On-Line Pattern Matching on Uncertain Sequences and Applications

Carl Barton[1], Chang Liu[2], and Solon P. Pissis[2](✉)

[1] The Blizard Institute, Barts and The London School of Medicine and Dentistry, Queen Mary University of London, London, UK
c.barton@qmul.ac.uk

[2] Department of Informatics, King's College London, London, UK
{chang.2.liu,solon.pissis}@kcl.ac.uk

Abstract. We study the fundamental problem of pattern matching in the case where the string data is *weighted*: for every position of the string and every letter of the alphabet a probability of occurrence for this letter at this position is given. Sequences of this type are commonly used to represent uncertain data. They are of particular interest in computational molecular biology as they can represent different kind of ambiguities in DNA sequences: distributions of SNPs in genomes populations; position frequency matrices of DNA binding profiles; or even sequencing-related uncertainties. A weighted string may thus represent many different strings, each with probability of occurrence equal to the product of probabilities of its letters at subsequent positions. In this article, we present new average-case results on pattern matching on weighted strings and show how they are applied effectively in several biological contexts. A free open-source implementation of our algorithms is made available.

1 Introduction

Uncertain sequences are common in many applications: (i) data measurements such as imprecise sensor measurements; (ii) flexible modelling of DNA sequences such as DNA binding profiles; (iii) when observations are private and thus sequences of observations may have artificial uncertainty introduced deliberately. For example, in computational molecular biology an uncertain sequence can be used to incorporate SNP distributions from a population of genomes into a reference sequence. This process can be realised by a IUPAC-encoded sequence [3,13] or by directly incorporating the results of SNP studies such as [11,19]. An uncertain sequence can also be used as a flexible model of DNA sequences such as DNA binding profiles, and is known as *position frequency matrix* [18].

As *pattern matching* is a core computational task in many real-world applications, we focus here on designing efficient on-line algorithms for pattern matching on uncertain sequences. On-line pattern matching algorithms process the text position-by-position, in the order that it is fed to the algorithm, without having the entire text at hand. Hence this type of algorithms is useful when one wishes

© Springer International Publishing AG 2016
T-H.H. Chan et al. (Eds.): COCOA 2016, LNCS 10043, pp. 547–562, 2016.
DOI: 10.1007/978-3-319-48749-6_40

to query for one or a few patterns in potentially many texts without having to pre-compute and store an index over the texts.

We start with a few definitions to explain our results. An *alphabet* Σ is a finite non-empty set of size σ, whose elements are called *letters.* A *string* on an alphabet Σ is a finite, possibly empty, sequence of elements of Σ. The zero-letter sequence is called the *empty string*, and is denoted by ε. The *length* of a string x is defined as the length of the sequence associated with the string x, and is denoted by $|x|$. We denote by $x[i]$, for all $0 \leq i < |x|$, the letter at index i of x. Each index i, for all $0 \leq i < |x|$, is a position in x when $x \neq \varepsilon$. The i-th letter of x is the letter at position $i-1$ in x. We refer to any string $x \in \Sigma^q$ as a *q-gram.*

The *concatenation* of two strings x and y is the string of the letters of x followed by the letters of y; it is denoted by xy. A string x is a *factor* of a string y if there exist two strings u and v, such that $y = uxv$. Consider the strings x, y, u, and v, such that $y = uxv$, if $u = \varepsilon$ then x is a *prefix* of y, if $v = \varepsilon$ then x is a *suffix* of y. Let x be a non-empty string and y be a string, we say that there exists an *occurrence* of x in y, or more simply, that x *occurs in* y, when x is a factor of y. Every occurrence of x can be characterised by a position in y; thus we say that x occurs at the *starting position* i in y when $y[i\,..\,i+|x|-1] = x$.

A weighted string x of length n on an alphabet Σ is a finite sequence of n sets. Every $x[i]$, for all $0 \leq i < n$, is a set of ordered pairs $(s_j, \pi_i(s_j))$, where $s_j \in \Sigma$ and $\pi_i(s_j)$ is the probability of having letter s_j at position i. Formally, $x[i] = \{(s_j, \pi_i(s_j)) | s_j \neq s_\ell \text{ for } j \neq \ell, \text{ and } \sum_j \pi_i(s_j) = 1\}$. A letter s_j *occurs* at position i of a weighted string x if and only if the *occurrence probability* of letter s_j at position i, $\pi_i(s_j)$, is greater than 0. A string u of length m is a *factor* of a weighted string if and only if it occurs at starting position i with *cumulative occurrence probability* $\prod_{j=0}^{m-1} \pi_{i+j}(u[j]) > 0$. Given a *cumulative weight threshold* $1/z \in (0,1]$, we say that factor u is *valid*, or equivalently that factor u has a valid occurrence, if it occurs at starting position i and $\prod_{j=0}^{m-1} \pi_{i+j}(u[j]) \geq 1/z$. Similarly, we say that letter s_j at position i is valid if $\pi_i(s_j) \geq 1/z$. For succinctness of presentation, if $\pi_i(s_j) = 1$ the set of pairs is denoted only by the letter s_j; otherwise it is denoted by $[(s_{j_1}, \pi_i(s_{j_1})), \ldots, (s_{j_k}, \pi_i(s_{j_k}))]$.

Suppose we are given a cumulative weight threshold $1/z$. Given a (weighted) string u and a weighted string v, both of length m, we say that u and v *match*, denoted by $u =_z v$, if there exists a (valid) factor of u of length m that is also a valid factor of v of length m. Given a string u and a weighted string v, both of length m, and a non-negative integer $k < m$, we say that u and v *match with k-mismatches*, denoted by $u =_{z,k} v$, if when at most k letters in u were replaced to create a new string u' then $u' =_z v$. We consider the following three problems.

