Text Sparsification via Local Maxima*

Pilu Crescenzi † Alberto Del Lungo ‡ Roberto Grossi § Elena Lodi ¶
Linda Pagli ‖ Gianluca Rossi **

Abstract

In this paper we investigate some properties and algorithms related to a text sparsification technique based on the identification of local maxima in the given string. As the number of local maxima depends on the *order* assigned to the alphabet symbols, we first consider the case in which the order can be chosen in an arbitrary way. We show that looking for an order that minimizes the number of local maxima in the given text string is an NP-hard problem. Then, we consider the case in which the order is fixed *a priori*. Even though the order is not necessarily optimal, we can exploit the property that the average number of local maxima induced by the order in an arbitrary text is approximately one third of the text length. In particular, we describe how to iterate the process of selecting the local maxima by one or more iterations, so as to obtain a sparsified text. We show how to use this technique to filter the access to unstructured texts, which appear to have no natural division in words. Finally, we experimentally show that our approach can be successfully used in order to create a space efficient index for searching sufficiently long patterns in a DNA sequence as quickly as a full index.

Keywords: computational complexity, NP-completeness, pattern matching, string algorithms, text indexing data structures.

1 Introduction

Several text algorithms solve a certain computational problem $\Pi(n)$ on a text string X of n characters by assuming that the position of each text character is an *access point* to the text itself from which it is then possible to navigate either to the left or to the right. In this sense, we have n access points to X as each character's position is considered as such. One well-know example is the string matching problem, where a pattern string P has to be found as an occurrence in X. Many other examples involve dynamic programming, information retrieval, and molecular biology, just to cite a few.

*A preliminary version of these results has been presented in the *Twentieth Conference on the Foundations of Software Technology and Theoretical Computer Science, 2000* [5]. Research partially supported by Italian MURST project "Algoritmi per Grandi Insiemi di Dati: Scienza e Ingegneria".

†Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Via C. Lombroso 6/17, 50134 Firenze, Italy (piluc@dsi.unifi.it).

‡Dipartimento di Matematica, Università degli Studi di Siena, Via del Capitano 15, 53100 Siena, Italy (dellungo@unisi.it).

§Contact author. Dipartimento di Informatica, Università degli Studi di Pisa, Corso Italia 40, 56125 Pisa, Italy. Email grossi@di.unipi.it, tel. +39 050 221 2793, fax +39 050 221 2726. Part of this work was done while visiting Institut Gaspard Monge at Université de Marne-la-Vallée.

¶Dipartimento di Matematica, Università degli Studi di Siena, Via del Capitano 15, 53100 Siena, Italy (lodi@unisi.it).

‖Dipartimento di Informatica, Università degli Studi di Pisa, Corso Italia 40, 56125 Pisa, Italy (pagli@di.unipi.it).

**Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Via C. Lombroso 6/17, 50134 Firenze, Italy (rossig@dsi.unifi.it).

For the given problem $\Pi(n)$, let $T(n)$ be the time complexity of an algorithm solving $\Pi(n)$, and $S(n)$ be its space occupancy. Text sparsification follows some rules to select a subset of n' access points, $n' < n$, so that one can alternatively solve problem $\Pi(n')$ on the sparsified text. If one can reconstruct the solution for $\Pi(n)$ from that of $\Pi(n')$, we have a new algorithm whose complexity is $O(T(n'))$ time and $O(S(n'))$ space plus the cost of obtaining the solution of $\Pi(n)$ from that of $\Pi(n')$. Sparsification is clearly useful when the latter bounds are respectively better than $T(n)$ or $S(n)$, that is, the new algorithm on the sparsified text is faster or less space demanding than the original algorithm on the full text.

One example of text sparsification is that induced by the run length encoding (RLE), which is a well-known lossless compression technique [10] based on the following simple idea. A sequence of k equal characters (also called *run*) can be encoded by a pair whose first component is the character and whose second component is its frequency k. Apart from its application in data compression, we can also view RLE as a kind of (lossless) text sparsification technique. The RLE implicitly selects a subset of the access points, namely, the ones in the first position of each run.

Another well known example of text sparsification applies to structured texts, in which the notion of *word* is precisely identifiable, for example, by means of delimiter symbols. In this case, a natural way of defining the access points consists of selecting the position of the first character of each word (for example, each character following a delimiter symbol). Differently from RLE, this technique does not immediately provide a lossless compression of the original text but it can be very useful to create a space efficient index for searching the text itself. For example, if we are looking for a given word X, this technique allows us to analyze only the words starting with the first character of X by means of additional data structures, such as suffix arrays [8]. The main drawback of this approach is that it does not generalize to unstructured texts, that is, texts for which there is no clear notion of word available, such as DNA sequences and documents written in Eastern languages. In these cases, one should take all the text positions as access points.

An alternative and simple text sparsification technique, which does not require the notion of word, is based on the *local maxima*. Given a string X over a finite alphabet Σ, let us assume that a total order of the symbols in Σ is specified. We say that an occurrence of a symbol x in X is an access point to X if x is a *local maximum*; that is, both symbols adjacent to x are smaller than x. As in the case of the previous technique based on words, this sparsification technique is *lossy*. For example, assume that x and y are two local maxima: from this information, we can deduce that the symbol immediately after x is smaller than x and the symbol immediately before y is smaller than y. Of course, in general this information is not sufficient to uniquely identify the sequence of symbols between x and y.

Nevertheless, the notion of local maxima has been proven very useful in string matching [1, 4, 11, 12] and dynamic data structures [9, 12] as an extension of the deterministic coin tossing technique [3]. It is well understood in terms of local similarities, by which independent strings that share equal portions have equal local maxima in those portions. This appears particularly useful in unstructured texts. Having identified the local maxima, one can replace the original text of size n with the sparsified text made up by its n' local maxima only. In this paper we will consider two questions related to local maxima and the resulting text sparsification.

- Q1: Suppose that *the order of alphabet* Σ *can be chosen arbitrarily:* How much can a given text be sparsified by applying the local maxima technique?

In order to answer this question, we will introduce the following combinatorial problem: given a text of length n over an alphabet Σ, find an order of Σ which minimizes the number of local maxima (that is, the number of access points). We will then prove that this problem is NP-hard for sufficiently large alphabets. Clearly, the problem can be solved in $O(|\Sigma|!\, n) = O(n)$ time for constant sized alphabets, even though the solution can become quickly impractical for useful values, such as $|\Sigma| \geq 26$ in text files.

- Q2: Suppose that *the order of alphabet* Σ *is fixed a priori:* Is anyway useful the local maxima sparsification technique to filter the access to textual data?

In order to answer this question we show that the sparsification technique produces approximately $n/3$ local maxima on the average for a text of length n. By iterating the technique we can obtain a sufficient degree of sparsification. As a case study, we apply it to the problem of creating a sparse index for searching a given unstructured text. In this case, our approach is able to parse somehow this kind of text having no natural division in words. We then give experimental results to prove the efficiency of our approach in the case of texts that are DNA sequences.

The paper is organized as follows. In Section 2, we prove the NP-hardness of the problem of local maxima related to question Q1 on the alphabet order. In Section 3, we describe our results related to question Q2 on the fixed alphabet order, namely, our sparsification technique. In Section 4, we apply our technique to string matching and text indexing, and discuss its experimental behavior. Finally, we draw some conclusions and observations in Section 5.

2 Arbitrary Alphabet Order and NP-Hardness

In this section, we first describe the combinatorial problem associated with the local maxima sparsification technique. We then show that this problem is NP-hard when one seeks for the best alphabet order to minimize the number of local maxima.

2.1 The combinatorial problem

Let $X = x_1 \cdots x_n$ be a string over a finite alphabet Σ and assume that an order π of the symbols in Σ (i.e., a one-to-one function π from Σ to $\{1, \ldots, |\Sigma|\}$) has been fixed. The *local maxima measure* $\mathcal{M}(X, \pi)$ of X with respect to π is then defined as the number of local maxima that appear in X, that is,

$$\mathcal{M}(X, \pi) = \big|\big\{i : (1 < i < n) \wedge (\pi(x_{i-1}) \leq \pi(x_i)) \wedge (\pi(x_i) > \pi(x_{i+1}))\big\}\big|.$$

The Minimum Local Maxima Number decision problem is then defined as follows:

Instance A string X over a finite alphabet Σ and an integer value K.

Question Does there exist an order π of Σ such that $\mathcal{M}(X, \pi) \leq K$?

Clearly, Minimum Local Maxima Number belongs to NP, since we just have to try non-deterministically all possible orders of the alphabet symbols.

2.2 The NP-hardness result

We now define a polynomial-time reduction from the Maximum Exactly-Two Satisfiability decision problem to Minimum Local Maxima Number. Recall that Maximum Exactly-Two Satisfiability is defined as follows: given a set of clauses with exactly two literals per clause and given an integer H, does there exist a truth-assignment that satisfies at least H clauses? It is well-known that Maximum Exactly-Two Satisfiability is NP-complete [6].

The basic idea of the reduction is to associate one symbol with each variable and with each clause and to force each pair of variable symbols to be either smaller or greater than all clause symbols. The variables whose both corresponding symbols are greater (respectively, smaller) than the clause symbols will be assigned the true (respectively, false) value. The implementation of this basic idea will require several additional technicalities which are described in the rest of the section.

The instance mapping. Let $C = \{c_1, \ldots, c_m\}$ be a set of clauses over the set of variables $U = \{u_1, \ldots, u_n\}$ such that each clause contains exactly two literals, and let H be a positive integer. The alphabet $\Sigma \equiv \Sigma(C)$ of the corresponding instance of Minimum Local Maxima Number contains the following symbols:

- For each variable u_i with $1 \leq i \leq n$, a variable symbol σ_i^u.
- For each clause c_j with $1 \leq j \leq m$, a clause symbol σ_j^c.
- Two special symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ which are the *extremal symbols* in any "reasonable" order π of Σ. That is, either $\pi(\sigma_{\mathtt{m}}) < \pi(\sigma) < \pi(\sigma_{\mathtt{M}})$ for every $\sigma \neq \sigma_{\mathtt{m}}, \sigma_{\mathtt{M}}$, or $\pi(\sigma_{\mathtt{M}}) < \pi(\sigma) < \pi(\sigma_{\mathtt{m}})$ for every $\sigma \neq \sigma_{\mathtt{m}}, \sigma_{\mathtt{M}}$.

