Efficient Computation of Statistics for Words with Mismatches

Cinzia Pizzi[1] *

Department of Information Engineering, University of Padova, Italy.

Abstract. Since early stages of bioinformatics, substrings played a crucial role in the search and discovery of significant biological signals. Despite the advent of a large number of different approaches and models to accomplish these tasks, substrings continue to be widely used to determine statistical distributions and compositions of biological sequences at various levels of details.
Here we overview efficient algorithms that were recently proposed to compute the actual and the expected frequency for words with k mismatches, when it is assumed that the words of interest occur at least once exactly in the sequence under analysis. Efficiency means these algorithms are polynomial in k rather than exponential as with an enumerative approach, and independent on the length of the query word.
These algorithms are all based on a common incremental approach of a preprocessing step that allows to answer queries related to any word occurring in the text efficiently. The same approach can be used with a sliding window scanning of the sequence to compute the same statistics for words of fixed lengths, even more efficiently.
The efficient computation of both expected and actual frequency of substrings, combined with a study on the monotonicity of popular scores such as z-scores, allows to build tables of feasible size in reasonable time, and can therefore be used in practical applications.

1 Introduction

The problems related to the computation of words statistics have an impact on the feasibility of the computation of both the actual and expected number of occurrences for words of interest, and consequently on measuring their over- or under- representation. In fact, frequent, common, over-/under- represented substrings within a sequence or among a set of sequences bring about information that can be exploited to discover or characterize common structures or functions of 'entities' (proteins, genes, etc.) showing some common behavior or properties.

Therefore, the problem of building such indexes arise ubiquitously in several fields of applications, for example motif discovery [17,9,18,2], protein classifications[13], and phylogeny [8,10,3] .

* Dipartimento di Ingegneria dell' Informazione, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy. `cinzia.pizzi@dei.unipd.it` This work was partially supported by the Research Prize "Avere Trent'anni", and by the Research Program of the University of Padova.

Functional and structural similarities among sequences can be posed in evidence by extracting significant patterns called *motifs*. From a biological point of view a motif can be defined as a signal that is responsible for functional or structural characterization, and shared among a set of biological related sequences. From the point of view of Computer Science a motif is a string defined over a given alphabet. In Bioinformatics and Computational Biology applications the choice of the alphabet depends upon the specific context (proteins, DNA, RNA). However, besides the standard symbols used to represent amino or nucleic acids, additional symbols can be used to relax the structure of the pattern and better describe biological objects of given functional or structural properties. In some cases, deterministic patterns, do not have enough expressive power to describe the specificity of the contributions of each symbol in any position of the site. Statistical matrices or graph-based models might offer a better framework in these cases. Several options have been considered during the past decades to model signals in biosequences, and to take into account for this intrinsic variability. Some of these models of choice are listed in Table 1.

solid word
wildcards-mismatches
insertion-deletion
generalized patterns
alignments
position weight matrices
hidden Markov models

Table 1. Various choices to model motifs in biological sequences. Starting from solid words the level of sensibility increases (allowing for variations), but the level of specificity consequently decreases, thus making more difficult the process of detection of the signal.

The choice of an appropriate model to describe motifs is a trade-off between the expressiveness of the model in describing particular biological properties and the efficiency of the algorithms that can be applied when that model is chosen.

Moreover, it is customary to partition the approaches to the problem of extracting unusually frequent or rare patterns from observed sequences into two main classes [7].

In the first class, the sample string is tested for occurrences of each and every motif in a family of *a priori* generated, abstract *models* (for example, [11]). This is methodologically sound but may pose severe computational burdens. The second class of approaches assumes that the search may be limited to substrings in the sample or to some more or less controlled neighborhood of those substrings (for example [12]). This may be less firm methodologically but brings about time and space savings.

Here we focus our attention on algorithms to compute statistics for *words* (i.e. substrings) with exactly or at most k mismatches that have extended the efficient framework for motif discovery, previously developed for solid words [2], to words with mismatches, under the assumption that the motifs occurs at least once exactly in the input sequence.

These studied were motivated by the observation that many biological studies, as those previously cited, are moving towards genomic size analysis. For this kind of analysis alignment-based techniques might be not feasible, and approaches based on composition analysis can be an alternative solution. However, this analysis still rely on the analysis of the frequency of solid words, and it is limited to short lengths m because of the exponential growth of the search space, which is proportional to $|\Sigma|^m$. Words with mismatches could be used for the same kind of analysis, allowing to examining longer words and including some variability. However, under these settings the intrinsic complexity of enumerative approaches would limit even more the lengths of words that could be examined. Thus the need of efficient incremental algorithms and compact approaches that will be described in the following sections.

In this paper we review some advanced techniques that were recently developed to efficiently compute both the expected number of occurrences (under different settings of the background distribution) and the actual number of occurrences for words with mismatches.

Sec.2 and Sec.3 describe, respectively, the algorithms for computing the expected frequency of words with mismatches under i.i.d. [4] and Markov chain distribution [16]. These algorithms are all based on a common incremental approach in a preprocessing step that allows to answer queries related to any word occurring in the text efficiently. The same approach can be used for scanning the input sequence with a sliding window to compute the statistics for words of fixed lengths, even more efficiently.

Sec.4 describes the adaptation of the latter technique to count the actual number of occurrences with mismatches [15].

In Sec.5 we will give an unified view of the algorithms that were previously discussed, and we will briefly sketch how these indexes can be used for compact pattern discovery, reducing considerably the size of the output.

We conclude with some considerations and open problems.

2 Computing the Expectation under i.i.d. hypothesis

We use capital letters to denote random strings and variables. In particular, $X = X_1X_2X_3 \dots X_n$ denotes a random text string produced by a source which emits symbols from Σ. Under i.i.d. assumptions the X_i are emitted independently and according to a given distribution $\mathcal{P} : \Sigma \rightarrow [0,1], \sum_{s\in\Sigma} p_s = 1$.

For an *observed* pattern $y = y_1y_2 \dots y_m$, the expected probability of y is calculated as follows. Let Z_i be an indicator variable such that:

$$Z_i = \begin{cases} 1 & \text{if } y \text{ occurs in } X \text{ starting at position } i \\ 0 & \text{otherwise} \end{cases}$$

The random variable $Z = \sum_{i=1}^{n-m+1} Z_i$ gives the number of occurrences of y in X. The expected number of occurrence of y is $E[Z|y]$. To compute this value one first need to compute the expected probability of y to occur at position i:

$$E[Z_i|y] = P[X_i = y_1, X_{i+1} = y_2, \ldots, X_{i+m-1} = y_m]$$

Because of independency:

$$E[Z_i|y] = P[X_i = y_1]P[X_{i+1} = y_2] \cdots P[X_{i+m-1} = y_m]$$

Because of identically distribution, the probability of one symbol to occur at some positions does not depend upon the position, but just to the probability assigned to the symbol by the given distribution. Hence:

$$E[Z_i|y] = p_{y_1}p_{y_2} \ldots p_{y_m}$$

.

