# Masking Patterns in Sequences: A New Class of Motif Discovery with Don't Cares☆

Giovanni Battaglia[a], Davide Cangelosi[a], Roberto Grossi[a], Nadia Pisanti[a]

[a]*Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy.*

**Abstract**

In this paper, we introduce a new notion of motifs, called *masks*, that succinctly represent the repeated patterns for an input sequence $T$ of $n$ symbols drawn from an alphabet $\Sigma$. We show how to build the set of all maximal masks of length $L$ and quorum $q$, in $O(2^L n)$ time and space in the worst case. We analytically show that our algorithms perform better than constant-time enumerating and checking all the potential $(|\Sigma| + 1)^L$ candidate patterns in $T$ after a polynomial-time preprocessing of $T$. Our algorithms are also cache-friendly, attaining $O(2^L\, sort(n))$ block transfers, where $sort(n)$ is the cache oblivious complexity of sorting $n$ items.

*Key words:* Motif inference, motifs with don't care, motif partial order, motifs with masks.

## 1. Introduction

We consider a new class of *motifs with don't cares*, also motivated by sequence analysis in biological data and data mining on sequences. Motifs are repeated patterns, where a pattern is an intermixed sequence of alphabet symbols (*solid* symbols) and special symbols ∘ (*don't care* symbols). The don't care symbol found in a position of the pattern specifies that the position may contain any alphabet symbol (so, it does not care which one). For example, A∘T∘∘C is a motif that repeats twice in the sequence $T$ = AAAATTACCCCATAGT at positions 2 and 3 (starting from 0).

Informally, motifs are those patterns that repeat at least a certain number $q$ of times, where $q$ is a user defined positive integer called the *quorum*. Given an input text $T$ of length $n$, a quorum $q \geq 2$, and a motif length $L$, *motif discovery* is the problem of finding the motifs of length $L$ in the sequence $T$. Each motif may have associated the list of the starting positions of its occurrences in the given sequence $T$. Unfortunately, due to the don't cares, the number of motifs can be exponentially large for increasing values of $L$. Potentially, there can be as many as $\Theta((|\Sigma| + 1)^L)$ motifs, where $\Sigma$ is the alphabet of the distinct symbols in the text $T$. Even though this number can be smaller for some particular instances, the known algorithms discovering these motifs still require, in the worst case, exponential time and space for increasing values of $L$. A lot of research has investigated these issues to mitigate the combinatorial explosion of motifs [1, 2, 3, 4, 5, 6].

**Problem formulation.** In this paper, we follow a new approach based on modeling motifs by using simple binary patterns, called *masks*, that implicitly represent *families of motifs* in $T$ (instead of individual motifs). For example, mask 101001 represents both A∘T∘∘C and T∘G∘∘A: each 1 represents a solid symbol while each 0 represents a don't care symbol. A mask is a *motif* if at least one of its represented patterns occurs $q$ or more times in the given sequence $T$.

As it should be clear from the above informal definition, we can describe interesting repetitions in a sequence, using a description (mask) that is more succinct than before. Therefore, we aim at giving rise to a smaller set of output motifs. Intuitively, consider some patterns that occur at least $q$ times each and that also share the *same structure*, meant as a certain concatenation of solid and don't care symbols. Since they originate from the same mask, we take this mask as a motif. Moreover, any two patterns sharing the same structure but having a different number of occurrences in $T$ (still at least $q$ in number), which were previously considered as different motifs, are now giving rise to the same motif by our definition of mask. Since each mask can be seen as a binary string, we have potentially $2^L$ masks to examine instead of $(|\Sigma| + 1)^L$ classical motifs.

*Email addresses:* gbattag@di.unipi.it (Giovanni Battaglia), cangelo@di.unipi.it (Davide Cangelosi), grossi@di.unipi.it (Roberto Grossi), pisanti@di.unipi.it (Nadia Pisanti)

In summary, our new class of motifs may summarize some regularities in the given sequence $T$, better than ever before. We study the problem of detecting *maximal masks*, namely, the most specific ones (maximal number of 1s) such that at least one of its represented patterns occurs $q$ times or more (see Section 2 for a formal definition).

For example, given the text $T$ = AAAATTACCCCATAGT, fixing $L = 4$ and $q = 2$, we obtain the maximal masks 1110, 0111, and 1101. Notice that 1110 and 0111 are equivalent since they originate the same patterns (three consecutive solid symbols) ignoring border effects, so we can treat them as the same mask. Therefore the patterns that are represented by the maximal masks are AAA, CCC, and AA$\circ$T, and so the parameter $L$ can equivalently be read as an upper bound on their length. Interestingly, the text positions of the occurrences of the patterns implicitly represented by a mask do not overlap (while the patterns can overlap), so we can always associate a partition of the text positions with the mask.

It is worth noting that motif discovery has many applications in the investigation of properties of biological sequences. In such applications, it is a must to allow distinct occurrences of a motif to show some differences. In other words, we actually infer *approximated* motifs. Such approximation can be realized in several ways, according to the kind of application one has in mind. Motifs of limited length with don't cares can typically model biological object such as transcription factors binding sites, that are characterized by a short length, and a high conservation of their structure. Also, they present a high conservation of the contents in certain positions while for others it does not matter at all. The don't care symbols of our masks indeed aim at *masking* the latter, while the solid character should *unmask* the former.

Moreover, our masks could also be employed as building blocks for longer and flexible motifs, of different kind, allowing also indels. In recent years, there has been a growing interest in *seeds* for several applications (preprocessing filtration prior to a multiple alignment, approximate search task, data base search, BLAST like homology search, profile search, probe design) in bioinformatics ([7, 8, 9, 10, 11, 12, 13]). Among them, many have focused the attention on *gapped seeds*, or *spaced seeds* ([14, 15, 16, 17, 18, 19]). It turns out that gapped seeds can be found using the masks, and thus using the algorithms introduced in this paper.

Finally, a new application of finding motifs with don't cares could help to detect structural similarities, with a suitable input sequence. For example, when investigating the folding of a DNA sequence, it can be interesting to rewrite the sequence itself into the alphabet {w, s} replacing each A and T with w (weak), and each C and G with s (strong). The motivation is that in the *base pairing* that assists in stabilizing the DNA structures, adenine (A) binds to thymine (T) via two hydrogen bonds, while cytosine (C) forms three hydrogen bonds with guanine (G). Hence, the latter bond is stronger than the former, and this has an influence on the actual structure of the molecule. In this framework, a motif on such sequence could represent a repeated structure, regardless of the actual DNA bases that form it.

**Our results.** In this paper, we introduce a new notion of motifs, the *masks*, and investigate how to build the set $\mathscr{M}$ of all *maximal masks* of length $L$ and quorum $q$, for an input sequence $T$ of $n$ symbols drawn from an alphabet $\Sigma$.

We first show that the partition of the text positions induced by an arbitrary mask can be conceptually represented as a pruned trie of the represented patterns, storing the classes that constitute the partition in the leaves of the trie. We then extend the Karp-Miller-Rosenberg doubling scheme [20] to the masks having length an increasing sequence of powers of 2 up to $L$. Interestingly, the resulting scheme avoids to actually create the tries and just needs scanning and sorting some suitable lists of consecutive pairs and triplets of integers (this is considered to be cache friendly).

However, the above method generates the set $\mathscr{Q}$ of all the masks having quorum $q$ for the given sequence $T$, where $\mathscr{Q} \supseteq \mathscr{M}$, including those that are not maximal. Maximality checking in $\mathscr{Q}$ to select the masks in $\mathscr{M}$ can be expensive: precisely, it may take $\Theta(|\mathscr{Q}|^2 L)$ time in the worst case (e.g. [21]), yielding a cost of $\Omega(2^{2L} L)$ time. We therefore introduce the crucial notion of *safe masks*, which includes the maximal masks as a special case. We show how to traverse the lattice of $2^L$ masks of length $L$ touching only safe masks, so that maximal masks can be efficiently detected. Our tests ([22]) show evidence of the advantages of this strategy.

Consequently, the final bound for discovering all the masks belonging to $\mathscr{M}$ is $O(2^L n)$ time and space in the worst case, which is our main result. To evaluate the efficiency of the proposed algorithms, consider the following scenario in which, after preprocessing the text $T$ in polynomial time, we ideally check, in constant time per pattern, if any of the $(|\Sigma| + 1)^L$ candidate patterns has quorum and is maximal. We compare the cost of $O(n^{O(1)} + (|\Sigma| + 1)^L)$ thus obtained against the cost of $O(2^L n)$ that we obtain in this paper. Since $2^L n = O((|\Sigma| + 1)^L)$ when $L = \Omega(\log_{|\Sigma|} n)$, and $2^L n = O(n^{O(1+1/\log_2 |\Sigma|)})$ otherwise, we can bound our $O(2^L n)$ cost by $O(n^{O(1+1/\log_2 |\Sigma|)} + \min\{2^L n, (|\Sigma|+1)^L\})$. As a result, the latter is always better than the ideal bound of $O(n^{O(1)} + (|\Sigma| + 1)^L)$ stated above. In other terms, our algorithms perform better than virtually *constant-time enumerating and checking* all the potential $(|\Sigma| + 1)^L$ candidate patterns in $T$.

At this point, we need to make an observation on the model of computation adopted in this paper, which is the standard RAM with word size of $w$ bits. Since we need to address potentially $\Theta(2^L)$ masks, we assume in the rest of the paper that $w = O(L)$. If this is not the case, the total cost must be multiplied by a factor of $O(L/w)$.

Finally, given the scan-and-sort nature of our algorithms, we naturally obtain a cache-oblivious solution to our problem as a byproduct. To our knowledge, this is the first cache-oblivious solution for a motif discovery problem, which is useful for long input sequence(s). Indeed, our algorithms work also in the *cache-oblivious model*, introduced by Frigo et al. [23], to deal with such a situation, where $M$ is the size of the fast memory, and $B$ is the size of the block in each transfer between fast and slow memories. The goal is to minimize the number of block transfers without letting the algorithms know the values of $M$ and $B$ (which are introduced for the purpose of the analysis only). For example, scanning $n$ consecutive elements has a complexity of $\Theta(n/B)$ block transfers while the optimal complexity of sorting is $sort(n) = \Theta\left(\frac{n}{B}\log_{M/B}\frac{n}{B}\right)$ block transfers [24, 25, 26, 23].