The string $X(C)$ over the alphabet Σ corresponding to the instance of Minimum Local Maxima Number is formed by the concatenation of several substrings $X_1^j, X_2^i, Y_{jh}^i, Z_j$ with different goals, where $1 \leq i \leq n$, $1 \leq j, h \leq m$ and $h \neq j$.

Gadgets. In order to define the substrings of $X(C)$, we first introduce the following gadget: given three distinct symbols a, b, and c, let $g(a, b, c) = abcbcba$. The next result states the basic property of the gadget, where $g(a, b, c)^r$ denotes the concatenation of r copies of the gadget.

Lemma 2.1 *Let Σ be an alphabet and let a, b, and c be three distinct symbols in Σ. For any order π of Σ and for any integer $r > 0$, the following holds:*

1. *If $\pi(c) < \pi(b) < \pi(a)$, then $\mathcal{M}(g(a, b, c)^r, \pi) = 2r - 1$.*
2. *If $\pi(a) < \pi(b) < \pi(c)$, then $\mathcal{M}(g(a, b, c)^r, \pi) = 2r$.*
3. *If none of the previous two cases applies, then $\mathcal{M}(g(a, b, c)^r, \pi) \geq 3r - 1$.*

Proof: The proof of the lemma is done by examining all possible cases. Indeed, in Table 1 the occurrences of the local maxima produced by one gadget in correspondence of the six possible orders of the three symbols a, b, and c are shown. Here, '+' means that the character is greater than or equal to the previous one, while '-' indicates that it is smaller. The first a is always preceded by a except for the first occurrence of the gadget in $g(a, b, c)^r$. By looking at Table 1, it is easy to verify the correctness of the three statements of the lemma. □

Minimum	Medium	Maximum	a	b	c	b	c	b	a	Maxima
a	b	c	+	+	+	-	+	-	-	$2r$
a	c	b	+	+	-	+	-	+	-	$3r$
b	a	c	+	-	+	-	+	-	+	$3r - 1$
b	c	a	+	-	+	-	+	-	+	$3r - 1$
c	a	b	+	+	-	+	-	+	-	$3r$
c	b	a	+	-	-	+	-	+	+	$2r - 1$

Table 1: The possible orders of the symbols in $g(a, b, c)^r$ and the number of local maxima.

We also need to extend the gadget by delimiting it with the extremal symbols. We obtain $\sigma_{\mathtt{m}}\, g(a, b, c)\, \sigma_{\mathtt{M}}$, whose property is stated below.

Lemma 2.2 *Let Σ be an alphabet and let $\sigma_{\mathtt{m}}$, $\sigma_{\mathtt{M}}$, a, b, and c be distinct symbols in Σ. For any order π of Σ with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$, and for any integer $r > 0$, the following holds:*

1. *If either* $\pi(a) < \pi(b) < \pi(c)$ *or* $\pi(c) < \pi(b) < \pi(a)$, *then* $\mathcal{M}\big((\sigma_{\mathtt{m}}\, g(a,b,c)\, \sigma_{\mathtt{M}})^r, \pi\big) = 3r-1$.

2. *Otherwise,* $\mathcal{M}\big((\sigma_{\mathtt{m}}\, g(a,b,c)\, \sigma_{\mathtt{M}})^r, \pi\big) = 4r-1$.

Proof: The proof is analogous to that of Lemma 2.1, by inspection of all possible cases in Table 2. □

	Minimum	Medium	Maximum		$\sigma_{\mathtt{m}}$	a	b	c	b	c	b	a	$\sigma_{\mathtt{M}}$	Maxima
$\sigma_{\mathtt{m}}$	a	b	c	$\sigma_{\mathtt{M}}$	-	+	+	+	-	+	-	-	+	$3r-1$
$\sigma_{\mathtt{m}}$	a	c	b	$\sigma_{\mathtt{M}}$	-	+	+	-	+	-	+	-	+	$4r-1$
$\sigma_{\mathtt{m}}$	b	a	c	$\sigma_{\mathtt{M}}$	-	+	-	+	-	+	-	+	+	$4r-1$
$\sigma_{\mathtt{m}}$	b	c	a	$\sigma_{\mathtt{M}}$	-	+	-	+	-	+	-	+	+	$4r-1$
$\sigma_{\mathtt{m}}$	c	a	b	$\sigma_{\mathtt{M}}$	-	+	+	-	+	-	+	-	+	$4r-1$
$\sigma_{\mathtt{m}}$	c	b	a	$\sigma_{\mathtt{M}}$	-	+	-	-	+	-	+	+	+	$3r-1$
$\sigma_{\mathtt{M}}$	a	b	c	$\sigma_{\mathtt{m}}$	+	-	+	+	-	+	-	-	-	$3r-1$
$\sigma_{\mathtt{M}}$	a	c	b	$\sigma_{\mathtt{m}}$	+	-	+	-	+	-	+	-	-	$4r-1$
$\sigma_{\mathtt{M}}$	b	a	c	$\sigma_{\mathtt{m}}$	+	-	-	+	-	+	-	+	-	$4r-1$
$\sigma_{\mathtt{M}}$	b	c	a	$\sigma_{\mathtt{m}}$	+	-	-	+	-	+	-	+	-	$4r-1$
$\sigma_{\mathtt{M}}$	c	a	b	$\sigma_{\mathtt{m}}$	+	-	+	-	+	-	+	-	-	$4r-1$
$\sigma_{\mathtt{M}}$	c	b	a	$\sigma_{\mathtt{m}}$	+	-	-	-	+	-	+	+	-	$3r-1$

Table 2: The possible orders of the the symbols in $\big(\sigma_{\mathtt{m}}\, g(a,b,c)\, \sigma_{\mathtt{M}}\big)^r$ and the number of local maxima.

Substrings X_1^j and X_2^i for extremal symbols. We now show how to build string $X(C)$ from the set C of clauses in polynomial time. The concatenation of the first $m+n$ substrings in the instance $X(C)$ will force $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ to be the extremal symbols of any reasonable order π of Σ. They are then defined as follows, where $r = 14nm^3$:

- For $j = 1, \ldots, m$, $X_1^j = g(\sigma_{\mathtt{m}}, \sigma_j^c, \sigma_{\mathtt{M}})^r$.
- For $i = 1, \ldots, n$, $X_2^i = g(\sigma_{\mathtt{m}}, \sigma_i^u, \sigma_{\mathtt{M}})^r$.

Fact 2.3 *For any order* π *of alphabet* Σ*:*

1. *If* $\sigma_{\mathtt{m}}$ *and* $\sigma_{\mathtt{M}}$ *are extremal symbols in* π, *then* $\mathcal{M}(\sigma_{\mathtt{m}} X_1^1 \cdots X_1^m X_2^1 \cdots X_2^n \sigma_{\mathtt{m}}, \pi) = 2r(m+n)$.

2. *Otherwise,* $\mathcal{M}(\sigma_{\mathtt{m}} X_1^1 \cdots X_1^m X_2^1 \cdots X_2^n \sigma_{\mathtt{m}}, \pi) \geq 2r(m+n) + r$.

Proof: The number of local maxima in each of the $n+m$ substrings X_1^j and X_2^i is that given by Lemma 2.1. Moreover, in case 1, the concatenation of any two adjacent substrings increases the number of maxima by one (including the concatenation of $\sigma_{\mathtt{m}}$ and X_1^1). In case 2, at least one (concatenated) substring gives $3r$ or more maxima, by Lemma 2.1, as it is preceded by a symbol that is smaller or equal according to π. So it produces at least r more maxima than in case 1. □

Substrings Y^i_{jh} for a feasible alphabet order. The concatenation of the next $nm(m-1)$ substrings Y^i_{jh} will force a feasible order, in which each variable symbol is either on the left or on the right of *all* clause symbols. (Intuitively, if symbol σ^u_i is on the left, the value of variable u_i is FALSE. Otherwise, the value of u_i is TRUE.)

Definition 2.4 An order π is *feasible* for alphabet Σ, if no variable symbol σ^u_i satisfies $\pi(\sigma^c_j) < \pi(\sigma^u_i) < \pi(\sigma^c_h)$ for two clause symbols σ^c_j and σ^c_h.

A feasible order π with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ permutes the symbols in Σ to obtain the following scenario. First we find one of the extremal symbols and then zero or more variable symbols (false variables) followed by all clause symbols. Next, we have the remaining variable symbols (true variables), ended by the other extremal symbol. In order to obtain a feasible order we fix $s = 4m$ and define, for $i = 1, \ldots, n$, $j = 1, \ldots, m$ and $h = 1, \ldots, m$, $h \neq j$,

$$Y^i_{jh} = (\sigma_{\mathtt{m}}\, g(\sigma^u_i, \sigma^c_j, \sigma^c_h)\, \sigma_{\mathtt{M}})^s.$$

Fact 2.5 *Let $Y = Y^1_{1,2}Y^1_{2,1}\cdots Y^n_{m,m-1}Y^n_{m-1,m}$ be the concatenation of the substrings Y^i_{jh}, for $i = 1, \ldots, n$, $j = 1, \ldots, m$ and $h = 1, \ldots, m$, $h \neq j$. Then, for any order π of alphabet Σ with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$, we have that π is feasible if and only if $\mathcal{M}(\sigma_{\mathtt{m}} Y \sigma_{\mathtt{m}}, \pi) = 7snm(m-1)/2$. Moreover, if π is not feasible, $\mathcal{M}(\sigma_{\mathtt{m}} Y \sigma_{\mathtt{m}}, \pi) \geq 7snm(m-1)/2 + s$.*

Proof: Suppose π is feasible. By Definition 2.4, for any variable symbol σ^u_i and distinct clause symbols σ^c_j and σ^c_h, we have either $\pi(\sigma^u_i) < \pi(\sigma^c_j), \pi(\sigma^c_h)$ or $\pi(\sigma^c_j), \pi(\sigma^c_h) < \pi(\sigma^u_i)$. Moreover, σ^c_j and σ^c_h must satisfy either $\pi(\sigma^c_j) < \pi(\sigma^c_h)$ or $\pi(\sigma^c_h) < \pi(\sigma^c_j)$. Let us consider the eight combinations of the previous disequalities and examine the substrings Y^i_{jh} and Y^i_{hj} under the resulting orders π obtained by the combinations. By Lemma 2.2, we obtain that one of the substrings gives raise to $3s-1$ local maxima while the other gives raise to $4s-1$ local maxima. We then have to add a local maximum since each substring is followed by $\sigma_{\mathtt{m}}$, giving a total of $7s$ local maxima per pair of substrings Y^i_{jh} and Y^i_{hj}. Since we have $nm(m-1)/2$ pairs of distinct substrings of this kind, we obtain $\mathcal{M}(Y, \pi) = 7snm(m-1)/2$ local maxima in Y.

