented and annotated, being part of a larger set of eight model organisms.

In addition we also consider a set of 398 virus genomes (whose sequence length are between 1 kb and 1 Mb). We will study in details two representatives of this set: *G. fumiferanae ichnovirus* and *C. herpesvirus*.

To avoid biased results due to unsequenced portions of genomes, we remove all runs of N's from the sequences. It explains why, for *H. sapiens*, chr1 is shorter than chr2: we had to remove large runs of N's from chr1, while chr2 was not containing that many.

3.2 Natural Language Texts

In addition to these large DNA sequences, we retrieve natural language texts from project Gutenberg [2] (plain ASCII texts, no extra tags). We also retrieved some corpora [3] of the Wikipedia encyclopedia (XML document, Wiki tags, plagiarism) in various languages [4] (Afrikaans, Basque, Bosnian, Estonian, Latin, Occitan).
From project Gutenberg, we obtained a single file of size 91,070,340 bytes by concatenating 224 files from which we removed specific Gutenberg headers and footers. Due to this processing, the size of the alphabet and the type of text considered, we do not expect to see large LCP values since authors are usually trying to avoid as much as possible repeats and plagiarism. In the following

[1] http://ncbi.nlm.nih.gov/pmc/articles/PMC1769404/bin/1471-2105-7-541-S1.pdf
[2] Downloaded from http://www.gutenberg.org
[3] Downloaded on March 30$^{\text{th}}$ 2009 at http://download.wikimedia.org
[4] The Wikipedia languages have been selected because of the length of the corpora which can fit within our computer main memory, each of them representing more than 70,000,000 bytes.

we refer to this text as "etext".
Regarding Wikipedia, each corpus for a given language is stored as a XML document and the content of the encyclopedia is written using the wiki language. The underlying XML structure implies more redundancy, compared to plain natural language texts, namely several predefined templates that are specific to the wiki language for displaying information in a standard format. These templates are often large duplicated pieces of codes, explaining a lot of redundancy that directly impacts LCP values. To evaluate the effect of the templates on the LCP values, we considered two versions of the Afrikaans corpus: the original one and the one from which we removed the most repeated template (see Fig. 6).
Moreover Wikipedia is known to be a cooperative encyclopedia. Some people creates redundant articles and start by filling them using the content of a previous article (e.g. content of "People's Republic of China" is used in "History of China", in the Basque version) leading to larger expected LCP values.

We compute suffix arrays and LCP tables for all the texts, using Manzini and Ferragina's algorithms [19,18] and implementation [5] . These algorithms are chosen for their low space-consumption properties. For each text, we calculate the values L_{max}, L_{ave}, L_{perc} as defined previously (see Sect. 3, page 8). We also compute R_n, R_{FW} and I_r. In the next section we will also plot curves corresponding to L_{ave} and the actual number of reorderings with respect to suffix lengths, for several texts after the insertion of a single letter. We will use these graphs for studying how these values are correlated and are affected by repetitions.

4 Results

In this section we present experimental results obtained from the sets of sequences we previously described. For each set of texts we compute their characteristics and the actual number of reorderings that are needed for maintaining the associated dynamic suffix array. Furthermore we compare the computed L_{ave} value with the real number of reorderings that had to be performed during our experiments on various texts, with various edit operations. We show that they are strongly correlated for all studied texts.

[5] http://web.unipmn.it/~manzini/lightweight/

4.1 Genomic Sequences

We are here presenting two analyses which are based on the values we compute from the *LCP* array or on the curves we plot from the experiments we conducted.

4.1.1 LCP values analysis

The characteristics for the five most repeated prokaryote genomes are presented in Table 1. These short genome sequences have been extensively studied and are well known for containing a lot of short repeats as well as very large repeated regions with a small number of copies. It is therefore unsurprising to observe very large L_{max} relatively to their length. As an example, in *M. flagellatus* 5% of the genome corresponds to a single repeat. The longest repeated region is 143 Kbp long while the second longest is less than 1 Kbp long. For *M. flagellatus*, the L_{ave} value is artificially large because of the L_{max} value. The largest I_r value is achieved for *M. flagellatus* although it possesses only one huge repeat. Therefore we can wonder if a text having a unique huge repeat has to be considered as "more repeated" than a text having many medium-size repeats. Since the average complexity of our algorithm depends on L_{ave}, and we want to test its efficiency in the worst conditions, it is nevertheless pertinent to consider most-repeated texts in the sense of the I_r definition.

Organism	Seq. Len.	L_{max}	L_{ave}	L_{perc}	R_n	R_{FW}	I_r
M. flagellatus	2,971,519	143,034	3,452	113,320	$1.16 \cdot 10^{-3}$	315.49	6.34
S. agalactiae	2,211,485	47,068	546	24,954	$2.47 \cdot 10^{-4}$	50.02	4.84
D. ethenogenes	1,469,720	21,106	377	11,918	$2.56 \cdot 10^{-4}$	36.41	4.03
F. tularensis	1,892,775	33,912	336	14,984	$1.78 \cdot 10^{-4}$	30.49	3.96
N. meningitidis	2,272,360	32,158	261	9,435	$1.15 \cdot 10^{-4}$	24.22	3.84

Table 1
Values for the five most repeated prokaryote genomes.