Employing the simple scan and the cache-oblivious sorting in our algorithms, we do not need to further orchestrate their memory accesses. Using this model, we can indeed obtain a complexity of $O(2^L\, sort(n))$ block transfers for finding $\mathcal{Q}$ and $\mathcal{M}$. Along the same lines, our algorithms can run in a distributed setting, such as a cluster of computers, using distributed sorting.

**Related problems and state of the art.** We are not aware of any previous work introducing our class of motifs. Hence, we relate our results in motif discovery to those of mining frequent itemsets, where more sophisticated techniques have been found over the years. The notion of masks comes naturally into play when performing data mining for frequent itemsets, where the "apriori" algorithm is intensively employed [27]. Here, a set of $L$ items is given, and each transaction (basket) corresponds to a subset of these items, which can be represented as a binary sequence in which the $i$th symbol is 1 if and only if the $i$th item is in the basket. A set of baskets can be therefore represented as a set of masks in our terminology. For the lattice of all possible $2^L$ masks, all possible itemsets should be examined. Note that, instead, our definition of masks has the goal of condensate patterns that have the same sequence of solid and don't care symbols. Moreover, our traversal of the lattice is different from the apriori, since we start from the top and generate candidates in a different way, namely, using safe masks.

As far as we know, the "dualize and advance" algorithm [28, 29] is the best theoretical approach that can be obtained in terms of running time. It sets up an interesting connection between mining itemsets in the lattice of $2^L$ masks and finding hypergraph traversals. In our terminology, suppose to have incrementally found some of the maximal masks, say $\mu_1, \mu_2, \ldots, \mu_k$. We can build an hypergraph in which the nodes are the $L$ available items numbered from 1 to $L$, and the hyperedges are the complement of $\mu_1, \mu_2, \ldots, \mu_k$ (in other words, the 0s indicate the chosen items). Then, in order to find additional maximal masks (and hence add hyperedges), it suffices to find all the hypergraph traversals as starting points for upward paths in the lattice, where each traversal is a minimal hitting set for the current set of $k$ hyperedges [30].

The problem of finding hypergraph traversals is intimately related to the dualization of monotone Boolean functions [31]. The known algorithms require $O(2^L)$ time in the worst case [30, 32] until the seminal result in [33, 34] showing a subexponential bound proportional to $t(k) = k^{O(\log k)}$ time, when the number of hyperedges $k$ is $o(2^L)$. This algorithm is plugged into the scheme of the "dualize and advance" algorithm, giving a bound of $O\left(n \times t(|\mathcal{M}| + |Bd^-(\mathcal{M})|)\right)$ as shown in [28], where we include the cost $O(n)$ of verifying the quorum, and $Bd^-(\mathcal{M})$ is a set of non-maximal masks that are "close" inside the lattice to the ones in $\mathcal{M}$. While $|\mathcal{M}|$ can be subexponential, there are cases in which $|\mathcal{M}| + |Bd^-(\mathcal{M})| = \Theta(2^L)$ (see [28]) and so the final bound can be $\Omega(2^{L^2} n)$.

Surprisingly, this and other approaches based on hypergraph traversals, which are the state of the art theoretically, are slower than our solution in the worst case. We have some preliminary experiments that our solution is also faster in practice. We counted the number of times a function checking the quorum for a given mask is invoked by the algorithms, which is a clear measure of their running time. Here, dualize and advance needs to query many more masks than our safe masks, thus suggesting that the notion of safeness of masks is crucial to our algorithms. We plan to run an extensive experimental analysis as future work.

**Paper organization.** This paper is organized as follows. In Section 2, we give a formal definition of our motifs based on masks. We then present efficient algorithms to compute these motifs, reviewing also basic notions as that of maximality in the light of this new class of motifs. In Section 3, we show how to discover the motifs of length $L$ in $O(2^L n)$ time by extending the Karp-Miller-Rosenberg approach to the masks. In Section 4, we represent the space of all possible $2^L$ masks of length $L$ as a lattice on which it is defined an oracle function for the quorum and introduce the notion of safe masks; we describe how to traverse implicitly this lattice for discovering maximal masks in it, querying the oracle only for safe masks. Finally, we draw our conclusions in Section 5.

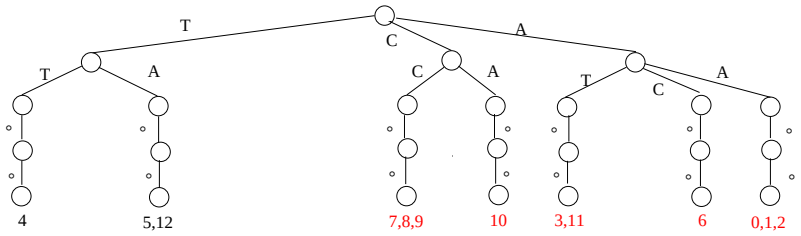


Figure 1: Trie for $\equiv_\mu$ where $\mu = 11\circ\circ$

## 2. A New Class of Motifs

In this section, we introduce our new class of motifs with don't cares. Starting from some basic notions, we describe some features of this class that will then be exploited by the algorithms in the rest of the paper. Let $T$ be an input text of size $n$ drawn over the alphabet $\Sigma$. The sequence $T$ can be seen as an array $T[0 \dots n-1]$ of symbols, where symbol $T[i] \in \Sigma$ is stored into position $i$, for $0 \leq i \leq n-1$. A substring $T[i]T[i+1]\cdots T[j]$ is represented as $T[i \dots j]$.

### 2.1. Masks and patterns

Given a positive integer $L$, we call *mask* any binary sequence in $\{\mathtt{0}, \mathtt{1}\}^L$ (hence, $L$ is the length of the mask). For a given mask $\mu = \mu[0 \dots L-1]$, we define $S_\mu = \{i \mid \mu[i] = \mathtt{1}, \text{for } 0 \leq i \leq L-1\}$ as the set of its *solid positions*, and $D_\mu = \{i \mid \mu[i] = \mathtt{0}, \text{for } 0 \leq i \leq L-1\}$ as the set of its *don't care positions*. For example, mask $\mu = \mathtt{1010}$ has $S_\mu = \{0, 2\}$ and $D_\mu = \{1, 3\}$.

A *pattern* is a regular expression $m \in (\Sigma \cup \{\circ\})^L$, where $\circ$ is the *don't care symbol* that matches any symbol in $\Sigma$. We say that a pattern $m$ is an *instance* of a mask $\mu$ when, for each position $0 \leq k \leq L-1$, it is $m[k] = \circ$ if and only if $\mu[k] = \mathtt{0}$. For example, given $\Sigma = \{\mathtt{A}, \mathtt{C}\}$, the mask $\mu = \mathtt{1010}$ has four instances, namely, $m_1 = \mathtt{A}\circ\mathtt{A}\circ$, $m_2 = \mathtt{C}\circ\mathtt{A}\circ$, $m_3 = \mathtt{A}\circ\mathtt{C}\circ$, and $m_4 = \mathtt{C}\circ\mathtt{C}\circ$.

By exploiting the fact that a mask $\mu$ implicitly represents the patterns that are its instances, we define a new relation among the text substrings according to $\mu$, as follows.

**Definition 1 ($\equiv_\mu$ relation).** *Given a mask $\mu$ of length $L$, a text $T$ of length $n$, and two text positions $0 \leq i, j \leq n-L+1$, we say that $i \equiv_\mu j$ if and only if $T[i+k] = T[j+k]$ for each solid position $k \in S_\mu$ in the mask.*

Definition 1 relates any two text substrings of length $L$, appearing at positions $i$ and $j$, when these substrings match in every solid position specified by the mask $\mu$. For example, given $T[i \dots i+L-1] = \mathtt{ACTACT}$ and $T[j \dots j+L-1] = \mathtt{AGTTCT}$, let us consider the two masks $\mu = \mathtt{101011}$ and $\mu' = \mathtt{110111}$. It holds $i \equiv_\mu j$ because $\mathtt{A}\circ\mathtt{T}\circ\mathtt{CT}$ occurs both at positions $i$ and $j$ of $T$, while $i \not\equiv_{\mu'} j$ because the two substrings $\mathtt{ACTACT}$ and $\mathtt{AGTTCT}$ mismatch at the solid positions in $\{1, 3\} \subseteq S_{\mu'}$.

It is easy to see that the relation $\equiv_\mu$ is an equivalence relation. Therefore, for a given mask $\mu$, the relation $\equiv_\mu$ induces a *partition* of the first $n-L+1$ positions in $T$. Namely, for each equivalence class $C$ in the partition, we have that $i, j \in C$ if and only if $i \equiv_\mu j$. Hence, each text position $i$ (for $0 \leq i \leq n-L+1$) belongs to exactly one equivalence class. We denote the partition resulting from $\mu$ by $\pi_\mu$. We use $|\pi_\mu|$ to indicate the number of equivalence classes in it, and $|C|$ to indicate the number of elements in a class $C \in \pi_\mu$.

Given an equivalence class $C$ of a partition $\pi_\mu$, we can associate a pattern $m_C$ of length $L$ with $C$. Specifically, symbol $m_C[k] = \circ$ if $k$ is a don't care position of the mask $\mu$ ($k \in D_\mu$) while $m_C[k] = T[i+k]$ if $k$ is a solid position ($k \in S_\mu$ and for any arbitrary $i \in C$).

In order to better illustrate the properties of the partition $\pi_\mu$ and the corresponding patterns $m_C$ where $C \in \pi_\mu$, we can use a trie (digital search tree [35]) built on the set of strings $\{m_C \mid C \in \pi_\mu\}$. In this trie, the special symbol $\circ$ can be treated as an ordinary symbol, since all the patterns share the same mask $\mu$. We can arrange the strings in this way since the arcs on the same level of the trie are either all labeled with a solid symbol or all labeled with a don't care symbol. Each root-to-leaf path spells out a distinct pattern $m_C$, and the corresponding leaf stores the text positions in the class $C$ of the partition $\pi_\mu$.