Vice versa, suppose that $\mathcal{M}(Y, \pi) = 7snm(m-1)/2$ and that, by contradiction, π is not feasible. By Definition 2.4, there are variable symbol σ^u_i and distinct clause symbols σ^c_j and σ^c_h, such that $\pi(\sigma^c_j) < \pi(\sigma^u_i) < \pi(\sigma^c_h)$. By Lemma 2.2, both Y^i_{jh} and Y^i_{hj} give raise to $4s-1$ local maxima each (plus an extra maximum). That is, $\mathcal{M}(Y, \pi) \geq 7snm(m-1)/2 + s$, which is a contradiction. So, π must be feasible. □

Substrings Z_j for a consistent truth assignment. Finally, for each clause c_j with $1 \leq j \leq m$, we have one substring Z_j whose definition depends on the type of the clause and whose goal is to decide the truth-value of each variable by fixing its symbol's position relatively to the clause symbols. In particular, with $1 \leq i, k \leq n$:

- If $c_j = u_i \vee u_k$, then $Z_j = \sigma_{\mathtt{m}}\sigma^c_j\sigma^c_j\sigma^u_k\sigma^c_j\sigma^u_i\sigma^u_i\sigma_{\mathtt{m}}$.
- If $c_j = \neg u_i \vee u_k$, then $Z_j = \sigma_{\mathtt{m}}\sigma^c_j\sigma^c_j\sigma^u_k\sigma^u_i\sigma^c_j\sigma^c_j\sigma_{\mathtt{m}}$.
- If $c_j = u_i \vee \neg u_k$, then $Z_j = \sigma_{\mathtt{m}}\sigma^c_j\sigma^c_j\sigma^u_i\sigma^u_k\sigma^c_j\sigma^c_j\sigma_{\mathtt{m}}$.
- If $c_j = \neg u_i \vee \neg u_k$, then $Z_j = \sigma_{\mathtt{m}}\sigma^u_i\sigma^u_i\sigma^c_j\sigma^u_k\sigma^c_j\sigma^c_j\sigma_{\mathtt{m}}$.

We now highlight the connection between a feasible order π of the alphabet and an assignment τ of truth-values to the variables in the clauses c_j.

Definition 2.6 Given a feasible order π for alphabet Σ with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ and an assignment τ of truth-values to the variables $u_1, \ldots, u_n$, we say that π and τ are *consistent* if, for $i = 1, 2, \ldots, n$,

- $\tau(u_i) = \text{FALSE}$ if and only if $\pi(\sigma_i^u) < \pi(\sigma_j^c)$ for every $1 \leq j \leq m$;
- $\tau(u_i) = \text{TRUE}$ if and only if $\pi(\sigma_i^u) > \pi(\sigma_j^c)$ for every $1 \leq j \leq m$.

Note that, given order π, there is only an assignment τ that is consistent. Vice versa, given τ, we may have more than one order that is consistent.

Fact 2.7 *Given a feasible order π for alphabet Σ with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ and an assignment τ of truth-values, such that π and τ are consistent, we have that, for $1 \leq j \leq n$,*

- $\tau(c_j) = \text{TRUE}$ *if and only if* $\mathcal{M}(Z_j, \pi) = 2$*;*
- $\tau(c_j) = \text{FALSE}$ *if and only if* $\mathcal{M}(Z_j, \pi) = 3$.

Proof: The proof is by inspection of all cases in Table 3. Here, the first five columns describe the 12 possible orderings in π of the symbols of Z_j and of the extremal symbols. The next two columns describe the truth values assigned by τ according to Definition 2.6 (we use the shorthand F for FALSE, and T for TRUE). The remaining eight columns must considered two by two. Each pair of such columns corresponds to one possible form of clause c_j. Its resulting truth-value assigned by τ and the number of local maxima in Z_j under π are reported. For example, let us examine the case in which $c_j = \neg u_i \vee u_k$ (and so, $Z_j = \sigma_{\mathtt{m}}\sigma_j^c\sigma_j^c\sigma_k^u\sigma_i^u\sigma_j^c\sigma_j^c\sigma_{\mathtt{m}}$). If we take the order π such that $\pi(\sigma_{\mathtt{m}}) < \pi(\sigma_k^u) < \pi(\sigma_j^c) < \pi(\sigma_i^u) < \pi(\sigma_{\mathtt{M}})$ (i.e., the eighth row in Table 3), we obtain that $\mathcal{M}(Z_j, \pi) = 3$ and $\tau(c_j) = \text{FALSE}$. □

Orderings in π					Values τ		$c_j = u_i \vee u_k$		$c_j = \neg u_i \vee u_k$		$c_j = u_i \vee \neg u_k$		$c_j = \neg u_i \vee \neg u_k$	
	Min	Med	Max		$\tau(u_i)$	$\tau(u_k)$	$\mathcal{M}(Z_j, \pi)$	$\tau(c_j)$	$\mathcal{M}(Z_j, \pi)$	$\tau(c_j)$	$\mathcal{M}(Z_j, \pi)$	$\tau(c_j)$	$\mathcal{M}(Z_j, \pi)$	$\tau(c_j)$
$\sigma_{\mathtt{m}}$	σ_i^u	σ_j^c	σ_k^u	$\sigma_{\mathtt{M}}$	F	T	2	T	2	T	3	F	2	T
$\sigma_{\mathtt{m}}$	σ_i^u	σ_k^u	σ_j^c	$\sigma_{\mathtt{M}}$	F	F	3	F	2	T	2	T	2	T
$\sigma_{\mathtt{m}}$	σ_j^c	σ_i^u	σ_k^u	$\sigma_{\mathtt{M}}$	T	T	2	T	2	T	2	T	3	F
$\sigma_{\mathtt{m}}$	σ_j^c	σ_k^u	σ_i^u	$\sigma_{\mathtt{M}}$	T	T	2	T	2	T	2	T	3	F
$\sigma_{\mathtt{m}}$	σ_k^u	σ_i^u	σ_j^c	$\sigma_{\mathtt{M}}$	F	F	3	F	2	T	2	T	2	T
$\sigma_{\mathtt{m}}$	σ_k^u	σ_j^c	σ_i^u	$\sigma_{\mathtt{M}}$	T	F	2	T	3	F	2	T	2	T
$\sigma_{\mathtt{M}}$	σ_i^u	σ_j^c	σ_k^u	$\sigma_{\mathtt{m}}$	F	T	2	T	2	T	3	F	2	T
$\sigma_{\mathtt{M}}$	σ_i^u	σ_k^u	σ_j^c	$\sigma_{\mathtt{m}}$	F	F	3	F	2	T	2	T	2	T
$\sigma_{\mathtt{M}}$	σ_j^c	σ_i^u	σ_k^u	$\sigma_{\mathtt{m}}$	T	T	2	T	2	T	2	T	3	F
$\sigma_{\mathtt{M}}$	σ_j^c	σ_k^u	σ_i^u	$\sigma_{\mathtt{m}}$	T	T	2	T	2	T	2	T	3	F
$\sigma_{\mathtt{M}}$	σ_k^u	σ_i^u	σ_j^c	$\sigma_{\mathtt{m}}$	F	F	3	T	2	T	2	T	2	T
$\sigma_{\mathtt{M}}$	σ_k^u	σ_j^c	σ_i^u	$\sigma_{\mathtt{m}}$	T	F	2	F	3	F	2	T	2	T

Table 3: All possible cases for the proof of Fact 2.7.

Fact 2.8 *Given a feasible order π for alphabet Σ with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ and an assignment τ of truth-values, such that π and τ are consistent, we have that τ satisfies H' clauses, where $0 \leq H' \leq m$, if and only if $\mathcal{M}(\sigma_{\mathtt{m}} Z_1 \cdots Z_m \sigma_{\mathtt{m}}) = 3m - H'$.*

Proof: By Fact 2.7, each clause that is satisfied contributes with two local maxima. If the clause is not satisfied, it contributes with three local maxima. □

The proof of correctness. The instance $X(C)$ corresponding to the set C of clauses is defined as the concatenation of the substrings $X_1^j, X_2^i, Y_{jh}^i, Z_j$, where $1 \leq i \leq n$, $1 \leq j, k \leq m$ and $h \neq j$:

$$X(C) = \sigma_{\mathtt{m}}\ X_1^1 \cdots X_1^m\ X_2^1 \cdots X_2^n\ \sigma_{\mathtt{m}}\ Y_{1,2}^1 Y_{2,1}^1 \cdots Y_{m,m-1}^n Y_{m-1,m}^n\ \sigma_{\mathtt{m}}\ Z_1 \cdots Z_m\ \sigma_{\mathtt{m}}.$$

We now prove the following result.