Although we have large *LCP* values on average, they are still far from the average number of reorderings in the worst case: $n/2$. Values R_n show that on average 0.116% (*M. flagellatus*) to 0.0115% (*N. meningitidis*) elements have to be reordered instead of 50% (the worst case one could expect). R_{FW} values are above 20, meaning that they are significantly different from the theoretical values one can expect from [4] showing that a Markovian model of order one is not suitable for such sequences.

Since we selected the five most repeated prokaryote genome sequences, we do not expect to deal with sequences having larger I_r values. Hence, the R_{FW} values which appear to be significantly larger than 1 are the maximal values one can possibly expect. Consequently, since L_{ave} is much closer to $\log n$ than n, according to R_{FW} and R_n values, maintaining the dynamic suffix array

Organism	Seq. Len.	L_{max}	L_{ave}	L_{perc}	R_n	R_{FW}	I_r
A. thaliana	86,188,477	39,960	53	188	$6.13 \cdot 10^{-7}$	3.9	1.87
C. elegans	100,269,917	38,987	45	196	$4.47 \cdot 10^{-7}$	3.2	1.73
D. melanogaster	120,290,946	30,892	66	1,655	$5.47 \cdot 10^{-7}$	4.8	1.89
S. cerevisiae	12,156,679	8,375	43	922	$3.53 \cdot 10^{-6}$	3.6	1.74

Table 2
Values for eukaryote genomes

Chromosome	Length	L_{max}	L_{ave}	L_{perc}	R_n	R_{FW}	I_r
1	226,212,984	67,631	40	171	$1.77 \cdot 10^{-7}$	2.8	1.61
2	237,898,220	43,034	26	86	$1.09 \cdot 10^{-7}$	1.8	1.76
9	120,983,611	51,976	76	688	$6.28 \cdot 10^{-7}$	5.4	2.82
10	131,735,771	30,751	32	208	$2.43 \cdot 10^{-7}$	2.3	1.60
14	88,290,585	1,292	16	68	$1.81 \cdot 10^{-7}$	1.2	0.41
15	81,920,097	25,713	41	369	$5 \cdot 10^{-7}$	3	1.88
17	79,601,503	15,692	32	302	$4.02 \cdot 10^{-7}$	2.4	2.41
19	56,037,509	3,412	21	88	$3.75 \cdot 10^{-7}$	1.6	0.67
20	59,505,253	866	15	57	$2.52 \cdot 10^{-7}$	1.2	0.53
22	35,058,629	2,331	19	114	$5.42 \cdot 10^{-7}$	1.5	0.84
X	152,538,530	51,821	52	215	$3.41 \cdot 10^{-7}$	3.7	2.40
Y	25,652,954	11,501	98	2,598	$3.82 \cdot 10^{-6}$	7.7	4.31

Table 3
Values for selected chromosomes from human genome.

Chromosome	Length	L_{max}	L_{ave}	L_{perc}	R_n	R_{FW}	I_r
1	191,477,429	7,279	28	345	$1.49 \cdot 10^{-7}$	2	0.85
2	178,392,072	137,338	154	508	$8.64 \cdot 10^{-7}$	10.8	3.22
7	141,878,210	77,175	167	1,755	$1.18 \cdot 10^{-6}$	11.9	3.67
9	120,720,222	8,705	24	237	$1.97 \cdot 10^{-7}$	1.7	0.74
12	117,459,310	90,253	123	623	$1.05 \cdot 10^{-6}$	8.8	2.92
14	121,635,309	79,256	158	1,539	$1.3 \cdot 10^{-6}$	11.3	3.13
15	100,439,974	6,418	25	267	$2.5 \cdot 10^{-7}$	1.8	0.78
17	91,898,202	26,807	35	310	$3.78 \cdot 10^{-7}$	2.5	2.23
19	58,142,230	4,368	20	167	$3.52 \cdot 10^{-7}$	1.5	0.67
X	162,080,892	82,006	219	2,680	$1.35 \cdot 10^{-6}$	15.4	3.64
Y	2,702,555	16,589	192	3,303	$7.09 \cdot 10^{-5}$	17.2	3.72

Table 4
Values for selected chromosomes from mouse genome.

still costs less than rebuilding the entire new suffix array using a linear time construction algorithm.

The eukaryote genomes like the ones considered in Table 2 (*A. thaliana*, *C. elegans*, *D. melanogaster*, *S. cerevisiae*), Table 3 (several chromosomes from *H. sapiens*) and Table 4 (several chromosomes from *M. musculus*), generally contain a smaller percentage of repeats than prokaryote genomes but they are representative among the different species. Hence, on average, LCP values are much smaller than for the five most repeated prokaryote genome sequences.

We still observe some large values for L_{max} but L_{ave} and L_{perc} are smaller, reflecting a different distribution of the LCP values. In this case, we have a very limited number of large repeats: as an example for *H. sapiens* chr. 1, L_{perc} that corresponds to the percentile LCP value is equal to 171 while L_{max} is equal to 67,631. All values between 171 and 67,631 are representing only 1% of all the LCP values. The remainder mostly consists in small repeats such as Short Tandem Repeats, micro and minisatellites or other well known biological structures. Once again, the large R_{FW} values for several chromosomes confirm that a Markovian model of order one is not suitable for every DNA sequences. However, for some other chromosomes, R_{FW} values are close to 1, indicating that despite the low Markovian model, the theoretical value can be a pretty good approximation for less repetitive texts.

