L p Pattern Matching in a Stream

Tatiana Starikovskaya, Michal Svagerka, Przemyslaw Uznański

L_p Pattern Matching in a Stream

Tatiana Starikovskaya
DIENS, École normale supérieure, PSL Research University, Paris, France
tat.starikovskaya@gmail.com

Michal Svagerka
ETH Zürich, Switzerland
michal.svagerka@alumni.ethz.ch

Przemysław Uznański
Institute of Computer Science, University of Wrocław, Poland
puznanski@cs.uni.wroc.pl

Abstract

We consider the problem of computing distance between a pattern of length n and all n-length subwords of a text in the streaming model.

In the streaming setting, only the Hamming distance (L_0) has been studied. It is known that computing the exact Hamming distance between a pattern and a streaming text requires $\Omega(n)$ space (folklore). Therefore, to develop sublinear-space solutions, one must relax their requirements. One possibility to do so is to compute only the distances bounded by a threshold k, see [SODA'19, Clifford, Kociumaka, Porat] and references therein. The motivation for this variant of this problem is that we are interested in subwords of the text that are similar to the pattern, i.e. in subwords such that the distance between them and the pattern is relatively small.

On the other hand, the main application of the streaming setting is processing large-scale data, such as biological data. Recent advances in hardware technology allow generating such data at a very high speed, but unfortunately, the produced data may contain about 10% of noise [Biol. Direct.'07, Klebanov and Yakovlev]. To analyse such data, it is not sufficient to consider small distances only. A possible workaround for this issue is the $(1 \pm \varepsilon)$-approximation. This line of research was initiated in [ICALP'16, Clifford and Starikovskaya] who gave a $(1 \pm \varepsilon)$-approximation algorithm with space $\widetilde{\mathcal{O}}(\varepsilon^{-5}\sqrt{n})$.

In this work, we show a suite of new streaming algorithms for computing the Hamming, L_1, L_2 and general L_p $(0 < p < 2)$ distances between the pattern and the text. Our results significantly extend over the previous result in this setting. In particular, for the Hamming distance and for the L_p distance when $0 < p \leq 1$ we show a streaming algorithm that uses $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{n})$ space for polynomial-size alphabets.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming algorithms, approximate pattern matching

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.35

Category APPROX

Funding *Tatiana Starikovskaya*: This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).
Przemysław Uznański: Supported by Polish National Science Centre grant 2019/33/B/ST6/00298.

1 Introduction

In the problem of pattern matching, we are given a pattern P of length n and a text T and must find all occurrences of P in T. A particularly relevant variant of this fundamental question is approximate pattern matching, where the goal is to find all subwords of the text that are similar to the pattern. This can be restated in the following way: given a

pattern P, a text T, and a distance function, compute the distance between P and every n-length subword of T. A very natural similarity measure for words is the Hamming distance. Furthermore, if both P and T are over an integer alphabet Σ, one can consider the Manhattan distance or the Euclidean distance.

▶ **Definition 1** (Hamming, Manhattan and Euclidean distances). *For a vector $U = u_1u_2 \dots u_n$, its Hamming norm is defined as $\|U\|_H = |\{i : u_i \neq 0\}|$, Manhattan norm is defined as $\|U\|_1 = \sum_i |u_i|$ and Euclidean norm is defined as $\|U\|_2 = \left(\sum_i u_i^2\right)^{1/2}$. For two words $V = v_1v_2 \dots v_n$ and $W = w_1w_2 \dots w_n$, their Hamming distance is defined as $\|V - W\|_H$, their Manhattan distance as $\|V - W\|_1$, and their Euclidean distance as $\|V - W\|_2$.*

Those distance functions naturally generalize to the so called L_p distances, where $p > 0$ is the exponent.

▶ **Definition 2** (p'th moment, p'th norm). *For a vector $U = u_1u_2 \dots u_n$ and $p \geq 0$, its p'th moment F_p is defined as $F_p(U) = \sum_i |u_i|^p$, and for $p > 0$ its L_p norm is defined as $\|U\|_p = F_p(U)^{1/p} = (\sum_i |u_i|^p)^{1/p}$. For two words $V = v_1v_2 \dots v_n$ and $W = w_1w_2 \dots w_n$ considered as vectors, the p'th moment of their difference is $F_p(V-W)$ and their L_p distance is defined as $\|V - W\|_p = F_p(V - W)^{1/p} = (\sum_i |v_i - w_i|^p)^{1/p}$.*

In other words, the Manhattan distance is the L_1 distance, the Euclidean distance is the L_2 distance, and the Hamming distance can be considered as the L_0 distance.

Below we assume that the length of the text is $2n$, as any algorithm on a text of larger length can be reduced to repeated application of an algorithm that runs on texts of length $2n$. This is done by splitting the text into blocks of length $2n$ which overlap by n characters.

Offline setting. For the Hamming distance, the problem has been extensively studied in the offline setting, where we assume random access to the input. The first algorithm, for a constant-size alphabet, was shown by Fischer and Paterson [21]. The algorithm uses $\mathcal{O}(n \log n)$ time and in substance computes the Boolean convolution of two vectors a constant number of times. This was later extended to polynomial-size alphabets in [1, 33]. With a somewhat similar approach, the same complexity can be achieved for the L_1 distance in [13]. Later, in [34, 35] the authors proved that these problems must have equal (up to polylogarithmic factors) complexities by showing reductions from the Hamming to the L_1 distance and back.

To improve the complexity for large alphabets, the natural next step was to study approximation algorithms. Until very recently, the fastest $(1 \pm \varepsilon)$-approximation algorithm for computing the Hamming distances was by Karloff [29]. The algorithm combines random projections from an arbitrary alphabet to the binary one and Boolean convolution to solve the problem in $\mathcal{O}(\varepsilon^{-2}n \log^3 n)$ time. In a breakthrough paper Kopelowitz and Porat [31] gave a new approximation algorithm improving the time complexity to $\mathcal{O}(\varepsilon^{-1}n \log^3 n \log \varepsilon^{-1})$, which was later significantly simplified [32]. Using a similar technique, Gawrychowski and Uznański [23] showed an approximation algorithm for computing the L_1 distance in $\mathcal{O}(\varepsilon^{-1}n \log^4 n)$ (randomized) time, later made deterministic in time $\mathcal{O}(\varepsilon^{-1}n \log^2 n)$ in [39]. Using similar techniques, the authors of [39] gave $\widetilde{\mathcal{O}}(\varepsilon^{-1}n)$-time $(1 + \varepsilon)$-approximation algorithm for L_p distances for any constant positive p.[1]

[1] Across the paper we use $\widetilde{\mathcal{O}}$ to indicate that we are suppressing poly-log(n) factors.

Streaming setting. In the streaming setting, we assume that the pattern and the text arrive as streams, one character at a time (the pattern arrives before the text). The main objective is to design algorithms that use as little space as possible, and we must account for all the space used by the algorithm, including the space required to store the input, in full or in part. It is also often the case that the text arrives at a very high speed and we must be able to process it faster than it arrives to fulfil the space guarantees, preferably, in real time. To this aim, the time complexity of streaming algorithms is defined as the worst-case amount of time spent on processing one character of the text, i.e. *per arrival*.

In the streaming setting, only the Hamming distance (L_0) has been studied. It is known that computing the Hamming distance between a pattern and a streaming text exactly requires $\Omega(n)$ space, even for the binary alphabet and with a small probability error allowed, which can be shown by a straightforward reduction to communication complexity (folklore).

Therefore, to develop sublinear-space solutions, one must relax their requirements. One possibility to do so is to compute only the distances bounded by a threshold k. This variant of the problem is often reffered to as *k-mismatch problem*. The k-mismatch problem has been extensively studied in the literature [15, 16, 25, 38], with this line of work reaching $\widetilde{\mathcal{O}}(k)$ memory complexity and $\widetilde{\mathcal{O}}(\sqrt{k})$ time per input character. The motivation for this variant of this problem is that we are interested in subwords of the text that are similar to the pattern, in other words, the distance between the pattern and the text should be relatively small. On the other hand, the main application of the streaming setting is processing large-scale data, such as biological data. To decrease the cost of generating such data, recently new hardware approaches have been developed. They have become widely used due to cost efficiency, but unfortunately, the produced data may contain about 10% of noise [30]. To analyse such data, it is not sufficient to consider small distances only, and a possible workaround for this issue is $(1 \pm \varepsilon)$-approximation. This line of research was initiated by Clifford and Starikovskaya [17] who gave a $(1 \pm \varepsilon)$-approximation algorithm with space $\widetilde{\mathcal{O}}(\varepsilon^{-5}\sqrt{n})$ that uses $\widetilde{\mathcal{O}}(\varepsilon^{-4})$ time per arriving character of the text.

Independently and in parallel with this work, authors of [12] showed a $(1\pm\varepsilon)$-approximation streaming algorithm for the k-mismatch problem that uses $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{k})$ space. For a special case of $k = n$, they show how to reduce the space further to $\widetilde{\mathcal{O}}(\varepsilon^{-1.5}\sqrt{n})$. Compared to our solution, their algorithm has worse time complexity of $\widetilde{\mathcal{O}}(\varepsilon^{-3})$ per arrival, and more importantly, it is not obvious whether it can be generalised to other L_p norms as it uses a very different set of techniques.

Sliding window. The problem of computing distance between P and every n-length subword of T in the streaming setting resembles the problem of maintaining the L_p norm of a n-length suffix of a streaming text, also referred to as *sliding window*. In fact, the latter is a simplification of the former, with setting $P = [0, 0, \ldots, 0]$. There is an extensive line of work on maintaining the L_p norm of a sliding window, refer to [4, 5, 6, 7, 8, 18] and references therein. The main message is that the norm of a sliding window can be maintained efficiently, e.g. for $1 \leq p \leq 2$ the L_p norms can be maintained $(1 \pm \varepsilon)$-approximately in space $\widetilde{\mathcal{O}}(\varepsilon^{-1})$. However, those results do not translate to our case: in the sliding window, one can easily isolate "heavy hitters", that is updates with a significant contribution to the output. In our case, the contribution of an update depends on its relative position to the pattern, and one can easily construct instances where a contribution of a position in the text changes drastically relative to its alignment with the pattern, which necessitates a significantly different approach.

1.1 Our results

In this work, we show a suite of new streaming algorithms for computing the Hamming, L_1, L_2 and general L_p ($0 < p \leq 2$) distances between the pattern and the text. Our results significantly improve and extend the results of [17].

