Finding Frequent Patterns in a String in Sublinear Time

Petra Berenbrink[1], Funda Ergun[2], and Tom Friedetzky[3]

[1] School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, Canada, WWW: http://www.cs.sfu.ca/~petra/
[2] School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, Canada, WWW: http://www.cs.sfu.ca/~funda/
[3] Department of Computer Science, Durham University, Durham, DH1 3LE, U.K., WWW: http://www.dur.ac.uk/tom.friedetzky/

Abstract. We consider the problem of testing whether (a large part of) a given string X of length n over some finite alphabet is covered by multiple occurrences of some (unspecified) pattern Y of arbitrary length in the combinatorial property testing model. Our algorithms randomly query a sublinear number of positions of X, and run in sublinear time in n. We first focus on finding patterns of a given length, and then discuss finding patterns of unspecified length.

1 Introduction

The problem of finding frequent occurrences of patterns in a string comes up in many areas such as telecommunications, e-commerce, and databases, where the applications generate long data streams to be analyzed. An example from data mining is efficient handling of iceberg queries, that is, identifying those objects in a data stream which occur with frequency over a threshold. In property testing of strings, testing whether a string consists of back-to-back repetitions of the same patterns is called *periodicity testing*. Usually, the efforts to efficiently identify such trends in data are hampered by the large size of the data, which can be too large to fit into main memory and to be efficiently analyzable, even by a linear time algorithm.

In this work, we are interested in detecting frequent repetitions of a pattern (of any size) in a string of length n in time sublinear in n. In contrast to previous work, the pattern boundaries are unrestricted, which, while more realistic, complicates matters. We explore this problem in the *combinatorial property testing* model ([10, 3]), and first obtain an algorithm which distinguishes between strings which are mostly covered with occurrences of one pattern of given size k and those that do not contain a large number of repeated patterns in $o(k)$ time. We then generalize our result to detecting repetitions of patterns of *unspecified* length, in $o(n)$ time. In both cases, if a frequent pattern exists, the algorithm can implicitly return a (likely approximate) copy of the pattern.

The fact that patterns can occur anywhere in the string means that there can be a linear number of possible patterns of linear size in a given string, and comparing them to one another can easily lead to inefficient algorithms. To handle this, we represent each pattern as a short "sketch" or a "signature". However, there can still be a linear number of signatures sharing one location. To deal with this, we use a sparse representation of our data which is based on the few locations sampled. We show that this small amount of information is sufficient to make conclusions about repeated trends in the input stream.

Our Results. We first present an algorithm which, given a string X and a length k, tests in $O(\sqrt{k}\text{polylog}k)$ time if there exists a pattern Y of length k that covers X (notice that Y is not given); allowing the occurrences of the pattern to overlap. We say that Y *approximately covers* an α-fraction of X if there exists a set of substrings $\mathcal{Z} = \{Z_1, \ldots, Z_j\}$ of X (each of length k) where $h(Z_i, Y) \leq \epsilon k$ for all $Z_i \in \mathcal{Z}$ and at least αn locations of X are covered by some $Z_i \in \mathcal{Z}$. Here $h(Z_i, Y)$ is the *Hamming distance* between Y and Z_i. If a pattern of length k exists that covers all of X, with probability $1-o(1)$ our algorithm outputs PASS. If there is no pattern Y of length k that covers an α-fraction of X, $\alpha, \epsilon \in (0,1)$, it outputs FAIL with probability $1 - o(1)$.

Next, we give an algorithm which, given X and k, tests in time $O(\sqrt{k}\text{polylog}k)$ whether there is a string of a length $\ell \in [\delta k, \ldots, k]$ covering an α-fraction of X; it outputs FAIL if there is no pattern Y of length k that approximately covers an $(1-\beta)\alpha$-fraction of X, with probability 3/4, with certain restrictions on α, β and ϵ. We use this variant to test if there is a pattern of *any* length that covers an α-fraction of X, or if no pattern exists that approximately covers a $(1-\beta)\alpha$-fraction of X.

Related Work. Combinatorial property testing was first defined by [10, 3]. For an overview of results see [9] and the references therein. Two recent related results testing whether a string is close to periodic are [2, 7]. They assume that the size of the period is fixed so that position in which the period (corresponding to "patterns") appear is fixed, too. This means that it is more or less clear which positions should be sampled. In our case, since a pattern can be anywhere, we need to make sure that we have samples in all possible places that a pattern can be [4]. Another related result tests in sublinear time whether two strings have large edit distance ([1]).

There have also been sublinear space streaming results. Iceberg queries for identifying objects that appear in more than a fraction of a string are explored in [6]. Periodicity testing is investigated using sketches in [5] where the running time is considered in terms of memory accesses. These techniques differ from ours that they are not bound by sublinear time, but are expected to return more accurate answers.

