Università di Pisa

Dipartimento di Informatica

Technical Report: TR-03-02

Bases of Motifs for Generating Repeated Patterns with Don't Cares

Nadia Pisanti Maxime Crochemore Roberto Grossi Marie-France Sagot

January 24, 2003

ADDRESS: via Buonarroti 2, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Bases of Motifs for Generating Repeated Patterns with Don't Cares *

Nadia Pisanti † Maxime Crochemore ‡ Roberto Grossi § Marie-France Sagot ¶

January 24, 2003

Abstract

We investigate the problem of determining the basis of repeated motifs with don't cares in an input string. We give new upper and lower bounds on the problem, introducing a notion of basis that is provably smaller than (and contained in) previously defined ones. Our basis can be computed in less time and space, and is still able to generate the same set of motifs. We also prove that the number of motifs in all these bases grows exponentially with the quorum, the minimal number of times a motif must appear, which was unnoticed in previous work. We show that a polynomial-time algorithm exists only for fixed quorum.

Keywords: string algorithmics, pattern discovery, motif extraction, repetitions, don't cares.

Running head: bases of motifs.

*A preliminary version of the results in this paper have been described in the technical report IGM-2002-10, July 2002 [17], and in an extended abstract submitted to conference. Work partially supported by the French-Italian program "Vinci" and by the French program bioinformatique EPST 2002 "Algorithms for Modelling and Inference Problems in Molecular Biology".

†Contact author. Dipartimento di Informatica, Università di Pisa, via F. Buonarroti 2, 56127 Pisa, Italy (pisanti@di.unipi.it). Tel. +39-050-2212700, Fax +39-050-2212726. Partially supported by ERCIM fellowship programme 2001-2002 at INRIA Rhône-Alpes, France, and by the Italian MIUR project PRIN "ALINWEB: Algorithmics for Internet and the Web".

‡Institut Gaspard-Monge, University of Marne-la-Vallée, 77454 Marne-la-Vallée CEDEX 2, France and King's College London (maxime.crochemore@univ-mlv.fr). Partially supported by CNRS action AlBio, NATO Science Programme grant PST.CLG.977017, and Wellcome Trust Foundation.

§Dipartimento di Informatica, Università di Pisa, via F. Buonarroti 2, 56127 Pisa, Italy (grossi@di.unipi.it). Partially supported by the Italian MIUR project PRIN "ALINWEB: Algorithmics for Internet and the Web".

¶INRIA Rhône-Alpes, Laboratoire de Biométrie et Biologie Évolutive, Université Claude Bernard, 69622 Villeurbanne cedex, France and King's College London (marie-france.sagot@inria.fr). Partially supported by CNRS-INRIA-INRA-INSERM action BioInformatique and Wellcome Trust Foundation.

1 Introduction

Identifying repetitive patterns in strings (sequences) is a computationally-demanding task on the large data sets available in computational biology, data mining, textual document processing, system security, and other areas; for instance, see [10]. We consider *patterns* with *don't cares* in a given string s of n symbols drawn over an alphabet Σ. The don't care is a special symbol '$\circ$' matching any symbol of Σ; for example, pattern $\mathtt{T{\circ}E}$ matches both TTE and TEE inside $s =$ COMMITTEE. (Note that a pattern cannot have a don't care at the beginning or at the end, as this is not considered informative.) Contrarily to string matching with don't cares, the pattern $\mathtt{T{\circ}E}$ is not given in advance for searching s. Instead, the patterns with don't cares appearing repeated in s are *unknown* and, as such, have to be *discovered* and *extracted* by processing s efficiently. In our example, $\mathtt{T{\circ}E}$ and $\mathtt{M{\circ}{\circ}T{\circ}E}$ are among the patterns appearing repeated in COMMITTEE. In this paper we focus on the patterns called *motifs*, which appear at least q times in s for an input parameter $q \geq 2$ called the *quorum*.

Motivations. Although more general (and computationally demanding) notions of motifs are possible [5, 18, 19], the discovery and extraction of motifs with don't cares is flexible enough for several applications. In computational biology, some properties of biological sequences may be extracted by looking at frequent repetitions with some degree of approximation. For example, motifs have been successfully applied to multiple sequence alignment based on pattern discovery [12], and on promoter detection [20]. In document duplication, similarity and plagiarism, common contents are detected in a set of documents divided into semantic units such as paragraphs or pages. We can represent the distinct (stemmed or categorized) words in each unit by bit vectors as in the vector model for information retrieval [4]. Paragraphs with similar contents are filtered for further processing since they share many common bits giving rise to motifs in the concatenation of the bit vectors. In data mining systems [9], researchers have realized that discovering repetitions called "frequent itemsets" are at the heart of the association rules. We can use the above approach with bit vectors, one bit vector per itemset. The association rules can be based not only upon items appearing together within the given quorum and confidence, but also upon their relation with the items that frequently do *not* appear (a sort of negative association rule). Motifs with don't cares are also useful in system security, where an intrusion is often indicated by a sequence of frequent operations appearing intermixed with other operations. Also in this case, we can break the log file of the operations into units and perform pattern discovery like in document duplication. Very recently also data compression has benefited from motifs with don't cares [2]. These are some well-known formulations addressing the common problem of detecting motifs in several contexts, and revealing its algorithmic relevance.

Unfortunately, the complexity of the algorithms for motif discovery may easily become exponential due to the explosive growth of the motifs in strings, such as in the artificial string $\mathtt{A} \cdots \mathtt{ATA} \cdots \mathtt{A}$ (same number of As on both sides of T) generating many motifs with As intermixed with don't cares, and in other "real" strings over a small alphabet occurring in practice, e.g., DNA sequences. Some heuristics try to alleviate this drawback by reducing the number of interesting motifs to make feasible any further processing of them, but they cannot guarantee sub-exponential bounds in the worst case [13].

In this paper we explore the algorithmic ideas behind motif discovery while getting some insight into their combinatorial complexity and their connections with string algorithmics. Given a motif x for a string s of length n, let us denote the set of positions on s at which the occurrences of x start by $\mathcal{L}_x \subseteq [0..n-1]$, where $|\mathcal{L}_x| \geq q$ is the number of occurrences of x. We single out the *maximal* motifs x, informally characterized as satisfying $|\mathcal{L}_x| \neq |\mathcal{L}_y|$ for any other motif y more *specific* than x, that is, obtained from x by adding don't cares and alphabet letters and/or by replacing one or more don't cares with alphabet letters. In other words, x appears in y but x occurs in s more times than y does, which is considered informative for discovering the repetitions in s. For example, $x = \mathtt{M\circ\circ T\circ E}$ is maximal in COMMITTEE for $q=2$ (and $\mathcal{L}_x = \{2,3\}$) while $z_1 = \mathtt{M\circ\circ\circ\circ E}$ and $z_2 = \mathtt{T\circ E}$ are not maximal since x is more specific than both of them, and has the same number of occurrences ($\mathcal{L}_{z_1} = \mathcal{L}_x$ and $\mathcal{L}_{z_2} = \{5,6\}$). Maximality provides an intuitive notion of relevance as each maximal motif x indirectly represents all non-maximal motifs z that are less specific than it, namely, $\mathcal{L}_z = \mathcal{L}_x + d$, where $\mathcal{L}_x + d$ denotes the set $\{i+d \mid i \in \mathcal{L}_x\}$ of occurrences found at exactly d positions apart from those in $\mathcal{L}_x$. Unfortunately, this property does not bound significantly the number of maximal motifs. For example, $\mathtt{A} \cdots \mathtt{ATA} \cdots \mathtt{A}$ contains an exponential number of maximal motifs for $q = 2$ (see Section 2).

A further requirement on the maximal motifs relies on the notion of *irredundant* motifs. A maximal motif x is redundant if there exist maximal motifs $y_1, \ldots, y_k \neq x$ such that their sets of occurrences satisfy $\mathcal{L}_x = \mathcal{L}_{y_1} \cup \ldots \cup \mathcal{L}_{y_k}$; it is irredundant otherwise. The set of occurrences of a redundant motif can be *covered* by other sets of occurrences while that of an irredundant motif is *not* the union of the sets of occurrences of other maximal motifs. The *basis of the irredundant motifs* of string s with quorum q is the set of all the irredundant motifs in s. Informally speaking, a basis can generate all the other maximal motifs by simple mechanical rules and can be expressed mathematically in the algebraic sense of the term. According to Parida et al. [13], what makes interesting the irredundant motifs is that their number is always upper bounded by $3n$ independently of any chosen $q \geq 2$; moreover, their discovery takes $O(n^3 \log n)$ time because of this bound, notwithstanding the exponential number of maximal motifs that are candidates for the basis.

Our results. We study the complexity of finding the basis of motifs introducing new notions and algorithms to represent all motifs succinctly. Our first finding is showing that, in the worst case, there is an infinite family of strings for which the basis contains $\Omega(n^2)$ irredundant motifs for $q = 2$ (see Section 2). This contradicts the upper bound of $3n$ for any $q \geq 2$ given in [13] as shown in Appendix A (the reader may find in that appendix a counterexample to the charging scheme in [13], which crucially relies on a lemma that is not valid). As a result, the bound of $O(n^3 \log n)$ time in [13] for any q does not hold since it relies on the upper bound of $3n$, thus leaving open the problem of discovering a basis in polynomial time for any q.

Our second finding is that of introducing a new definition called *the basis of the tiling motifs* for a string s with quorum q. The condition for tiling motifs is stronger than that of irredundancy. A maximal motif x is *tiled* if there exist maximal motifs $y_1, \ldots, y_k \neq x$ such that their sets of occurrences satisfy $\mathcal{L}_x = (\mathcal{L}_{y_1} + d_1) \cup \ldots \cup (\mathcal{L}_{y_k} + d_k)$ for some integers $d_1, \ldots, d_k$; it is tiling otherwise. Note that the motifs $y_1, \ldots, y_k$ are not necessarily distinct and the union of their occurrences is taken after displacing them by $d_1, \ldots, d_k$, respectively. Since a redundant motif is also tiled with $d_1 = \cdots = d_k = 0$, a tiling motif is surely irredundant. Hence the basis for the tiling motifs is included in the basis for irredundant motifs while both of them are able to generate the *same* set of motifs with mechanical rules. Although the definition of tiling motifs is derived from that of irredundant ones, the difference is much more substantial than it may appear. The basis of tiling motifs is symmetric, namely, the tiling motifs of $\widetilde{s}$ (the string s in reversed order) are the reversals of the tiling motifs of s whereas the irredundant motifs for strings s and $\widetilde{s}$ are apparently unrelated. Note that symmetry holds also for the entropy and other properties related to the repetitions in strings. Moreover, the number of tiling motifs can be provably upper bounded in the worst case by $n - 1$ for $q = 2$ and they occur in s for a total of $2n$ times at most, whereas we demonstrate that there can be $\Omega(n^2)$ irredundant motifs. We give more details in Section 3, discussing also how to find the longest motifs with a limited number of don't cares.

