A Basis for Repeated Motifs in Pattern Discovery and Text Mining*

N. Pisanti † M. Crochemore ‡ R. Grossi § M.-F. Sagot ¶

July 12, 2002

Abstract

We present a new notion of basis that is able to generate the repeated motifs (possibly exponential in number) that appear at least twice with don't care symbols in a string of length n over an alphabet Σ. Our basis has some interesting features such as being (a) a subset of the previously defined bases, (b) truly linear as its motifs are less than n in number and appear in the string for a total of $2n$ times at most; (c) symmetric as the basis of the reversed string is the reverse of the basis; (d) computable in polynomial time, namely, in $O(n^2 \log n \log |\Sigma|)$ time. In addition, several other computational questions related to the use of our basis are discussed. Our notion provides the best-known tradeoff between the size of the basis and the complexity of its construction.

1 Introduction

Identifying repeats in a string is a common task in several areas, notably computational biology, data mining, system security, etc. Just to name a few examples, biologists aiming at understanding some properties of biological sequences may look for repetitions with some degree of approximation [3]; designers of data mining systems have realized that repetitions (called frequent itemsets) are at the heart of discovering the so-called "association rules"

---

*Supported by VINCI French-Italian University program.

†Work of this author is partially supported by ERCIM *fellowship programme*. Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy and INRIA Rhône Alpes, France; `pisanti@di.unipi.it`

‡Work by this author is partially supported by CNRS action AlBio, NATO Science Programme grant PST.CLG.977017, and Wellcome Trust Foundation. Institut Gaspard-Monge, University of Marne-la-Vallée, 77454 Marne-la-Vallée CEDEX 2, France and King's College London; `http://www-igm.univ-mlv.fr/~mac`

§Work by this author is partially supported by the United Nations Educational, Scientific and Cultural Organization (UNESCO) under contract UVO-ROSTE 875.631.9 and by the Italian Ministry of Research and Education. Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy; part of the work of this authour is done while visiting the Institut Gaspard-Monge, University of Marne-la-Vallée, France; `grossi@di.unipi.it`

¶Work by this author is partially supported by CNRS-INRIA-INRA-INSERM action BioInformatique and Wellcome Trust Foundation. Inria Rhône-Alpes, Laboratoire de Biométrie et Biologie Évolutive, Université Claude Bernard, 69622 Villeurbanne cedex, France; `Marie-France.Sagot@inria.fr`

which are considered the basic primitives in these systems [5] (interestingly, several of their problems can be formulated as pattern discovery in strings with large alphabets). A plethora of different formulations in the known literature face the problem of detecting repetitions in several contexts and formulate the problem in different ways depending on the specific applications, thus revealing its algorithmic relevance.

In this paper, we focus on repetitive patterns with *don't cares*, where a don't care is a special symbol that matches anything. For example, pattern B·T matches both BIT and BUT, where the symbol '·' denotes the don't care. Note that, contrarily to string matching with don't cares, we are not given a pattern to search for in the text. Here, given a string of length n over an alphabet Σ, we want identify all possible patterns with don't cares that appears in s at least twice. Unfortunately, this is hard as their number can be exponential. Several heuristics, based on relevant properties in biology or notions of quorum and confidence in text mining, try to reduce the number of interesting patterns to make feasible any further processing of them. In this context, for a fixed parameter $q > 1$ called *quorum*, we say that a pattern is a *motif* if it appears *at least* q times in s. Among the many proposed methods to select motifs, we single out those based on the notion of *maximality* or *specificity*. Informally, a motif is maximal if cannot be extended to the left or to the right by adding further symbols or by replacing any of its don't care symbols by an alphabet letter, without losing any occurrences. For example, motif M · ·T · E is maximal in COMMITTEE.

Unfortunately, the notion of maximality is not sufficient to bound the number of interesting motifs [7]. A significant step in this field has been the introduction of the notion of *basis*.