2 Preliminaries

Given string X of length n, let $X[i]$ refer to the ith character of X. $r = [i : j]$ denotes $\{i, i+1, \ldots, j\}$, where i and j are called respectively the *left and right*

[4] To achieve this, we use an extra stage of sampling where one of the stages makes sure that two copies of the same pattern will be correctly aligned.

endpoints of r. $X[i : j]$ denotes the substring of X starting at location i and ending at j. $[n]$ is short for $[1 : n] = \{1, \ldots, n\}$.

Given a location i in X, we say that a length k substring (or pattern) Y "covers", "contains", or "appears around" i if and only if $Y = X[j : j + k - 1]$ such that $i \in [j : j+k-1]$. Here Y only refers to the contents of the substring, and thus, can be repeated elsewhere in X. We call each such repetition an *occurrence*. $h(X, Y)$ denotes the Hamming distance between X and Y.

3 Finding frequent patterns of given length k

We first consider finding patterns of length *exactly* k.

3.1 Length exactly k

We now formally define the problem of testing for frequent patterns. Formally, given a string X of length n and $1 \leq k \leq \alpha n$, we would like to have an algorithm with the following behavior.

- If there is a pattern Y of length k which covers all locations of X, then the algorithm returns Pass with probability $1 - o(1)$.
- If there is no pattern Y of length k such that a set of substrings $\mathcal{Z} = \{Z_1, \ldots, Z_j\}$ of X (of length k) exist where $h(Z_i, Y) \leq \epsilon k$ for all $Z_i \in \mathcal{Z}$ and at least αn locations of X are covered by some $Z_i \in \mathcal{Z}$, then the algorithm returns Fail with probability $1 - o(1)$.

Note that the occurrences of the pattern can overlap. For a visual intuition on the Fail condition, consider a scheme to mark an X with respect to a pattern Y of length k. For $j = 1, \ldots, n - k + 1$, if $h(Y, X[j : j + k - 1]) \leq \epsilon k$ then mark locations $X[j], \ldots, X[j + k - 1]$. In the end, if some $X[i]$ remains unmarked, then there exists no substring Z of X of length k that covers location i such that $h(Z, Y) \leq \epsilon k$. The Fail condition holds if and only if the marking of X with respect to Y results in at least $(1 - \alpha)n$ unmarked locations for any Y of of length k.

Consider $X =$ `abcabcaabcaabca`. $Y =$ `abca` covers X fully, where the `a` in location 4 is covered by two overlapping copies of Y, thus the algorithm should return Pass. Now let $X' =$ `abcdbaddacdedbcbe`. Substrings `abcd`, `dbad`, `dacd`, `dbcb`, cover all but two characters of X', and each has Hamming distance 1 to $Y =$ `dbcd`; thus the algorithm can return either Pass of Fail. Later, we will show how our results translate into the non-overlapping case, where our definition of distance will be equivalent to the usual one.

The approach of randomly choosing a few locations in X and checking whether there is a pattern which covers all of these locations is not straightforward to implement in sublinear time, for two reasons. First, even if two random locations i and j lie in two occurrences of the same pattern, they are likely to be in different positions within the two occurrences, with k^2 possible location pairs. This hurdle is not present in periodicity testing where the pattern boundaries are fixed. Second, the fact that locations p_1 and p_2, as well as p_1 and p_3 occur within the same pattern does not imply that p_2 and p_3 do. Two patterns can share $\Theta(k)$ many patterns of length k; thus, finding one shared by all of the sample points in $o(n)$ time is nontrivial.

A Three-Stage Sampling Approach We now present our approach which tackles the above problems in time $o(k)$. To do that, we will use sampling and keep small summaries of our samples in "signatures". Let $\ell_p = \Theta(\text{polylog} n)$, $\ell_s = \Theta(\sqrt{k} \log \log k)$, $\ell_t = \Theta(\text{polylog} n)$, with large enough constants hidden in the Θ. Our sampling has three stages, where Stages 1 and 2 obtain *primary* and *secondary locations* and Stage 3 the actual samples.
Stage 1: Construct set $P = \{p_0, p_1, \ldots, p_{\ell_p}\}$, of *primary locations*, where each p_i is chosen independently and uniformly at random (i.u.r.) from $[n]$.
Stage 2: For each $p_i \in P$, construct set S_i of *secondary locations*, said to be *owned* by p_i, of the form $S_i = \{s_{i,0}, s_{i,1}, \ldots, s_{i,l_s}\}$, [5] where each $s_{i,j} \in S_i$ is chosen i.u.r. from $[p_i - k : p_i + k]$. [6]
Stage 3: Construct a *sorted* list of locations $T = t_1 t_2 \ldots t_{\ell_t}$ where the t_i are picked i.u.r. from $[-2k : 2k]$ and are in ascending order. Now consider any secondary location $s_{i,j} \in S_i$. Obtain samples $T_{i,j} = s_{i,j}+t_1, s_{i,j}+t_2, \ldots, s_{i,j}+t_{\ell_t}$; these will be owned by $s_{i,j}$. The elements of T_{ij} are uniformly distributed in $[s_{i,j} - 2k : s_{i,j} + 2k]$; furthermore, the locations of the samples relative to any secondary location s that owns them is identical across all s (Fig. 1).

