Streaming for Aibohphobes: Longest Palindrome with Mismatches

Elena Grigorescu[1], Erfan Sadeqi Azer[2], and Samson Zhou[3]

1 **Department of Computer Science, Purdue University, West Lafayette, USA**
elena-g@purdue.edu.
2 **School of Informatics and Computing, Indiana University, Bloomington, USA**
esadeqia@indiana.edu.
3 **Department of Computer Science, Purdue University, West Lafayette, USA**
samsonzhou@gmail.com.

Abstract

A palindrome is a string that reads the same as its reverse, such as "aibohphobia" (fear of palindromes). Given a metric and an integer $d > 0$, a *d-near-palindrome* is a string of Hamming distance at most d from its reverse.

We study the natural problem of identifying the longest d-near-palindrome in data streams. The problem is relevant to the analysis of DNA databases, and to the task of repairing recursive structures in documents such as XML and JSON.

We present the first streaming algorithm for the longest d-near-palindrome problem that returns a d-near-palindrome whose length is within a multiplicative $(1+\epsilon)$-factor of the longest d-near-palindrome. Our algorithm also returns the set of mismatched indices in the d-near-palindrome, and uses $\mathcal{O}\left(\frac{d\log^7 n}{\epsilon\log(1+\epsilon)}\right)$ bits of space, and $\mathcal{O}\left(\frac{d\log^6 n}{\epsilon\log(1+\epsilon)}\right)$ update time per arrival symbol. We show that for $d = o(\sqrt{n})$, any randomized algorithm with multiplicative approximation $(1+\epsilon)$ that succeeds with probability at least $1 - 1/n$ requires $\Omega(d\log n)$ space.

We further obtain a streaming algorithm that returns a d-near-palindrome whose length is within an additive E-error of the longest d-near-palindrome. The algorithm uses $\mathcal{O}\left(\frac{dn\log^6 n}{E}\right)$ bits of space and $\mathcal{O}\left(\frac{dn\log^5 n}{E}\right)$ update time. As before, we show that any randomized streaming algorithm that solves the longest d-near-palindrome problem for additive error E with probability at least $1 - \frac{1}{n}$, uses $\Omega\left(\frac{dn}{E}\right)$ space.

Finally, we give an *exact* two-pass algorithm that solves the longest d-near-palindrome problem using $\mathcal{O}\left(d^2\sqrt{n}\log^6 n\right)$ bits of space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Longest palindrome with mismatches, Streaming algorithms, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.31

1 Introduction

A palindrome is a string that reads the same as its reverse, such as the common construct "racecar", or the deliberate construct "aibohphobia". Given a metric and an integer $d > 0$, we say that a string is a *d-near-palindrome* if it is at distance at most d from its reverse. In this paper, we study the problem of identifying the longest d-near-palindrome substring in the *streaming* model, under the Hamming distance. In the streaming model, the input data is streamed one symbol at a time, and we are allowed to perform computation using

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 31; pp. 31:1–31:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

only a small amount of working memory. Specifically, our goal is to approximate the length of the longest near-palindrome in a string of length n, using only $o(n)$ space. A related question regarding approximating the length of the longest palindrome in RNA sequences under removal of elements was explicitly asked at the Bertinoro Workshop on Sublinear Algorithms 2014 [1].

Finding near-palindromes is widely motivated in string processing of databases relevant to bioinformatics.

Specifically, since the development of the Human Genome Project, advances in biological algorithms have quickened the sequencing for genes and proteins, leading to increasingly large databases of strings representing both nucleic acids for DNA or RNA, and amino acids for proteins. Tools to analyze these sequences, such as the basic local alignment search tool (BLAST) [2] often require the removal of "low-complexity" regions (long repetitive or palindromic structures). However, these long sequences frequently contain small perturbations through mutation or some other form of corruption (including human error), so that identifying "near"-palindromes under either Hamming distance or edit distance is important for preprocessing sequences before applying the heuristic tools. In particular, the streaming model is relevant to contemporary data-sequencing technologies for near-palindromes, as further discussed in [9, 12].

Our contributions

We initiate the study of finding near-palindromes in the streaming model, and provide several algorithms for the longest near-palindrome substring.

Given a stream S of length n and an integer $d = o(\sqrt{n})$, let ℓ_{max} be the length of a longest d-near-palindrome substring in S.

▶ **Theorem 1.** *There exists a one-pass streaming algorithm that returns a d-near-palindrome of length at least $\frac{1}{1+\epsilon} \cdot \ell_{max}$, with probability $1 - \frac{1}{n}$. The algorithm uses $\mathcal{O}\left(\frac{d \log^7 n}{\epsilon \log(1+\epsilon)}\right)$ bits of space and update time $\mathcal{O}\left(\frac{d \log^6 n}{\epsilon \log(1+\epsilon)}\right)$ per arriving symbol.*

▶ **Theorem 2.** *There exists a one-pass streaming algorithm that returns a d-near-palindrome of length at least $\ell_{max} - E$, with probability $1 - \frac{1}{n}$. The algorithm uses $\mathcal{O}\left(\frac{dn \log^6 n}{E}\right)$ bits of space and update time $\mathcal{O}\left(\frac{dn \log^5 n}{E}\right)$ per arriving symbol.*

If two passes over the stream are allowed, one can find an *exact* longest d-near-palindrome.

▶ **Theorem 3.** *There exists a two-pass streaming algorithm that returns a d-near-palindrome of length ℓ_{max}, with probability $1 - \frac{1}{n}$. It uses $\mathcal{O}\left(d^2 \sqrt{n} \log^6 n\right)$ bits of space and $\mathcal{O}\left(d^2 \sqrt{n} \log^5 n\right)$ update time per arriving symbol.*

We complement our results with lower bounds for randomized algorithms.