Informally speaking, a basis is a set of motifs that can generate all the maximal motifs by simple mechanical rules. The paper by Parida *et al.* [7] proposes a mathematical way of expressing this notion in a novel way. It relies on the idea that a basis of motifs can be defined in the algebraic sense of the term. The maximal motifs in the basis defined in [7] are *irredundant*. Representing each occurrence of a motif by its starting position in the string s, a maximal motif is irredundant if its set of occurrences is *not* the union of the sets of occurrences of other maximal motifs. Parida *et al.* prove that the number of motifs in the basis is at most $3n$ and that the basis can be computed in $O(n^3 \log n)$ time notwithstanding the possibly exponential number of maximal motifs that are candidates for the basis (an algorithm in [8, Sect.4.1], which is apparently the same, is claimed to require $O(n^4 \log n)$ time).

Motivated by an attempt to have insight into the mathematical background behind the ideas introduced in [7], we investigate a new definition of basis. Among the initial motivations, we found several examples in which the size of the basis defined in [7] is not bound by $3n$. We actually could not find any reasonable constant c giving an upper bound of cn in our examples. Furthermore, we were unable to grasp all the details of the algorithms presented in [7] so as to run them correctly on our examples. For instance, the algorithm for building the basis includes some redundant motifs in the basis, and discard some irredundant motifs [10]. A following paper [8] presented the same difficulties to us, therefore leaving many open questions. For example, there is no dependency on the quorum q in the time complexity, while we could observe a non-constant growth of the basis for increasing values of q up to $O(\log n)$. Another point is that the basis of the string in reversed order s^R and the basis of s are apparently unrelated. In a certain sense, this is a bit surprising. If we consider other properties of s, such as its entropy (related to repetitions), we expect that the entropy of s and that of s^R are related (actually, they are the same).

In this paper, we follow up the seminal idea of Parida *et al.* and propose an alternative notion of basis with quorum $q = 2$ that is able to generate all maximal motifs. Our basis has some interesting features such as being (a) a subset of the basis defined in [7]; (b) truly linear as its motifs are less than n in number and also appear in s for a total of $2n$ times at most; (c) symmetric as the basis of the reversed string s^R is made up by the reversed motifs in the basis of s; (d) computable in polynomial time, namely, in $O(n^2 \log n \log |\Sigma|)$ time. The motifs in our basis, called *tiling motifs*, are maximal motifs such that their sets of occurrences are not the union of the sets of *shifted* occurrences of maximal motifs. This notion is more stringent than irredundancy, as a tiling motif is irredundant but the vice versa is not necessarily true.

Very recently, Pelfrene *et al.* [9] have introduced in a two-page poster another notion of irredundancy, claiming by examples that the corresponding basis for quorum $q = 2$ has linear size and can be computed in $O(n^3)$ time. As far as we know, our basis is also a subset of theirs (which is not symmetric), and the total number of occurrences of the motifs in their basis can be $\Theta(n^2)$. They leave as an open problem the case for quorum $q > 2$.

Example Consider sequence $\mathtt{AAAAA} \cdots \mathtt{AAAA} = \mathtt{A}^n$. Both basis in [7] and [9] contain $n - 1$ motifs, namely $\mathtt{A}^i$ with list of occurrences $\{0, 1, \ldots, n - i\}$ for all i's between 1 and $n - 1$. Therefore the sum of the size of all occurrence lists of motifs in their basis is in $O(n^2)$. Our basis contains only the motif $\mathtt{A}^{n-1}$ with occurrence list $\{0, 1\}$, and in this data we have all the information we need to generate the other motifs. Another example showing the differences with previous work is in Section 2.

In our paper, we also indicate a polynomial algorithm for calculating the basis in the case where the motifs must appear at least q times for $q > 2$. Finally, we show that given any pattern, we can determine in $O(k \times \min(n, b))$ time whether it appears repeated in s where k is the number of blocks of don't cares in the pattern and b the total length of motifs in the basis.

The paper is organized as follows. Section 2 presents the new definition of basis of motifs together with basic properties. Section 3 describes structural properties of bases and provides important elements related to the complexity of subsequent algorithms. In Section 4, the algorithm to compute a basis of motifs for a string is described and analyzed. Finally, section 5 is concerned with the use of bases and related algorithms.