▶ **Theorem 3.** *Given a pattern P of length n and a text T over an alphabet $\Sigma = [1, 2, \ldots, \sigma]$, where $\sigma = n^{\mathcal{O}(1)}$, there is a streaming algorithm that computes a $(1 \pm \varepsilon)$-approximation of the L_p distance between P and every n-length subword of T correctly w.h.p.*

1) *in $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{n} + \log \sigma)$ space, and $\widetilde{\mathcal{O}}(\varepsilon^{-2})$ time per arrival when $p = 0$ (Hamming distance);*
2) *in $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{n} + \log^2 \sigma)$ space and $\widetilde{\mathcal{O}}(\sqrt{n} \log \sigma)$ time per arrival when $p = 1$ (Manhattan distance);*
3) *in $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{n} + \log^2 \sigma)$ space and $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{n})$ time per arrival when $0 < p < 1/2$;*
4) *in $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{n} + \log^2 \sigma)$ space and $\widetilde{\mathcal{O}}(\varepsilon^{-3}\sqrt{n})$ time per arrival when $p = 1/2$;*
5) *in $\widetilde{\mathcal{O}}(\varepsilon^{-2}\sqrt{n} + \log^2 \sigma)$ space and $\widetilde{\mathcal{O}}(\sigma^{\frac{2p-1}{1-p}}\sqrt{n}/\varepsilon^{2+3\cdot\frac{2p-1}{1-p}})$ time per arrival when $1/2 < p < 1$;*
6) *in $\widetilde{\mathcal{O}}(\varepsilon^{-2-p/2}\sqrt{n}\log^2 \sigma)$ space and $\mathcal{O}(\varepsilon^{-p/2}\sqrt{n} + \varepsilon^{-2}\log \sigma)$ time per arrival for $1 < p \leq 2$.*

We also improve and extend the space lower bound of [17], who showed that any streaming algorithm that computes a $(1 \pm \varepsilon)$-approximation of the Hamming distance between a pattern and a streaming text must use $\Omega(\varepsilon^{-2} \log^2 n)$ bits for all ε such that $1/\varepsilon < n^{1/2-\gamma}$ for some constant γ (condition inherited from [27]). We show the following result:

▶ **Lemma 4.** *Let $2 \leq 1/\varepsilon < n$ and $0 \leq p \leq 2$. Any $(1 \pm \varepsilon)$-approximation algorithm that computes the L_p distance between a pattern and a streaming text for each alignment, must use $\Omega(\min(1/\varepsilon^2, n))$ bits of space.*

1.2 Techniques

At a very high level, the structure of all algorithms presented in this paper is similar to that of [17]. We process the text by blocks of length $b \approx \sqrt{n}$. To compute an approximation of the distance / the p'th moment at a particular alignment, we divide the pattern into two parts: a prefix of length $\leq b$ aligned with a suffix of some block of the text, and the remaining suffix (see Fig. 1). We compute an approximation of the distance / the p'th moment for both of the parts and sum them up to obtain the final answer. Our main contribution is a set of new tools that allows computing the approximations efficiently.

To be able to compute the approximation of the distance / the p'th moment between the prefix and the corresponding block of the text, we compute, while reading each block of the text, its compact lossy description that we refer to as *prefix encoding*. The prefix encoding captures the relation between the read block and the prefix of the pattern of length b. To compute the distance / the p'th moment between the suffix and the text, we will use *suffix sketches*. For each position i of the text, the suffix sketch describes the subword $T[b \cdot k + 1, i]$ of the text where k is the smallest integer such that $i - b \cdot k \leq n$ (see Fig. 1).

For the Hamming distance, we define the prefix encodings in Section 2.1 and the suffix sketches in Section 3.1. Our Hamming prefix encoding introduces a novel use of a known technique called *subsampling*. The prefix encodings are used to approximate the distance between any suffix of one word and the prefix of another word of the same length. In brief, the idea is to replace each character of the two words by the don't care character "?", a special character that matches any other character of the alphabet. We repeat the process a logarithmic number of times to create a logarithmic number of pairs of "subsamples". For each pair, we find the longest suffix of one subsample that matches the prefix of the

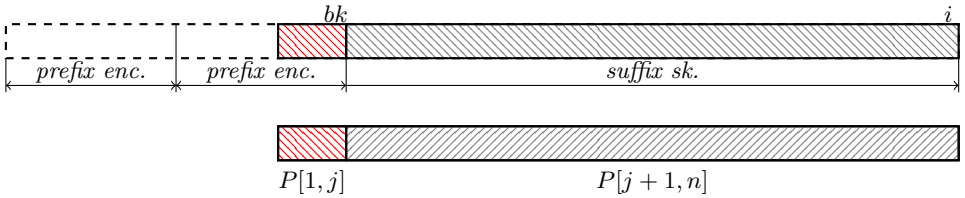

Figure 1 High level structure of the algorithms. To compute the distance between the prefix (red) of the pattern and the text, we use the prefix encoding, between the suffix (grey) and the text we use the suffix sketch.

second subsample up to at most $\Theta(1/\varepsilon^2)$ mismatches. We then show that this information can be used to approximate the Hamming distance between any suffix-prefix pair. Similar techniques were used in [3, 19, 22, 24, 28, 37] for estimating the Hamming norm in streams. The crucial difference with our approach is that we must be able to compute the Hamming norm of any suffix-prefix pair of the two words, and we must be able to do it efficiently. As for the suffix sketches, for the binary alphabet we use the sketches introduced in [17]. We then show a reduction from arbitrary alphabets to the binary alphabet, which improves the space consumption of Hamming suffix sketches by a factor of $1/\varepsilon^2$.

We can solve the problem of L_1 (Manhattan distance) pattern matching by replacing each character of the pattern and of the stream with its unary encoding and running the solution for the Hamming distance. However, this would introduce a multiplicative factor of σ (the size of the alphabet) to the time complexity. We show efficient randomised reductions from the Manhattan to Hamming distance that allow simulating the solution for the Hamming distance without a significant overhead. In particular, to design the prefix encodings we use random shifting and rounding, while for the suffix sketches we use range-summable hash functions [9]. We show the Manhattan prefix encodings in Section 2.2 and the Manhattan suffix sketches in Section 3.2.

For generic L_p distances, $0 < p \leq 2$, we discuss the prefix encodings in Section 2.4 and the suffix sketches in Section 3.3. Our approach to L_p prefix encodings is rather involved. In the case of $0 < p < 1$, we construct a novel embedding from L_p^p space into the Hamming space, which might be of independent interest. While the target dimension of the Hamming space is large, we construct the embedding in such a way that each value is mapped into a compressible sequence of form $c_1^{d_1} \dots c_t^{d_t}$ for some small value of t, and where values of $d_1, \dots, d_t$ are constant across all input values. Such compressed representation allows us to efficiently apply the subsampling framework and reduce the problem to the Hamming distance case. For $1 < p \leq 2$, we identify a logarithmic number of anchor suffixes, and partition each of them into ε^{-p} words of roughly even contribution to the distance. We then use the partition to decode prefix-suffix distance queries for arbitrary length queries. Such construction is a generalization and improvement of the approach presented in [17]. For suffix sketches, we simply use the p-stable distributions [26].

Finally, we combine the prefix encodings and the suffix sketches to prove Theorem 3 in Section 4. To simplify the notation, we use $x \overset{\varepsilon}{=} y$ to denote $(1-\varepsilon)y \leq x \leq (1+\varepsilon)y$ from now on. We will also use the fact that for $p > 0$ we can speak of approximating the p'th moment of differences between the pattern and the n-length substrings of the text and the L_p distances between the pattern and the n-length substrings of the text interchangeably, it changes the complexities up to a constant factor only:

▶ **Observation 5.** *For any constant $p > 0$ and $\varepsilon < 1/2$, there is a constant C_p such that finding a $(1 \pm C_p \cdot p\varepsilon)$ approximation of the p'th moment of a vector suffices for $(1 \pm \varepsilon)$-approximating its p'th norm, and finding a $(1 \pm C_p \cdot \varepsilon/p)$ approximation of its p'th norm suffices for $(1 \pm \varepsilon)$-approximating its p'th moment.*

2 Prefix encodings

In this section we present a solution to the following problem. Imagine we have a block of text $T'[1, b] = T[i + 1, i + b]$ and a prefix of the pattern $P' = P[1, b]$. We want to find a compressed representation (encoding) of T' so that the following is possible: given any $1 \le d \le b$, the compressed representation of T', and P' (explicitly), we can $1 \pm \varepsilon$ approximate $\|T'' - P''\|_p$, where $T'' = T'[b - d + 1, b]$ is a suffix of T' and $P'' = P'[1, d]$ is a prefix of P'.

We start by presenting a solution to the Hamming distance case, which is a basis to our solution for all other L_p norms for $0 < p \le 2$.

2.1 Hamming (L_0) distance

Recall that "?" is the don't care character, a special character that matches any other character of the alphabet.

▶ **Definition 6** (Hamming subsampling). *Consider a word U of length n. Let $q = \lceil 3 \log n \rceil$ and let $h(i) : [n] \to \{0, 1\}^q$ be a function drawn at random from a pairwise independent family. For $r = 0, \ldots, q$, we define the r-th level Hamming subsample of U, $\mathsf{hSub}_r(U)$, as follows:*

$$\mathsf{hSub}_r(U)[i] = \begin{cases} U[i], & \text{if the } r \text{ lowest bits of } h(i) \text{ are all } 0; \\ ?, & \text{otherwise.} \end{cases}.$$

In particular, $\mathsf{hSub}_0(U) = U$.

Fix an integer $k = \Theta(1/\varepsilon^2)$ large enough. For two words U, V, consider the following estimation procedure:

▶ **Algorithm 7.**

1. *Denote X_r to be the Hamming distance between $\mathsf{hSub}_r(U)$ and V and let $f = \min\{i : X_i \le k\}$.*[2]
2. *Output $Z_f = 2^f \cdot X_f$ as an estimate of $\|U - V\|_H$.*

The following lemma is a rephrasing of a known result regarding subsampling in estimation of the Hamming norm (cf. [3, Theorem 3], or [24, Theorem 2]).

▶ **Lemma 8.** *For Z_f as in Algorihtm 7 there is $Z_f \overset{\varepsilon}{=} \|U - V\|_H$ with probability at least $3/4$.*

Since the subsampling is performed independently for each position, one can use subsampling to approximate the Hamming distance between any suffix of B and any prefix of P of equal lengths in a similar fashion.

We are now ready to define the Hamming prefix encoding of a block. For brevity, let $B_r^j = \mathsf{hSub}_r(B)[b - j + 1, b]$ and $P_r^j = P[1, j]$ (the same for all r). Furthermore, given two words U, V of equal length, define the *mismatch information* $\mathsf{MI}(U, V) = \{(i, U[i], V[i]) : U[i] \text{ does not match } V[i]\}$.

[2] We emphasize that $\mathsf{hSub}_r(U)$ contains don't care characters, so the Hamming distance is defined as the number of pairs of characters of $\mathsf{hSub}_r(U)$ and V that do not match.

▶ **Definition 9.** *Consider a b-length block B of the text T. For each $0 \le r \le \lceil 3 \log n \rceil$, let $j^*(r)$ be the maximal integer such that the Hamming distance between $B_r^{j^*(r)}$ and $P_r^{j^*(r)}$ is at most $k = \Theta(\varepsilon^{-2})$. We define the Hamming prefix encoding of B to be a tuple of pairs $j^*(r), \mathsf{MI}(B_r^{j^*(r)}, P_r^{j^*(r)})$.*

Note that the prefix encoding of B uses $\mathcal{O}(k \log n) = \mathcal{O}(\varepsilon^{-2} \log n)$ space. We can compute it efficiently:

▶ **Lemma 10.** *Assume constant-time random access to $P[1, b]$. Given a b-length block B of the text T, its Hamming prefix encoding can be computed in $\widetilde{\mathcal{O}}(kb) = \widetilde{\mathcal{O}}(b\varepsilon^{-2})$ time.*

Proof. To compute the encoding, we use the algorithm of [14]. Formally, for each r we create a word T' by appending b don't care characters to the subsample hSub_r. The algorithm of [14] can be used to find all b-length subwords of T' that match $P[1, b]$ with up to k mismatches, moreover for each of these subwords the algorithm outputs the mismatch information. We take the leftmost subword only, which corresponds to $j^*(r)$ because of the don't care characters. In total, our algorithm uses $\widetilde{\mathcal{O}}(kb) = \widetilde{\mathcal{O}}(\varepsilon^{-2}b)$ time. ◀

We now show how to compute the Hamming distance between any j-length suffix of B and any j-length prefix of P given $P[1, b]$ and the Hamming prefix encoding of a block B.