▶ **Theorem 4.** *Let $d = o(\sqrt{n})$. Then any randomized streaming algorithm that returns a d-near-palindrome of length at least $\frac{\ell_{max}}{1+\epsilon}$ with probability at least $1 - \frac{1}{n}$ must use $\Omega\left(d \log n\right)$ bits of space.*

▶ **Theorem 5.** *Let $d = o(\sqrt{n})$ and $E > d$ be an integer. Then any randomized streaming algorithm that returns a d-near-palindrome of length at least $\ell - E$, with probability at least $1 - \frac{1}{n}$ must use $\Omega\left(\frac{dn}{E}\right)$ bits of space.*

A summary of our results and comparison with related work appears in Table 1.

Table 1 Summary of our results and comparison to related work

	Space for Algorithms		Lower Bounds	
Model	d-Near-Palindrome	Palindrome	d-Near-Palindrome	Palindrome
1-Pass, Multiplicative $(1+\epsilon)$	$\mathcal{O}\left(\frac{d\log^7 n}{\epsilon\log(1+\epsilon)}\right)$	$\mathcal{O}\left(\frac{\log^2 n}{\epsilon\log(1+\epsilon)}\right)$ [5]	$\Omega(d\log n)$	$\Omega\left(\frac{\log n}{\log(1+\epsilon)}\right)$ [10]
1-Pass, Additive E	$\mathcal{O}\left(\frac{dn\log^6 n}{E}\right)$	$\mathcal{O}\left(\frac{n\log n}{E}\right)$ [5]	$\Omega\left(\frac{dn}{E}\right)$	$\Omega\left(\frac{n}{E}\right)$ [5]
2-Pass, Exact	$\mathcal{O}\left(d^2\sqrt{n}\log^6 n\right)$	$\mathcal{O}\left(\sqrt{n}\log n\right)$ [5]	-	-

Background and Related Work

Our techniques extend previous work on the Longest Palindromic Substring Problem, the Pattern Matching Problem, and the d-Mismatch Problem in the streaming model.

In the *Longest Palindromic Substring Problem*, the goal is to output a longest palindromic substring of an input of length n, while minimizing the computation space. Manacher [14] introduces a linear-time online algorithm, which reports whether all symbols seen at the time of query form a palindrome. Berenbrink *et al.* [5] achieve $\mathcal{O}\left(\frac{\log^2 n}{\epsilon\log(1+\epsilon)}\right)$ space for multiplicative error $(1+\epsilon)$, and show a space lower bound for algorithms with additive error. Gawrychowski *et al.* [10] recently generalize the aforementioned lower bounds for additive error, and also produce a space lower bound of $\Omega\left(\frac{\log n}{\log(1+\epsilon)}\right)$ for algorithms with multiplicative error $(1+\epsilon)$, which is essentially tight.

In the *Pattern Matching Problem*, one is given a pattern of length m and the goal is to output all occurrences of the pattern in the input string, while again minimizing space or update time. In order to achieve space sublinear in the size of the input, many pattern matching streaming algorithms use Karp-Rabin fingerprints [13]. Porat and Porat [15] present a randomized algorithm for exact pattern matching using $\mathcal{O}(\log m)$ space and $\mathcal{O}(\log m)$ update time, which Breslauer and Galil [6] further improve to constant update time.

In the *d-Mismatch Problem*, one is given a pattern and the goal is to find all substrings of the input that are at most Hamming distance d from the pattern. A line of exciting work (e.g., [3, 15, 7, 4]) culminates in a recent algorithm by Clifford *et al.* [8] that uses $\mathcal{O}\left(d^2\mathsf{polylog}\, m\right)$ space and $\mathcal{O}\left(\sqrt{d}\log d + \mathsf{polylog}\, m\right)$ update time per arriving symbol.

2 Preliminaries

We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. We assume an input stream of length n over alphabet Σ. Given a string $S[1, n]$, we denote its length by $|S|$, its i^{th} character by $S[i]$, and the substring between locations i and j (inclusive) by $S[i, j]$.

The Hamming distance between S and T, denoted $\mathsf{HAM}(S, T)$ is the number of indices whose symbols do not match: $\mathsf{HAM}(S, T) = \left|\{i \mid S[i] \neq T[i]\}\right|$. We denote the concatenation of S and T by $S \circ T$. Each index i such that $S[i] \neq S[n - i + 1]$ is a *mismatch*. We say S is a *d-near-palindrome* if $\mathsf{HAM}(S, S^R) \leq d$. Without loss of generality, our algorithms assume the lengths of d-near-palindromes are even, since for any odd length d-near-palindrome, we may apply the algorithm to $S[1]S[1]S[2]S[2]\cdots S[n]S[n]$ instead of $S[1, n]$.

▶ **Definition 6** (Karp-Rabin Fingerprint)**.** For a string S, a prime P and an integer B with $1 \leq B < P$, the Karp-Rabin forward and reverse fingerprints [13] are defined as follows:

$$\phi^F(S) = \left(\sum_{x=1}^{|S|} S[x]\cdot B^x\right) \bmod P, \qquad \phi^R(S) = \left(\sum_{x=1}^{|S|} S[x]\cdot B^{-x}\right) \bmod P.$$

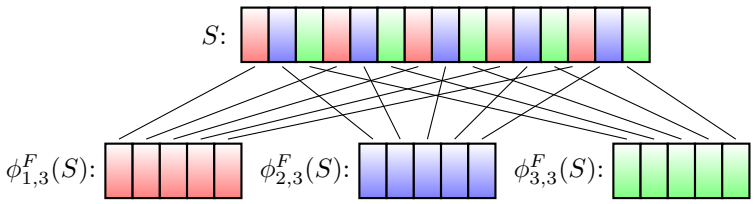

Figure 1 Karp-Rabin Fingerprints for first-level subpattern.

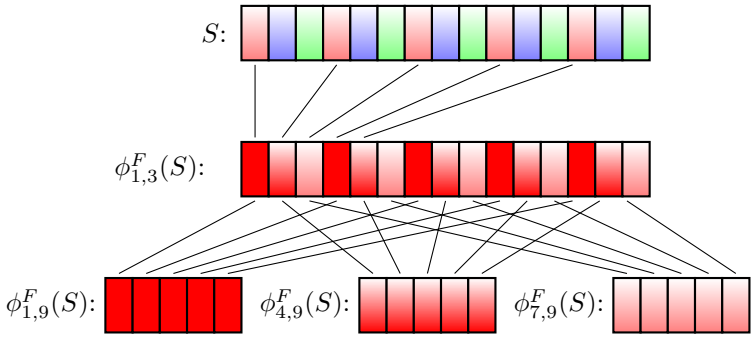

Figure 2 Karp-Rabin Fingerprints for second-level subpattern.

