

Monotone Scoring of Patterns with Mismatches (extended abstract)

Alberto Apostolico^{1*} and Cinzia Pizzi^{2**}

¹ University of Padova & Purdue University

² University of Padova

Abstract. We study the problem of extracting, from given source x and error threshold k , substrings of x that occur unusually often in x within k substitutions or mismatches. Specifically, we assume that the input textstring x of n characters is produced by an i.i.d. source, and design efficient methods for computing the probability and expected number of occurrences for substrings of x with (either *exactly* or *up to*) k mismatches. Two related schemes are presented. In the first one, an $O(nk)$ time preprocessing of x is developed that supports the following subsequent queries: for any substring w of x arbitrarily specified as input, the probability of occurrence of w in x within (either exactly or up to) k mismatches is reported in $O(k^2)$ time. In the second scheme, a length or length range is arbitrarily specified, and the above probabilities are computed for all substrings of x having length in that range, in overall $O(nk)$ time. Further, monotonicity conditions are introduced and studied for probabilities and expected occurrences of a substring under unit increases in its length, allowed number of errors, or both. Over intervals of constant frequency count, these monotonicities translate to some of the scores in use, thereby reducing the size of tables at the outset and enhancing the process of discovery. These latter derivations extend to patterns with mismatches an analysis previously devoted to exact patterns.

1 Preliminaries

The problem of extracting unusually frequent or rare patterns from observed sequences arises ubiquitously in applications and has been the subject of much study in Molecular Biology. This problem may take up different flavors, depending on the assumptions about the source and on the character and structure

* Dipartimento di Ingegneria dell' Informazione, Università di Padova, Padova, Italy and Department of Computer Sciences, Purdue University, Computer Sciences Building, West Lafayette, IN 47907, USA. Work Supported in part by an IBM Faculty Partnership Award, by the Italian Ministry of University and Research under the National Projects FIRB RBNE01KNFP, and PRIN "Combinatorial and Algorithmic Methods for Pattern Discovery in Biosequences", and by the Research Program of the University of Padova. axa@dei.unipd.it

** Dipartimento di Ingegneria dell' Informazione, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy. cinzia.pizzi@dei.unipd.it

of the patterns themselves. It is customary to partition the approaches to discovery into two main classes. 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* or templates. This is methodologically sound but may pose daunting 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. This may be less firm methodologically but brings about time and space savings. Some hybrid variants consist of postulating or building the models by inference from their incarnations in the sample itself. Except for the case of solid patterns, however, all of the available techniques present some intrinsic exponential buildup that often translates into unbearable computational overhead. We refer to the quoted sample of literature for details. In this paper, we study the approach that consists of extracting from given source x and error threshold k , substrings of x that occur unusually often in x within k mismatches. To quantify “unusually often” for a substring w of x , this is measured by comparing, e.g., the observed frequency and the expected number of occurrences for w with (either exactly, or up to) k mismatches. We study problems related to the efficient computation and representation for these measures.

Throughout the rest of the discussion, a *motif* is a pair (w, k) where w is a string of characters from an alphabet Σ and k is the number of errors or mismatches allowed on w . Thus, the pair (w, k) identifies a family of strings over Σ , whereas the same string belongs to more than one family. We will use $w_{(k)}$ to refer to a string in (w, k) . To avoid clutter in notation, we will let the context specify whether k denotes the maximum or exact number of errors. When $k = 0$ we talk of *solid* strings or patterns.

Given a textstring x and a length range $m \pm \delta$ with constant δ , we are interested in particular in the efficient construction of a table $\mathcal{W}(x)$ containing all motifs (w, k) of length between $m - \delta$ and $m + \delta$ and such that w is a substring of x , together with their individual probabilities, and expected number of occurrences. In practical applications such as, e.g., regulatory sequence detection, values of $m \approx 10 - 15$ and $\delta \approx 3 - 6$ are typical. As mentioned, the above expectations combine with frequency counts to yield some of the z -scores in use. Since $\mathcal{W}(x)$ can be quite bulky, we will seek to reduce its size, by limiting it to entries representing local maxima w.r.t. the score.

We use capital letters to denote random strings and variables. In particular, $X = X_1 X_2 X_3 \dots X_n$ denotes a random textstring produced by a source which emits symbols from Σ under i.i.d. assumptions, i.e., the X_i are emitted independently and according to the same distribution. This is denoted by setting $P[X_i = s \in \Sigma] = p_s = p_i \forall i$, with obvious meaning.