As a consequence, the expected number of occurrences of y in X is:

$$E[Z|y] = E[\sum_{i=1}^{n-m+1} Z_i|y] = \sum_{i=1}^{n-m+1} E[Z_i|y] = (n-m+1)E[Z_i|y]$$

.

In the following, and whenever this causes no confusion, we will use $E[y]$ as shorthand for $E[Z|y]$ and $P(y)$ for $E[Z_i|y]$. Moreover, for a given string $x = x_1x_2 \ldots x_n$ we will indicate with $x[i,j]$ the substring $x_i \ldots x_j$.

2.1 Expectation of solid words

A solid word is a model in which only an exact match would give an occurrence of the word in the input sequence. Let $\mathcal{L}(y)$ be the set of occurrences of a string y in x. We have:

$$\mathcal{L}(y) = \{i : x[i, i + |y| - 1] = y\}$$

Problem 1: Given an input text x, generated under i.i.d hypothesis from a distribution $\mathcal{P}$, and two indexes b and e, compute the expected number of occurrences of the substring $x[b,e]$ in x.

Note that a naive approach to solve *Problem 1* requires $O(m)$ time, where $m = b - e + 1$, by a simple multiplication of the probabilities of the symbols. However, applying such an algorithm to the construction of an index can be prohibitive for large n. Suppose one wants to index all the substring of x. In this case the complexity using this approach is $O(n^3)$.

An alternative approach that can be used to efficiently compute indexes is presented, among other related results, in [2], and described below.

Consider an array A of size n. Each location $A[j]$ stores the probability of the prefix of x of length j, namely $P(x[1,j]) = \prod_{i=1}^{j} p_{x_i}$. This array can be filled in linear time in n, by posing $A[1] = p_{x_1}$, and $A[i] = A[i-1]p_{y_i}$ for $i = 2 \dots n$.

Given the array A and the indexes b and e the probability of $x[b,e]$ is given by $\frac{A[e]}{A[b-1]}$. In fact:

$$\frac{A[e]}{A[b-1]} = \frac{p_{x_1} \cdots p_{x_{b-1}} p_{x_b} \cdots p_{x_e}}{p_{x_1} \cdots p_{x_{b-1}}} = p_{x_b} \dots p_{x_e} = P(x[b,e])$$

This algorithm solves *Problem 1* in constant time after $O(n)$ time and space pre-processing. Using such algorithm to build an index for all the substrings of a string thus requires overall $O(n^2)$ time and space. Nevertheless, with the help of appropriate data structures and exploiting properties of monotonicity (see also Sec.5), the overall time and space needed to store over-represented solid words can be reduced to $O(n)$ without loss of information.

2.2 Expectation of words with k mismatches

When mismatches are introduced there is a new parameter to consider, their number, an integer $0 \leq k \leq m$ (the case $k = 0$ corresponds to solid words).

Throughout the rest of the discussion, a *motif* is a pair (y,k) where y is a substring of x, and k is the number of allowed mismatches for the occurrences of y in x. For shorthand we use the notation $y_{(k)}$ for (y,k). We can devise two frameworks characterizing the set of occurrences of (y,k):

1. Exactly k mismatches: $\mathcal{L}_E(y_{(k)}) = \{i : d(x[i \dots x_{i+m-1}], y) = k\}$
2. Up to k mismatches: $\mathcal{L}_U(y_{(k)}) = \{i : d(x[i \dots x_{i+m-1}], y) \leq k\}$

where $d(\cdot,\cdot)$ is the Hamming distance between the two strings taken as arguments. Thus, $y_{(k)}$ identifies a family of strings over Σ, whereas the same string belongs to more than one family.

Problem 2: Given an input text x, generated under i.i.d hypothesis from a distribution $\mathcal{P}$, and two indexes b and e, defining the beginning and end of a word $y = x[b,e]$ in the input sequence, compute the expected probability of $y_{(k)}$ to occur in x.

A naive approach. A naive approach to the computation of the expected number of occurrences of $y_{(k)}$ consists in the following steps:

1. generate the set of strings S that belong to $y_{(k)}$
2. compute the expected number of occurrences $E[Z|s] = \prod_{j=0}^{m} p_{s_i}$, where $s = s_1 \dots s_m$, for each $s \in S$
3. sum all the contributions computed at the previous step

In case of exactly k mismatches the set S has size $\begin{pmatrix} m \\ k \end{pmatrix} |\Sigma|^k$, whereas for at most k mismatches the size is $\sum_{i=0}^{k} \begin{pmatrix} m \\ i \end{pmatrix} |\Sigma|^i$.

Step 2) requires m operations for each of the strings in S. The overall complexity is therefore exponential in the number of mismatches.

The Correction Factor. A less expensive, incremental approach can be built on the notion of *correction factor* [4]. The *symbol correction factor* f_s is the quantity by which one need to multiply $E[y]$ (or, equally, $P(y)$) in order to take into account of a mismatch in correspondence of a symbol $s \in \Sigma$ occurring in y.

Example 1. As an example, consider the pattern $y = abaccaa$ on $\Sigma = \{a, b, c\}$, we have $P(y) = p_a p_b p_a p_c p_c p_a p_a$. A mutation in the first position would change y into *bbaccaa* or *cbaccaa*, with associated probability: $p_b p_b p_a p_c p_c p_a p_a + p_c p_b p_a p_c p_c p_a p_a = \frac{p_b}{p_a} p_a p_b p_a p_b p_b p_a p_a + \frac{p_c}{p_a} p_a p_b p_a p_b p_b p_a p_a = \frac{p_b + p_c}{p_a} p_a p_b p_a p_a p_b p_a p_a = \frac{p_b + p_c}{p_a} P(y)$.

The value $\frac{p_b + p_c}{p_a}$ is the symbol correction factor for the symbol a.

Definition 1. *(Symbol Correction Factor) Given an alphabet Σ, and an associated probability distribution $\mathcal{P}$ for its symbols, the symbol correction factor for $s \in \Sigma$ is:*

$$f_s = \sum_{s' \in \Sigma \setminus \{s\}} p_{s'} / p_s$$

In a general framework, where not just one, but (exactly) k mismatches occur, all the possible distribution of the k mismatches along the string y must be considered.