Karp-Rabin Fingerprints have the following easily verifiable properties:

1. $\phi^R(S) \cdot B^{|S|+1} = \phi(S^R) \bmod P$
2. $\phi^F(S[x,y]) = B^{1-x}(\phi^F(S[1,y]) - \phi^F(S[1,x-1])) \bmod P$
3. $\phi^R(S[x,y]) = B^{x-1}(\phi^R(S[1,y]) - \phi^R(S[1,x-1])) \bmod P$

We use Karp-Rabin Fingerprints for certain subpatterns of S, as in [8]. For a string S and integers $a \le b$, define the *first-level subpattern* $S_{a,b}$ to be the subsequence $S[a]S[a+b]S[a+2b]\ldots$. In this case, define $S^R_{a,b} = (S_{a,b})^R$ (as opposed to $(S^R)_{a,b}$). Similarly, define $S_{a,b}[x,y] = S_{a,b} \cap S[x,y]$ (as opposed to $(S[x,y])_{a,b}$). Then for $1 \le a \le b$, define the fingerprints for $S_{a,b}$ and its reverse:

$$\phi^F_{a,b}(S) = \phi^F(S_{a,b}) = \left(\sum_{x \equiv a \bmod b} S[x] \cdot B^{\lceil x/b \rceil} \right) \bmod P$$

$$\phi^R_{a,b}(S) = \phi^R(S_{a,b}) = \left(\sum_{x \equiv a \bmod b} S[x] \cdot B^{-\lceil x/b \rceil} \right) \bmod P$$

For an example, see Figure 1.

Given a first-level subpattern $T = S_{a,b} = S[a]S[a+b]S[a+2b]\ldots$ and integers $r \le s$, define the *second-level subpattern* $T_{r,s} = T[r]T[r+s]T[r+2s]\ldots$. Observe that $T_{r,s} = S_{a+rb,sb}$ and thus, second-level subpatterns are simply more refined first-level subpatterns. For an example, see Figure 2.

Observe the following properties of fingerprints on first-level and second-level subpatterns:

1. $\phi^R_{a,b}(S) \cdot B^{|S|+1} = \phi_{|S|-a+1,b}(S^R) \bmod P$
2. $\phi^F_{a,b}(S[x,y]) = B^{\lceil (1-x)/b \rceil}(\phi^F_{a,b}(S[1,y]) - \phi^F_{a,b}(S[1,x-1])) \bmod P$
3. $\phi^R_{a,b}(S[x,y]) = B^{\lceil (x-1)/b \rceil}(\phi^R_{a,b}(S[1,y]) - \phi^R_{a,b}(S[1,x-1])) \bmod P$

We also use the following application of the Prime Number Theorem:

▶ **Lemma 7.** *(Adaptation of Lemma 4.1 [8]) Given two distinct integers $a, b \in [1,n]$ and a random prime number $p \in \left[\frac{d}{\beta}\log^2 n, \frac{34d}{\beta}\log^2 n\right]$ where $\beta = \frac{1}{16}$, then $\mathbf{Pr}\left[a \equiv b \bmod p\right] \le \frac{\beta}{32d}$.*

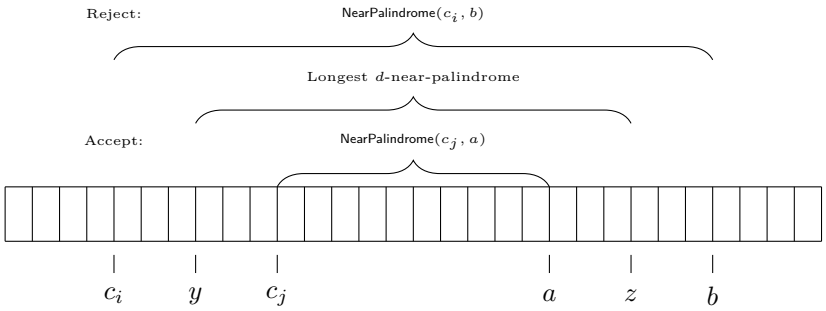

Figure 3 The longest d-near-palindrome will be sandwiched within checkpoints to provide a $(1+\epsilon)$-approximation of ℓ_{max}. That is, $(1+\epsilon)(a-c_j) \geq (z-y)$.

Finally, we remark that problems in bioinformatics, such as the *RNA Folding Problem* [1], use the following notion of complementary palindromes:

▶ **Definition 8.** Let $f : \sum \to \sum$ be a pairing of symbols in the alphabet. A string $S \in \sum^n$ is a complementary palindrome if $S[x] = f(S[n+1-x])$ for all $1 \leq x \leq n$.

Our algorithms can be modified to recognize complementary palindromes with the same space usage and update time. Indeed, we only need to modify the forward fingerprints to use $f(S[x])$ instead of $S[x]$:

$$\phi^F_{a,b}(S) = \left(\sum_{x \equiv a \bmod b} f(S[x]) \cdot B^{\lceil x/b \rceil} \right) \bmod P.$$

3 Overview and Techniques

One-pass Multiplicative Approximation Algorithm

Our algorithm combines and extends ideas and techniques from the solution to the d-Mismatch Problem in [8] and the solution to the Longest Palindrome Problem in [5]. For additional clarity, we provide several figures in [11].

As the stream progresses, we keep a set of checkpoints $\mathcal{C}$, where $c \in \mathcal{C}$ is the beginning of potential d-near-palindromes that we output. We also maintain a sliding window that contains the $2d$ most recently seen symbols. The sliding window identifies any d-near-palindrome of length at most $2d$. It also guesses that the midpoint of the sliding window is the midpoint of a potential d-near-palindrome of length $> 2d$.