We remark that for the most repeated chromosome sequence from *M. musculus*, that is chr. Y, only 0.00709% elements might be reordered on average, while in the worst case only $16,589/2,702,555 \times 100 = 0.61\%$ might be.

Organism	Seq. Len.	L_{max}	L_{ave}	L_{perc}	R_n	R_{FW}	I_r
G. f. ichnovirus	291,597	1,622	22.42	501	$7.69 \cdot 10^{-5}$	3.46	1.47
C. herpesvirus	241,087	297	8.77	13	$3.64 \cdot 10^{-5}$	1.38	1.26

Table 5
Values for virus genomes

Similarly to what we observed for the bacteria and the eukaryotes, in Table 5, we remark that for *G. fumiferanae ichnovirus*, only 0.00769% elements might be reordered on average, while in the worst case only 0.56% might be.

On average a very limited number of reorderings have to be performed with respect to the length of the genomic sequences. Even for the most repeated sequence (*M. flagellatus*) a maximal 5 % of the whole sequence should be reordered if needed.

4.1.2 Curves analysis

In Fig. 5 we trace graphs that study how many reorderings are needed to update the indexes with respect to the length of the texts. For this purpose, we compute L_{ave} for each text length and we count the number of reorderings after the insertion of a single letter for each chosen suffix length. A random letter is inserted at random positions 100,000 times and the average number of reorderings is computed over those insertions.

We choose to plot L_{ave} and the actual number of reorderings for the most repeated prokaryote genome, *M. flagellatus*, whose sequence contains one huge repeated region. We focus on this specific genome that is a representative of all the other prokaryote genomes we mentioned. They all contain several large repeated regions, their curves are similar to Fig. 5. We also consider the two