For an *observed* pattern $y = y_1 y_2 \dots y_m$, the probability of y is $P(y) = p_1 p_2 \dots p_{|y|}$, which is also the expected value of the indicator variable $Z_i|y$, taking value 1 when y occurs beginning at position i in X and 0 otherwise. Thus, $E[Z_i|y] = E[Z_1|y] = P(y)$. The random variable representing the number of occurrences of y in X is $Z|y = \sum_{i=1}^{n-m+1} Z_i|y$. In the following, and whenever this

causes no confusion, we will use $E[y]$ shorthand for $E[Z|y] = \sum_{i=1}^{n-m+1} E[Z_i|y] = (n-m+1)P(y)$. Likewise, we will use $P_k(y)$ for $P(y_{(k)})$, $E_k[y]$ for $E[y_{(k)}]$.

The rest of this paper is organized as follows. In the next section, we give efficient computations for motif probabilities and expected number of occurrence. The monotonicities of these and related parameters are established in Section 3. Section 4 completes our constructions and concludes the paper.

2 String Probabilities and their Correction Factors

With linear-time preprocessing of a standard prefix computation on the textstring x , it is trivial to compute the probability or expected number of occurrences of any substring of x in constant time. The pre-processing consists of building the array $A[i] = \prod_{h=1}^i p_h$, $i = 1, 2, \dots, n$ with $A[0] = 1$, so that, for instance, for any pair (b, e) of positions, the probability of $\bar{x} = x[b \dots e]$ is $P(\bar{x}) = \prod_{i=b}^e p_i = \frac{A[e]}{A[b-1]}$ and $E[\bar{x}] = A[e]/A[b-1](|x| - e + b)$.

The same computation for patterns having exactly k mismatches with \bar{x} risks to incur exponential cost, since we need to tally all of the $\binom{|\bar{x}|}{k}$ ways to position k mismatches in \bar{x} . A less expensive, incremental approach can be built on the notion of *correction factor*. As an example, consider the pattern $y = abaabaa$ on $\Sigma = \{a, b\}$ and let $P(y) = p_a p_b p_a p_a p_b p_a p_a = p_a^5 p_b^2$. A mutation in the first position would change y into $y' = bbaabaa$, with associated probability: $P(y') = p_b p_b p_a p_a p_b p_a p_a = \frac{p_b}{p_a} p_b p_b p_a p_a p_b p_a p_a = \frac{p_b}{p_a} P(y)$.

We define $f_a = \frac{p_b}{p_a}$ as the correction factor for the character a . More in general, if a character s is allowed to mutate into any one of the characters in the subset $\Sigma_s \subseteq \Sigma$, we define the Σ_s -*correction factor* for s as $\sum_{s' \in \Sigma_s} p_{s'}/p_s$. Hereafter, we assume for simplicity that $\Sigma_s = \Sigma \setminus \{s\}$ for all characters of Σ and set the correction factor $f_s = \sum_{s' \in \Sigma \setminus \{s\}} p_{s'}/p_s$.

Clearly, if $\hat{p} = P(y)$ is the probability of y , the probability of any string y' differing from y due to the change of a character s is obtained by multiplying $P(y)$ by the correction factor for s . In the example above, the change in probability is the same when the error occurs at any of the positions 1,3,4,6,7. For the remaining positions we would have to multiply $P(y)$ by f_b . In conclusion, if y is a substring of a text x , the probability of occurrence in x of a string y' differing from y in exactly one position is: $P_1(y) = n_a f_a \hat{p} + n_b f_b \hat{p} = \hat{p}(n_a f_a + n_b f_b)$, and $E_1[y] = E[y](n_a f_a + n_b f_b)$.