We keep an estimate $\tilde{\ell}$ of the length ℓ_{max} of the longest d-near-palindrome seen throughout the stream, as well as the starting index c_{start} of the d-near-palindrome, and the locations of the mismatches, a set of size at most d. Upon reading symbol $S[x]$ of the stream, we call procedure NearPalindrome to see if $S[c_i, x]$ is a d-near-palindrome, for each checkpoint c_i such that $x - c_i > \tilde{\ell}$. Using the framework of [5], we create and update checkpoints throughout the stream so that we find a d-near-palindrome of length at least $\frac{\ell_{max}}{1+\epsilon}$, as in Figure 3.

The algorithm in [5] maintains a list of potential midpoints associated with each checkpoint. This list can be linear in size, however it satisfies nice structural results that can be used to succinctly represent the list of candidate midpoints. Directly adapting these structural results to our setting would incur an extra factor of d in our space complexity. We avoid this extra factor by circumventing the list of candidate midpoints in the one-pass algorithms altogether.

We now overview the procedure NearPalindrome that we use repeatedly in our algorithms. It returns whether $S[c_i, x]$ is a d-near-palindrome, and if so it returns the mismatches.

The procedure NearPalindrome follows along the lines of the d-mismatch streaming algorithm from [8]. Recall that in the d-Mismatch Problem, we are given a pattern R and a text S and the algorithm is required to output a string $S[x - |R| + 1, x]$ such that $\mathsf{HAM}(R, S[x - |R| + 1, x]) \leq d$. While in the d-Mismatch Problem the pattern is fixed, here we essentially use a variable-length pattern. Namely, we check if for checkpoint c_i it is the case that $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) \leq d$. To achieve this, we maintain a dynamic set of fingerprints that we compare against the dynamic set of text fingerprints.

The procedure has two stages. In the first stage it eliminates strings T with $\mathsf{HAM}(T, T^R) \geq 2d$, while in the second stage it eliminates strings with $d < \mathsf{HAM}(T, T^R) < 2d$. This can be achieved by estimating the distance between T and T^R using fingerprints of equivalence classes modulo different primes.

Picking random primes should distribute the mismatches into different equivalence classes. The procedure estimates the number of mismatches by comparing the fingerprints of the substrings whose indices are in the same congruence class modulo p with the reverse fingerprints, namely $T_{r,p}$ and $T^R_{r,p}$ for all $1 \leq r \leq p$. Denote by $T_{r,p}$ and $T^R_{r,p}$ the *first-level fingerprints*.

By the second stage we are only left with the strings with a small number of mismatches. In order to recover the mismatches, one needs to refine each subpattern $\tilde{T} = T_{r,p}$ by picking smaller primes p', and comparing the fingerprints of the strings $\tilde{T}_{r',p'}$ and $\tilde{T}^R_{r',p'}$ for all $1 \leq r' \leq p'$. Denote by $\tilde{T}_{r',p'}$ and $\tilde{T}^R_{r',p'}$ the *second-level fingerprints* (see Figure 2).

In the first stage, we sample $2 \log n$ primes uniformly at random from $\left[\frac{d}{\beta} \log^2 n, \frac{34d}{\beta} \log^2 n\right]$, where $\beta = 1/16$. Each prime generates p subpatterns containing positions in the same equivalence class (mod p). Therefore, there are $\mathcal{O}\left(d \log^3 n\right)$ first-level subpatterns. In the second stage, we take all primes in $[\log n, 3 \log n]$ that together with the primes picked in the first stage generate a total of $\mathcal{O}\left(d \log^5 n\right)$ second-level subpatterns.

Finally, we assume throughout the paper that the fingerprints of any subpattern do not fail. Since there are at most n^3 subpatterns, and the probability that a particular fingerprint fails is at most $\frac{1}{n^5}$ for $P \in [n^5, n^6]$ (by Theorem 1 in [6]), then by a union bound, the probability that no fingerprint fails is least $1 - \frac{1}{n^2}$.

Our choice of parameters is more space-efficient compared to the data structure given by [8], which uses $\mathcal{O}\left(d^2 \log^7 n\right)$ space, since we no longer need the sliding functionality provided by their data structure. Also, the data structure given by [16] does not suffice, as it does not support concatenation, which is needed for maintaining the checkpoints.

One-pass Additive Approximation and Two-Pass Exact Algorithms

To obtain the one-pass additive approximation, we modify our checkpoints, so that they appear in every $\left\lfloor \frac{E}{2} \right\rfloor$ positions. Hence, the longest d-near-palindrome must have some checkpoint within $\left\lfloor \frac{E}{2} \right\rfloor$ positions of it, and the algorithm will recover a d-near-palindrome with length at least $\ell_{max} - E$.

To obtain the two-pass *exact* algorithm, we set $E = \sqrt{n}$ and modify the additive error algorithm so that it returns a list $\mathcal{L}$ of candidate midpoints of d-near-palindromes. Moreover, we show a structural result in Lemma 16, which allows us to compress certain substrings in the first pass, so that the second pass can recover mismatches for any potential d-near-palindromes within these substrings.

In the second pass, we carefully keep track of the $\frac{\sqrt{n}}{2}$ characters before the starting positions of long d-near-palindromes identified in the first pass. We use the compressed

information from the first pass to reconstruct the fingerprints and calculate the number of mismatches within these long d-near-palindromes identified in the first pass. However, the actual d-near-palindromes may extend beyond the estimate returned in the first pass. Thus, we compare the $\frac{\sqrt{n}}{2}$ characters after the d-near-palindromes identified in the first pass with the $\frac{\sqrt{n}}{2}$ characters that we track. This allows us to exactly identify the longest d-near-palindrome during the second pass.

Lower Bounds

To show lower bounds for randomized algorithms solving the d-near palindrom problem we use Yao's Principle [17], and construct distributions for which any deterministic algorithm fails with significant probability unless given a certain amount of space. We first reduce the problem of approximating longest d-near-palindromes to the problem of exactly identifying whether two strings have Hamming distance at most d. We then carefully construct hard distributions, and show via counting arguments that deterministic algorithms using a little of space will fail with significant probability on inputs from these distributions. We also use some ideas from [10] who showed lower bounds for streaming palindromes. Due to space constraints, the proofs of Theorem 4 and Theorem 5 are detailed in [11].

