Pattern Matching on Weighted Sequences

Manolis Christodoulakis, Costas S. Iliopoulos,
Laurent Mouchard, and Kostas Tsichlas

ABSTRACT. Weighted sequences are used extensively as profiles for protein families, in the representation of binding sites and often for the representation of sequences produced by a shotgun sequencing strategy. We present various fundamental pattern matching problems on weighted sequences and their respective algorithms. In addition, we define two matching probabilistic measures and we give algorithms for each of these measures. The uncertainty introduced in weighed sequences can also be used as a means to perform approximate string matching. To the best of our knowledge, this is the first time these problems are tackled in this setting.

1 Introduction

The complete sequence of *the* human genome chromosomes is now almost obtained. Various donors of diverse ethnogeographic categories (e.g., African-American, Chinese, Caucasian, etc.) have been enrolled and offered samples that were used to obtain the final reconstructed genome sequence. Despite the fact that natural polymorphism is necessarily faded by this *consensus*, the final sequence corresponds, more or less, to the individual sequences from the donors.

Hopefully, the sequences of the genes, that contain coding information which produces proteins, are usually much more conserved than non-coding regions, and genes are what most of the research teams are looking for. The facts that several (up to six) codons can encode the same amino acid and that different amino acids can have identical chemical properties explain that close (or even distant) DNA sequences can produce protein sequences that fold identically and are able to perform exactly the same task.

In order to represent the various nucleotides that can be found at a given location, the original DNA alphabet $\{A, C, G, T\}$ has been naturally extended to a more complex one, the IUB/IUPAC where a letter represents several nucleotides, e.g. $D \rightarrow A, G, T$ or $M \rightarrow A, C$.

An important region, upstream from the gene, controls its expression. This region contains several important binding sites where additional proteins, initiating or regulating the transcription, attach themselves to the DNA sequence. These binding sites are usually very specific DNA sequences, that tolerate only very elementary changes, they are the keyholes of specific keys and any violent alteration prevents the additional protein to bind, and moreover the gene sequence from being transcripted [Stormo, 2000]. The sequences that are found between the binding sites might differ from one organism to the other.

To illustrate these concepts, let us consider the regulatory regions of the hemoglobin genes ($\alpha, \beta, \gamma, \zeta$ hemoglobins more precisely) where uppercase letters correspond to the gene sequence and bold lowercase letters correspond to a specific binding site sequence.

Hemoglobin α	... cg**gcactct**tctggtcccc	... atacccaccgATGGTGCT	...
Hemoglobin β	... tgacacaact**gcaacct**ca	... aacagacaccATGGTGCA	...
Hemoglobin γ	... gccgcta**ccgccct**gcgcg	... atgcgcgagtATGGTGCT	...
Hemoglobin ζ	... gct**gcaacct**gcccactcc	... ggcagcgcacATGTCTCT	...

We obtained four sequences we have to align:

Hemoglobin α	g	c	a	c	t	c	t
Hemoglobin β	g	c	a	a	c	c	t
Hemoglobin γ	c	c	g	c	c	c	t
Hemoglobin ζ	g	c	a	a	c	c	t
Consensus	g	c	a	M	c	c	t

As a result, various techniques are used to represent these polymorphisms, using either the extended alphabet we presented before or a more precise encoding that takes into account the relative frequency of each nucleotide. We can consider mainly two techniques, named Position Weight Matrices (PWM for short) where for each position, the probability of each nucleotide is given, and logo sequences [Schneider and Stephens, 1990].

The Position Weight Matrix [Thompson *et al.*, 1994] of a set of strings of length m is a $4 \times m$-matrix that reports the frequency of each nucleotide for all possible locations. In our example, we have:

	1	2	3	4	5	6	7
a	0	0	0.75	0.5	0	0	0
c	0.25	1	0	0.5	0.75	1	0
g	0.75	0	0.25	0	0	0	0
t	0	0	0	0	0.25	0	1

Weighted sequences are used for representing relatively short sequences such as binding sites as in the example above as well as long sequences such as profiles of protein families (see [Gusfield, 1997], 14.3). In addition, they are also used to represent complete chromosome sequences ([Gusfield, 1997], 16.15.3) that have been obtained using a whole-genome shotgun strategy [Venter and Corporation, 2001; Myers and Corporation, 2000] with an adequate cover. The cover is the average number of fragments that appear at a given location. Usually, the cover is large enough so that errors as well as SNPs are clearly spotted and removed by the consensus step.