▶ **Lemma 11.** *Given the prefix encoding of a b-length block B of the text T, there is an algorithm that computes, for any $j = 1, \ldots, b$, a $(1 + \varepsilon)$-approximation of the Hamming distance between the j-length suffix of B and the j-length prefix of P in $\widetilde{\mathcal{O}}(kb) = \widetilde{\mathcal{O}}(b\varepsilon^{-2})$ time.*

Proof. Denote X_r to be the Hamming distance between P_r^j and B_r^j. We compute the smallest f such that $X_f \le k$ in the following way. For each r, we use $\mathsf{MI}(B_r^{j^*(r)}, P_r^{j^*(r)})$ to restore $B_r^{j^*(r)}$. We then append $P_r^{j^*(r)}$ with b don't care characters and run the algorithm of [14] for the resulting text and the pattern. This allows to compute X_r for all $j \le j^*(r)$, and if $j > j^*(r)$, then $X_f > k$ by definition. In total, the algorithm takes $\widetilde{\mathcal{O}}(kb) = \widetilde{\mathcal{O}}(\varepsilon^{-2}b)$ time. ◀

2.2 Manhattan (L_1) distance

Recall a word morphism $\nu : \Sigma \to \{0,1\}^\sigma$, $\nu(a) = 1^a 0^{\sigma - a}$. Our goal in this section is to simulate implicitly procedures from Lemma 10 and Lemma 11 on words $\nu(B)$ and $\nu(T)$ without introducing any significant overhead.

▶ **Definition 12** (Manhattan scaling)**.** *Consider a word U of length n. Let $q = \lceil 3 \log n\sigma \rceil$ and let $h : [n] \to 2^q$ be a function drawn at random from a 4-wise independent family. For $r = 0, \ldots, q$, we define the r-th level Manhattan subsample of U, $\mathsf{mSub}_r(U)$, as a word of length n such that $\mathsf{mSub}_r(U)[i] = \left\lfloor \frac{U[i] + (h(i) \bmod 2^r)}{2^r} \right\rfloor$. In particular, $\mathsf{mSub}_0(U) = U$.*

Fix an integer $k = \Theta(1/\varepsilon^2)$ large enough. For words U, V, consider $\mathsf{mSub}_r(U), \mathsf{mSub}_r(V)$ for all $r = 0, \ldots, q$, and the following estimation procedure:

▶ **Algorithm 13.**
1. *Denote $X_r = \|\mathsf{mSub}_r(U) - \mathsf{mSub}_r(V)\|_1$ and let $f = \min\{i : X_i \le k\}$.*
2. *Output $Z_f = 2^f \cdot X_f$ as an estimate of $\|U - V\|_1$.*

▶ **Lemma 14.** *For Z_f as in Algorihtm 13 there is $Z_f \overset{\varepsilon}{=} \|U - V\|_1$ with probability $\ge 3/4$.*

To approximate the Manhattan distance between any suffix of B and any prefix of P of equal lengths, we define the encoding similar to the Hamming distance case. Specifically, we still use the *mismatch information*, building on the fact that for any two words $\|U - V\|_H \leq \|U - V\|_1$ and from the mismatch information the exact value of $\|U - V\|_1$ can be found. We define $B_r^j = \mathsf{mSub}_r(B)[b - j + 1, b]$ as before, but change the definition of P_r^j slightly. Intuitively, we define P_r^j to be the j-length prefix of P subsampled in a synchronized way with B_r^j. Formally, $P_r^j[i] = \left\lfloor \frac{P[i] + (h(b-j+i) \bmod 2^r)}{2^r} \right\rfloor$.

▶ **Definition 15.** *Consider a b-length block B of the text T. For each $0 \leq r \leq \lceil 3 \log n\sigma \rceil$, let $j^*(r)$ be the maximal integer such that the Manhattan distance between $B_r^{j^*(r)}$ and $P_r^{j^*(r)}$ is at most $k = \Theta(\varepsilon^{-2})$. We define the Manhattan prefix encoding of B to be a tuple of pairs $j^*(r), \mathsf{MI}(B_r^{j^*(r)}, P_r^{j^*(r)})$.*

Note that the prefix encoding of B uses $\mathcal{O}(k \log n\sigma) = \mathcal{O}(\varepsilon^{-2} \log n)$ space.

▶ **Lemma 16.** *Assume constant-time random access to $P[1, b]$. Given a b-length block B of the text T, its Manhattan prefix encoding can be computed in $\widetilde{\mathcal{O}}(b^2)$ time and $\widetilde{\mathcal{O}}(b)$ space.*

Proof. Let $q = \lceil 3 \log n\sigma \rceil$. For each $r = 0, \ldots, q$ and $j = 1, \ldots, b$ we compare B_r^j and P_r^j character by character in $\mathcal{O}(b)$ time to find $j^*(r)$ and the corresponding mismatch information. The claim follows. ◀

▶ **Lemma 17.** *Given the prefix encoding of a b-length block B of the text T, there is an algorithm that computes, for all $j = 1, \ldots, b$, a $(1 \pm \varepsilon)$-approximation of the Manhattan distance between the j-length suffix of B and the j-length prefix of P in $\widetilde{\mathcal{O}}(b^2)$ time.*

Proof. Denote $X_r = \|P_r^j - B_r^j\|_H$. We compute the smallest f such that $X_f \leq k$ in the following way. For each r, we use $\mathsf{MI}(B_r^{j^*(r)}, P_r^{j^*(r)})$ to restore $B_r^{j^*(r)}$. If $j > j^*(r)$, the Manhattan distance between P_r^j and B_r^j is at least k. Otherwise, we compare P_r^j and B_r^j character by character to compute the Manhattan distance in $\mathcal{O}(b)$ time. The claim follows. ◀

2.3 Generic (L_p) distance for $0 < p < 1$

Our goal is to construct a morphism (parametrised by p) acting as a randomized embedding of $(L_p)^p$ into the Hamming distance. The intuition behind our approach is as follows. Let $r_0, r_1, \ldots \in [0, 1]$ be a sequence of real numbers picked independently and u.a.r. Define a sequence of values

$$d_i = \begin{cases} \varepsilon^{-1} \cdot (1+\varepsilon)^{pi} & \text{when } i > 0 \\ \varepsilon^{-1} \cdot \frac{(1+\varepsilon)^p}{(1+\varepsilon)^p - 1} & \text{when } i = 0 \end{cases}$$

and for a character $c \in \Sigma$ consider sequence of characters $c_0, c_1, \ldots$ where $c_i = \lfloor \frac{c}{(1+\varepsilon)^i} + r_i \rfloor$ (similarly, a character c' defines a sequence $c'_0, c'_1, \ldots$). Now consider two characters $c, c' \in \Sigma$ such that $|c - c'| = (1+\varepsilon)^\ell$ for some integer ℓ and a random variable $x = \sum_{i=0}^{\infty} d_i \cdot \|c_i - c'_i\|_H$.

There is

$$\begin{aligned}\mathbb{E}[x] &= \sum_{i=0}^{\infty} d_i \cdot \Pr[c_i \neq c'_i] = \sum_{i=0}^{\ell} d_i \cdot 1 + \sum_{i=\ell+1}^{\infty} d_i \cdot \frac{|c-c'|}{(1+\varepsilon)^i} = \\ &= \varepsilon^{-1}\frac{(1+\varepsilon)^p}{(1+\varepsilon)^p-1} + \varepsilon^{-1}\sum_{i=1}^{\ell}\left((1+\varepsilon)^p\right)^i + \varepsilon^{-1}|c-c'|\sum_{i=\ell+1}^{\infty}\left((1+\varepsilon)^{p-1}\right)^i = \\ &= \varepsilon^{-1}\frac{((1+\varepsilon)^{\ell+1})^p}{(1+\varepsilon)^p-1} + |c-c'|\varepsilon^{-1}\frac{((1+\varepsilon)^{\ell+1})^{p-1}}{1-(1+\varepsilon)^{p-1}} = \\ &= |c-c'|^p\left(\frac{(1+\varepsilon)^p}{(1+\varepsilon)^p-1} + \frac{1}{(1+\varepsilon)^{1-p}-1}\right)\varepsilon^{-1} \approx \varepsilon^{-2}|c-c'|^p\frac{1}{p(1-p)}. \end{aligned} \tag{1}$$

We thus see that an idealized morphism of the form $\varphi : c \to c_0^{d_0}c_1^{d_1}\dots$ would have the property that $\|U-V\|_p^p \sim \|\varphi(U)-\varphi(V)\|_H$ on words of length n. But there are the following issues: (i) characters are mapped into infinite length words, (ii) number of repetitions of characters (d_i) is fractional, (iii) we cannot guarantee that character distance is always of form $(1+\varepsilon)^i$ and (iv) the distance is preserved only in expectation. We show how to overcome these issues to achieve the following result:

▶ **Theorem 18.** *Given $0 < p < 1$ and $\varepsilon > 0$ there is a word morphism $\varphi : c \in \Sigma \to c_0^{d_0}c_2^{d_2}\dots c_{t-1}^{d_{t-1}}$ such that:*

1) *$t = \widetilde{\mathcal{O}}(\varepsilon^{-2})$ when $0 < p < 1/2$, $t = \widetilde{\mathcal{O}}(\varepsilon^{-3})$ when $p = 1/2$ and $t = \widetilde{\mathcal{O}}(\sigma^{\frac{2p-1}{1-p}}/\varepsilon^{2+3\cdot\frac{2p-1}{1-p}})$ when $1/2 < p < 1$.*
2) *values of t and $d_0, \dots, d_{t-1}$ do not depend on c,*
3) *there exists a constant $\alpha = \alpha(p,\varepsilon)$ such that for any two words U, V of length at most n, we have $\|U-V\|_p^p \overset{\varepsilon}{=} \alpha \cdot \|\varphi(U)-\varphi(V)\|_H$ with probability at least $9/10$,*
4) *it is enough for the randomness to be realized by a hash function $r : [t] \to [D]$ from a 4-independent hash function family for some $D = \mathrm{poly}(n\sigma\varepsilon^{-1})$, which can be generated from a $\widetilde{\mathcal{O}}(\log\sigma)$ bits size seed.*

We now describe how to use the morphism φ to approximate the L_p distances in a small space. To design an efficient algorithm, we take advantage of the fact that $\varphi(U)$ has a compressed representation of size comparable with the length of U (at least when $p \le 1/2$).

▶ **Definition 19** (L_p scaling)**.** *Consider a word $S = s_1^{e_1}s_2^{e_2}\dots s_m^{e_m}$ of length $m' = \sum_i e_i$. Let $h : [m] \to 2^q$ be a function drawn at random from a 4-wise independent family, where $q = \lceil 3\log m'\rceil$. For $r = 0, \dots, q$, we define the r-th level subsample of S,*

$$\mathsf{Sub}_r(S) = (s_1)^{\left\lfloor \frac{e_1+(h(1) \bmod 2^r)}{2^r}\right\rfloor}(s_2)^{\left\lfloor \frac{e_2+(h(2) \bmod 2^r)}{2^r}\right\rfloor}\dots(s_m)^{\left\lfloor \frac{e_m+(h(m) \bmod 2^r)}{2^r}\right\rfloor}$$

In particular, $\mathsf{Sub}_0(U) = U$.

Consider two words S, Q of form $S = s_1^{e_1}\dots s_m^{e_m}$ and $Q = q_1^{e_1}\dots q_m^{e_n}$. Fix an integer $k = \Theta(1/\varepsilon^2)$ large enough and consider $\mathsf{Sub}_r(S), \mathsf{Sub}_r(Q)$ for all $r = 0, 1, \dots, \lceil 3\log m'\rceil$, where $m' = \sum_i e_i$.