2 Basis of Motifs

We consider strings that are finite sequences of letters drawn from an alphabet Σ. Elements of Σ are also named *solid characters*. An additional letter not in Σ is considered, denoted by $\cdot$, and called the *don't care symbol* (it matches any other letter). The length of a string t (in Σ^* or $(\Sigma \cup \{\cdot\})^*$) is denoted by $|t|$ and the letter at position i in t by $t[i]$. We therefore have $t = t[0]t[1] \cdots t[|t| - 1]$.

Definition 1 (pattern) *A* pattern *is a string in $\Sigma \cup \Sigma(\Sigma \cup \{\cdot\})^*\Sigma$, that is, a string on the* *alphabet $\Sigma \cup \{\cdot\}$ that starts and ends with a solid character.*

Definition 2 (specifity) *The* specifity relation *denoted by $\preceq$ is defined on $\Sigma \cup \{\cdot\}$ as follows: for $\sigma_1, \sigma_2 \in \Sigma \cup \{\cdot\}$, $\sigma_1 \preceq \sigma_2$ iff $\sigma_1 = \sigma_2$ or $\sigma_1 = \cdot$.*

We view strings as if they were padded to the left and to the right with an infinite number of don't cares. In other words, for a string s, we consider that $t[j] = \cdot$ for any integer j such that $j < 0$ or $j \geq |t|$. The relation $\preceq$ extends then to strings: for $u, v \in (\Sigma \cup \{\cdot\})^*$, $u \preceq v$ iff $u[j] \preceq v[j]$ for any integer j. We say that u is *less specific* than v when $u \preceq v$.

Definition 3 (occurrence) *For $u, v \in (\Sigma \cup \{\cdot\})^*$, we say that u* occurs at position ℓ in v *if $u[j] \preceq v[\ell + j]$, for $0 \leq j \leq |u| - 1$.*

Let $s \in \Sigma^*$ be the fixed string where repeated motifs are searched for, and let $|s| = n$.

Definition 4 (motif) *For a pattern x, the* location list $\mathcal{L}_x$ *contains the positions of all the occurrences of x in s. Given a special parameter $q \geq 2$, called* quorum, *we say that pattern x is a* motif *(according to s and q) if $|\mathcal{L}_x| \geq q$.*

Given a string s and a quorum q, we are interested in finding all motifs of s, that is all patterns that appear at least q times in s. The problem of finding such motifs is computationally difficult. The difficulty is basically due to the possible exponential size of the output. In fact, an exponential number of motifs can be found. We therefore define two concepts that allow to reduce this quantity. Since motifs appearing only once are not interesting, from now on we assume $q = 2$.

Given a set $\mathcal{L}$ of integers and an integer i, the *i-shift of $\mathcal{L}$*, denoted by $\mathcal{L} + i$, is the set $\{x + i \mid x \in \mathcal{L}\}$.

Definition 5 (maximality) *A pattern x is* maximal *if, for any pattern y and integer d such that $\mathcal{L}_y = \mathcal{L}_x + d$, we have $d \geq 0$ and y occurs in x at position d.*

Intuitively, a motif x is maximal if no other motif y is more specific than it (*i.e.* y has a solid character at a position where x has a don't care, or x is a segment of y), *and* simultaneously has exactly the same occurrences as x in the input string (positions are possibly shifted by some integer).

Example For example, let $s =$ `FABCXFADCYZEADCEADC`. The occurrence list of motif $x_1 =$ `A·C` is $\mathcal{L}_1 = \{1, 6, 12, 16\}$, and that of motif $x_2 =$ `FA·C` is $\mathcal{L}_{x_2} = \{0, 5\}$. They are maximal because if they are extended or their don't cares are replaced by a solid character, they lose at least one of their occurrences. On the contrary, motif $x_3 =$ `DC` has list $\mathcal{L}_{x_3} = \{7, 13, 17\}$ and is not maximal since it occurs in $x_4 =$ `ADC` that has the same number of occurrences ($\mathcal{L}_{x_4} = \{6, 12, 16\}$) at positions shifted by 1. Indeed, x_4 is maximal.

Definition 5 is an equivalent formulation of the one given in [7] and [9]. It is known from [7] that there can still be an exponential number of maximal motifs in a string. Therefore, a further requirement should be made in order to restrict to a polynomial number of interesting motifs.