By keeping all the information the whole-genome shotgun produces, we would like to dig out information that has been previously undetected after being faded during the consensus step (for example the consensus step wrongly chooses a symbol for a specific position than another). As a result, errors in the genome are not removed by the consensus step but remain and a probability is assigned to them based on the frequency of symbols in each position.

These are some biological examples where weighted sequences can be important. We want to be able to support various pattern matching operations on these sequences. Suppose for example that a biologist wants to find whether a sequence at hand (which may as well be a weighted sequence) occurs in a specific protein family with high probability in order to decide whether a specific protein belongs in a family of proteins. This can be accomplished by pattern matching algorithms on weighted sequences. Weighted sequences have also been used in event management systems [Wang *et al.*, 2003].

In this paper we propose various pattern matching problems on weighted sequences under two probabilistic measures. Assume a weighted sequence s of length n and a pattern p of length m over the alphabet Σ. The pattern may be a weighted sequence or not. We propose an $O(n \log m)$ algorithm for finding p in s given that the occurrence probability of p in s is larger than $\frac{1}{k}$. In addition, we propose an algorithm for finding p in s with gaps with complexity $O(mn)$.

Our results can be compared with the Weighted Suffix Tree (WST) [Iliopoulos *et al.*, 2003]. The WST has all the merits of the usual suffix tree with the difference that its construction is heavily based on the choice of the probability of occurrence $\frac{1}{k}$. It is crucial for its construction that k is a fixed and small constant. When k changes then the suffix tree must be reconstructed from scratch. In addition, for arbitrary k, the size of the WST is prohibitive. With respect to the pattern matching problem, our solutions are more general since they allow for arbitrary k, while at the same time we investigate pattern matching with gaps.

The paper is organised as follows. In Sect. 2, we provide basic definitions on weighted sequences. In Sect. 3, algorithms for the exact pattern matching problem are presented while Sect. 4 describes an algorithm for locating simple models (two strings seperated by a gap). Section 5 is dedicated to pattern matching with gaps among the occurrences of the characters of the pattern inside the text. Finally, some concluding remarks are given in Sect. 6.

2 Preliminaries

Let $\Sigma = \{1, 2, \ldots, \sigma\}$ be an alphabet of cardinality $\sigma = |\Sigma|$. A sequence s of length n is represented by $s[1..n] = s[1]s[2]\cdots s[n]$, where $s[i] \in \Sigma$ for $1 \leq i \leq n$, and $n = |s|$ is the length of s. Sequence s is also called a *solid* sequence in order to distinguish them from weighted sequences. An empty sequence is denoted by ε; we write $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$. A factor f of s is a substring of s, that is $f = s[i \ldots j]$. A model in a string s consists of two factors f_1 and f_2 of s seperated by a gap, where the gap is the number of symbols between f_1 and f_2. A weighted sequence is defined as follows.

DEFINITION 1. A weighted sequence $s = s_1 s_2 \cdots s_n$ over an alphabet Σ is a sequence of sets of couples. In particular, each s_i is a set $((1, \pi_i(1)), (2, \pi_i(2)), \ldots, (\sigma, \pi_i(\sigma)))$, where $\pi_i(q)$ is the occurrence probability of character q at position i. For every position $1 \leq i \leq n$, $\sum_{q=1}^{\sigma} \pi_i(q) = 1$.

We represent each position of the weighted sequence as a vector that contains all the symbols of the alphabet and their corresponding probabilities; if a character does not appear in a specific position then its probability will be zero. As a result, the weighted matrix representation will be used for weighted sequences. We will represent each couple $(q, \pi_i(q))$ as $\pi_i(q)_q$, as shown in the example below.