**Example 1.** Consider the alphabet $\Sigma_{DNA} = \{\mathtt{A}, \mathtt{T}, \mathtt{C}, \mathtt{G}\}$ and the text $T = \mathtt{AAAATTACCCCATAGT}$ of length $n = 16$. For a mask $\mu = \mathtt{1100}$ of length $L = 4$, the partition induced by $\mu$ is $\pi_\mu = \{C_0 C_1 \cdots C_6\}$, where $C_0 = \{4\}$, $C_1 = \{5, 12\}$,

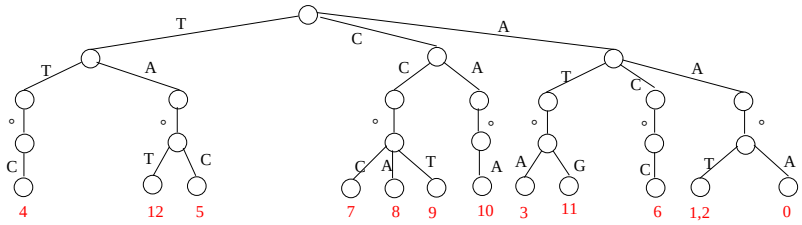


Figure 2: Trie for $\equiv_{\mu'}$ where $\mu' = 11\circ1$.

$\ldots$, $C_6 = \{0, 1, 2\}$ are the classes labeling the leaves of the trie shown in Figure 1. The instances of $\mu$ are the patterns $m_{C_0} = \texttt{TT}\circ\circ$, $m_{C_1} = \texttt{TA}\circ\circ$, $\ldots$, $m_{C_6} = \texttt{AA}\circ\circ$.

### 2.2. *Partial order of masks and maximality*

We now draw our attention to the masks $\mu$ that have the maximum number of solid symbols while inducing a partition $\pi_\mu$ that contains at least one class $C$ such that $|C| \geq q$ for the given *quorum* $q$. For this, we need to introduce a partial order on the masks.

**Definition 2 ($\preceq$ relation).** *Given two masks $\mu$ and $\mu'$ of length $L$, we say that $\mu$ is* less specific *than $\mu'$ (denoted by $\mu \preceq \mu'$) if and only if $\mu[i] \leq \mu'[i]$ for $0 \leq i \leq L-1$.*

When Definition 2 holds, we also say that $\mu'$ is *more specific* than $\mu$, and that $\mu$ is a predecessor of $\mu'$, and $\mu'$ is a successor of $\mu$. For example, $\texttt{0001} \preceq \texttt{1101}$, while $\texttt{0001} \not\preceq \texttt{0010}$. The mask $\mu$ is an *immediate predecessor* of $\mu'$ (denoted $\mu \preceq_1 \mu'$) if $\mu \preceq \mu'$ and they differ in exactly one symbol. For example, $\texttt{1001} \preceq_1 \texttt{1101}$. We can define the immediate successor analogously.

Relation $\preceq$ is a partial order among the masks because it is reflexive, antisymmetric ($\texttt{1001} \preceq \texttt{1101}$, but $\texttt{1101} \not\preceq \texttt{1001}$) and transitive. Hence, it gives rise to the partially ordered set $\mathscr{L} = \langle \{1, 0\}^L, \preceq \rangle$, which is a finite lattice of $2^L$ masks. The top mask of $\mathscr{L}$ is $1 \ldots 1$ and the bottom mask is $\texttt{0} \ldots \texttt{0}$. The lattice $\mathscr{L}$ is isomorphic to the power set lattice $\mathscr{P} = \langle \mathcal{P}(\{0, \ldots, L-1\}), \subseteq \rangle$ because each mask represents the characteristic vector of a set in the powerset $\mathcal{P}(\{0, \ldots, L-1\})$, and $\mu \preceq \mu'$ if and only if $S_\mu \subseteq S_{\mu'}$.

Analogously, we can define the $\preceq$ relation between patterns. Given two patterns $m$ and $m'$ of length $L$, we say that $m$ is *less specific* than $m'$ (written $m \preceq m'$) if and only if either $m[i] = m'[i]$ or $m[i] = \circ$ for $0 \leq i \leq L-1$. For example, $\circ\circ\circ\texttt{A} \preceq \texttt{AT}\circ\texttt{A}$ while $\circ\circ\circ\texttt{A} \not\preceq \texttt{AT}\circ\texttt{C}$ and $\circ\texttt{C}\circ\texttt{A} \not\preceq \texttt{A}\circ\circ\texttt{A}$. Note that $\preceq$ is a partial order also for the patterns. However, it gives rise to a lattice of $(|\Sigma| + 1)^L$ patterns, and so in the rest of the paper we prefer to adopt the binary lattice $\mathscr{L}$ of $2^L$ masks.

At this point, we may wonder what is the connection between the partitions induced by the masks (Section 2.1) and the lattice $\mathscr{L}$ formed by the masks. The following result shows interesting properties of two masks regarding inclusive relation between their equivalence classes.

**Proposition 1.** *For any two masks $\mu$ and $\mu'$ such that $\mu \preceq \mu'$, $\pi_{\mu'}$ is a refinement of $\pi_\mu$. Namely:*

*(i) For each class $C \in \pi_\mu$, there exist classes $C_0, C_1, \ldots, C_{s-1} \in \pi_{\mu'}$ that form a partition of $C$.*

*(ii) For each class $C' \in \pi_{\mu'}$, there exists a class $C \in \pi_\mu$ such that $C' \subseteq C$.*

**Example 2 (continued).** Consider mask $\mu' = \texttt{1101}$, for which $\mu \preceq \mu'$, where $\mu = \texttt{1100}$. Consider the equivalence classes of $\pi_\mu$ (Figure 1) and $\pi_{\mu'}$ (Figure 2). We have that $C_0 = \{4\} = C'_0$, $C_1 = \{5, 12\} = \{12\} \cup \{5\} = C'_1 \cup C'_2$, $C_2 = \{7, 8, 9\} = \{7\} \cup \{8\} \cup \{9\} = C'_3 \cup C'_4 \cup C'_5$, $C_3 = \{10\} = C'_6$, $C_4 = \{3, 11\} = \{3\} \cup \{11\} = C'_7 \cup C'_8$, $C_5 = \{6\} = C'_9$, and $C_7 = \{0, 1, 2\} = \{1, 2\} \cup \{0\} = C'_{10} \cup C'_{11}$. Therefore, both properties (i) and (ii) hold.

We can easily extend the above properties to the patterns, obtaining the following corollary.

**Corollary 2.** *Let $\mu$ and $\mu'$ be any two masks such that $\mu \preceq \mu'$. For any class $C \in \pi_\mu$, there exists a class $C' \in \pi_{\mu'}$ such that their corresponding patterns satisfy $m_C \preceq m_{C'}$. (The reverse holds as well, namely, for any class $C' \in \pi_{\mu'}$, there exists a class $C \in \pi_\mu$ such that $m_C \preceq m_{C'}$.)*

*2.3. Problem statement*

We are given a text $T$ of length $n$ and a length $L$ for the masks, along with a quorum $q \geq 2$. We say that a mask $\mu$ has *quorum* if there is an equivalence class $C$ in the partition $\pi_\mu$ such that $|C| \geq q$. The relation $\preceq$ and the quorum $q$ are the key ingredients to find interesting masks.

Let $\mathscr{Q}(L, T, q)$ be the set of all masks of length $L$ that have quorum $q$. A mask $\mu$ is *maximal* if it has quorum and no other mask $\mu'$ of the same length has quorum and is more specific than $\mu$ (i.e. $\mu \preceq \mu'$). We denote by $\mathscr{M}(L, T, q) \subseteq \mathscr{Q}(L, T, q)$ the set of all maximal masks ($\mathscr{M}$ and $\mathscr{Q}$ for short, respectively), and address the following problem in this paper.

**Maximal Masks Problem (MMP).** Given a mask length $L$, a text $T$, and a quorum $q$, find the set of all the maximal masks $\mathscr{M}(L, T, q)$.

**Example 3 (continued).** Using the same text $T$ = AAAATTACCCCATAGT, for $L = 4$ and $q = 2$, the set of masks with quorum is $\mathscr{Q}(4, T, 2)$ = {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1100, 1101, 1110}, while the maximal masks are only those of the set $\mathscr{M}(4, T, 2)$ = {0111, 1101, 1110}. The patterns that are instances of the maximal masks are ∘AAA, AA∘T, and ∘CCC. Notice that masks 0111 and 1110, that are both in $\mathscr{M}$, actually represent the same patterns, except possibly border effects, because one is the shift of the other obtained by only changing the number of ∘s at the sides. We will show in Section 3 how to remove this sort of redundancy, which will actually obtain an important reduction of the search space, as a positive side effect.

We can see that checking whether a mask $\mu$ has quorum $q$ corresponds to evaluating a Boolean predicate $P_T(\mu)$ that returns true if and only if there exists a pattern $m$ that is an instance of $\mu$ and that has at least $q$ matches in $T$. Note that $P_T(\mu)$ is anti-monotone. [1] Hence, MMP can be equivalently restated as checking an anti-monotone predicate $P_T(\mu)$ on a binary lattice of $2^L$ masks, to discover all the maximal masks where the predicate holds. In the rest of the paper, we will describe how to solve this equivalent problem for sequences, and we will make use of this view of MMP.

## 3. The KMR Approach for Masks with Quorum

We now describe our first algorithm to solve MMP and build the sets $\mathscr{Q}(L, T, q)$ and $\mathscr{M}(L, T, q)$. Conceptually, we want to build the partition $\pi_\mu$ for each mask $\mu \in \{0, 1\}^L$, to check whether $\mu$ has quorum (i.e. there exists a class $C \in \pi_\mu$ such that $|C| \geq q$), and that no mask $\mu'$ with $\mu \preceq \mu'$ has quorum. Note that a straightforward way of checking whether a mask has quorum requires to scan the text $T$ and build a trie like that shown in Figure 2, in $\Theta(Ln)$ time. This would give a total cost of $\Omega(L2^L n)$ time since there are $\Omega(2^L)$ masks to check.

We reduce the above cost to $O(2^L n)$ time using a different and simple approach that avoids explicitly building the trie for each mask. Our idea is to maintain the partitions induced by the masks of increasing lengths as follows. Assuming without loss of generality that $L$ is a power of 2, we first compute the partitions induced by the masks of length 1; inductively, given the partitions induced by the masks of length $2^i$, we show how to compute the partitions induced by the masks of length $2^{i+1}$ in a way that does not explicitly need the tries, even though we will implicitly refer to them during the description of our algorithm.