WEIGHTEDTEXTMATCHING (WTM)
Input: a string x of length m, a weighted string y of length $n > m$, and a cumulative weight threshold $1/z \in (0,1]$
Output: all positions i of y such that $x =_z y[i\,..\,i+m-1]$

GeneralWeightedPatternMatching (GWPM)
Input: a weighted string x of length m, a weighted string y of length $n > m$, and a cumulative weight threshold $1/z \in (0, 1]$
Output: all positions i of y such that $x =_z y[i \,.\,. i + m - 1]$

ApproxWeightedTextMatching (AWTM)
Input: a string x of length m, a weighted string y of length $n > m$, a non-negative integer $k < m$, and a cumulative weight threshold $1/z \in (0, 1]$
Output: all positions i of y such that $x =_{z,k} y[i \,.\,. i + m - 1]$

Our computational model. We assume word-RAM model with word size $w = \Omega(\log(nz))$. We consider the log-probability model of representations of weighted strings in which probability operations can be realised exactly in $\mathcal{O}(1)$ time. We assume that $\sigma = \mathcal{O}(1)$ since the most commonly studied alphabet is $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. In this case a weighted string of length n has a representation of size $\mathcal{O}(n)$. A position on a weighted string is viewed as a non-empty subset of the alphabet such that each letter of this subset has probability of occurrence greater than 0. For the analysis, we assume all possible non-empty subsets of the alphabet are independent and identically distributed random variables uniformly distributed.

Related results. Problem WeightedTextMatching can be solved in time $\mathcal{O}(n \log z)$ [14]. Moreover, in [14], the authors showed that their solution to WeightedTextMatching can be applied to the well-known *profile-matching* problem [17]. Problem GeneralWeightedPatternMatching can be solved in time $\mathcal{O}(zn)$ [4]. Problem ApproxWeightedTextMatching can be solved in time $\mathcal{O}(n\sqrt{m \log m})$ using FFTs [2]. All these results are worst-case complexities. Problem WeightedTextMatching can be solved in average-case search time $o(n)$ for *weight ratio* $\frac{z}{m} < \min\{\frac{1}{\log z}, \frac{\log \sigma}{\log z(\log m + \log\log \sigma)}\}$ [5].

Our contribution. We provide efficient on-line algorithms for solving these problems and provide their *average-case* analysis, obtaining the following results. Note that preprocessing resources below denote worst-case complexities, $0 < c < 1/2$ is an absolute constant, $v = \frac{2^\sigma - 1}{2^{\sigma - 1}}$, $d = 1 + (1 - c)\log_v(1 - c) + c \log_v c$, and $a = 4\sqrt{c(1 - c)}$.

Problem	Preprocessing space	Preprocessing time	Search time	Conditions
WTM	$\mathcal{O}(m)$	$\mathcal{O}(m)$	$\mathcal{O}(\frac{nz \log m}{m})$	
GWPM	$\mathcal{O}(zm)$	$\mathcal{O}(zm)$	$\mathcal{O}(\frac{nz \log m}{m})$	
AWTM	$\mathcal{O}(\sigma^q)$	$\mathcal{O}(mq\sigma^q)$	$\mathcal{O}(\frac{nz(\log m + k)}{m})$	$q \geq \frac{3 \log_v m - \log_v a}{d}$, $\frac{k}{m} \leq c - \frac{2cq}{m}$

We also provide extensive experimental results, using both real and synthetic data: (i) we show that our implementations outperform state-of-the-art approaches by more than one order of magnitude; (ii) furthermore, we demonstrate the suitability of the proposed algorithms in a number of real biological contexts.

2 Tools for Standard and Weighted Strings

Suffix trees are used as computational tools; for an introduction see [9]. The *suffix tree* of a non-empty standard string y is denoted by $\mathcal{T}(y)$.

Fact 1 [10]. *Given a non-empty string y of length n, $\mathcal{T}(y)$ can be constructed in time and space $\mathcal{O}(n)$. Checking whether a string x of length m occurs in y can be performed in time $\mathcal{O}(m)$ using $\mathcal{T}(y)$.*

We next define some primitive operations on weighted strings. Suppose we are given a cumulative weight threshold $1/z$. Let u be a string of length m and v be a weighted string of length m. We define operation $\mathsf{VER1}(u, v, z)$: it returns *true* if $u =_z v$ and *false* otherwise. Let u be a weighted string of length m and v be a weighted string of length m. We define operation $\mathsf{VER2}(u, v, z)$: it returns *true* if $u =_z v$ and *false* otherwise. For a weighted string v, by $\mathcal{H}(v)$ we denote a string obtained from v by choosing at each position the heaviest letter, that is, the letter with the maximum probability (breaking ties arbitrarily). We call $\mathcal{H}(v)$ the *heavy string* of v. Let u be a string of length m and v be a weighted string of length m. Given a non-negative integer $k < m$, we can check whether $u =_{z,k} v$ using Function $\mathsf{VER3}$. An implementation of $\mathsf{VER3}$ is provided below. Intuitively, we replace at most k letters of u with the heaviest letter in v based on how much these letter replacements contribute to the cumulative probability.

```
Function VER3($u, v, z, k$)
    $v' \leftarrow \mathcal{H}(v)$;
    $A \leftarrow$ EmptyList();
    foreach $i$ such that $u[i] \neq v'[i]$ do
        if $\pi_i(u[i]) = 0$ then
            $u[i] \leftarrow v'[i]$;
            $k \leftarrow k - 1$;
            if $k < 0$ then
                return false;
        else
            $\alpha \leftarrow \pi_i(v'[i])/\pi_i(u[i])$;
            $A \leftarrow$ Insert($< i, \alpha >$);
    Find the $k$th largest element in $A$ with respect to $\alpha$;
    Add the $k$ largest elements of $A$ with respect to $\alpha$ to set $A_k$;
    foreach $< i, \alpha > \in A_k$ do
        $u[i] \leftarrow v'[i]$;
    if $\prod_{j=0}^{m-1} \pi_j(u[j]) \geq 1/z$ then
        return true;
    return false;
```

Lemma 1. *VER1, VER2, and VER3 can be implemented to work in time $\mathcal{O}(m)$, $\mathcal{O}(mz)$, and $\mathcal{O}(m)$, respectively.*

Proof. For VER1 we can check whether $u =_z v$ in time $\mathcal{O}(m)$ by checking $\prod_{j=0}^{m-1} \pi_j(u[j]) \geq 1/z$. For VER2 we can check whether $u =_z v$ in time $\mathcal{O}(mz)$ using the algorithm of [4].