Lemma 2.9 *For any set C of m clauses with exactly two literals per clause and for any integer H, there exists a truth-assignment that satisfies at least H clauses in C if and only if there exists an order π of alphabet $\Sigma \equiv \Sigma(C)$ such that*

$$\mathcal{M}(X(C), \pi) \leq 14nm^2(2nm + 2m^2 + m - 1) + 3m - H.$$

Proof: Assume that there exists a truth-assignment τ to the variables $u_1, \ldots, u_n$ that satisfies at least H clauses in C. We then define a feasible order π of Σ with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$, such that π and τ are consistent (see Definitions 2.4 and 2.6).

1. For any symbol σ different from $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$, $\pi(\sigma_{\mathtt{m}}) < \pi(\sigma) < \pi(\sigma_{\mathtt{M}})$.
2. For any i with $1 \leq i \leq n$, if $\tau(u_i) =$ FALSE then $\pi(\sigma_i^u) < \pi(\sigma_j^c)$ for every $1 \leq j \leq m$, else $\pi(\sigma_j^c) < \pi(\sigma_i^u)$ for every $1 \leq j \leq m$.
3. The remaining order relations between symbols can be easily fixed so that π is feasible.

We now compute $\mathcal{M}(X(C), \pi)$, the number of local maxima in $X(C)$ under alphabet order π. From Fact 2.3 it follows that $\mathcal{M}(\sigma_{\mathtt{m}} X_1^1 \cdots X_1^m X_2^1 \cdots X_2^n \sigma_{\mathtt{m}}, \pi) = 2r(m+n)$, where $r = 14nm^3$. Moreover, because of Fact 2.5, $\mathcal{M}(\sigma_{\mathtt{m}} Y_{1,2}^1 Y_{2,1}^1 \cdots Y_{m,m-1}^n Y_{m-1,m}^n \sigma_{\mathtt{m}}, \pi) = 7snm(m-1)/2$, where $s = 4m$. Hence,

$$\begin{aligned} \mathcal{M}(\sigma_{\mathtt{m}}\, X_1^1 \cdots X_1^m X_2^1 \cdots X_2^n\, \sigma_{\mathtt{m}}\, Y_{1,2}^1 Y_{2,1}^1 \cdots Y_{m,m-1}^n Y_{m-1,m}^n\, \sigma_{\mathtt{m}}, \pi) &= 2r(m+n) + 7snm(m-1)/2 \\ &= 14nm^2(2nm + 2m^2 + m - 1). \end{aligned}$$

Successively, the number of maxima in Z_j will depend on whether clause c_j is satisfied as stated in Fact 2.7. Since we have $H' \geq H$ clauses satisfied in C by the assignment τ, we apply Fact 2.8 to obtain that the number of maxima generated by π on the string $X(C)$ is $\mathcal{M}(X(C), \pi) = 14nm^2(2nm + 2m^2 + m - 1) + (3m - H') \leq 14nm^2(2nm + 2m^2 + m - 1) + 3m - H$.

Conversely, let $W = 14nm^2(2nm+2m^2+m-1)+3m$ and assume that an order π of the symbols in Σ is given such that the number of maxima generated on $X(C)$ is $\mathcal{M}(X(C), \pi) \leq 14nm^2(2nm + 2m^2 + m - 1) + 3m - H \leq W$. We first show that $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ are extremal symbols in π. Suppose that this is not the case. Then, by Fact 2.3, where $r = 14nm^3$, we would have $\mathcal{M}(X(C), \pi) \geq \mathcal{M}(\sigma_{\mathtt{m}} X_1^1 \cdots X_1^m X_2^1 \cdots X_2^n \sigma_{\mathtt{m}}, \pi) \geq 2r(m+n)+r > W$, which leads to a contradiction. Next, we show that order π with extremal symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ is feasible. Otherwise, we would have $\mathcal{M}(X(C), \pi) \geq \mathcal{M}(\sigma_{\mathtt{m}} X_1^1 \cdots X_1^m X_2^1 \cdots X_2^n \sigma_{\mathtt{m}}, \pi) + \mathcal{M}(\sigma_{\mathtt{m}} Y \sigma_{\mathtt{m}}, \pi) \geq 2r(m+n) + 7snm(m-1)/2 + s > W$ by Fact 2.3 (where $r = 14nm^3$) and Fact 2.5 (where $s = 4m$), leading to a contradiction.

As a result, π must be feasible and must contain $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ as extremal symbols. Moreover, $\mathcal{M}(X(C), \pi) = 2r(m+n) + 7snm(m-1)/2 + M = 14nm^2(2nm + 2m^2 + m - 1) + M$, where $M = \mathcal{M}(\sigma_{\mathtt{m}} Z_1 \cdots Z_m \sigma_{\mathtt{m}})$ and $M \leq 3m - H$. Let us rewrite M as $M = 3m - H'$, where $H \leq H' \leq m$, so that $\mathcal{M}(\sigma_{\mathtt{m}} Z_1 \cdots Z_m \sigma_{\mathtt{m}}) = 3m - H'$. We now define τ, such that $\tau(u_i) =$ FALSE when $\pi(\sigma_i^u) < \pi(\sigma_j^c)$ for every $1 \leq j \leq m$, and $\tau(u_i) =$ TRUE when $\pi(\sigma_j^c) < \pi(\sigma_i^u)$ for every $1 \leq j \leq m$. We have that π and τ are consistent by Definition 2.6. By Fact 2.8, since $\mathcal{M}(\sigma_{\mathtt{m}} Z_1 \cdots Z_m \sigma_{\mathtt{m}}) = 3m - H'$, there are $H' \geq H$ clauses satisfied by τ and the lemma is proved. □

From the polynomial-time reduction stated in Lemma 2.9 and from the fact that MINIMUM LOCAL MAXIMA NUMBER belongs to NP it follows the following theorem.

Theorem 2.10 MINIMUM LOCAL MAXIMA NUMBER *is* NP*-complete.*

3 Fixed Alphabet Order and Text Sparsification

We have seen in Section 2 that the problem of assigning an ordering to the alphabet characters of a sequence which minimizes the number of local maxima is a hard problem. However, for any fixed string, the number of local maxima produced by *any* ordering is at most half of the length of the string. Hence, we consider the situation in which the alphabet order is *fixed a priori* and see what happens on arbitrary strings. Now, given a string $X = x_1 \cdots x_n$, a local maximum x_i satisfies the condition $\big(x_{i-1} \leq x_i\big) \wedge \big(x_i > x_{i+1}\big)$ since the permutation π is the identity. In Section 3.1, we count the average number of local maxima. In Section 3.2, we exploit this property to devise an iterative sparsification procedure, which is then applied to the problem of string matching, as a case study in Section 4.

3.1 Average number of local maxima

The following lemma guarantees that, for any fixed ordering π, the number of local maxima produced by π on a randomly chosen string is at most one third of the length of the string.

Lemma 3.1 *Let π be a fixed order over an alphabet Σ. If X is a randomly chosen string over Σ of length n, then the expected value of $\mathcal{M}(X, \pi)$ is at most $n/3$.*

Proof: Let $X = x_1 \cdots x_n$ be the randomly chosen string over Σ and let $R(x_k)$ be the random variable that equals to 1 if x_k is a maximum and 0 otherwise, for any k with $1 \leq k \leq n$. For any k with $2 \leq k \leq n-1$,

$$\mathtt{Pr}\left[R(x_k) = 1\right] = \mathtt{Pr}\left[\pi(x_{k-1}) \leq \pi(x_k)\right] \mathtt{Pr}\left[\pi(x_{k+1}) < \pi(x_k)\right].$$

Hence, the probability that x_k is a maximum, assuming that $\pi(x_k) = i$, is

$$\begin{aligned} \mathtt{Pr}\left[R(x_k) = 1 | \pi(x_k) = i\right] &= \sum_{j=1}^{i} \mathtt{Pr}[\pi(x_{k-1}) = j] \sum_{j=1}^{i-1} \mathtt{Pr}[\pi(x_{k+1}) = j] \\ &= \frac{i(i-1)}{|\Sigma|^2}. \end{aligned}$$

Finally, the probability that x_k is a maximum is

$$\begin{aligned} \mathtt{Pr}\left[R(x_k) = 1\right] &= \sum_{i=1}^{|\Sigma|} \mathtt{Pr}\left[R(x_k) = 1 | \pi(x_k) = i\right] \mathtt{Pr}\left[\pi(x_k) = i\right] \\ &= \frac{1}{|\Sigma|^3} \sum_{i=1}^{|\Sigma|} i(i-1) = \frac{1}{3} - \frac{1}{3|\Sigma|^2}. \end{aligned}$$

By linearity of expectation, the expected number of local maxima is

$$\frac{n-2}{3}\left(1 - \frac{1}{|\Sigma|^2}\right)$$

and the lemma follows. □

One implication of Lemma 3.1 is that random strings (which are hard to compress) can be sparsified by means of the local maxima technique so that the number of resulting access points is roughly most one third of the length of the original string.

3.2 The sparsification procedure

We now describe how to exploit Lemma 3.1 to design a sparsification procedure that replaces a given string $X = x_1 \cdots x_n$ with a shorter one made up of only the local maxima. (The new string will not clearly contain the whole original information.) We repeat this simple procedure by computing the local maxima of the new string to obtain an even shorter string, until the required degree of sparsification is obtained. That is, the compressed string is short enough to be efficiently processed, but it still contains enough information to solve a given problem, as we will see later on.

Formally, the sparsification procedure takes string X and the number $k > 0$ of iterations in input. Let us introduce two special symbols $\sigma_{\mathtt{m}}$ and $\sigma_{\mathtt{M}}$ to be used as string delimitators, where $\sigma_{\mathtt{m}}$ is smaller than any character in Σ, while $\sigma_{\mathtt{M}}$ is larger. After delimiting X with $\sigma_{\mathtt{m}}$, the procedure outputs the sparsified string resulting from $\sigma_{\mathtt{m}} X \sigma_{\mathtt{m}}$ in the form of a sequence of pairs $(\sigma_{\mathtt{m}}, l_0)$, (c_1, l_1), (c_2, l_2), $\ldots$, (c_s, l_s), $(\sigma_{\mathtt{m}}, 1)$, where each $c_i \in \Sigma$ is a local maxima in the string that occurs l_i positions to the left of the next local maxima (l_0 is the distance of the first maximum from the beginning of the string, and l_s is the distance of the last maximum from the end of the string including $\sigma_{\mathtt{m}}$).