EXAMPLE 2. Consider the alphabet $\Sigma = \{A, C, G, T\}$. Then

$$s = \begin{pmatrix} 0.3_A & 0.25_A & 0_A & 0.4_A & 0.8_A & 0_A \\ 0_C & 0.25_C & 1_C & 0.2_C & 0.05_C & 0.5_C \\ 0.2_G & 0.5_G & 0_G & 0.2_G & 0.1_G & 0_G \\ 0.5_T & 0_T & 0_T & 0.2_T & 0.05_T & 0.5_T \end{pmatrix} \quad (1)$$

is the weighted matrix that represents a weighted sequence of length 6.

DEFINITION 3. A *symbol* q occurs at position i of a weighted sequence $s = s[1 \ldots n]$ if and only if the probability of occurrence of symbol q at position i is greater than zero, $\pi_i(q) > 0$

EXAMPLE 4. Let s be the weighted sequence defined in (1). Then symbol A occurs at positions 1, 2, 4, and 5 of s. Similarly, symbol C occurs at 2, 3, 4, 5, and 6.

The following definition clarifies when a solid pattern p occurs in a weighted text t.

DEFINITION 5. The solid pattern $p = p[1 \dots m]$ occurs at position i of the weighted text $t = t[1 \dots n]$ if and only if $p[j]$ occurs at position $t[i + j - 1]$ for all $1 \leqslant j \leqslant m$; that is, if and only if $\pi_{i+j-1}(p[j]) > 0$, for all $1 \leqslant j \leqslant m$ by Def. 3. We also say that p matches t at position i.

EXAMPLE 6. Let t be the weighted sequence defined in (1). Then $p = ACTA$ occurs in t at position 2, since $\pi_2(A) = 0.25$, $\pi_3(C) = 1$, $\pi_4(T) = 0.2$, and $\pi_5(A) = 0.8$.

Since each symbol at position i of the text t is assigned a probability of occurrence it is logical to assume that an occurrence of a solid pattern p in the weighted text t must also have a probability of occurrence. In this way, we define how likely is to find an occurrence of p in a specific position of t. In the following we provide such matching measures.

DEFINITION 7. Let $p = p[1 \dots m]$ be a solid pattern, and $t = t[1 \dots n]$ be a weighted text. Also assume that p matches t at position i. Then the probability of the match can be defined in one of the following ways:

1. **Multiplicative Probability** is the product of the probabilities of the symbols of t that match p: $P^i_{prod} = \prod_{j=1}^{m} \pi_{i+j-1}(p[j])$
2. **Average Probability** is the average of the probabilities of the symbols of t that match p: $P^i_{aver} = \frac{\sum_{j=1}^{m} \pi_{i+j-1}(p[j])}{m}$

Note that the Average Probability Measure (APM) allows for characters with zero probability. In general, the average probability measure is by far less strict than the Multiplicative Probability Measure (MPM) for a given cut-off probability $\frac{1}{k}$. It is easy to see that all matches of the MPM are contained in the set of matches for APM. As a result, by using the APM we are more loose with respect to the matches we are going to get while by using MPM we become much more strict.

EXAMPLE 8. Let t be the weighted sequence defined in (1) and $p = ACTA$, which occurs in t at position 2. Then the multiplicative probability of this match is $P^2_{prod} = 0.25 \cdot 1 \cdot 0.2 \cdot 0.8 = 0.04$ and the average probability of the same match is $P^2_{aver} = \frac{0.25+1+0.2+0.8}{4} = \frac{2.25}{4}$

The above definitions can be extended to cover the case where both the pattern p and the text t are weighted sequences. In this case we denote by $\pi_{t_i}(q)$ the probability of the symbol q at position i of t, and by $\pi_{p_j}(q)$ the probability of the symbol q at position j of p.

DEFINITION 9. Let $p = p[1..m]$ and $t = t[1..n]$ be weighted sequences. We say that positions $p[j]$ and $t[i]$ *match with respect to symbol* q if $\pi_{p_j}(q) \times \pi_{t_i}(q) > 0$; that is, if q occurs at both $p[j]$ and $t[i]$.

However, note that $p[j]$ and $t[i]$ may match with respect to many symbols. As a result, we define the match between two positions as follows:

DEFINITION 10. Let the pattern $p = p[1..m]$ and text $t = t[1..n]$ be weighted sequences. We say that $p[j]$ and $t[i]$ *match* if $\sum_{q=1}^{\sigma} \pi_{p_j}(q) \times \pi_{t_i}(q) > 0$; that is, if $p[j]$ and $t[i]$ match with respect to (at least one) q.