Let us denote by $E = \{e_1, \dots, e_{|E|}\}$, $|E| \leq k$, the set of positions of the input string u, which we replace in VER3 by the heaviest letter of v. We denote the resulting string by u'. Towards contradiction, assume we can guess a set $F = \{f_1, \dots, f_{|F|}\}$, $|F| \leq k$, of positions over u resulting in another string u'' such that $P_{u''} = \prod_{j=0}^{m-1} \pi_j(u''[j]) > P_{u'} = \prod_{j=0}^{m-1} \pi_j(u'[j]) \geq P_u = \prod_{j=0}^{m-1} \pi_j(u[j])$. It must hold that

$$\frac{P_{u''}}{P_u} = \frac{\pi_{f_1}(u''[f_1]) \dots \pi_{f_{|F|}}(u''[f_{|F|}])}{\pi_{f_1}(u[f_1]) \dots \pi_{f_{|F|}}(u[f_{|F|}])} > \frac{P_{u'}}{P_u} = \frac{\pi_{e_1}(u'[e_1]) \dots \pi_{e_{|E|}}(u'[e_{|E|}])}{\pi_{e_1}(u[e_1]) \dots \pi_{e_{|E|}}(u[e_{|E|}])}.$$

In case $E = F$, this implies that there exist letters heavier than the corresponding heaviest letters, a contradiction. In case $E \neq F$, given that there exists no letter heavier than the heaviest letter at each position, this implies that there exists an element $< i, \alpha >$, $i \in F$, in list A which is the rth largest, $r \leq k$, with respect to α, and it is larger than the rth element picked by VER3, a contradiction. All operations in VER3 can be trivially done in time $\mathcal{O}(m)$ except for finding the kth largest element in a list of size $\mathcal{O}(m)$, which can be done in time $\mathcal{O}(m)$ using the *introselect* algorithm [16]. $\square$

We say that u is a *(right-)maximal factor* of a weighted string x at position i if u is a valid factor of x starting at position i and no string $u' = u\alpha$, for $\alpha \in \Sigma$, is a valid factor of x at this position.

Fact 2 [1]. *A weighted string has at most z different maximal factors starting at a given position.*

3 Algorithms

Let us start with a few auxiliary definitions. An *indeterminate string* x of length m on an alphabet Σ is a finite sequence of m sets, such that $x[i] \subseteq \Sigma$, $x[i] \neq \emptyset$, for all $0 \leq i < m$. Naturally, we refer to any indeterminate string of length q as *indeterminate q-gram*. We say that two indeterminate strings x and y *match*, denoted by $x \approx y$, if $|x| = |y|$ and for each $i = 0, \dots, |x|-1$, we have $x[i] \cap y[i] \neq \emptyset$. Intuitively, we view a weighted string as an indeterminate string in order to conduct the average-case analysis of the algorithms according to our model.

3.1 Weighted Text Matching

In this section we present a remarkably simple and efficient algorithm to solve problem WeightedTextMatching. We start by providing a lemma on the probability that a random indeterminate q-gram and a standard q-gram match.

Lemma 2. *Let u be a standard q-gram and v be a uniformly random indeterminate q-gram. The probability that $u \approx v$ is no more than $(\frac{2^{\sigma-1}}{2^{\sigma}-1})^q$, which tends to $\frac{1}{2^q}$ as σ increases.*

Proof. There are less than 2^{σ} non-empty subsets of an alphabet of size σ. Let $a \in \Sigma$, then $2^{\sigma-1}$ of these subsets include a. Clearly then the probability that two positions of u and v match is no more than $\frac{2^{\sigma-1}}{2^{\sigma}-1}$. Therefore the probability of them matching at every position is no more than $(\frac{2^{\sigma-1}}{2^{\sigma}-1})^q$. □

```
**Algorithm** WTM($x, m, y, n, z, q$)
    Construct $\mathcal{T}(x)$;
    $i \leftarrow 0$;
    **while** $i < n - m + 1$ **do**
        $j \leftarrow i + m - q$;
        Let $\mathcal{A}$ denote the set of all valid $q$-grams starting at position $j$ in $y$;
        **foreach** $s \in \mathcal{A}$ **do**
            Check if $s$ occurs in $x$ using $\mathcal{T}(x)$;
        **if** *no* $s \in \mathcal{A}$ *occurs in* $x$ **or** $\mathcal{A} = \emptyset$ **then**
            $i \leftarrow j + 1$;
        **else**
            **if** VER1($x, y[i \mathinner{.\,.} i + m - 1], z$) $=$ *true* **then**
                **output** $i$;
            $i \leftarrow i + 1$;
```

Theorem 3. *Algorithm WTM solves problem* WeightedTextMatching *in average-case search time $\mathcal{O}(\frac{nz \log m}{m})$ if we set $q \geq 3 \log_{\frac{2^{\sigma}-1}{2^{\sigma-1}}} m$, which tends to $3 \log_2 m$ as σ increases. The worst-case preprocessing time and space is $\mathcal{O}(m)$.*

Proof. By Fact 1 the time and space required for constructing $\mathcal{T}(x)$ is $\mathcal{O}(m)$. We consider a sliding window of size m of y and read q-grams backwards from the end of this window and check if they occur anywhere within x. By Fact 1 this check can be done in time $\mathcal{O}(q)$ per q-gram. If a q-gram occurs anywhere in x then we verify the entire window, otherwise we shift the window $m - q + 1$ positions to the right. Clearly none of the skipped positions can be the starting position of any occurrence of x as if this was the case, the q-gram must occur in x; so the algorithm is correct. Verifying (all starting positions of) the window takes time $\mathcal{O}(m^2)$ by Lemma 1 and the probability that a q-gram matches within a pattern of length $m > q$ is no more than $m(\frac{2^{\sigma-1}}{2^{\sigma}-1})^q$ by Lemma 2. We note that reading the q-grams takes time $\mathcal{O}(zq)$ per position by Fact 2, so to achieve the claimed runtime we must pick a value for q such that the expected cost per window is $\mathcal{O}(zq)$. This is achieved when $\mathcal{O}(m^3(\frac{2^{\sigma-1}}{2^{\sigma}-1})^q) = \mathcal{O}(zq)$. It is always true when $q \geq 3 \log_{\frac{2^\sigma-1}{2^{\sigma-1}}} m$, which tends to $3 \log_2 m$ as σ increases. There are $\mathcal{O}(\frac{n}{m})$ non-overlapping windows of length m and this proves the theorem. □