▶ **Algorithm 20.**

1. *Denote $X_r = \|\mathsf{Sub}_r(S) - \mathsf{Sub}_r(Q)\|_H$ and let $f = \min\{i : X_i \le k\}$.*
2. *Output $Z_f = 2^f \cdot X_f$ as an estimate of $\|S-Q\|_H$.*

▶ **Lemma 21.** *For Z_f as in Algorihtm 20 there is $Z_f \overset{\varepsilon}{=} \|S-Q\|_H$ with probability $\ge 3/4$.*

We are now ready to define L_p prefix encodings. Consider a b-length block B of the text and define $B_r^j = \mathsf{Sub}_r(\varphi(B))[(b-j)t+1, bt]$ (t is defined as in Theorem 18). Also, define P_r^j to be the (tj)-length prefix of $\varphi(P)$ subsampled in a synchronized way with B_r^j.

▶ **Definition 22.** *Consider a b-length block B of the text T. For each $r = 0, \ldots, \lceil 3 \log n' \rceil$, where $n' = |\varphi(B)|$, let $j^*(r)$ be the maximal integer such that the Hamming distance between $B_r^{j^*(r)}$ and $P_r^{j^*(r)}$ is at most $k = \Theta(\varepsilon^{-2})$. We define the L_p prefix encoding of B to be a tuple of pairs $j^*(r), \mathsf{MI}(B_r^{j^*(r)}, P_r^{j^*(r)})$.*

The L_p prefix encoding of B uses $\mathcal{O}(k \log n') = \mathcal{O}(\varepsilon^{-2} \log(n\sigma\varepsilon^{-1}))$ space.

▶ **Lemma 23.** *Assume constant-time random access to $P[1, b]$. Given a b-length block B of the text T, its L_p prefix encoding can be computed in $\mathcal{O}(b^2 \cdot t \log n\sigma\varepsilon^{-1})$ time and $\mathcal{O}(b + \varepsilon^{-2} \log n\sigma\varepsilon^{-1})$ space.*

Proof. For each $r = 0, \ldots, \lceil 3 \log n' \rceil$ and $j = 1, \ldots, b$, we compute the Hamming distance between B_r^j and P_r^j in $\mathcal{O}(bt)$ time using the compressed representation to find $j^*(r)$ and the corresponding mismatch information. The claim follows. ◀

▶ **Lemma 24.** *Given the L_p prefix encoding of a b-length block B of the text T, there is an algorithm that computes, for all $j = 1, \ldots, b$, a $(1 \pm \varepsilon)$-approximation of the L_p distance between the j-length suffix of B and the j-length prefix of P in $\widetilde{\mathcal{O}}(b^2 \cdot t \log n\sigma\varepsilon^{-1})$ time and $\mathcal{O}(b + \varepsilon^{-2} \log n\sigma\varepsilon^{-1})$ space.*

Proof. Denote $X_r = \|P_r^j - B_r^j\|_H$. We compute the smallest f such that $X_f \le k$ in the following way. For each r, we use $\mathsf{MI}(B_r^{j^*(r)}, P_r^{j^*(r)})$ to restore $B_r^{j^*(r)}$. If $j > j^*(r)$, the Hamming distance between P_r^j and B_r^j is at least k. Otherwise, we compare P_r^j and B_r^j to compute the Hamming distance in $\mathcal{O}(bt)$ time. The claim follows. ◀

2.4 Generic (L_p) distance for $1 < p \le 2$

For $1 < p \le 2$, we use a scheme similar to the one developped in [17] for the Hamming distance, but adapt it to generic L_p distances. Particularly, we plug in a standard tool used in this situation, the p-stable distribution. We additionally have to adapt the scheme a bit, taking into account that L_p norm is sub-additive under concatenation when $p > 1$.

▶ **Definition 25** (p-stable distribution [40])**.** *For a parameter $p > 0$, we say that a distribution $\mathcal{D}$ is p-*stable *if for all $a, b \in \mathbb{R}$ and random variables X, Y drawn independently from $\mathcal{D}$, the variable $aX + bY$ is distributed as $(|a|^p + |b|^p)^{1/p} Z$, where Z is a random variable with distribution $\mathcal{D}$.*

Consider a word $X = x_1 x_2 \ldots x_n$, and let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be independent random variables drawn from a p-stable distribution $\mathcal{D}$ with expected value $\mu_{\mathcal{D}}$. By Definition 25, we have $\mathbb{E}\left[\sum_i \alpha_i x_i\right] / \mu_{\mathcal{D}} = \|X\|_p$. The p-stable distributions exist for all $0 < p \le 2$, and a random variable X from a p-stable distribution can be generated using the formula $X = \frac{\sin(p\Theta)}{\cos^{1/p}(\Theta)} \left(\frac{\cos(\Theta(1-p))}{\ln(1/r)}\right)^{(1-p)/p}$ [11, 40], where Θ is uniform on $[-\pi/2, \pi/2]$ and r is uniform on $[0, 1]$.

However, to be able to design an efficient sketching scheme that allows to approximate the L_p norm with high probability, there are three technicalities to be overcome: First, one must show that $\sum_i \alpha_i x_i$ concentrates well, second, the formula above assumes infinite precision of computation, and finally, one cannot use fully independent random variables α_i as above

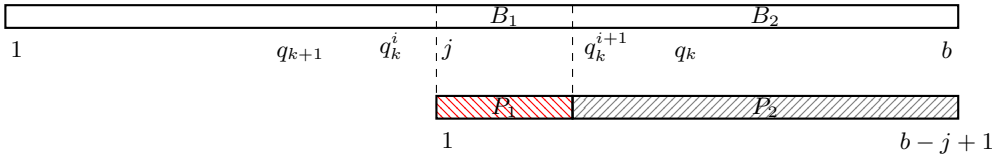

Figure 2 Using the prefix encoding of B to compute the L_p distance between a suffix of B and a prefix of the pattern. To compute the distance between B_1 and P_1, we replace B_1 with a subword of the pattern, and between B_2 and P_2 we use the sketches.

as this would require much space. To overcome these issues, Indyk [26] combined p-stable distributions and pseudorandom generators for bounded space computation [36]. We restate the final result of Indyk below, in the form that will be convenient for us later.

▶ **Theorem 26** (cf. Theorem 2, Theorem 4 [26]). *For any $0 < p \leq 2$, there is a non-uniform streaming algorithm that maintains a sketch $\mathsf{Sketch_p}(S)$ of a word S of length n over an alphabet of size σ such that:*

1) *when a new character of S arrives, the sketch can be updated in $\mathcal{O}(\varepsilon^{-2}\log(n/\varepsilon))$ time;*

2) *the algorithm and the sketch use $\mathcal{O}(\varepsilon^{-2}\log(\sigma n/\varepsilon)\log(n/\varepsilon))$ bits of space.*

Given the sketches $\mathsf{Sketch_p}(X), \mathsf{Sketch_p}(Y)$ of two words X, Y of length n, one can estimate $\|X-Y\|_p$ up to a factor $1\pm\varepsilon$ with probability at least $9/10$ in time $\widetilde{\mathcal{O}}(1/\varepsilon^2)$.

We now proceed to building the L_p prefix encoding by using $\mathsf{Sketch_p}$ and the landmarking technique.

▶ **Definition 27** (L_p prefix encoding). *Let $1 < p \leq 2$. Consider a word S of length b on the alphabet of size σ. Define $q_0 = b$. For $k = 0, \ldots, \lceil \log b\sigma^p \rceil$, let $q_k \leq q_{k-1}$ be the leftmost position such that the p'th moment of the difference between $S[q_k, b]$ and $P[1, b-q_k+1]$, i.e. $\|S[q_k,b] - P[1,b-q_k+1]\|_p^p$, is at most 2^k.*

Further, divide $S[q_k, b]$ into $\Theta(1/\varepsilon^p)$ blocks such that each block is either a single character, or the p'th moment of the difference between each block and the corresponding subword of $P[1, b-p_k+1]$ is at most $\varepsilon^p \cdot 2^k$. Let $q_k = q_k^0 \leq q_k^1 \leq \ldots q_k^{\ell_k} = b$ be the block borders. We choose $q_k^1, q_k^2, \ldots, q_k^{\ell_k}$ from left to right, and each position q_k^i is chosen to be the rightmost possible.

The L_p prefix encoding *of S is defined to contain sorted lists of the positions q_k and q_k^i, characters $S[q_k^i]$, and sketches for $(1 \pm C_p \cdot \varepsilon/p)$-approximating the p'th norm of $S[q_k^j, b]$, for all k, j and C_p as in Observation 5, see also Theorem 26.*

The encoding takes $\widetilde{\mathcal{O}}(\varepsilon^{-2-p}\log\sigma\log(\sigma n/\varepsilon)\log(n/\varepsilon))$ bits of space. We now show that given the L_p prefix encoding of a block B of the text of length b, one can compute a $(1\pm\varepsilon)$-approximation of the L_p distance between any prefix $P[1, b-j+1]$ of the pattern P and the corresponding suffix $B[j, b]$ of B.

▶ **Lemma 28.** *Let $1 < p \leq 2$. For any two vectors X, Y of equal length, $\left| \|X+Y\|_p^p - \|X\|_p^p \right| = \mathcal{O}(\|Y\|_p^p + \|Y\|_p \cdot \|X\|_p^{p-1})$.*

▶ **Lemma 29.** *Let $1 < p \leq 2$. Given the L_p prefix encoding of a block B of the text T of length b, one can find $(1\pm\varepsilon)$-approximation of the p'th moment of the difference between any prefix $P[1, b-j+1]$ of the pattern P and the corresponding suffix $B[j, b]$ of B in time $\widetilde{\mathcal{O}}(\varepsilon^{-2} + \log\sigma)$.*

Proof. Let q_k be the position that is closest to i from the left, and $q_k^i \le j < q_k^{i+1}$ (see Fig. 2). We can find q_k, q_k^i, q_k^{i+1} in time $\mathcal{O}(\log(b\sigma^p) + 1/\varepsilon^p)$ by iterating over the sorted lists.

The position q_k^{i+1} divides $P[1, b-j+1]$ into two parts, P_1 and P_2. Denote B_1 and B_2 the respective subwords of B they are aligned with (see Fig. 2). Let $m_1 = F_p(P_1 - B_1)$ and $m_2 = F_p(P_2 - B_2)$. Then $m = F_p(P[1, b-j+1] - B[j, b])$, being the value we need to approximate, is equal to $m_1 + m_2$.

We can find $m_2' \overset{\varepsilon}{=} m_2$ using the sketches for $B_2 = B[q_k^{i+1}, b]$ and P_2 in time $\widetilde{\mathcal{O}}(1/\varepsilon^2)$. Furthermore, if $q_k^i = q_k^{i+1} - 1$, then we can compute m_1 exactly as we store $B[q_k^i]$. Otherwise, we consider the subword $\tilde{P} = P[j - q_k + 1, q_k^{i+1} - q_k + 1]$ of the pattern P. Denote $m_1' = F_p(P_1 - \tilde{P})$ and use it as our estimation of m_1.