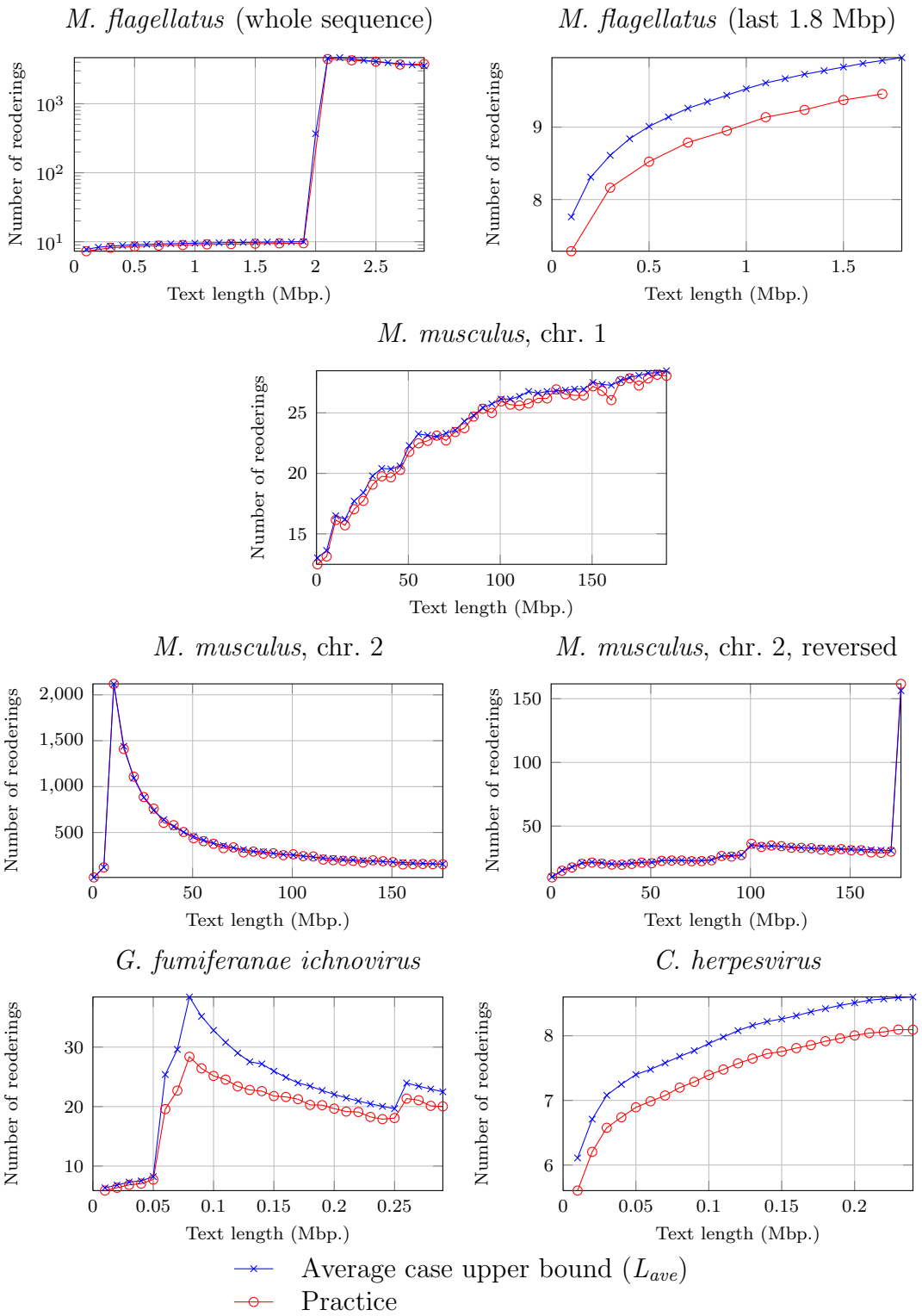


Fig. 5. Theoretical and actual number of reorderings for inserting a single letter in suffixes of DNA sequences.

largest chromosome sequences from *M. musculus* as well as the two viruses that have been reported in Table 5. In both cases sizes are similar but sequences have totally different LCP distribution.

We remark that, since we consider suffixes, when a repetition appears at the beginning of the text, a peak will appear at the end of the graph. The *M. flagellatus* genome sequence, that possesses a single large repeat in the first Mbp, illustrates what has been described before (Fig. 5, whole sequence).

Moreover, when a repeat is close to the end, the number of LCP values that have to be considered is small, and therefore the L_{ave} value will be larger compared to the same repeat closer to the start. Hence, two peaks of the same height, one at the beginning another at the end of a graph, represent two different repeat lengths. When necessary, we also provide the graph for the reversed text so that we can appreciate the bias due to a repeat at the end of a text (Fig. 5, *M. musculus*, chr. 2, reversed). The curve we obtain shows the same behavior for the peak while it reveals a slightly different aspect for the rest of the values.

Due to the length of chr. 1 from *M. musculus*, the respective values for L_{max} and L_{ave} (see Table 4), we are not expecting steep slopes or noticeable peaks. The curve that has been computed confirms our expectations. Contrarily, for chr. 2, L_{max} is twenty times larger than L_{max} for chr. 1 while L_{ave} of chr. 2 is only six times larger than L_{ave} of chr. 1. We are expecting one or several significant peaks. They are confirmed by the curves: it appears that one huge peak (corresponding to a large repeated region located at the end of the sequence) is masking the remaining LCP values.

4.2 Natural Language Texts

Similarly to the study we conducted for genome sequences, we now focus on both LCP values and curves analyses.

4.2.1 LCP values analysis

For etexts, we have very small LCP values as shown in Table 6. L_{perc} is as low as 32 meaning that updating the suffix array is done by reordering at most 32 elements in 99% of the cases.

On the contrary, we have much larger LCP values with Wikipedia corpora. It was also expected, mainly because of inherent duplicates as explained in subsection 3.2. Moreover, removing a single duplicated template in the Afrikaans corpus makes L_{max}, L_{ave} or R_{FW} values drop drastically.

Text	Length	L_{max}	L_{ave}	L_{perc}	R_n	R_{FW}	I_r
etexts	91,070,340	1,727	12	32	$1.32 \cdot 10^{-7}$	1.7	0.82
Afrikaans	68,989,658	34,205	66	383	$9.57 \cdot 10^{-7}$	10.1	3.00
Af. (cleaned)	68,743,934	6,831	31	295	$4.5 \cdot 10^{-7}$	4.7	1.98
Basque	139,868,091	13,879	45	406	$3.22 \cdot 10^{-7}$	6.3	2.16
Bosnian	122,215,463	18,314	40	318	$3.27 \cdot 10^{-7}$	5.9	2.28
Estonian	213,877,916	26,119	39	284	$1.82 \cdot 10^{-7}$	5.4	2.16
Latin	85,735,379	8,099	38	346	$4.43 \cdot 10^{-7}$	5.6	2.15
Occitan	70,250,160	11,003	64	914	$9.11 \cdot 10^{-7}$	9.4	2.64

Table 6
Values for various natural language texts. etexts is a concatenation of several texts from Gutenberg project. The other texts correspond to the Wikipedia corpus in the given language.

The L_{ave} and I_r values are significantly smaller for etexts than for Wikipedia texts. The redundant templates inserted in the latter being absent from the former, it reduces automatically these values. Nevertheless, even for the most repeated text, Afrikaans, we observe that less than 0.000 1% elements should be reordered on average and at most 0.05% in the worst case.

4.2.2 Curves analysis

For etexts we are expecting the L_{ave} to quickly grow until it stabilizes. Doubling the size of the text will not have a significant impact on the L_{ave} value. The curve clearly follows [4], the R_{FW} value indicating that it is close to their theoretical result.

In Fig. 6, the graph for etexts also confirms that we have very short repeats in natural language texts: we do not observe any significant peak. On the other hand, Afrikaans corpus in Wikipedia possesses some large repeats that can be observed with the rapid growth of L_{ave} around 60 MB.
In Fig. 6 (1), original Afrikaans is plotted and a peak clearly appears on the right. It corresponds to one single repeat of length 34,205 within a template of length 245,724. We then removed that particular template from Afrikaans, suppressing the corresponding peak in Fig. 6 (2). Similarly to what we did for *M. musculus*, we reversed the cleaned Afrikaans in order to attenuate the leftmost peak. We observe that Fig. 6 (3) is rather close to the etexts curve and that globally these two curves are following [4].