3.2 General Weighted Pattern Matching

In this section we present an algorithm, denoted by GWPM, to solve problem GeneralWeightedPatternMatching. Algorithm GWPM largely follows algorithm WTM.

Lemma 3. *Let u and v be uniformly random indeterminate q-grams. The probability that $u \approx v$ is no more than $(1-(1-(\frac{2^{\sigma-1}}{2^\sigma-1})^2)^\sigma)^q$, which tends to $(1-(\frac{3}{4})^\sigma)^q$ as σ increases.*

Proof. There are less than 2^σ non-empty subsets of an alphabet of size σ. Let $a \in \Sigma$, then $2^{\sigma-1}$ of these subsets include a. Clearly then the probability that a occurs at both positions is no more than $(\frac{2^{\sigma-1}}{2^\sigma-1})^2$. It then follows that the probability that the two sets have a non-empty intersection is no more than $1-(1-(\frac{2^{\sigma-1}}{2^\sigma-1})^2)^\sigma$. Therefore the probability that all positions have a non-empty intersection is no more than $(1-(1-(\frac{2^{\sigma-1}}{2^\sigma-1})^2)^\sigma)^q$. □

```
Algorithm GWPM($x, m, y, n, z, q, \Sigma$)
    Let $\mathcal{F} = \{x_1, \ldots, x_f\}$ denote the set of all valid factors of length $m$ of $x$;
    if $\mathcal{F} = \emptyset$ then
        return;
    Construct string $X = x_1\$ \ldots \$x_f$, where $\$ \notin \Sigma$;
    Construct $\mathcal{T}(X)$;
    $i \leftarrow 0$;
    while $i < n - m + 1$ do
        $j \leftarrow i + m - q$;
        Let $\mathcal{A}$ denote the set of all valid $q$-grams starting at position $j$ in $y$;
        foreach $s \in \mathcal{A}$ do
            Check if $s$ occurs in $X$ using $\mathcal{T}(X)$;
        if no $s \in \mathcal{A}$ occurs in $X$ or $\mathcal{A} = \emptyset$ then
            $i \leftarrow j + 1$;
        else
            if VER2($x, y[i \mathinner{..} i + m - 1], z$) = true then
                output $i$;
            $i \leftarrow i + 1$;
```

Theorem 4. *Algorithm GWPM solves problem* GeneralWeightedPatternMatching *in average-case search time $\mathcal{O}(\frac{nz \log m}{m})$ if we set $q \geq 3 \log_u m$, where*

$u = \frac{(2^\sigma-1)^{2\sigma}}{(2^\sigma-1)^{2\sigma}-((2^\sigma-1)^2-2^{2\sigma-2})^\sigma}$, *which tends to* $3\log_{1/(1-(\frac{3}{4})^\sigma)} m$ *as* σ *increases. The worst-case preprocessing time and space is* $\mathcal{O}(zm)$.

Proof. Let $x_1, x_2, \ldots, x_f$ denote all f valid factors of length m of x. We construct string $X = x_1\$x_2\$\ldots\$x_f$, where $\$ \notin \Sigma$. By Facts 1 and 2 the time and space required for constructing $\mathcal{T}(X)$ is $\mathcal{O}(zm)$. Plugging Lemmas 1 and 3 to the proof of Theorem 3 yields the result. This is achieved when $\mathcal{O}(m^3 z(1-(1-(\frac{2^{\sigma-1}}{2^\sigma-1})^2)^\sigma)^q) = \mathcal{O}(zq)$. It is always true when $q \geq 3\log_u m$, where $u = \frac{(2^\sigma-1)^{2\sigma}}{(2^\sigma-1)^{2\sigma}-((2^\sigma-1)^2-2^{2\sigma-2})^\sigma}$, which tends to $3\log_{1/(1-(\frac{3}{4})^\sigma)} m$ as σ increases. □

3.3 Approximate Weighted Text Matching

In this section we present an algorithm to solve problem ApproxWeighted-TextMatching. The algorithm, denoted by AWTM, is split into two distinct stages: preprocessing the pattern x and searching the weighted text y.

Preprocessing. We build a q-gram index in a similar way as that proposed by Chang and Marr in [8]. Intuitively, we wish to determine the minimum possible Hamming distance (mismatches) between every q-gram on Σ and any q-gram of x. An index like this allows us to lower bound the Hamming distance between a window of y and x without computing the Hamming distance between them. To build this index, we generate every string of length q on Σ, and find the minimum Hamming distance between it and all factors of length q of x. This information can easily be stored by generating a numerical representation of the q-gram and storing the minimum Hamming distance in an array at this location. If we know the numerical representation, we can then look up any entry in constant time. This index has size $\mathcal{O}(\sigma^q)$ and can be trivially constructed in worst-case time $\mathcal{O}(mq\sigma^q)$ and space $\mathcal{O}(\sigma^q)$.

Lemma 4. *Let u be a standard q-gram and v be a uniformly random indeterminate q-gram. The probability that u and v match with cq-mismatches is at most* $\left(\frac{2^{\sigma-1}}{2^\sigma-1}\right)^q (1-c)^{-(1-c)q} c^{-cq} \frac{1}{4\sqrt{c(c-1)}}$, *where* $0 < c < 1/2$.