Definition 6 (tiling motif) *A maximal motif x is a* tiling motif *if, for any maximal motifs y_1, y_2, $\ldots$, y_k and 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 is one of the y_i's. When, instead, all y_i's are different from x, pattern x is said to be a* tiled motif. *We also say that motifs y_i, y_2, $\ldots$, y_k* tile *x.*

According to the definition, if x is tiled by the motifs y_1, y_2, $\ldots$, y_k (given integers d_1, d_2, $\ldots$, d_k), then $k > 1$. Indeed, the equality $\mathcal{L}_x = \mathcal{L}_{y_1} + d_1$ with $x \neq y_1$ would contradict the maximality of both x and y_1.

Among the maximal motifs of $s =$ `FABCXFADCYZEADCEADC`, motif $x_1 =$ `A·C` is tiled since its occurrence list, $\mathcal{L}_{x_1} = \{1, 6, 12, 16\}$, can be recovered from the lists of $x_4 =$ `ADC` and $x_2 =$ `FA·C`, which are respectively $\mathcal{L}_{x_4} = \{6, 12, 16\}$ and $\mathcal{L}_{x_2} = \{0, 5\}$. The latter list should be shifted by one position.

The notion of tiling motifs with respect to maximality relies on the fact that motifs are considered as invariant by shift. Note that the y_i's in Definition 6 are not necessarily distinct. We suggest the set of tiling motifs as a notion of *basis* for the complete set of all (maximal) motifs. In fact, we show (Section 5) that such a set of motifs is enough to recover all motifs. The size of the basis is not greater than the length of the input string (Section 3). Moreover, the set of tiling motifs can be computed in polynomial time with respect to the length of the input string (Section 4).

In the following lemma, we state a characterization of the maximal motifs which contribute to make another maximal motif a tiled motif.

Lemma 1 *Let x be a maximal motif tiled by maximal motifs $y_1, y_2, \ldots, y_k$ with corresponding integers d_1, d_2, $\ldots$, d_k ($k > 1$). Then for each integer i, $1 \leq i \leq k$, x occurs at position d_i in y_i.*

Proof: By definition, we have that $\mathcal{L}_x = \cup_{i=1}^{k}(\mathcal{L}_{y_i} + d_i)$ and x is different from all other y_i's. We therefore have $(\mathcal{L}_{y_i} + d_i) \subset \mathcal{L}_x$ for all i's. We now show that for all integers j, $0 \leq j \leq |x| - 1$ such that $x[j]$ is a solid character $\sigma \in \Sigma$, we have $y_i[j + d_i] = \sigma$ for all y_i's for integers i between 1 and k.

Let us assume by absurd that this is not the case; that is, there exist integers i and j such that $y_i[j + d_i] = \cdot$ or $y_i[j + d_i] = \gamma$ with $\gamma \neq \sigma$.

Since $\mathcal{L}_x = \cup_{i=1}^{k}(\mathcal{L}_{y_i} + d_i)$, then for all $p \in \mathcal{L}_x \cap \mathcal{L}_{y_i}$, both $x[j]$ and $y_i[j + d_i]$ align with $s[p + j]$. Since $x[j] = \sigma$, then, for all $p \in \mathcal{L}_x$ and, in particular, for all $p \in \mathcal{L}_x \cap \mathcal{L}_{y_i}$, $s[p + j] = \sigma$. If $y_i[j + d_i]$ were a $\cdot$, this would contradict the maximality of y_i. If $y_i[j + d_i]$ were equal to γ with $\gamma \neq \sigma$, $y_i[j + d_i]$ would not align with $s[p + j]$ and this would contradict the fact that $p \in \mathcal{L}_{y_i}$. $\bowtie$

Observe that our definition of tiling motifs is symmetric: each element of the basis of the reverse of s is the reverse of an element of the basis of s (this follows directly from the definition of tiling motifs and from the fact that maximality is also symmetric). This is indeed a *sine qua non* condition to have a correct notion of basis, but the property is not true for the definitions in [7] and [9].