We implement our idea using the *KMR approach* proposed by Karp, Miller and Rosenberg [20]. The KMR approach addresses the problem of identifying exact repeated substructures of fixed size in a given combinatorial structure. It applies to finding repeated substrings in strings, repeated subtrees in trees, and repeated segments in arrays. For strings, KMR uses a relation $E_k$ according to which two substrings of length $k$ beginning at positions $i$ and $j$ of the text $T$ are *k-equivalent*, written $i\, E_k\, j$, if and only if they are identical in every position. Given this, KMR provides a characterization of $E_{k+k'}$ in terms of $E_k$ and $E_{k'}$, so that it constructs inductively the sets $\{E_2, E_4, E_8, \ldots, E_L\}$ by setting $k = k'$. That is, it doubles the length of the substrings by means of a concatenation of two substrings from the previous iteration. KMR starts out from the set $E_1$ (obtained by a simple scan of the input sequence $T$), and ends at the required length $L$. Each iteration takes time $O(n)$, where $n$ is the length of the text $T$. Since the number of iterations is $O(\log L)$, the overall complexity of KMR is $O(n \log L)$ time.

In our case, MMP differs from the problem solved by KMR since we use masks as one further level of abstraction. We do not apply KMR directly to the text substrings; instead, we double the length of the masks and, as a side effect,

[1] We recall that, given a partial order $\preceq$, a predicate $p$ is *anti-monotone* if, for any $x$ and $y$ such that $x \preceq y$, we have that $p(y) = true$ implies $p(x) = true$. Conversely, $p$ is *monotone* if $p(x) = true$ implies $p(y) = true$.

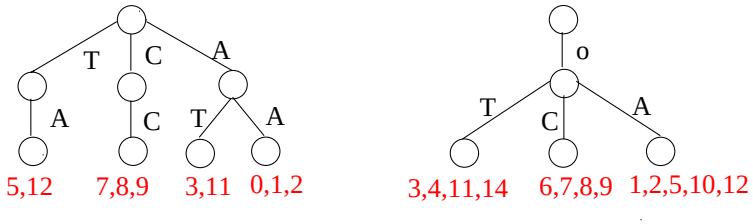


Figure 3: Partitions for masks $\mu = 11$ (left) and $\mu' = 01$ (right).

we double the length of the induced equivalent substrings in the text (i.e they match on all the solid positions of the given mask). In this way, the original KMR can be seen as a special case of our approach when the mask is made by all solid symbols $11\cdots1$.

We observe that, for any given mask, we can employ KMR when the relation $E_k$ is replaced by the $\equiv_\mu$ relation introduced in Definition 1. We also replace the concatenation performed by KMR at each iteration by the concatenation operation among masks. Given two masks $\mu$ and $\mu'$, we indicate their concatenation by $\mu\mu'$. The following result relates the $\equiv_\mu$ equivalence relation to the mask concatenation operation, showing how the KMR paradigm can be generalized to our case.

**Lemma 3.** *Given a string $T$ of length $n$, two masks $\mu, \mu'$, two positions $i, j$ in $T$ then $i \equiv_{\mu\mu'} j$ if and only if $i \equiv_\mu j$ and $(i + |\mu|) \equiv_{\mu'} (j + |\mu|)$.*

*Proof*: We start by showing how $i \equiv_{\mu\mu'} j$ implies $i \equiv_\mu j$ and $(i + |\mu|) \equiv_{\mu'} (j + |\mu|)$. If $i \equiv_{\mu\mu'} j$, then by definition we have that

$$T[i + k] = T[j + k] \text{ for all } k \in S_{\mu\mu'}. \tag{1}$$

Notice that $S_{\mu\mu'} = S_\mu \cup (S_{\mu'} + |\mu|)$. Hence, (1) implies that $(i)$ $T[i+k] = T[j+k]$ for all $k \in S_\mu$, and $(ii)$ $T[i+k] = T[j+k]$ for all $k \in (S'_\mu + |\mu|)$. Observe that $(i)$ exactly matches the definition of $i \equiv_\mu j$, which is then proved. On the other hand, $(ii)$ implies that $T[i + k] = T[j + k]$ holds for $k = |\mu| + k'$ for all $k' \in S_{\mu'}$, that is to say $T[i + |\mu| + k'] = T[j + |\mu| + k']$ for all $k' \in S_{\mu'}$, and hence that $(i + |\mu|) \equiv_{\mu'} (j + |\mu|)$. In order to show that if $i \equiv_\mu j$ and $(i + |\mu|) \equiv_{\mu'} (j + |\mu|)$, then $i \equiv_{\mu\mu'} j$, it is enough to observe that all steps above can be inverted, and hence the result is proved. □

### *3.1. Partition construction and generation of masks*

Using Lemma 3 requires an efficient procedure that computes all the related equivalence classes in a partition. In this section we describe an algorithm that solves this problem.

Given a quorum $q \geq 2$ and two partitions $\pi_\mu$ and $\pi_{\mu'}$, where $\mu$ is not necessarily different from $\mu'$, the algorithm returns a new partition $\pi_{\mu\mu'}$ built according to Lemma 3, possibly filtered in order to satisfy the quorum constraint. The key point is illustrated in Figure 3, where we are given two partitions $\pi_\mu$ and $\pi_{\mu'}$ and we want to obtain the new partition $\pi_{\mu\mu'}$ shown in Figure 2. (Note that the text is the same as before $T =$ `AAAATTACCCCATAGT`, that $q = 2$, and that the tries are shown for the sake of presentation since we do not actually employ them in our implementation.) In order to concatenate two masks $\mu$ and $\mu'$ of length $\ell$, the main steps can be summarized as follows (we refer to Figure 2 and Figure 3 as an example).

1. We are given masks $\mu$ and $\mu'$ and their induced partitions $\pi_\mu$ and $\pi_{\mu'}$. We consider only the equivalence classes $C$ such that $|C| \geq q$, and number these classes so that each class has its own *class name* inside its partition. (In our example, $\pi_\mu = [\{5, 12\}_0, \{7, 8, 9\}_1, \{3, 11\}_2, \{0, 1, 2\}_3]$ and $\pi_{\mu'} = [\{3, 4, 11, 14\}_0, \{6, 7, 8, 9\}_1, \{1, 2, 5, 10, 12\}_2]$, where we ignore classes with less than $q$ elements and report the numbering of each relevant class as its subscript.)
2. We create a (multiset) list $LP$ of pairs as follows. First, for each class $C \in \pi_\mu$, we add the pairs $\langle i, n_C\rangle$ to $LP$ for all positions $i \in C$, where $n_C$ is the number assigned to $C$ in step 1. Second, for each class $C' \in \pi_{\mu'}$, we add the pairs $\langle i' - |\mu|, n_{C'}\rangle$ to $LP$ for all positions $i' \in C'$ such that $i' \geq |\mu'|$, where $n_{C'}$ is the number assigned to $C'$ in step 1. (We obtain $LP = [\langle 5, 0\rangle, \langle 12, 0\rangle, \langle 7, 1\rangle, \langle 8, 1\rangle, \langle 9, 1\rangle, \langle 3, 2\rangle, \langle 11, 2\rangle, \langle 0, 3\rangle, \langle 1, 3\rangle, \langle 2, 3\rangle, \langle 1, 0\rangle, \langle 2, 0\rangle, \langle 9, 0\rangle, \langle 12, 0\rangle, \langle 4, 1\rangle, \langle 5, 1\rangle, \langle 6, 1\rangle, \langle 7, 1\rangle, \langle 3, 2\rangle, \langle 8, 2\rangle, \langle 10, 2\rangle]$ in our example since $|\mu'| = 2$.)
3. We sort the list $LP$ in a *stable* way according to the first component of each pair in it. We drop from the list the pairs $\langle i, j\rangle$ such that no other pair has $i$ as its first component in the list. (We obtain $LP = [\langle 1, 3\rangle, \langle 1, 0\rangle, \langle 2, 3\rangle, \langle 2, 0\rangle, \langle 3, 2\rangle, \langle 3, 2\rangle, \langle 5, 0\rangle, \langle 5, 1\rangle, \langle 7, 1\rangle, \langle 7, 1\rangle, \langle 8, 1\rangle, \langle 8, 2\rangle, \langle 9, 1\rangle, \langle 9, 0\rangle, \langle 12, 0\rangle, \langle 12, 0\rangle]$.)

4. The actual concatenation between $\mu$ and $\mu'$ takes place. Indeed, there is an occurrence of a pattern $m$ (instance of mask $\mu\mu'$) in position $i$ if and only if $\langle i, j\rangle$ and $\langle i, j'\rangle$ are consecutive pairs in $LP$ for some $0 \leq j, j' \leq n-1$. Hence, we generate a triplet $\langle j, j', i\rangle$ from these two pairs (note that $\langle i, j\rangle$ precedes $\langle i, j'\rangle$ in $LP$), thus forming a list $LT$ of triplets. (We obtain $LT = [\langle 3,0,1\rangle, \langle 3,0,2\rangle, \langle 2,2,3\rangle, \langle 0,1,5\rangle, \langle 1,1,7\rangle, \langle 1,2,8\rangle, \langle 1,0,9\rangle\ \langle 0,0,12\rangle]$.)
5. We lexicographically sort the list $LT$ according to the first two components of each triplet in it. We drop from the list the triplets $\langle j, j', i\rangle$ such that there are less than $q$ triplets in the list having $j$ and $j'$ as their first component in the list, since they do not reach the quorum. (We obtain $LT = [\langle 3,0,1\rangle, \langle 3,0,2\rangle]$.)
6. We start from an empty partition $\pi_{\mu\mu'}$. For each maximal run of consecutive triplets $\langle i, j, k_1\rangle, \langle i, j, k_2\rangle, \ldots \langle i, j, k_r\rangle$ in $LT$ ($r \geq q$), we add the class $\{k_1, k_2, \ldots, k_r\}$ to $\pi_{\mu\mu'}$. We return $\pi_{\mu\mu'}$ after completing the scan of $LT$. (In our example, $\pi_{\mu\mu'} = [\{1,2\}]$ since only one class contains at least $q$ elements in Figure 2.)