EXAMPLE 11. Consider

$$p_j = \begin{pmatrix} 0.3_A \\ 0.1_C \\ 0_G \\ 0.6_T \end{pmatrix} \quad \text{and} \quad t_i = \begin{pmatrix} 0_A \\ 0.8_C \\ 0_G \\ 0.2_T \end{pmatrix}$$

for some position j of the pattern, and some position i of the text. Then $p[j]$ and $t[i]$ match because there is at least one symbol (precisely two: C and T) that occurs in both of them. Confirm that $\sum_{q\in\{A,C,G,T\}} \pi_{p_j}(q) \times \pi_{t_i}(q) = 0.2 > 0$.

Similarly to Def. 5 we get the following definition of a match, in the case where both the pattern and the text are weighted.

DEFINITION 12. Let $p = p[1..m]$ and $t = t[1..n]$ be weighted sequences. We say that p *occurs* in (or *matches*) t at position i if and only if

$$P^i_{w\_aver} = \frac{\sum_{j=1}^{m} \sum_{\forall s\in\Sigma} \pi_{p_j}(s) \times \pi_{t_{i-m+j}}(s)}{m} > 0$$

We call $P^i_{w\_aver}$ the *weighted average probability of the match* of p at position i of t. Similarly, we can define the *weighted multiplicative probability of the match* of p at position i of t.

For the rest of the paper we will assume that p is a solid pattern. For the cases where the algorithm is applicable to weighted patterns as well, it will be explicitly indicated. Moreover, we will be interested in occurrences of the pattern with probability larger than a threshold $\frac{1}{k}$, $k \geqslant 1$. In particular:

DEFINITION 13. Given a pattern p, a text t and an integer k, we say that p matches t at position i over the probability measure $\mathcal{C}$ if and only if $\mathcal{C}^i \geqslant \frac{1}{k}$.

If we do not interpret the weigth as a probability then other matching measures may become interesting. For example, the maximum weight matching measure, where the weight of a match between a solid pattern p of

length m and a weighted text t at position i is $\max_{1\leqslant j\leqslant m}\{w_{i+j-1}(p[j])\}$. By $w_i(q)$ we represent the weight of symbol q at position i of text t. We believe that our algorithms can be extended to tackle these matching measures too.

3 Exact Pattern Matching

In this section we provide algorithms for the problem of exact pattern matching on weighted sequences. The problem we are going to tackle is the following:

PROBLEM 14. Given a solid pattern $p = p[1 \dots m]$, a weighted text $t = t[1 \dots n]$ and an arbitrary constant $k \geq 1$, find all occurrences of p in t with matching probability $\geqslant \frac{1}{k}$.

The solution to the exact pattern matching problem depends on the matching measure (multiplicative or average) in use. We present the solutions for both measures.

3.1 Average Probability

We are interested in finding all occurrences of p in t with average probability $\geqslant 1/k$, that is $P^i_{aver} \geqslant 1/k$. For a small constant k, the construction of the weighted suffix tree cannot be accomplished in linear time, as in [Iliopoulos *et al.*, 2003], because the number of factors of t is no longer linear.

For instance, consider a text t where each position contains exactly two symbols, from an alphabet Σ, each of which has probability 0.5, e.g.

$$t = \begin{matrix} A & A & G & T & \cdots & C \\ C & G & C & A & \cdots & A \end{matrix}$$

Then, every possible substring of length m has an average probability 0.5 while the number of such substrings in t is $O(n2^m)$, which results in a suffix tree of size $O(n2^m)$. Consequently, a small constant k does not allow us to adopt an approach similar to [Iliopoulos *et al.*, 2003].

An $O(n \log m)$-time algorithm, that works for arbitrary k, is possible based on the Fast Fourier Transform (FFT). First we find the number of matches between the pattern p and the text t for each position of t by using the FFT (this is the match-count problem [Gusfield, 1997]).

Let $M(t, p, i)$ be the number of characters of t and p that match when $p[1]$ is aligned with $t[i]$. We allow $p[1]$ to be aligned to the left of $t[1]$. Negative numbers specify the positions to the left of $t[1]$. As a result, the vector $M(t, p)$ stores all values of $M(t, p, i)$ for $-m + 1 \leqslant i \leqslant n$.