Since $1 \le p \le 2$, by definition, $F_p(B_1 - \tilde{P}) \le \varepsilon^p \cdot 2^{k-1}$, and $F_p(P_1 - B_1) \le 2^k$. By Lemma 28 with $X = P_1 - B_1$ and $Y = B_1 - \tilde{P}$,

$$|m_1' - m_1| = \mathcal{O}(\|B_1 - \tilde{P}\|_p^p + \|B_1 - \tilde{P}\|_p \|P_1 - B_1\|_p^{p-1}) = \mathcal{O}(\varepsilon^p 2^k + \varepsilon (2^k)^{\frac{1}{p}} (2^k)^{\frac{p-1}{p}}) = \mathcal{O}(\varepsilon 2^k)$$

and finally $|(m_1 + m_2) - (m_1' + m_2')| \le \mathcal{O}(\varepsilon m) + \varepsilon m_2 = \mathcal{O}(\varepsilon m)$. ◀

▶ **Lemma 30.** *Let $1 < p \le 2$. The L_p prefix encoding of a b-length block B of the text can be computed in time $\tilde{\mathcal{O}}(b^2 + \varepsilon^{-2} b \log \sigma)$ and space $\tilde{\mathcal{O}}(b + \varepsilon^{-2-p} \log^2 \sigma)$.*

Proof. For $j = 1, \ldots, b$, we naively compute the L_p distance between the suffix of B and the prefix of P in $\mathcal{O}(b)$ time. We then find the positions q_k. For each $k = 0, \ldots, \lceil \log b\sigma^p \rceil$, we can find the positions q_k^i in $\mathcal{O}(b)$ time and compute the sketches in $\tilde{\mathcal{O}}(\varepsilon^{-2} b)$ time by Theorem 26. ◀

3 Suffix sketches

In this section, we give the definitions and explain how we maintain the suffix sketches for each of the distances.

3.1 Hamming distance

We first recall Euclidean suffix sketches as presented in [17]. In fact, we will not use them for the Euclidean distance as for it we can use the generic solution of Section 3.3, but they will serve as a foundation of Hamming suffix sketches.

All sketches presented in this section are correct with constant probability, which can be amplified to $1 - \delta$ for arbitrarily small δ by a standard method of repeating sketching independently $\Theta(\log \delta^{-1})$ times and taking the median of the estimates.

▶ **Lemma 31** (Euclidean sketches [2])**.** *Let M be a random matrix of size $d \times n$ filled with 4-wise independent random ± 1 variables, for $d = \Theta(\varepsilon^{-2})$ chosen big enough. For a vector $X \in \mathbb{R}^n$ there is $\frac{1}{\sqrt{d}} \|MX\|_2 \overset{\varepsilon}{=} \|X\|_2$ with constant probability $9/10$, taken over all possible choices of M. We say that a vector MX of dimension d is a Euclidean sketch of X.*

▶ **Definition 32** (Euclidean suffix sketches [17])**.** *Consider a word X of length n. We define its Euclidean suffix sketch as follows.*

Let b be the block length*. Let $\mathcal{R}$ be a random matrix of size $d \times b$ filled with 4-wise independent random ± 1 variables and let $\alpha_1, \ldots, \alpha_{\lceil n/b \rceil}$ be 4-wise independent random coefficients with values ± 1 as well. We define a matrix M of size $d \times n$ such that $M_{i,jb+k} = \alpha_j \cdot \mathcal{R}_{i,k}$.*

Let X' be a word of length $\lceil n/b \rceil \cdot b$ obtained from X by appending an appropriate number of zeroes. The Euclidean suffix sketch *of X is defined as $\mathsf{eSketch}(X) = MX'$, where X' is considered as a vector.*

Observe that the matrix M does not need to be accessed explicitly. Indeed, from $MX' = \sum_i \alpha_i \cdot \mathcal{R} \cdot \left[X'[bi],\ \ldots,\ X'[bi+b-1]\right]^T$ it follows that the Euclidean suffix sketch can be computed by first sketching each block of X' using the matrix $\mathcal{R}$, and then taking a linear combination of the sketches of the blocks (using the random ± 1 coefficients α_i).

▶ **Lemma 33** ([17]). *Selecting $d = \Theta(\varepsilon^{-2})$ gives $\frac{1}{\sqrt{d}}\|\mathsf{eSketch}(X)\|_2 \overset{\varepsilon}{=} \|X\|_2$ with probability at least $9/10$ (taken over all possible choices of $\mathcal{R}, \alpha_i$).*

By linearity of sketches, we obtain $\|X - Y\|_2 \overset{\varepsilon}{=} \frac{1}{\sqrt{d}}\|\mathsf{eSketch}(X) - \mathsf{eSketch}(Y)\|_2$ with probability at least $9/10$ as well.

We now define Hamming suffix sketches. First note that for binary words X, Y there is $\mathsf{Ham}(X,Y) = \|X - Y\|_2$, and therefore in the case of the binary alphabet we can use the Euclidean suffix sketches. We will now show how to reduce the case of arbitrary polynomial-size alphabets to the case of the binary alphabet.

To this end, [17] used a random mapping of Karloff [29] as a black-box reduction, which led to sketches of size $\sim \varepsilon^{-4}$. We now show a more careful reduction to avoid this overhead and to achieve dependency ε^{-2} in total. Consider a word morphism defined on alphabet as $\mu : \Sigma \to \{0,1\}^\sigma$, $\mu(a) = 0^a 10^{\sigma-a-1}$ (and acting on words by concatenating the images of each character of the input word). Note that $\|\mu(X) - \mu(Y)\|_2^2 = 2 \cdot \|X - Y\|_H$, thus using the Euclidean suffix sketches on top of $\mu(X)$ and $\mu(Y)$ allows computation of the respective Hamming distance. Formally,

▶ **Definition 34** (Hamming suffix sketches [17]). *Consider a word X of length n on the alphabet of size σ. We define its Hamming suffix sketch as follows.*

Let b be the block length*, $\mathcal{R}$ be a random matrix of size $d \times \sigma b$ filled with 4-wise independent random ± 1 variables, and $\alpha_1, \ldots, \alpha_{\lceil n/b \rceil}$ be 4-wise independent random coefficients with values ± 1 as well. We define a matrix M of size $d \times \sigma n$ such that $M_{i,\sigma j b+k} = \alpha_j \cdot \mathcal{R}_{i,k}$.*

Let X' be a word of length $\lceil n/b \rceil \cdot b$ obtained from X by appending an appropriate number of zeroes. The Hamming suffix sketch *of X is defined as $\mathsf{hSketch}(X) = M\mu(X')$, where $\mu(X')$ is considered as a vector.*

▶ **Lemma 35.** *Selecting $d = \Theta(\varepsilon^{-2})$ gives $\frac{1}{2d}\|\mathsf{hSketch}(X)\|_2^2 \overset{\varepsilon}{=} \|X\|_H$ with probability at least $9/10$ (taken over all possible choices of $\mathcal{R}, \alpha_i$).*

Proof. Follows immediately as a corollary of Lemma 33 and the properties of the embedding μ. In more detail, the following holds with probability at least $9/10$:

$$\frac{1}{2d} \cdot \|\mathsf{hSketch}(X)\|_2^2 =$$
$$= \frac{1}{2d}\|M\mu(X')\|_2^2 = \frac{1}{2d}\|M\mu(X)\|_2^2 = \frac{1}{2d}\|\mathsf{eSketch}(\mu(X))\|_2^2 \overset{\varepsilon}{=} \frac{1}{2}\|\mu(X)\|_2^2 = \|X\|_H. \qquad \blacktriangleleft$$

As $\mu(X), \mu(Y)$ are sparse, there is an efficient streaming algorithm for maintaining the Hamming suffix sketches of a text:

▶ **Lemma 36.** *Given a text T, there is a streaming algorithm that for every position i outputs the Hamming suffix sketch of a word $T[b \cdot k + 1, i]$, where k is the largest integer such that $i - b \cdot k \le n$. The algorithm takes $\mathcal{O}(dn/b + \log d\sigma n)$ space and $\mathcal{O}(d(1 + n/b^2))$ time per character.*

3.2 Manhattan (L_1) distance

To show efficient suffix sketches for the Manhattan distance, we consider a word morphism $\nu : \Sigma \to \{0,1\}^\sigma$, $\nu(a) = 1^a 0^{\sigma-a}$. Note that $\|\nu(X) - \nu(Y)\|_2^2 = \|\nu(X) - \nu(Y)\|_H = \|X - Y\|_1$, thus using the Hamming suffix sketches on top of $\nu(X)$ and $\nu(Y)$ allows computation of the respective Manhattan distance.

However, if we apply the morphism straightforwardly, we will have to pay an extra σ factor per character to compute the Manhattan suffix sketches. To improve the running time, we will use range-summable hash functions. Range-summable hash functions were introduced by Feigenbaum et al. [20], and later their construction was improved by Calderbank et al. [9].

▶ **Definition 37** (cf. [9]). *A family $\mathcal{H}$ of hash functions $h(x;\xi) : [t] \times \{0,1\}^s \to \{-1,1\}$ (here x is the argument and ξ is the seed) is called k-independent, range-summable if it satisfies the following properties for any $h \in \mathcal{H}$:*

1. (k-independent) *for all distinct $0 \le x_1, \ldots, x_k < t$ and all $b_1, \ldots, b_k \in \{-1,+1\}$,*

$$\Pr_{\xi \in \{0,1\}^s}[h(x_1;\xi) = b_1 \wedge \cdots \wedge h(x_k;\xi) = b_k] = 2^{-k}$$

2. (range-summable) *there exists a function g such that given a pair of integers $0 \le \alpha, \beta \le \sigma$, and a seed ξ, the value $g(\alpha,\beta;\xi) = \sum_{\alpha \le x < \beta} h(x;\xi)$ can be computed in time polynomial in $\log t$.*[3]

▶ **Corollary 38** (cf. Theorem 3.1 [9]). *There is a 4-independent, range-summable family of hash functions $h(x;\xi) : [t] \times \{0,1\}^s \to \{-1,+1\}$ with a random seed ξ of length $s = \mathcal{O}(\log^2 t)$ such that any range-sum $g(\alpha,\beta;\xi)$ can be computed in $\mathcal{O}(\log^3 t)$ time.*

▶ **Observation 39.** *For a word $X = x_1 x_2 \ldots x_n$, let $Y = \nu(X) = y_1 y_2 \ldots y_{n\sigma}$. Let h, g be as in Corollary 38 with $t = n\sigma$. Then $\sum_{i=1}^{n} g(i\sigma, i\sigma + x_i; \xi) = \sum_{i=1}^{n\sigma} y_i h(i;\xi)$.*

Thus, we see that range-summable hash functions can be used to efficiently simulate ν.

▶ **Definition 40** (Manhattan suffix sketches). *Consider X be a word of length n. We define its Manhattan suffix sketch as follows.*

Let b be the block length. *Let h, g be as in Corollary 38 with $t = bd\sigma$. Let $\mathcal{R}$ be a random matrix of size $d \times \sigma b$ filled with 4-wise independent random ± 1 variables, such that $\mathcal{R}_{i,k} = h(ib\sigma + k;\xi)$ and let $\alpha_1, \ldots, \alpha_{\lceil n/b \rceil}$ be 4-wise independent random coefficients with values ± 1 as well. We define a matrix M of size $d \times \sigma n$ such that $M_{i,\sigma jb+k} = \alpha_j \cdot \mathcal{R}_{i,k} = \alpha_j \cdot h(dk+i;\xi)$.*

Let X' be a word of length $\lceil n/b \rceil \cdot b$ obtained from X by appending an appropriate number of zeroes. The Manhattan suffix sketch *of X is defined as $\mathsf{mSketch}(X) = M\nu(X')$, where $\nu(X')$ is considered as a vector.*

▶ **Lemma 41.** *Selecting $d = \Theta(1/\varepsilon^2)$ gives $\frac{1}{d}\|\mathsf{mSketch}(X)\|_2^2 \overset{\varepsilon}{=} \|X\|_1$ with probability at least $9/10$ (taken over all possible choices of α_i and ξ).*

Proof. Follows immediately as a corollary of Lemma 33 and the properties of the embedding ν. In more detail, the following holds with probability at least $9/10$:

$$\frac{1}{d} \cdot \|\mathsf{mSketch}(X)\|_2^2 =$$
$$= \frac{1}{d}\|M\nu(X')\|_2^2 = \frac{1}{d}\|M\nu(X)\|_2^2 = \frac{1}{d}\|\mathsf{eSketch}(\nu(X))\|_2^2 \overset{\varepsilon}{=} \|\nu(X)\|_2^2 = \|X\|_1. \qquad \blacktriangleleft$$

[3] In [9], the function h was defined to take values in $\{0,1\}$. We can change the range of values to $\{-1,+1\}$ by taking $h' = 1 - 2h$ while preserving the properties.