Proof. Without loss of generality assume that cq is an integer. By Lemma 2, the probability the q-grams match with *exactly* i mismatches is

$$\binom{q}{i}\left(\frac{2^{\sigma-1}}{2^\sigma-1}\right)^q.$$

Therefore, by Lemma 3.8.2 in [15] on the sum of binomial coefficients, the probability that u and v match with cq-mismatches is

$$\sum_{i=0}^{cq}\binom{q}{i}\left(\frac{2^{\sigma-1}}{2^\sigma-1}\right)^q = \left(\frac{2^{\sigma-1}}{2^\sigma-1}\right)^q \sum_{i=0}^{cq}\binom{q}{i} \leq \left(\frac{2^{\sigma-1}}{2^\sigma-1}\right)^q (1-c)^{-(1-c)q} c^{-cq} \frac{1}{4\sqrt{c(c-1)}}.$$

This decreases exponentially in q when $(1-c)^{-(1-c)} c^{-c} > 0$ which holds for $0 < c < 1/2$. It tends to $2^{-q}(1-c)^{-(1-c)q} c^{-cq} \frac{1}{4\sqrt{c(c-1)}}$ as σ increases. □

Searching. We wish to read backwards enough indeterminate q-grams from a window of size m such that the probability that we must verify the window is small and the amount we can shift the window by is sufficiently large. By Lemma 4, we know that the probability of a random indeterminate q-gram occurring in a string of length m with cq-mismatches is no more than $m\big(\frac{2^{\sigma-1}}{2^{\sigma}-1}\big)^q(1-c)^{-(1-c)q}c^{-cq}\frac{1}{4\sqrt{c(1-c)}}$. For the rest of the discussion let $a = 4\sqrt{c(1-c)}$.

In the case when we read $k/(cq)$ indeterminate q-grams, we know that with probability at most $(k/(cq))m\big(\frac{2^{\sigma-1}}{2^{\sigma}-1}\big)^q(1-c)^{-(1-c)q}c^{-cq}1/a$ we have found at most k mismatches. This does not permit us to discard the window if all q-grams occur with at most cq mismatches. To fix this, we instead read $\ell = 1 + k/(cq)$ q-grams. If any indeterminate q-gram occurs with less than cq mismatches, we will need to verify the window; but if they all occur with at least cq mismatches, we must exceed the threshold k and can shift the window. When shifting the window we have the case that we shift after verifying the window and the case that the mismatches exceed k so we do not verify the window. If we have verified the window, we can shift past the last position we checked for an occurrence: we can shift by m positions. If we have not verified the window, as we read a fixed number of indeterminate q-grams, we know the minimum-length shift we can make is one position past this point. The length of this shift is at least $m-(q+k/c)$ positions. This means we will have at most $\frac{n}{m-(q+k/c)} = \mathcal{O}(\frac{n}{m})$ windows. The previous statement is only true assuming $m > q + k/c$, as then the denominator is positive. From there we see that we also have the condition that $q + k/c$ can be at most ϵm, where $\epsilon < 1$, so the denominator will be $\mathcal{O}(m)$. This puts the condition on c, that is, $c > \frac{k}{\epsilon m - q}$. Therefore, for each window, we verify with probability at most $(1+k/(cq))m\Big(\frac{2^{\sigma-1}}{2^{\sigma}-1}\Big)^q(1-c)^{-(1-c)q}c^{-cq}1/a$. So the probability that a verification is triggered is

$$(1+k/(cq))m\Big(\frac{2^{\sigma-1}}{2^{\sigma}-1}\Big)^q(1-c)^{-(1-c)q}c^{-cq}1/a.$$

By Lemma 1 verification takes time $\mathcal{O}(m^2)$; per window the expected cost is

$$\mathcal{O}(m^2)(1+k/(cq))m\Big(\frac{2^{\sigma-1}}{2^{\sigma}-1}\Big)^q(1-c)^{-(1-c)q}c^{-cq}1/a = \mathcal{O}\left(\frac{(q+k)m^3\big(\frac{2^{\sigma-1}}{2^{\sigma}-1}\big)^q(1-c)^{-(1-c)q}c^{-cq}}{qa}\right).$$

We wish to ensure that the probability of verifying a window is small enough that the average work done is no more than the work we must do if we skip a window without verification. When we do not verify a window, we read $\ell = 1 + k/(cq)$ indeterminate q-grams and shift the window. This means that we process $q + k/c = \mathcal{O}(q + k)$ positions. So a sufficient *condition* is the following:

$$\frac{(q+k)m^3\left(\frac{2^{\sigma-1}}{2^{\sigma}-1}\right)^q(1-c)^{-(1-c)q}c^{-cq}}{qa} = \mathcal{O}(q+k).$$

For sufficiently large m, this holds if we set $q \geq \frac{3\log_v m - \log_v a - \log_v q}{1+(1-c)\log_v(1-c)+c\log_v c}$, where $v = \frac{2^{\sigma}-1}{2^{\sigma-1}}$, which tends to $\frac{3\log_2 m - \log_2 a - \log_2 q}{1+(1-c)\log_2(1-c)+c\log_2 c}$ as σ increases.

```
Algorithm AWTM($x, m, y, n, z, k, q, \ell, \Sigma$)
    $\mathsf{D}[0\,..\,|\Sigma|^q - 1] \leftarrow 0$;
    foreach $s \in \Sigma^q$ do
        Compute the minimal Hamming distance $e$ between $s$ and any
        factor of $x$, and set $\mathsf{D}[s] \leftarrow e$;
    $i \leftarrow 0$;
    while $i < n - m + 1$ do
        $d \leftarrow 0$;
        foreach $t \in [1, \ell]$ do
            $j \leftarrow i + m - t \times q$;
            Let $\mathcal{A}$ denote the set of all valid $q$-grams starting at position $j$
            in $y$;
            if $\mathcal{A} = \emptyset$ then
                break;
            else
                $d_{\min} \leftarrow q$;
                foreach $s \in \mathcal{A}$ do
                    $d_{\min} \leftarrow \min\{d_{\min}, \mathsf{D}[s]\}$;
                $d \leftarrow d + d_{\min}$;
        if $d > k$ or $\mathcal{A} = \emptyset$ then
            $i \leftarrow j + 1$;
        else
            if VER3($x, y[i\,..\,i + m - 1], z, k$) = true then
                output $i$;
            $i \leftarrow i + 1$;
```