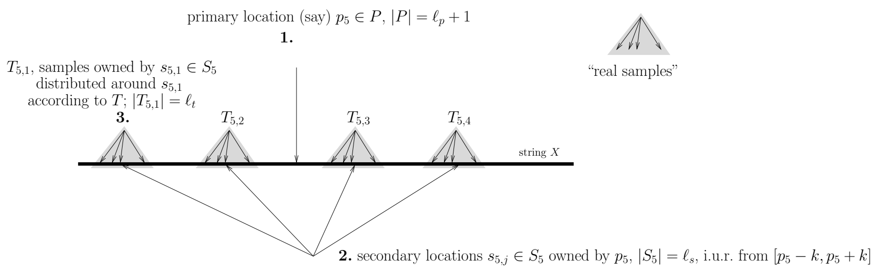

Fig. 1. Example for primary location p_5; steps (1,2,3) indicate order of selection

Templates and Signatures. We will represent each substring of length k of X with a short signature. To do this, decompose T into sublists which we call *templates*, each of which contains the offsets to obtain samples to form a signature.

Definition 1. *A list τ is said to be a template (of T) if for some $-2k < i \leq k+1$, τ, τ is the maximal sublist of T whose elements are in the range $[i : i + k - 1]$.*

The following lemma shows templates are large enough.

Lemma 1. *Let $\ell_t = 24\bar{c} \log k$ for some large enough constant c. With probability at least $1 - o(1/k)$ every template consists of at least $\bar{c} \log k$ characters.*

Proof. Consider the probability that there exists a $j \in [-2k : 2k - j + 1]$ such that there are fewer than $\bar{c} \log k$ elements in T whose values are in $[j : j + k - 1]$. To do this, partition the interval $[-2k : 2k]$ into 12 subintervals of length $k/3$.

[5] We will drop subscripts later when they are obvious.
[6] If $p_i < k$ ($p_i > n - k$) then the area $[1 : p_i + k]$ ($[p_i - k : n]$) is sampled.

Any interval $[j : j+k-1]$ will fully contain such subinterval. The expected number of elements of T in a subinterval is $2\bar{c}\log k$, which is a lower bound on the expected length of a signature. Using Chernoff bounds (see e.g. [4]) the probability that a particular subinterval will have fewer than $\bar{c}\log k$ samples is at most $e^{-\bar{c}\log k} = o(1/k)$. Thus, the probability that at least one subinterval will have fewer than that many elements is also at most $o(1/k)$.

The next observation holds by symmetry, showing that the sample points are uniformly distributed over a template.

Observation 1 *For an interval $r = [i : i+k-1]$ for $-2k \leq i \leq k+1$, given that the template representing r contains p locations, the set consisting of these p locations is uniformly distributed in the subsets of size p of $\{i, \ldots, i+k-1\}$.*

Using the elements of a template as offsets with respect to a secondary location to obtain actual samples, we obtain a *signature*:

Definition 2. *Let $s = s_{l,m}$ be a secondary location and $\tau = t_i, t_{i+1}, \ldots, t_j$ be a template representing some interval $[u : u+k-1]$ for $-2k \leq u < k+1$. The signature corresponding to τ with respect to $s_{l,m}$ is $sig_\tau(l,m) = X[s+t_i], X[s+t_{i+1}], \ldots, X[s+t_j]$, representing the interval $[s+u : s+u+k-1]$.*

Let $\mathcal{T} = \tau_1, \tau_2, \ldots$ denote the list of all templates of T. Below we show that there are not too many distinct templates. This imposes an $O(\sqrt{k} \cdot \text{polylog}k)$ bound on the total number of signatures generated. The proof is omitted.

Lemma 2. $|\mathcal{T}| \leq 2\ell_t$. *Furthermore, the total number of signatures generated from X from the locations and samples obtained as above is at most $2\ell_t\ell_s(\ell_p+1)$.*

Since there are many more intervals of length k than templates, we now build a succinct representation of their correspondance.