Definition 2. *(Word Correction Factor) Given an alphabet Σ and an associated probability distribution $\mathcal{P}$ for its symbols, the word correction factor $C_k(y)$ for a string y is the quantity by which we need to multiply the expected probability $P(y)$ to obtain $P_k(y)$, i.e. the expected number of occurrence of y when* exactly *k mismatches occur:*

$$P_k(y) = C_k(y) P(y)$$

The problem of computing the expected probability of $P_k(y)$ is then moved to the problem of computing its correction factor when exactly or at most k mismatches occur. In the following discussion we will use $C_k(y)$ to indicate the correction factor when *exactly* k mismatches occur, and $\bar{C}_k(y)$ to indicate the case when at most k mismatches occur. In should be noted that $\bar{C}_k(y) = \sum_{i=0}^{k} C_i(y)$, hence an efficient computation of $C_k(y)$ leads to an efficient computation of $\bar{C}_k(y)$.

By using the correction factor, we reduce the term $|\Sigma|^k$, to just k. Similarly, the number of comparisons are reduced from m to k. The time to compute the symbol correction factors is just $O(\Sigma)$. However, the dominant term in the complexity due to the binomial coefficient remains.

Besides being expensive, the naive approach does not lends itself to iterated computations where, e.g., the probabilities of occurrences with (up to) k mismatches of all substrings of x are seek for. The approach presented in [4] achieves this in $O(k^2)$ per iteration, hence in $O(k^2n^2)$ for indexing all the words, or $O(k^2n)$ for indexing all the words of a given fixed length. It requires an $O(kn)$ time pre-processing of x, which is described next.

2.3 $O(kn)$ input sequence pre-processing

The preprocessing for solid words builds an array that stores the probability of each prefix of the string. To store a similar information for words with mismatches a table T of size $k \times n$ is needed. The cell $T[i,j]$ stores the correction factor for having exactly i mismatches in the prefix of length j of x. Clearly, $T[0,j] = 1, \forall j$; and $T[i,j] = 0, \forall i > j$. This table can be built in $O(kn)$ time with a dynamic programming approach. In fact, to have i mismatches in the prefix of length j, either we have 1) i mismatches in the previous prefix, and a match at position j; or 2) $i-1$ mismatches in the previous prefix, and a mismatch at position j:

$$T[i,j] = T[i,j-1] + T[i-1,j-1]f_{x_j}, \forall j > i > 0$$

In summary, with f_{x_j} the correction factor of x_j, the following holds:

$$T[i][j] = \begin{cases} 1 & \text{if } i = 0 \text{ and } \forall j \\ 0 & \text{if } i \neq 0 \text{ and } j < i \\ f_{x_1} & \text{if } i = j = 1 \\ T[i][j-1] + T[i-1][j-1]f_{x_j} & \text{if } i > 0 \text{ and } j > i \end{cases}$$

This formula can be used also to directly compute the expected probability of $y = x[b,e]$ with an incremental approach. It suffices to build table T on y in $O(km)$ time. The space required is just $O(m)$ since T can be computed row by row and to compute row i only the previous one is necessary at every step. Finally, $T[k,m]$ stores the expected probability of $y_{(k)}$. However, since this approach depends on the length m of the pattern, it would not be efficient to compute indexes for a subset, or for all, the words in x.

Before giving a formal description of the methodology used to compute $C_k(y)$ from T, we show here an example to give a flavor of the overall idea behind this approach. We use the uniform distribution to give results that can be easily verified, but the approach is not at all limited to this particular distribution.

Example 2. Consider the uniform distribution, we have $T[i,j] = \binom{j}{i} f^i$.

Table 2 shows the content of T for the string *abaccaa* (in this particular case any string of length 7 will have the same T). Let us now compute the correction factor for $x[3,5]$ with $i = 1,2,3$ mismatches (we fill T up to row m, just to show its content, but we could have stop at 3 for this example).

The cell $T[1,5]$ stores the correction factor for having 1 mismatch from position 1 to position 5 in x. To get rid of the contribution of the mismatches occurring until position 2, we need to subtract $T[1,2]$:

$$C_1(3,5) = T[1,5] - T[1,2] = 3f$$

The cell $T[2,5]$ stores the correction factor for having 2 mismatch from position 1 to position 5 in x. In this case we need to get rid of two contributions: i) having 2 mismatches before position 3 (that is $T[2,2]$), and ii) having 1 mismatch before position 3 ($T[1,2]$) and 1 mismatch between positions 3 and 5 (which is $C_1(3,5)$):

$$C_2(3,5) = T[2,5] - T[2,2] - T[1,2]C_1(3,5) = 3f^2$$

For $k = 3$ mismatches we start from $T[3,5]$ and subtract the contribution of the 3 combinations of mismatches we are not interested in: 3 mismatches before position 3, 2 mismatches before position 3 and 1 within the substring, 1 mismatch before position 3 and 2 within the substring. All the terms are available:

$$C_3(3,5) = T[3,5] - T[3,2] - T[2,2]C_1(3,5) - T[1,2]C_2(3,5) = f^3$$

i	a	b	a	c	c	a	a
0	1	1	1	1	1	1	1
1	f	$2f$	$3f$	$4f$	$5f$	$6f$	$7f$
2	0	f^2	$3f^2$	$6f^2$	$10f^2$	$15f^2$	$21f^2$
3	0	0	f^3	$4f^3$	$10f^3$	$20f^3$	$35f^3$
4	0	0	0	f^4	$5f^4$	$15f^4$	$35f^4$
5	0	0	0	0	f^5	$6f^5$	$21f^5$
6	0	0	0	0	0	f^6	$7f^5$
7	0	0	0	0	0	0	f^7

Table 2. Table T computed for $x = abaccaa$ and $k = 7$ under the uniform distribution.

Processing. Since the indexes play a crucial role in the definition of the formulas, to avoid clutter in the notation, we will use $C_k(b,e)$ to indicate $C_k(x[b,e])$.

Let then $C_k(b,e)$ be the correction factor for $y = x[b,e]$: $P_k(y) = C_k(b,e)P(y)$. We have:

$$C_k(b,e) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{if } k > e - b + 1 \\ T[k][e] - \sum_{i=0}^{k} T[k-i][b-1]C_i(b,e) & \text{if } 0 < k \leq m \end{cases}$$

The first two conditions are obvious. The last formula can be explained with the help of Fig.1.

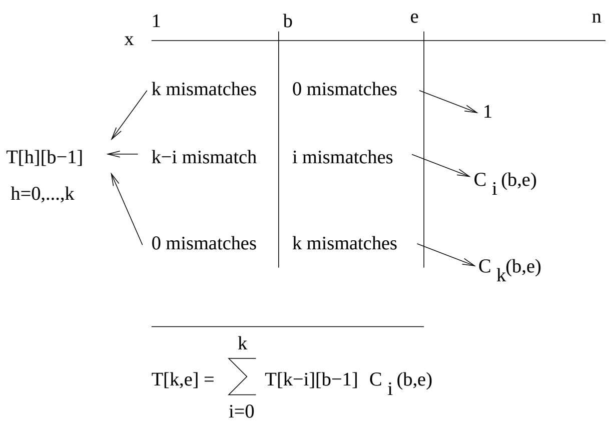

Fig. 1. Distribution of k mismatches.