First, we break this problem into $|\Sigma|$ subproblems, one for each character of the alphabet Σ. Let $M_q(t, p, i)$ be the number of matches of character q when $p[1]$ is aligned to $t[i]$. The $(n + m)$-length vector $M_q(t, p)$ holds all these values. It is straightforward to see that:

$$M(t,p) = \sum_{\forall q \in \Sigma} M_q(t,p) \quad (2)$$

As a result, the problem is reduced to finding the match-count for each character. For each character q construct two bit vectors $\bar{p}_q$ and $\bar{t}_q$, where the i-th position is 1 if q occurs in $p[i]$ and $t[i]$ respectively with non-zero probability, otherwise it is 0. We pad the right end of $\bar{p}_q$ with n zeros and the right end of $\bar{t}_q$ with m zeros. This is necessary for the application of the FFT algorithm [Fischer and Paterson, 1974]. By renumbering the indices of both bit vectors to run from 0 to $n+m-1$, the number of matches for symbol q at position $t[i]$ is:

$$M_q(t,p,i) = \sum_{j=0}^{n+m-1} \bar{p}_q[j] \times \bar{t}_q[j+i] \quad (3)$$

where the indices are modulo $n+m$. The extra zeros were added so that when the right end of $\bar{p}_q$ is to the right of the end of t then no additional false matches are counted. This is the cyclic correlation of $\bar{p}_q$ and $\bar{t}_q$, and $M_q(t,p)$ is computed by the FFT algorithm in $O(n \log m)$ operations. As a result the vector $M(t,p)$ by Eq. 2 can be computed in $O(|\Sigma| n \log m)$ time. To this point, we have computed the occurrences of p in t without taking into account the probabilities. By scanning $M(t,p)$ we can report all positions that contain m in $O(m+n)$ time.

The problem now is to compute the average probability of each occurrence of p in t. A straightforward computation will lead to a time complexity of $O(mn)$. If the number of occurrences is up to $\frac{n \log m}{m}$ then we compute the average probability straightforwardly in $O(n \log m)$ time. However, if the number of occurrences is larger than $\frac{n \log m}{m}$ then we use the FFT algorithm again to compute the probabilities.

In the same manner as the construction of $\bar{p}_q$ and $\bar{t}_q$ we construct the vectors $\hat{p}_q$ and $\hat{t}_q$ containing the probabilities of occurrence of character q in each position. Then in the same way:

$$\hat{M}(t,p) = \sum_{\forall q \in \Sigma} \hat{M}_q(t,p) \quad (4)$$

where $\hat{M}_q(t,p)$ is the sum of the probabilities of occurrence of p in t for character q.

$$\hat{M}_q(t,p,i) = \sum_{j=0}^{n+m-1} \hat{p}_q[j] \times \hat{t}_q[j+i] \quad (5)$$

This is the cyclic correlation of two vectors and by using the FFT algorithm it can be computed in $O(n \log m)$ time. The computation of $\hat{M}(t,p)$ needs $O(|\Sigma| n \log m)$ time by Eq. 4. As a result, by using vectors $M(t,p)$ and $\hat{M}(t,p)$ we can find all occurrences with average probability greater than $\frac{1}{k}$ by just locating positions i in $M(t,p)$ which contain m, and checking whether $\frac{\hat{M}(t,p,i)}{m} > \frac{1}{k}$. Note that this algorithm will work even in the case where p is a weighted sequence, while k can be arbitrarily large.

3.2 Multiplicative Probability

The exact pattern matching problem using the multiplicative probability measure can be solved by constructing the weighted suffix tree [Iliopoulos *et al.*, 2003] of the text t in $O(n)$ time and space, and then traversing the tree top-down to locate occurrences of the pattern. Iliopoulos et al. proved in [Iliopoulos *et al.*, 2003] that the exact pattern matching can be solved in this manner in $O(n+m)$ time, assuming that the alphabet Σ is of fixed size. However, this method works well only in the case where k is a small constant and stays fixed; otherwise the number of factors in the suffix tree would grow exponentially and therefore the size of the suffix tree and its construction time would grow similarly. Additionally if k changes then the tree must be constructed all over again.

