On–line Approximate String Matching with Bounded Errors

Marcos Kiwi[1⋆], Gonzalo Navarro[2⋆⋆], and Claudio Telha[3⋆⋆⋆]

[1] Departamento de Ingeniería Matemática, Centro de Modelamiento Matemático UMI 2807 CNRS-UChile. www.dim.uchile.cl/∼mkiwi.
[2] Department of Computer Science, University of Chile. gnavarro@dcc.uchile.cl.
[3] Operations Research Center, MIT. ctelha@mit.edu.

Abstract. We introduce a new dimension to the widely studied on-line approximate string matching problem, by introducing an *error threshold* parameter ϵ so that the algorithm is allowed to miss occurrences with probability ϵ. This is particularly appropriate for this problem, as approximate searching is used to model many cases where exact answers are not mandatory. We show that the relaxed version of the problem allows us breaking the average-case optimal lower bound of the classical problem, achieving average case $O(n \log_\sigma m/m)$ time with any $\epsilon = \text{poly}(k/m)$, where n is the text size, m the pattern length, k the number of errors for edit distance, and σ the alphabet size. Our experimental results show the practicality of this novel and promising research direction.

1 Introduction

In string matching one is interested in determining the positions (sometimes just deciding the occurrence) of a given pattern P on a text T, where both pattern and text are strings over some fixed finite alphabet Σ of size σ. The lengths of P and T are typically denoted by m and n respectively. In approximate string matching there is also a notion of distance between strings, given say by $d : \Sigma^* \times \Sigma^* \to \mathbf{R}$. One is given an additional non-negative input parameter k and is interested in listing all positions (or just deciding the occurrence) of substrings S of T such that S and P are at distance at most k. In the "on-line" or "sequential" version of the problem, one is not allowed to preprocess the text.

Since the 60's several approaches were proposed for addressing the approximate matching problem, see for example the survey by Navarro [5]. Most of the work focused on the *edit* or *Levenshtein* distance d, which counts the number of character insertions, deletions, and substitutions needed to make two strings equal. This distance turns out to be sufficiently powerful to model many relevant applications (e.g., text searching, information retrieval, computational biology,

⋆ Gratefully acknowledges the support of CONICYT via FONDAP in Applied Mathematics and Anillo en Redes ACT08.
⋆⋆ Funded in part by Fondecyt Grant 1-050493, Chile.
⋆⋆⋆ Gratefully acknowledges the support of CONICYT via Anillo en Redes ACT08 and Yahoo! Research Grant "Compact Data Structures".

transmission over noisy channels, etc.), and at the same time sufficiently simple to admit efficient solutions (e.g., $O(mn)$ and even $O(kn)$ time).

A lower bound to the (worst-case) problem complexity is obviously $\Omega(n)$ for the meaningful cases, $k < m$. This bound can be reached by using automata, which introduce an extra additive term in the time complexity which is exponential in m or k. If one is restricted to polynomially-bounded time complexities on m and k, however, the worst-case problem complexity is unknown.

Interestingly, the average-case complexity of the problem is well understood. If the characters in P and T are chosen uniformly and independently, the average problem complexity is $\Theta(n(k + \log_\sigma m)/m)$. This was proved in 1994 by Chang and Marr [2], who gave an algorithm reaching the lower bound for $k/m < 1/3 - O(\sigma^{-1/2})$. In 2004, Fredriksson and Navarro [3] gave an improved algorithm achieving the lower bound for $k/m < 1/2 - O(\sigma^{-1/2})$. In addition to covering the range of interesting k values for virtually all applications, the algorithm was shown to be highly practical.

It would seem that, except for determining the worst-case problem complexity (which is mainly of theoretical interest), the on-line approximate string matching problem is closed. In this paper, however, we reopen the problem under a relaxed scenario that is still useful for most applications and admits solutions that beat the lower bound. More precisely, we relax the goal of listing *all* positions where pattern P occurs in the text T to that of listing each such position with probability $1 - \epsilon$, where ϵ is a new input parameter.

There are several relevant scenarios where fast algorithms that make errors (with a user-controlled probability) are appropriate. Obvious cases are those where approximate string matching is used to increase recall when searching data that is intrinsically error-prone. Consider for example an optical character recognition application, where errors will inevitably arise from inaccurate scanning or printing imperfections, or a handwriting recognition application, or a search on a text with typos and misspells. In those cases, there is no hope to find exactly all the correct occurrences of a word. Here, uncertainty of the input translates into approximate pattern matching and approximate searching is used to increase the chance of finding relevant occurrences, hopefully without introducing too many false matches. As the output of the system, even using a correct approximate string matching technique, is an approximation to the ideal answer, a second approximation might be perfectly tolerable, and even welcome if allows for faster searches.

A less obvious application arises in contexts where we might have a priori knowledge that some pattern is either approximately present in the text many times, or does not even approximately occur. Some examples are genetic markers that might often appear or not at all, some modisms that might appear in several forms in certain type of texts, some typical pattern variants that might appear in the denomination of certain drugs, people names, or places, of which typically several instances occur in the same text. Further, we might only be interested in determining whether the pattern occurs or not. (A feature actually available in the well known `grep` string searching utility as options `-l` and `-L`, and also

the approximate string searching utilities `agrep` and `ngrep`.) In this context, a text with N approximate pattern occurrences will be misclassified by the inexact algorithm with very low probability, ϵ^N.