If we were to compute probabilities and expectations for occurrences of y with exactly k errors, then we would have to compute the above for all possible choices of k positions among m , which entails a complexity of $O(m^k)$. For small Σ , such as in DNA, some improvement is obtainable by distributing errors among the at most $|\Sigma|$ characters rather than the m positions. In our example string, for instance, we have $n_a = 5$ and $n_b = 2$ and the probability of occurrences with 2 mismatches is:

$$\binom{5}{2} f_a^2 + \binom{2}{2} f_b^2 + \sum_{i=1}^1 \binom{5}{i} \binom{2}{2-i} f_a^i f_b^{2-i} = 10 f_a^2 + f_b^2 + 10 f_a f_b.$$

The dominant term in the calculation for k errors is the one involving all the characters of the alphabet, and it requires $|\Sigma| - 1$ nested cycles totaling $O(k^{|\Sigma|-1})$ time.

Besides being expensive, neither one of these approaches lends itself to iterated computations where, e.g., the probabilities of occurrences with (up to) k mismatches of all m -character substrings of a text are sought. The approach presented next achieves this in $O(k^2)$ per iteration, hence in $O(k^2n)$ time for a text of n characters. It requires an $O(kn)$ time pre-processing of the text, which is described next.

Text pre-processing. Given a text x of length n , and a fixed number of errors k , we build a $[k \times n]$ matrix A whose generic entry $A[i][j]$ is the correction factor to be applied to the probability of string $x[1 \dots j]$ with exactly i errors. Array A is readily computed in time $O(kn)$.

Lemma 1. *With $f_{x[j]}$ the correction factor of $x[j]$, the following holds:*

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

Proof. We neglect the obvious boundary conditions and concentrate on the last row. The recurrence (see also Figure 1) states that the correction factor to be applied to $x[1 \dots j]$ when i errors are allowed comes from two tributaries:

1. The symbol at position j is correct and exactly i errors occur in $x[1 \dots j-1]$
2. There is an error at position j and exactly $i-1$ errors in $x[1 \dots j-1]$. \square

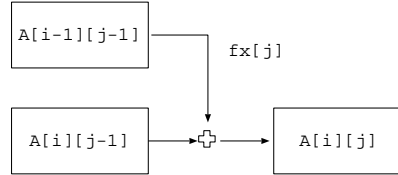


Fig. 1. Computing $A[i][j]$

Processing. Let $C_k(b, e)$ be the global correction factor to be applied to substring $\bar{x} = x[b \dots e]$ in order to obtain the probability of \bar{x} when exactly k errors are imposed. Thus, $\hat{p}C_k(b, e) = P(\bar{x}_k)$ and this value depends on string x and on the indices (b, e) .

Lemma 2. $C_0(b, e) = 1$. For $k > 0$, $C_k(b, e) = A[k][e] - \sum_{i=0}^{k-1} A[k-i][b-1] \cdot C_i(b, e)$.

i	a	b	c	c	b
0	1	1	1	1	1
1	f_a	$f_a + f_b$	$(f_a + f_b) + f_c$	$(f_a + f_b + f_c) + f_c$	$(f_a + f_b + f_c + f_c) + f_b$
2	0	$f_a f_b$	$(f_a f_b) + (f_a + f_b) f_c$	$[f_a f_b + (f_a + f_b) f_c] + (f_a + f_b + f_c) f_c$	$[f_a f_b + (f_a + f_b) f_c + (f_a + f_b + f_c) f_c] + (f_a + f_b + f_c) f_b$
3	0	0	$(f_a f_b) f_c$	$(f_a f_b f_c) + [f_a f_b + (f_a + f_b) f_c] f_c$	$f_a f_b f_c + [f_a f_b + (f_a + f_b) f_c] f_c + [f_a f_b + (f_a + f_b) f_c + (f_a + f_b + f_c) f_c] f_b$
4	0	0	0	$(f_a f_b f_c) f_c$	$f_a f_b f_c f_c + [f_a f_b f_c + (f_a f_b) + (f_a + f_b) f_c] f_c f_b$

Table 1. Computing the array A for $x = abc cb$ and $k = 4$

Proof. That $C_0(b, e) = 1$ is obvious. Consider first $C_1(b, e)$. In order to compute it, it suffices to take the correction factor for text $x[1 \dots e]$ and subtract the errors contributed by positions preceding b . Thus, $C_1(b, e) = A[1][e] - A[1][b - 1]C_0(b, e)$. Consider now the case $k = 2$. By construction, $A[2][e]$ contains the correction factor for $x[1 \dots e]$ with 2 errors. From this value we need to subtract the following:

- the contribution $A[2][b - 1]$ of 2 errors occurring in positions which precede b ;
- the contribution due to 1 error occurring in $x[1 \dots b - 1]$ and 1 error occurring in $x[b \dots e]$

Thus, the final correction factor is: $C_2(b, e) = A[2][e] - A[2][b - 1] \cdot C_0(b, e) - A[1][b - 1] \cdot C_1(b, e)$. Continuing in this fashion we obtain, for general k : $C_k(b, e) = A[k][e] - \sum_{i=0}^{k-1} A[k - i][b - 1] \cdot C_i(b, e)$ which is the formula of the claim. \square

Lemma 3. *Once the table A has been built, the computation of the correction factor for any substring $y = x[b \dots e]$ with $1, 2, \dots, k$ errors takes $O(k^2)$ steps.*

Proof. We exhibit an algorithm based on the previous lemma that fulfills the claim.

```

1. array C[0...k]
2. C[0]=1;
3. for i=1 to k
4.   Sum = 0
5.   for j=0 to i-1
6.     Sum = Sum + A[i-j][b-1] C[j]
7.   C[i] = A[i][e] - Sum

```

The values $C[j]$ at row 6 have been already computed in the previous cycles, so that line requires constant time. The total complexity is then charged by the two nested cycles, leading to an $O(k^2)$ algorithm. \square

Unlike that of the methods described earlier, this time complexity is independent from both the alphabet size and the pattern length.

Note that in order to compute $C_k(b, e)$ we also compute $C_i(b, e)$, $0 \leq i \leq k$, so that in time $O(k^2)$ we actually determine the value for k consecutive correction factors. On average, the time needed to compute a single factor is thus $O(k)$.

In conclusion, after $O(kn)$ time and space pre-processing of the text, it is possible to obtain in $O(k^2)$ time, from input 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. 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, we can achieve a substantial improvement in both time and space for this case. The on-line algorithm presented next proceeds by dynamically adjusting the desired values in a sliding window of any given fixed length m . To avoid clutter in notation we assume $m \geq k$, but this constraint can be removed without penalty.

First, note that a recurrence similar to the one used to compute the correction factor $A[i][j]$ from $A[i-1][j]$ and $A[i-1][j-1]$ can be used to derive the correction factor of a string $w' = x[b \dots e+1]$ from the correction factor of the string $w = x[b \dots e]$. Indeed, we have $C_0(b, e+1) = 1$ and, for $k > 0$: $C_k(b, e+1) = C_k(b, e) + C_{k-1}(b, e)f_{x[e+1]}$.

Lemma 4. $C_0(b+1, e) = 1$; for $k > 0$, $C_k(b+1, e) = C_k(b, e) - C_{k-1}(b+1, e)f_{x[b]}$.

Proof. Observe that $C_k(b, e) = C_k(e, b)$, i.e., computing the C_k 's for a string from left to right or from right to left will not change their values. Therefore, the correction factors of the string w are related to those of the string $w'' = x[b+1 \dots e]$ by the formula:

$$C_0(b, e) = 1; \quad C_k(b, e) = C_k(b+1, e) + C_{k-1}(b+1, e)f_{x[b]} \quad \text{for } k > 0.$$

which is equivalent to the one in the claim. \square

Triggered by the ‘‘universal’’ initial condition $C_0 = 1$, Lemma 4 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 $i = 1, 2, \dots, k$. At that point, we can apply the above Equation, rewritten as:

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

to similarly compute the $C_i(b+1, e+1)$'s.

Clearly, 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. By extending this treatment to all substrings of length $m \pm \delta$ would enable us to weigh $\mathcal{W}(x)$ within these bounds. As mentioned, the table at the outset risks to be too bulky, hence we address next monotonicities that shall enable us to neglect part of $\mathcal{W}(x)$ without loss of information.

3 Monotonicities

We begin by making an assumption that limits the skewedness on our probability distributions in exchange for some useful consequences. The assumption is quite reasonable for genomic as well as general applications.

Assumption 1: $p_a \leq \sum_{s \in \Sigma \setminus \{a\}} p_s$

As an immediate consequence of this assumption, we get:

Property 1: $f_a \geq 1 \quad \forall a \in \Sigma$; **Property 2:** $A[i][j] \geq 0 \quad \forall i, j$.

Property 2 follows from the observation that only positive values are added in our algorithm, once $A[i][j] = 0$ for $i < j$, and $A[1][1]$ is positive.