The sparsification procedure reads string X and integer k, and apply k sparsification iterations as follows. It starts out from the initial sequence $(\sigma_{\mathtt{m}}, 1)$, $(x_1, 1)$, $(x_2, 1)$, $\ldots$, $(x_n, 1)$, $(\sigma_{\mathtt{m}}, 1)$ obtained in linear time from X. Here, all string characters and the two delimiters are possible maxima. For example, let us consider the following string over the alphabet $\Sigma = \{$ `A`, `C`, `G`, `T` $\}$:

$$X{=}\texttt{ACCCACGTGACGAGCACACACGTCGCAGATGCATA}.$$

The initial sequence obtained from X is

$$X_0{=}(\sigma_{\mathtt{m}},1)(\texttt{A},1)(\texttt{C},1)(\texttt{C},1)(\texttt{C},1)(\texttt{A},1)(\texttt{C},1)(\texttt{G},1)(\texttt{T},1)(\texttt{G},1)(\texttt{A},1)(\texttt{C},1)\ldots(\texttt{A},1)(\sigma_{\mathtt{m}},1).$$

Assuming that the characters are ordered according to the ASCII order, the number of local maxima contained in the above string is 11, which is approximately one third of the total length of the string, i.e., 35. The new string obtained after the first iteration of the sparsification procedure is then the following one, where $\sigma_{\mathtt{m}}$ is by default the delimiter at both ends of the string:

$$X_1{=}(\sigma_{\mathtt{m}},4)(\texttt{C},4)(\texttt{T},4)(\texttt{G},2)(\texttt{G},3)(\texttt{C},2)(\texttt{C},4)(\texttt{T},2)(\texttt{G},3)(\texttt{G},2)(\texttt{T},4)(\texttt{T},2)(\sigma_{\mathtt{m}},1).$$

In general, the sparsification iteration takes a long input sequence $(\sigma_{\mathtt{m}}, l_0)$, (c_1, l_1), (c_2, l_2), $\ldots$, (c_f, l_f), $(\sigma_{\mathtt{m}}, 1)$ and produces a shorter output sequence $(\sigma_{\mathtt{m}}, l'_0)$, (c'_1, l'_1), (c'_2, l'_2), $\ldots$, (c'_g, l'_g), $(\sigma_{\mathtt{m}}, 1)$, where each $c'_i \in \Sigma$ is a local maxima in the input sequence and occurs l'_i string positions to the left of the next local maxima (l'_0 is the distance of c'_1 from the beginning of the string and l'_g is the distance of c'_g from the delimiter $\sigma_{\mathtt{m}}$ at the end of the string). We have seen that $g = f/2$ in the worst case and $g \approx f/3$ on the average by Lemma 3.1.

Some care must be taken with alphabets of small size. At each iteration in the sparsification procedure, at least one character of the alphabet disappears from the new string, since the smallest character in the alphabet is not selected as local maximum. This fact can be a limitation, for instance, in binary sequences ($|\Sigma| = 2$) or DNA sequences ($|\Sigma| = 4$). Indeed, we can apparently apply the sparsification procedure no more than $|\Sigma|$ times. We can circumvent this problem by extending alphabet Σ to the new alphabet $\Sigma \times \mathbb{N}$ under the lexicographic order. Here, the characters become the pairs belonging to the sequences obtained by sparsification, and the alphabet order is the lexicographic one defined over pairs. That is, when two local maxima c_i and c_j are equal, we use their offsets l_i and l_j to decide the outcome of their comparison.

Going on in our example, we observe that the new alphabet consists of six characters, each one composed by a character of $\Sigma - \{\texttt{A}\}$ and a natural number. Since these new characters are ordered according to the lexicographical order, we have that the number of local maxima contained in the above string is 4, which is approximately one third of the total length of the string, i.e., 11. The

new string obtained after the second iteration of the sparsification procedure is then the following one:

$$X_2 = (\sigma_{\mathtt{m}},8)(\mathtt{T},6)(\mathtt{G},9)(\mathtt{T},7)(\mathtt{T},6)(\sigma_{\mathtt{m}},1).$$

Sparsification terminates successfully when it obtains a nonempty sequence X_k. Its time complexity is proportional to the total length of the sequences X_i, where $|X_0| = n$ (without accounting for the delimiters) and $|X_i| < |X_{i-1}|/2$ for $1 \leq i \leq k$.

Lemma 3.2 *Given a string of length n, its sparsification takes a total of $O(n)$ time.*

In order to see why the sparsification technique is useful, suppose to have a problem $\Pi(n)$ on strings of total size n with time complexity $T(n)$ and space complexity $S(n)$. Sparsification reduces the size of the original input strings from n to n', so that problem $\Pi(n')$ is solved in time $O(T(n'))$ or space $O(S(n'))$ plus the cost of obtaining the solution of $\Pi(n)$ from that of $\Pi(n')$. We give an example of this method in Section 4 for the exact string matching problem. The application of our method to other important string problems, such as multiple sequence alignment and matching with errors, deserves further study.

4 Text Sparsification for String Matching and Text Indexing

As a case study, we consider now the basic problem of searching a pattern string P in a text string T of length n, called string matching. We denote the ith character of T by $T[i]$, and the substring made up of characters in positions $i, \ldots, j$ by $T[i, j]$. Here, one wants to identify all text positions $1 \leq i \leq n - |P| + 1$, such that $T[i, i + |P| - 1] = P$, where $|P|$ denotes the length of P. In order to solve the string matching problem, one can process both P and T by means of the sparsification procedure described in Section 3.2. Then, searching for P in T can be done by comparing and matching the local maxima only. Whenever a match is detected, it is sufficient to perform a full comparison of P against the text substring at hand to verify that is not a false occurrence. Unfortunately, some border effects may complicate the approach so far described. They are discussed in Section 4.1 and Section 4.2. Moreover, the number of times we apply the sparsification on the text must be related to the length of the patterns we are going to search for. At iteration i of the sparsification, let

$$m_i = \max\{l_0, l_1, \ldots, l_f\}$$

denote the longest distance between two consecutive maxima in the resulting sparsified text $T_i = (\sigma_{\mathtt{m}}, l_0)$, (c_1, l_1), (c_2, l_2), $\ldots$, (c_f, l_f), $(\sigma_{\mathtt{m}}, 1)$. It follows that $|P|$ cannot be less than m_i. Indeed, performing too many iterations of the sparsification could drop too many characters between two consecutive local maxima selected in the last iteration. As a result, we cannot find the pattern because it is too short compared to the distance between consecutive maxima. Consequently, we adopt the assumption that $|P| \geq m_i$ from now on.

The ideas described in this section are useful to solve the problem of building a succinct text index on strings that are difficult to compress, such as DNA sequences. Given the text T, we wish to build an index for fast searching in T, that is, the $O(|P|+n)$ searching cost of the string matching algorithms drops to $O(|P|)$ plus an output sensitive cost on the occurrences found, at the price of increasing the additional space from $O(m)$ to $O(n)$, needed to keep the index in memory. Let's see how to build an index for T. We run the sparsification procedure described in Section 3.2 on T, so that we identify the positions $1 < i_1 < i_2 < \cdots < i_t < n$ in T corresponding to the local maxima found by sparsification. Then, we build a text index (e.g., suffix array) storing only the suffixes starting at those positions, namely, suffixes $T[i_1, n]$, $T[i_2, n]$, $\ldots$, $T[i_t, n]$ (see [7] for a definition

of sparse index). For example, using the suffix array we obtain a permutation of $i_1, i_2, \ldots, i_t$ such that the corresponding suffixes are in lexicographic order. Searching P in T is implemented as a binary search of P in the (sorted) permutation thus obtained.

In this way, if $S(n)$ is the space for the index employed in the regular way, we obtain a smaller space complexity $S(t)$, where t can be made as small as approximately $n/3^k$ by running k sparsification iterations. As a result, the occupied space is small compared to the text size itself, and this seems to be a rather interesting feature. When searching for a pattern P under certain conditions on its length, we run the sparsification procedure on P and identify at most two local maxima. One of them is M with the property of being part of any occurrence of P in T. Then, we search only for the pattern suffix starting at M. That is, if $M \equiv P[h]$, we search for $P\big[h, |P|\big]$ in the suffix array. Note that the property of local maxima avoids us the computational bottleneck of searching *all* $|P|$ possible suffixes. Instead, we use local maxima as "synchronizing points" to select *only one* pattern suffix such that, if P occurs in T, then $P[h, |P|]$ must match in the sparse index (recall that we store a subset of the text suffixes in the suffix array).

Finally, we consider only the exact version of string searching. As the search of patterns in DNA applications has to be performed considering the possibility of errors, one should use the approximate, rather than the exact string matching. However, in several algorithms used in practice, finding the exact occurrences of some portions of the pattern in the text [2] is a basic filtering step towards solving the approximate problem due to the large size of the text involved.

The rest of the section is organized as follows. In Section 4.1, we examine the effects of sparsification on the border of the pattern when using the plain alphabet order to determine local maxima. In Section 4.2, we discuss the effects when extending the order to the lexicographic order of $\Sigma \times \mathbb{N}$ (this can be useful with alphabets of small size). This extension adds a new difficulty to the problem as the order among the symbols become context-dependent. We describe the practical behavior our index based on the above ideas in Section 4.3.