4 One-Pass Streaming Algorithm with Multiplicative Error $(1+\epsilon)$

In this section, we prove Theorem 1. Namely, we provide a one-pass streaming algorithm with multiplicative error $(1+\epsilon)$, using space $\mathcal{O}\left(\frac{d\log^7 n}{\epsilon\log(1+\epsilon)}\right)$ bits of space.

4.1 Algorithm

As described in the overview, similar to [5], we maintain a sliding window of size $2d$, along with master fingerprints, and a series of checkpoints. From the sliding window, we observe any d-near-palindrome with length at most $2d$, as well as any candidate midpoints. Then prior to seeing element $S[x]$ in the stream, we keep the following in memory:

Initialization:

1. Pick a prime P from $[n^5, n^6]$ and an integer $B < P$ (the modulo and the base of the Karp-Rabin fingerprints, respectively).
2. For the first-level fingerprints, create a set $\mathcal{P}$ consisting of $2\log n$ primes $p_1, p_2, \ldots, p_{2\log n}$ sampled independently and uniformly at random from $\left[\frac{d}{\beta}\log^2 n, \frac{34d}{\beta}\log^2 n\right]$, where $\beta = \frac{1}{16}$.
3. For the second-level fingerprints, let $\mathcal{Q}$ be the set of all primes in $[\log n, 3\log n]$.
4. Initialize a sliding window of size $2d$.
5. Initialize the sets of *Master Fingerprints*, $\mathcal{F}^F$ and $\mathcal{F}^R$:
   a. Set $\phi^F_{r,p}(S) = 0$, $\phi^R_{r,p}(S) = 0$ for all $p \in \mathcal{P}$ and $1 \le r \le p$.
   b. Set $\phi^F_{r',pq}(S) = 0$, $\phi^R_{r',pq}(S) = 0$ for all $p \in \mathcal{P}$, $q \in \mathcal{Q}$ and $1 \le r' \le pq$.
   c. Let $\mathcal{F}^F$ be the set of all $\phi^F(S)$.
   d. Let $\mathcal{F}^R$ be the set of all $\phi^R(S)$.
6. Set $k_0 = \frac{\log(1/\alpha)}{\log(1+\alpha)}$, where $\alpha = \sqrt{1+\epsilon} - 1$.
7. Initialize a list of checkpoints $\mathcal{C} = \emptyset$.
8. Set the starting index c_{start} to be 1, the length estimate $\tilde{\ell}$ of the longest d-near-palindrome found so far to be 0, and the at most d mismatched indices $\mathcal{M} = \emptyset$.

We now formalize the steps outlined in the overview. The data structure relies on the procedure NearPalindrome that we describe and analyze in detail in Section 4.2.

Maintenance:

1. Read $S[x]$. Update the sliding window to $S[x-2d, x]$.
2. Update the *Master Fingerprints* to be $\mathcal{F}^F(1,x)$ and $\mathcal{F}^R(1,x)$:
   a. Update the first-level fingerprints: for every $p \in \mathcal{P}$, let $r \equiv x \mod p$, and increment $\phi^F_{r,p}(S)$ by $S[x] \cdot B^{\lceil x/p \rceil} \mod P$ and increment $\phi^R_{r,p}(S)$ by $S[x] \cdot B^{-\lceil x/p \rceil} \mod P$.
   b. Update the second-level fingerprints: for every $p \in \mathcal{P}$ and $q \in \mathcal{Q}$, let $r' \equiv x \mod pq$, and increment $\phi^F_{r',pq}(S)$ by $S[x] \cdot B^{\lceil x/(pq) \rceil} \mod P$ and increment $\phi^R_{r',pq}(S)$ by $S[x] \cdot B^{-\lceil x/(pq) \rceil} \mod P$.
3. For all $k \geq k_0$:
   a. If x is a multiple of $\lfloor \alpha(1+\alpha)^{k-2} \rfloor$, then add the checkpoint $c = x$ to $\mathcal{C}$. Set $level\,(c) = k$, $fingerprints\,(c) = \mathcal{F}^F(1,x) \cup \mathcal{F}^R(1,x)$.
   b. If there exists a checkpoint c with $level\,(c) = k$ and $c < x - 2(1+\alpha)^k$, then delete c from $\mathcal{C}$.
4. For every checkpoint $c \in \mathcal{C}$ such that $x - c > \tilde{\ell}$, we call NearPalindrome (described in Section 4.2) to see if $S[c,x]$ is a d-near-palindrome. If $S[c,x]$ is a d-near-palindrome, then set $c_{start} = c$, $\tilde{\ell} = x - c$ and $\mathcal{M}$ to be the indices returned by NearPalindrome.
5. If $x = n$, then report c_{start}, $\tilde{\ell}$, and $\mathcal{M}$.

4.2 NearPalindrome and its analysis

In this section, we describe and analyze the randomized procedure NearPalindrome that receives as input a string, and decides whether it is a d-near-palindrome or not. Moreover, if the string is a d-near-palindrome, NearPalindrome returns the locations of the mismatched indices. As mentined, NearPalindrome adapts ideas from [8] for solving the d-mismatch problem. Our proofs of the properties of NearPalindrome follow almost verbatim from the statements in [8], with the only difference being that we make the magnitudes of the chosen primes as large as to withstand patterns of length $\mathcal{O}(n)$. For completeness, we include all the proofs in [11].

Before formally describing NearPalindrome we introduce some notation. Given a string $S[x,y]$, and prime p_j let $\Delta_j(x,y)$ be the number of $r \in [p_j]$ such that the subpatterns $S_{r,p_j}[x,y]$ and $S^R_{r,p_j}[x,y]$ are different. Note that we can compute $\Delta_j(x,y)$ from the fingerprints $\mathcal{F}^F(x,y)$ and $\mathcal{F}^R(x,y)$ as the number of indices r such that $\phi^F_{r,p_j}[x,y] \neq B^{k+1} \cdot \phi^R_{r,p_j}[x,y] \mod P$, where k is the length of $S_{r,p_j}[x,y]$. Define $\Delta(x,y) = \max_j \Delta_j(x,y)$. We may assume throughout that $S[x,y]$ has even length. Next we summarize some useful properties of $\Delta(x,y)$.