Definition 3. *Let τ be a template. Let $Q = \{i \mid -2k \leq i \leq k$ and interval $[i : i+k-1]$induces $\tau\}$. The range of τ, $r(\tau)$, is $[a : b]$, with a and b as the left and right endpoints of Q.*

The notion of the range of a template extends naturally to the range of a signature. Let $s_{i,j}$ be any secondary location and $[a : b]$ be the range of some template τ. Then the range of $sg = sig_\tau(i,j)$, denoted $r(sg)$, is $[a+s_{i,j} : b+s_{i,j}]$. We observe below how to compute the range of a template (the range of a signature is computed similarly). Let $t_0 = -2k-1$ and $t_{l_t+1} = 2k+1$.

Observation 2 *Let $\tau = t_i, t_{i+1}, \ldots, t_j$ be a template. Then, $r(\tau) = [\max\{t_{i-1}+1,\ t_j-k+1\} :\ \min\{t_i,\ t_{j+1}-k\}]$.*

The Basic Sampling Algorithm Our algorithm consists of two phases. In the initialization phase we construct data structure D with signatures related to the first primary location, p_0. In the next phase we compare signatures of other primary locations, to those already considered. If we identify a pattern which occurs around all our primary locations, we return Pass. In what follows, let c be a sufficiently large constant.

Initialization Phase:

Obtain sets P, S of primary, secondary locations, T of offsets, and $\mathcal{T}$ of templates
If there exists a template $\tau \in \mathcal{T}$ with less than $c \log k$ sample points return FAIL
Set $D = \phi$; $G = \phi$;
For each secondary location $s_{0,i}$ for $1 \leq i \leq \ell_s$ and each template $\tau \in \mathcal{T}$
    $sg = sig_\tau(0, i)$
    let $r = r(\tau) \cap [p_0 - s_{0,i} - k + 1 : p_0 - s_{0,i}]$
    $R = \{r\}$
    if sg does not exist in D, insert $< sg, R >$ in D;
    otherwise let R' be the range of the entry found in D for sg
    change the range of the entry for sg in D to $R' \cup R$
    $G = G \cup R$

The operation taking intersection of the ranges ensures that substrings which do not intersect with p_0 are not considered[7].

Iterative Phase:

For $m = 1$ to ℓ_p do
    $D' = \phi$; $G' = \phi$;
    Fill out D' with signatures around p_m as D was filled above for p_0
    For each signature sg in D' with range R
        If sg exists in D with range R', $G' = G' \cup R$
    $G = G \cap G'$

Output: If $D \neq \phi$ return PASS, otherwise return FAIL.

Data Stuctures. Data structures D and D' store modified signatures and their ranges. A modified signature is obtained (from, say, $sg = sig_\tau(i, j)$) tagging sg with a prefix, namely the smallest index in τ. This ensures two matching (modified) signatures will come from the same template. Each node contains a signature and its current range set R, representing the (candidate) frequent substrings which have this signature. Both D, D' and R can be implemented by using any standard data structure that supports linear time construction and logarithmic time search and updates, as well as constant time *prev* and *next* operations. G and G' store ranges in a similar way. Inserting a range into R can take linear time due to the deletions of small ranges during merging. The deletion of a range can be charged to its insertion, maintaining logarithmic amortized insertion and deletion times. The union and intersection operations all are performed in logarithmic time per range.

Analysis of the Algorithm

Theorem 3. *Let X be a string of length n and parameter k be such that $1 \leq k \leq \alpha n$. Let $\ell_p = c \cdot \log k$, $\ell_s = c' \sqrt{k \log k}$ and let $\ell_t = 24\bar{c} \log k$ with sufficiently large constants $c, c', \bar{c}$.*

(a) If there is a pattern Y of length k that covers 100% of X, then the algorithm returns PASS *with probability at least $1 - o(1)$.*

[7] When we take a union of ranges, ranges which touch or overlap are merged.

(b) If there is no pattern Y of length k such that at least αn characters of X can be covered by substrings $Z_1, \dots, Z_w$ (of length $|Y|$) where $h(Z_i, Y) \le \epsilon k$ then the algorithm returns Fail *with probability at least* $1 - o(1)$.

The algorithm runs in $O(\sqrt{k}polylogk)$ time and space.

Proof. We start with the proof of Part a. The runtime analysis is submitted in this short version.