4.1 Local maxima with Σ-order

We now discuss the effects of sparsification on the borders of the pattern and of its occurrences in the string matching problem. The same considerations apply to text indexing, which is the problem of interest in the rest of the section. We use the plain order of the symbols in Σ to define the local maxima (see Section 4.2 when the order is extended to $\Sigma \times \mathbb{N}$). We first examine the case in which we do not add delimiters $\sigma_{\mathtt{m}}$ or $\sigma_{\mathtt{M}}$ to the two borders of pattern P. Suppose that we are looking for a pattern in text T=ACAHCBARDBAQAWABQARABCVFL, and observe what happens because of the border effects. The first iteration of sparsification produces

$$T_1 = (\sigma_{\mathtt{m}},2)(\mathtt{C},2)(\mathtt{H},4)(\mathtt{R},4)(\mathtt{Q},2)(\mathtt{W},3)(\mathtt{Q},2)(\mathtt{R},4)(\mathtt{V},2)(\mathtt{L},1)(\sigma_{\mathtt{m}},1),$$

while the second iteration produces $T_2 = (\sigma_{\mathtt{m}},8)(\mathtt{R},6)(\mathtt{W},9)(\mathtt{V},3)(\sigma_{\mathtt{m}},1)$. Let us now apply the first iteration of the sparsification procedure to the pattern AHCBARDBAQAWAB, which occurs in T at position 3. We obtain string the sparsified pattern (H,4)(R,4)(Q,2)(W,3) which correctly occurs in T_1 at position 3. However, if we choose pattern P = WABQARABCVF, which occurs in T at position 14, we obtain P_1 = (Q,2)(R,4)(V,2) after the first iteration; the first character W of P is not chosen as a local maximum (no characters at its left) although it appears in the pattern occurrence in T_1 as (W,3). This situation can arise at each iteration of the sparsification thus limiting its power. Namely, at the next iteration $i = 2$ applied to P_1, all the maxima disappear and P_2 is empty. That is in contrast with the hypothesis that $|P| \geq m_2$, which guarantees that any of the occurrences of P in T contains at least a local maximum (after sparsification). As a result, we cannot proceed with the search, since P_2 is empty whereas $|P| = 11 > 9 = m_2$ implies that we should find a local maximum in P_2.

In the worst case, we can loose one local maximum per border of the pattern at each iteration. Because of this drawback, the stronger condition $|P| > 2m_i$ to search on sparsified strings is introduced in [5]. Here we propose a more refined technique that allows us to use the less strict condition $|P| \geq m_i$. Let's pose at the beginning and at the end of the pattern a delimiter, which can be chosen as one of two special characters $\sigma_{\mathtt{M}}$ and $\sigma_{\mathtt{m}}$.

Let us confine our study to the left border of the strings since the reasoning for the right border is symmetric. Posing $\sigma_{\mathtt{M}}$ as delimiter, we obtain the same effect as that of having a null delimiter; namely, the first character in the pattern at each iteration cannot be a local maximum. In the previous example, character `W` of P will disappear in the presence of $\sigma_{\mathtt{M}}$ as delimiter.

It seems therefore reasonable to put $\sigma_{\mathtt{m}}$ as delimiter. Referring to the previous example, we now have $P_1 =$ `(`$\sigma_{\mathtt{m}}$`,1)(W,3)(Q,2)(R,4)(V,2)(`$\sigma_{\mathtt{m}}$`,1)`. In this case, $\sigma_{\mathtt{m}}$ lets `W` emerge as a local maximum in P and P_2 is no more empty. However, using $\sigma_{\mathtt{m}}$ we observe another phenomenom when the pattern occurs in the text. The leftmost local maximum identified in the pattern is not found in the corresponding occurrence in the text, because the text character preceding that occurrence is greater than $P[1]$. In this case, we will call *spurious* the maximum in P that does not appear in every pattern occurrence. In our example, some occurrences of P could be immediately preceded by `Z` while other occurrences could be preceded by `A`, thus making local maxima `(W,3)` spurious after the first iteration of sparsification.

In general, let P_i denote the outcome of iteration $i > 0$ of the sparsification procedure described in Section 3.2, where $P_0 = P$. Recall that iteration i takes P_{i-1} in input to obtain P_i in output. We use T_i to denote the sparsified text at iteration i, as previously done in Section 3.2.

Definition 4.1 A local maximum of P is *spurious* on the left border of P_i if the following conditions hold:

1. There is an occurrence of P in T that appears as substring P'_{i-1} in T_{i-1} and as substring P'_i in T_i, respectively.
2. The first non-$\sigma_{\mathtt{m}}$ character of P_{i-1} is smaller than the character of T_{i-1} preceding P'_{i-1}.
3. The first non-$\sigma_{\mathtt{m}}$ character of P_{i-1} becomes a local maximum when the sparsification iteration is applied to P_{i-1}. (Hence, the first non-$\sigma_{\mathtt{m}}$ character of P_i does not appear in P'_i.)

The definition of spurious on the right border contains the conditions analogous to 1–3.

In general, we call spurious a local maximum without specifying the border, when this is clear from the context. For an example of spurious local maximum, let us take $P =$ `ACBEBCADACBDCFBCB` and $T =$ `ABADACBFA`<u>`ACBEBCADACBDCFBCB`</u>`BDABAEBCA`, so that sparsification yields after iteration $i = 1$ (here, the pattern occurrence P'_i in T_i is underlined),

$$P_1 = (\sigma_{\mathtt{m}},2)(\mathtt{C},2)(\mathtt{E},2)(\mathtt{C},2)(\mathtt{D},2)(\mathtt{C},2)(\mathtt{D},2)(\mathtt{F},2)(\mathtt{C},2)(\sigma_{\mathtt{m}},1),$$
$$T_1 = (\sigma_{\mathtt{m}},2)(\mathtt{B},2)(\mathtt{D},2)(\mathtt{C},2)(\mathtt{F},3)\underline{(\mathtt{C},2)(\mathtt{E},2)(\mathtt{C},2)(\mathtt{D},2)(\mathtt{C},2)(\mathtt{D},2)(\mathtt{F},2)(\mathtt{C},3)}(\mathtt{D},2)(\mathtt{B},2)(\mathtt{E},2)(\mathtt{C},2)(\sigma_{\mathtt{m}},1).$$

After iteration $i = 2$,

$$P_2 = (\sigma_{\mathtt{m}},4)(\mathtt{E},4)(\mathtt{D},6)(\mathtt{F},4)(\sigma_{\mathtt{m}},1),$$
$$T_2 = (\sigma_{\mathtt{m}},4)(\mathtt{D},4)(\mathtt{F},5)\underline{(\mathtt{E},4)(\mathtt{D},6)(\mathtt{F},5)}(\mathtt{D},4)(\mathtt{E},4)(\sigma_{\mathtt{m}},1).$$

After iteration $i = 3$,

$$P_3 = (\sigma_{\mathtt{m}},4)(\mathtt{E},10)(\mathtt{F},4)(\sigma_{\mathtt{m}},1),$$
$$T_3 = (\sigma_{\mathtt{m}},8)(\mathtt{F},15)\underline{(\mathtt{F},9)}(\mathtt{E},4)(\sigma_{\mathtt{m}},1).$$

The first iteration in which a spurious local maximum appears is $i = 3$. Here, the pair `(E,10)` in P_i corresponds to a spurious local maximum since `E` does not appear in P'_i = `(F,9)`. Indeed, the pattern occurrence P'_{i-1} = `(E,4)(D,6)(F,5)` is preceded by `(F,5)` and `F` > `E`, so that `E` does not appear in P'_i.

We now describe how to identify a local maximum M at iteration i that is surely not spurious in P_i (and so in P), provided that $|P| \geq m_i$. (Recall that M is crucial to achieve efficiency while performing the pattern searching discussed at the beginning of Section 4.) We need two results.

Lemma 4.2 *Assume that $|P| \geq m_i$ and P occurs in T. Then, there is at least one (not spurious) local maximum in P_i at iteration $i > 0$.*

Proof: Let us take an occurrence of P in the text, say, $T[s, s + |P| - 1]$. After iteration i, the condition $|P| \geq m_i$ implies that there is at least a local maximum M inside $T[s, s + |P| - 1]$ at iteration i, because $T[s, s + |P| - 1]$ is at least as long as the maximum distance m_i between two consecutive local maxima in T_i. To see that we have at least one (not spurious) local maximum in P, it suffices to show that M appears also inside P at iteration i, i.e., it appears in P_i. By contradiction, suppose that M is not in P_i. In other words, while sparsifying P_j, with $j < i$, a character C adjacent to M is such that $C > M$. Character C cannot be $\sigma_{\mathtt{m}}$, the smallest by definition, so it must be character C inside P, which therefore appears also in all the pattern occurrences in T. Let us now consider occurrence $T[s, s + |P| - 1]$ and its appearance P'_j in the sparsified text T_j. Let's examine the neighbors of M in P'_j. Either C is still a neighbor of M in P'_j, or it has been deleted because the neighbor of M in P'_j is $C' > C$. In any case, both $C, C' > M$ and so M cannot be a local maximum in P'_j. This means that M cannot be a local maximum in T_j, with $j < i$. Hence, M cannot be a local maximum in $T[s, s + |P| - 1]$ (and T_i) at iteration i, as supposed instead. □

Lemma 4.3 *Assume that $|P| \geq m_i$ and P occurs in T. Then, P has at most one spurious local maximum near to its left (resp., right) border. This spurious maximum corresponds to the first (resp., last) character in P_i.*

Proof: We examine the first iteration j such that P has a spurious local maximum. If $i < j$, then the lemma trivially holds. So, let's assume that $j \leq i$ and that, without loss of generality, the spurious maximum is near to the left border of P (the case of right border is analogous). By Definition 4.1, a spurious maximum near the left border is also the leftmost maximum in P_j. We can therefore write

$$P = P'aP''bP'''$$

because of the sparsification done to obtain

$$P_j = (\sigma_{\mathtt{m}}, l_0), (a, l_1), (b, l_2), \ldots,$$

where a is the spurious maximum and b is the next local maximum (and $l_0 = |P'a|$, $l_1 = |P''b|$). Note that b exists by Lemma 4.2 because $|P| \geq m_i$. Moreover, P' and P'' do not contain local maxima and all maxima in bP''' are non spurious.