Given b and e the input string x can be divided in three parts: $x[1, b-1]$, $x[b, e]$, $x[e+1, n]$. The last part is of no interest for the computation, so let us concentrate on the first two. $T[k, e]$ holds the correction factor for the prefix of $x[1, e]$ with exactly k mismatches. Since this prefix is divided in two parts, the value of $T[k, e]$ can be seen as given by the sum of the correction factors of all possible partitioning of the k mismatches along these two parts. These are exactly k: i mismatches in $x[b, e]$ and $k-i$ in $x[1, b-1]$, for $i = 0 \dots k$:

$$T[k, e] = \sum_{i=0}^{k} T[k-i, b-1] C_i(b, e)$$

From this formula one can extrapolate:

$$C_k(b, e) = T[k, e] - \sum_{i=0}^{k-1} T[k-i, b-1] C_i(b, e)$$

that is the the correction factor for $x[b, e]$ when exactly k mismatches occur (the right side should be divided by $T[0, b-1]$, but this is equal to 1, so it can be simplified). Here follows a pseudo code for the algorithm:

The algorithm takes time quadratic in k. In fact, line 7 requires constant time since all the terms are available at any step of the iteration. The complexity is then given by the two nested cycles: $O(k^2)$.

In conclusion, after $O(kn)$ time and space pre-processing of the text, it is possible to obtain in $O(k^2)$ time, from given initial and final position of any substring of the text, the correction factor for that substring. Combined with probabilities and length, this also yields the desired probability and expected frequency with mismatches for that substring, at no extra cost.

```
CORRECTION_FACTOR($b, e$)
int $C[0 \dots k]$ correction factors
$C[0] \leftarrow 1$
for $i \leftarrow 1$ to $k$
        do
            $C[i] \leftarrow T[i, e]$
            for $j \leftarrow 0$ to $i - 1$
                do $C[i] \leftarrow C[i] - T[i - j, b - 1]C[j]$

```

Fig. 2. Pseudo code for computing the correction factors $C_i(b, e)$ of a substring $x[b, e]$ for $i = 0 \dots k$.

The case for all substring of fixed length *m*. For any fixed length m, the algorithm above supports also the computation of correction factors for all m-character substrings of text x in $O(nk^2)$ time. However, a substantial improvement in both time and space for this case is possible. Another algorithm presented in [4] proceeds by dynamically adjusting the desired values in a sliding window of any given fixed length m.

First, note that a recurrence similar to the one used to compute the correction factor $T[i, j]$ from $T[i-1, j]$ and $T[i-1, j-1]$ can be used to derive the correction factor of a string $y' = x[b, e+1]$ from the correction factor of the string $y = x[b, e]$. Indeed, we have $C_0(b, e + 1) = 1$ and, for $k > 0$:

$$C_k(b, e + 1) = C_k(b, e) + C_{k-1}(b, e)f_{x[e+1]} \tag{1}$$

Similarly, we can extend the word to the left, thus computing the correction factor for $y = x[b, e]$ from those of $y' = x[b + 1, e]$, using the trigger condition $C_0(b, e + 1) = 1$, and $C_k(b, e) = C_k(b + 1, e) + C_{k-1}(b + 1, e)f_{x_b}$ for $k > 0$.

This latter formula can be rewritten to explicit $C_k(b + 1, e)$:

$$C_k(b + 1, e) = C_k(b, e) - C_{k-1}(b + 1, e)f_{x_b} \tag{2}$$

Triggered by the "universal" initial condition $C_0 = 1$, eq. (2) enables us to extract in succession the $C_i(b+1, e)$'s values from the $C_i(b, e)$'s, for consecutive values of $i = 1, 2, ..., k$. Next, applying eq.(1) to the indexes $(b+1, e)$ the $C_i(b+1, e+1)$'s can also be computed.

The process takes twice k steps and requires knowledge of as many auxiliary values, hence the computation for all m-character strings completes in $O(nk)$ time and $O(k)$ auxiliary space.

3 Expectation under Markov chain distribution

For the case of a background distribution following an order p Markov chain an incremental approach similar to what was seen for i.i.d. is possible. However, the intrinsic dependency between symbols due to the nature of the distribution makes the updating formulas more involved. In this section we describe the naive algorithm, along with three other more advanced techniques. The general problem we want to solve is:

Problem 3: Given an input text x, generated by a distribution $\mathcal{P}$ that follows a Markov chain of order p, and two indexes b and e, defining the beginning and end of a word $y = x[b, e]$ in the input sequence, compute the expected probability of $y_{(k)}$ to occur in x.

Moreover, as before, we put more emphasis, on techniques that can be applied to compute indexes that involve the calculation of the expected number of occurrences for a subset of substring in an input sequence (for example all the substring of a given fixed length).

3.1 The naive algorithm

The naive approach consists in the enumeration of all the possible ways to distribute the k mismatches among the m position of the pattern, and next compute the expectation for all of them. Hence it has the same time complexity than for the i.i.d. naive approach that is exponential in the number of mismatches.

3.2 Algorithm 1

A more efficient algorithm to compute the expectation for a single string was proposed in [16]. We give the description for a Markov chain of order $p = 1$ to simplify the discussion, and then give the generalization.

Let M_{s_i} and $M_{\bar{s}_i}$ be two $|\Sigma| \times |\Sigma|$ matrices that are associated with the probability of having, respectively, a match or a mismatch in correspondence of a symbol $s_i \in \Sigma$. The two matrices are defined as follows:

$$M_{s_i} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \\ P_{s_i|s_1} & P_{s_i|s_1} & \dots & P_{s_i|s_{|\Sigma|}} \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad M_{\bar{s}_i} = \begin{bmatrix} P_{s_1|s_1} & P_{s_1|s_2} & \dots & P_{s_1|s_{|\Sigma|}} \\ \dots & \dots & \dots & \dots \\ P_{s_{i-1}|s_1} & P_{s_{i-1}|s_2} & \dots & P_{s_{i-1}|s_{|\Sigma|}} \\ 0 & 0 & \dots & 0 \\ P_{s_{i+1}|s_1} & P_{s_{i+1}|s_2} & \dots & P_{s_{i+1}|s_{|\Sigma|}} \\ \dots & \dots & \dots & \dots \\ P_{s_{|\Sigma|}|s_1} & P_{s_{|\Sigma|}|s_2} & \dots & P_{s_{|\Sigma|}|s_{|\Sigma|}} \end{bmatrix}$$

Suppose we want to compute the expectation of a string y of length m with k mismatches. Fig.3 shows a direct acyclic graph (DAG) with two source nodes S and S'. The DAG is build in such a way to have exactly $k \times (m - k) + 1$ nodes.