We showed that L_{ave} is an upper bound of the number of elements to reorder on average. We can observe that the actual number of reorderings is very close to the L_{ave} for the texts we studied. It confirms that computing L_{ave} gives some precious information about the expected number of reorderings that might be performed. It therefore permits to select a dynamic method or another strategy depending on L_{ave}.

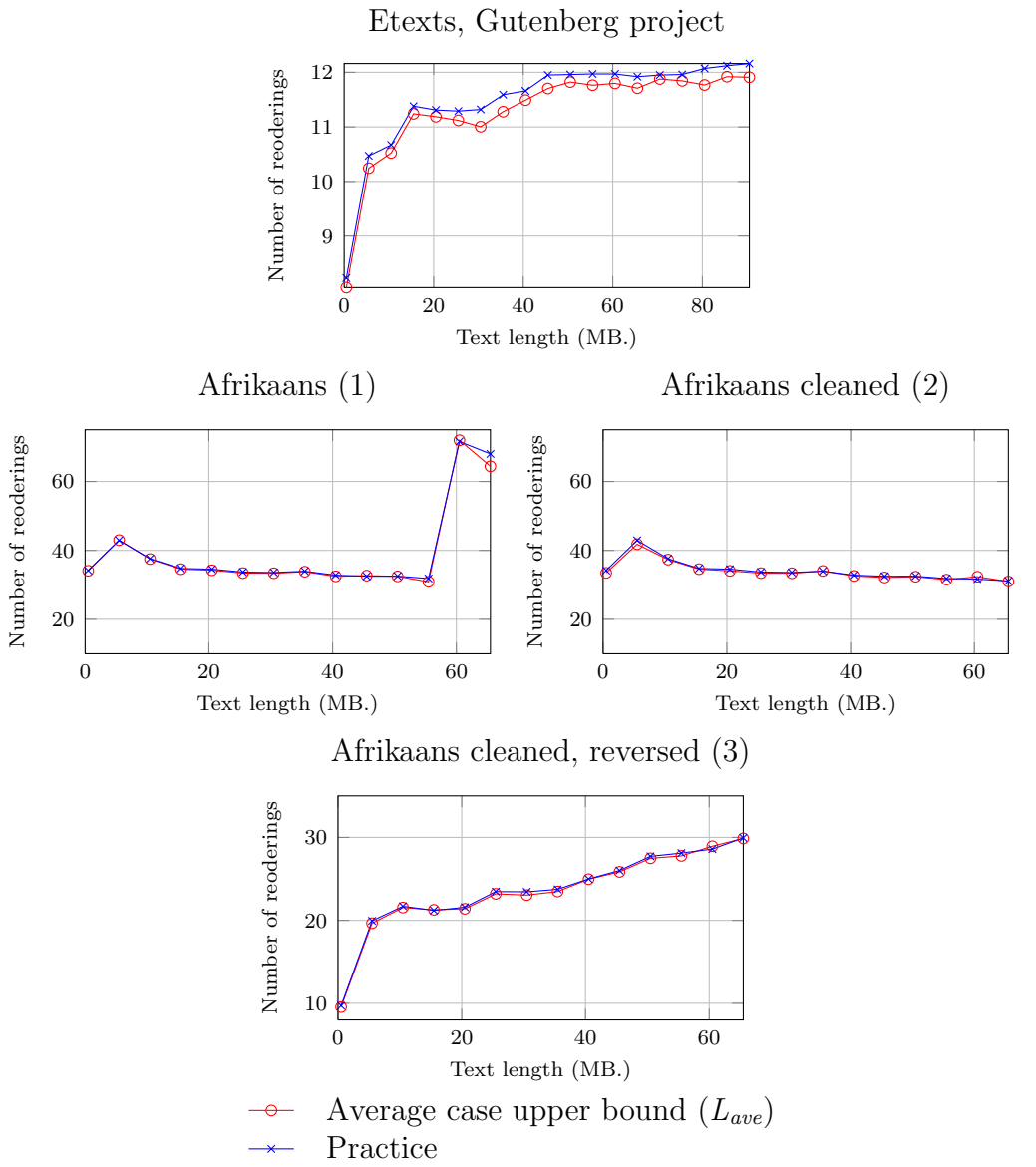


Fig. 6. Actual and theoretical number of reorderings for inserting a single letter in suffixes of natural language texts.

4.3 Studying L_{ave} over a collection of texts

In the previous examples, we saw that L_{ave} is always small with respect to n. In order to confirm this assertion, we consider the L_{ave} of a larger set of texts (bacteria, eukaryotes, viruses, etexts and Wikipedia corpora). More precisely for one given type we have a specific graph where each plot corresponds to one text, its length on the x-axis and its L_{ave} on the y-axis. We also plot a logarithmic function computed using non-linear regression, denoted by *NLR*, that fits best to the data.