Finally, our third finding in Section 4 is that of revealing an *exponential* dependency on the quorum q for the number of motifs, both for the basis of irredundant motifs and for the basis of tiling motifs, which was unnoticed in previous work. We prove that there is an infinite family of strings for which the basis contains at least $\binom{\frac{n-1}{2}-1}{q-1} = \Omega(\frac{1}{2^q}\binom{n-1}{q-1})$ tiling (hence, irredundant) motifs.

Consequently, *no* polynomial-time algorithm can exist for finding the basis with *arbitrary* values of $q \geq 2$. Nonetheless, we can prove that the tiling motifs in our basis are less than $\binom{n-1}{q-1}$ in number and occur in s a total of $q\binom{n-1}{q-1}$ times at most. For them there exists a pseudo-polynomial algorithm taking $O\left(q^2\binom{n-1}{q-1}^2\right)$ time, which shows that the tiling motifs can be determined in polynomial time if and only if the quorum q satisfies either $q = O(1)$ or $q = n - O(1)$ (the latter case is hardly meaningful in practice). Experimenting with small strings exhibits a non-constant growth of the basis for increasing values of q up to $O(\log n)$ but larger values of q are possible in the worst case.

Related work. Motif discovery in general has important applications in many areas of research and as a consequence, quite a large amount of literature may be found on the subject. Computational biology is one of the most recent areas that has generated a big need for sophisticated motif discovery techniques. The field is mature enough that papers have started appearing which survey various models and algorithms for inferring different types of motifs from DNA or protein sequences [5, 20, 15]. These surveys contain lists of pointers to other papers, or to publicly available software. Among the mentioned papers, one previous work from one of the authors showed a first concern with bounding the number of motifs produced [18]. The motifs considered there were more complex than the ones treated in this paper and there was no idea of a basis. Consequently, the number of motifs produced could be exponential.

As previously mentioned, the seminal idea of basis was introduced by Parida et al. [13]. That idea was then explored by Pelfrene et al. [16], who introduced in a poster a basis using the notion of *primitive motifs*. They give two alternative definitions (one in the two-page abstract accompanying the poster, and one in the poster itself) claimed to be equivalent. The basis defined in the poster is not symmetric and is a superset of the one presented in this paper. On the other hand, the definition of primitive motifs given in the abstract of [16] is indeed equivalent to that given in this paper and introduced in our technical report [17]. Both were produced independently. In [16], no algorithm for computing the basis is given, although the existence of one in $O(n^3)$ time is claimed for the case $q = 2$ only.

Paper organization. We give some formal definitions and the lower bound for the basis of the irredundant motifs in Section 2. We introduce the basis of the tiling motifs in Section 3, describing its properties and its construction with applications. In Section 4 we study the dependency on the quorum in the complexity of the algorithms. Finally, we draw our conclusions in Section 5.

2 Irredundant Motifs: The Basis and its Size for Quorum $q = 2$

2.1 Notation and terminology

We consider strings that are finite sequences of letters drawn from an alphabet Σ, whose elements are also called *solid characters*. We introduce an additional letter (denoted by $\circ$ and called *don't care*) that does not belong to Σ and matches any letter. The length of a string t with don't cares, denoted by $|t|$, is the number of letters in t, and $t[i]$ indicates the letter at position i in t for $0 \leq i \leq |t| - 1$ (hence, $t = t[0]t[1] \cdots t[|t| - 1]$ also noted $t[0\,..\,|t| - 1]$).

A *pattern* is a string in $\Sigma \cup \Sigma(\Sigma \cup \{\circ\})^*\Sigma$, that is, it starts and ends with a solid character. The pattern occurrences are related to the *specificity relation* $\preceq$. For individual characters $\sigma_1, \sigma_2 \in \Sigma \cup \{\circ\}$, we have $\sigma_1 \preceq \sigma_2$ if $\sigma_1 = \circ$ or $\sigma_1 = \sigma_2$. Relation $\preceq$ extends to strings in $(\Sigma \cup \{\circ\})^*$ under the convention that each string t is implicitly surrounded by don't cares, namely, letter $t[j]$ *is* $\circ$ when $j < 0$ or $j \geq |t|$. Hence, v is *more specific* than u (shortly, $u \preceq v$) if $u[j] \preceq v[j]$ for any integer j. Finally, we say that u *occurs at position* ℓ *in* v if $u[j] \preceq v[j + \ell]$, for $0 \leq j \leq |u| - 1$. Equivalently, we say that u *matches* $v[\ell\,..\,\ell + |u| - 1]$.

For the input string $s \in \Sigma^*$ with $n = |s|$, we consider the occurrences of arbitrary patterns x in s. The *location list* $\mathcal{L}_x \subseteq [0\,..\,n-1]$ denotes the set of all the positions on s at which x occurs. For example, the location list of $x = \mathtt{T}\circ\mathtt{E}$ in $s = \mathtt{COMMITTEE}$ is $\mathcal{L}_x = \{5, 6\}$.

Definition 2.1 (motif) *Given a parameter $q \geq 2$ called* quorum*, we say that pattern x is a* motif *according to s and q when $|\mathcal{L}_x| \geq q$.*

Given any location list $\mathcal{L}_x$ and any integer d, we adopt the notation $\mathcal{L}_x + d = \{\ell + d \mid \ell \in \mathcal{L}_x\}$ for indicating the occurrences in $\mathcal{L}_x$ "displaced" by the offset d.

Definition 2.2 (maximality) *A motif x is* maximal *if any other motif y such that x occurs in y satisfies $\mathcal{L}_y \neq \mathcal{L}_x + d$ for some integer d.*

In other words, making a maximal motif x more specific (thus obtaining y) reduces the number of its occurrences in s. Definition 2.2 is equivalent to that in [13] stating that x is maximal if there exist no other motif y and no integer $d \geq 0$ verifying $\mathcal{L}_x = \mathcal{L}_y + d$, such that $x[j] \preceq y[j + d]$ for $0 \leq j \leq |x| - 1$ (that is, x occurs in y at position d in our terminology).

Definition 2.3 (irredundant motif) *A maximal motif x is* irredundant *if, for any maximal motifs $y_1, y_2, \ldots, y_k$ such that $\mathcal{L}_x = \cup_{i=1}^{k} \mathcal{L}_{y_i}$, motif x must be one of the y_i's. Vice versa, if all the y_i's are different from x, pattern x is said to be* covered *by motifs $y_i, y_2, \ldots, y_k$.*

The *basis of irredundant motifs* for string s is the set of *all* irredundant motifs in s, useful as a generator for all maximal motifs in s (see [13]). The size of the basis is the number of irredundant motifs contained in it. We illustrate the notions given so far by employing the example string $s = \mathtt{FABCXFADCYZEADCEADC}$. For this string the location list of motif $x_1 = \mathtt{A{\circ}C}$ is $\mathcal{L}_{x_1} = \{1, 6, 12, 16\}$, and that of motif $x_2 = \mathtt{FA{\circ}C}$ is $\mathcal{L}_{x_2} = \{0, 5\}$. They are both maximal because they loose at least one of their occurrences when extended with solid characters and don't cares to one side or when their don't cares are replaced by solid characters. Instead, motif $x_3 = \mathtt{DC}$ having list $\mathcal{L}_{x_3} = \{7, 13, 17\}$ is not maximal. It occurs in $x_4 = \mathtt{ADC}$, where $\mathcal{L}_{x_4} = \{6, 12, 16\}$, and its occurrences can be obtained from those of x_4 by a displacement of $d = 1$ positions. The basis of the irredundant motifs for s is made up of $x_1 = \mathtt{A{\circ}C}$, $x_2 = \mathtt{FA{\circ}C}$, $x_4 = \mathtt{ADC}$, and $x_5 = \mathtt{EADC}$. The location list of each of them cannot be obtained from the union of any of the others'.

2.2 A quadratic lower bound for irredundant motifs with quorum $q = 2$

In this section, we show the existence of an infinite family of strings s_k ($k \geq 5$) for which there are $\Omega(n^2)$ irredundant motifs in the basis for quorum $q = 2$, where $n = |s_k|$. In this way, we disprove the previous upper bound of $3n$ discussed in Section 1. Each string s_k is the suitable extension of $t_k = \mathtt{A}^k\mathtt{T}\mathtt{A}^k$, where $\mathtt{A}^k$ denotes the letter $\mathtt{A}$ repeated k times (our argument works in general for $z^k w z^k$, where z and w are strings of equal length and not sharing any common character). String t_k has an exponential number of maximal motifs, including those having the form $\mathtt{A}\{\mathtt{A}, \circ\}^{k-2}\mathtt{A}$ with exactly two don't cares. To see why, each such motif x occurs four times in t_k: specifically, two occurrences of x match the first and the last k letters in t_k while each distinct don't care in x matching the letter $\mathtt{T}$ in t_k contributes to one of the two remaining occurrences. Extending x or replacing a don't care with a solid character reduces the number of these occurrences, so x is maximal. However, each such x is covered in t_k. The idea for our proof is to obtain strings s_k by prefixing t_k with $O(|t_k|)$ symbols so that these motifs x become irredundant in s_k. Since there are $\Omega(k^2)$ of them, and $n = |s_k| = \Theta(|t_k|) = \Theta(k)$, this leads to the result.