(a): If the Pass condition is satisfied, we can get an outcome of Fail if one the two following cases happens.
(i) For some p_i, we do not have a pair of "well aligned" secondary samples s, s' belonging to p_0 and p_i respectively. By assumption, we have a copy of a string Y covering p_0 and one covering p_i. To detect that these two copies are identical, we need to get identical signatures from them, for which we need to have secondary locations s, s' with identical relative locations w.r.t. the first and the second copy of Y respectively. By the birthday paradox (see [8], Page 45), the probability that we will not have such a "well aligned" pair of secondary locations for one particular p_i is at most $e^{-\ell_s(\ell_s-1)/2k} = e^{-(c'^2 k \log k - \sqrt{c' k \log k})/2k} \le e^{-(c'^2 k \log k)/4k} \le k^{-(c')^2/4}$ for $c' \ge 2\sqrt{c+1}$. Using the union bound, the probability that this situation might arise for some p_j is $1/k$.

(ii) We get a Fail answer due to a signature which is smaller than the threshold. By Lemma 1 this can only happen with a probability of $o(1)$. Thus, the probability of an incorrect Fail answer is at most $o(1)$.

(b): If the Fail condition is satisfied, a Pass can be returned as as a result of two events, analyzed below.
(i) Choice of primary locations: Call two substrings of size k Z_1 and Z_2 *similar* if $h(Z_1, Z_2) \le \epsilon k$. With the Fail condition, a small number of the primary locations $p_1, \dots, p_{\ell_p}$ may be covered by substrings which are similar to one particular substring around p_0. p_0 is covered by at most k different substrings of length k; WLOG consider $Y = X[p_0 - k + 1 : p_0]$. Due to the Fail assumption, marking X w.r.t. Y will leave at least $(1-\alpha)n$ positions unmarked. The probability that a fixed primary location will fall on a marked position for a fixed string Y then is at most $1 - \alpha$.
(ii) Unlucky choice of templates: The signatures for two substrings Y and Y' can be identical even if Y and Y' differ in more that ϵk locations. This is a problem only if the signatures are at least $\bar{c} \log k$ characters long, since otherwise the algorithm automatically returns Fail. (Note that for a match of signatures to be found, the two signatures must be generated from the same template, which guarantees that the two substrings are being compared at corresponding locations.)

Note that, due to how the signature of Y has been picked, the second statement of Lemma 1 and the bound on the size of a signature, the samples in the signature of Y correspond to a uniformly chosen subset of $\bar{c} \log k$ samples from Y. Assume that the signature for Y' has been obtained from the same template as that of Y (the opposite of this only helps us). For the two signatures to match, none of the samples in the signatures must be from locations where Y and Y' differ. Since Y and Y' are not similar, the probability of this is at most $\epsilon^{\bar{c}\log k} \leq 1/k^3$. For any pair of primary locations p_0 and p_i we compare at most $\ell_s^2 \cdot 2\ell_t$ signatures with each other (see Lemma 2). The probability to find identical signatures for a pair of primary locations p_0 and p_i is at most $(\ell_s^2 \cdot 2\ell_t) \cdot \frac{1}{k^3} \leq \frac{1}{k}$. Since, given p_0, there are at most k possible choices for Y, the probability of a false negative/false positive is at most $k \cdot \left((1-\alpha) + \frac{1}{k}\right)^{\ell_p} \leq o(1)$.

We now present a lemma relating the result with overlapping patterns to non-overlapping patterns. The proof is omitted.

Lemma 3. *If αn characters of a string X of length n are covered by overlapping patterns of length k, then at least $\alpha n/2$ characters are covered by non-overlapping patterns.*

3.2 Length approximately k

In this section we show how to test if any pattern of length in the range $[\delta k : k]$ for constant $\delta < 1$ occurs over a large fraction of a given string X. We first develop a high level algorithm similar to that in Section 3.1. Later, we will use this algorithm to find out if there is *any* pattern (of any size) which occurs frequently in X.

We define our modified algorithm in terms of its differences from the algorithm in Section 3.1. First, a template is now defined as the maximal sublist of T whose elements represent a range $[i : i + \delta k - 1]$ (see Definition 1). Consequently, a signature now spans an area of size δk.

The second change is in our data structures. In our previous algorithm, to identify a pattern as frequent, we confirmed that it occurred around all our primary locations. Here, we will check that a pattern occurs around a large number of primary locations. To count the occurrences of patterns around primary locations, we replace D with DR, described below. In the new algorithm, at the end of the iterative section, rather than taking an intersection of the ranges (along with signatures) found for the new p_m with the existing candidates ranges in D, we now simply add the new ranges found to DR. (Which keeps track of how many times a range has been added). At the end, if there is a particular pattern that occurs around many of the primary locations, it will be witnessed by DR that the signature and range representing the pattern have a large count (one for each occurrence around a primary location).

The algorithm outputs PASS if there is a signature and corresponding range (thus, a pattern) found around at least $\alpha\delta\ell_p$ primary locations, for some constants $\alpha, \delta < 1$, according to the count obtained from DR.