Example Given the string $s = \mathtt{G\underline{ACA}XG\underline{ACA}YYT\underline{ACA}ZZZT\underline{ACA}CBUUUAGA}$, motif ACA is in both the basis of [7] and [9] (and not in ours). In the reverse string $s^R = \mathtt{AGAUUUBC\underline{ACA}TZZZ\underline{ACA}TYY\underline{ACA}GX\underline{ACA}G}$, motif ACA (which is the reverse of itself as it is a palindrome) does not belong to the two bases above anymore because it is eliminated by ACAT and ACAG.

3 Linearity of the Basis

Definition 7 *We define an operator $\oplus$ between elements of Σ in the following way: given $\sigma, \gamma \in \Sigma$ with $\sigma \neq \gamma$, $\sigma \oplus \sigma = \sigma$, and $\sigma \oplus \gamma = \cdot$. This operator is extended to two strings $x, y \in \Sigma^*$ with $|x| = |y|$ in the obvious way: $u = x \oplus y$ is such that $u[i] = x[i] \oplus y[i]$ for all integers i between 0 and $|x| - 1$.*

Definition 8 *Let $s_k[i] = s[i] \oplus s[i+k]$ for $0 \leq i \leq |s| - k - 1$. We denote by $Merge_k$ the string obtained from s_k by removing all don't cares to the left of the leftmost solid character and to the right of the rightmost solid character.*

In other words, given a string $s \in \Sigma^*$ (recall that strings are padded to the left and right with an infinite number of don't cares), $Merge_k$ is the result of the application of the $\oplus$ operation to the string s with s itself shifted k positions to the right.

Let us consider again the string $s =$ FABCXFADCYZEADCEADC. The non empty *Merge*s are $Merge_4 =$ EADC, $Merge_5 =$ FA·C, $Merge_6 = Merge_{10} =$ ADC and $Merge_{11} = Merge_{15} =$ A·C.

Lemma 2 *Any $Merge_k$ of length at least one is a maximal motif.*

Proof: Let $x = Merge_k$ be non empty. It is a motif since it has at least two occurrences, namely $i, i+k \in \mathcal{L}_m$ for the first integer i between 0 and $|s| - k$ such that $s[i] \oplus s[i+k] \neq \cdot$. Motif x may have other occurrences. Suppose it were not maximal. This would mean that x could be extended in length or one of its don't cares replaced with a letter without losing any occurrence. Let us suppose that one of the don't cares in x can be replaced with a letter or that x can be extended to the right (the reasoning for an extension to the left would be the same). Let j be one of the positions ($0 \leq j \leq |s| - k - i$) such that $x[j] = \sigma$ for $\sigma \in \Sigma$ while $s_k[i+j] = s[i+j] \oplus s[i+k+j] = \cdot$. Then either i or $i+k$ (or both) would not be an occurrence of x any longer, which is a contradiction. $\bowtie$

Lemma 3 *Each tiling motif of a string s is equal to $Merge_k$ for some integer k between 1 and $n-1$.*

Proof: Let x be a tiling motif for a string s. If $|\mathcal{L}_x| = q = 2$ with $\mathcal{L}_x = \{i, j\}$ and $j > i$, then obviously $x = Merge_{j-i}$.

If $|\mathcal{L}_x| > 2$, let $u_i = s[i]s[i+1]\ldots s[i+|x|-1]$ be the occurrence of x at position i of s for $i \in \mathcal{L}_x$. Given $i, j \in \mathcal{L}_x$, let $u_{ij} = u_i \oplus u_j$. For all $i, j \in \mathcal{L}_x$, we have that $x \preceq u_{ij}$ and $\mathcal{L}_{u_{ij}} \subseteq \mathcal{L}_x$. Furthermore, $\mathcal{L}_x = \bigcup_{i,j \in \mathcal{L}_x} \mathcal{L}_{u_{ij}}$.

We now show that for at least a pair (i, j) with $i \neq j$, we have $x = u_{ij}$. The proof is by absurd. Suppose that for each $i, j \in \mathcal{L}_x$ we have $x \neq u_{ij}$, that is $x \prec u_{ij}$.