In order to define the strings s_k on the alphabet $\Sigma = \{\mathtt{A}, \mathtt{T}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_{k-2}\}$, we introduce a few notations. Let $\widetilde{u}$ denote the reversal of u, and let ev_k, od_k, u_k, v_k be the strings thus defined

$$
\begin{array}{llll}
\text{if } k \text{ is even}: & ev_k &=& \mathtt{a}_2\mathtt{a}_4\cdots\mathtt{a}_{k-2}, \\
& od_k &=& \mathtt{a}_1\mathtt{a}_3\cdots\mathtt{a}_{k-3}, \\
& u_k &=& ev_k\,\mathtt{u}\,\widetilde{ev_k}\,\mathtt{vw}\,ev_k, \\
& v_k &=& od_k\,\mathtt{xy}\,\widetilde{od_k}\,\mathtt{z}\,od_k,
\end{array}
\qquad
\begin{array}{llll}
\text{if } k \text{ is odd}: & ev_k &=& \mathtt{a}_2\mathtt{a}_4\cdots\mathtt{a}_{k-3}, \\
& od_k &=& \mathtt{a}_1\mathtt{a}_3\cdots\mathtt{a}_{k-2}, \\
& u_k &=& ev_k\,\mathtt{uv}\,\widetilde{ev_k}\,\mathtt{wx}\,ev_k, \\
& v_k &=& od_k\,\mathtt{y}\,\widetilde{od_k}\,\mathtt{z}\,od_k.
\end{array}
$$

```
2 4 u v 4 2 w x 2 4 1 3 5 y 5 3 1 z 1 3 5 A A A A A A A T A A A A A A A
  4 ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ A A A A ∘ A A
        4 ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ A A A A ∘ A A
                  4 ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ ∘ A A A A ∘ A A
```

Figure 1: Example string s_7, first line ($\mathtt{a}_i$ of the definition is simply denoted by $\mathtt{i}$). The other three lines show the occurrences of motif $x = \mathtt{4} \circ^{19} \mathtt{AAAA} \circ \mathtt{AA}$ in s_7. The letter $\mathtt{4}$ corresponds to position 4 of the don't care symbol in $\mathtt{AAAA} \circ \mathtt{AA}$.

The strings s_k are then defined by $s_k = u_k v_k t_k$ for $k \geq 5$. Figure 1 shows the case for $k = 7$.

Fact 2.4 *The length of $u_k v_k$ is $3k$, and that of s_k is $n = 5k + 1$.*

Proof: Whatever the parity of k, the string $u_k v_k$ contains the six letters $\mathtt{u}$, $\mathtt{v}$, $\mathtt{w}$, $\mathtt{x}$, $\mathtt{y}$, $\mathtt{z}$, two occurrences each of ev_k and od_k, and one occurrence each of $\widetilde{ev_k}$ and $\widetilde{od_k}$. Since od_k and ev_k together contain one occurrence of each letter $\mathtt{a}_1$, $\mathtt{a}_2$, $\ldots$, $\mathtt{a}_{k-2}$, we have $|od_k| + |ev_k| = k - 2$. Moreover, $|\widetilde{ev_k}| = |ev_k|$ and $|\widetilde{od_k}| = |od_k|$, so that $|u_k v_k| = 6 + 3(k-2) = 3k$. This proves the first statement. For the second statement, the total length of s_k follows by observing that $|t_k| = 2k+1$, and so $n = |s_k| = 3k + 2k + 1 = 5k + 1$. $\bowtie$

Proposition 2.5 *For $1 \leq p \leq k-2$, any motif of the form $\mathtt{A}^p \circ \mathtt{A}^{k-p-1}$ with one don't care cannot be maximal in s_k. Also motif $\mathtt{A}^k$ cannot be maximal in s_k.*

Proof: Let w be an arbitrary motif of the form $\mathtt{A}^p \circ \mathtt{A}^{k-p-1}$, with $1 \leq p \leq k-2$. Its location list is $\mathcal{L}_w = \{0, k-p, k+1\} + |u_k v_k| = \{3k, 4k-p, 4k+1\}$ since $|u_k v_k| = 3k$ by Fact 2.4 and w matches the two substrings $\mathtt{A}^k$ of s_k as well as $\mathtt{A}^p\,\mathtt{T}\mathtt{A}^{k-p-1}$. No other occurrences are possible. Let us consider the position, say i, of the leftmost appearance of letter $\mathtt{a}_p$ in s_k (recall that there are three positions on s_k at which letter $\mathtt{a}_p$ occurs). We claim that motif $y = \mathtt{a}_p \circ^{3k-i-1} w$ satisfies $\mathcal{L}_y = \mathcal{L}_w - (3k-i)$. Since w appears in y, it follows that w cannot be maximal in s_k by Definition 2.2 (setting $d = -3k+i$). To see why $\mathcal{L}_w = \mathcal{L}_y + (3k-i)$, it suffices to prove that the distance in s_k between the positions of the two leftmost letters $\mathtt{a}_\mathtt{p}$ is $k-p$ while that of the leftmost and the rightmost $\mathtt{a}_\mathtt{p}$ is $k+1$. The verification is a bit tedious because four cases arise according to the fact that each of k and p can be even or odd. Since the cases are analogous, we detail only two of them, namely, when both k and p are even, and when k is even and p is odd. In the first case, the three occurrences of $\mathtt{a}_p$ are all in u_k. Moreover, the distance between the two leftmost letters $\mathtt{a}_p$ is the length of the substring $\mathtt{a}_p \mathtt{a}_{p+2} \cdots \mathtt{a}_{k-2} \mathtt{u} \mathtt{a}_{k-2} \mathtt{a}_{k-4} \cdots \mathtt{a}_{p+2}$, that is, $2|\mathtt{a}_{p+2} \cdots \mathtt{a}_{k-2}| +$

$2 = 2(k-2-p)/2+2 = k-p$. The distance between the leftmost and rightmost $\mathtt{a}_p$ is the length of $\mathtt{a}_p\mathtt{a}_{p+2}\cdots\mathtt{a}_{k-2}\mathtt{u}\,\widetilde{ev_k}\,\mathtt{vwa}_2\mathtt{a}_4\cdots\mathtt{a}_{p-2}$. This is also the length of $\mathtt{u}\,\widetilde{ev_k}\,\mathtt{vwa}_2\mathtt{a}_4\cdots\mathtt{a}_{p-2}\mathtt{a}_p\mathtt{a}_{p+2}\cdots\mathtt{a}_{k-2} = \mathtt{u}\,\widetilde{ev_k}\,\mathtt{vw}ev_k$, that is, $2(k-2)/2+3 = k+1$ as expected. In the second case where k is even and p is odd, the occurrences of $\mathtt{a}_p$ are all in v_k. Analogously to the first case, the distance between the two leftmost letters $\mathtt{a}_p$ is the length of $\mathtt{a}_p\mathtt{a}_{p+2}\cdots\mathtt{a}_{k-3}\mathtt{xya}_{k-3}\cdots\mathtt{a}_{p+2}$, that is, $2|\mathtt{a}_{p+2}\cdots\mathtt{a}_{k-3}|+3 = 2(k-3-p)/2+3 = k-p$. The distance between the leftmost and the rightmost $\mathtt{a}_p$ is the length of the string $\mathtt{a}_p\mathtt{a}_{p+2}\cdots\mathtt{a}_{k-3}\mathtt{xy}\widetilde{od_k}\mathtt{za}_1\mathtt{a}_3\cdots\mathtt{a}_{p-2}$, which equals $k+1$, the length of $\mathtt{xy}\widetilde{od_k}\mathtt{z}od_k$. The analogous verification of the other two cases yields the fact that w cannot be maximal.

The second part of the lemma for motif $\mathtt{A}^k$ proceeds along the same lines, except that we choose $y = \mathtt{a}_p\circ^{3k-i-1}\mathtt{A}^k$ (note that y is not required to be maximal and that the motifs in the statement are maximal in t_k). $\bowtie$

Proposition 2.6 *Each motif of the form $\mathtt{A}\{\mathtt{A},\circ\}^{k-2}\mathtt{A}$ with exactly two don't cares is irredundant in s_k.*

Proof: Let x be an arbitrary motif of the form $\mathtt{A}\{\mathtt{A},\circ\}^{k-2}\mathtt{A}$ with two don't cares, namely, $x = \mathtt{A}^{p_1}\circ\mathtt{A}^{p_2-p_1-1}\circ\mathtt{A}^{k-p_2-1}$ for $1 \le p_1 < p_2 \le k-2$. To prove that x is an irredundant motif, we first show that x is maximal. Its location list is $\mathcal{L}_x = \{0, k-p_2, k-p_1, k+1\} + 3k$ since $|u_kv_k| = 3k$ by Fact 2.4 and x matches the two substrings $\mathtt{A}^k$ of s_k as well as $\mathtt{A}^{p_1}\mathtt{T}\mathtt{A}^{k-p_1-1}$ and $\mathtt{A}^{p_2}\mathtt{T}\mathtt{A}^{k-p_2-1}$. Any other motif y such that x occurs in y can be obtained by replacing at least one don't care (at position p_1 or p_2) in x with a solid character, but this would cause the removal of position $4k-p_1$ or $4k-p_2$ from $\mathcal{L}_x$. Analogously, extending x to the right by putting a solid character at position $|x|$ or larger would eliminate position $4k+1$ from $\mathcal{L}_x$. Finally, extending x to the left by a solid character would eliminate at least one position from $\mathcal{L}_x$ because no symbol occurs four times in u_kv_k. In conclusion, for any motif y such that x occurs in y, we have $\mathcal{L}_y \ne \mathcal{L}_x + d$ for any integer d, and thus x is a maximal motif by Definition 2.2.