▶ **Lemma 42.** *Given a text T, there is a streaming algorithm that for every position i outputs the Manhattan suffix sketch of a word $T[b \cdot k + 1, i]$, where k is the smallest integer such that $i - b \cdot k \leq n$. The algorithm takes $\mathcal{O}(d \cdot (n/b) + \log^2 \sigma)$ space, and $\mathcal{O}(d(1 + n/b^2) \cdot \log^3(bd\sigma))$ time per character.*

3.3 Generic (L_p) distance for $0 < p \leq 2$

For generic L_p distances, we use the approach of [26] based on p-stable distributions.

▶ **Corollary 43.** *Given a text T, there is a streaming algorithm that for every position i outputs the L_p suffix sketch of a word $T[b \cdot k + 1, i]$, where k is the smallest integer such that $i - b \cdot k \leq n$. The algorithm takes $\mathcal{O}(\varepsilon^{-2}(n/b) \cdot \log(\sigma n/\varepsilon) \log(n/\varepsilon))$ bits of space and $\mathcal{O}(\varepsilon^{-2}(n/b) \log(n))$ time per character.*

Proof. We start a new instance of the sketching algorithm of Theorem 26 at every block border and continue running it for the next $\lceil n/b \rceil$ blocks. At each moment, there are $\mathcal{O}(n/b)$ active instances of the algorithm. The bounds follow. ◀

4 Proof of Theorem 3

Recall the structure of the algorithms. During the preprocessing, we compute the suffix sketches of suffixes $P[1, n], P[2, n], \ldots, P[b, n]$ of P. During the main stage, the text is processed by blocks of length b. To compute an approximation of the distance / the p'th moment at a particular alignment, we divide the pattern into two parts: a prefix of length at most b, and the remaining suffix. We compute an approximation of the distance / the p'th moment for both of the parts and sum them up to obtain the final answer. To compute an approximation of the distance / the p'th moment between the prefix and the corresponding block of the text, we compute, while reading each block of the text, its prefix encoding, and to compute an approximation of the distance / the p'th moment between the suffix and the text, we use the suffix sketches.

1) **Hamming (L_0) distance.** When we receive a new block of the text, we compute its Hamming prefix encoding using the algorithm of Lemma 10 in $\mathcal{O}(b)$ space. We de-amortize the computation over the subsequent block and spend $\widetilde{\mathcal{O}}(\varepsilon^{-2})$ time per character. We store the resulting encoding for the next $\mathcal{O}(n/b)$ blocks. In total, the encodings require $\widetilde{\mathcal{O}}(\varepsilon^{-2}n/b)$ space. The Hamming suffix sketches of $P[1, n], P[2, n], \ldots, P[b, n]$ occupy $\mathcal{O}(\varepsilon^{-2}b)$ space. The algorithm of Lemma 36 that computes the suffix sketches takes $\mathcal{O}(\varepsilon^{-2}n/b + \log(\varepsilon^{-2}\sigma n))$ space and $\mathcal{O}(\varepsilon^{-2}(1 + n/b^2))$ time per character. Consider a block starting with position p. To compute the Hamming distances between n-length subwords that end in this block and the pattern, we apply the following approach. First, while reading the block preceding the current one, we decode the Hamming prefix encoding of the block that starts at position $p - n$ using Lemma 11. We de-amortize the algorithm to spend $\widetilde{\mathcal{O}}(\varepsilon^{-2})$ time per character. Hence, at the position i, we know the $(1 \pm \varepsilon)$-approximation between the prefixes of the pattern and the corresponding subwords of the text. At each position, we can compute the Hamming distance between the corresponding suffix of the pattern and the text in $\widetilde{\mathcal{O}}(\varepsilon^{-2})$ time using the Hamming suffix sketch. By taking $b = \sqrt{n}$, we obtain the claim.
2) **Manhattan (L_1) distance.** We proceed analogously to the Hamming distance case. The Manhattan prefix encoding of each block is computed using Lemma 16, in $\widetilde{\mathcal{O}}(b)$ time per character. We store the resulting encoding for the next $\mathcal{O}(n/b)$ blocks, giving

in total $\widetilde{\mathcal{O}}(\varepsilon^{-2}n/b)$ space. The Manhattan suffix sketches of $P[1,n], P[2,n], \ldots, P[b,n]$ occupy $\mathcal{O}(\varepsilon^{-2}b)$ space. Algorithm of Lemma 42 takes $\widetilde{\mathcal{O}}(\varepsilon^{-2}(b+n/b)+\log^2\sigma)$ space and $\widetilde{\mathcal{O}}(\varepsilon^{-2}(1+n/b^2))$ time per character. For decoding the prefix encoding we use Lemma 17, spending $\widetilde{\mathcal{O}}(b)$ time per character. Once again we take $b=\sqrt{n}$, and assume w.l.o.g. $\varepsilon^{-1}\leq\sqrt{n}$ (as otherwise we can use a naive algorithm with $\mathcal{O}(n)$ space and $\mathcal{O}(n)$ time per character).

3) **Generic (L_p) distance for $0<p<1$.** The L_p prefix encodings of the blocks are computed using Lemma 23, using $\widetilde{\mathcal{O}}(t\cdot b)$ time per character. We store the resulting encodings for the next $\mathcal{O}(n/b)$ blocks, giving in total $\widetilde{\mathcal{O}}(\varepsilon^{-2}n/b)$ space. The L_p suffix sketches of $P[1,n], P[2,n], \ldots, P[b,n]$ occupy $\widetilde{\mathcal{O}}(\varepsilon^{-2}b\log\sigma)$ space. Algorithm of Corollary 43 computes the L_p suffix sketches for the text in $\widetilde{\mathcal{O}}(\varepsilon^{-2}(n/b)\log\sigma)$ space and $\widetilde{\mathcal{O}}(\varepsilon^{-2}n/b)$ time per character. For decoding the prefix encoding we use Lemma 24, spending $\widetilde{\mathcal{O}}(t\cdot b)$ time per character. We take $b=\sqrt{n}$, and substitute t accordingly to Theorem 18.
4) **Generic (L_p) distance for $1<p<2$.** Note that for $\varepsilon<1/n$ we can use a naive algorithm, that is to store S itself in $\mathcal{O}(n)$ space. The update takes constant time, and computing the L_p norm takes $\mathcal{O}(n)$ time which is better than the guarantees of the theorem for such values of ε. For $\varepsilon\geq 1/n$, the algorithm of Lemma 30 computes the L_p prefix encodings of the blocks in $\widetilde{\mathcal{O}}(b+\varepsilon^{-2-p}\log^2\sigma)$ space and $\widetilde{\mathcal{O}}(b+\varepsilon^{-2}\log\sigma)$ time per character. The encodings occupy $\widetilde{\mathcal{O}}(\varepsilon^{-2-p}(n/b)\log^2\sigma)$ space. The L_p suffix sketches of $P[1,n], P[2,n], \ldots, P[b,n]$ occupy $\widetilde{\mathcal{O}}(\varepsilon^{-2}b\log\sigma)$ space. Algorithm of Corollary 43 computes the L_p suffix sketches for the text in $\widetilde{\mathcal{O}}(\varepsilon^{-2}(n/b)\log\sigma)$ space and $\widetilde{\mathcal{O}}(\varepsilon^{-2}n/b)$ time per character. Taking $b=\varepsilon^{-p/2}\sqrt{n}$ and assuming w.l.o.g. $\varepsilon^{-1}<\sqrt{n}$, we obtain the claim.

5 Conclusion

We pose several open questions. First is whether the time-complexity for $1/2<p<1$ can be improved to not involve any dependency on σ. For this we need a better technique than bounding variance of the embedding into Hamming distance: in our technique, the tail gets "too heavy". Another pressing question is whether for all values of $p>0$ we could improve upon $\sqrt{n}$ time per character. We also remark that it seems unlikely that an embedding to Hamming space could be used to reduce space complexity for $p>1$: L_p^p does not admit the triangle inequality while the Hamming distance does, and the L_p distance is not additive with respect to concatenation, while the Hamming distance is.

References

1 Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

2 Dimitris Achlioptas. Database-friendly random projections: Johnson–Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003. doi:10.1016/S0022-0000(03)00025-4.

3 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM 2002*, pages 1–10, 2002. doi:10.1007/3-540-45726-7_1.

4 Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch L_2-heavy-hitters on sliding windows. *Theor. Comput. Sci.*, 554:82–94, 2014.

5 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *FOCS 2007*, pages 283–293, 2007.

6 Vladimir Braverman and Rafail Ostrovsky. Effective computations on sliding windows. *SIAM J. Comput.*, 39(6):2113–2131, 2010.

7 Vladimir Braverman, Rafail Ostrovsky, and Alan Roytman. Zero-one laws for sliding windows and universal sketches. In *APPROX-RANDOM 2015*, pages 573–590, 2015.

8 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012.

9 A. Robert Calderbank, Anna C. Gilbert, Kirill Levchenko, S. Muthukrishnan, and Martin Strauss. Improved range-summable random variable construction algorithms. In *SODA 2005*, pages 840–849, 2005.

10 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-Hamming-distance. In *STOC 2011*, pages 51–60, 2011. doi:10.1145/1993636.1993644.

11 J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random variables. *J. of the American Statistical Association*, 71(354):340–344, 1976.

12 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *STOC 2020*, pages 643–656, 2020. doi:10.1145/3357713.3384266.

13 Peter Clifford, Raphaël Clifford, and Costas S. Iliopoulos. Faster algorithms for delta, gamma-matching and related problems. In *CPM 2005*, pages 68–78, 2005. doi:10.1007/11496656_7.

14 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *SODA 2009*, pages 778–784, 2009.

15 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *SODA 2016*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142.

16 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *SODA 2019*, pages 1106–1125, 2019. doi:10.1137/1.9781611975482.68.

17 Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming distance in a stream. In *ICALP 2016*, pages 20:1–20:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.20.

18 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. doi:10.1137/S0097539701398363.

19 Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In *ESA 2003*, pages 605–617, 2003. doi:10.1007/978-3-540-39658-1_55.

20 Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. An approximate L1-difference algorithm for massive data streams. *SIAM J. Comput.*, 32(1):131–151, 2002. doi:10.1137/S0097539799361701.

21 Michael J. Fischer and Michael S. Paterson. String-matching and other products. Technical report, Massachusetts Institute of Technology, 1974.

22 Sumit Ganguly. Counting distinct items over update streams. *Theor. Comput. Sci.*, 378(3):211–222, 2007. doi:10.1016/j.tcs.2007.02.031.

23 Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *ICALP 2018*, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62.

24 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *SPAA 2001*, pages 281–291, 2001. doi:10.1145/378580.378687.

25 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *ICALP 2018*, pages 65:1–65:16, 2018.

26 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006. doi:10.1145/1147954.1147955.

27 Thathachar S. Jayram and David P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Trans. Algorithms*, 9(3):26, 2013.

28 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS 2010*, pages 41–52, 2010. doi:10.1145/1807085.1807094.

29 Howard J. Karloff. Fast algorithms for approximately counting mismatches. *Inf. Process. Lett.*, 48(2):53–60, 1993. doi:10.1016/0020-0190(93)90177-B.