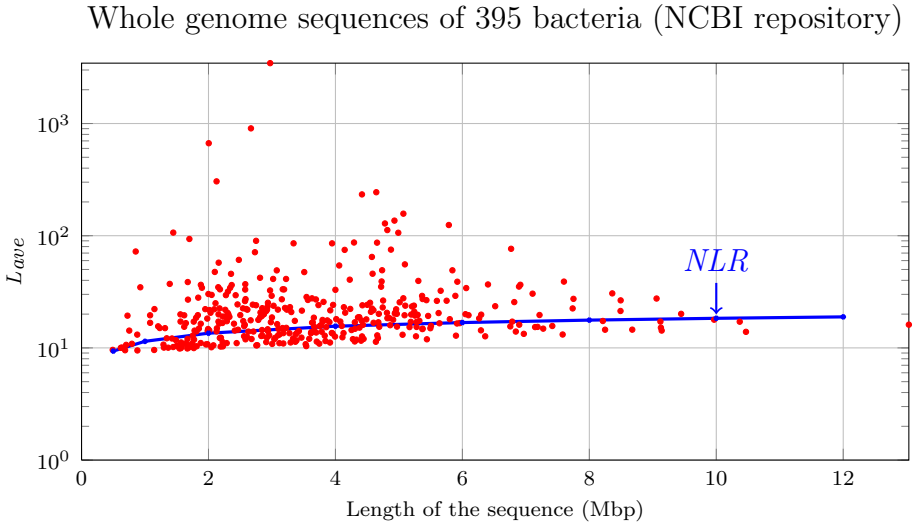


Fig. 7. L_{ave} for bacteria genome sequences

Results for genome sequences are given in Fig. 7 (bacteria), 8 (eukaryotes), 9 (viruses) and for natural language texts in Fig. 10 (etexts) and 11 (wikipedia).

Apart from few outliers, we observe that in all cases L_{ave} values are at most about 100, and mainly concentrated around NLR. This tends to show that L_{ave} is logarithmic not only for random texts, as shown by Fayolle and Ward [4], among others, but also for more specific texts such as genome sequences and natural language texts.

Now, let us focus on each figure and on the few outliers that can be observed. In Fig. 7, a dozen of bacteria have a L_{ave} larger than 100. Their genome sequences contain very large duplicated regions that create a bias in the computation of the L_{ave} (as presented in Table 1).

In Fig. 8, a majority (62 out of 78) of chromosome sequences have a L_{ave} value below 50. However there is clearly a little number of chromosomes that are not so close to NLR. Indeed the eukaryotic sequences are well known for containing numerous repeats, as an example, more than 80% of the human genome sequence is made of repeats. Moreover there exists a high variability between chromosomes, as shown in Table 2. The maximal L_{ave} value in Table 3 is smaller than 100 while the L_{ave} values in Table 4 are larger, these values are still very modest with respect to the length of the chromosomes. The largest values can be observed for sexual chromosomes in both cases. They are due to the recombination rate that usually affects these chromosomes, mainly due to evolution.

In Fig. 9, apart from eight virus sequences whose L_{ave} values are above 30, all the other values are mainly located between 5 and 20.

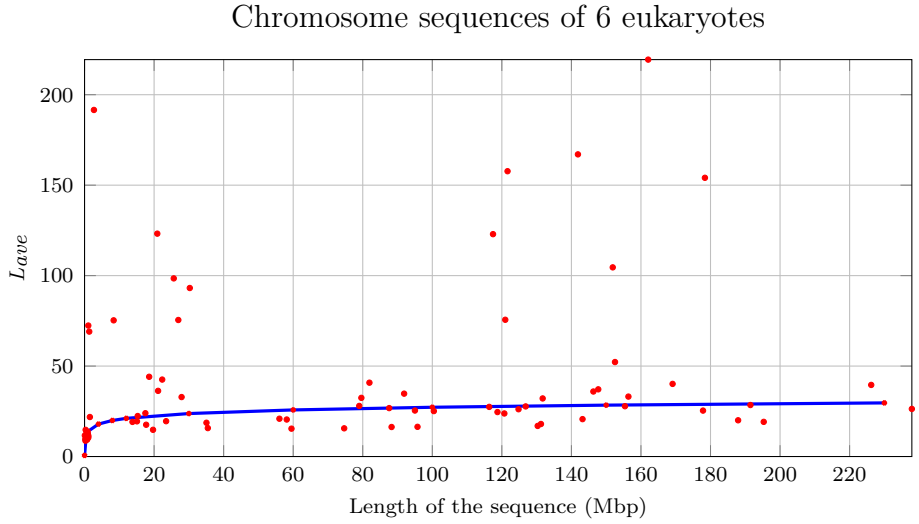


Fig. 8. L_{ave} for eukaryote chromosome sequences (*A. thaliana*, *C. elegans*, *D. melanogaster*, *H. sapiens*, *M. musculus*, *S. cerevisiae*)