We now prove that x is irredundant according to Definition 2.3. Let us consider an arbitrary set of maximal motifs $y_1, y_2, \ldots, y_h$ such that $\mathcal{L}_x = \cup_{i=1}^{h}\mathcal{L}_{y_i}$. We claim that at least one y_i is of the form $\mathtt{A}\{\mathtt{A},\circ\}^{k-2}\mathtt{A}$. Indeed, there must exist a location list $\mathcal{L}_{y_i}$ containing position $4k+1$ since that position belongs to $\mathcal{L}_x$. This implies that y_i occurs in the suffix $\mathtt{A}^k$ of s_k. It cannot be that $|y_i| < k$ since y_i would occur also in some position $j > 4k+1$ whereas $j \notin \mathcal{L}_x$, so it is impossible. Consequently, y_i is of length k and matches $\mathtt{A}^k$, thus being of the form $\mathtt{A}\{\mathtt{A},\circ\}^{k-2}\mathtt{A}$. We observe that y_i cannot contain zero or one don't cares, as it would not be maximal by Proposition 2.5. Also, y_i cannot contain three or more don't cares, as each distinct don't care would match the letter T in s_k giving $|\mathcal{L}_{y_i}| > |\mathcal{L}_x|$, which is impossible. The only possibility is that y_i contains exactly two don't cares as x does at the same positions because $\mathcal{L}_y \subseteq \mathcal{L}_x$ and they are maximal. It follows that $y_i = x$ proving the proposition. (Note that the motifs in the statement are covered in t_k.) $\bowtie$

Theorem 2.7 *The basis for string s_k contains $\Omega(n^2)$ irredundant motifs, where $n = |s_k|$ and $k \geq 5$.*

Proof: By Proposition 2.6, the number of irredundant motifs in s_k is at least $\binom{k-2}{2} = \Omega(k^2)$, the number of choices of two positions in $\{\mathtt{A}, \circ\}^{k-2}$. Since $|s_k| = 5k + 1$ by Fact 2.4, we get the conclusion. $\bowtie$

3 Tiling Motifs: The Basis and its Properties

3.1 Terminology and properties

In this section we introduce a natural notion of basis for generating all maximal motifs occurring in a string s of length n. Analogously to what was done for maximal motifs in Definition 2.2, we introduce displacements while defining tiling motifs for this purpose.

Definition 3.1 (tiling motif) *A maximal motif x is* tiling *if, for any maximal motifs $y_1, y_2, \ldots, y_k$ and for any integers $d_1, d_2, \ldots, d_k$ such that $\mathcal{L}_x = \cup_{i=1}^{k}(\mathcal{L}_{y_i} + d_i)$, motif x must be one of the y_i's. Vice versa, if all the y_i's are different from x, pattern x is said to be* tiled *by motifs $y_1, y_2, \ldots, y_k$.*

The notion of tiling is in general more selective than that of irredundancy. Going on with our example string $s = \mathtt{FABCXFADCYZEADCEADC}$, we saw in Section 2.1 that motif $x_1 = \mathtt{A{\circ}C}$ is irredundant for s. Now, x_1 is tiled by $x_2 = \mathtt{FA{\circ}C}$ and $x_4 = \mathtt{ADC}$ according to Definition 3.1 since its location list, $\mathcal{L}_{x_1} = \{1, 6, 12, 16\}$, can be obtained from the union of $\mathcal{L}_{x_2} = \{0, 5\}$ and $\mathcal{L}_{x_4} = \{6, 12, 16\}$ with respective displacements $d_2 = 1$ and $d_4 = 0$.

Remark 3.2 A fairly direct consequence of Definition 3.1 is that if x is tiled by $y_1, y_2, \ldots, y_k$ with associated displacements $d_1, d_2, \ldots, d_k$, then x occurs at position d_i in each y_i for $1 \leq i \leq k$. As a consequence, we have that $d_i \geq 0$ in Definition 3.1. Note also that the y_i's in Definition 3.1 are not necessarily distinct and that $k > 1$ for tiled motifs. (It follows from the fact that $\mathcal{L}_x = \mathcal{L}_{y_1} + d_1$ with $x \neq y_1$ would contradict the maximality of both x and y_1.) As a result, a maximal motif x occurring exactly q times in s is tiling as it cannot be tiled by any other motifs because such motifs would have a quorum lower than q.

The *basis of tiling motifs* is the complete set of all tiling motifs for s, and the size of the basis is the number of these motifs. For example, the basis, let us denote it by $\mathcal{B}$, for `FABCXFADCYZEADCEADC` contains `FA∘C`, `EADC`, and `ADC` as tiling motifs. Although Definition 3.1 is derived from that of irredundant motifs given in Definition 2.3, the difference is much more substantial than it may appear. The basis of tiling motifs relies on the fact that tiling motifs are considered as invariant by displacement as for maximality. Consequently, our definition of basis is symmetric, that is, each tiling motif in the basis for the reverse string $\widetilde{s}$ is the reverse of a tiling motif in the basis of s. This follows from the symmetry in Definition 3.1 and from the fact that maximality is also symmetric in Definition 2.2. It is a *sine qua non* condition for having a notion of basis invariant by the left-to-right or right-to-left order of the symbols in s (like the entropy of s), while this property does not hold for the irredundant motifs.

The basis of tiling motifs has further interesting properties. In Section 3.2, we show that our basis is linear for quorum $q = 2$ (that is, its size is at most $n - 1$) and, in Section 3.3, that the total size of the location lists for the tiling motifs is less than $2n$, describing how to find them in $O(n^2 \log n \log |\Sigma|)$ time. In Section 3.4, we discuss some applications such as generating all maximal motifs with the basis and finding motifs with a constraint on the number of don't cares.

3.2 A linear upper bound for the tiling motifs with quorum $q = 2$

Given a string s of length n, let $\mathcal{B}$ denote its basis of tiling motifs for quorum $q = 2$. Although the number of maximal motifs may be exponential and the basis of irredundant motifs may be at least quadratic (see Section 2), we show that the size of $\mathcal{B}$ is always less than n. For this, we introduce an operator $\oplus$ between the symbols of Σ to define the *merges*, which are at the heart of the properties on $\mathcal{B}$. Given two letters $\sigma_1, \sigma_2 \in \Sigma$ with $\sigma_1 \neq \sigma_2$, the operator satisfies $\sigma_1 \oplus \sigma_2 = \circ$ and $\sigma_1 \oplus \sigma_1 = \sigma_1$. The operator applies to any pair of strings $x, y \in \Sigma^*$, so that $u = x \oplus y$ satisfies $u[j] = x[j] \oplus y[j]$ for all integers j. A merge is the motif resulting from applying the operator $\oplus$ to the string s and to its suffix beginning at position k.

Definition 3.3 (Merge) *For $1 \leq k \leq n - 1$, let s_k be the string whose character at position i is $s_k[i] = s[i] \oplus s[i + k]$. If s_k contains at least one solid character, $Merge_k$ denotes the motif obtained by removing all the leading and trailing don't cares in s_k (that is, those appearing before the leftmost solid character and after the rightmost solid character).*

For example, `FABCXFADCYZEADCEADC` has $Merge_4 =$ `EADC`, $Merge_5 =$ `FA∘C`, $Merge_6 = Merge_{10} =$ `ADC` and $Merge_{11} = Merge_{15} =$ `A∘C`. The latter is the only merge that is not a tiling motif.

Lemma 3.4 *If $Merge_k$ exists, it must be a maximal motif.*

Proof: Motif $x = Merge_k$ occurs at positions, say, i and $i+k$ in s. Character $s_k[i]$ is solid by Definitions 2.1 and 3.3. We use the fact that x at occurs at least twice in s for showing that it is maximal. Suppose it is not maximal. By Definition 2.2, there exists y distinct from x such that x occurs in y and $\mathcal{L}_y = \mathcal{L}_x + d$ for some integer d (in this case, $d \leq 0$). Since y is more specific than x displaced by d, there must exist at least one position j with $0 \leq j < |y|$ such that $x[j+d] = \circ$ and $y[j] = \sigma \in \Sigma$. Hence $x[j+d] = s[i+(j+d)] \oplus s[i+k+(j+d)]$ is a don't care, and so $s[(i+d)+j] \neq s[(i+k+d)+j]$. Since $y[j]$ cannot match both of the latter symbols in s, at least one of $i+d$ or $i+k+d$ is not a position of y in s. This contradicts the hypothesis that $\mathcal{L}_y = \mathcal{L}_x + d$ whereas both $i, i+k \in \mathcal{L}_x$. $\bowtie$

Lemma 3.5 *For each tiling motif x in the basis $\mathcal{B}$, there is at least one k for which $Merge_k = x$.*

Proof: As mentioned in Remark 3.2, a maximal motif occurring exactly twice in s is tiling. Hence, if $|\mathcal{L}_x| = 2$, say $\mathcal{L}_x = \{i, j\}$ with $j > i$, then $x = Merge_k$ with $k = j - i$ by the maximality of x and that of the merges by Lemma 3.4. Let us now consider the case where $|\mathcal{L}_x| > 2$. For any pair $i, j \in \mathcal{L}_x$, we denote by u_{ij} the string $s[i\,..\,i+|x|-1] \oplus s[j\,..\,j+|x|-1]$ obtained by applying the operator $\oplus$ to the two substrings of s matching x at positions i and j, respectively. We have $x \preceq u_{ij}$ since x occurs at positions i and j, and $\mathcal{L}_x = \bigcup_{i,j \in \mathcal{L}_x} \mathcal{L}_{u_{ij}}$ since we are taking all pairs of occurrences of x. Letting $k = |j - i|$ for $i, j \in \mathcal{L}_x$, we observe that u_{ij} is a substring of $Merge_k$ occurring at position, say, δ_k in it. Thus,

$$\bigcup_{i,j \in \mathcal{L}_x} \mathcal{L}_{u_{ij}} \;=\; \bigcup_{k=|j-i|\,:\,i,j \in \mathcal{L}_x} (\mathcal{L}_{Merge_k} + \delta_k) \;=\; \mathcal{L}_x.$$

By Definition 3.1, the fact that x is tiling implies that x must be one $Merge_k$, proving the lemma. $\bowtie$

We now state the main property of tiling bases:

Theorem 3.6 (Linearity of the basis) *Given a string s of length n and the quorum $q = 2$, let $\mathcal{M}$ be the set of $Merge_k$, for $1 \leq k \leq n-1$ such that $Merge_k$ exists. The basis $\mathcal{B}$ of tiling motifs for s satisfies $\mathcal{B} \subseteq \mathcal{M}$, and therefore the size of $\mathcal{B}$ is at most $n-1$.*

Proof: This immediately follows from Lemma 3.5. $\bowtie$

A simple consequence of Theorem 3.6 implies a tight bound on the number of tiling motifs for periodic strings. If $s = w^e$ for a string w repeated $e > 1$ times, then s has at most $|w|$ tiling motifs.