▶ **Lemma 9.** *(Adaptation of Lemma 5.1 and Lemma 5.2 [8]) Let $\beta = 1/16$.*

1. *If $\mathsf{HAM}(S[x,y], S^R[x,y]) \leq d$, then $\Delta(x,y) \leq d$.*
2. *If $\mathsf{HAM}(S[x,y], S^R[x,y]) \geq 2d$, then $\Delta(x,y) > (1+\beta) \cdot d$ with probability at least $1 - \frac{1}{n^3}$.*

A position $i \in [x,y]$ is an *isolated mismatch* under p_j if there exists some $r \leq p_j$ for which the subpatterns $S_{r,p_j}[x,y]$ and $S^R_{r,p_j}[x,y]$ differ only in position i. Let $\mathcal{I}_j(x,y)$ be the number of isolated mismatches in $S[x,y]$ under p_j, and let $\mathcal{I}(x,y)$ be the union of $\mathcal{I}_j(x,y)$, over all primes p_j. The next lemma shows that if $\mathsf{HAM}(S[x,y], S^R[x,y]) \leq 2d$, then $\mathcal{I}(x,y)$ is precisely $\mathsf{HAM}(S[x,y], S^R[x,y])$ with high probability over the set of primes.

▶ **Lemma 10.** *(Adaptation of Lemma 4.2 [8]) If $\mathsf{HAM}(S[x,y], S^R[x,y]) \leq 2d$, then $\mathsf{HAM}(S[x,y], S^R[x,y]) = \mathcal{I}(x,y)$ with probability at least $1 - \frac{1}{n^7}$.*

▶ **Lemma 11.** *The set of mismatches can be identified using the second-level fingerprints.*

We are now ready to present the algorithm in full.

NearPalindrome(c_i, x): (determines if $S[c_i, x]$ is a d-near-palindrome)
1. For each $j \in [2\log n]$, initialize $\Delta_j = 0$.
2. For each $j \in [2\log n]$ and $r \in [p_j]$:
   If $\phi^F_{r,p_j}(S[c_i, x]) \neq B^{k+1} \cdot \phi^R_{r,p_j}(S[c_i, x]) \bmod P$, where k is the length of $S_{r,p_j}[c_i, x]$, then increment $\Delta_j(c_i, x) = \Delta_j(c_i, x) + 1$.
3. Let $\Delta(c_i, x) = \max_j\{\Delta_j(c_i, x)\}$.
4. If $\Delta(c_i, x) > (1 + \beta) \cdot d$, then we immediately reject $S[c_i, x]$. (Recall that $\beta = \frac{1}{16}$.)
5. Initialize $\mathcal{I} = \emptyset$.
6. For each mismatch in $S[c_i, x]$, if there exists $q \in \mathcal{Q}$ and such that $\phi^F_{r',q}(S_{r,p}[c_i, x]) \neq B^{k'+1} \cdot \phi^R_{r',q}(S_{r,p}[c_i, x]) \bmod P$, where k' is the length of $S_{r'+rp,pq}[c_i, x]$, for exactly one $r \in [p], r' \in [q]$, then insert the mismatch into $\mathcal{I}(c_i, x)$. (This is the set of *isolated mismatches*.)
7. If $|\mathcal{I}(c_i, x)| > d$, then we reject $S[c_i, x]$.
8. Else, if $|\mathcal{I}(c_i, x)| \leq d$, then we accept $S[c_i, x]$ and return $\mathcal{I}(c_i, x)$.

▶ **Theorem 12.** *With probability at least* $1 - \frac{1}{n^3}$*, procedure* NearPalindrome *returns whether* $S[c_i, x]$ *is a d-near-palindrome.*

Proof of Theorem 12. If $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) > 2d$, then by Lemma 9, $\Delta(c_i, x) > (1 + \beta) \cdot 2d$ with probability at least $1 - \frac{1}{n^3}$ and so NearPalindrome will reject $S[c_i, x]$. Conditioned on $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) \leq 2d$, by Lemma 10 $\mathcal{I}(c_i, x) = \mathsf{HAM}(S[c_i, x], S^R[c_i, x])$ with probability at least $1 - \frac{1}{n^5}$, and so if $\mathsf{HAM}(S[c_i, x], S^R[c_i, x]) > d$ the algorithm safely rejects, and otherwise it accepts. Finally, by Lemma 11 the entire set of mismatches $\mathcal{I}(c_i, x)$ can be computed from the second-level subpattern fingerprints. ◀

4.3 Correctness and Space Complexity

In this section, we finish the proof of Theorem 1 by claiming correctness and analyzing the space used by the one-pass streaming algorithm described in Section 4.1. Since we used the spacing of the checkpoints as in [5], we have the following properties.

▶ **Observation 13.** *([5], Observation 16, Lemma 17) At reading* $S[x]$*, for all* $k \geq k_0 = \left\lceil \frac{\log\left(\frac{(1+\alpha)^2}{\alpha}\right)}{\log(1+\alpha)} \right\rceil$*, let* $C_{x,k} = \{c \in \mathcal{C} \mid level\,(c) = k\}$.
1. $C_{x,k} \subseteq [x - 2(1 + \alpha)^k, x]$.
2. *The distance between two consecutive checkpoints of* $C_{x,k}$ *is* $\lfloor \alpha(1 + \alpha)^{k-2} \rfloor$.
3. $|C_{x,k}| = \left\lceil \frac{2(1+\alpha)^k}{\lfloor \alpha(1+\alpha)^{k-2} \rfloor} \right\rceil$.
4. *At any point in the algorithm, the number of checkpoints is* $\mathcal{O}\left(\frac{\log n}{\epsilon \log(1+\epsilon)}\right)$.

▶ **Corollary 14.** *The total space used by the algorithm is* $\mathcal{O}\left(\frac{d \log^7 n}{\epsilon \log(1+\epsilon)}\right)$ *bits. The update time per arriving symbol is also* $\mathcal{O}\left(\frac{d \log^6 n}{\epsilon \log(1+\epsilon)}\right)$.