For this analysis to hold we must be able to read the required number of indeterminate q-grams. Note that the above probability is the probability that at least one of q-grams matches with less than cq mismatches. To ensure we have enough unread random q-grams for the above analysis to hold, the window must be of size $m \geq 2q + 2k/c$. Consider the case where $2q + 2k/c > m \geq 2q + k/c$. If we have verified a window then we have enough new random q-grams, and if we have just shifted, then we know that all the q-grams we previously read matched with at least cq mismatches and we have at least one new q-gram. The probability that one of these matches with less than cq mismatches is less than the one used above so the analysis holds in both cases. Note that this technique can work for any ratio which satisfies $k/m \leq c - \frac{2cq}{m}$.

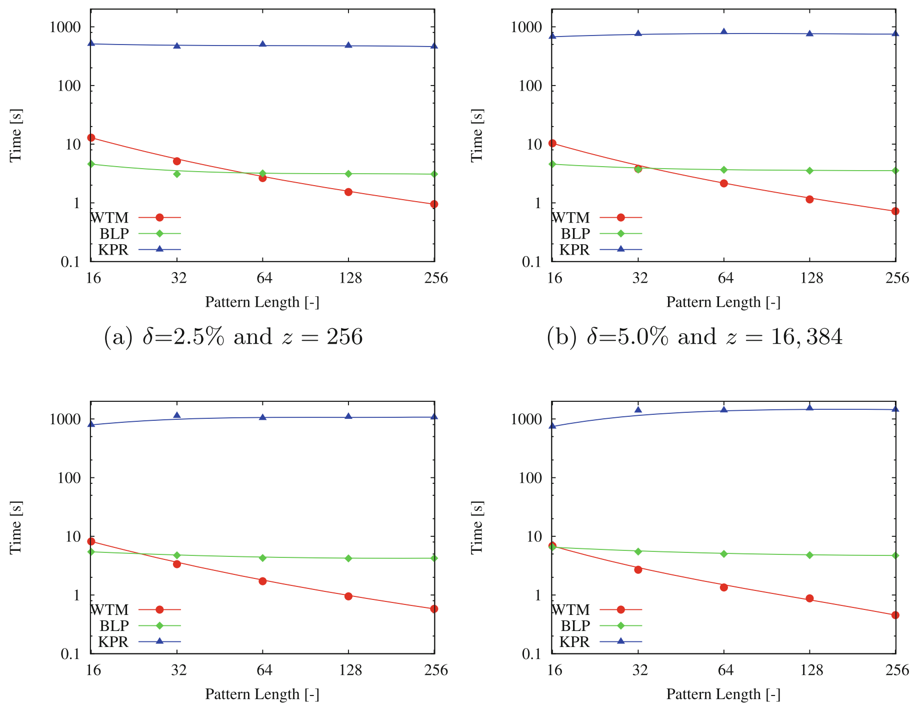
Now recall that in fact we are processing a *weighted* text, not an indeterminate one. By the aforementioned analysis, we can choose a suitable value for c and q, to obtain the following result, noting also that it takes time $\mathcal{O}(z(q+k))$, by Fact 2, to obtain the *valid* q-grams of $\mathcal{O}(q+k)$ positions of the weighted text.

Theorem 5. *Algorithm* AWTM *solves problem* ApproxWeightedText-Matching *in average-case search time* $\mathcal{O}\left(\frac{nz(\log m+k)}{m}\right)$ *if* $k/m \leq c - \frac{2cq}{m}$ *and we set* $q \geq \frac{3\log_v m - \log_v a}{1+(1-c)\log_v(1-c)+c\log_v c}$, *where* $v = \frac{2^\sigma-1}{2^{\sigma-1}}$, $a = 4\sqrt{c(1-c)}$ *and* $0 < c < 1/2$, *which tends to* $\frac{3\log_2 m - \log_2 a}{1+(1-c)\log_2(1-c)+c\log_2 c}$ *as* σ *increases. The worst-case preprocessing time and space is* $\mathcal{O}(mq\sigma^q)$ *and* $\mathcal{O}(\sigma^q)$, *respectively.*

4 Experimental Results

Algorithms WTM and GWPM were implemented as a program to perform exact weighted string matching, and algorithm AWTM was implemented as a program to perform approximate weighted string matching. The programs were implemented in the C++ programming language and developed under the GNU/Linux operating system. The input parameters for WTM and GWPM are a pattern (weighted string for GWPM), a weighted text, and a cumulative weight threshold. The output of this program is the starting positions of all valid occurrences. The input parameters for AWTM are a pattern, a weighted text, a cumulative weight threshold, and an integer k for mismatches. The output of this program is the starting positions of all valid occurrences. These implementations are distributed under the GNU General Public License (GPL). The implementation for WTM and GWPM is available at https://github.com/YagaoLiu/GWSM, and the implementation for AWTM is available at https://github.com/YagaoLiu/HDwtm. The experiments were conducted on a Desktop PC using one core of Intel Core i5-4690 CPU at 3.50 GHz under GNU/Linux. All programs were compiled with g++ version 4.8.4 at optimisation level 3 (-O3).