30 Lev Klebanov and Andrei Yakovlev. How high is the level of technical noise in microarray data? *Biol Direct.*, 2:9, April 2007. doi:10.1186/1745-6150-2-9.

31 Tsvi Kopelowitz and Ely Porat. Breaking the variance: Approximating the Hamming distance in $1/\epsilon$ time per alignment. In *FOCS 2015*, pages 601–613, 2015. doi:10.1109/FOCS.2015.43.

32 Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern Hamming distance. In *SOSA@SODA 2018*, pages 10:1–10:5, 2018. doi:10.4230/OASIcs.SOSA.2018.10.

33 S. R. Kosaraju. Efficient string matching. Manuscript, 1987.

34 Karim Labib, Przemysław Uznański, and Daniel Wolleb-Graf. Hamming distance completeness. In *CPM 2019*, pages 14:1–14:17, 2018. doi:10.4230/LIPIcs.CPM.2019.14.

35 Ohad Lipsky and Ely Porat. L_1 pattern matching lower bound. *Inf. Process. Lett.*, 105(4):141–143, 2008. doi:10.1016/j.ipl.2007.08.011.

36 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12:4:449–461, 1992. doi:10.1007/BF01305237.

37 A. Pavan and Srikanta Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM J. Comput.*, 37(2):359–379, 2007. doi:10.1137/050643672.

38 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *FOCS 2009*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.

39 Jan Studený and Przemysław Uznański. Approximating approximate pattern matching. In *CPM 2019*, volume 128, pages 15:1–15:13, 2019. doi:10.4230/LIPIcs.CPM.2019.15.

40 Vladimir M. Zolotarev. *One-dimensional stable distributions*, volume 65 of *Translations of Mathematical Monographs*. American Mathematical Soc., 1986. Translated from Russian by H.H. McFaden, Translation edited by B. Silver.

6 Omitted proofs

▶ **Lemma 4.** *Let $2 \leq 1/\varepsilon < n$ and $0 \leq p \leq 2$. Any $(1 \pm \varepsilon)$-approximation algorithm that computes the L_p distance between a pattern and a streaming text for each alignment, must use $\Omega(\min(1/\varepsilon^2, n))$ bits of space.*

Proof. Let us first show the lower bound for $p = 0$, i.e., for Hamming distance. We show the lower bound by reduction to a two-party communication complexity problem called GAP-Hamming-distance. In this problem, the two parties, Alice and Bob are given two binary words of length n and a parameter $g = \varepsilon n$, $1 \leq g \leq n/2$. Alice sends Bob a message, and Bob's task is to output 1 if the Hamming distance between his and Alice's word is larger than $n/2 + g$, and zero if it is at most $n/2 - g$. Otherwise, he can output "don't know". By Proposition 4.4 [10], the communication complexity of this problem is $\Omega(\min\{1/\varepsilon^2, n\})$.

We can now show a space lower bound for any $(1\pm\varepsilon)$-approximate algorithm for computing the Hamming distance between the pattern and the text by a standard reduction. Suppose that $2 \leq 1/\varepsilon \leq n$ there is an algorithm that uses $o(\min\{1/\varepsilon^2, n\})$ bits of space. Let P be Alice's word, T Bob's word. After reading P, the algorithm stores all the information about it in $o(\min\{1/\varepsilon^2, n\})$ bits of space. We construct the communication protocol as follows: Alice sends the information about P to Bob. Using it, Bob can continue running the algorithm and compute the approximation of the Hamming distance between P and T. We have thus developed a communication protocol with complexity $o(\min\{1/\varepsilon^2, n\})$, a contradiction.

We can now show the lower bound for $0 < p \leq 2$. We immediately obtain a space lower bound for any $(1 \pm \varepsilon)$-approximate algorithm for computing the p'th moment between the pattern and the text at every alignment. Indeed, on binary words the p'th moment is equal to the Hamming distance for all $0 < p \leq 2$. The lower bound for the L_p distance follows by Observation 5. ◀

▶ **Lemma 8.** *For Z_f as in Algorihtm 7 there is $Z_f \overset{\varepsilon}{=} \|U-V\|_H$ with probability at least $3/4$.*

Proof. Denote $m = \|U-V\|_H$. Consider a fixed value r. Let $I_1, I_2, \ldots, I_n$ be binary variables indicating existence of a mismatch between $\mathsf{hSub}_r(U)$ and V at positions $1, \ldots, n$, so that $X_r = \sum_j I_j$. We observe that $\mathbb{E}[X_r] = m/2^r$ and therefore $\mathbb{E}[Z_r] = m$, because each of the m positions with mismatch between U and V generates a mismatch between $\mathsf{hSub}_r(U)$ and V with probability $1/2^r$.

Furthermore, as the function h in Definition 6 is drawn from a pairwise independent family, there is $\mathrm{Var}[X_r] = \sum_j \mathrm{Var}[I_j] \le \sum_j \mathbb{E}\left[(I_j)^2\right] = \sum_j \mathbb{E}[I_j] = \mathbb{E}[X_r] = m/2^r$. Let $c = \min\{i : \mathbb{E}[X_i] \le k\} = \lceil \log_2\left(\frac{m}{k}\right)\rceil$. By Chebyshev's inequality, we have

$$\Pr[|Z_r - m| \ge 4\sqrt{m2(c+1)}] = \Pr[|X_r - m/2^r| \ge 2^{2+(c+1-r)/2}\sqrt{m/2^r}] \le 1/2^{4+(c+1-r)} \quad (2)$$

We estimate $\Pr[f > c+1] = \Pr[X_{c+1} > k]$. Assume w.l.o.g. that $k \ge 32$. Observe that $m/2^c \le k$, which implies, for $k \ge 32$, $m/2^{c+1} + 4\sqrt{m/2^{c+1}} \le k/2 + 4\sqrt{k/2} \le k$. By Equation 2, there is

$$\Pr[X_{c+1} > k] \le \Pr[X_{c+1} \ge m/2^{c+1} + 4\sqrt{m/2^{c+1}}] \le 1/16.$$

It follows that $\Pr[f > c+1] = \Pr[X_{c+1} > k] \le 1/16$. Hence, we obtain

$$\begin{aligned}
\Pr[|Z_f - m| \ge 4\sqrt{2/k}\cdot m] &\le \Pr[|Z_f - m| \ge 4\sqrt{m2^{c+1}}] \le \\
&\le \Pr[f > c+1] + \sum_{r=0}^{c+1} \Pr[|Z_f - m| \ge 4\sqrt{m2^{c+1}} \text{ and } f = r] \le \\
&\le \Pr[f > c+1] + \sum_{r=0}^{c+1} \Pr[|Z_r - m| \ge 4\sqrt{m2^{c+1}}] \le \\
&\le 1/16 + \sum_{r=1}^{c+1} 1/2^{4+(c+1-r)} < 1/4.
\end{aligned}$$

It follows that we can choose $k = \Theta(1/\varepsilon^2)$ large enough so that $Z_f \overset{\varepsilon}{=} \|U-V\|_H$ with probability $\ge 3/4$. ◀

▶ **Lemma 14.** *For Z_f as in Algorihtm 13 there is $Z_f \overset{\varepsilon}{=} \|U-V\|_1$ with probability $\ge 3/4$.*

Proof. Take some position i and denote for short $a = \mathsf{mSub}_r(U)[i]$ and $b = \mathsf{mSub}_r(V)[i]$ and $c = \frac{U[i]-V[i]}{2^r}$. There is $|a-b| \in \left\{\lfloor|c|\rfloor, \lceil|c|\rceil\right\}$ and $\mathbb{E}[|a-b|] = |c|$. Since $|a-b| - \lfloor|c|\rfloor$ is a 0/1 variable, there is $\mathrm{Var}[|a-b|] = \mathrm{Var}\left[(|a-b| - \lfloor|c|\rfloor)\right] \le \mathbb{E}\left[(|a-b| - \lfloor|c|\rfloor)\right] \le \mathbb{E}[|a-b|]$. Summing for all values of i, we reach that

$$\mathrm{Var}[X_r] = \mathrm{Var}[\|\mathsf{mSub}_r(U) - \mathsf{mSub}_r(V)\|_1] \le \mathbb{E}[\|\mathsf{mSub}_r(U) - \mathsf{mSub}_r(V)\|_1] = \mathbb{E}[X_r].$$

Since we have reached an identical variance bound, the proof follows step-by-step the proof of Lemma 8. ◀

▶ **Theorem 18.** *Given $0 < p < 1$ and $\varepsilon > 0$ there is a word morphism $\varphi : c \in \Sigma \to c_0^{d_0} c_2^{d_2} \ldots c_{t-1}^{d_{t-1}}$ such that:*

1) *$t = \widetilde{\mathcal{O}}(\varepsilon^{-2})$ when $0 < p < 1/2$, $t = \widetilde{\mathcal{O}}(\varepsilon^{-3})$ when $p = 1/2$ and $t = \widetilde{\mathcal{O}}(\sigma^{\frac{2p-1}{1-p}}/\varepsilon^{2+3\cdot\frac{2p-1}{1-p}})$ when $1/2 < p < 1$.*
2) *values of t and $d_0, \ldots, d_{t-1}$ do not depend on c,*

3) *there exists a constant $\alpha = \alpha(p, \varepsilon)$ such that for any two words U, V of length at most n, we have $\|U - V\|_p^p \overset{\varepsilon}{=} \alpha \cdot \|\varphi(U) - \varphi(V)\|_H$ with probability at least $9/10$,*
4) *it is enough for the randomness to be realized by a hash function $r : [t] \to [D]$ from a 4-independent hash function family for some $D = poly(n\sigma\varepsilon^{-1})$, which can be generated from a $\widetilde{\mathcal{O}}(\log \sigma)$ bits size seed.*

Proof. We will consider three cases: $0 < p < 1/2$, $p = 1/2$, and $1/2 < p < 1$.

Case $0 < p < 1/2$. Our plan is to build upon the scheme highlighted earlier in this section. Specifically, we preserve the values of c_i.

Consider a pair of characters c, c'. First, note that $\mathbb{E}[x]$ is an increasing function of $|c - c'|$. From this and Equation 1 we obtain that $\mathbb{E}[x] \overset{\varepsilon}{=} |c - c'|^p \left(\frac{(1+\varepsilon)^p}{(1+\varepsilon)^p - 1} + \frac{1}{(1+\varepsilon)^{1-p} - 1} \right) \varepsilon^{-1}$ for all values of $|c - c'|$.

Second, fix $q = \lceil \frac{1}{1-p} \log_{1+\varepsilon}(\sigma\varepsilon^{-3}) \rceil$ and observe that truncating the sum after the $(q-1)$-th term introduces an additional factor $1 \pm \Theta(\varepsilon)$ to the approximation, since for $c \neq c'$ we have

$$\sum_{i \geq q} d_i \cdot \frac{|c - c'|}{(1+\varepsilon)^i} = \varepsilon^{-1}|c - c'| \frac{((1+\varepsilon)^q)^{p-1}}{1 - (1+\varepsilon)^{p-1}} \leq \frac{\varepsilon^{-1}\sigma}{(1 - (1+\varepsilon)^{p-1})\sigma\varepsilon^{-3}} = \Theta(\varepsilon).$$

We also round d_i down to the nearest integer, which introduces an additional $1 \pm \Theta(\varepsilon)$ relative error, since $\forall_i d_i \geq \varepsilon^{-1}$. Finally, we set $\varphi(c) = c_0^{d_0} \dots c_{q-1}^{d_{q-1}}$. We then have $\mathbb{E}[\|\varphi(c) - \varphi(c')\|_H] = \Theta(\varepsilon^{-2}|c - c'|^p \frac{1}{p(1-p)})$.