Assume that pattern p is a solid sequence and that k is arbitrary. Pattern p has an occurrence at position $t[i]$ with probability $P^i_{prod} = \prod_{j=1}^{m} \pi_{i+j-1}(p[j])$. Applying the logarithm we get:

$$\log(P^i_{prod}) = \sum_{j=1}^{m} \log(\pi_{i-m+j}(p[j])) \quad (6)$$

By this simple trick we got rid of the product and so we can apply the same $O(n \log m)$ algorithm described in Sect. 3.1 to find all occurrences. Note that this trick does not work when both the pattern and the text are weighted sequences. In this case, the only known algorithm is the straightforward $O(mn)$ dynamic programming algorithm.

4 Searching for a Model

In this section we consider the following problem, concerned with 1-Dimensional range searching [de Berg *et al.*, 2000].

PROBLEM 15. Given solid patterns $p = p[1 \ldots m]$ and $q = q[1 \ldots m']$, a weighted text $t = t[1 \ldots n]$, a constant $k \geq 1$, and constant α, $0 \leq \alpha \leq n-m$, find all occurrences of p followed by a possible gap of size at most α and then by q, in t with matching probability $\geqslant 1/k$.

First by using the algorithms of Sect. 3.1 – 3.2 we find all occurrences of p and q in t and we construct two data structures D_p and D_q that store in increasing order the positions for p and q respectively in t. The time complexity for this step is $O(n \log m)$ for the matching measures we consider.

We traverse structure D_p and for each occurrence of p we find the interval of occurrences of q in structure D_q. D_p can be implemented as a simple linear list. Assume the occurrence of p at position j. Then, the query to structure D_q is of the form $[j + m - 1, j + m + \alpha]$. If we allow overlapping then the query will be of the form $[j, j + m + \alpha]$. If D_q is implemented as a binary tree then since the maximum number of positions in D_p and D_q is n, this procedure is completed in $O(n \log n)$ time. The use of finger search trees [Brodal *et al.*, 2003] reduces the time complexity to $O(n)$ by exploiting the fact that the list D_p is sorted. This means that the ranges we are searching are increasing in value and so instead of initiating the search from the root of D_q we initiate it from the leaves where the previous search ended. In addition, it is easy to change the solution so that a lower bound on the length of gaps applies, that is the gap is in the range $[\alpha_s, \alpha_l]$.

5 Pattern Matching with Gaps

Finally, in this section we consider the problem of pattern matching on weighted sequences by allowing gaps between occurrences of successive symbols of the pattern p in the text t.

PROBLEM 16. Given a solid pattern $p = p[1 \dots m]$, a weighted text $t = t[1 \dots n]$, a constant $k \geq 1$ and a constant α, find all occurrences of p, allowing the existence of gaps between the occurrences of consecutive symbols of p, in t with probability larger than $1/k$. The gap g_i between the occurrences of $p[i]$ and $p[i + 1]$ in t must satisfy: $|g_i| \leq \alpha$.

EXAMPLE 17. Let $p = TGA$,

$$t = \begin{pmatrix} 0.3_A & 0.25_A & 0_A & 0.4_A & 0.8_A & 0_A \\ 0_C & 0.25_C & 1_C & 0.2_C & 0.05_C & 0.5_C \\ 0.2_G & 0.5_G & 0_G & 0.2_G & 0.1_G & 0_G \\ 0.5_T & 0_T & 0_T & 0.2_T & 0.05_T & 0.5_T \end{pmatrix}$$

$k = 10$, and $\alpha = 1$. Then p occurs in t at position 1, with α-bounded gaps, since $\pi_1(T) = 0.5$, $\pi_2(G) = 0.5$, and $\pi_4(A) = 0.4$; thus: $P^4_{prod} = \pi_1 \cdot \pi_2 \cdot \pi_4 = 0.1 \geq \frac{1}{k}$. Note the gap (of size 1) between the second and the third symbols of p inside t.

Since gaps are allowed between the occurrences of symbols of p, it is possible that more than one occurrences of p end in a single position i in t.