WTM vs.Stateoftheart. We first compared the time performance of WTM against two other state-of-the-art algorithms: the worst-case $\mathcal{O}(n \log z)$-time algorithm of [14], denoted by KPR; and the average-case $o(n)$-time algorithm of [5], denoted by BLP. For this experiment we used synthetic DNA data generated by a randomised script. The length of the input weighted texts was 32MB. Four texts with four different uncertain positions percentages, denoted by δ, were used: 2.5 %, 5 %, 7.5 %, and 10 %. The length of the input patterns ranged between 16 and 256. The cumulative weight threshold $1/z$ was set according to δ to make sure that all patterns could potentially have valid occurrences. Specifically we set: $z = 256$ when $\delta = 2.5\,\%$; $z = 16,384$ when $\delta = 5\,\%$; $z = 1,048,576$ when $\delta = 7.5\,\%$; and $z = 67,108,864$ when $\delta = 10\,\%$. The length of q-grams was



(a) δ=2.5% and $z = 256$

(b) δ=5.0% and $z = 16,384$

(c) δ=7.5% and $z = 1,048,576$

(d) δ=10% and $z = 67,108,864$

Fig. 1. Elapsed time of KPR, BLP, and WTM for on-line pattern matching using synthetic weighted texts of length 32MB over the DNA alphabet ($\sigma = 4$)

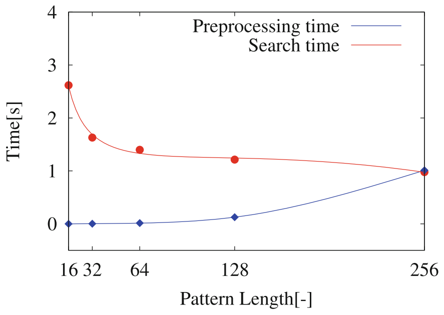
set to $q = 8$. The results plotted in Fig. 1 show that WTM was between one and three orders of magnitude faster than the state-of-the-art approaches with these datasets. In addition, our theoretical findings in Theorem 3 are confirmed: for increasing m and constant n and z, the elapsed time of WTM decreases.

Application I. DnaA proteins are the universal initiators of replication from the chromosomal replication origin (oriC) in bacteria. The DnaA protein recognises and binds specifically to the IUPAC-encoded sequence `TTWTNCACA`, named DnaA box, which is present in all studied bacterial chromosomal replication origins [7]. We transformed sequence `TTWTNCACA` into the weighted sequence `TT[(A,0.5),(T,0.5)]T[(A,0.25),(T,0.25), (G,0.25),(T,0.25)]CACA`; and then we searched for this sequence in a dozen of bacterial genomes obtain from the NCBI genome database. The length of q-gram was set to $q = 4$ and the cumulative weight threshold to $1/z = 1/8$ to ensure all factors of length 9 are valid. We compared the time performance of algorithm GWPM against the worst-case $\mathcal{O}(nz^2 \log z)$-time algorithm of [6], denoted by WPT, for this assignment. The bacteria used, the number of occurrences found, and the elapsed times are shown in Table 1. The results show that GWPM is one order of magnitude faster than WPT.

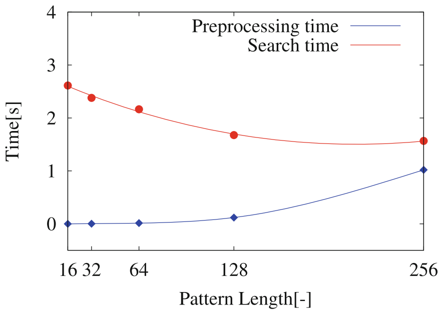
Table 1. Elapsed time of GWPM and WPT for searching for the DnaA box `TTWTNCACA` in 12 bacterial genomes

Bacteria	DNA length	Number of occurrences	Elapsed time (s)	
			GWPM	WPT
Bacillus subtilis	4,215,606	321	1.19	9.83
Escherichia coli	4,641,652	165	1.27	10.45
Haemophilus influenzae	1,830,138	177	0.56	4.83
Helicobacter pylori	1,667,867	172	0.48	3.96
Mycobacterium tuberculosis	4,411,532	21	1.14	9.54
Proteus mirabilis	4,063,606	294	1.16	9.23
Pseudomonas aeruginosa	6,264,404	66	1.60	14.66
Pseudomonas putida	6,181,873	141	1.62	13.55
Salmonella enterica	4,857,432	190	1.31	12.03
Staphylococcus aureus	2,821,361	202	0.80	7.75
Streptomyces lividans	8,345,283	18	2.08	18.42
Yersinia pestis	4,653,728	190	1.29	10.72

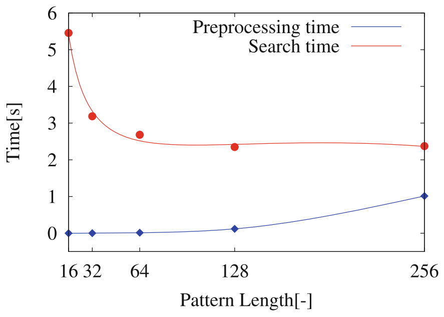
Application II. Mutations in 14 known genes on human chromosome 21 have been identified as the causes of monogenic disorders. These include one form of Alzheimer's disease, amyotrophic lateral sclerosis, autoimmune polyglandular disease, homocystinuria, and progressive myoclonus epilepsy; in addition, a locus for predisposition to leukaemia has been mapped to chromosome 21 [12]. To this end, we evaluated the time performance of AWTM for pattern matching in a genomes population. Pattern matching is useful for evaluating whether SNPs occur in experimentally derived transcription factor binding sites [11,18]. As input text we used the human chromosome 21 augmented with a set of genomic variants obtained from the 1000 Genomes Project. The SNPs present in the population were incorporated to transform the chromosome sequence into a weighted text. The length of human chromosome 21 is $48,129,895$ base pairs. The input patterns were selected randomly from the text; their length ranged between 16 and 256. In this real scenario, δ was found to be $0.7\,\%$; we therefore set the cumulative weight threshold to the constant value of $1/z = 1/1,024$. For a pattern of length m, the maximum allowed number of mismatches k was set to $2.5\,\% \times m$, $5\,\% \times m$, $7.5\,\% \times m$, and $10\,\% \times m$. The length of q-grams was set to $q = \log_2 m$ and the number of q-grams read backwards was set to $\ell = \lfloor \frac{k}{0.2 \times q} + 1 \rfloor$. The exact values for q, k, and ℓ are presented in Table 2. The results plotted in Fig. 2 demonstrate the effectiveness of AWTM: all pattern occurrences can be found within a few seconds, even for error rates of $10\,\%$. In addition, our theoretical findings in Theorem 5 are confirmed: for increasing m and constant n, k, and z, the preprocessing time of AWTM increases but the search time decreases.