For the sake of simplicity, we described steps 3 and 5 as sorting steps, but it suffices a stable pairing of the input (for example by using the Google's `mapReduce` primitive), meaning that pairs (triplets) having the same first (two) component(s) should be consecutive in the resulting list.

**Lemma 4.** *Given partitions $\pi_\mu$ and $\pi_{\mu'}$ and a quorum threshold $q \geq 2$, steps 1–6 correctly compute the set $\{C \in \pi_{\mu\mu'} \mid |C| \geq q\}$ in $O(n)$ time and space.*

*Proof*: We begin by proving the method to be correct. Given two partitions $\pi_\mu$ and $\pi_{\mu'}$, in step 1, we label each class $C$ of theirs, by a distinct class name $n_C$, while in step 2 we rewrite each occurrence of a $\mu$'s instance (i.e. a text position $i$ in a class $C \in \pi_\mu$) as a pair $\langle i, n_C\rangle$ and each occurrence $j'$ of an instance of $\mu'$ as a pair $\langle j, n_{C'}\rangle$ where $j = j' - |\mu|$. In this way, given two pairs $\langle i, n_C\rangle$ and $\langle j, n_{C'}\rangle$, if $i$ is equals to $j$ then $i \equiv_\mu j$ (since $i$ belongs to an equivalence class in $\pi_\mu$) and $(i + |\mu|) \equiv_{\mu'} (j + |\mu|)$ (since $j'$ belongs to an equivalence class in $\pi_{\mu'}$). In order to detect pairs having the same first component, in step 3 we sort them in a stable way, discarding all pairs that do not share the first value with any other. At step 4 for each couple of consecutive pairs $\langle i, n_C\rangle, \langle i, n_{C'}\rangle$ the triplets $\langle n_C, n_{C'}, i\rangle$ representing the text position $i$ of a new equivalence class in $\pi_{\mu\mu'}$ is created. At this point all the new classes with strictly less than $q$ text positions are discarded and the partition $\pi_{\mu\mu'}$ containing the remaining classes is finally returned (detecting equivalence classes having more than $q$ text positions is easy thanks to the sorting of step 5: triplets representing text positions in the same class $C'' \in \pi_{\mu\mu'}$ are now grouped as they agree on their first two components $n_C$ and $n_{C'}$).

We prove that steps 1–6 take $O(n)$ time and space. Since there can be at most $O(n)$ positions, the list generated at step 1 has size in $O(n)$. Its sorting in step 2 can be done in $O(n)$, for example, using radix sort since the integers are small. After sorting, the detection of pairs to be dropped at step 2, as well as that of triplets to be generated at step 3 can also be done in linear time, because pairs that start with the same position are now consecutive. Each newly generated list is either a permutation of the previous one, or even a subset of it, and thus the size remains in $O(n)$. Therefore, also the sorting of step 5 can be done in linear time using radix sort because also the number of distinct classes cannot be larger than $n$. This sorting allows us to detect in linear time the triplets to be dropped at the same step. Similarly, it permits the final detection of maximal runs to be done in linear time as well. Therefore, the overall time and space complexity is $O(n)$. □

Notice that the elimination, at each iteration, of masks that do not satisfy the quorum, actually results in a practical important reduction of the search space. Notice also that if we build the classes of a new mask obtained by the overlapping of two shorter masks (rather than their concatenation), then the very same procedure can be applied (having the same complexity) with the only difference that Step 2 should build pairs $\langle i' - \delta, n_{C'}\rangle$ instead of $\langle i' - |\mu|, n_{C'}\rangle$, where $\delta$ is the size of the overlap.

### *3.2. Equivalent masks*

We now list some interesting properties that allow us to generate half of all possible $2^L$ masks. Although these properties do not improve over the worst-case complexity, they are useful optimizations for the practical behavior of our algorithms.

We first need to introduce some notation. Intuitively, consider the partitions shown in Figures 1, 4, and 5, respectively, for masks $\mu = \texttt{1100}$, $\mu' = \texttt{0110}$, and $\mu'' = \texttt{0011}$. The classes in these partitions are reported in Table 1, so that their mutual dependence is highlighted. Ignoring border effects in the first and the last $L-1$ positions of the text, we can say that $\mu$, $\mu'$, and $\mu''$ conceptually represent the same set of patterns: those that have only two solid and adjacent symbols. Each row of Table 1 puts the classes of different partitions into a one-to-one correspondence, in which a class can be obtained from another by adding an integer $d$ to the positions, such that $|d|$ is exactly the amount

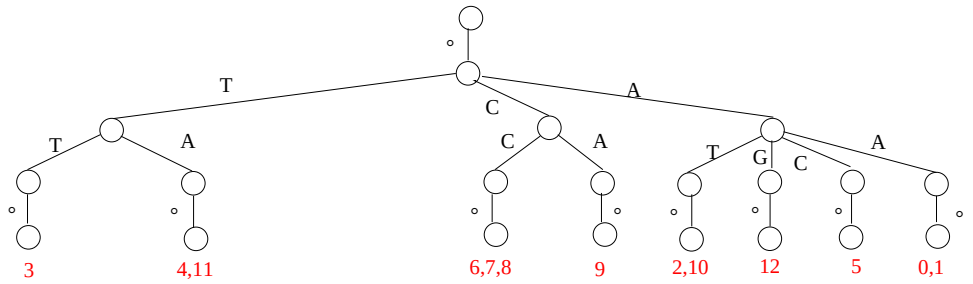


Figure 4: Trie for $\pi_{\mu'}$ where $\mu' = \texttt{0110}$

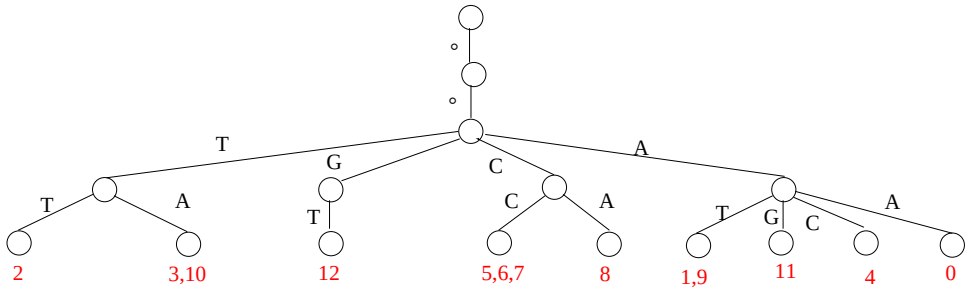


Figure 5: Trie for $\pi_{\mu''}$ where $\mu'' = \texttt{0011}$.

of shifted symbols in their masks needed to make them equal. For example, class $\{5, 6, 7\}$ can be obtained from $\{7, 8, 9\}$ by adding the integer $-2$ to the positions of the latter, since $\mu''$ can be transformed into $\mu$ shifting its symbols by 2 positions to the left.

Given two masks $\mu$ and $\mu'$ of the same length, we say that they are *equivalent* if they have the same number of 1s and $\mu$ can be obtained from $\mu'$ by a shift of the symbols, as long as only $\texttt{0}$s exceed the mask border and the empty positions are filled with $\texttt{0}$s (also the inverse holds). We define the following notations for sets of positions. Given two sets $C$ and $C'$ of text positions in $[0 \ldots n-1]$, we say that $C \equiv_d C'$ for an integer $d$ if two conditions hold:

1. for each $i' \in C'$ such that $0 \leq i' + d \leq n-1$, we have that $i' + d \in C$;
2. for each $i \in C$ such that $0 \leq i - d \leq n-1$, we have that $i - d \in C'$.

It is easy to see that, removing border effects, there is a one-to-one correspondence between the classes of the partitions induced by two equivalent masks that actually coincides with the relation $\equiv_d$, where $d$ is the size of the shift. In other words, the following result clearly holds.

**Lemma 5.** *For any two equivalent masks $\mu$ and $\mu'$, there exists an integer $d$ inducing a one-to-one correspondence between the classes of the partitions $\pi_\mu$ and $\pi_{\mu'}$ as follows: for each class $C \in \pi_\mu$, we have a unique class $C' \in \pi_{\mu'}$ such that $C \equiv_d C'$ (and vice versa).*

It is worth noting that, ignoring the first and the last $L-1$ text positions, $\pi_\mu$ and $\pi_{\mu'}$ have the same number of equivalence classes, the same number of elements, and any position shifted by an integer $d > 0$. Alternatively, we can extend the text with $L-1$ endmarker symbols to its left and its right. In this way, Lemma 5 makes partitions $\pi_{\mu'}$ redundant with respect to $\pi_\mu$.

Consequently, for each group of equivalent masks, we choose as representative the one having 1 as its first symbol. (Such a mask always exists except for the mask $\texttt{00}\cdots\texttt{0}$, which forms a trivial singleton group whose partition contains just one class made up of all the text positions.) We then eliminate the other masks in the class from our computation (steps 1–6), since we can always recover their partitions by adding a suitable integer $d$. For example, with $L = 4$, we now build explicitly only the representative masks when applying Lemma 4, which illustrates the fact that the number of representatives is *at most half* the number of equivalent masks: only those that start with a 1. The masks of length $\ell$ actually examined are those in the sets $F_\ell$ below ($\ell = 1, 2, 4$):

$F_1 = \{\texttt{1}\}$,

$F_2 = \{\texttt{11}, \texttt{10}\}$,

$F_4 = \{\texttt{1111}, \texttt{1110}, \texttt{1101}, \texttt{1100}, \texttt{1011}, \texttt{1010}, \texttt{1001}, \texttt{1000}\}$.

| *Figure* 1 | *Figure* 4 | *Figure* 5 |
|---|---|---|
| {4} | {3} | {2} |
| {5,12} | {4,11} | {3,10} |
| - | - | {12} |
| {7,8,9} | {6,7,8} | {5,6,7} |
| {10} | {9} | {8} |
| {3,11} | {2,10} | {1,9} |
| - | {12} | {11} |
| {6} | {5} | {4} |
| {0,1,2} | {0,1} | {0} |

Table 1: The partitions shown in Figures 1, 4, and 5.