Corollary 3.7 *The number of tiling motifs for s is at most p, the smallest period of s.*

The bound in Corollary 3.7 is not valid for irredundant motifs. String s = `ATATATATA` has period $p = 2$ and only one tiling motif `ATATATA`, while its irredundant motifs are `A`, `ATA`, `ATATA` and `ATATATA`.

3.3 A simple algorithm for computing tiling motifs with quorum $q = 2$

We describe how to compute the basis $\mathcal{B}$ for string s when $q = 2$. A brute-force algorithm generating first all maximal motifs of s takes exponential time in the worst case. Theorem 3.6 plays a crucial role in that we first compute the motifs in $\mathcal{M}$ and then discard those being tiled. Since $\mathcal{B} \subseteq \mathcal{M}$, what remains is exactly $\mathcal{B}$. To appreciate this approach, it is worth noting that we are left with the problem of selecting $\mathcal{B}$ from $n - 1$ maximal motifs in $\mathcal{M}$ at most, rather than selecting $\mathcal{B}$ among *all* the maximal motifs in s, which may be exponential in number. Our simple algorithm takes $O(n^2 \log n \log |\Sigma|)$ time and is faster than previous (and more complicated) methods discussed in Section 1.

Step 1. Compute the multiset $\mathcal{M}'$ of merges. Letting $s_k[i]$ be the leftmost solid character of string s_k in Definition 3.3, we define $occ_x = \{i, i + k\}$ to be the positions of the two occurrences of x whose superposition generates $x = Merge_k$. For $k = 1, 2, \ldots, n - 1$, we compute string s_k in $O(n - k)$ time. If s_k contains some solid characters, we compute $x = Merge_k$ and occ_x in the same time complexity. As a result, we compute the multiset $\mathcal{M}'$ of merges in $O(n^2)$ time. Each merge x in $\mathcal{M}'$ is identified by a triplet $\langle i, i + k, |x| \rangle$, from which we can recover the jth symbol of x in constant time by simple arithmetic operations and comparisons.

Step 2. Transform the multiset $\mathcal{M}'$ into the set $\mathcal{M}$ of merges. Since there can be two or more merges in $\mathcal{M}'$ that are identical and correspond to the same merge in $\mathcal{M}$, we put together all identical merges in $\mathcal{M}'$ by radix sorting them. The total cost of this step is dominated by radix sorting, giving $O(n^2 \log |\Sigma|)$ time. As byproduct, we produce the temporary location list $T_x = \bigcup_{x'=x \,:\, x' \in \mathcal{M}'} occ_{x'}$ for each distinct $x \in \mathcal{M}$ thus obtained.

Lemma 3.8 *Each motif $x \in \mathcal{B}$ satisfies $T_x = \mathcal{L}_x$.*

Proof: For a fixed $x \in \mathcal{B}$, the fact that x is equal to at least one merge by Lemma 3.5 implies that T_x is well defined, with $|T_x| \geq 2$. Since $T_x \subseteq \mathcal{L}_x$, let us assume by contradiction that $\mathcal{L}_x - T_x \neq \emptyset$. For each pair $i \in \mathcal{L}_x - T_x$ and $j \in T_x$, let $m_{ij} = Merge_{|j-i|}$, which is maximal by Lemma 3.4. Note that each $m_{ij} \neq x$ by our assumption as otherwise i would belong to T_x; however, x must occur in m_{ij}, say at position δ_{ij} in m_{ij}. Consequently, $\bigcup_{i \in \mathcal{L}_x - T_x, j \in T_x} (\mathcal{L}_{m_{ij}} + \delta_{ij}) = \mathcal{L}_x$ since any occurrence of x is either $i \in \mathcal{L}_x - T_x$ or $j \in T_x$. At this point, we apply Definition 3.1 to the tiling motif x, getting the contradiction that x must be equal to one m_{ij}. $\bowtie$

Notice that the conclusion of Lemma 3.8 does not necessarily hold for the motifs in $\mathcal{M} - \mathcal{B}$. For the previous example string FADABCXFADCYZEADCEADCFADC, one such motif is $x =$ ADC with $\mathcal{L}_x = \{8, 14, 18, 22\}$ while $T_x = \{8, 18\}$.

Step 3. Select $\mathcal{M}^* \subseteq \mathcal{M}$, where $\mathcal{M}^* = \{x \in \mathcal{M} : T_x = \mathcal{L}_x\}$. In order to build $\mathcal{M}^*$, we employ the Fischer-Paterson algorithm based on convolution [7] for string matching with don't cares to compute the whole list of occurrences $\mathcal{L}_x$ for each merge $x \in \mathcal{M}$. Its cost is $O((|x| + n) \log n \log |\Sigma|)$ time for each merge x. Since $|x| < n$ and there are at most $n - 1$ motifs $x \in \mathcal{M}$, we obtain $O(n^2 \log n \log |\Sigma|)$ time to construct all lists $\mathcal{L}_x$. We can compute $\mathcal{M}^*$ by discarding the merges $x \in \mathcal{M}$ such that $T_x \neq \mathcal{L}_x$ in additional $O(n^2)$ time.

Lemma 3.9 *The set $\mathcal{M}^*$ satisfy the conditions $\mathcal{B} \subseteq \mathcal{M}^*$ and $\sum_{x \in \mathcal{M}^*} |\mathcal{L}_x| < 2n$.*

Proof: The first condition follows from the fact that the motifs in $\mathcal{M} - \mathcal{M}^*$ are surely tiled by Lemma 3.8. The second condition follows from the definition of $\mathcal{M}^*$ and from the observation that

$$\sum_{x \in \mathcal{M}^*} |\mathcal{L}_x| = \sum_{x \in \mathcal{M}^*} |T_x| \leq \sum_{x \in \mathcal{M}} |occ_x| < 2n,$$

since $|occ_x| = 2$ (see Step 1) and there are less than n of them. $\bowtie$

The property of $\mathcal{M}^*$ in Lemma 3.9 is crucial in that $\sum_{x \in \mathcal{M}} |\mathcal{L}_x| = \Theta(n^2)$ when many lists contain $\Theta(n)$ entries. For example, $s = \mathtt{A}^n$ has $n-1$ distinct merges, each of the form $x = \mathtt{A}^i$ for $1 \leq i \leq n-1$, and so $|\mathcal{L}_x| = n - i + 1$. This would be a sharp drawback in Step 4 when removing tiled motifs as it may turn into an $\Theta(n^3)$ algorithm. Using $\mathcal{M}^*$ instead, we are guaranteed that $\sum_{x \in \mathcal{M}^*} |\mathcal{L}_x| = O(n)$; hence, we may still have some tiled motifs in $\mathcal{M}^*$, but their total number of occurrences is $O(n)$.

Step 4. Discard the tiled motifs in $\mathcal{M}^*$. We can now check for tiling motifs in $O(n^2)$ time. Given two distinct motifs $x, y \in \mathcal{M}^*$, we want to test whether $\mathcal{L}_x + d \subseteq \mathcal{L}_y$ for some integer d and, in that case, we want to mark the entries in $\mathcal{L}_y$ that are also in $\mathcal{L}_x + d$. At the end of this task, the lists having *all* entries marked are tiled (see Definition 3.1). By removing their corresponding motifs from $\mathcal{M}^*$, we eventually obtain the basis $\mathcal{B}$ by Lemma 3.9. Since the meaningful values of d are equal to the individual entries of $\mathcal{L}_y$, we have only $|\mathcal{L}_y|$ possible values to check. For a *given* value of d, we avoid to merge $\mathcal{L}_x$ and $\mathcal{L}_y$ in $O(|\mathcal{L}_x| + |\mathcal{L}_y|)$ time to perform the test, as it would contribute to a total of $\Theta(n^3)$ time. Instead, we exploit the fact that each list has values ranging from 1 to n, and use a couple of bit-vectors of size n to perform the above check in $O(|\mathcal{L}_x| \times |\mathcal{L}_y|)$ time for *all* values of d. This gives $O(\sum_y \sum_x |\mathcal{L}_x| \times |\mathcal{L}_y|) = O(\sum_y |\mathcal{L}_y| \times \sum_x |\mathcal{L}_x|) = O(n^2)$ by Lemma 3.9.

We therefore detail how to perform the above check with $\mathcal{L}_x$ and $\mathcal{L}_y$ in $O(|\mathcal{L}_x| \times |\mathcal{L}_y|)$ time. We use two bit-vectors V_1 and V_2 initially set to all zeros. Given $y \in \mathcal{M}^*$, we set $V_1[i] = 1$ if $i \in \mathcal{L}_y$. For each $x \in \mathcal{M}^* - \{y\}$ and for each $d \in \mathcal{L}_y$, we then perform the following test. If *all* $j \in \mathcal{L}_x + d$ satisfy $V_1[j] = 1$, we set $V_2[j] = 1$ for all such j. Otherwise, we take the next value of d, or the next motif if there are no more values of d, and we repeat the test. After examining all $x \in \mathcal{M}^* - \{y\}$, we check whether $V_1[i] = V_2[i]$ for all $i \in \mathcal{L}_y$. If so, y is tiled as its list is covered by possibly shifted location lists of other motifs. We then reset the ones in both vectors in $O(|\mathcal{L}_y|)$ time.

Summing up Steps 1–4, the dominant cost is that of Step 3, leading to the following result.

Theorem 3.10 *Given an input string s of length n over the alphabet Σ, the basis of tiling motifs with quorum $q = 2$ can be computed in $O(n^2 \log n \log |\Sigma|)$ time. The total number of motifs in the basis is less than n, and the total number of their occurrences in s is less than $2n$.*

We implemented the algorithm behind Theorem 3.10 and we report here the lessons learned from experiments. Step 1 requires in practice less than the predicted $O(n^2)$ running time. If $p = 1/|\Sigma|$ denotes the probability that two randomly chosen symbols of Σ match in the uniform distribution, the probability of finding the first solid character in a merge follows the binomial distribution, and so the expected number of examined characters in s is $O(1/p) = O(|\Sigma|)$, yielding $O(n|\Sigma|)$ time on the average to locate the *first* (scanning s from the beginning) and the *last* solid character (scanning s form the end backward) in each merge.