In what follows we will compute for each position i of t only one occurrence of p, namely the occurrence of p at i with the maximum probability, given that this probability is larger than $1/k$.

5.1 α-bounded Gaps

The solution we provide is based on the dynamic programming approach. The basic idea of the algorithm is the computation of continuously increasing prefixes of pattern p in the weighted sequence t. Define the set of all non-empty prefixes of pattern p to be $\Pi(p)$. Formally, $\Pi(p) = \{p[1], p[1..2], p[1..3], \ldots, p[1..m]\}$. We denote by $\Delta(p)$ the set of positions ℓ in the sequence t such that there is an occurrence of p with α-bounded gaps that ends at position ℓ given that the probability of occurrence is greater than $\frac{1}{k}$. Note that when extending the prefix $p[1..i-1]$ to $p[1..i]$, due to an occurrence of character $p[i]$, we choose always the prefix $p[1..i-1]$ with the maximum probability.

Let D be an $(m+1) \times (n+1)$ matrix. Each $D(i,j)$, for $1 \leq i \leq m$ and $1 \leq j \leq n$, will indicate whether there is an occurrence of the prefix $p[1..i]$ ending at position $t[j]$, where the gaps are bounded by α and the probability of occurrence is larger than $1/k$. For this problem the base conditions are:

$$\text{(a) } D(i,0) = 0,\ 0 \leq i \leq m \quad \text{AND} \quad \text{(b) } D(0,j) = j,\ 0 \leq j \leq n$$

The base condition $D(i,0) = 0$ is correct since there is no occurrence of prefixes of p in the empty string. The base condition $D(0,j) = j$ is also correct since the empty string is assumed to match each of the characters of t. Before defining the recurrence relation it is necessary to give some notation with respect to probabilities. Assume that we are currently working on cell $D(i,j)$. Attached to this cell is a list of occurrences of prefixes $p[1..i-1]$, of maximum size α (the gap size). By $P^{max}_{i-1,j}$ we denote the occurrence of the prefix $p[1..i-1]$ with the maximum probability (out of those occurrences of $p[1..i-1]$ that appear in the list of the cell $D(i,j)$). The recurrence relation for $D(i,j)$ (without considering the update of the lists) is as follows:

$$D(i,j) = \begin{cases} j & \text{if } (p[i] \text{ in } t[j]) \ \&(j - D(i-1,j-1) - 1 \leqslant \alpha) \ \& \\ & (D(i-1,j-1) > 0) \ \&(P^{max}_{i-1,j} \times \pi_j(p[i]) \geqslant \frac{1}{k}) \\ D(i,j-1) & \text{if (previous case does not hold) \&} \\ & (j - D(i,j-1) - 1 < \alpha) \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

If $D(i,j) = j$, then there is a match between $t[j]$ and $p[i]$ while the prefix $p[1..i-1]$ may have an occurrence at $D(i-1,j-1)$ and the gap is

$\leqslant \alpha$. In addition, the probability of occurrence for this prefix is $\geqslant \frac{1}{k}$. If $D(i,j) = D(i,j-1)$, then there is no match between $t[j]$ and $p[i]$ and thus we are not able to extend the occurrence of prefix $p[1..i-1]$ to $p[1..i]$. As a result, $D(i,j) = D(i,j-1)$, as long as the gap invariant is satisfied. In every other case, $D(i,j) = 0$.

For each position the occurrence with the maximum probability (given that this probability is larger than $1/k$) is given in the m-th row. The only remaining detail are the lists of prefixes that must be maintained during the construction of the matrix. This list must support three operations: 1) insert a new element in the head of the list, 2) delete an element from the tail of the list and 3) find the maximum element among the elements in the list. By using a heap-ordered queue we can support all three operations in constant time. If we want to store the whole matrix so that after its computation we are able to trace it back for occurrences then each cell must have its own list. This means, that when moving from $D(i-1,j)$ to $D(i,j)$ we have to copy the whole list and maybe make changes to its head and tail. This means that the time complexity of the algorithm will be $O(mn\alpha)$ and the same goes for the space complexity.