### *3.3. Algorithm KMR for masks*

We now have all the ingredients for describing our algorithm that applies KMR to solve MMP. Given a string $T$, a length $L$ and a quorum $q$, we first compute the partition for $F_1 = \{1\}$ (since that for the mask $\mathtt{0}$ is trivial). Next, we compute $F_2$, $F_4$ and so on, as described in Sections 3.1–3.2. In particular, we just compute the representative for each equivalent class of masks, when running steps 1–6: for each pair of masks $\mu, \mu' \in F_\ell$ (not necessarily distinct), we build $F_{2\ell}$ by processing the concatenation of $\mu$ with all possible shifts of $\mu'$ (whose partitions we can quickly recover from that of $\mu'$). Summing up, we actually perform the following two steps:

1. Scan $T$ to construct the partition induced by relations $\equiv_1$ and build $F_1$.
2. Use Lemma 3– 5 to construct, successively, $F_2, F_4, F_8, \ldots, F_{2^r}$, and $F_L$, where $r$ is the largest value such that $2^r < L$.

**Theorem 6.** *Using the KMR approach, we can build the set $\mathscr{Q}(L,T,q)$ of masks for* MMP*, along with their induced partitions, in $O(2^L n)$ time and space.*

*Proof*: Given a length $L$, we have at most $\sum_{\ell=1}^{r} 2^\ell + 2^L < 2^{L+1}$ different masks $\mu$ to consider, and so as many equivalence relations $\equiv_\mu$, during the execution of the procedure referred to by Lemma 4. Since this latter guarantees that such execution takes $O(n)$ per mask, we have that KMR requires a total of $O(2^L n)$ time and space. □

In order to fully solve MMP, we need to select the maximal masks that will form the set $\mathscr{M}(L,T,q) \subseteq \mathscr{Q}(L,T,q)$. This maximality check on the set $\mathscr{Q}(L,T,q)$ computed in Theorem 6 can be done *a posteriori* using the LESS algorithm proposed in [21], with a cost that is linear on the average and quadratic in the worst-case with the size of the input. In our case, we need to apply it to $\mathscr{Q}$ in $O(L|\mathscr{Q}|^2)$ worst-case time. Given that we have up to $2^L$ masks of length $L$ in $\mathscr{Q}$, this method has $O(L2^L)$ average time complexity, and $O(L2^{2L})$ worst-case time complexity.

**Corollary 7.** *Using the KMR approach, we can solve* MMP *in $O(L2^L n)$ average time and space, and $O(L2^{2L} n)$ worst-case time and space.*

One drawback of the algorithm in Corollary 7 is that if a maximal mask $\mu$ is discovered and has a certain number $k$ of 1s in it, we have to generate anyway all the $\Theta(2^k)$ masks $\mu'$ such that $\mu' \preceq \mu$. In Section 4, we define a hybrid approach using KMR to evaluate the $P_T(\mu)$ predicate and a branch-and-bound strategy that exploits the anti-monotonicity of the above predicate, in order to obtain a better $O(2^L n)$ algorithm for MMP (see Theorem 9).

## 4. Adaptive KMR for Maximal Masks

In this section, we describe how to improve the worst-case complexity $O(L2^{2L}n)$ of Corollary 7. The reason for this bound is that, when using the KMR approach described in Section 3, we first generate all the masks of length $L$ having quorum and, then, select *a posteriori* those being maximal too. Our task can be better viewed in terms of the lattice $\mathscr{L} = \langle \{1, \mathtt{0}\}^L, \preceq \rangle$ of $2^L$ masks, introduced in Section 2.2 and illustrated in Figure 6. Given a mask $\mu$ having quorum, we would like to avoid to compute the partitions for mask $\mu'$, such that $\mu' \preceq \mu$. Recall that $\mu'$ is called predecessor of $\mu$, and the latter is called successor of $\mu'$. In general, given a mask, its successors are those masks that are reachable going upward in the lattice $\mathscr{L}$ and its predecessors are those reachable going downward.

Our idea can be summarized as follows. Suppose that we compute the set $\mathscr{Q}(L/2,T,q)$ of all masks of length $L/2$ that have quorum $q$, using Theorem 6, in $O(2^{L/2}n)$ time and space. (We store these masks using perfect hashing [36],

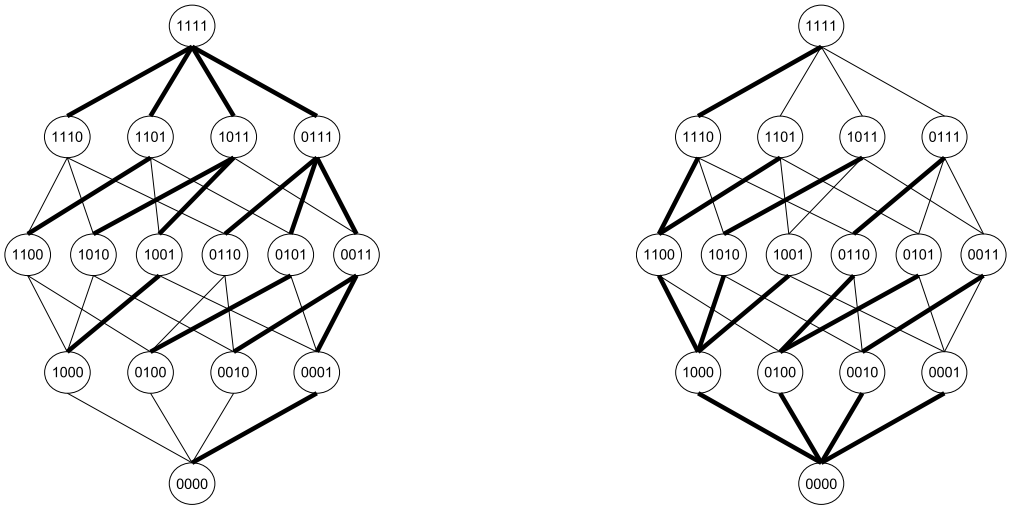


Figure 6: Two binomial (spanning) trees $B_L$, rooted at the top and at the bottom of lattice $\mathscr{L}$.

so each can be retrieved in $O(1)$ worst-case time; also, we assume without loss of generality that $L$ is an even number.) Instead of considering all $O(2^{L/2}) \times O(2^{L/2})$ possible concatenations of two masks in $\mathscr{Q}(L/2, T, q)$, we perform concatenation on demand, thus obtaining an *adaptive KMR approach*. The masks of length $L/2$, which we decide to concatenate in $O(n)$ time using Lemma 4, are chosen according to a suitable traversal of the lattice $\mathscr{L}$. We can employ two different pruning strategies, analogously to the *apriori* algorithm [27], where $\mu$ is the current mask of length $L$ in a traversal of $\mathscr{L}$:

1. if $\mu$ has the quorum, we do not check its predecessors in $\mathscr{L}$ since they have quorum but cannot be maximal (since they are less specific);
2. if $\mu$ has not the quorum, we do not check its successors because they cannot have quorum either (since having quorum is an anti–monotone property).

In the former case, we simulate the traversal of $\mathscr{L}$ by enumerating the masks of length $L$ starting from $1 \ldots 1$ and proceeding to less specific masks, which are downward in $\mathscr{L}$, while in the latter case we start from $0 \ldots 0$ and go upward looking for more specific masks. In both cases, whenever we need to build the partition $\pi_\mu$ for the current mask $\mu$, we split it as $\mu = \mu_1\mu_2$ such that $|\mu_1| = |\mu_2| = L/2$ (if $L$ is not a power of 2, we can proceed with an overlap of $\mu_1$ and $\mu_2$). If both $\mu_1, \mu_2 \in \mathscr{Q}(L/2, T, q)$, we apply Lemma 4 on them to check whether $\mu$ has quorum and compute its partition $\pi_\mu$. Otherwise, we declare that $\mu$ does not have quorum in $T$. We denote this checking operation by the predicate $P_T(\mu)$.

Since we apply Lemma 4 to at most $2^L$ masks of length $L$, the complexity of the algorithm is $O(2^L n)$. It remains to see how to enumerate the masks avoiding those that are non-maximal.

Since the mask lattice $\mathscr{L}$ is isomorphic to the powerset $\mathcal{P}(\{0, \ldots, L-1\})$ (see Section 2.2), our implicit traversal of $\mathscr{L}$ can be obtained by enumerating all the subsets of $\{0, \ldots, L-1\}$ visiting the corresponding *binomial tree* [37], which is also a spanning tree for $\mathscr{L}$. We recall that a binomial tree $B_k$ is an ordered tree representing the set $\mathcal{P}(\{0, \ldots, k-1\})$, and can be defined recursively as follows: $B_0$ consists of a single node and, for $k > 0$, $B_k$ consists of two binomial trees $B_{k-1}$, where the root of the former is added as the rightmost child to the root of the latter. Figure 6 shows two possible binomial trees $B_L$, both spanning $\mathscr{L}$. The first tree, shown on the left, is rooted at the top of the lattice and can be employed to implement the first pruning strategy, avoiding to visit the predecessors of the mask $\mu$ (downward in the lattice). The second tree, shown on the right, is rooted at the bottom of the lattice and can be employed to implement the second pruning strategy, avoiding to visit the successors of $\mu$. Since we want to identify the maximal masks, we opt for the first tree, starting from the top of the lattice $\mathscr{L}$.

Algorithm 1 shows the main steps when starting on top. Line 2 checks if mask $1 \ldots 1$ has quorum and, if this is so, that mask is the only one returned since any other masks would be less specific. Otherwise, create a queue $D$ containing $1 \ldots 1$. As long as $D$ is not empty, the main loop on line 6 selects a mask $\mu$ and generates one of its immediate predecessors $\mu'$ that are not yet visited. (In order to obtain $\mu'$, function next_immediate_predecessor systematically switches a 1 into 0 in $\mu$ at a time, so $\mu' \preceq \mu$ and they differ in one bit.) If $\mu'$ does not exist, then it means that all of $\mu$ predecessors have been already visited and $\mu$ is dequeued; otherwise, line 12 checks if $\mu'$ has quorum. If this is so, $\mu'$ is added to $\mathscr{M}$ if and only if a more specific mask is not already in it (lines 13–14). Otherwise, $\mu'$ is not

**Algorithm 1** *Top-Down Binomial-Tree Traversal of the Lattice* $\mathscr{L}$

**Input:** The predicate $P_T(\mu)$ for checking if $\mu$ has quorum $q$ in text $T$.
**Out:** The set $\mathscr{M}(L, T, q)$ of all maximal masks with quorum $q$ in $T$.

```
1:  M := ∅
2:  if P_T(1...1) then
3:     return {1...1}
4:  end if
5:  Queue D := {1...1}
6:  while not empty(D) do
7:     μ := top(D)
8:     μ' := next_immediate_predecessor(μ)
9:     if μ' = null then
10:       remove_top(D)
11:    else
12:       if P_T(μ') then
13:          if M does not contains μ'' s.t. μ' ≤ μ'' then
14:             M := M ∪ {μ'}
15:          end if
16:       else
17:          insert(μ', D)
18:       end if
19:    end if
20: end while
21: return M
```

enqueued, because anti–monotonicity of $P_T(\mu)$ guarantees that all of its predecessors have also quorum but they are less specific. Instead, if $\mu'$ has not quorum, it is enqueued because one of its predecessors could have quorum.