Another interesting scenario is that of processing data streams which flow so fast that there is no hope for scanning them exhaustively (e.g. radar derived meteorological data, browser clicks, user queries, IP traffic logs, peer-to-peer downloads, financial data, etc.). Hence even an exact approximate search over part of the data would give only partial results. A faster inexact algorithm could even give better quality answers as it could scan a larger portion of the data, even if making mistakes on it with controlled probability.

The new framework proposed in this work comes from the so-called testing and property testing literature where the aim is to devise sublinear time algorithms obtained by avoiding having to read all of the input of a problem instance. These procedures typically read a very small fraction of the input. For most natural problems the algorithm must use randomization and provide answers which in some sense are approximate, or wrong with some probability. See the many surveys on the topic, e.g. [7, 8].

1.1 Main Contributions

We focus in particular on the so-called filtering algorithms [5, § 8]. These algorithms quickly discard areas of the text that cannot approximately match the pattern, and then verify the remaining areas with a classical algorithm. In practice, filtering algorithms are also the fastest approximate string matching algorithms. They also turn out to be natural candidates to design probabilistic variants in this paper.

In Section 3.1 we describe a procedure based on sampling q-grams motivated by the filtering algorithm of Ukkonen [9]. For a fixed constant $t > 0$ and $k < m/\log_\sigma m$, the derived algorithm has an average case complexity of $O(tn \log_\sigma m/m)$ and misses pattern occurrences with probability $\epsilon = O((k \log_\sigma m/m)^t)$. Note that the time equals Yao's lower bound for *exact* string matching ($k = 0$). In contrast, Ukkonen's original algorithm takes $O(n)$ time. In Section 3.2 we describe an algorithm based on Chang and Marr's [2] average-optimal algorithm. For fixed $t > 0$, we derive an $O(tn \log_\sigma m/m)$ average-time approximate matching algorithm with error $\epsilon = O((k/m)^t)$. Note that the latter achieves the same time complexity for a smaller error, and that it works for any $k < m$, whereas the former needs $k < m/\log_\sigma m$.

The discrepancy between both algorithms inherits from that of the original classical algorithms they derive from, where the original differences in time complexities has now translated into their error probabilities. It is important to stress that both algorithms beat the average-complexity lower bound of the problem when errors are not allowed, $\Omega(n(k + \log_\sigma m)/m)$, as they remove the $\Omega(kn/m)$ term in the complexity (the k/m term now shows up in the error probability).

The aforementioned average case complexity results are for random text, but hold even for fixed patterns. Our analyzes focus exclusively on Levenshtein distance d, but should be easily adapted to other metrics.

In Section 4 we present some experimental results that corroborate the theoretical results of Section 3 and give supporting evidence for the practicality of our proposals. In particular, the experiments favor the technique of Section 3.1 over that of Section 3.2, despite the theoretical superiority of the latter.

2 Model for Approximate Searching allowing Errors

In this section we formalize the main concepts concerning the notion of approximate matching algorithms with errors. We adopt the standard convention of denoting the substring $S_i \dots S_j$ of $S = S_1 \dots S_n$ by $S_{i..j}$ and refer to the number of characters of S by the *length of* S which we also denote by $|S|$. We start by recalling the formal definition of the approximate string matching problem when the underlying distance function is d. Henceforth, we abbreviate d-Approximate String Matching as d-ASM.

Problem d-Approximate String Matching
Input Text $T \in \Sigma^*$, pattern $P \in \Sigma^*$ and parameter $k \in \mathbf{N}$.
Output $S = S(T, P, k) \subseteq \{1, \dots, n\}$ such that $j \in S$ if and only if there is an i such that $d(T_{i..j}, P) \leq k$.

When the text T and pattern P are both in Σ^*, and the parameter k is in $\mathbf{N}$ we say that (T, P, k) is *an instance of the* d-ASM *problem*, or simply *an instance* for short. We henceforth refer to $S(T, P, k)$ as the *solution set* of instance (T, P, k). We say that algorithm $\mathcal{A}$ solves the d-ASM problem if on instance (T, P, k) it outputs the solution set $S(T, P, k)$. Note that $\mathcal{A}$ might be a probabilistic algorithm, however its output is fully determined by (T, P, k).

For a randomized algorithm $\mathcal{A}$ that takes as input an instance (T, P, k), let $\mathcal{A}(T, P, k)$ be the distribution over sets $S \subseteq \{1, \dots, n\}$ that it returns.

Henceforth we denote by $X \leftarrow \mathcal{D}$ the fact that the random variable X is chosen according to distribution $\mathcal{D}$. For a set C we denote the probability that $X \in C$ when X is chosen according to the distribution $\mathcal{D}$ by $\mathbf{Pr}[X \in C; X \leftarrow \mathcal{D}]$ or $\mathbf{Pr}_{X \leftarrow \mathcal{D}}[X \in C]$. Also, we might simply write $\mathbf{Pr}_X[X \in C]$ or $\mathbf{Pr}[X \in C]$ when it is clear from context that $X \leftarrow \mathcal{D}$. The notation generalizes in the obvious way to the case where X is a random vector, and/or when instead of a probability one is interested in taking expectation.