Data Structure for the Modified Algorithm. We use a data structure DR to store ranges in terms of their endpoints. DR is, like D, a standard data structure. Each node contains three fields: a *value* for an endpoint of a range, a *count* tracking the times that an endpoint has been encountered, and a one bit field containing the values *left* or *right* to qualify an endpoint. Here one can insert a range in logarithmic time, output how many times each (sub)range has been inserted in linear time for all of the ranges. For instance, if $[2:8]$ and $[6:14]$ have been inserted, DR has value 1 for $[2:5]$ and $[9:14]$, and 2 for the intersection, $[6:8]$.

To insert a range $[a, b]$, we first look for a in DR. If it is not found, we insert $(a, left, 1)$ into DR. If a exists and the endpoint bit shows *left*, we increment the count field for that entry; if the endpoint bit shows *right* we decrement the count. We treat b similarly: if the value is not found, we insert $(b, right, 1)$. If an entry exists and the endpoint is $right$, we increment the count; if the endpoint is $left$, we decrement the count. If at any point the count at a node reaches zero, we delete the node.

To obtain a count of the ranges, we use a range counter (initially set to 0), starting from the smallest value in DR and following the *next* pointers. For every node we see with endpoint $left$ the we increment the range counter by the count in that node; for every node with endpoint *right* we decrement by the count in that node. The value of the counter between two nodes in DR represents how many times the range delimited by the values in those two nodes has been inserted.

Analysis of the Algorithm In this section we will prove that the algorithm works correctly. First we show that whenever there is a pattern of length k covering an α-fraction of the string, then there is a pattern of length δk covering an $\alpha\delta$-fraction. The proof is omitted.

Lemma 4. *Let $\alpha \in (0, 1)$. Let X be a string of length n. Let $\delta \in (0, 1)$.*

(a) Whenever there exists a pattern of length ℓ with $\delta k \leq \ell \leq k$ that covers at least an α-fraction of C, then there also exists a pattern of length δk that covers at least an $(\alpha\delta)$-fraction of the string.

(b) Whenever there exists a pattern Y of length ℓ with $\delta k \leq \ell \leq k$ such that at least an α-fraction of X can be approximately covered by substrings $Z_1, \ldots, Z_j$ (of length $|Y|$) where $h(Z_i, Y) \leq \epsilon k$, then there also exist a pattern Y' of length δk and substrings $Z'_1, \ldots, Z'_j$, such that least an $(\alpha\delta)$-fraction of the string can be covered by the $Z'_1, \ldots, Z'_j$ with $h(Z'_i, Y') \leq \epsilon' \delta k$ where $\epsilon' = \epsilon/\delta$.

Theorem 4. *Let $k \geq 100$. Let $\alpha \in [\frac{4}{5}, 1)$, $\beta \in (0, 1)$ with $\alpha(1-\beta) \leq \frac{2}{3}$. Let $\delta = \frac{40}{41}$. Let X be a string of length n. Let $\ell_p = c \cdot \log k$, $\ell_s = c'\sqrt{k \log k}$ and let $\ell_t = \bar{c} \log k$ for large enough constants $c, c', \bar{c}$.*

(a) If there is a pattern Y of length ℓ with $\delta k \leq \ell \leq k$ that covers an α-fraction of X, then the algorithm returns Pass *with probability at least $3/4$.*

(b) If there is no pattern Y of length ℓ with $\delta k \leq \ell \leq k$ such that at least an $\alpha(1-\beta)$-fraction of X can be covered by substrings $Z_1, \ldots, Z_j$ (of length $|Y|$) where $h(Z_i, Y) \leq \epsilon k$ then the algorithm returns Fail *with probability at least $3/4$.*

The algorithm runs in $O(\sqrt{k}$polylog $k)$ time and space.

Proof. **(a)** We can get a Fail answer if one of the following three cases happen. (i) For some p_i, we do not have a pair of "well aligned" secondary samples s and s' belonging to p_0 and p_i respectively. Using the birthday paradox we can show

that the probability that there exists a primary location that we can not align to p_0 is $1 - o(1)$.
(ii) We get a FAIL due to a signature which is too small. Lemma 1 shows that this will only happen with a probability of $o(1)$.
(iii) We get a FAIL because not sufficiently many of our primary location positions fall into the pattern. Using Lemma 4, the probability that a fixed primary location does hit the occurrence of the pattern is at least $\alpha\delta$, thus p_0 will not be in the pattern with a probability of $1 - \alpha\delta$. From the remaining ℓ_p primary locations, expected $\alpha\delta\ell_p$ samples will fall into an occurrence of the pattern. Using Chernoff bounds from [4], we can show that the probability that fewer than $(1-\gamma)\cdot\alpha\delta\ell_p$ of $p_1, \ldots, p_{\ell_p}$ fall within an occurrence of the pattern is at most $1/k$. We can make γ a constant as close to zero as we wish by making the constant c (the coefficient of $\log k$ in ℓ_p) large enough.