Since x is maximal and has more than two occurrences, $\mathcal{L}_{u_{ij}}$ is a proper subset of $\mathcal{L}_x$. Moreover, for all $i \neq j$, u_{ij} is a motif satisfying the quorum, but it is not necessarily maximal. Let v_{ij} be the maximal motif having u_{ij} as occurrence (v_{ij} is obtained from u_{ij} by possibly extending it or replacing some of its don't cares with a letter without losing any occurrences). Clearly $\mathcal{L}_{v_{ij}}+\delta_{ij} = \mathcal{L}_{u_{ij}}$ for some integer $\delta_{ij} \geq 0$. Moreover, $\bigcup_{i,j\in\mathcal{L}_x}(\mathcal{L}_{v_{ij}}+\delta_{ij}) = \bigcup_{i,j\in\mathcal{L}_x}\mathcal{L}_{u_{ij}} = \mathcal{L}_x$ while $x \neq v_{ij}$ for all i, j because $x \prec u_{ij} \preceq v_{ij}$. Motif x would then not be a tiling motif, which is a contradiction.

We just demonstrated that there exist $i, j \in \mathcal{L}_x$ such that $u_{ij} = x$. Hence $u_{i,j}$ is a substring of Merge_{j-i}. We now have to show that for at least one pair $i, j \in \mathit{List}_x$, we have that $u_{ij} = \mathit{Merge}_{j-i}$. We proceed again by absurd. If all u_{ij} which are equal to x were proper substrings of Merge_{j-i}, then we would have $\bigcup_{i,j\in\mathcal{L}_x}\mathcal{L}_{\mathit{Merge}_{j-i}} = \mathcal{L}_x$ and x would thus not be a tiling motif for string s, a contradiction. For at least one pair $i, j \in \mathit{List}_x$, we must therefore have $\mathit{Merge}_{j-i} = u_{ij} = x$. $\bowtie$

We can now state the following result whose proof is a direct consequence of Lemma 2 and Lemma 3.

Theorem 1 *Given a string s, the tiling motifs in the basis for s are a subset of $M = \{\mathit{Merge}_k \mid 1 \leq k \leq n-1\}$.*

Corollary 1 (Linearity of the basis) *The number of elements in the basis of a string s of length n is no more than $n-1$.*

The tiling motifs of the string $s =$ `FABCXFADCYZEADCEADC` are `FA·C`, `EADC`, and `ADC`. All correspond to at least one *Merge*. The only *Merge* which is not a tiling motif is `A·C`.

The notion of basis presented in [7] and the different notion of basis given in [9] both lead to larger sets. For instance, the first notion produces a basis containing 289 elements for the string `AAXAAAXAAAAXAAAAAXAAAAAAXAAAAAAA` of length 32. Our basis has only 19 elements.

For any quorum q, since no pattern occurring less that q times is a motif, the following is trivially true.

Proposition 1 *For all quorum q, all maximal motifs that occur exactly q times are tiling motifs.*

4 Computation of the Basis

In this section, we describe how to compute a basis for the input string s. A trivial algorithm that generates first all (maximal) motifs of s and then selects the tiling motifs takes exponential time. Theorem 1 plays a crucial role in that it implies the existence of a polynomial-time algorithm. Indeed, we know that the basis $\mathcal{B}$ for quorum $q = 2$ is contained in the set M of distinct motifs Merge_k, where $1 \leq k \leq n-1$. Corollary 1 states that the size of M is less than n. Consequently, we are left with the problem of selecting $\mathcal{B}$ as a subset of the motifs in M, which are less than n in number, rather than selecting $\mathcal{B}$ from the set of all possible maximal motifs, which are exponential in number.

We now give an algorithm to find a basis in $O(n^2 \log n \log |\Sigma|)$ time with quorum at least $q = 2$. We first run a brute-force algorithm on s to compute the multiset M' of non empty $Merge_k$, where $1 \leq k \leq n-1$, in $O(n^2)$ time. With each motif $x = Merge_k \in M'$, we associate the set $occ_x = \{i, i+k\}$ of two occurrences of x whose overlap generates $Merge_k$ (they can be easily computed from it). Note that, knowing the string s, the motif length $|x|$ and the set occ_x, we can recover x in $O(|x|)$ time. In other words, given s, we can represent each motif in M' in constant space by using the triplet $\langle i, i+k, |x| \rangle$. Since there can be two or more motifs in M' that are the same motif $x \in M$, we unite all identical motifs. We can compute M from M' in $O(n^2)$ time with radix sorting, and produce also the temporary location lists $T_x = \bigcup_{x'=x, x' \in M'} occ_{x'}$ for each distinct $x \in M$.