We say that randomized algorithm $\mathcal{A}$ *solves the* d-ASM *problem with* (ϵ, ϵ')-*error* provided that on any instance (T, P, k) the following holds:

Completeness: if $i \in S(T, P, k)$, then $\mathbf{Pr}[i \in S'; S' \leftarrow \mathcal{A}(T, P, k)] \geq 1 - \epsilon$,
Soundness: if $i \notin S(T, P, k)$, then $\mathbf{Pr}[i \in S'; S' \leftarrow \mathcal{A}(T, P, k)] \leq \epsilon'$,

where the two probabilities above are taken only over the source of randomness of $\mathcal{A}$.

When $\epsilon' = 0$ we say that $\mathcal{A}$ has *one–sided* ϵ-*error* or that it *is one–sided* for short. When $\epsilon = \epsilon' = 0$ we say that $\mathcal{A}$ is an *errorless* or *exact* algorithm.

We say that randomized algorithm $\mathcal{F}$ is a d-ASM *probabilistic filter with α-error* or simply *is an α-filter* for short, provided that on any instance (W, P, k) the following holds: if $d(P_{i..j}, W) \leq k$ for some pattern substring $P_{i..j}$, then $\mathbf{Pr}\left[\mathcal{F}(W, P, k) = \texttt{Check}\right] \geq 1 - \alpha$, where the probability is taken over the source of randomness of $\mathcal{F}$. If a filter does not return `Check` we assume without loss of generality that it returns `Discard`.

The notion of an α-filter is crucial to the ensuing discussion. Roughly said, a filter $\mathcal{F}$ will allow us to process a text T by considering non-overlapping consecutive substrings W of T, running the filter on instance (W, P, k) and either: (1) in case the filter returns `Check`, perform a costly approximate string matching procedure to determine whether P approximately occurs in T in the surroundings of window W, or (2) in case the filter does not return `Check`, discard the current window from further consideration and move forward in the text and process the next text window. The previously outlined general mechanism is the basis of the generic algorithm we illustrate in Fig. 1 and describe below. The

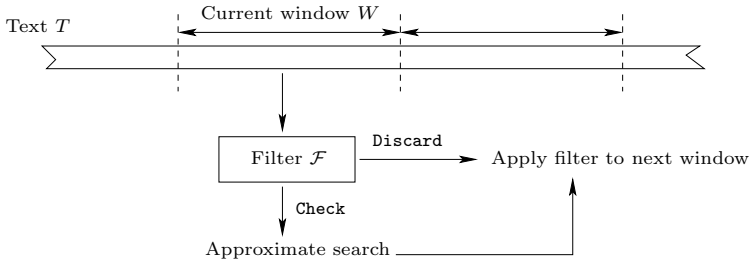

Fig. 1. Generic algorithm d-Approximate String Matching algorithm.

attentive reader would have noticed that when defining probabilistic filters we substituted the notation T for texts by W. This is done in order to stress that the probabilistic filters that we will talk about access the text T by sequentially examining substrings of T which we will refer to as *windows*. These windows will typically have a length which is independent of n, more precisely they will be of length $O(m)$.

We now precisely describe the central role played by probabilistic filters in the design of d-ASM algorithms with errors. First, from now on, let w denote $\lfloor (m - k)/2 \rfloor$. Henceforth let $W_1, \ldots, W_s$ be such that $T = W_1 \ldots W_s$ and $|W_p| = w$ (pad T with an additional character not in Σ as necessary). Note that $s = \lceil n/w \rceil$ and $W_p = T_{(p-1)w+1..pw}$. Given any probabilistic filter $\mathcal{F}$ and an exact algorithm $\mathcal{E}$ we can devise a generic d-ASM algorithm with errors such as the one specified in Algorithm 1.[4]

We will shortly show that the generic algorithm $\mathcal{G}$ is correct. We also would like to analyze its complexity in terms of the efficiencies of both the probabilistic filter $\mathcal{F}$ and the exact algorithm $\mathcal{E}$. However, we first need to introduce the complexity measures that we will be looking at. Let $\mathbf{Time}_{\mathcal{A}}(T, P, k) \in \mathbf{N} \cup$

[4] For $A \subseteq \mathbf{Z}$ we use the standard convention of denoting $\{a + x : x \in A\}$ by $a + A$.

Algorithm 1 Generic d-Approximate String Matching with Errors

1: **procedure** $\mathcal{G}(T, P, k)$ ▷ $T \in \Sigma^n$, $P \in \Sigma^m$, $k \in \mathbf{N}$
2: $S \leftarrow \emptyset$
3: $w \leftarrow \lfloor (m-k)/2 \rfloor$
4: $s \leftarrow \lceil n/w \rceil$
5: **for** $p \in \{1, \ldots, s\}$ **do**
6: **if** $\mathcal{F}(W_p, P, k) = \texttt{Check}$ **then** ▷ Where $W_p = T_{(p-1)w+1..pw}$
7: $S \leftarrow S \cup \big((pw - m - k + 1) + \mathcal{E}(T_{pw-m-k+1..(p-1)w+m+k-1}, P, k)\big)$
8: **return** S

$\{+\infty\}$ be the expected time complexity of $\mathcal{A}$ on the instance (T, P, k), where the expectation is taken over the random choices of $\mathcal{A}$. We also associate to $\mathcal{A}$ the following average time complexity measures:

$$\mathbf{Avg}_{\mathcal{A}}(n, P, k) = \mathbf{Ex}_T \left[\mathbf{Time}_{\mathcal{A}}(T, P, k)\right],$$
$$\mathbf{Avg}_{\mathcal{A}}(n, m, k) = \mathbf{Ex}_{T,P} \left[\mathbf{Time}_{\mathcal{A}}(T, P, k)\right].$$