Putting things together, the probability that the algorithm outputs FAIL in the PASS case is is most $o(1) + (1 - \alpha\delta) + 1/k = o(1) + (1 - (40/41)\alpha) + 1/k = o(1) + (1/41)\alpha + 1/k \leq 1/4$ for k a large enough constant.

(b) We now consider the probability of a PASS answer if the FAIL condition is satisfied. Notice that our algorithm now allows for finding patterns (of length between δk and k) that actually do not contain primary locations. As we choose secondary locations within a $\pm k$ radius around primary locations, a primary location may have a distance of up to $(1-\delta)k$ from an endpoint of an occurrence of the pattern, and still be able to identify the pattern as such. We refer to these regions of size $(1 - \delta)k$ to the left and to the right of an occurrence as *extra regions*.

Consider the modification of the marking game from Section 3.1, that marks all locations that allow for identifying occurrences of the pattern Y, by marking both the occurrences of Y itself, as well as all the corresponding extra regions. It is easy to see that if there does *not* exist a pattern of length ℓ with $\delta k \leq \ell \leq k$ that covers an $\alpha(1 - \beta)$-fraction of X, the modified marking scheme will mark at most $\alpha(1 - \beta)n + (\alpha(1 - \beta)n/\delta k) \cdot 2(1 - \delta)k = \alpha(1 - \beta)n \cdot (2/(40/41) - 1) = \alpha(1-\beta)n\cdot(1+1/20)$ locations. The first term, $\alpha(1-\beta)n$, is an upper bound on how much the actual pattern can cover, whereas the second term is an upper bound on the number of occurrences of the pattern, multiplied with the size of the extra regions (of which there are two for every occurrence). Let $\mu = \alpha(1-\beta)\left(1 + \frac{1}{20}\right)$, i.e., the coefficient of n in the above expression.

Fix an occurrence Y of length δk that is identifiable by p_0, i.e., p_0 is contained in either Y itself, or in one of the two extra regions around Y. Notice that there are $k + 2(1 - \delta)k = (3 - 2\delta)k$ many choices for Y. There are two cases that let the algorithm find a pattern between p_0 and some p_i

(i) Unlucky choice of primary locations: too many of the primary locations $p_1, \ldots, p_{\ell_p}$ may be covered by substrings which are similar to Y. Hence, the probability that a primary location is close enough to an occurrence of the pattern Y is at most μ (as defined above).

(ii) Unlucky choice of templates: The signatures for two substrings Y and Y' can be identical even if Y and Y' differ in more that $\epsilon' k$ locations (see Lemma

4, part (b)). Similar to Theorem 3, we can show that in this case the probability to find identical signatures for a pair of primary locations p_0 and p_i for $i \in \{1, \dots, \ell_p\}$ is at most $1/k$.

Similar to the proof of Theorem 3 we can argue that the probability to find identical signatures for a pair of primary locations p_0 and p_i is at most $\mu + 1/k$ for a fixed pattern Y. Hence, the expected value for the counter of Y is $(\mu + 1/k) \cdot \ell_p$. Using Chernoff bounds [4], it is easy to show that the probability that the algorithm finds more than $(1+\gamma')(\mu + 1/k)\ell_p$ copies of Y, is at most $1/k^3$. Again, we can obtain a (constant) γ' as close to zero as we wish by choosing a sufficiently large value of c.

For the Pass and Fail case to be distinguishable, we need $(1+\gamma')(\mu + 1/k) = (1+\gamma')(\alpha(1-\beta)(1+1/20)+1/k) + \lambda \leq (1-\gamma)\cdot \alpha\delta$ for some (constant) $\lambda > 0$. Choose c large enough such that $(1+\gamma') \leq \frac{100}{99}$. Since $k \geq 100$, $(1+\gamma')(\alpha(1-\beta)(1+1/20)+1/k) \leq 71/99$. Furthermore, we can choose $(1-\gamma) \geq \frac{41}{42}$, and thus $(1-\gamma)\alpha\delta \geq \frac{41\cdot 4\cdot 40}{42\cdot 5\cdot 41} = \frac{32}{42} = \frac{16}{21}$. Therefore, we indeed have a gap of $\lambda = \frac{16}{21} - \frac{71}{99} = \frac{31}{693}$.

Since there are at most $k + (1-\delta)k = (2-\delta)k$ possible choices for Y, the probability of a false negative/false positive is at most $(2-\delta)k \cdot \frac{1}{k^3} = o(1)$.