Each horizontal edge represents a match, and it is labeled with a C, meaning "correct" (in the figure not all the edges are label for clearness of the representation). Each vertical edge represents a mismatch, and it is labeled with an M.

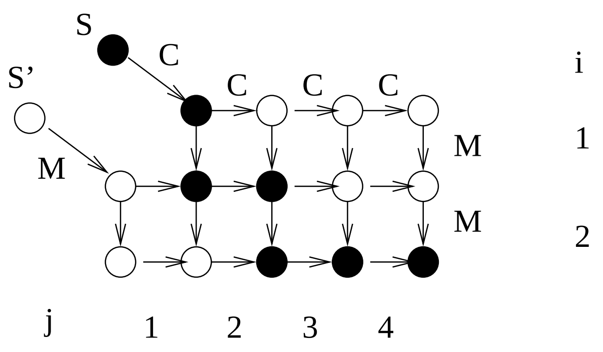

Fig. 3. A DAG representing the distribution of $k = 2$ mismatches in a string y of length 6. Each vertical edge is a mismatch (M), while each horizontal edge is a match (C=correct). Each path from one of the sources S or S' to the lower-right node represent a distribution of mismatches. For example the path in black shows the pattern for $y_1\bar{y}_2y_3\bar{y}_4y_5y_6$, i.e. the two mismatches occur at position 2 and 4.

An edge that it is reached from one of the sources after traversing $d - 1$ edges, is associated with the symbol at position d in y, namely y_d. Depending whether the label is C or M this edge will use the matrix M_{y_d} or $M_{\bar{y}_d}$ at computation time.

Node S is the initial node when it is assumed that the the first symbol in the string is not the site for a mismatch. The node S' plays the complementary role. Every path from one of the source nodes to the lower-right corner node has length m, and represents a way to distribute the k mismatches in the pattern y. To reach the bottom-right node from either node S or S', one needs to traverse exactly k vertical edges, and exactly $m - k$ horizontal edges, hence describing a pattern of length m with exactly k mismatches.

Apart from nodes S and S', all the other nodes store a vector $\mathbf{t}$ of size $|\Sigma|$. For a node that is in a path that traverse i vertical edges and j horizontal edges, $|\mathbf{t}|_1$ gives the expected probability of having i mismatches in the prefix of y of length $i + j$. Each component t_s holds the term due to the contribution of having symbol $s \in \Sigma$ as the last symbol considered in the path.

The values of the vectors for the other nodes are computed as follows. Let $\mathbf{t}_{i,j}$ be the vector associated with the node that has been reached by traversing i vertical edges and j horizontal edges.

The incoming edges to this node either come from node reached by a path with $i - 1$ vertical edges (and the same number j of horizontal edges) or from a node reached after traversing i vertical edges and $j - 1$ horizontal edges. So the incoming vectors would be $\mathbf{t}_{i-1,j}$ and $\mathbf{t}_{i,j-1}$. Before summing these vectors, it is needed to take into account of the last edge that has been traversed and multiply each vector by $M_{\bar{y}_{i+j}}$ and $M_{y_{i+j}}$ respectively:

$$\mathbf{t_{i,j}} = M_{y_{i+j}}\mathbf{t_{i-1,j}} + M_{\bar{y}_{i+j}}\mathbf{t_{i,j-1}}$$

Example 3. Consider the string $y = abb$ defined over an alphabet $\Sigma = \{a, b, c, d\}$. The vectors associated with nodes following S and S' respectively are:

$$\mathbf{t} = [p_a, 0, 0, 0] \text{ and } \mathbf{t} = [0, p_b, p_c, p_d]$$

The value of $\mathbf{t}$ for the first black node following S is $[0, p_b|p_a p_a, 0, 0]$.

The values must be computed row by row (or column by columns). There are $(m-k+1)\times(k+1)$ vectors to compute, and each of them requires $O(|\Sigma|^2)$ time (due to the matrix product). The total complexity is $O(mk|\Sigma|^2)$. This algorithm is polynomial in both the string length and the number of mismatches.

The procedure we just described can be generalized for a Markov chain of order p. In this case the matrix M has size $|\Sigma|^p \times |\Sigma|$, so the complexity is $O(mk|\Sigma|^{p+1})$.

3.3 Algorithm 1bis

In [6] a different approach to the computation of the expected number of occurrences for a Markov chain distribution of order p is presented. However it also takes $O(mk|\Sigma|^{p+1})$ to compute the expectation with k mismatches for a word of length m.

If we apply this algorithm or the previous one, to any word in the input sequence we have a total time complexity of $O(n^3k|\Sigma|^{p+1})$. We will discuss in the following subsection how it is possible to devise an algorithm that after $O(nk|\Sigma|^{p+1})$ pre-processing, can compute the same information in $O(k^2n^2|\Sigma|)$ [16].

3.4 Algorithm 2

In this section we sketch an algorithm that apply the same incremental approach seen for the i.i.d. case in Sec.1 to build a table for each prefix of x. Next it uses the table for fast computation of the expected probability of any words that occur in the text.

The major difficulty is that particular care needs to be taken for dealing with the contribution at the "breaking" index b. Moreover, the expectation need to be store in a vector to keep the components separated according to the last symbol that is sought. The correction factor in this case does not help, because it aggregates the contribution of different symbols for a mismatch at some position.

The algorithm, after $O(nk|\Sigma|^{p+1})$ preprocessing, computes the expectation with k mismatches of any word in the text in $O(k^2|\Sigma|)$ time. We explain the algorithm for the case $p = 1$ for ease of discussion.

We introduce some notation, and recall previous definitions for ease of readability. $P_k(y)$ is the expected probability of a string y to occur in a given input sequence x with k mismatches. $x[b, e]$ indicates the substring $x_b \dots x_e$, x_b indicates the symbol at position b, and X^j represent the prefix of x of length j, i.e. $x_1 \dots x_j$.

3.5 Preprocessing

In this phase a table V is built, such that the entry $V^{i,j}$ is a vector of size Σ:

$$V^{i,j} = (v_t^{ij})_{t=1\ldots|\Sigma|}$$

The single entry v_t^{ij} is the contribution to the expected probability $P_i(X^j)$ of the prefix X^j with i mismatches, when considering strings $w_1 \ldots w_j$ in the neighborhood $X_{(i)}^j$ such that $w_j = s_t$, for $s_t \in \Sigma$.