Unfortunately, it may be $T_x \subset \mathcal{L}_x$ and so we are missing some occurrences of x. We therefore need to compute the whole list of occurrences $\mathcal{L}_x$ for all merges $x \in M$. Given one such merge x, we employ the algorithm by Fischer and Paterson based on convolution [4] for string matching with don't cares. Its cost is $O((|x| + n) \log n \log |\Sigma|)$ time. Since $|x| \leq n$ and there are at most n such x's, we obtain a total of $O(n^2 \log n \log |\Sigma|)$ time to construct all lists $\mathcal{L}_x$, where $x \in M$.

We then proceed by processing the lists $\mathcal{L}_x$ to filter M, that is, we remove the motifs $x \in M$ that are tiled. Yet, this may turn into an $\Omega(n^3)$ algorithm, as the total size of the lists can be $\Omega(n^2)$ and a list can contain $\Omega(n)$ entries. One such example is for the string $s = \mathtt{A}^n$ having $n-1$ distinct *Merge*s of the form $\mathtt{A}^i$, where $1 \leq i \leq n-1$. Here, $|\mathcal{L}_{\mathtt{A}^i}| = n - i + 1$, and so $\sum_{i=1}^{n} |\mathcal{L}_{\mathtt{A}^i}| = \Theta(n^2)$.

In order to select $\mathcal{B}$ efficiently, we need to exploit some properties of the motifs in $\mathcal{B}$ and in $M - \mathcal{B}$.

Lemma 4 *For each motif $x \in \mathcal{B}$, we have $T_x = \mathcal{L}_x$.*

Proof: Obviously, we have that $T_x \subseteq \mathcal{L}_x$. Let us assume by contradiction that $\mathcal{L}_x \setminus T_x$ is not empty and let $i \in \mathcal{L}_x \setminus T_x$. Since x is equal to at least one *Merge* (Theorem 1), it must be $|T_x| \geq 2$. Let then be $T_x = \{j_1, \ldots, j_h\}$ with $h \geq 2$ and for all $j_k \in T_x$ consider $m_k = Merge_{|j_k - i|}$. These are nonempty and definitely all different from x because no merge $m = x$ is such that $i \in occ_m$ (otherwise i would belong to T_x). These m_k are maximal by Lemma 2. Moreover, x occurs in all m_k's, and thus $\mathcal{L}_{m_k} \subseteq \mathcal{L}_x$ for all $j_k \in T_x$. This holds for all $j_k \in T_x$ and for all the $i \in \mathcal{L}_x \setminus T_x$, thus involving all the occurrences of x. Therefore, possibly collecting all these m_k's for all $j_k \in T_x$ and for all the $i \in \mathcal{L}_x \setminus T_x$, we have that the $Merge_{|j_k - i|}$'s would tile x, which is a contradiction. $\bowtie$

Note that Lemma 4 does not hold for the motifs in $M - \mathcal{B}$. For example, given the string $s = \mathtt{FADABCXFADCYZEADCEADCFADC}$, we have $x = \mathtt{ADC}$ with $\mathcal{L}_x = \{8, 14, 18, 22\}$ while $T_x = \{8, 18\}$. However, we can use Lemma 4 in a positive way. Let $\widehat{M}$ be the set of motifs $x \in M$, such that $T_x = \mathcal{L}_x$. Clearly, $\mathcal{B} \subseteq \widehat{M}$ but there are examples in which $\mathcal{B} \subset \widehat{M}$. Nevertheless, set $\widehat{M}$ has the following interesting property:

Lemma 5

$$\sum_{x \in \widehat{M}} |\mathcal{L}_x| < 2n$$

Proof: Follows from the definition of $\widehat{M}$ and from the observation that $\sum_{x \in \widehat{M}} |T_x| \leq \sum_{x \in M} |occ_x| < 2n$, since $|occ'_m| = 2$ and there are less than n of them. $\bowtie$

After having computed M and the lists $\mathcal{L}_x$ for $x \in M$, our algorithm computes $\widehat{M}$ in $O(n^2)$ time by simply discarding the motifs x such that $T_x \neq \mathcal{L}_x$, as they cannot be in the basis $\mathcal{B}$. After this further selection, we may still have left some tiled motifs in $\widehat{M}$, but these are now such that their total size is $O(n)$ by Lemma 5.