If $j = i$, we are done with the proof of the lemma (base step). Otherwise, $j < i$, we must consider iteration $j + 1$ and show that the lemma holds also for this iteration (inductive step). For the sake of discussion, assume that no spurious maximum in on the right border of P. Let us write

$$T = XzYPZ = XzY\underline{P'aP''bP'''}Z$$

where P occurs in T and

$$T_j = \cdots (z, |YP'a| + l_1), (b, l'_2) \cdots$$

Note that a disappears in T_j as it is spurious and that the next local maximum c after b is $l'_2 \geq l_2$ positions apart. Moreover, $z > a$ is the nearest local maxima to the left of the pattern occurrence P'_j by Definition 4.1. (If XzY is empty, then P is a prefix of T and P cannot have a spurious maximum on its left border.) We have three possible cases in P_j and T_j:

1. Case $b < a < z$: $P_{j+1} = (\sigma_{\mathtt{m}}, l_0), (a, l'_1), \ldots$, where $l'_1 \geq l_1 + l_2$, since a remains spurious while b is eliminated. In T_{j+1}, b does not appear since $b < z$. As a result, a is the only spurious maximum near to the left border of P.

2. Case $a < b < z$: $P_j = (\sigma_{\mathtt{m}}, l_0 + l_1), (b, l_2), \ldots$, since a disappears and b becomes spurious (if there is a local maximum c to its right which is $c < b$; note that c cannot be spurious in the left border). Indeed, both a and b do not appear in T_{j+1}.

3. Case $a < z < b$: $P_j = (\sigma_{\mathtt{m}}, l_0 + l_1), (b, l_2), \ldots$, since b becomes the leftmost local maximum (if the next local maximum c to its right is smaller) as a disappears. Moreover, b is not spurious as it appears in T_{j+1}.

The three cases discussed above imply that P has at most one spurious local maximum near to its left border after iteration $j + 1$. This spurious maximum is the leftmost maximum in P and corresponds to the first character in P_{j+1}. More precisely, the leftmost maximum in P is spurious only in case 1 and in case 2 (when $c < b$).

In general, we can use induction on j and an argument similar to the above one. The inductive hypothesis says that, in the worst case, we have a possible spurious maximum in the left border, followed by one or more non spurious maxima, and then another possible spurious maximum in the right border. The first and last character in P_j correspond to these two spurious maxima. When $j = i$, we complete the proof of the lemma. □

In order to detect a non spurious maximum in P, with $|P| \geq m_i$, we apply Lemma 4.2 and Lemma 4.3. In particular, Lemma 4.2 says that P has at least one non spurious maximum M, and so P_i cannot be empty because it contains at least this maximum since $|P| \geq m_i$. Lemma 4.3 gives a clear snapshot of the situation. We have a possible spurious maximum in the left border and in the right border, while all other local maxima between them are non spurious maxima (there is at least one such maximum). We therefore run the sparsification on P and stop it at iteration i. If P_i is empty, then P cannot occur in T. Otherwise, we compute the length of P_i. if P_i contains at least three characters, we choose M as the second character in P_i. If P_i has less than three characters, we choose at most two characters as local maxima. Surely one of them will be M. This allows us to search for patterns of length at least m_i. In summary:

Theorem 4.4 *Let T_i be the sparsified text obtained by running $i > 0$ iterations on T, by using the standard alphabet order. We can search for patterns P of length at least m_i in T by selecting at most two local maxima in P_i.*

4.2 Local maxima with $(\Sigma \times \mathbb{N})$-order

When running sparsification on a text drawn from certain Σ, such as alphabets of small size, we obtain a better result by adopting the lexicographic order on the pairs in $\Sigma \times \mathbb{N}$ to define the local maxima, as described in Section 3.2. However, the order induced by a pair (c, l) is *not* static but it depends on its right context, due to the fact that l may change in the pattern occurrences when c is the last character in P_i, after iteration i of sparsification. This means that (c, l) in P_i may appear as (c, l') in T_i, with $l' > l$.

Let's see an example by choosing text T be string X illustrated in Section 3.2 (and $T_i = X_i$), where $\Sigma = \{$ `A`, `C`, `G`, `T` $\}$. Suppose that $P_{i-1} =$ `(`$\sigma_{\mathtt{m}}$`,7)(C,9)(T,12)(G,3)(T,5)(T,2)(`$\sigma_{\mathtt{m}}$`,1)`.

Consequently, P_i = (σ_m,16)(T,15)(T,7)(σ_m,1), since (T,5) is greater than (T,2). Now, we can have an occurrence of P in T which is followed by a local maximum (T,3) in T_{i-1} at distance 7. Then, (T,2) in P_{i-1} corresponds to (T,9) in T_{i-1} because there are 2+7 positions between the two consecutive local maxima in T_{i-1}. As a result, the pattern occurrence in T_i has the form ...(T,15)(T,9)..., and we miss the occurrence. In other words, (T,5) is not a local maximum in T_{i-1} and so it is spurious in T_i and, moreover, (T,9) is a local maximum inside the pattern occurrence in T_{i-1} that we miss in P_{i-1} because it corresponds to (T,2), which does not appear in P_i.

In general, the last character in the pattern may have an offset shorter than that it has in actual pattern occurrences. This situation may cause a local maximum to disappear and to be replaced by a spurious one. Unfortunately, this situation may propagate at each iteration of the sparsification. We need to handle the right border differently from the left border. In this section, we propose a solution to this drawback, which requires $|P| \geq m'_i$ for a value $m_i \leq m'_i \leq 2m_i$ thus defined. Letting $T_i = (\sigma_\mathtt{m}, l_0)$, (c_1, l_1), (c_2, l_2), ..., (c_f, l_f), $(\sigma_\mathtt{m}, 1)$, we have

$$m'_i = \max\{l_0 + l_1, l_1 + l_2, \ldots, l_{f-1} + l_f\}.$$

We adopt a simple variation of the sparsification of the pattern described in Section 4.1. The first iteration $j = 1$ is like that in Section 4.1. In any successive iteration $j > 1$, we remove from P_j the pair (c, l) immediately preceding $(\sigma_\mathtt{m}, 1)$, which is in the last position of P_j, and we add l to the offset of the pair preceding (c, l). In this way, we drop from the right border of the pattern a character whose offset might change in the occurrences. We then apply the sparsification iteration j as done in Section 4.1. In the left border, we may have a spurious maximum, for which Lemma 4.3 holds. We must therefore study the properties of the right border. As a result, the simple modification of sparsification requires that P must be longer than m_i and, precisely, $|P| \geq m'_i$.

Lemma 4.5 *Assume that $|P| \geq m'_i$ and that P occurs in T. Apply the modified sparsification on P. Then, there is at least one (not spurious) local maximum in P_i at iteration $i > 0$.*

Proof: An immediate generalization of Lemma 4.2 states that, if $|P| \geq m'_i$, there are at least two consecutive non spurious maxima M_i and M'_i in P. In other words, for any pattern occurrence in T, we can find M_i and M'_i in T_j, for $0 \leq j \leq i$. Unfortunately, there is no guarantee that they are also in P_j. However, we now prove that at least M_i appears in P_j.

The basic idea is that, given any two local maxima a and c that are consecutive after iteration j of sparsification, there must exist at least a local maximum b that is between a and c after iteration $j-1$, such that $a \geq b$ and b is the right neighbor of a before starting iteration j. Hence, let's define a_{i-1} as the local maximum between M_i and M'_i in T_{i-1}. Moreover, for $1 \leq j \leq i-2$, we define a_j as the local maximum between a_{j+1} and M'_i in T_j. Note that $(M_i, \cdot) > (a_{i-1}, \cdot)$ and a_{i-1} is the right neighbor of M_i in T_{i-1}. Analogously, $(a_{j+1}, \cdot) > (a_j, \cdot)$ and a_j is the right neighbor of a_{j+1} in T_j, for $1 \leq j \leq i-2$. Here, the offsets are represented by a dot "$\cdot$".

We now define an invariant on P_j that it is maintained by the modified sparsification on P. We always have

$$P_1 = (\sigma_\mathtt{m}, \cdot) \cdots (M_i, \cdot) \cdots (a_{i-1}, \cdot) \cdots (a_{i-2}, \cdot) \ldots (a_1, \cdot) \cdots (M'_i, \cdot) \cdots (\sigma_\mathtt{m}, 1),$$

and for $2 \leq j \leq i-1$,

$$P_j = (\sigma_\mathtt{m}, \cdot) \cdots (M_i, \cdot) \cdots (a_{i-1}, \cdot) \cdots (a_{i-2}, \cdot) \ldots (a_j, \cdot) \cdots (\sigma_\mathtt{m}, 1).$$

It follows from the invariant that

$$P_i = (\sigma_\mathtt{m}, \cdot) \cdots (M_i, \cdot) \cdots (\sigma_\mathtt{m}, 1),$$

and so at least M_i appears in P_i, which proves the lemma. We have therefore to show that the modified sparsification on P preserves the invariant. The base case (P_1) holds as iteration $i = 1$ is standard. Indeed, M_i, a_{i-1}, ..., a_1, and M'_i appear in any pattern occurrence and so in $P_0 = P$. After iteration $i = 1$, they still appear in P_1 by their definition (of local maxima).

For the inductive step (P_j with $j > 1$), the modified sparsification on P removes the pair preceding $(\sigma_{\mathtt{m}}, 1)$ in the last position of P_{j-1}. The removed pair must follow $(a_j, \cdot)$, or in the worst case, it must be $(a_j, \cdot)$ itself. However, $(a_j, \cdot)$ would be anyway deleted at iteration j, as $(a_{j+1}, \cdot) > (a_j, \cdot)$ and $(M_i, \cdot) > (a_{i-1}, \cdot)$. So, P_{j+1} will be of the form claimed in the invariant. This completes the inductive proof. □

From an algorithmic point of view, after the modified sparsification of P, we proceed as described in Section 4.1. Namely, if P_i contains at least three characters, we choose M as the second character in P_i. If P_i has less than three characters, we choose at most two characters as local maxima. Surely one of them will be M. We obtain a result similar to that of Theorem 4.4, except that it refers to the extended alphabet.