$|V^{i,j}|_1$ gives the expected probabilities of the prefix X_j to occur in x with exactly i mismatches:

$$P_i(X_j) = \sum_{t=1}^{|\Sigma|} v_t^{ij}$$

The values of $V^{i,j}$ are computed by dynamic programming:

$$V^{i,j} = M_{x_j} V^{i,j-1} + M_{\bar{x}_j} V^{i-1,j-1}$$

Table V is built in $O(nk|\Sigma|^2)$ for Markov chains of order $p = 1$. The generalization takes $O(nk|\Sigma|^{p+1})$.

For the computation it is also needed an array A in which $A[i]$ stores the probability of the prefix of length i according to the given symbols distribution. This array is built in $O(n)$: $A[1] = p_{x_1}$, $A[j] = p_{x_j|x_{j-1}} A[j-1]$ for $1 < j \leq n$.

3.6 Processing

Given table V and array A, a pair of indexes (s, e), and a number of mismatches k, let us compute the expectation with k mismatches of $x[s, e]$.

The contributions to the expectation of $x[1, e]$ with i mismatches are stored in $V^{i,e}$. Starting from these values one need to get rid of the contribution to the expectation of the prefix $x[1, s-1]$.

Position s is critical, since a mismatch there will affect both $p_{s+1|s}$ and $p_{s|s-1}$.

To manipulate this contribution another vector $U^{ie} = (u_t^{ie})$ is build from $P_i(X^e)$, where each component u_t^{ie} stores the contribution to $P_i(X^e)$ when the first symbol of the pattern we are interested in (i.e. position s in x) is the symbol s_t.

The corresponding contribution to the expectation between s and e will be denoted by $C_{s_t}^{i,s,e}$. It is straightforward to observe that $C^{0,s,e} = A[e]/A[s]$ and $P_0(x[s, e]) = C^{0,s,e} p_{x_s}$.

Starting from this condition the number of mismatches in increased at each iteration, till the desired number k is reached.

At each iteration i the values of U_{ie} and $C^{i,s,e}$ are computed. The full description of the calculation involves long formulas for which we suggest a reading of the original paper. Here we report the results just for $k = 2$ to give an idea of the approach, and then give the generalization.

Example 4. (Computing $U^{2,e}$). We assume to have at hand $V^{1,e}$, $V^{2,e}$, $U^{1,e}$, $C^{1,s,e}$, that were computed at previous steps.

If $s_t \neq x_s$ there is 1 mismatch at position s, so the contributions to the expectation of $x[1,e]$ with 2 mismatches that involve s_t are:

$1 \dots s-1$	s	$s+1 \dots e$
1 mismatch	x	0 mismatch
0 mismatch	x	1 mismatch

$$\begin{aligned} u_t^{2,e} &= (\textstyle\sum_{i=1}^{|\Sigma|} v_i^{1,s-1} p_{s_t|s_i}) p_{x_{s+1}|s_t} (p_{x_{s+2}|x_{s+1}} \cdots p_{x_e|x_{e-1}}) \\ &\quad + (\textstyle\sum_{i=1}^{|\Sigma|} c_i^{1,s+1,e} p_{s_i|s_t})(p_{x_1} \cdots p_{x_{s-1}|x_{s-2}}) p_{s_t|x_{s-1}} \\ &= v_{s_t}^{1,s} c_{s_t}^{1,s,e} + c_{s_t}^{1,s,e} c_{s_t}^{1,1,s} \end{aligned}$$

The computation takes $O(1)$ for each $s_t \neq x_s$.

If $s_t = x_s$ the 2 mismatches must occur at positions other than s.

$1 \dots s-1$	s	$s+1 \dots e$
2 mismatches	=	0 mismatch
1 mismatch	=	1 mismatch
0 mismatch	=	2 mismatches

This component can be computed directly:

$$u_t^{2,e} = \sum_{t \in \Sigma} v_i^{2,e} - \sum_{t : s_t \neq x_s} u_t^{2,e} = P_2(X^e) - \sum_{t : s_t \neq x_s} u_t^{2,e}$$

The overall time needed to compute $U^{2,e}$ is $O(|\Sigma|)$.

Example 5. (Computation of $C^{2,s,e}$)

The probability $P_2(x[1,e])$ of $x[1,e]$ with 2 mismatches is:

$$P_0(x[1,s-1])P_2(x[s,e]) + P_1(x[1,s-1])P_1(x[s,e]) + P_2(x[1,s-1])P_0(x[s,e])$$

We want to compute $P_2(x[s,e])$:

$$P_2(x[s,e]) = \frac{P_2(x[1,e]) - P_2(x[1,s-1])P_0(x[s,e]) - P_1(x[1,s-1])P_1(x[s,e])}{P_0(x[1,s-1])}$$

For ease of notation we assign letters to the factors:

$$P_2(x[s,e]) = [A - BC - DE]/F$$

The factor A is taken from $U^{2,e}$. The product BC assumes that x_s is correct, so it will contribute only to the value of $u^{2,e}_{x_s}$.

$$\sum_{i=1}^{|\Sigma|} v_i^{2,s-1} p_{x_s|x_{s_i}} (p_{x_{s+1}|x_s} \cdots p_{x_e|x_{e-1}}) = v^{2,s}_{x_s} C^{0,s,e}_{x_s}$$

In the product DE there might or might not be a mismatch at position s. Hence the two cases are considered separately.

1. s is a site of a mismatch.
   Then the other mismatch must occur in the prefix of length $s-1$. The contribution to subtract to each $s_t \neq x_s$ is:

$$(\sum_{i=1}^{|\Sigma|} v_i^{1,s-1} p_{s_t|s_i}) p_{x_{s+1}|s_t} (p_{x_{s+2}|x_{s+1}} \cdots p_{x_e|x_{e-1}}) = v^{2,s}_{s_t} C^{1,s,e}_{s_t}$$

2. s is not the site for a mismatch.
   Then one mismatch occurs in the prefix of length $s-1$ and the other must occur between $s+1$ and e. Such contribution is:

$$\sum_{i=1}^{|\Sigma|} v_i^{1,s-1} p_{x_s|s_i} \sum_{j=1}^{|\Sigma|} p_{x_j|x_s} C^{1,s+1,e}_{s_j} = v^{1,s}_{x_s} C^{1,s,e}_{x_s}$$

In summary:

$$C^{2,s,e} = \begin{bmatrix} (u_1^{2,e} - v^{2,s}_{s_1} C^{1,s,e}_{s_1})/p_{s_1|x_{s-1}} \\ (u_2^{2,e} - v^{2,s}_{s_2} C^{1,s,e}_{s_2})/p_{s_2|x_{s-1}} \\ \cdots \\ (u^{2,e}_{x_s} - v^{2,s}_{x_s} C^{0,s,e}_{x_s} - v^{1,s}_{x_s} C^{1,s,e}_{x_s})/p_{x_s|x_{s-1}} \\ \cdots \\ (u^{2,e}_{|\Sigma|} - v^{2,s}_{s_{|\Sigma|}} C^{1,s,e}_{s_{|\Sigma|}})/p_{s_{|\Sigma|}|x_{s-1}} \end{bmatrix} \frac{1}{p_{x_1} \cdots p_{x_{s-1}|x_{s-2}}}$$

3.7 Generalization for i mismatches

When i mismatches occur, if $s_t \neq x_s$ there is 1 mismatch at position s. The other $i-1$ mismatches can be distributed in any of the $i-1$ combinations (Table - left) that we need to subtract to the components of $U^{i,e}$. If $s_t = x_s$ the ways to distribute the mismatches are i (Table - right).