Proof. The first-level and second-level Karp-Rabin fingerprints consist of integers modulo P for each of the $\mathcal{O}\left(d \log^5 n\right)$ subpatterns. Since $P \in [n^5, n^6]$, then $\mathcal{O}\left(d \log^6 n\right)$ bits of

space are necessary for each fingerprint. Furthermore, by Observation 13, there are $\frac{\log n}{\epsilon \log(1+\epsilon)}$ checkpoints, so the total space used is $\mathcal{O}\left(d \log^7 n\right)$ bits. For each arriving symbol $S[x]$, the algorithm checks possibly the fingerprints of each checkpoint whether the substring is a d-near-palindrome. There are $\mathcal{O}\left(\frac{\log n}{\epsilon \log(1+\epsilon)}\right)$ checkpoints, each with fingerprints of size $\mathcal{O}\left(d \log^5 n\right)$. Each subpattern of a fingerprint may be compared in constant time, so the overall update time is $\mathcal{O}\left(\frac{d \log^6 n}{\epsilon \log(1+\epsilon)}\right)$. ◀

Proof of Theorem 1. Let ℓ_{max} be the length of the longest d-near-palindrome, $S[x, x + \ell_{max} - 1]$, with midpoint m. Let k be the largest integer so that $2(1+\alpha)^{k-1} < \ell_{max}$, where $\alpha = \sqrt{1+\epsilon} - 1$. Let $y = m + (1+\alpha)^{k-1}$ so that $x < y < x + \ell_{max} - 1$. By Observation 13, there exists a checkpoint in the interval $[y - 2(1+\alpha)^{k-1}, y]$. Furthermore, Observation 13 implies consecutive checkpoints of level $k-1$ are separated by distance $\lfloor \alpha(1+\alpha)^{k-2} \rfloor$. Thus, there exists a checkpoint c in the interval $\left[y - 2(1+\alpha)^{k-1}, y - 2(1+\alpha)^{k-1} + \alpha(1+\alpha)^{k-3}\right]$. If procedure NearPalindrome succeeds for this checkpoint on position $m + (m - c)$, then the output $\tilde{\ell}$ of the algorithm is at least

$$2(m-c) \geq 2m - 2y + 4(1+\alpha)^{k-1} - 2\alpha(1+\alpha)^{k-3} = 2(1+\alpha)^{k-1} - 2\alpha(1+\alpha)^{k-3}.$$

Comparing this output with ℓ_{max},

$$\frac{\ell_{max}}{\tilde{\ell}} \leq \frac{2(1+\alpha)^k}{2(1+\alpha)^{k-1} - 2\alpha(1+\alpha)^{k-3}} = \frac{(1+\alpha)^3}{(1+\alpha)^2 - \alpha} \leq (1+\alpha)^2 = 1 + \epsilon.$$

Thus, if procedure NearPalindrome succeeds for all substrings then $\tilde{\ell} \leq \ell_{max} \leq (1+\epsilon)\tilde{\ell}$. Taking Theorem 12 and a simple union bound over all $\mathcal{O}\left(n^2\right)$ possible substrings, procedure NearPalindrome succeeds for all substrings with probability at least $1/n$, and the result follows. ◀

5 Two-Pass *Exact* Streaming Algorithm

In this section, we prove Theorem 3. Namely, we present a two-pass streaming algorithm which returns the longest d-near-palindrome with space $\mathcal{O}\left(d^2 \sqrt{n} \log^6 n\right)$.

Recall that we assume the lengths of d-near-palindromes are even. Thus, for any substring $S[x, y]$ of even length, we define its *midpoint* $m = \left\lfloor \frac{x+y}{2} \right\rfloor$. Upon reading x, we say that $x - \sqrt{n}$ is a candidate midpoint if the sliding window $S[x - 2\sqrt{n}, x]$ is a d-near-palindrome.

First, we modify the one-pass streaming algorithm with additive error in Section 4 so that it returns a list $\mathcal{L}$ of candidate midpoints of d-near-palindromes with length at least $\ell - \frac{\sqrt{n}}{2}$, where ℓ is an estimate of the maximum length output by the algorithm. However, we show in Lemma 16 that the string has a periodic structure which allows us to keep only $\mathcal{O}(d)$ fingerprints in order to recover the fingerprint for any substring between two midpoints.

In the second pass, we explicitly keep the $\frac{\sqrt{n}}{2}$ characters before the starting positions and candidate midpoints of long d-near-palindromes identified in the first pass. We use a procedure Recover to exactly identify the number and locations of mismatches within the d-near-palindromes identified in the first pass. We then use the $\frac{\sqrt{n}}{2}$ characters to extend the near-palindromes until the number of mismatches exceeds $d + 1$.

For an example, see Figure 4.

We first describe a structural property of a series of overlapping d-near-palindromes, showing that they are "almost" periodic.

▶ **Definition 15.** A string S is said to have period π if $S[j] = S[j+\pi]$ for all $j = 1, \ldots, |S'| - \pi$.

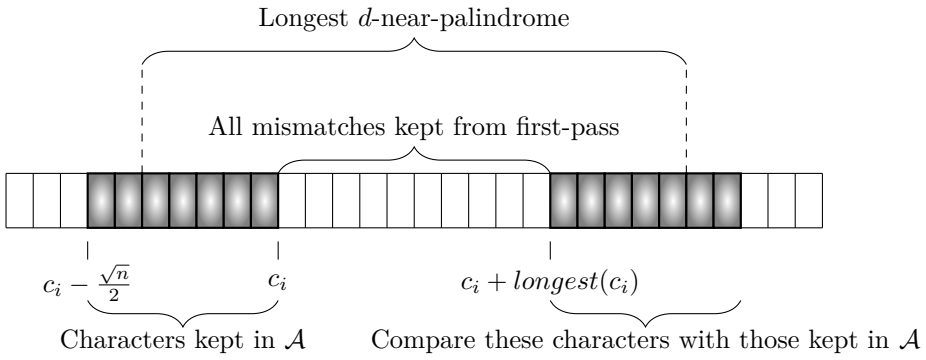

Figure 4 The second pass allows us to find the longest d-near-palindrome by explicitly comparing characters.