Let $\mathbf{Mem}_{\mathcal{A}}(T, P, k) \in \mathbf{N} \cup \{+\infty\}$ be the maximum amount of memory required by $\mathcal{A}$ on instance (T, P, k), where the maximum is taken over all possible sequences of random bits on which $\mathcal{A}$ may act, and let

$$\mathbf{Mem}_{\mathcal{A}}(n, P, k) = \max_{T \in \Sigma^n} \mathbf{Mem}_{\mathcal{A}}(T, P, k),$$
$$\mathbf{Mem}_{\mathcal{A}}(n, m, k) = \max_{T \in \Sigma^n, P \in \Sigma^m} \mathbf{Mem}_{\mathcal{A}}(T, P, k).$$

We similarly define $\mathbf{Rnd}_{\mathcal{A}}(T, P, k)$, $\mathbf{Rnd}_{\mathcal{A}}(n, P, k)$, and $\mathbf{Rnd}_{\mathcal{A}}(n, m, k)$, but with respect to the maximum number of random bits used by $\mathcal{A}$. Also, the same complexity measures can be defined for probabilistic filters and exact algorithms.

Theorem 1. *Suppose $m > k$. Let $\mathcal{F}$ be an α-filter and let $\mathcal{E}$ be the standard deterministic $O(kn)$ dynamic programming algorithm for the d-*ASM *problem. Let $w = \lfloor (m-k)/2 \rfloor$, $s = \lceil n/w \rceil$, and $\mathcal{W} \subseteq \Sigma^w$. Then, the generic algorithm $\mathcal{G}$ is a d-*ASM *algorithm with one-sided α-error such that*

$$\mathbf{Avg}_{\mathcal{G}}(n, P, k) \leq s \cdot \mathbf{Avg}_{\mathcal{F}}(w, P, k)$$
$$+ s \cdot O(mk) \cdot (\mathbf{Pr}_{W \leftarrow \Sigma^w}[W \in \mathcal{W}] + \max_{W \notin \mathcal{W}} \mathbf{Pr}[\mathcal{F}(W, P, k) = \texttt{Check}]) + O(s).$$

Also, $\mathbf{Mem}_{\mathcal{G}}(n, P, k) = \mathbf{Mem}_{\mathcal{E}}(3w + 4k + 2, P, k)$ *(ignoring the space required to output the result), and* $\mathbf{Rnd}_{\mathcal{G}}(n, P, k) = O(\frac{n}{m-k}) \cdot \mathbf{Rnd}_{\mathcal{F}}(w, P, k)$.

Proof. First, let us establish completeness of $\mathcal{G}$. Assume $i \in S(T, P, k)$. Let $p+1$ be the index of the window to which the character T_i belongs. As any occurrence has length at least $m-k$, W_p is completely contained in the occurrence finishing at i, and thus W_p must be at distance at most k of a substring of P. It follows that $\mathcal{F}(W_p, P, k) = \texttt{Check}$ with probability at least $1 - \alpha$, in which case line 7 of the algorithm will run an exact verification with $\mathcal{E}$ over a text area comprising any substring of length $m + k$ that contains W_p. Since $m + k$ is the maximum

length of an occurrence, it follows that i will be included in the output returned by $\mathcal{G}$. Hence, with probability at least $1-\alpha$ we have that i is in the output of $\mathcal{G}$.

To establish soundness, assume $i \notin S(T,P,k)$. In this case, i will never be included in the output of $\mathcal{G}$ in line 7 of the algorithm.

We now determine $\mathcal{G}$'s complexity. By linearity of expectation and since $\mathbf{Time}_{\mathcal{E}}(O(m),m,k) = O(mk)$, we have

$$\begin{aligned}
&\mathbf{Avg}_{\mathcal{G}}(n,P,k) = \\
&\sum_{p=1}^{s} \left(\mathbf{Ex}_T\left[\mathbf{Time}_{\mathcal{F}}(W_p,P,k)\right] + O(mk)\cdot \mathbf{Pr}_T\left[\mathcal{F}(W_p,P,k) = \texttt{Check}\right] + O(1)\right) \\
&= s\cdot \mathbf{Avg}_{\mathcal{F}}(w,P,k) + O(mk)\cdot \sum_{p=1}^{s} \mathbf{Pr}_T\left[\mathcal{F}(W_p,P,k) = \texttt{Check}\right] + O(s)\,.
\end{aligned}$$

Conditioning according to whether W_p belongs to $\mathcal{W}$, we get for any $\mathcal{W}$ that

$$\mathbf{Pr}_T\left[\mathcal{F}(W_p,P,k) = \texttt{Check}\right] \leq \mathbf{Pr}_{W\leftarrow\Sigma^w}\left[W\in\mathcal{W}\right] + \max_{W\notin\mathcal{W}} \mathbf{Pr}\left[\mathcal{F}(W,P,k) = \texttt{Check}\right].$$

The stated bound on $\mathbf{Avg}_{\mathcal{G}}(n,P,k)$ follows immediately. The memory and randomized complexity bounds are obvious. □

The intuition behind the preceding theorem is that, given any class $\mathcal{W}$ of "interesting" windows, if we have a filter that discards the uninteresting windows with high probability, then the probability that the algorithm has to verify a given text window can be bounded by the sum of two probabilities: (i) that of the window being interesting, (ii) the maximum probability that the filter fails to discard a noninteresting window. As such, the theorem gives a general framework to analyze probabilistic filtration algorithms. An immediate consequence of the result is the following:

Corollary 1. *Under the same conditions as in Theorem 1, if in addition*

$$\mathbf{Pr}_{W\leftarrow\Sigma^w}\left[W\in\mathcal{W}\right] = \max_{W\notin\mathcal{W}} \mathbf{Pr}\left[\mathcal{F}(W,P,k) = \texttt{Check}\right] = O\left(1/m^2\right),$$

then $\mathbf{Avg}_{\mathcal{G}}(n,P,k) = O(s\cdot\mathbf{Avg}_{\mathcal{F}}(w,P,k))$. *This also holds if* $\mathcal{E}$ *is the classical* $O(m^2)$ *time algorithm.*

The previous results suggests an obvious strategy for the design of d-ASM algorithms with errors. Indeed, it suffices to identify a small subset of windows $\mathcal{W}\subseteq\Sigma^w$ that contain all windows of length w that are at distance at most k of a pattern substring, and then design a filter $\mathcal{F}$ such that: (1) the probability that $\mathcal{F}(W,P,k) = \texttt{Check}$ is high when $W\in\mathcal{W}$ (in order not to miss pattern occurrences), and (2) the probability that $\mathcal{F}(W,P,k) = \texttt{Check}$ is low when $W\notin\mathcal{W}$ (in order to avoid running an expensive procedure over regions of the text where there are no pattern occurrences).

The next result is a simple observation whose proof we omit since it follows by standard methods (running $\mathcal{A}$ repeatedly).

Proposition 1. *Let $\mathcal{A}$ be a randomized algorithm that solves the d-*ASM *problem with (ϵ, ϵ')-error.*

- *Let $\alpha \leq \epsilon = \epsilon' < 1/2$ and $N = O(\log(1/\alpha)/(1-2\epsilon)^2)$. Then, there is a randomized algorithm $\mathcal{A}'$ that solves the d-*ASM *problem with (α, α)-error such that $\mathbf{Avg}_{\mathcal{A}'}(n, P, k) = N \cdot \mathbf{Avg}_{\mathcal{A}}(n, P, k)$, $\mathbf{Mem}_{\mathcal{A}'}(n, P, k) = \mathbf{Mem}_{\mathcal{A}}(n, P, k) + O(\log N)$, and where $\mathbf{Rnd}_{\mathcal{A}'}(n, P, k) = N \cdot \mathbf{Rnd}_{\mathcal{A}}(n, P, k)$.*
- *If $\mathcal{A}$ is one-sided, then there is a randomized algorithm $\mathcal{A}'$ solving the d-*ASM *problem with $(\epsilon^N, 0)$-error such that $\mathbf{Avg}_{\mathcal{A}'}(n, P, k) = N \cdot \mathbf{Avg}_{\mathcal{A}}(n, P, k)$, $\mathbf{Mem}_{\mathcal{A}'}(n, P, k) = \mathbf{Mem}_{\mathcal{A}}(n, P, k) + O(\log N)$, and $\mathbf{Rnd}_{\mathcal{A}'}(n, P, k) = N \cdot \mathbf{Rnd}_{\mathcal{A}}(n, P, k)$.*

3 Algorithms for Approximate Searching with Errors

In this section we derive two probabilistic filters inspired on existing (errorless) filtration algorithms. Note that, according to the previous section, we focus on the design of the window filters, and the rest follows from the general framework.

3.1 Algorithm based on q-gram sampling

A q-gram is a substring of length q. Thus, a pattern of length m has $(m-q+1)$ overlapping q-grams. Each error can alter at most q of the q-grams of the pattern, and therefore $(m-q+1-kq)$ pattern q-grams must appear in any approximate occurrence of the pattern in the text. Ukkonen's idea [9] is to sequentially scan the text while keeping count of the last q-grams seen. The counting is done using a suffix tree of P and keeping the relevant information attached to the $m-q+1$ important nodes at depth q in the suffix tree. The key intuition behind the algorithms design is that in random text it is difficult to find substrings of the pattern of length $q > \log_\sigma m$. The opposite is true in zones of the text where the pattern approximately occurs. Hence, by keeping count of the last q-grams seen one may quickly filter out many bad pattern alignments.

We now show how to adapt the ideas mentioned so far in order to design a probabilistic filter. The filtering procedure randomly chooses several indices $i \in \{1, \ldots, |W| - q + 1\}$ and checks whether the q-gram $W_{i..i+q-1}$ is a pattern substring. Depending on the number of q-grams that are present in the pattern the filter decides whether or not to discard the window. See Algorithm 2 for a formal description of the derived probabilistic filter **Q-PE-**$\mathcal{F}_{c,\rho,q}$, where c and ρ are parameters to be tuned later. Using the filter as a subroutine for the generic algorithm with errors described in Algorithm 1 gives rise to a procedure to which we will henceforth refer to as **Q-PE**.