A similar approach can be followed in Step 2 for finding the distinct merges. In this case, the merges are first partially sorted using hashing and exploiting the fact that the input is *almost* sorted. Insertion sort is then the best choice and works very efficiently in our experiments (at least 50% faster than quicksort). We do not compute yet the *full* merges at this stage, but we delay this expensive part to a later stage on a small set of buckets that require explicit representation of the merges. As a result, the average case is almost linear. For example, executing Steps 1–2 on chromosome V of *C.elegans* containing over 21 million bases took around 15 minutes on a machine with 512Mb of RAM running Linux on a 1Ghz AMD Athlon processor.

Step 3 is expensive also in practice and the worst case predicted by theory shows up in the experiments. We tried several approaches based on pattern matching with don't cares but still the problem persists. Running this step on sequences much shorter than chromosome V of *C.elegans* took hours. Step 4 is not much of a problem. As a result, an alternative way of selecting $\mathcal{M}^*$ from $\mathcal{M}$ in Step 3 working fast in practice would improve considerably the overall performance.

3.4 Some applications

Checking whether a pattern is a motif. The main property underlying the notion of basis is that it is a generator of all motifs. The generation can be done as follows: first select segments of motifs in the basis that start and end with solid characters, then replace any number of internal solid characters by don't cares. However, since the number of motifs, and even maximal motifs, can be exponential, this is not really meaningful unless this number is small and the time complexity of the algorithm is proportional to the total size of the output. An attempt in this way is done in [14].

The dual problem concerns testing only one pattern. We show how, given a pattern x, it can be tested if x is a motif for string s, that is, if pattern x occurs at least q times in string s. There are two possible ways of performing such a test depending on whether we test directly on the string or on the basis. The answer to the second question relies on iterative applications of the observation made in Remark 3.2 according to which any tiled motif must occur in at least one tiling motif. The next two statements deal with the alternative. In both cases, we assume that integer k comes from the decomposition of pattern x in the form $u_o\circ^{\ell_0}u_1\circ^{\ell_1}\cdots u_{k-1}\circ^{\ell_{k-1}}u_k$ where the subwords u_i contain no don't cares ($u_i \in \Sigma^*$, $0 \leq i \leq k$, and ℓ_j are positive integers, $0 \leq j \leq k-1$).

Proposition 3.11 *The positions of the occurrences of a pattern x in a string of length n can be computed in time $O(k \times n)$.*

Proof: This is a mere application of matching a pattern with don't cares inside a text without don't cares. Using for instance the Fischer and Paterson's algorithm [7] is not necessary. Instead, the positions of the subwords u_i are computed by a multiple string-matching algorithm, such as the Aho-Corasick algorithm [1]. For each position p, a counter associated with position $p - \ell$ on s is incremented, where ℓ is the position of u_i in x (ℓ is the offset of u_i in x). Counters whose value is $k+1$ correspond then to occurrences of x in s. It remains to check if x occurs at least q times in s. The running time is governed by the string-matching algorithm, which is $O(k \times n)$ (equivalent to running k times a linear-time string matching algorithm). $\bowtie$

Proposition 3.12 *Given the basis $\mathcal{B}$ of string s, testing if pattern x is a motif or a maximal motif can be done in $O(k \times b)$ time, where $b = \sum_{y \in \mathcal{B}} |y|$.*

Proof: From Remark 3.2, testing if x is a maximal motif requires only finding if x occurs in an element y of the basis. To do this, we can apply the procedure of the previous proof because don't cares in y should be viewed as extra characters that do not match any letter of Σ. The time complexity of the procedure is thus $O(k \times b)$. Since a non maximal motif occurs in a maximal motif, the same procedure applies to test if x is a general motif. $\bowtie$

As a consequence of Propositions 3.11 and 3.12, we get an upper bound on the time complexity for testing motifs.

Corollary 3.13 *Testing if pattern $u_o \circ^{\ell_0} u_1 \circ^{\ell_1} \cdots u_{k-1} \circ^{\ell_{k-1}} u_k$ is a motif in a string of length n having a basis of total size b can be done in time $O(k \times \min\{b, n\})$.*

Remark 3.14 Inside the procedure described in the proofs of Propositions 3.11 and 3.12, it is also possible to use bit-vector pattern matching methods [3, 11, 21] to compute the occurrences of x. This leads to practically efficient solutions running in time proportional to the length of the string n or the total size of the basis b, in the bit-vector model of machine. This is certainly a method of choice for short patterns.

Finding the longest motif with bounded number of don't cares. We address an interesting question concerning the computation of a longest motif occurring in a string. Given an integer $g \geq 0$, let $LM_g(s)$ be the maximal length of motifs occurring in a string s of length n with quorum $q = 2$, and containing no more than g don't cares. If $g = 0$, the value can be computed in $O(n \log |\Sigma|)$ time with the help of the suffix tree of s (see [6] or [8]). For $g > 0$, we can show that $LM_g(s)$ can be computed in $O(g \times n^2)$ time using the suffix tree augmented (in linear time) to accept longest common ancestor (LCA) queries as follows. For each possible pair (i, j) of positions on s for which $s[i] = s[j]$, we compute the longest common prefix of $s[i \,..\, n-1]$ and $s[j \,..\, n-1]$ in constant time through an LCA query on the suffix tree. If ℓ is the length of the prefix, we get the first part $s[i \,..\, i+\ell-1]\circ$ of a possible longest motif. The second part is found similarly by considering the pair of positions $(i+\ell+1, j+\ell+1)$. The process is iterated g times (or less) and provides a longest motif containing at most g don't cares and occurring at positions i and j. Length $LM_g(s)$ is obtained by taking the maximum length of motifs for all pairs of positions (i, j). This yields the next result.

Proposition 3.15 *Using the suffix tree, $LM_g(s)$ can be computed in $O(g \times n^2)$ time.*

What makes interesting the use of the basis of tiling motifs is that computing $LM_g(s)$ becomes a mere pattern matching exercise because of the strong properties of the basis. Indeed, we claim that any motif yielding $LM_g(s)$ must be the longest segment of a tiling motif in $\mathcal{B}$ that contains no more than g don't care symbols. Therefore its computation can be done in linear time with only two pointers on s, which gives the next result.

Proposition 3.16 *Using the basis $\mathcal{B}$ of tiling motifs, $LM_g(s)$ can be computed in time $O(b)$, where $b = \sum_{y \in \mathcal{B}} |y|$.*

By Proposition 3.16, it is clear that a small basis $\mathcal{B}$ leads to an efficient computation once $\mathcal{B}$ is given. If we have to build $\mathcal{B}$ from scratch, we can observe that no (maximal) motif can give a larger value of $LM_g(s)$ if it does not belong to $\mathcal{B}$. With this observation, we have $O(n^2)$ running time, which always beats the $O(g \times n^2)$ cost of using the suffix tree. In particular, it is interesting to notice that the running time of the algorithm using the basis is independent of parameter g.

4 Beyond $q = 2$: Pseudo-Polynomial Bases for Higher Quorum

We now discuss the general case of quorum $q \geq 2$ for finding the basis of a string of length n. Differently from previous work, we show in Section 4.1 that *no* polynomial-time algorithm can exist for *any* arbitrary value of q in the worst case, both for the basis of irredundant motifs and for the basis of tiling motifs. The size of these bases provably depends *exponentially* on suitable values of $q \geq 2$, that is, we give a lower bound of $\binom{\frac{n-1}{2}-1}{q-1} = \Omega(\frac{1}{2^q}\binom{n-1}{q-1})$. In practice, this size has an exponential growth for increasing values of q up to $O(\log n)$, but larger values of q are theoretically possible in the worst case. Fixing $q = (n-1)/4+1$ in our lower bound, we get a size of $\Omega(2^{(n-1)/4})$ motifs in the bases. On the average $q = O(\log_{|\Sigma|} n)$ by extending the argument after Theorem 3.10, namely, using the fact that on the average the number of simultaneous comparisons to find the first solid character of a merge is $O(|\Sigma|^{q-1})$, which must be less than n.

We show a further property for the basis of tiling motifs in Section 4.2, giving an upper bound of $\binom{n-1}{q-1}$ on its size with a simple proof. Since we can find an algorithm taking time proportional to the square of that size, we can conclude that a polynomial-time algorithm for finding the basis of tiling motifs exists in the worst case if and only if the quorum q satisfies either $q = O(1)$ or $q = n - O(1)$ (the latter condition is hardly meaningful in practice).

4.1 A lower bound of $\binom{\frac{n-1}{2}-1}{q-1}$ on the bases

We show the existence of a family of strings for which there are at least $\binom{\frac{n-1}{2}-1}{q-1}$ tiling motifs for a quorum q. Since a tiling motif is also irredundant, this gives a lower bound for the irredundant motifs to be combined with that in Section 2.2 (note that the lower bound in Section 2.2 still gives $\Omega(n^2)$ for $q \geq 2$). For $q > 2$, this gives a lower bound of $\Omega(\binom{\frac{n-1}{2}-1}{q-1}) = \Omega(\frac{1}{2^q}\binom{n-1}{q-1})$ for the number of both tiling and irredundant motifs.

The strings are this time of the form $t_k = \mathtt{A}^k\mathtt{T}\mathtt{A}^k$ $(k \geq 5)$, without the left extension used in the bound of Section 2.2. The proof proceeds by exhibiting $\binom{k-1}{q-1}$ motifs that are maximal and have each exactly q occurrences, from whence it follows immediately that they are tiling. Indeed Remark 3.2 for tiling motifs holds for any $q \geq 2$. Namely, all maximal motifs that occur exactly q times in a string are tiling.