The time complexity for the component $s_t = x_s$ when i mismatches occur needs i operations, while for the remaining components it needs $i-1$. In both cases it is $O(i)$, so the overall complexity is $O(i|\Sigma|)$.

$1 \dots s-1$	s	$s+1 \dots e$
1 mismatch	x	i-2 mismatches
2 mismatches	x	i-3 mismatches
...	x	...
i-1 mismatches	x	0 mismatches

$1 \dots s-1$	s	$s+1 \dots e$
1 mismatches	=	i-1 mismatches
2 mismatches	=	i-2 mismatches
...	=	...
i mismatches	=	0 mismatches

To compute the expectation for a number of mismatches k, the expectations for a number of mismatches i for $i = 0 \dots k-1$ is needed. The total complexity is then:

$$\sum_{i=1}^{k} i|\Sigma| = |\Sigma|\frac{k(k+1)}{2} = O(|\Sigma|k^2)$$

The computation of the expectation of all the words in a text this takes $O(kn^2|\Sigma|^2 + nk|\Sigma|^{p+1})$.

4 Counting occurrences with mismatches

Problem 4: Given a string x of length n, a positive integer m and a number of mismatches $k \leq m$, find the number of occurrences with at most (or exactly) k mismatches for all the substrings (words) of length m in x.

Several algorithms are described in literature to solve the slightly different k-mismatch problem that consists of: given a string x, a query y and a number of mismatches k, find all the occurrences of y in x with at most k mismatches (see, for example, references in [14]). The best known algorithm solve this problem in $O(\sqrt{k \log k}n)$ [1]. By applying such algorithm to all the words of fixed length m in x the complexity would be $O(\sqrt{k \log k}n^2)$, which have a factor that depends on the number of mismatches. In [15] an algorithm is proposed to solve the problem in $O(n^2)$, independent on k.

The algorithm exploits the relation between two consecutive substrings of fixed length. Given a word $x[i, i+m-1]$ of length m and its successor $x[i+1, i+m]$ it can be easily seen that they share most of their composition (see Fig.4).

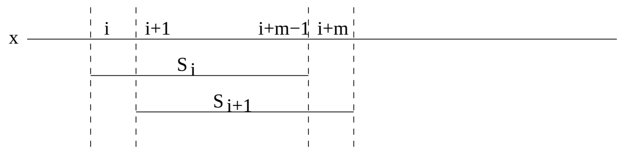

Fig. 4. Two consecutive substrings of x of length m starting at position i (substring S_i) and $i+1$ (substring S_{i+1}), respectively. They share the common substring $x[i+1, i+m]$.

The number of mismatches with which $x[i+1, i+m]$ occurs in x can be deduced given the same information for $x[i, i+m-1]$.

Suppose $x[i, i+m-1]$ occurs with h mismatches at position j in x. If we align $x[i+1, i+m]$ at position $j+1$ (see Fig.5) its number of mismatches is computed in constant time with the updating formula:

$$h - (x_i \neq x_j?) + (x_{i+m} \neq x_{j+m}?)$$

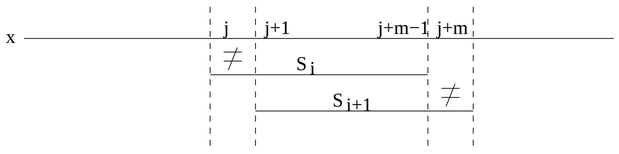

Fig. 5. Alignment of S_i and S_{i+1} at some position j and $j+1$, respectively. If S_i occurs in j with h mismatches, S_{i+1} occurs in $j+1$ with a number of mismatches h' that falls in the range $h \pm 1$.

The idea is to build an $n \times n$ table T, in which $T[i, j]$ stores the number of mismatches of the occurrence of the substring $x[i, i+m-1]$ starting at position j. In other words $T[i, j] = d(x[i, i+m-1], x[j, j+m-1])$, where $d(\cdot, \cdot)$ is the Hamming distance between the two strings. Table T is symmetric, hence one need to compute its value explicitly only for $j > i$ (for $j = i$ the value is clearly 0). Triggered by this condition, after computing the first row (even with a naive $O(mn)$ algorithm), it is possible to compute the values for the second row, and so on, in constant time per position, hence $O(n)$ time per row. When the value is equal to (or at most) k, a counter for the row i and for row j is incremented. At the end of the computation, that takes overall $O(n^2)$ time such counter will contain, for each word, its number of occurrences with exactly (up to) k mismatches.

The space required to store the table is $O(n^2)$, however, if the interest is only in the number of occurrences, and not in the actual localization, then the space can be reduce to $O(n)$ since in order to compute row i only the values of row $i-1$ are needed.

5 An unified view and its applications

The algorithms we overview in this paper are all based on an incremental approach that builds the probability and/or the counting of a word with mismatches based on the (already computed) values of its prefix(es). The idea is that, given a string $w = va$, where $a \in \Sigma$, the computation of a function $g_k(w)$ that holds some statistics involving k mismatches, can be done knowing the values of $g_{k-1}(v)$, $g_k(v)$, and a score function $f(a)$ assigned to the symbol a:

$$g_k(w) = h(g_{k-1}(v), g_k(v), f(a))$$

The one-symbol extension can be done equally to the left rather than to the right to obtain the statistics of $w = av$. By an opportune manipulation of this formula one can obtain a formula to compute the statistics of v with k mismatches:

$$g_k(v) = h'(g_{k-1}(v), g_k(w), f(a))$$

The combination of these two formulae allows one to extend of one symbol to the right the word under analysis and, subsequently, to cut one symbol on the left side, basically simulating a sliding window. This allows fast computation of statistics for words with mismatches.

Examples of such approach are the computation of the correction factor under i.i.d. for fixed lengths, and the counting of words with mismatches for fixed lengths.

The first formula can be used to preprocess the entire input text, and build a table that holds statistics with $i = 0 \dots k$ mismatches of all its prefixes. From this information, given two indexes (b, e) that univocally identify a substring of x, one can use the information stored in the table, in particular for the prefixes of length $b - 1$ and those of length e, the required statistics for the substring $x[b, e]$ with an iterative subtraction of the contributions due to an assignment of $k - i$ mismatches before b and i mismatches within $x[b, e]$.