Using several runs of the algorithm together with simple majority vote it is easy to strengthen the results such that the algorithm gives the right answers with a polynomial small probability. In the following we refer to this algorithm as the reliable version of the algorithm that finds variable length patterns. The runtime is still $O(\sqrt{k}\text{polylog}k)$.

Note that the algorithm can be easily modified to answer Pass if, say, x percent of the string is covered with a pattern, and Fail if less than $x - \alpha$ percent of the string are covered with a pattern.

4 Finding frequent patterns of unspecified length

In this part we will use the reliable version of the algorithm that finds variable length patterns in order to search for all patterns that cover most parts of the string. The new algorithm works in $\log_{1/\delta} n$ rounds. In round i ($1 \leq i \leq \log_{1/\delta} n$), we search for patterns of length ℓ with $\delta^i n \leq \ell \leq \delta^{i+1} n$.

The algorithm works on an *output table* which has an entry for every i with $1 \leq i \leq \log_{1/\delta} n$. It writes Pass (Fail) in position i of the array if the algorithm outputs Pass (Fail) in round i. We can prove the following Theorem.

Theorem 5. *Use the same definitions as in Theorem 4 and run the modified algorithm for $\Theta(\log n)$ times per round. Furthermore, fix $\delta < 1$ and choose r such that $\delta^r \geq 100$.*

(a) For every $i \leq r$ such there exists a pattern Y of length ℓ with $\delta^{i+1} n \leq \ell \leq \delta^i n$ that covers an α-fraction of X, the algorithm writes a Pass *into position i of the output array with a probability of $1 - n^{-1}$.*

(b) For every $i \leq r$ such there exists no pattern Y of length ℓ with $\delta^i n \leq \ell \leq \delta^{i+1} n$ such that at least an $\alpha(1-\beta)$-fraction of X can be covered by substrings

$Z_1, \ldots, Z_j$ *(of length* $|Y|$*) where* $h(Z_i, Y) \le \epsilon k$ *the algorithm writes a* FAIL *into position* i *of the output array with a probability of* $1 - n^{-1}$.

The algorithm runs in $O(\sqrt{k} polylog k)$ *time and space.*

Proof. The proof follows directly from Theorem 4. The array has $o(n)$ entries and for every i the algorithm answers PASS (FAIL) correctly with a probability of $1 - n^{-2}$.

5 Conclusions

It is also possible to define an algorithm for which a constant number of primary locations is sufficient, rather than $O(\log k)$ as in the previous sections. However, since "nothing is for free" there is a bigger gap in the pattern length between the PASS and FAIL cases. Notice that for our algorithms a constant number of primary locations is not enough since we essentially search for the k possible patterns of length k that contain the primary location p_0. This means that, for a fixed pattern Y which includes p_0, the probability that all primary locations are contained in the same pattern has to be at most $1/k$ for the FAIL case. Since the probability that a fixed primary location is contained in a fixed pattern (*not* a fixed occurrence of a pattern) is constant, we need $\log k$ many primary locations. This algorithm will be presented in the full version. Unfortunately, it is in general not possible to determine the *longest* pattern occurring in the string, whilst guaranteeing a probability for correctness of the answer, using our model. See the full version of this paper for a more detailed discussion

References

1. T. Batu, F. Ergun, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld and R. Sami. *A sublinear algorithm for weakly approximating edit distance. STOC 2003, 316–324.*
2. F. Ergun, S. Muthukrishnan, and C. Sahinalp, *Sublinear methods for detecting periodic trends in data streams. Latin American Symposium on Theoretical Informatics (LATIN), 2004.*
3. O. Goldreich, S. Goldwasser and D. Ron. *Property testing and its connection to learning and approximation*, *Journal of the ACM 45(4):653–750, 1998.*
4. T. Hagerup and C. Rüb. *A Guided Tour of Chernoff Bounds. Information Processing Letters* **33** (1989), pp. 305–308.
5. P. Indyk, N. Koudas and S. Muthukrishnan, *Identifying Representative Trends in Massive Time Series Data Sets Using Sketches. Proc. VLDB 2000. 363–372.*
6. R. Karp, S. Shenker, and C. Papadimitriou, *A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. 28: 51-55 (2003)*
7. O. Lachish and I. Newman, *Periodicity Testing. Proc. RANDOM 2005, to appear.*
8. R. Motwani and P. Raghavan, *Randomized Algorithms. Cambridge University Press (1995)*
9. D. Ron, *Property Testing (A Tutorial). Handbook of Randomization, 2000.*
10. R. Rubinfeld and M. Sudan, *Robust Characterization of Polynomials with Applications to Program Testing*, *SIAM Journal of Computing 25(2):252–271, 1996.*