Implementing $D$ as a queue actually leads to a breadth-first visit of the binomial tree, because we visit a node of level $i + 1$ when all the node of level $i$ have been removed from the queue. Conversely, implementing $D$ as a stack leads to a depth first visit of the tree, but we have to pay more attention when adding a new mask to $\mathscr{M}$ (i.e. all of the predecessors of the mask must be removed).

Although in the worst case scenario almost all the masks of $\mathscr{L}$ must be checked, the pruning strategies can affect heavily the performance of the algorithms. Proceeding top-down with Algorithm 1, if all the interesting masks are close to the top, they are quickly found visiting only a small fraction of the $2^L$ masks of the lattice. For example, if only $1 \ldots 1$ has quorum, the predicate $P_T(\mu)$ is evaluated only once. (Similar considerations hold for the bottom-up traversal of $\mathscr{L}$.)

Unfortunately, Algorithm 1 cannot yet obtain $O(2^L n)$ time, since checking the condition in line 13 can take time $O(L|\mathscr{M}|) = O(L2^L)$ per mask, thus giving a total cost of $O(2^L(n + L2^L))$. We therefore discuss how to refine the breadth-first traversal of Algorithm 1 to get $O(2^L n)$ time.

Consider the lattice $\mathscr{L}$ and call level 0 the top mask $1 \ldots 1$, level 1 its predecessors, and so on, up to level $L$, which is the bottom mask $0 \ldots 0$. Also, consider the maximal masks in $\mathscr{L}$ (for the given predicate $P_T(\mu)$) and their predecessors.

**Definition 3.** *We call a mask* $\mu$ *on level* $i$ safe*, where* $0 \leq i \leq L$*, if* $\mu$ *is not predecessor of any maximal mask on levels* $0, 1, \ldots, i - 1$.

Note that a safe mask $\mu$ itself can be maximal (i.e. $P_T(\mu)$ holds), which is consistent with our definition of safeness.

Our goal is to modify Algorithm 1, so that it runs $P_T(\mu)$ only for safe masks $\mu$ on each level $i = 0, 1, \ldots, L$. The rationale is that, having traversed the first $i$ levels of the lattice $\mathscr{L}$ and having found the maximal masks on these levels, we cannot eliminate a priori any safe mask $\mu$ on level $i$ without first testing $P_T(\mu)$ on it. We show how to find safe masks on each level. Initially, for $i = 0$, the mask $1 \ldots 1$ is trivially safe.

During the top-down (breadth-first) traversal of $\mathscr{L}$, let us call $S_i$ the set of safe masks on level $i$. We enforce the invariant that $S_i$ is indeed the set of masks that are in queue $D$ on level $i$. The other masks on level $i$ are not of interest to us, since they surely have a maximal successor. We show how to produce $S_{i+1}$, so that the traversal can put the

masks from $S_{i+1}$ into the queue $D$ for the next level $i+1$. We need the crucial lemma below.

**Lemma 8.** *Let $M_i$ be the set of maximal masks on level i, where $0 \leq i \leq L$. Then, the following properties hold for each mask $\mu$:*

*(i) $\mu \in M_i$ if and only if $\mu \in S_i$ and $P_T(\mu)$ holds (hence, $M_i \subseteq S_i$).*
*(ii) $\mu \in S_{i+1}$ if and only if all the* immediate successors *of $\mu$ are in $S_i - M_i$.*

*Proof*: We consider the two properties in order. (i) Since $\mu$ is maximal, it has quorum while none of its successors can have the quorum, hence it is safe. The converse also holds because if $\mu$ has quorum and none of its successors is maximal (hence has quorum), then it is maximal by definition.

(ii) A mask $\mu$ on level $i$ either is maximal ($\mu \in M_i$), or it has quorum but is not maximal, or it is safe but not maximal ($\mu \in S_i - M_i$), since no other cases are possible. (In particular if $\mu$ is not safe, then it must have quorum since it is the predecessor of a maximal mask). The first implication ($\Rightarrow$) is equivalent to say that if there exists $\mu'$ immediate successor of $\mu$ such that $\mu' \notin S_i - M_i$ then $\mu \notin S_{i+1}$. From the above observation, it follows that $\mu' \notin S_i - M_i$ implies that $\mu'$ has quorum; hence, either $\mu'$ or one of its successors must be maximal, and $\mu$ cannot be safe having a maximal mask among its successors. To prove the second implication ($\Leftarrow$) it suffices to observe that if all the immediate successors $\mu'$ of $\mu$ are in $S_i - M_i$, they all have not the quorum and, by the anti-monotonicity of $P_T(\mu)$, none of their successors upward in the lattice can have the quorum. Hence, $\mu$ is safe. □

We now show how to exploit Lemma 8 during the traversal. First, since the queue $D$ stores the set $S_i$, we examine the masks $\mu \in S_i$ and perform the check with $P_T(\mu)$ by Lemma 8(i). Second, we remove $M_i$ from the queue $D$, which now stores the set $S_i - M_i$. In order to apply Lemma 8(ii), we recall that each of the immediate predecessors and immediate successors of a mask $\mu$ differs from $\mu$ in exactly one position. We generate all immediate predecessors of the masks in the queue $D = S_i - M_i$. In this way, we create a superset of $S_{i+1}$ from which we select only the masks that have all their *immediate successors* in the queue. We detail more this task and state its complexity.

Recall that $D$ denotes the set $S_i - M_i$ (this is indeed the content of the queue after the removal of the maximal motifs in it). We generate all the immediate predecessors of the masks in $D$ as follows. Given a mask $\mu \in D$ of arbitrary length $L$, for any position $j$ of a symbol `1` in $\mu$, we generate the immediate predecessors of $\mu$ that have all symbols equal to those in $\mu$ except that it contains symbol `0` in position $j$. Let $P_D$ be the multiset of immediate predecessors so built, which represent the predecessors of the masks in $D$. By Lemma 8, we have that $P_D$ is a superset of $S_{i+1}$ and a mask $\mu \in S_{i+1}$ if and only if $\mu$ has multiplicity $i+1$, that is, $\mu$ occurs $i+1$ times in the multiset $P_D$. For example, supposing $S_2 = \{$`10011`, `11001`, `10101`$\}$, we have $S_3 = \{$`10001`$\}$ since it appears three times in $P_D = \{$`00011`, `10001`, `10010`, `01001`, `10001`, `11000`, `00101`, `10001`, `10100`$\}$.

We can proceed in several ways for this checking. Either we sort the multiset $P_D$ and output the masks that appear (consecutively) $i+1$ times in the sorted multiset (e.g. Google's `mapReduce`) or we build a trie on the strings in $P_D$ and output those stored in the leaves with multiplicity $i+1$. From a theoretical point of view, we can build a perfect hash function $f$ on the distinct values in $P_D$ in $O(|P_D|)$ time and space [36]. In this way, given any two masks $\mu$ and $\mu'$, we have that $f(\mu) = f(\mu')$ implies $\mu = \mu'$. We then use an array of counters $C$ initially set to zero. For each mask $\mu \in P_D$, we increment $C[f(\mu)]$ by one. At the end, with a further scan of $P_D$, for each mask $\mu$, if $C[f(\mu)] = i+1$ then we output $\mu$ and reset $C[f(\mu)]$ to zero.

The overall cost for finding the sets $S_i$ for all levels $i$ can be bounded by $O(\sum_{i=0}^{L-1}(|S_i|\,L))$ time since $|P_D| \leq |S_i|\,(L-i)$ for level $i+1$. Using the fact the $|S_i| \leq \binom{L}{i}$, we obtain a cost of $O(\sum_{i=0}^{L-1}\binom{L}{i}L) = O(L2^L)$ for all the masks (instead of paying this cost for a single mask as before).

We can now apply Algorithm 1 in which we do *not* run the test in line 13 but, rather, we follow the traversal indicate by sets $S_i$ on each level $i$ of the lattice of $\mathscr{L}$. In this way the cost is dominated by $O(2^L n)$ due to checking $P_T(\mu)$ for the masks $\mu \in \cup_{i=0}^{L} S_i$, since the cost of generating the sets $S_i$ is $O(2^L L) = O(2^L n)$. The required space depends on the chosen visit strategy. Since we use the breadth-first traversal, the queue $D$ can contain all the $\binom{L}{i}$ masks on level $i$ (i.e. it may happen $|S_i| = \binom{L}{i}$). We have thus proved our main result.

**Theorem 9.** *Using the adaptive KMR approach, we can solve* MMP *computing $\mathscr{M}(L, T, q)$ in $O(2^L n)$ worst-case time and space.*

In order to evaluate the cost stated in Theorem 9, consider the following scenario in which, after preprocessing the text $T$ in polynomial time, we ideally check, in constant time per pattern, if any of the $(|\Sigma|+1)^L$ candidate patterns has

quorum and is maximal. We compare the cost of $O(n^{O(1)} + (|\Sigma| + 1)^L)$ thus obtained against the cost of $O(2^L n)$ stated in Theorem 9. Since $2^L n = O((|\Sigma| + 1)^L)$ when $L = \Omega(\log_{|\Sigma|} n)$, and $2^L n = O(n^{O(1+1/\log_2 |\Sigma|)})$ otherwise, we can conclude that the cost in Theorem 9 can be upper bounded by $O(n^{O(1+1/\log_2 |\Sigma|)} + \min\{2^L n, (|\Sigma| + 1)^L\})$. The latter is always better than the ideal bound of $O(n^{O(1)} + (|\Sigma| + 1)^L)$, that is, than constant-time enumerating and checking all the potential $(|\Sigma| + 1)^L$ patterns in $T$. As we discussed in Section 1, more sophisticated techniques that are the state of the art cannot improve, in the worst case, over the bound given in Theorem 9.