However, based on the facts that the matrix D is computed column by column and that the only difference between two adjacent lists are only in their head and tail we can reduce the time complexity to $O(mn)$. We use a simplified version of the persistent lists described in [Kaplan *et al.*, 2000]. These lists support the operations of removing an element from the tail of a list, adding an element to its head, identifying the maximum element and copying the list (in fact it records the history of the structure with respect to update operations) in constant amortized time (for worst case constant time complexity refer to [Kaplan, 1998]). This means, that over a sequence of n operations the total cost will be $O(n)$.

The algorithm presented in this section refers to the multiplicative probability measure. However, it can easily be adapted for the average probability by adding the maximum probabilities of the characters of the pattern instead of multiplying them. For unbounded gaps we can use the same algorithm by setting $\alpha = n - m + 1$.

6 Conclusion and Open Problems

In the following table the results described in this paper are given.

Problem	Multiplicative	Average
Exact Pattern Matching	$O(n \log m)$	$O(n \log m)$
Patterns Separated by Gap	$O(n \log m)$	$O(n \log m)$
α-bounded gaps	$O(mn)$	$O(mn)$

We presented algorithms on various problems on weighted sequences. These sequences seem to model various real life problems. Apart from the use of weighted sequences that was described in the introduction, they also appear in the field of event management for complex networks, where each event has a timestamp [Wang *et al.*, 2003], as well as in DNA micro-array analysis, where expression levels of genes are recorded under different experimental conditions.

Weighted sequences are approximate by definition. However, this approximation measure is not the same as the usual distance metrics, like the Hamming distance or the edit distance. The probabilities in the weighted sequence provide a measure of our uncertainty concerning the data of the sequence. The error introduced by metrics like Hamming distance, provide a measure of the error that really exists between two sequences. As a result, it would be very interesting to design approximate pattern matching algorithms for weighted sequences.

BIBLIOGRAPHY

[Brodal *et al.*, 2003] G. S. Brodal, G. Lagogiannis, C. Makris, A. Tsakalidis, and K. Tsichlas. Optimal finger search trees in the pointer machine. *Journal of Computer and System Sciences, Special Issue on STOC 2002*, 67(2):381–418, 2003.

[de Berg *et al.*, 2000] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[Fischer and Paterson, 1974] M. J. Fischer and M. S. Paterson. String matching and other products. Technical report, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[Gusfield, 1997] D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.

[Iliopoulos *et al.*, 2003] C. S. Iliopoulos, C. Makris, I. Panagis, K. Perdikuri, E. Theodoridis, and A. Tsakalidis. Computing the repetitions in a weighted sequence using weighted suffix trees. In *European Conference on Computational Biology*, pages 539–540, 2003. poster paper.

[Kaplan *et al.*, 2000] H. Kaplan, C. Okasaki, and R. E. Tarjan. Simple confluently persistent catenable lists. *SIAM Journal on Computing*, 30(3):965–977, 2000.

[Kaplan, 1998] H. Kaplan. *Purely Functional Lists*. PhD thesis, Department of Computer Science, Princeton University, 1998.

[Myers and Corporation, 2000] E.W. Myers and Celera Genomics Corporation. The whole-genome assembly of drosophila. *Science*, 287:2196–2204, 2000.

[Schneider and Stephens, 1990] T.D. Schneider and R.M. Stephens. Sequence logos: A new way to display consensus sequences. *Nucleic Acids Res.*, 18:6097–6100, 1990.

[Stormo, 2000] G.D. Stormo. Dna binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

[Thompson *et al.*, 1994] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22:4673–4680, 1994.

[Venter and Corporation, 2001] J.C. Venter and Celera Genomics Corporation. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

[Wang *et al.*, 2003] H. Wang, C. S. Perng, W. Fan, S. Park, and P. S. Yu. Indexing weighted-sequences in large databases. In *Proc. of the 19th International Conference on Data Engineering (ICDE)*, 2003.

Manolis Christodoulakis, Costas S. Iliopoulos, and Kostas Tsichlas
Department of Computer Science, King's College London
Strand, London WC2R 2LS, England
{manolis,csi,kostas}@dcs.kcl.ac.uk

Laurent Mouchard
ABISS, Atelier Biology, Informatics, Statistics and Sociolinguistics
Université de Rouen, 76821 Mont Saint Aignan Cedex, France
Laurent.Mouchard@univ-rouen.fr