Let $\mathcal{W}$ be the collection of all windows in Σ^w for which at least β of its q-grams are substrings of the pattern. Let $w' = w - q + 1$ be the number of q-grams (counting repetitions) in a window of length w. Finally, let p denote the probability that a randomly chosen q-gram is a substring of the pattern P, i.e.

$$p = \frac{1}{\sigma^q} \cdot |\{P_{i..i+q-1} : i = 1, \ldots, m-q+1\}| \ .$$

Algorithm 2 Probabilistic filter based on q-grams

1: **procedure Q-PE-**$\mathcal{F}_{c,\rho,q}(W,P,k)$ ▷ $W \in \Sigma^w, P \in \Sigma^m$, $k \in \mathbf{N}$
2: $ctr \leftarrow 0$
3: **for** $i \in \{1,\ldots,c\}$ **do**
4: Choose j_i uniformly at random in $\{1,\ldots,|W|-q+1\}$
5: **if** $W_{j_i..j_i+q-1}$ is a substring of P **then** $ctr \leftarrow ctr+1$
6: **if** $ctr > \rho \cdot c$ **then return** `Check` **else return** `Discard`

The following result shows that a window chosen randomly in Σ^w is unlikely to be in $\mathcal{W}$.

Lemma 1. *Let $\beta \geq pw'$. Then,* $\mathbf{Pr}_{W \leftarrow \Sigma^w}[W \in \mathcal{W}] \leq \exp\left(-\frac{24(\beta-pw')^2}{25q(\beta+2pw')}\right)$.

Proof. For $i = 1,\ldots,w'$ let Y_i be the indicator variable of the event "$W_{i..i+q-1}$ is a substring of P" when W is randomly chosen in Σ^w. Clearly, $\mathbf{Ex}[Y_i] = p$. Moreover, $W \in \mathcal{W}$ if and only if $\sum_{i=1}^{w'} Y_i \geq \beta$. Unfortunately, a standard Chernoff type bound cannot be directly applied given that the Y_i's are not independent. Nevertheless, the collection $\{Y_1,\ldots,Y_{w'}\}$ can be partitioned into q families according to i mod q, each one an independent family of variables. The desired result follows applying a Chernoff type bound for so called q-independent families [4, Corollary 2.4]. □

Lemma 2. *If $W \notin \mathcal{W}$, then*

$$\mathbf{Pr}\left[\textbf{Q-PE-}\mathcal{F}_{\rho,c,q}(W,P,k) = \texttt{Check}\right] \leq \exp\left(\rho c - \frac{c\beta}{w'}\right)\left(\frac{\beta}{\rho w'}\right)^{\rho c}.$$

Proof. Let X_{j_i} denote the indicator of whether $W_{j_i..j_i+1-1}$ turns out to be a substring of the pattern P in line 5 of the description of **Q-PE-**$\mathcal{F}_{\rho,c,q}$. Note that the X_{j_i}'s are independent, each with expectation at most β/w' when $W \notin \mathcal{W}$. The claim follows by a standard Chernoff type bound from the fact that:

$$\mathbf{Pr}_{W \leftarrow \Sigma^w}\left[\textbf{Q-PE-}\mathcal{F}_{\rho,c,q}(W,P,k) = \texttt{Check}\right] = \mathbf{Pr}\left[\sum_{i=1}^{c} X_{j_i} \geq \rho \cdot c\right],$$

where the probabilities are taken exclusively over the sequence of random bits of the probabilistic filter. □

Lemma 3. *If $kq \leq w'(1-\rho)$, then* **Q-PE-**$\mathcal{F}_{\rho,c,q}$ *is an α-filter for*

$$\alpha \leq \exp\left((1-\rho)c - \frac{ckq}{w'}\right)\left(\frac{kq}{w'(1-\rho)}\right)^{c(1-\rho)}.$$

Proof. Let $W \in \Sigma^w$. Assume $d(P_{i..j}, W) \leq k$ for some pattern substring $P_{i..j}$. Then, at least $w' - kq$ of W's q-grams are substrings of P. Defining X_{j_i} as in Lemma 2 we still have that the X_{j_i}'s are independent but now their expectation

is at least $1 - kq/w'$. The claim follows by a standard Chernoff type bound from the fact that:

$$\mathbf{Pr}\left[\textbf{Q-PE-}\mathcal{F}_{\rho,c,q}(W,P,k) = \texttt{Discard}\right] = \mathbf{Pr}\left[\sum_{i=1}^{c} X_{j_i} \leq \rho \cdot c\right],$$

where the probabilities are taken exclusively over the sequence of random bits of the probabilistic filter. □

Theorem 2. *If $k < (m - 2\log_\sigma m)/(1 + 4\log_\sigma m)$, then* **Q-PE** *is a d-*ASM *algorithm with one-sided error $\epsilon = O((k\log_\sigma m/m)^t)$ for any constant $t > 0$, running in average time $\mathbf{Avg}_{\textbf{Q-PE}}(n,P,k) = O(tn\log_\sigma m/m)$.*

Proof. The result follows from Theorem 1 and Corollary 1.

Choose $q = 2\lceil\log_\sigma m\rceil$, so $p \leq m/\sigma^q \leq 1/m$. Taking $\beta = \Theta(\log^2 m)$ where the hidden constant is sufficiently large, we have by Lemma 1 that $\mathbf{Pr}_{W\leftarrow\mathcal{W}}[W \in \mathcal{W}] = O(1/m^2)$. By Lemma 2 and taking $\rho = 1/2$ and c a sufficiently large constant, we get that $\mathbf{Pr}\left[\textbf{Q-PE-}\mathcal{F}_{\rho,c,q}(W,P,k) = \texttt{Check}\right] = O(1/m^2)$ when $W \notin \mathcal{W}$.