As seen, a recurrence similar to the one used to compute $A[i][j]$ can be used to compute correction factors of the string $x[b \dots e]$ from that of $x[b \dots e - 1]$. Specifically, we have:

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

This shows that correction factors are always non-negative, and in fact that they are in positive for words longer than the number of errors k .

Property 3: $C_k(w) \geq 0 \quad \forall k, w$ and $C_k(w) > 1 \quad \forall w : |w| \geq k > 0$.

$C_k(b, e)$ is the correction factor to be applied to string $x[b \dots e]$ in order to allow for *exactly* k errors. It is natural to extend the notion of correction factor to the case where one wants to consider *at most* k errors in a string. The corresponding expression for $x[b \dots e]$ is obtained by taking the sum of all the correction factors from 0 to k :

$$\bar{C}_k(b, e) = \sum_{i=0}^k C_i(b, e) \quad (2)$$

We discuss next some properties of monotonicity of correction factors both for the case of *exactly* k errors (ECF) as well as for the case (UCF) of *up to* k errors. We are specifically interested in the behavior of these factors for a word under each one of the following scenarios:

1. the word length is increased, keeping error number fixed;
2. the number of errors is increased, keeping word size fixed;
3. both word length and number of errors are increased.

We set $w = v \cdot a$ where $w, v \in \Sigma^*$ and $a \in \Sigma$ and study first C_k . We shall make frequent use in our proofs of Equation 1 that is reported below in the crisper version:

$$C_k(w) = C_k(v) + C_{k-1}(v) f_a. \quad (3)$$

Lemma 5. For $w = va$, $C_k(w) \geq C_k(v)$.

Proof. From Equation 3, considering Property 1 we have $C_k(w) = C_k(v) + C_{k-1}(v) f_a \geq C_k(v) + C_{k-1}(v)$. By Property 2, $C_{k-1} \geq 0$. We can conclude that $C_k(w) \geq C_k(v) + C_{k-1}(v) \geq C_k(v)$. \square

Lemma 6. For $w = va$, $C_k(w) \geq C_{k-1}(v)$.

Proof. By the argument in the previous Lemma, since $C_k(v) \geq 0$ we also have: $C_k(w) \geq C_k(v) + C_{k-1}(v) \geq C_{k-1}(v)$. \square

A counterexample will show that, in general, the correction factor is not monotonically increasing when the number of errors allowed is increased while the length of the string is kept fixed. To see this, assume that the characters of Σ have the same probability. Hence:

$$p_{s_1} = p_{s_2} = \dots = p_{s_{|\Sigma|}} = p = \frac{1}{|\Sigma|} \text{ and } f_{s_1} = f_{s_2} = \dots = f_{s_{|\Sigma|}} = f = |\Sigma| - 1$$

In this special case, for a word w we have:

$$C_k(w) = \binom{|w|}{k} f^k.$$

We claim that there is a value \bar{k} for k such that the correction factor is monotonically increasing for $k \leq \bar{k}$ and monotonically decreasing for $k > \bar{k}$. To determine \bar{k} , observe that by the definition of $C_k(w)$ we have:

$$C_k(w) < C_{k+1}(w) \implies \binom{|w|}{k} f^k < \binom{|w|}{k+1} f^{k+1} \implies f > \frac{k+1}{|w|-k}$$

Hence $C_k(w) < C_{k+1}(w)$ holds for $k < (|w|f - 1)/(f + 1)$. Combined with its symmetric argument, this leads to conclude that, with $\bar{k} = \lfloor \frac{|w|f-1}{f+1} \rfloor$, we have:

$$\begin{cases} C_k(w) < C_{k+1}(w) & \text{for } k \leq \bar{k} \\ C_k(w) > C_{k+1}(w) & \text{for } k > \bar{k} \end{cases}$$

Thus $C_k(w)$ is bi-tonic in this case. Our counterexample suggests that monotonicity might be preserved within restricted ranges of errors. If we apply the above to almost-evenly distributed DNA strands, for instance, we have $\Sigma = \{a, c, g, t\}$, $f = 3$ and $\bar{k} = \lfloor \frac{3|w|-1}{4} \rfloor$. Hence for words of length 16 we can assume the correction factor to increase monotonically up to 11 errors, which is a very large fraction of the string length for most practical purposes. In addition, we also get that for $12 \leq k \leq 16$ the correction factor is monotonically decreasing.