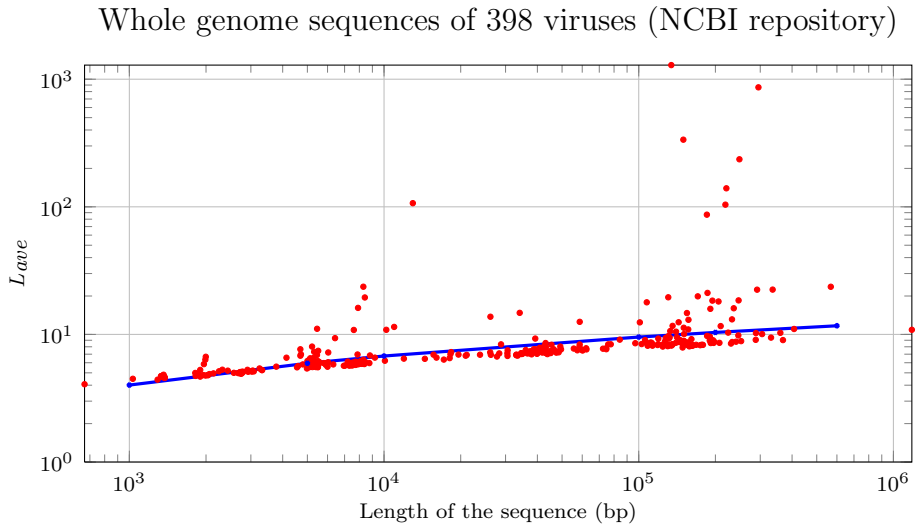


Fig. 9. L_{ave} for virus genome sequences

In Fig. 10, only 31 texts have a L_{ave} above 15. These are all relatively small texts and can be classified in two categories. The first one consists in very small texts (around a thousand letters) which contain lyrics of English traditional songs. For instance one of them is "Heave Away", its refrain is 110 letters long and is repeated twice in this song which is 1,020 letters long. This repetition is not very large but it represents an important ratio of the text (about 20%), which explains that L_{ave} is 32.74.
The second category consists in larger texts (about a hundred thousand letters) where the content is centered using spaces, leading to large repetition of spaces.

In Fig. 11 plots are more scattered than in the other figures. However still a

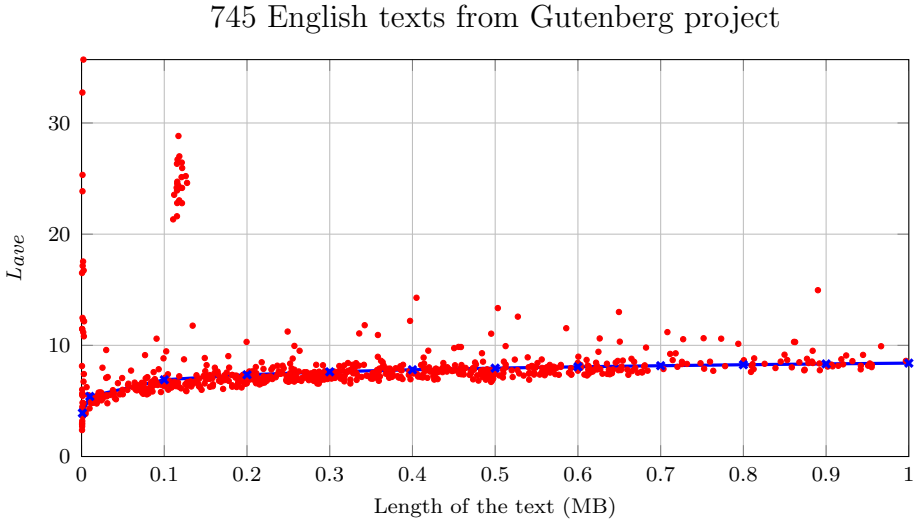


Fig. 10. L_{ave} for etexts from Gutenberg project

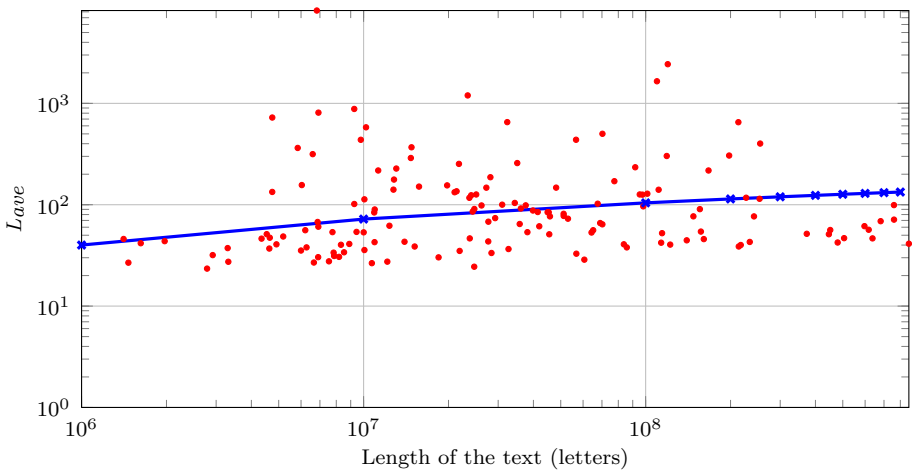


Fig. 11. L_{ave} for Wikipedia corpora

limited number of texts have a quite large L_{ave} value. Namely, 26 texts out of 151 have a L_{ave} value larger than 200. We already explained why Wikipedia corpora contain more repetitions. This is mainly due to "copy and paste" of article content and template duplication. However these duplications appear in large corpora and therefore L_{ave} is generally not much affected. Even in the worst case, L_{ave} is 8,238 for a corpus of size 6,826,160. That means L_{ave} represents only 0.12% of the total size.