Proposition 4.1 *For $2 \leq q \leq k$ and $1 \leq p \leq k-q+1$, any motif $\mathtt{A}^p \circ \{\mathtt{A}, \circ\}^{k-p-1} \circ \mathtt{A}^p$ with exactly q don't cares is tiling (and so irredundant) in t_k.*

Proof: Let x be an arbitrary motif $\mathtt{A}^p \circ \{\mathtt{A}, \circ\}^{k-p-1} \circ \mathtt{A}^p$ with $1 \leq p \leq k-q+1$ and q don't cares; namely, $x = \mathtt{A}^{p_1} \circ \mathtt{A}^{p_2-p_1-1} \circ \cdots \circ \mathtt{A}^{p_{q-1}-p_{q-2}-1} \circ \mathtt{A}^{k-p_{q-1}-1} \circ \mathtt{A}^{p_1}$ for $1 \leq p_1 < p_2 < \cdots < p_{q-1} \leq k-1$ and $p = p_1$. We have to prove first that x is a maximal motif according to Definition 2.2. Its length is $k+1+p_1$ and its location list is $\mathcal{L}_x = \{0, k-p_{q-1}, \ldots, k-p_2, k-p_1\}$. Observe that the number of its occurrences is exactly the number of times the don't care symbol appears in x, which is equal to q. A motif y different from x such that x occurs in y can be obtained by replacing the don't care at position p_i with a solid symbol, for $1 \leq i \leq q-1$, but this eliminates $k-p_i$ from the location list of y. Also, y can be obtained by extending x to the right by a solid symbol (at any position $\geq |x|$), but then position $k-p_1$ is not in $\mathcal{L}_y$ because the last symbol in that occurrence of y occupies position $(k-p_1)+|y|-1 \geq (k-p_1)+|x| = (k-p_1)+(k+1+p_1) > |t_k|-1$ in t_k, which is impossible. Analogously, y can be obtained by extending x to the left by a solid symbol (at any position $d < 0$) but position 0 is no longer in $\mathcal{L}_y$. Consequently, for any motif y more specific than x we have $\mathcal{L}_y \neq \mathcal{L}_x + d$, implying that x is maximal. As previously mentioned, x is tiling because it has exactly q occurrences. $\bowtie$

Theorem 4.2 *String t_k has $\binom{\frac{n-1}{2}-1}{q-1} = \Omega(\frac{1}{2^q}\binom{n-1}{q-1})$ tiling (and irredundant) motifs, where $n = |t_k|$ and $k \geq 2$.*

Proof: By Proposition 4.1, the number of tiling or irredundant motifs in t_k is at least $\binom{k-1}{q-1}$, the number of choices of $q-1$ positions on $\mathtt{A}^{k-1}$. Since $n = 2k+1$, we get the conclusion. $\bowtie$

4.2 An upper bound of $\binom{n-1}{q-1}$ tiling motifs

We now prove that $\binom{n-1}{q-1}$ is an upper bound for the size of a basis of tiling motifs for a string s and quorum $q \geq 2$. Let us denote as before such a basis by $\mathcal{B}$. To prove the upper bound, we use again the notion of a merge except that it involves q strings. The operator $\oplus$ between the elements of Σ extends to more than two arguments, so that the result is a don't care if at least two arguments differ. Let k denote now an *array* of $q-1$ positive values $k_1, \ldots, k_{q-1}$ with $1 \leq k_i < k_j \leq n-1$ for all $1 \leq i < j \leq q-1$. A merge is the (non empty) pattern that results from applying the operator $\oplus$ to the string s and to s itself $q-1$ times, at each time shifted by k_i positions to the right for $1 \leq i \leq q-1$.

Definition 4.3 (Merge) *Let s_k denote the string such that its jth character is $s_k[j] = s[j] \oplus s[j + k_1] \oplus \cdots \oplus s[j + k_{q-1}]$ for all integers j. $Merge_k$ is the pattern obtained by removing all the leading and trailing don't cares in s_k (that is, appearing before the leftmost solid character and after the rightmost solid character).*

Lemmas 4.4–4.5 reported below extend Lemmas 3.4–3.5 for $q > 2$.

Lemma 4.4 *If $Merge_k$ exists for quorum q, it must be a maximal motif.*

Proof: Let $x = Merge_k$ denote the (non empty) pattern, and let $s_k[i]$ be its first character, which is solid by Definition 4.3. Since x occurs at least q times in s, at positions $i, i + k_1, \ldots, i + k_{q-1}$, then x is a motif for quorum q. We show that x is maximal. Suppose it were not maximal. By Definition 2.2, there exists y distinct from x such that x occurs in y and $\mathcal{L}_y = \mathcal{L}_x + d$ for some integer d. This implies there exists at least one position j with $0 \leq j < |y|$ such that $y[j] = \sigma \in \Sigma$ and $x[j + d] = \circ$. Since $x[j + d] = s[i + j + d] \oplus s[i + j + k_1 + d] \oplus \cdots \oplus s[i + j + k_{q-1} + d]$, then at least one among $i + d, i + k_1 + d, \ldots, i + k_{q-1} + d$ would not be an occurrence of y, which contradicts the hypothesis that $\mathcal{L}_y = \mathcal{L}_x + d$ (since $i, i + k_1, \ldots, i + k_{q-1} \in \mathcal{L}_x$). $\bowtie$

Lemma 4.5 *For each tiling motif x in the basis $\mathcal{B}$ with quorum q, there is at least one k for which $Merge_k = x$.*

Proof: If $|\mathcal{L}_x| = q = \{i_1, \ldots, i_q\}$ with $i_1 < \cdots < i_q$, then $x = Merge_k$ where k is the array of values $i_2 - i_1, i_3 - i_1, \ldots, i_q - i_1$. Let us now consider the case where $|\mathcal{L}_x| > q$. Given any q-uple $i_1, \ldots, i_q \in \mathcal{L}_x$, let u_k denote $s[i_2 - i_1 \mathinner{..} i_2 - i_1 + |x| - 1] \oplus \cdots \oplus s[i_q - i_1 \mathinner{..} i_q - i_1 + |x| - 1]$, which is a substring of $Merge_k$ introduced in Definition 4.3. We have that $x \preceq u_k$ and $\mathcal{L}_x = \bigcup_{i_1, i_2, \ldots, i_q \in \mathcal{L}_x} \mathcal{L}_{u_k}$. Since each u_k for $i_1, i_2, \ldots, i_q \in \mathcal{L}_x$ is a substring of $Merge_k$, we infer that $\mathcal{L}_x = \bigcup_{i_1, i_2, \ldots, i_q \in \mathcal{L}_x} (\mathcal{L}_{Merge_k} + \delta_k)$

where the δ_k's are non negative integers. By Definition 3.1, if $Merge_k$ were different from x, then x would not be tiling, which is a contradiction. Therefore, at least one $Merge_k$ is x. $\bowtie$

We now state the main property of tiling bases:

Theorem 4.6 (Linearity of the basis) *Given a string s of length n and a quorum $q \geq 2$, let $\mathcal{M}$ be the set of $Merge_k$, for any of the $\binom{n-1}{q-1}$ possible choices of k for which $Merge_k$ exists. The basis $\mathcal{B}$ of tiling motifs for s satisfies $\mathcal{B} \subseteq \mathcal{M}$, and therefore the size of $\mathcal{B}$ is at most $\binom{n-1}{q-1}$.*

Proof: This follows from Lemma 4.4 and Lemma 4.5. $\bowtie$

The tiling motifs in our basis appear in s for a total of $q\binom{n-1}{q-1}$ times at most. A variation of the algorithm given in Section 3.3 gives a pseudo-polynomial time complexity of $O\left(q^2\binom{n-1}{q-1}^2\right)$. When this upper bound is combined with the lower bound of Section 4.1, we obtain that there exists a polynomial-time algorithm for finding the basis if and only if either $q = O(1)$ or $q = n - O(1)$.

5 Conclusions

In this paper we have investigated the problem of discovering motifs with don't cares in strings. For a given string of length n, we demonstrated that the basis of irredundant motifs has size $\Omega(n^2)$ for quorum $q = 2$, the minimal number of times a motif must appear, thus disproving a previous upper bound of $3n$ for any $q \geq 2$ and leaving open the problem of finding a basis in polynomial time for any q. We introduced a more selective basis, named basis of tiling motifs, having size $O(n)$ for $q = 2$ and keeping the ability of generating all motifs. For the general case of $q \geq 2$, we give upper and lower bounds showing that a polynomial-time algorithm for finding the basis of tiling motifs exists in the worst case if and only if the quorum q satisfies $q = O(1)$ or $q = n - O(1)$.

References

[1] A. Aho and M. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[2] A. Apostolico and L. Parida. Compression and the wheel of fortune. In *Data Compression Conference (DCC'2003)*, 2003.

[3] R. Baeza-Yates and G. Gonnet. A new approach to text searching. *Communications of the ACM*, 35:74–82, 1992.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, May 1999.

[5] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):279–305, 1998.

[6] M. Crochemore and W. Rytter. *Jewels of Stringology.* World Scientific Publishing, Hong-Kong, 2002.

[7] M. Fischer and M. Paterson. String matching and other products. In R. Karp, editor, *SIAM AMS Complexity of Computation*, pages 113–125, 1974.

[8] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge University Press, Cambridge, 1997.

[9] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39:58–64, 1999.

[10] H. Mannila. Local and global methods in data mining: basic techniques and open problems. In P. W. et al., editor, *29th International Colloquium on Automata, Languages, and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 57–68. Springer-Verlag, 2002.

[11] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3):395–415, 1999.

[12] L. Parida, A. Floratos, and I. Rigoutsos. Musca: An algorithm for constrained alignment of multiple data sequences. In *Proceedings 9th Workshop on Genome Informatics*, 1998.

[13] L. Parida, I. Rigoutsos, A. Floratos, D. Platt, and Y. Gao. Pattern Discovery on Character Sets and Real-valued Data: Linear Bound on Irredundant Motifs and Efficient Polynomial Time Algorithm. In *SIAM Symposium on Discrete Algorithms (SODA)*, 2000.

[14] L. Parida, I. Rigoutsos, and D. Platt. An output-sensitive flexible pattern discovery algorithm. In A. Amir and G. Landau, editors, *Combinatorial Pattern Matching, Lecture Notes in Computer Science 2089*, pages 131–142, 2001.

[15] G. Pavesi, G. Mauri, and G. Pesole. Methods for pattern discovery in unaligned biological sequences. *Briefings in Bioinformatics*, 2(4):417–430, 2001.