Theorem 4.6 *Let T_i be the sparsified text obtained by running $i > 0$ iterations on T, by using the lexicographic order on $\Sigma \times \mathbb{N}$. We can search for patterns P of length at least m'_i in T by selecting at most two local maxima in P_i.*

4.3 Experimental results

We experimented the sparsification procedure with up to $k = 3$ iterations, with the purpose of building a suffix array [8] on the sparsified text for string matching queries. In our experiments, we consider DNA sequences, where $\Sigma = \{\mathtt{A}, \mathtt{T}, \mathtt{C}, \mathtt{G}\}$ is a small alphabet. The sequences are *Saccharomyces Cervisiæ* (file `IV.fna`), *Archeoglobus Fulgidus* (file `aful.fna`) and *Escherichia Coli* (file `ecoli.fna`).

In Table 4, we report some values for iterations $i = 1, 2, 3$ in the text sparsification. The rows of Table 4 are associated with the DNA sequences mentioned above. Column $n_0 = n = |T|$ reports the number of characters in the input texts. For $i > 0$, column $n_i = |T_i|$ contains the number of pairs in the text after iteration i of sparsification. We observe a reduction of about 1/3 at each iteration on the values of n_i. Associated with n_i are two values m_i and m'_i, defined at the beginning of Section 4 and in Section 4.2, respectively. They provide a lower bound on the size of the patterns to search for. We fix the values of m_i and m'_i with an heuristics since there are very few offsets that are very much larger than the majority.

Let's consider for example, the DNA sequence for *Escherichia Coli* (file `ecoli.fna`). In Table 5, we report the distribution of the offsets d between consecutive maxima in the sequence after each iteration $i = 1, 2, 3$ of sparsification. After iteration $i = 1$ almost all the offsets are concentrated in a small range of values. There are 10 pairs of consecutive local maxima in T_1 that are at offset $d = 14$. As for $d > 14$, we have only 3 pairs at offset 15 and one pair at offset 16. If we store these $3 \times 15 + 16 = 61$ characters apart, we can safely set $m_1 = 14$ for file `ecoli.fna` in Table 4. We then add +61 characters (to be indexed in standard fashion) to the suffix array built on the $n_1 = 1418905$ suffixes, sampled in the text with sparsification. In the next iterations $i = 2, 3$, the distribution of the offset values d in Table 5 is kept, with a larger range of offset values which become less frequent. Some considerations are in order.

First, the thresholds on the minimum pattern length of m_i in Theorem 4.4 and of m'_i in Theorem 4.6 are overly pessimistic. In our experiments, we successfully found patterns of smaller length. That is, we ran the sparsification on the patterns and successfully found a non spurious local maximum. For example, in file `ecoli.fna`, we had $m'_3 = 150$. We searched for patterns of length ranging from 120 to 300, and the searches were successful.

	n_0	n_1	m_1	m'_1
`IV.fna`	1532027	459027	18 (+616)	23 (+1312)
`aful.fna`	2178460	658396	13 (+101)	19 (+144)
`ecoli.fna`	4639283	1418905	14 (+61)	19 (+361)

	n_2	m_2	m'_2	n_3	m_3	m'_3
`IV.fna`	146408	37 (+1814)	56 (+3931)	47466	92 (+6618)	151 (+11817)
`aful.fna`	213812	35 (+952)	52 (+1863)	69266	94 (+3781)	138 (+14531)
`ecoli.fna`	458498	37 (+851)	54 (+1655)	148134	98 (+4572)	150 (+10621)

Table 4: Sparsification values for three DNA sequences

Second, it may seem that a number of false matches are caused by discarding the first characters in the pattern and searching just one pattern suffix. Instead, the great majority of these searches did not give raise to false matches due to the local maxima, except for a minority. Specifically, we searched sequence *Saccharomyces Cervisiæ* (file `IV.fna`) for pattern lengths ranging from 151 to 241; sequence *Archeoglobus Fulgidus* (file `aful.fna`) for pattern lengths ranging from 138 to 228; sequence *Escherichia Coli* (file `ecoli.fna`) for pattern lengths ranging from 150 to 240. We increased the pattern length by 10 and, for each fixed length, we repeated 10,000 searches with the patterns of that length randomly chosen as substrings of the text. On a total of 300,000 searches, we counted 332, 55 and 214 searches giving false matches, respectively. The total percentage of false matches with respect to the total number of searches was roughly 0.02%.

Finally, the most important feature of the index is that it saves a significant amount of space. For example, a plain suffix array for indexing file `ecoli.fna` requires about 17.7 megabytes (assuming 4 bytes per access point). Applying one iteration of the sparsification procedure reduces the space to 5.4 megabytes, provided that the pattern length $|P|$ is at least 19; the next two iterations give 1.8 megabytes (for $|P| \geq 54$) and 0.6 megabytes (for $|P| > 150$), respectively. These figures compare favorably with the text size of 1.1 megabytes obtained by encoding symbols with two bits each. The tradeoff between minimum length of searchable patterns and index space seems inevitable as the DNA strings are hard to compress.

5 Conclusion and open questions

In this paper, we have investigated some properties of a text sparsification technique based on the identification of local maxima. In particular, we have shown that looking for the best order of the alphabet symbols is an NP-hard problem. (The strings employed as building blocks in the NP-hardness proof were partially generated by computer.) Then, we have described how the local maxima sparsification technique can be used to filter the access to unstructured texts. Finally, we have experimentally shown that this approach can be successfully used in order to create a space efficient index for searching a DNA sequence as quickly as a full index. The application of our method to other important string problems, such as multiple sequence alignment and matching with errors, seems promising and it is object of further study.

Regarding the combinatorial optimization problem, the main question left open by this paper is whether the optimization version of Minimum Local Maxima Number admits a polynomial-time approximation algorithm. It would also be interesting to accompany the experimental results obtained with DNA sequences by some theoretical results, such as the evaluation of the expected maximal distance between two local maxima or the expected number of false matches.

d	$i=1$	$i=2$	$i=3$
1	1		
2	497067		
3	435434	1	
4	257978	13908	
5	127093	35566	
6	61231	42196	
7	24894	50310	
8	9552	46326	3
9	3924	52318	130
10	1170	38229	230
11	395	34335	598
12	118	34585	1076
13	33	23905	1600
14	10	19100	1942
15	3	17930	3369
16	1	11830	3155
17		9281	4000
18		8318	4570
19		5203	4707
20		3944	4590
21		3407	6255
22		2075	4867
23		1575	5016
24		1412	6100
25		796	4887
26		547	4838
27		450	5765
28		291	4704
29		182	4472
30		156	5224
31		95	4061
32		60	3792
33		64	4542
34		31	3589
35		17	3400
36		20	3973
37		13	2933
38		6	2828

d	$i=1$	$i=2$	$i=3$
39		5	3247
40		1	2552
41		1	2319
42		3	2636
43		2	2073
44		1	1853
45		1	2183
46		1	1635
47			1515
48			1681
49			1354
50			1238
51			1279
52			972
53			914
54			1053
55			808
56			698
57			734
58			552
59			519
60			539
61			444
62			439
63			448
64			301
65			295
66			299
67			220
68			230
69			220
70			167
71			155
72			156
73			122
74			117
75			128
76			82

d	$i=1$	$i=2$	$i=3$
77			75
78			63
79			68
80			60
81			56
82			41
83			39
84			41
85			17
86			30
87			30
88			25
89			22
90			18
91			26
92			18
93			22
94			9
95			6
96			9
97			9
98			13
99			8
100			5
101			2
102			6
103			3
104			1
105			3
107			1
108			2
109			3
111			2
113			1
116			3
127			1
134			1
144			1

Table 5: Distribution of distances d of consecutive local maxima for file `ecoli.fna`. Empty cells denote 0. The value d of the distance is shown in the first column of each table. Data for iteration $i = 1$ of sparsification is reported in the second columns, where distances range from 1 to 16. Data for $i = 2$ is in the third columns, where distances range from 3 to 46. Data for $i = 3$ is in the fourth columns, where distances range from 8 to 144.

References

[1] A. Alstrup, G. S. Brodal, and T. Rauhe. Pattern matching in dynamic texts. In *Proceedings of the 11th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 819–828, San Francisco, CA, 2000.

[2] S. Burkhardt, A. Crauser, H.P. Lenhof, P. Ferragina, E. Rivals, and M. Vingron. Q-gram based database searching using a suffix array (QUASAR). In *Proceedings of the Annual International Conference on Computational Biology (RECOMB)*, 1999.

[3] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, July 1986.

[4] G. Cormode, M. Paterson, S.C. Sahinalp, and U. Vishkin. Communication complexity of document exchange. In *Proceedings of the 11th ACM-SIAM Annual Symposium on Discrete Algorithms*, 2000.

[5] P. Crescenzi, A. Del Lungo, R. Grossi, E. Lodi, L. Pagli, and G. Rossi. Text sparsification via local maxima. In *Proceedings of the Twentieth Conference on the Foundations of Software Technology and Theoretical Computer Science(FSTTCS)*, volume 1974 of *Lecture Notes in Computer Science*, pages 290–301, 2000.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[7] Juha Kärkkäinen and Esko Ukkonen. Sparse suffix trees. *Lecture Notes in Computer Science*, 1090:219–230, 1996.

[8] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[9] K. Mehlhorn, R. Sundar, and C. Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, February 1997.

[10] M. Nelson and J.-L. Gailly. *The Data Compression Book*. M&T Books, 1996.

[11] S.C. Ṣahinalp and U. Vishkin. Symmetry breaking for suffix tree construction (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 300–309, Montréal, Québec, Canada, 23–25 May 1994.

[12] S.C. Ṣahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *37th Annual Symposium on Foundations of Computer Science*, pages 320–328. IEEE, 14–16 October 1996.