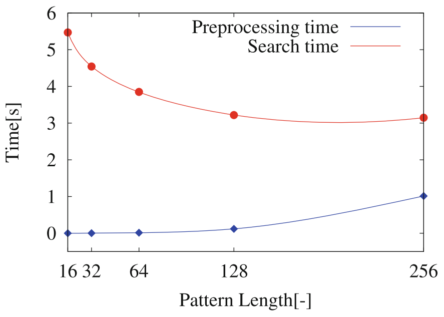
(a) $k = 2.5\% \times m$ and $z = 1,024$



(b) $k = 5.0\% \times m$ and $z = 1,024$



(c) $k = 7.5\% \times m$ and $z = 1,024$



(d) $k = 10\% \times m$ and $z = 1,024$

Fig. 2. Elapsed time of AWTM for on-line approximate pattern matching in human chromosome 21 augmented with SNPs from the 1000 Genomes Project

Table 2. Pattern length m, q-grams length q, number k of maximum allowed mismatches, and number ℓ of q-grams read backwards for different error rates

m	q	2.5 %		5.0 %		7.5 %		10 %	
		k	ℓ	k	ℓ	k	ℓ	k	ℓ
16	4	1	2	1	2	2	3	2	3
32	5	1	2	2	3	3	4	4	5
64	6	2	2	4	4	5	5	7	6
128	7	4	3	7	6	10	8	13	10
256	8	7	5	13	9	20	13	26	17

5 Final Remarks

In this article, we provided efficient on-line average-case algorithms for solving various weighted pattern matching problems. We also provided extensive experimental results, using both real and synthetic data, showing that our implementations outperform state-of-the-art approaches by more than one order of magnitude. Furthermore, we demonstrated the suitability of the proposed algorithms in a number of real biological contexts. We would like to stress though that the applicability of these algorithms is not exclusive to molecular biology.

Our immediate target is to investigate other ways to measure approximation in weighted pattern matching. Another direction is to study pattern matching on the following generalised notion of weighted strings. A *generalised weighted string* x of length n on an alphabet Σ is a finite sequence of n sets. Every $x[i]$, for all $0 \leq i < n$, is a set of ordered pairs $(s_j, \pi_i(s_j))$, where s_j is a possibly empty string on Σ and $\pi_i(s_j)$ is the probability of having string s_j at position i.

References

1. Amir, A., Chencinski, E., Iliopoulos, C.S., Kopelowitz, T., Zhang, H.: Property matching and weighted matching. Theor. Comput. Sci. **395**(2–3), 298–310 (2008)
2. Amir, A., Iliopoulos, C., Kapah, O., Porat, E.: Approximate matching in weighted sequences. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 365–376. Springer, Heidelberg (2006). doi:10.1007/11780441_33
3. Barton, C., Iliopoulos, C.S., Pissis, S.P.: Optimal computation of all tandem repeats in a weighted sequence. Algorithms Mol. Biol. **9**(21), 1–12 (2014)
4. Barton, C., Kociumaka, T., Pissis, S.P., Radoszewski, J.: Efficient index for weighted sequences. In: CPM 2016, LIPIcs, vol. 54, pp. 4: 1–4: 13. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
5. Barton, C., Liu, C., Pissis, S.P.: Fast average-case pattern matching on weighted sequences. CoRR abs/1512.01085 (2015). (submitted to IPL)
6. Barton, C., Pissis, S.P.: Linear-time computation of prefix table for weighted strings. In: Manea, F., Nowotka, D. (eds.) WORDS 2015. LNCS, vol. 9304, pp. 73–84. Springer, Heidelberg (2015). doi:10.1007/978-3-319-23660-5_7
7. Caspi, R., Helinski, D.R., Pacek, M., Konieczny, I.: Interactions of DnaA proteins from distantly related bacteria with the replication origin of the broad host range plasmid RK2. J. Biol. Chem. **275**(24), 18454–18461 (2000)
8. Chang, W.I., Marr, T.G.: Approximate string matching and local similarity. In: Crochemore, M., Gusfield, D. (eds.) CPM 1994. LNCS, vol. 807, pp. 259–273. Springer, Heidelberg (1994). doi:10.1007/3-540-58094-8_23
9. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, New York (2007)
10. Farach, M.: Optimal suffix tree construction with large alphabets. In: FOCS 1997, pp. 137–143. IEEE Computer Society (1997)
11. Guo, Y., Jamison, D.C.: The distribution of SNPs in human gene regulatory regions. BMC Genom. **6**(1), 1–11 (2005)
12. Hattori, M., et al.: The DNA sequence of human chromosome 21. Nature **405**, 311–319 (2000)
13. Huang, L., Popic, V., Batzoglou, S.: Short read alignment with populations of genomes. Bioinformatics **29**(13), 361–370 (2013)
14. Kociumaka, T., Pissis, S.P., Radoszewski, J.: Pattern matching and consensus problems on weighted sequences and profiles. In: ISAAC 2016, LIPIcs. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
15. Lovász, L., Pelikán, J., Vesztergombi, K.: Discrete Mathematics: Elementary and Beyond. Springer, New York (2003)

16. Musser, D.R.: Introspective sorting and selection algorithms. Softw. Pract. Exp. **27**(8), 983–993 (1997)
17. Pizzi, C., Ukkonen, E.: Fast profile matching algorithms - a survey. Theor. Comput. Sci. **395**(2–3), 137–157 (2008)
18. Sandelin, A., Alkema, W., Engström, P., Wasserman, W.W., Lenhard, B.: JASPAR: an open-access database for eukaryotic transcription factor binding profiles. Nucleic Acids Res. **32**(1), D91–D94 (2004)
19. Varela, M.A., Amos, W.: Heterogeneous distribution of SNPs in the human genome: microsatellites as predictors of nucleotide diversity and divergence. Genomics **95**(3), 151–159 (2010)