The following structural result is a generalization of a structural result about palindromes from [5].

▶ **Lemma 16.** *Let $m_1 < m_2 < \ldots < m_h$ be indices in S that are consecutive midpoints of d-near-palindromes of length ℓ^*, for some integer $\ell^* > 0$. If $m_h - m_1 \leq \ell^*$, then*

1. *$m_1, m_2, \ldots, m_h$ are equally spaced in S', so that $|m_2 - m_1| = |m_i - m_{i+1}|$ for all $i \in [h-1]$.*
2. *There exists string E_h with at most d nonzero entries such that $E_h + S[m_1 + 1, m_h]$ is a prefix of $ww^Rww^R\ldots$ of length at least ℓ^*, for some string w of length $|w| = m_2 - m_1$.*

In the first pass, we specify that the algorithm has sliding window size $2\sqrt{n}$. Thus, if the longest d-near-palindrome has length less than $2\sqrt{n}$, the algorithm can identify it. Otherwise, if the longest d-near-palindrome has length at least $2\sqrt{n}$, then the algorithm finds at most $\frac{\sqrt{n}}{2}$ non-overlapping d-near-palindromes of length at least $\ell - \sqrt{n}$. Hence, $\mathcal{O}\left(d^2\sqrt{n}\log^6 n\right)$ is enough space to store the fingerprints for the substrings between any two candidate midpoints, as well as between checkpoints $s_i \in \mathcal{L}$ and midpoints. This concludes the first pass of the algorithm. Details appear in the [11].

Before we proceed to the second pass, we describe procedure Recover(m_i, m_j, L_c) which either outputs that $S[m_i, m_j]$ is not a d-near-palindrome, or returns the number of mismatches, as well as their indices. The procedure crucially relies on structural result from Lemma 16 to reconstruct the fingerprints of $S[m_i, m_j]$ from fingerprints stored by the first pass. From the reconstructed fingerprints, the subroutine can then determine whether $S[m_i, m_j]$ is a d-near-palindrome, and identify the location of the mismatches, if necessary. The details of procedure Recover appear in [11].

Before the second pass, we first prune the list of checkpoints $\mathcal{C}$ to greedily include only those who are the starting indices for d-near-palindromes of length at least $\tilde{\ell} - \frac{\sqrt{n}}{2}$ and do not overlap with other d-near-palindromes already included in the list. In the second pass, the algorithm keeps track of the $\frac{\sqrt{n}}{2}$ characters before c, for each starting index $c \in \mathcal{C}$. We call procedure Recover to fully recover the mismatches in a region following c. After reading the last symbol in the region, we compare each subsequent symbol with the corresponding symbol before c, counting the total number of mismatches. When the total number of mismatches reaches $d + 1$ after seeing character $S[c + k + j + 1]$, where k is the size of the region, then the previous symbol is the end of the near-palindrome. Hence, the near-palindrome is $S[c - j, c + k + j]$, and if $k + 2j > \tilde{l}$, then we update the information for $\tilde{\ell}$ accordingly. For an example, see Figure 5.

We defer the details of the 2nd pass and the remaining proofs to [11].

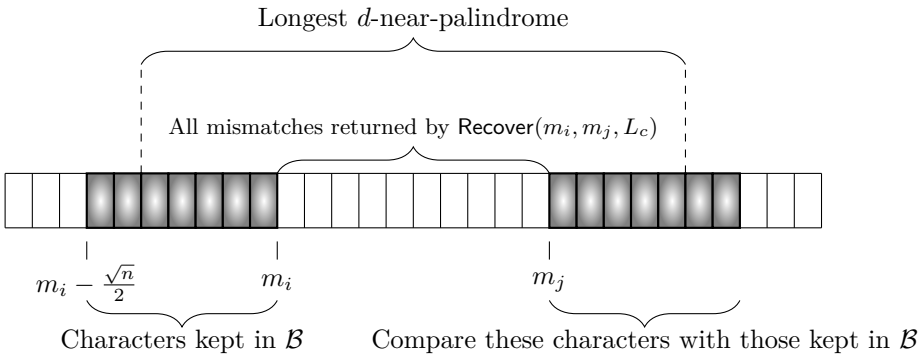

Figure 5 The second pass allows us to find the longest d-near-palindrome by explicitly comparing characters.

Acknowledgments. We would like to thank Funda Ergün, Tatiana Kuznetsova, and Qin Zhang for helpful discussions and pointers.

References

1 List of open problems in sublinear algorithms: Problem 61. URL: `http://sublinear.info/61`.

2 Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

3 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

4 Alexandr Andoni, Assaf Goldberger, Andrew McGregor, and Ely Porat. Homomorphic fingerprints under misalignments: sketching edit and shift distances. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 931–940, 2013.

5 Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome recognition in the streaming model. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 149–161, 2014.

6 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014.

7 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. *Inf. Comput.*, 209(4):731–736, 2011.

8 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2039–2052, 2016.

9 Albert A. Conti, Tom Van Court, and Martin C. Herbordt. Processing repetitive sequence structures with mismatches at streaming rate. In *Field Programmable Logic and Application, 14th International Conference , FPL Proceedings*, pages 1080–1083, 2004.

10 Pawel Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemyslaw Uznanski. Tight tradeoffs for real-time approximation of longest palindromes in streams. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM*, pages 18:1–18:13, 2016.

11 Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming for aibohphobes: Longest palindrome with mismatches. *CoRR*, abs/1705.01887, 2017. URL: `http://arxiv.org/abs/1705.01887`.

12 Martin C. Herbordt, Josh Model, Bharat Sukhwani, Yongfeng Gu, and Tom Van Court. Single pass streaming BLAST on fpgas. *Parallel Computing*, 33(10-11):741–756, 2007.

13 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

14 Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975.

15 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 315–323, 2009.

16 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM Proceedings*, pages 173–182, 2007.

17 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 222–227, 1977.