[16] J. Pelfrêne, S. Abdeddaïm, and J. Alexandre. Un algorithme d'indexation de motifs approchés. In *Journée Ouvertes Biologie Informatique Mathématiques (JOBIM)*, pages 263–264, 2002.

[17] N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot. A basis for repeated motifs in pattern discovery and text mining. Technical Report IGM 2002-10, Institut Gaspard-Monge, University of Marne-la-Vallée, July 2002.

[18] M.-F. Sagot and A. Viari. A double combinatorial approach to discovering patterns in biological sequences. In D. Hirschberg and G. Myers, editors, *Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 186–208. Springer-Verlag, 1996.

[19] M.-F. Sagot, A. Viari, and H. Soldano. Multiple comparison: a peptide matching approach. *Theoret. Comput. Sci.*, 180:115–137, 1997.

[20] A. Vanet, L. Marsan, and M.-F. Sagot. Promoter sequences and algorithmical methods for identifying them. *Res. Microbiol*, 150(1):1–21, 1999.

[21] S. Wu and U. Manber. Path-matching problems. *Algorithmica*, 8(2):89–101, 1992.

Appendix

A On the upper bound of $3n$ irredundant motifs

This section[1] presents a counterexample to the argument given in [13] showing that the basis defined with maximal and irredundant motifs contains less than $3n$ motifs *for any* $q \geq 2$. We use the string of the form `aaxaaaxaaaax...` as running example, in which the number of `a`'s between two consecutive `x`'s increases by 1 in each run. In particular, we focus on the string $s =$ `aaxaaaxaaaaxaaaaaxaaaaaa` of length $n = 24$ whose basis contains $93 > 3n$ motifs for $q = 2$. This situation is even worse when extending s to larger values of n; here, we describe the shortest value of n such that it occurs. In [13] the upper bound of $3n$ is given through a charging argument like in amortized analysis. In its essence, it establishes that we can assign unique positions on the input string s to motifs, and, when this is not possible, we have a situation that crucially relies on [13, Lemma 2], reported below:

Lemma A.1 *Consider an instance each of a set of motifs $m_1, m_2, \ldots, m_\ell$ such that (1) for that instance of m_i, the starting position $x \in \mathcal{L}_{m_i}$ is marked ON w.r.t. m_i, $1 \leq i \leq \ell$, and, (2) for every motif m_i there exists a motif m_j, $j \neq i$, $1 \leq i, j \leq \ell$, such that the two instances are 2-connected, then there exist* distinct *positions $j_1, j_2, \ldots, j_\ell$ on the input string s,[2] with the corresponding positions $j'_1, j'_2, \ldots, j'_\ell$ such that $m_1[j'_1], m_2[j'_2], \ldots, m_\ell[j'_\ell]$ are solid characters.*

We recall from [13] that a position $x \in \mathcal{L}_{m_1}, \mathcal{L}_{m_2}, \ldots, \mathcal{L}_{m_\ell}$ is marked ON with respect to motif m_i ($1 \leq i \leq \ell$) if there exists no m_j ($j \neq i$) such that $\mathcal{L}_{m_j} \subset \mathcal{L}_{m_i}$ holds. Moreover, two instances of motifs are 2-connected if the solid characters of their corresponding motifs occupy at least two common positions on s. The charging argument for the upper bound of $3n$ exploits Lemma A.1 when a position x is ON for more than one motif. In this case we cannot directly assign position x

[1]Only here we follow the notation in [13] by which the positions on a string start from 1 (whereas they start from 0 in our paper) and use x to denote one such position, m_i to denote a motif and a mere dot to denote the don't care.

[2]Note that these positions are inside the instances mentioned in the lemma.

to each of the (instances of) motifs $m_1, m_2, \ldots, m_\ell$, but we can take one instance per motif under the conditions mentioned in Lemma A.1. The latter shows that we can assign a distinct position (on s) inside each instance so that each position corresponds to a solid character of one distinct motif m_i. As a result, the charge for x in $m_1, m_2, \ldots, m_\ell$ is transferred to these distinct positions $j_1, j_2, \ldots, j_\ell$. Unfortunately, this assignment is not possible with our string s, thus contradicting Lemma A.1. For instance, consider position $x = 5$, which is ON for the following set of motifs, whose instances at x are pairwise 2-connected and occur in $s[x \mathinner{..} x+12] = \texttt{aaxaaaaxaaaaa}$:

```
002: a....aa.aa          5 8 11 14 15
012: a..a.a..aa.a        5 6 8 11
015: a..a.aa..a.aa       5 8 10 11
016: a..aa...aaaa        5 6 11 13
018: a..aa.a..aaa        5 10 11 13
019: a..aa.a.aa.a        2 5 11 13
061: aa...aa.a           5 8 9 14 15 16
066: aa..a.a.a           5 9 13 15 16
072: aa..aaa..aaaa       4 5 10
079: aa.a.aa             5 8 10 14 16
080: aa.aa               1 5 10 13 16 19 20
082: aa.aa.a             5 10 13 16
```

There are only 11 occurrences of symbol $\texttt{a}$ in $s[x \mathinner{..} x+12]$, while there are $\ell = 12$ motifs m_1, $m_2, \ldots, m_\ell$. Consequently, we cannot assign *distinct* positions to their instances in correspondence with solid characters in $m_1, m_2, \ldots, m_\ell$. We list below all the motifs in the basis for s:

```
001: a....a...a.a...aa   4 5 6 8
002: a....aa.aa          5 8 11 14 15
003: a...a...a.aa        5 6 9 11 13
004: a...a.a..a.aa..a.aaa 2 4 5
005: a...a.aa.a.aaa      2 4 10
006: a...aa...aaa...aaaa 4 5 6
007: a...aaa...aaaa      4 9 10 11
008: a...aaa..aaaa       4 5 10 11
009: a..a..a..aa         5 10 11 13 14
010: a..a..a.aa.aa.aa    2 5 8
011: a..a..aa.a.aaaa     2 8 10
012: a..a.a..aa.a        5 6 8 11
013: a..a.aa             5 8 10 11 14 16 17
014: a..a.aa..a          5 8 10 11 14
015: a..a.aa..a.aa       5 8 10 11
```

016: a..aa...aaaa	5 6 11 13
017: a..aa..aaa..aaaa..aaaaa	1 2
018: a..aa.a..aaa	5 10 11 13
019: a..aa.a.aa.a	2 5 11 13
020: a..aa.aa.a.a	2 10 13
021: a..aa.aaa	2 13 16
022: a..aaa..aaa.aa	1 11
023: a..aaa.a	1 6 10 16 17
024: a..aaaxaaaa..aaaa.a	1 6
025: a.a....a.aa	4 6 13 14
026: a.a...aaa..aaaaa	2 8 9
027: a.a..a...aa	4 6 11 14
028: a.a..a..a..a.a	6 8 9 11
029: a.a..a..aa	6 8 11 14 15
030: a.a..a.a...a.a.a	4 6 8 9
031: a.a..a.a.aa	4 6 14
032: a.a..aa..a	4 8 11 14 15
033: a.a..aa.a	8 9 11 14 15
034: a.a.a..a..aa	4 6 9 13
035: a.a.a..a.a.a.a.a.aa	2 4 6
036: a.a.a.a..a.aaa	2 4 11
037: a.a.a.aa...aaa.a	2 4 9
038: a.a.a.aa.a.aaa.a.aaaa	2 4
039: a.a.a.aaa..a	2 9 13
040: a.a.a.aaaa	2 13 15
041: a.a.aa....aa.a	4 6 9 11
042: a.a.aa.a..aa.a.a	4 6 9
043: a.a.aaa	4 9 11 15 17
044: a.a.aaaa	4 9 15 17
045: a.aa	2 6 8 11 13 14 17 19 20 21
046: a.aa...aaa.a.aaa	2 6 8
047: a.aa..a.aa.aaa	2 8 11
048: a.aa..aaaa	2 8 13 14
049: a.aa..aaaaxaaaaa	2 8
050: a.aa.a	6 8 11 14 17 19
051: a.aa.a..aa	6 8 11 14
052: a.aaa	2 6 11 13 17 19 20
053: a.aaa..aaa.a	2 6 13
054: a.aaa.a.aa.a	2 11 13
055: a.aaa.aa	2 13 17
056: a.aaaxaaaa.a	2 13

057: aa	1 4 5 8 9 10 13 14 15 16 19 20 21 22 23
058: aa....aa.aa	4 10 13 14
059: aa....aaa	8 9 13 14 15 16
060: aa...aa..a.aa	4 5 8 10
061: aa...aa.a	5 8 9 14 15 16
062: aa...aaa...aaa	4 8 9 10
063: aa...aaa.aa	4 10 14
064: aa...aaaa	8 9 14 15 16
065: aa..a.a..aaa	4 5 10 13
066: aa..a.a.a	5 9 13 15 16
067: aa..a.aa..aa	4 9 10 13
068: aa..aa.a.aa.aa.aa.aaa	1 4
069: aa..aa.aa.a.aaaa	1 9
070: aa..aa.aaa	1 15
071: aa..aaa..a	4 5 10 15
072: aa..aaa..aaaa	4 5 10
073: aa..aaaa	4 9 10 15 16
074: aa..aaaa..aaaa	4 9 10
075: aa.a	1 5 8 10 13 14 16 19 20 21
076: aa.a..aa	8 10 13 14 16
077: aa.a.a.aaa..aaaaa	1 8
078: aa.a.a.aaaa	1 14
079: aa.a.aa	5 8 10 14 16
080: aa.aa	1 5 10 13 16 19 20
081: aa.aa..aaaa	1 13
082: aa.aa.a	5 10 13 16
083: aa.aaa	1 5 10 16 19
084: aaa	4 8 9 13 14 15 19 20 21 22
085: aaa...aa.a	4 8 13 14 15
086: aaa..aaa.a	4 8 14 15
087: aaa.a	4 9 13 15 19 20
088: aaa.a.aa	4 9 13 15
089: aaa.aa	4 9 15 19
090: aaaxaaaa	4 9 15
091: aaxaaa..aaa.a.aaa.aa	1 5
092: aaxaaa.a.aa.aaa	1 10
093: aaxaaa.aa	1 16