Now, let $k^* = w'(1-\rho)/q$ and observe that $k < k^*$ satisfies the hypothesis of Lemma 3. Choose $c(1-\rho) \geq t$ and note that $kq/((1-\rho)w') = 4k\log_\sigma m/(m - k - 2\log_\sigma m)$. Lemma 3 thus implies that **Q-PE-**$\mathcal{F}_{\rho,c,q}$ has $O((k\log_\sigma m/m)^t)$-error.

Clearly $\mathbf{Avg}_{\textbf{Q-PE-}\mathcal{F}_{\rho,c,q}}(w,P,k) = \mathbf{Time}_{\textbf{Q-PE-}\mathcal{F}_{\rho,c,q}}(W,P,k) = O(cq) = O(t\log_\sigma m)$. □

3.2 Algorithm based on covering by pattern substrings

In 1994 Chang and Marr [2] proposed a variant of SET [1] with running time $O(n(k + \log_\sigma m)/m)$ for $k/m \leq 1/3 - O(\sigma^{-1/2})$. As in SET, Chang and Marr consider blocks of text of size $(m-k)/2$, and pinpoint occurrences of the pattern by identifying blocks that approximately match a substring of the pattern. This identification is based on splitting the text into contiguous substrings of length $\ell = t\log_\sigma m$ and sequentially searching the text substrings of length ℓ in the pattern allowing errors. The sequential search continues until the total number of errors accumulated exceeds k. If k errors occur before $(m-k)/2$ text characters are covered, then the rest of the window can be safely skipped.

The adaptation of Chang and Marr's approach to the design of probabilistic filters is quite natural. Indeed, instead of looking at ℓ-grams sequentially we just randomly choose sufficiently many non-overlapping ℓ-substrings in each block. We then determine the fraction of them that approximately appear in the pattern. If this fraction is small enough, then the block is discarded. See Algorithm 3 for a formal description of the derived probabilistic filter **CM-PE-**$\mathcal{F}_{c,\rho,\ell,g}$. Using the filter as a subroutine for the generic algorithm with errors described in Algorithm 1 gives rise to a procedure to which we will henceforth refer to as **CM-PE**.

Remark 1. Note that $asm(S,P)$ of Algorithm 3 can be precomputed for all values of $S \in \Sigma^\ell$.

Algorithm 3 Probabilistic filter based on covering by pattern substrings

1: **procedure CM-PE-**$\mathcal{F}_{c,\rho,\ell,g}(W, P, k)$ ▷ $W \in \Sigma^w, P \in \Sigma^m$, $k \in \mathbf{N}$
2: $ctr \leftarrow 0$
3: **for** $i \in \{1, \ldots, c\}$ **do**
4: Choose j_i uniformly at random in $\{1, \ldots, \lfloor w/\ell \rfloor\}$
5: **if** $asm(W_{(j_i-1)\ell+1..j_i\ell}, P) \leq g$ ▷ $asm(S, P) = \min_{a \leq b} d(S, P_{a..b})$
6: **then** $ctr \leftarrow ctr + 1$
7: **if** $ctr > \rho \cdot c$ **then return** `Check` **else return** `Discard`

The analysis of Algorithm 3 establishes results such as Lemmas 1-3, but concerning **CM-PE-**$\mathcal{F}_{\rho,c,\ell,g}$. We can derive the following (proof omitted due to lack of space):

Theorem 3. *If* $k < m/5$*, then* **CM-PE** *is a* d-ASM *algorithm with one sided error* $\epsilon = (4k/(m-k))^t$*, for any constant* $t > 0$*. Its average running time is* $\mathbf{Avg}_{\mathbf{Q\text{-}PE}}(n, P, k) = O(tn \log_\sigma m/m)$.

4 Experimental Results

We implemented the algorithms of Sections 3.1 and 3.2. We extracted three real-life texts of 50MB from *Pizza&Chili* (`http://pizzachili.dcc.uchile.cl`): English text, DNA, and MIDI pitches. We used patterns of length 50 and 100, randomly extracted from the text, and some meaningful k values. Each data point is the average over 50 such search patterns, repeating each search 15 times in the case of the probabilistic algorithms. We measured the average number of character inspections and the average percentage of missed occurrences.

We used the following setup for the algorithms. For q-gram algorithms (Section 3.1), we used $q = 4$. Our preliminary results show that $\rho = 0.7$ is a good choice. For covering by pattern substrings (Section 3.2), we used $\epsilon = 0.2$ and $\rho = 0.3$. In our algorithms, we only moved parameter c in order to change the accuracy/time trade-off. We compared our algorithms with the corresponding errorless filtering algorithms.

Figure 2 shows the experimental results for the q-gram based procedure, and Fig. 3 for the covering by pattern substrings process. The errorless version of the q-grams algorithm inspects all text characters. In contrast, our q-gram based procedure achieves less than 1% error rate and looks at up to 6 times less characters on English and MIDI corpora. For our second algorithmic proposal, the result of the comparison against the errorless version is not as good. Nevertheless, we emphasize that it beats the average-optimal (errorless) algorithm by a wide margin, specifically it inspects about half the characters with 15% errors on the English corpus.