A more general formula expressing \bar{k} must depend on the distribution and might be hard to come by. However, our next lemma establishes an acceptable lower bound for \bar{k} , that corresponds to half the length of the string w .

Lemma 7. $C_k(w) \geq C_{k-1}(w) \quad \forall k \leq \frac{|w|}{2}$.

Proof. Let $w = w_1 w_2 \dots w_m$. The inequality holds for $k = 1$, since $C_0(w) = 1$ and:

$$C_1(w) = \sum_{i=1}^m f_i \geq \sum_{i=1}^m 1 = m \geq 1 = C_0(w).$$

The contribution of each position in this case is the correction factor of the character occupying that position. Hence we obtain a set of $\binom{m}{1} = m$ terms, which may be expressed as $\mathcal{C}_1 = (f_1, f_2, \dots, f_m)$. For $k = 2$, we obtain a set of $\binom{m}{2} = m(m-1)/2$ terms, where each term results from the combination of the characters at two positions of w , say, w_i and w_j , and consists of the product of the corresponding correction factors $f_i f_j$. Specifically, the set of contributions for $k = 2$ is given by $\mathcal{C}_2 = (f_1 f_2, f_1 f_3, \dots, f_1 f_m, f_2 f_3, \dots, f_2 f_m, \dots, f_{m-1} f_m)$. Since $\forall i \ f_i \geq 1$, then $f_i f_j = f_j f_i \geq f_i \ \forall i, j$, so that for every term f in \mathcal{C}_1 we have at least one element \bar{f} of \mathcal{C}_2 such that $\bar{f} \geq f$. This argument propagates from one \mathcal{C} to the next for as long as the number of terms increases. But the number of terms is given by the binomial coefficients, hence our condition is preserved only for values of k up to $\frac{k}{2}$. We conclude that $|w|/2$ is always safe as a lower bound for \bar{k} . \square

We consider next the modified correction factor $\bar{C}_k(w)$ defined in Equation 2. Interestingly, monotonicity holds here with no restrictions.

Lemma 8. *Let $w = va$. Then, $\bar{C}_k(w) \geq \bar{C}_k(v)$; $\bar{C}_k(w) \geq \bar{C}_{k-1}(w)$; $\bar{C}_k(w) \geq \bar{C}_{k-1}(v)$.*

Proof. From Equation 2 and that of Lemma 5, it immediately follows that:

$$\bar{C}_k(w) = \sum_{i=0}^k C_i(w) \geq \sum_{i=0}^k C_i(v) = \bar{C}_k(v),$$

leading to the first inequality. We also have from Equation 2:

$$\bar{C}_k(w) = \sum_{i=0}^k C_k(w) = C_k(w) + \sum_{i=0}^{k-1} C_i(w) = C_k(w) + \bar{C}_{k-1}(w) \geq \bar{C}_{k-1}(w)$$

which establishes the second inequality. The third inequality follows from the previous two, whence $\bar{C}_k(w) \geq \bar{C}_k(v) \geq \bar{C}_{k-1}(v)$. \square

We now turn to probabilities of strings with errors. By definition, the probability of occurrence for string w when k errors are allowed is given by the product of that string probability and its correction factor for k errors:

$$P_k(w) = P(w)D_k(w)$$

where $D_k(w)$ is either $C_k(w)$ or $\bar{C}_k(w)$ depending on whether we are considering ECFs or UCFs. Our next lemma can be stated in terms of $D_k(w)$ but when $D = C$ the additional assumption that $|w|/2 > k$ is needed.