5 Conclusions

We studied in this article the average number of reorderings when updating a suffix array-related structure. We showed the correlation between theory and practice, in Lemma 1 and showed in Lemma 2 that L_{ave} is an upper-bound for the number of elements to reorder on average. We conducted experiments on very different types of texts: several DNA sequences of various repetitiveness and natural language texts. These experiments confirmed the correlation between the number of reorderings and L_{ave}.

We furthermore studied the distribution of L_{ave} when the size of the text grows and we have seen that it follows a logarithmic function not only for random texts but also for the texts we mentioned above.

We conclude that the number of elements to be reordered when updating a suffix array-related structure is small and generally logarithmic even on DNA sequences or natural language texts. It is therefore reasonable to update the index rather than building a new one when insertion, deletion or substitution operations occur with the frequencies we measured for both natural language texts and genomic sequences (*eg.* see Fig. 6 in [21]).

References

[1] M. Burrows, D. J. Wheeler, A block-sorting lossless data compression algorithm., Tech. Rep. 124, DEC, Palo Alto, California (1994).

[2] J. G. Cleary, W. J. Teahan, I. Witten, Unbounded length contexts for PPM, Comput. J. 40 (2/3) (1997) 67–76.

[3] J. G. Cleary, I. Witten, Data compression using adaptive coding and partial string matching, IEEE Trans. Commun. 32 (4) (1984) 396–402.

[4] J. Fayolle, M. Ward, Analysis of the average depth in a suffix tree under a Markov model, in: International Conference on the Analysis of Algorithms, 2005.

[5] P. Ferragina, G. Manzini, Opportunistic data structures with applications, in: Proc. of Foundations of Computer Science (FOCS), 2000.

[6] P. Ferragina, G. Manzini, V. Mäkinen, G. Navarro, Compressed representation of sequences and full-text indexes, ACM Trans. Alg. 3 (2007) article 20.

[7] P. Ferragina, G. Manzini, S. Muthukrishnan, The Burrows-Wheeler Transform (special issue), Theor. Comput. Sci. 387 (3) (2007) 197–360.

[8] M. Gallé, P. Peterlongo, F. Coste, In-place update of suffix array while recoding words, in: J. Holub, J. Žďárek (eds.), Proceedings of the Prague Stringology Conference 2008, Czech Technical University in Prague, Czech Republic, 2008.

[9] G. H. Gonnet, R. A. Baeza-Yates, T. Snider, New indices for text: Pat trees and pat arrays, Information Retrieval: Data Structures & Algorithms (1992) 66–82.

[10] B. Haubold, T. Wiehe, How repetitive are genomes?, BMC Bioinformatics 7 (2006) 541+.

[11] J. Kärkkäinen, P. Sanders, Simple linear work suffix array construction, in: Proc. of International Colloquium on Automata, Languages, and Programming (ICALP), 2003.

[12] T. Kasai, G. Lee, H. Arimura, S. Arikawa, K. Park, Linear-time longest-common-prefix computation in suffix arrays and its applications, in: Proc. of Combinatorial Pattern Matching (CPM), vol. 2089 of Lecture Notes in Computer Science, 2001.

[13] P. Ko, S. Aluru, Space efficient linear time construction of suffix arrays, in: Proc. of Combinatorial Pattern Matching (CPM), 2003.

[14] T. W. Lam, W. K. Sung, S. L. Tam, C. K. Wong, S. M. Yiu, Compressed indexing and local alignment of DNA, Bioinformatics 24 (6) (2008) 791–797.

[15] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biology 10 (3) (2009) R25.

[16] K. Liolios, K. Mavrommatis, N. Tavernarakis, N. C. Kyrpides, The genome on line database (GOLD) in 2007: status of genomic and metagenomic projects and their associated metadata., Nuc. Acid Research 36 (2008) 475–479.
URL `http://genomesonline.org`

[17] U. Manber, G. Myers, Suffix arrays: a new method for on-line string searches, in: Proc. of Symposium on Discrete Algorithms (SODA), 1990.

[18] G. Manzini, Two space saving tricks for linear time LCP array computation., in: Proc. 9th Scandinavian Workshop on Algorithm Theory, vol. 3111, 2004.

[19] G. Manzini, P. Ferragina, Engineering a lightweight suffix array construction algorithm, Algorithmica 40 (1) (2004) 33–50.

[20] S. J. Puglisi, W. F. Smyth, A. Turpin, A taxonomy of suffix array construction algorithms, ACM Comp. Surv. 39 (2) (2007) 1–31.

[21] M. Salson, T. Lecroq, M. Léonard, L. Mouchard, A four-stage algorithm for updating a Burrows-Wheeler Transform, Theor. Comput. Sci. 410 (43) (2009) 4350–4359.

[22] M. Salson, T. Lecroq, M. Léonard, L. Mouchard, Dynamic extended suffix array, Journal of Discrete Algorithms 8 (2010) 241–257.

[23] J. Seward, On the performance of BWT sorting algorithms, in: Proc. of Data Compression Conference (DCC), 2000.