5 Final Comments

In this paper we have advocated considering a new dimension of the approximate string matching problem, namely the probability of missing an approximate oc-

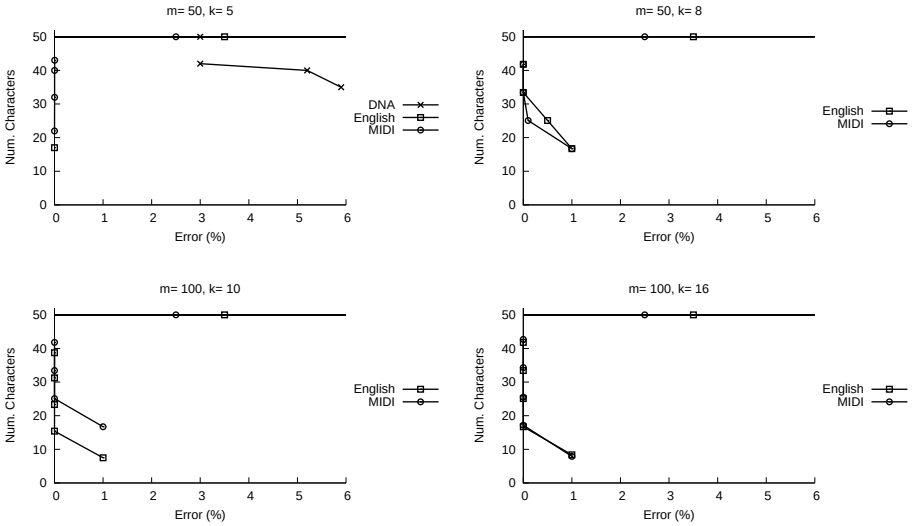

Fig. 2. Experimental results for **Q-PE**. Straight horizontal lines correspond to the errorless version. The y axis represents the number of character inspections times 1024.

currence. This relaxation is particularly natural for a problem that usually arises when modeling processes where errors have to be tolerated, and it opens the door to novel approaches to approximate string matching which break the average-case lower bound of the original problem. In particular, we have shown that much faster text scanning is possible if one allows a small probability of missing occurrences. We achieved $O(n \log_\sigma m/m)$ time (which is the complexity of exact string matching, $k = 0$) with error probability bounded by any polynomial in k/m. Empirically, we have shown that our algorithms inspect a fraction of the text with virtually no mistakes.

We have just scratched the surface of this new area. In particular, we have not considered filtration algorithms that use sliding instead of fixed windows. Sliding-window algorithms have the potential of being more efficient (cf. Fredriksson and Navarro's variant [3] with the original Chang and Marr's average-optimal algorithms [2]). It is not hard to design those variants, yet analyzing them is more challenging. On the other hand, it is rather simple to extend our techniques to multiple ASM. We also applied the techniques to indexed algorithms, where the text can be preprocessed [6]. Several indexes build on sequential filtration algorithms, and thus adapting them is rather natural.

Finally, it is interesting to determine the average complexity of this relaxed problem, considering the error probability ϵ in the formula. This would give an idea of how much can one gain by allowing errors in the outcome of the search. For example, our algorithms break the $\Omega(nk/m)$ term in the problem complexity, yet a term poly(k/m) appears in the error probability. Which are the best tradeoffs one can achieve?

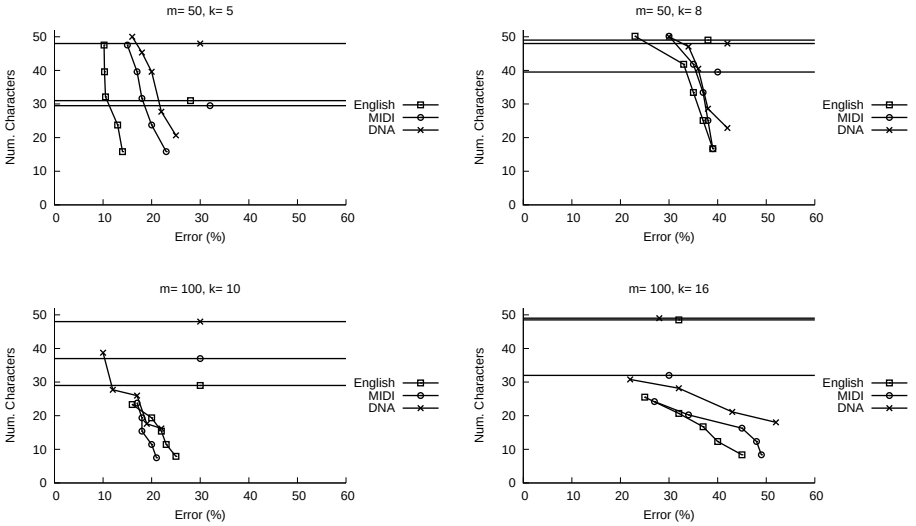

Fig. 3. Experimental results for **CM-PE**. Straight horizontal lines correspond to the errorless version. The y axis represents the number of character inspections times 1024.

References

1. W. Chang and E. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4-5):327–344, 1994.
2. W. Chang and T. Marr. Approximate string matching and local similarity. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 259–273. Springer-Verlag, 1994.
3. K. Fredriksson and G. Navarro. Average-optimal single and multiple approximate string matching. *ACM Journal of Experimental Algorithmics*, 9:article 1.4, 2004.
4. S. Janson. Large deviations for sums of partly dependent random variables. *Random Structure & Algorithms*, 24(3):234–248, 2004.
5. G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
6. G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001.
7. D. Ron. *Handbook of Randomized Computing*, volume II of *Combinatorial Optimization, Vol. 9*, chapter Property Testing. Springer, 2001.
8. R. Rubinfeld and R. Kumar. Algorithms column: Sublinear time algorithms. *SIGACT News*, 34(4):57–67, 2003.
9. E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92:191–211, 1992.