Lemma 9. *Let $w = va$ ($a \in \Sigma$), then $P_k(w) \leq P_k(v)$, and $P_k(w) \geq P_{k-1}(w)$*

Proof. By definition, $P(w) = P(v) \cdot p_a$ where $w = va$. From Equation 3 and the definition of f_a , it follows:

$$\begin{aligned} P_k(w) &= p_a P(v) (D_k(v) + D_{k-1}(v) f_a) \\ &= p_a P(v) D_k(v) + D_{k-1}(v) P(v) p_a \times \frac{\sum_{s \neq a} p_s}{p_a} \end{aligned}$$

Since $D_k(v) \geq D_{k-1}(v)$, we obtain the inequality:

$$P_k(w) \leq p_a P(v) D_k(v) + P(v) D_k(v) \times \sum_{s \neq a} p_s$$

By the definition of $P_k(v)$ we finally have:

$$P_k(w) \leq p_a P_k(v) + P_k(v) \times \sum_{s \neq a} p_s = P_k(v) \times \sum_{s \in \Sigma} p_s = P_k(v)$$

From Lemma 5 we have: $P_k(w) = P(w) D_k(w) \geq P(w) D_{k-1}(w) = P_{k-1}(w)$. \square

Our analysis of monotonicities is summarized in the following

Theorem 1. *Under both ECF and UCF, for any v, z and $w = vz$ and any integer $k < |w|/2$, it is $P_k(w) \leq P_k(v)$ and $P_k(w) \geq P_{k-1}(w)$.*

Proof. By the above discussion and lemmas. \square

For words w in a text x such that $|w| = m \pm \delta \ll n = |x|$ the above inequalities translate to expectations, i.e., under the conditions of Theorem 1 we get also that $E_k(w) \leq E_k(v)$ and $E_k(w) \geq E_{k-1}(w)$.

4 Monotone Scores

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., $z_1(w) = F(w) - E(w)$, $z_2(w) = F(w)/E(w)$, and

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

where F denotes frequency, E expected frequency and Var variance. The expression and computation of the expected values, moments and related scores of significance depend substantially on the particular notion used. For sequence families, the frequency of a pattern can be defined in at least two ways, depending on whether we count the total number of pattern occurrences or the number of sequences containing each at least one occurrence of that pattern. In this paper, attention is restricted to notions involving the *total number* of occurrences, whether in a single sequence or sequence family (the latter being reduced to

a singleton thru concatenation of its members), and using no higher moments. Whereas the first one of our restrictions is not hard to forfeit, the efficient computation of scores involving variance and higher moments have proved to pose serious algorithmic challenges even for solid patterns [3].

In appropriate synergy with frequencies, the monotonicity of expectations extends to the related scores. In particular, such scores are monotone over intervals of constant frequency. Here we limit consideration to this simplest case, however, much broader domains of monotonicity can be identified through the interplay of expectation and frequency, and these are under study.

For our application, we add k as a subscript to indicate the number of errors. Thus, e.g., $F_k(w)$ is the number of observed subwords of x at a distance k from w . As there is no substantial difference in our computation whether k is the exact or maximum number of errors we make here no distinction of treatment nor belabor this point further. We only need to discuss the computation of $F_k(w)$ for all words of x of size $|w| = m \pm \delta$. This is done by established techniques in $O(nk)$ or even expected sublinear time (see, e.g., [3]). There are $O(n)$ subwords of length m in x , whence the total computation with δ a constant is $O(n^2k)$ or expected $O(n^2)$. This information can be organized in the $n \times (2\delta + 1)$ table $\mathcal{W}(x)$ at the outset, such that the frequencies of substrings of length $[m - \delta \dots m + \delta]$ beginning at position i form the i -th column of the table. Looking now at every single column will suffice to firm the intervals of monotonicity for F , whence only one extreme in each class is retained and weighed with expectation and score. The overall cost including expectations is $O(n^2k)$ or expected $O(n^2)$, depending on the method used.

As an illustration, Table 2 displays the results of computations performed on a sequence of $n = 5,000$ bases randomly generated according to a genomic distribution (specifically: $p_a = p_t = 0.30$; $p_c = p_g = 0.20$, the approximate base composition of yeast nuclear chromosomes), for various values of m and $\delta = 3$. For each triplet (m, k, δ) , the substrings of length between $m - \delta$ and $m + \delta$ were considered in succession, and the number of occurrences of each string with exactly (left half of the table) or up-to (right part) k errors were computed. The elimination of entries with zero frequency led to the table sizes reported under “# entries”. From these, runs of identical counts corresponding to consecutive extensions of a same substring were identified and compacted each to a single entry. With reference to Table 2, for instance, the leftmost column at $k = 1$ states that of all substrings of length from 9 to 15 only 3,326 had a non-zero F_1 value. Of these, 16% consisted of runs with an average length of 2.3 characters. Using one representative entry per run yields a 21.20% reduction in the size of the table. For comparison, the number of strings having length in this range is 1,073,741,824.