Since there are k such partition the complexity of this approach necessarily proportional to k. Indeed, since it is necessary to compute the statistics also for all the $i < k$ the approach is quadratic if at each step the number of operations to perform is constant.

Such method turn out to be useful especially for computing indexes for a subset of words that occur in the input string. In fact, the time required by the pre-processing is usually negligible with respect to the time required for a direct computation of all the statistics of interests with an approach dependent on the length of the substring.

5.1 Application to compact pattern discovery

The degree of surprise associated with the recurrence of a word or motif in a sequence or family of sequences is measured by some *z-score* that takes into account the observed and expected frequencies, perhaps normalized by some parameter such as expectation or higher moments. Basic scores in use are, e.g.,

$$z_1(w) = F(w) - E(w), \quad z_2(w) = F(w)/E(w)$$

$$z_3(w) = \frac{F(w) - E(w)}{\sqrt{Var(w)}}; \quad z_4(w) = \frac{(F(w) - E(w))^2}{E(w)}$$

where F denotes frequency, E expected frequency and Var variance.

An interesting aspect the expected number of occurrences is that both for solid words [2] and for words with mismatches (with some reasonable assumptions) [?] its value decreases when the sequence length increases. As a consequence, for a string that is an extension of another, but that shows the same number of occurrences, the z-scores are monotonically increasing.

Monotonicity of the score allows in turn to develop compact approaches for pattern discovery. In fact, for runs of substrings $x[i, i + l]$, with increasing l, but the same number of occurrences, the output can be limited to the longer of the strings for which the frequency is constant. The idea is that for any word v without a score, there is a scored extension vy which is at least equally surprising. In other words, the space of the substrings is partitioned into classes, and only one representative element for each class is given a score (the longest string). The output size is thus reduced from $O(|\text{search space}|)$ to $O(\text{number of classes})$

5.2 Solid words

For the case of solid words, the set of $O(n^2)$ substrings of x is partitioned in classes in which each word has the same location set (and thus number of occurrences) by linear time and space data structures such as the suffix tree [2]. In a suffix tree, all the strings that are described by a concatenation of symbols that starts from the root and ends in the middle of an arc, have the same number occurrences of the string that ends at the end of the same arc. In this case each node represent a class and the representative is the string associated with the node. This guarantee that for any word v that has not be given a score, there exist an extension of it, vy that has been score and that it is at least equally surprising (has a z-score greater or equal).

5.3 Words with mismatches

For words with mismatches a data structure such the suffix tree cannot be used because the frequency with mismatches oscillate along the arcs. However, one can devise a framework in which, for a given fixed length m and interval span δ, for each position i in x all substrings of length $m \pm \delta$ are potential motifs. As mentioned, the table at the outset risks to be too bulky. However, the use of monotonicities shall enable us to neglect part of table without loss of information. In fact, for strings that in this range show constant frequency it suffices to compute a single score. This allow for substantial reduction of the size of table that holds statistics for words with mismatches.

6 Conclusions

We have review some algorithms for the efficient computation of indexes for words with mismatches. All these algorithms are based on a common framework that rely on a preprocessing step, based in an incremental approach, in which all the prefixes are indexed to allow subsequent efficient query time for any word that occurs in the text. The same incremental approach can be adapted to scan the sequence with a sliding window to compute statistics for words of fixed length even more efficiently.

We conclude the discussion with some open questions: i) is it possible to devise a data structure that partition the search space in classes such the suffix tree does for solid words without incurring in an exponential explosion of the time and space required by the approach? ii) is it possible to devise much broader domains of monotonicity through the interplay of expectation and frequency than intervals of constant frequency? iii) the algorithm for counting occurrences for mismatches takes $O(n^2)$ time and $O(n)$ space (which is also the size of the output, if only the number of occurrences is required). It is possible to reduce the time of computation?

References

1. Abrahamson, K. Generalized string matching *SIAM J. Computing*, 16(6):1039–1051, 1987.
2. Apostolico, A., Bock, M. E., and Lonardi, S. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10:3-4, 283–311, 2003.
3. Apostolico, A., and Denas O. Fast algorithms for computing sequence distances by exhaustive substring composition. *Algorithms for Molecular Biology* 3:13, 2008.
4. Apostolico, A., and Pizzi, C. Motif Discovery by Monotone Scores. *Discrete Applied Mathematics*, 155(6-7): 695-706, 2007.
5. Apostolico, A., and Pizzi, C. Scoring Unusual Words with Varying Mismatch Errors. *Mathematics in Computer Science*, Special Issue in Combinatorial Algorithms, 1(4):639-653, 2008.
6. V.Boeva, J.Clément, M.Régnier, M.Vandenbogaert Assessing the Significance of Sets of Words. *In proceedings of Combinatorial Pattern Matching 2005*, LNCS 3537: 358–370.
7. Brāzma, A., Jonassen, I., Ukkonen, E., and Vilo, J. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research*, 8(11), 1202–1215, 1998.
8. Chor, B., Horn, D., Goldman, N., Levy, Y., and Massingham, T Genomic DNA k-mer Spectra: Models and Modalities *Genome Biology*, 10:R108, 2009.
9. Hertz, G. Z., and Stormo, G. D. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics 15*, 563–577, 1999.
10. Qi, J., Wang, B., and Hao,B. Whole genome prokaryote phylogeny without sequence alignment: a K-string composition approach *J. Mol. Evol.*, 58(1), 1-11, 2004.
11. Keich, and Pevzner. Finding motifs in the twilight zone. In *Annual International Conference on Computational Molecular Biology* (Washington, DC, Apr. 2002), pp. 195–204.
12. Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., and Wootton, J. C. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science 262* (Oct. 1993), 208–214.
13. Leslie C.S., Eskin, E., Cohen, A., Weston, J., ans Stafford Noble W. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4),2004.
14. Navarro, G. A Guided Tour to Approximate String Matching. *ACM Computing Surveys* (CSUR), 33(1):31–88, 2001.

15. Pizzi, C. k-difference matching in amortized linear time for all the words in a text. *Theoretical Computer Science*, 410(8-10): 983–987, 2008.
16. Pizzi, C., and Bianco, M. Expectation of Strings with Mismatches Under Markov Chain Distribution. In *Proc. of 16th Symposium on String Processing and Information Retrieval (SPIRE 2009)*, Saariselka, Finland, LNCS 5721 pp 222–233, 2009.
17. Shina, S., and Tompa, M. Discovery of novel transcription factor bind- ing sites by statistical overrepresentation. *Nucleic Acids Research*, 30(24):5549–5560, 2002.
18. van Helden, J., Andre, B., Collado-Vides, J. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281:827-842, 1998.