In order to get a cache-oblivious solution, note that both the construction of the safe masks and the concatenation of two masks of length $L/2$ needed to check $P_T(\mu)$ on each safe mask, require scanning and sorting. Since scanning of consecutive elements is trivially cache-oblivious, it suffices to employ a cache-oblivious sorting algorithm, whose cost is $sort(n) = \Theta\left(\frac{n}{B} \log_{M/B} \frac{n}{B}\right)$ block transfers [24, 25, 26, 23]. Note that the sorting cost is the dominating cost for each mask.

**Corollary 10.** *Using the adaptive KMR approach and a cache-oblivious sorting, we can build $\mathscr{Q}(L, T, q)$ and $\mathscr{M}(L, T, q)$ with $O(2^L\, sort(n))$ block transfers, where $sort(n)$ is the I/O complexity of sorting $n$ items.*

We conclude this section by observing that our bounds hold also for the case in which MMP has the additional requirement of reporting one representative mask for each equivalence class of masks. We recall from Section 3.2 that any two masks $\mu$ and $\mu'$ of the same length are *equivalent* if they have the same number of 1s and $\mu$ can be obtained from $\mu'$ by a shift of the symbols (and vice versa). In our introductory example, `1110` and `0111` are equivalent and we take the leftmost shift as the representative.

We can implement this extension with minor variations in our algorithms. In particular, we append $L - 1$ copies of a new special symbol \$ that is an endmarker for the text $T$, and so it does not belong $\Sigma$. We then run our algorithms on the resulting text $T\$\$\cdots\$$ and traverse the lattice $\mathscr{L}$ by considering only the masks having 1 as first symbol (see the subtree induced by these masks in each of the binomial trees shown in Figure 6). In this way, whenever we consider a mask, it is always the leftmost shift of its equivalence class, and we cover all this kind of masks.

**Corollary 11.** *Using the adaptive KMR approach, we can solve MMP modulo the equivalence between the masks, by selecting the leftmost shifts of the maximal masks in $\mathscr{M}(L, T, q)$ in $O(2^L n)$ worst-case time and space, with a cache complexity of $O(2^L\, sort(n))$ block transfers.*

## 5. Conclusions

We have introduced a new notion of motifs that succinctly represent the repeated patterns for an input sequence. We have described how to build the set of all maximal masks of length $L$ in $O(2^L n)$ time and space in the worst case, with a cache complexity of $O(2^L\, sort(n))$ block transfers. This bound is better than constant-time enumerating and checking all the potential $(|\Sigma| + 1)^L$ candidate patterns in $T$ after a polynomial-time preprocessing of $T$. It is also better than the bound obtained by the algorithms for mining frequent itemsets using hypergraph traversals. Interestingly, our algorithms work well also in external memory and in a distributed setting since they hinge on scanning and sorting sequential data. We plan to make a systematic experimental study to find practical incarnations of our ideas using real-life data sets. It is an open problem to obtain an output-sensitive algorithm, whose time complexity is of the form $o(2^L) + \Theta(|\mathscr{M}|)$ in the worst case.

*Acknowledgments*

We are in debt to Romeo Rizzi for having brought to our attention the connection of the problem to hypergraph traversals. We thank Khaled Elbassioni and Elias C. Stavropoulos for having kindly sent us their software for generating hypergraph transversals.

## References

[1] R. Eres, G. Landau, L. Parida, A combinatorial approach to automatic discovery of cluster-patterns, in: WABI, Vol. 2812 of LNCS, Springer, Budapest, Hungary, 2003, pp. 139–150.

[2] E. Eskin, From profiles to patterns and back again: a branch and bound algorithm for finding near optimal motif profiles, in: RECOMB '04: Proceedings of the eighth annual international conference on Resaerch in computational molecular biology, ACM, New York, NY, USA, 2004, pp. 115–124. doi:http://doi.acm.org/10.1145/974614.974630.

[3] M. Michael, F. Nicolas, E. Ukkonen, On the complexity of finding gapped motifs, CoRR abs/0802.0314.

[4] L. Parida, I. Rigoutsos, A. Floratos, D. Platt, Y. Gao, Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm, in: SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000, pp. 297–308.
[5] N. Pisanti, M. Crochemore, R. Grossi, M.-F. Sagot, Bases of motifs for generating repeated patterns with wild cards, IEEE/ACM Trans. Comput. Biology Bioinform. 2 (1) (2005) 40–50.
[6] E. Ukkonen, Structural analysis of gapped motifs of a string, in: MFCS, Vol. 4708 of LNCS, Springer, Ceský Krumlov, Czech Republic, 2007, pp. 681–690.
[7] S. Feng, E. Tillier, A fast and flexible approach to oligonucleotide probe design for genomes and gene families, Bioinformatics 23 (10) (2007) 1195–1202.
[8] L. Ilie, S. Ilie, Fast computation of good multiple spaced seeds, in: WABI, Vol. 4645 of LNBI, Springer, Philadelphia, PA, USA, 2007, pp. 346–358.
[9] G. Kucherov, L. Noé, M. Roytberg, Multiseed lossless filtration, IEEE/ACM Trans. Comput. Biology Bioinform. 2 (1) (2005) 51–61.
[10] G. Kucherov, L. Noé, M. Roytberg, A unifying framework for seed sensitivity and its application to subset seeds, J. Bioinform. and Comput. Biology 4 (2) (2006) 553–570.
[11] G. Kucherov, L. Noé, M. A. Roytberg, Subset seed automaton, in: CIAA, Vol. 4783 of LNCS, Springer, Praque, Czech Republic, 2007, pp. 180–191.
[12] M. Li, B. Ma, D. Kisman, J. Tromp, Patternhunter ii: Highly sensitive and fast homology search, J. Bioinform. and Comput. Biology 2 (3) (2004) 417–440.
[13] Y. Sun, J. Buhler, Designing patterns for profile hmm search, Bioinformatics 23 (2) (2007) 42–43.
[14] B. Brejová, D. Brown, T. Vinar, Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity, in: WABI, Vol. 2812 of LNCS, Springer, Budapest, Hungary, 2003, pp. 39–54.
[15] S. Burkhardt, J. Kärkkäinen, Better filtering with gapped q-grams, in: CPM, Vol. 2089 of LNCS, Springer, Jerusalem, Israel, 2001, pp. 73–85.
[16] K. Choi, F. Zeng, L. Zhang, Good spaced seeds for homology search, Bioinformatics 20 (7) (2004) 1053–1059.
[17] J.-E. Duchesne, M. Giraud, N. El-Mabrouk, Seed-based exclusion method for non-coding RNA gene search, in: COCOON, Vol. 4598 of LNCS, Springer, Banff, Canada, 2007, pp. 27–39.
[18] U. Keich, M. Li, B. Ma, J. Tromp, On spaced seeds for similarity search, Discr. Appl. Math. 138 (3) (2004) 253–263.
[19] Y. Sun, J. Buhler, Designing multiple simultaneous seeds for dna similarity search, J. Comput. Biology 12 (6) (2005) 847–861.
[20] R. Karp, R. Miller, A. Rosenberg, Rapid identification of repeated patterns in strings, trees and arrays, in: STOC, ACM, Denver, Colorado, USA, 1972, pp. 125–136.
[21] P. Godfrey, R. Shipley, J. Gryz, Algorithms and analyses for maximal vector computation, VLDB Journal: Very Large Data Bases 16 (1) (2006) 5–28.
[22] G.Battaglia, R.Grossi, R.Marangoni, N.Pisanti, Mining biological sequences with masks, submitted.
[23] M. Frigo, C. E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, in: 40th Annual Symposium on Foundations of Computer Science (FOCS), New York, NY, USA, 1999, pp. 285–297.
[24] G. S. Brodal, R. Fagerberg, Cache oblivious distribution sweeping, in: Proc. 29th International Colloquium on Automata (ICALP), Languages, and Programming (ICALP), Vol. 2380 of LNCS, Springer, Malaga, Spain, 2002, pp. 426–438.
[25] G. S. Brodal, R. Fagerberg, On the limits of cache-obliviousness, in: STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, ACM, New York, NY, USA, 2003, pp. 307–315. doi:http://doi.acm.org/10.1145/780542.780589.
[26] G. Franceschini, Proximity mergesort: optimal in-place sorting in the cache-oblivious model, in: SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004, pp. 291–299.
[27] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, 2001.
[28] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, R. S. Sharma, Discovering all most specific sentences, ACM Trans. Database Syst. 28 (2) (2003) 140–174. doi:http://doi.acm.org/10.1145/777943.777945.
[29] D. Gunopulos, H. Mannila, R. Khardon, H. Toivonen, Data mining, hypergraph transversals, and machine learning (extended abstract), in: PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, 1997, pp. 209–216. doi:http://doi.acm.org/10.1145/263661.263684.
[30] C. Berge, Hypergraphs: Combinatorics of Finite Sets, Vol. 45, Elsevier, 1989.
[31] T. Eiter, K. Makino, G. Gottlob, Computational aspects of monotone dualization: A brief survey, Discrete Applied Mathematics 156 (11) (2008) 2035–2049.
[32] D. Kavvadias, E. Stavropoulos, An efficient algorithm for the transversal hypergraph generation, J. Graph Algorithms and Applications 9 (2) (2005) 239–264.
[33] M. Fredman, L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, J. Algorithms 21 (3) (1996) 618–628. doi:http://dx.doi.org/10.1006/jagm.1996.0062.
[34] L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation, Discr. Appl. Math. 154 (16) (2006) 2350–2372.
[35] D. Knuth, The Art of Computer Programming: Sorting and Searching, 2nd Edition, Vol. 3, Addison Wesley, 1997.
[36] Z. J. Czech, G. Havas, B. S. Majewski, Perfect hashing, Theoret. Comput. Sci. 182 (1-2) (1997) 1–43.
[37] J. Vuillemin, A data structure for manipulating priority queues, CACM 21 (4) (1978) 309–315. doi:http://doi.acm.org/10.1145/359460.359478.