Acknowledgement We are indebted to one of the Referees for the careful scrutiny of an earlier version of this manuscript.

		m = 12	m = 13	m = 14	m = 15	m = 16	m = 17			m = 12	m = 13	m = 14	m = 15	m = 16	m = 17
k	# entries	34853	34846	34839	34832	34825	34818	k	34853	34846	34839	34832	34825	34818	
=	# runs	5015	4994	4987	4982	4977	4974	=	5015	4994	4987	4982	4977	4974	
0	avg length	6.93	6.97	6.98	6.99	6.99	6.99	0	6.93	6.97	6.98	6.99	6.99	6.99	
	% saving	85.35	85.61	85.67	85.69	85.70	85.71		85.35	85.61	85.67	85.69	85.70	85.71	
k	# entries	3326	1108	332	106	42	18	k	34853	34846	34839	34832	34825	34818	
=	# runs	543	200	66	24	10	4	=	5538	5192	5051	5006	4989	4980	
1	avg length	2.30	2.33	2.36	2.33	2.20	2.00	1	5.92	6.59	6.86	6.95	6.98	6.99	
	% saving	21.20	23.92	27.11	30.19	28.57	22.22		78.13	83.29	85.02	85.50	85.63	85.67	
k	# entries	13119	8186	3847	1398	474	156	k	34853	34846	34839	34832	34825	34818	
=	# runs	1106	6880	605	261	95	32	=	5909	5940	5625	5279	5090	5016	
2	avg length	2.27	2.30	2.32	2.30	2.27	2.22	2	4.04	4.84	5.76	6.45	6.80	6.93	
	% saving	10.72	15.95	20.72	24.25	25.52	25.00		51.51	65.50	76.81	82.66	84.70	85.38	
k	# entries	22949	18075	13156	8230	3960	1522	k	34853	34846	34839	34832	34825	34818	
=	# runs	1171	1204	1211	1072	635	263	=	5073	5802	6064	6050	5703	5310	
3	avg length	2.20	2.24	2.25	2.26	2.29	2.30	3	2.65	3.26	3.97	4.77	5.67	6.40	
	% saving	6.14	8.23	11.55	16.46	20.70	22.54		24.08	37.60	51.69	65.49	76.55	82.33	

Table 2. Sample table size reductions for exactly (left half) and up-to (right half) k errors.

References

1. APOSTOLICO, A. Pattern discovery and the algorithmics of surprise. In *Artificial Intelligence and Heuristic Methods for Bioinformatics* (2003), P. Frasconi and R. Shamir, Eds., IOS Press, pp. 111–127.
2. APOSTOLICO, A., AND GALIL, Z., Eds. *Pattern matching algorithms*. Oxford University Press, 1997.
3. APOSTOLICO, A., BOCK, M. E., AND LONARDI, S. Monotony of surprise and large-scale quest for unusual words (extended abstract). In *Proc. of Research in Computational Molecular Biology RECOMB* (Washington, DC, April 2002), G. Myers, S. Hannenhalli, S. Istrail, P. Pevzner, and M. Waterman, Eds. Also, *J. Comp. Bio.*, 10:3-4, (July 2003), 283–311.
4. APOSTOLICO, A., AND PARIDA, L. Incremental Paradigms of Motif Discovery. *J. Comput. Bio.* 7, 11:1, (Jan. 2004), 15–25.
5. BAILEY, T. L., AND ELKAN, C. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning* 21, 1/2 (1995), 51–80.
6. BRÁZMA, A., JONASSEN, I., UKKONEN, E., AND VILO, J. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research* 8, 11 (1998), 1202–1215.
7. BUHLER, J., AND TOMPA, M. Finding motifs using random projections. *J. Comput. Bio.* 9, 2 (2002), 225–242.
8. HERTZ, G. Z., AND STORMO, G. D. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15 (1999), 563–577.
9. JONASSEN, I. Efficient discovery of conserved patterns using a pattern graph. *Comput. Appl. Biosci.* 13 (1997), 509–522.
10. KEICH, AND PEVZNER. Finding motifs in the twilight zone. In *Annual International Conference on Computational Molecular Biology* (Washington, DC, Apr. 2002), pp. 195–204.
11. 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.