We can now start our check for tiling in $O(n^2)$ time. Given two distinct motifs $x, y \in \widehat{M}$, we want to test for any integer d whether $\mathcal{L}_x + d \subseteq \mathcal{L}_y$ 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 6). By removing their corresponding motifs from $\widehat{M}$, we eventually obtain basis $\mathcal{B}$. 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 can merge $\mathcal{L}_x$ and $\mathcal{L}_y$ in $O(|\mathcal{L}_x| + |\mathcal{L}_y|)$ time to perform the test. The cost of processing $\mathcal{L}_x$ and $\mathcal{L}_y$ for any d would be $O(|\mathcal{L}_x| \cdot |\mathcal{L}_y| + |\mathcal{L}_y|^2)$, which can contribute to a total of $\Omega(n^3)$ time.

We therefore 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 $O(|\mathcal{L}_x| \cdot |\mathcal{L}_y|)$ time. This gives a total cost of $O(\sum_y \sum_x |\mathcal{L}_x| \cdot |\mathcal{L}_y|) = O(\sum_y |\mathcal{L}_y| \cdot \sum_x |\mathcal{L}_x|) = O(n^2)$ by Lemma 5. We now describe how to perform the above check with $\mathcal{L}_x$ and $\mathcal{L}_y$ in $O(|\mathcal{L}_x| \cdot |\mathcal{L}_y|)$ time.

We use two bit-vectors V_1 and V_2 initially set to all zeros. Given $y \in \widehat{M}$, we set $V_1[i] = 1$ if and only if $i \in \mathcal{L}_y$. For each $x \in \widehat{M} - \{y\}$ 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 no more values of d and repeat the test. After examining all $x \in \widehat{M} - \{y\}$, 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. Then we reset the ones in both vectors in $O(|\mathcal{L}_y|)$ time.

Summing up, we have proven the following result.

Theorem 2 *Given an input string s of length n over the alphabet Σ, the basis for it 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 $2n$.*

5 Further Considerations on the Basis

The main property underlying the notion of basis is that it is a generator for all motifs. That is, it is possible to generate all motifs occurring in string s knowing only its basis. It is fairly easy to design a straightforward algorithm to generate all motifs from the basis. However, since the number of motifs, and even maximal motifs, can be exponential, this is not really meaningful unless the time complexity of the algorithm is proportional to the total size of the output. An attempt in this way is done in [8].

The dual problem concerns testing only one pattern. We show how, given a pattern x, we can test if x is a motif for string s, *i.e.*, 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 Lemma 1.

The next two statements deal with the alternative. In both of them, we assume that integer k comes from the decomposition of pattern x in the form $u_o \cdot^{\ell_0} u_1 \cdot^{\ell_1} \ldots u_{k-1} \cdot^{\ell_{k-1}} u_k$ where the u_i's contain no don't cares ($u_i \in \Sigma^*$, $0 \leq i \leq k$, and ℓ_j are non negative integers, $0 \leq j \leq k-1$).

Lemma 6 *The positions of 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 [4] is not necessary. Instead, positions of u_i's 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). $\bowtie$

Lemma 7 *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 Lemma 1, 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 the preceding lemmas, we get an upper bound on the time complexity for testing motifs.

Corollary 2 *Testing if pattern $u_o \cdot^{\ell_0} u_1 \cdot^{\ell_1} \ldots u_{k-1} \cdot^{\ell_{k-1}} u_k$ 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 Inside the procedure sketched in the proofs of the lemmas, it is also possible to use bit-vector pattern matching methods [2, 11, 6] 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 a machine. This is certainly a method of choice for short patterns.

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

[3] 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.

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

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

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

[7] 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.

[8] 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.

[9] 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.

[10] N. Pisanti. *Segment-based distances and similarities in genomic sequences.* PhD thesis, University of Pisa, Italy, 2002.

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