To guarantee that the equality holds with probability at least $9/10$ and not just in expectation, we repeat the scheme several times, with independent random seeds. That is, consider morphisms $\varphi_1(c), \varphi_2(c), \dots, \varphi_s(c)$ and define a morphism $\varphi(c) = \varphi_1(c)\varphi_2(c) \dots \varphi_s(c)$ with property:

$$\mathbb{E}[\|\varphi(c) - \varphi(c')\|_H] = s \cdot \mathbb{E}[\|\varphi_i(c) - \varphi_i(c')\|_H] = s \cdot \Theta(\varepsilon^{-2}|c - c'|^p \frac{1}{p(1-p)}).$$

Assume w.l.o.g. that $(1+\varepsilon)^{\ell-1} < |c - c'| \leq (1+\varepsilon)^\ell$. We proceed to bound

$$\begin{aligned}
\mathrm{Var}[\|\varphi(c) - \varphi(c')\|_H] &\leq s \cdot \sum_{i=\ell+1}^{q} (d_i)^2 \cdot \Pr[c_i \neq c_i'] \leq \\
&\leq s \cdot \sum_{i=\ell+1}^{q} \varepsilon^{-2}((1+\varepsilon)^{2p})^i \frac{|c - c'|}{(1+\varepsilon)^i} \leq \\
&\leq s \cdot \varepsilon^{-2}|c - c'| \sum_{i=\ell+1}^{\infty} ((1+\varepsilon)^{2p-1})^i \leq \\
&\leq s \cdot \varepsilon^{-2}|c - c'|^{2p} \frac{(1+\varepsilon)^{2p-1}}{1 - (1+\varepsilon)^{2p-1}} \leq \\
&= s \cdot \mathcal{O}(|c - c'|^{2p}\varepsilon^{-3} \frac{1}{1-2p}).
\end{aligned}$$

We set $s = \Theta(\frac{|c-c'|^{2p}\varepsilon^{-3}(p(p-1))^2}{\varepsilon^2(|c-c'|^p\varepsilon^{-2})^2(1-2p)}) = \mathcal{O}(\varepsilon^{-1}\frac{1}{1-2p})$ for the claim to hold via Chebyshev's inequality. The error probability coming from Chebyshev's inequality can be made arbitrarily small constant by fixing the constant factor in s to be large enough. We finally set $t = sq$.

Case $p = 1/2$. Note that for p, p' such that $|p - p'| \leq \log_\sigma(1+\varepsilon)$ we have $|x|^p \overset{\varepsilon}{=} |x|^{p'}$ for all $-\sigma \leq x \leq \sigma$. We can therefore reduce this case to $p = 1/2 - \log_\sigma(1+\varepsilon)$. However, we have to take into account that the asymptotic growth of t hides $1/(1-2p)$ dependency on p for $0 < p < 1/2$, hence $t = \widetilde{\mathcal{O}}(\varepsilon^{-3})$ for $p = 1/2$.

Case $1/2 < p < 1$. The proof follows the steps of the case $0 < p < 1/2$. We first bound the variance:

$$\begin{aligned}\operatorname{Var}\left[\|\varphi(c) - \varphi(c')\|_H\right] &\leq s \cdot \sum_{i=\ell+1}^{q} (d_i)^2 \cdot \Pr[c_i \neq c'_i] = \\ &= s \cdot \varepsilon^{-2}|c - c'| \sum_{i=\ell+1}^{q} ((1+\varepsilon)^{2p-1})^i = \\ &= s \cdot \mathcal{O}(\varepsilon^{-3}|c - c'|((1+\varepsilon)^q)^{2p-1}) = \\ &= s \cdot \mathcal{O}(\varepsilon^{-3}|c - c'|\sigma^{\frac{2p-1}{1-p}}/\varepsilon^{3\cdot\frac{2p-1}{1-p}}).\end{aligned}$$

We set $s = \Theta\left(\frac{\varepsilon^{-3}|c-c'|\sigma^{\frac{2p-1}{1-p}}/\varepsilon^{3\cdot\frac{2p-1}{1-p}}}{\varepsilon^{-2}|c-c'|^{2p}}\right) = \mathcal{O}(\sigma^{\frac{2p-1}{1-p}}/\varepsilon^{1+3\cdot\frac{2p-1}{1-p}})$, so that by Chebyshev's inequality, the probability of obtaining $\|U - V\|_p^p \overset{\varepsilon}{=} \alpha \cdot \|\varphi(U) - \varphi(V)\|_H$ is an arbitrarily small constant (by setting s to be large enough).

Randomness. The only source of randomness in the description are the values $r_i \in [0,1]$ picked u.a.r. and independently. We note that the values r_i can be picked instead as a finite precision floating-point numbers. Since all the values we are working with are bounded by $\operatorname{poly}(n\sigma\varepsilon^{-1})$, it is enough to set precision accordingly. We also observe that our concentration argument involves only Chebyshev's inequality and thus only the variance and the expected value, so it suffices to require that r_i are 4-wise independent. ◀

▶ **Lemma 21.** *For Z_f as in Algorihtm 20 there is $Z_f \overset{\varepsilon}{=} \|S - Q\|_H$ with probability $\geq 3/4$.*

Proof. Consider a fixed subsampling level r. For simplicity, let $\mathsf{Sub}_r(S) = s_1^{e'_1} s_2^{e'_2} \dots s_m^{e'_m}$ and $\mathsf{Sub}_r(Q) = q_1^{e'_1} q_2^{e'_2} \dots q_m^{e'_m}$. Define a random variable x_i to be the contribution of of $s_i^{e'_i}, q_i^{e'_i}$ to the Hamming distance X_r, i.e.

$$x_i = \|s_i^{e'_i} - q_i^{e'_i}\|_H = e'_i \cdot \|s_i - q_i\|_H.$$

Since $e'_i \in \{\lceil e_i/2^r \rceil, \lfloor e_i/2^r \rfloor\}$ and $\mathbb{E}[e'_i] = e_i/2^r$, we have $\mathbb{E}[x_i] = e_i \cdot \|s_i - q_i\|_H$ and

$$\operatorname{Var}[x_i] = \operatorname{Var}[x_i - \lfloor e_i/2^r \rfloor] \leq \mathbb{E}[x_i - \lfloor e_i/2^r \rfloor] \leq \mathbb{E}[x_i].$$

Summing over all values of i, we reach $\mathbb{E}[X_r] = \|S - Q\|_H$ and $\operatorname{Var}[X_r] \leq \mathbb{E}[X_r]$. These bounds are identical to that of Lemma 8 and we can proceed in a similar fashion to obtain the claim. ◀

▶ **Lemma 28.** *Let $1 < p \leq 2$. For any two vectors X, Y of equal length, $\left|\|X+Y\|_p^p - \|X\|_p^p\right| = \mathcal{O}(\|Y\|_p^p + \|Y\|_p \cdot \|X\|_p^{p-1})$.*

Proof. Consider $x, y \in \mathbb{R}$. If $|x| \geq |y|$, then by Taylor expansion, $|x+y|^p = |x|^p(1+y/|x|)^p = |x|^p(1 + \mathcal{O}(|y/x|)) = |x|^p \pm \mathcal{O}(|y||x|^{p-1})$. If $|x| < |y|$, then $|x+y|^p = \mathcal{O}(|y|^p)$. Thus for any real values, we have

$$|x+y|^p = |x|^p + \mathcal{O}(|y|^p + |y| \cdot |x|^{p-1}).$$

Denote $X = [x_1, x_2, \ldots, x_n]^T$ and $Y = [y_1, y_2, \ldots, y_n]^T$. There is

$$\|X+Y\|_p^p = \sum_i |x_i + y_i|^p = \sum_i |x_i|^p \pm \mathcal{O}\left(\sum_i |y_i|^p + \sum_i |y_i||x_i|^{p-1}\right).$$

Pick $q = p/(p-1)$ so that $1/p + 1/q = 1$. By Hölder's inequality:

$$\sum_i |y_i||x_i|^{p-1} \leq \left(\sum_i |y_i|^p\right)^{1/p} \left(\sum_i |x_i|^{(p-1)q}\right)^{1/q} = \|Y\|_p \|X\|_p^{p-1}. \quad \blacktriangleleft$$

▶ **Lemma 36.** *Given a text T, there is a streaming algorithm that for every position i outputs the Hamming suffix sketch of a word $T[b \cdot k + 1, i]$, where k is the largest integer such that $i - b \cdot k \leq n$. The algorithm takes $\mathcal{O}(dn/b + \log d\sigma n)$ space and $\mathcal{O}(d(1 + n/b^2))$ time per character.*

Proof. We fix the matrix $\mathcal{R}$ and the random coefficients $\alpha_1, \ldots, \alpha_{\lceil n/b \rceil}$ from Definition 34. We do not store $\mathcal{R}$ and α_i explicitly, but generate them using two hash functions drawn at random from a 4-wise independent family. For example, to generate $\mathcal{R}$ we can consider a family of polynomials $2((ax^3 + bx^2 + cx + d \bmod p) \bmod 2) - 1$, with parameters a, b, c, d chosen u.a.r. from the prime field $\mathbb{F}_p$ for $p \geq db$, and α_i can be generated in a similar fashion. This way, we need to store only $\mathcal{O}(\log(d\sigma b) + \log(n/b)) = \mathcal{O}(\log d\sigma n)$ random bits that define the coefficients of two polynomials to generate $\mathcal{R}$ and α_i.

We process the text T by blocks $B_1, B_2, \ldots$ of length b. For each block B_k we compute its sketch using the matrix $\mathcal{R}$. That is, at the beginning of each block we initialize its sketch as a zero vector of length d. When a new character $T[i]$ of a block B_k arrives, we compute and add $[M[1, i \cdot b\sigma + T[i]], M[2, i \cdot b\sigma + T[i]], \ldots, M[d, i \cdot b\sigma + T[i]]]^T$ to the sketch, which takes $\mathcal{O}(d)$ time. We store the sketch of B_k until the block $B_{k+\lceil n/b \rceil}$ and use it to compute the suffix sketches for the positions in this block.

Consider now a block $B_{k+\lceil n/b \rceil}$. We first compute the suffix sketch for the position $b \cdot (k + \lceil n/b \rceil)$, which is the position preceding the block $B_{k+\lceil n/b \rceil}$. The suffix sketch for it is simply a linear combination of the sketches of the blocks $B_{k+\lceil n/b \rceil - 1}, B_{k+\lceil n/b \rceil - 2}, \ldots, B_k$ with coefficients $\alpha_1, \ldots, \alpha_{\lceil n/b \rceil - 1}$. Since each sketch is a vector of length d, we can compute the linear combination in $\mathcal{O}(dn/b)$ time. To make this computation time-efficient, we start it b positions before position $b \cdot (k + \lceil n/b \rceil)$ arrives, and de-amortise the computation over these b positions. This way, we use only $\mathcal{O}(dn/b^2)$ time per character.

Now, using the suffix sketch for the position $b \cdot (k + \lceil n/b \rceil)$, we can compute the suffix sketches for all positions in the block $B_{k+\lceil n/b \rceil}$ one-by-one, using only $\mathcal{O}(d)$ time per character: When a new character $T[i]$ arrives, we add $[\alpha_{\lceil n/b \rceil} M[1, i \cdot b\sigma + T[i]], \alpha_{\lceil n/b \rceil} M[2, i \cdot b\sigma + T[i]], \ldots, \alpha_{\lceil n/b \rceil} M[d, i \cdot b\sigma + T[i]]]^T$ to the suffix sketch to update it.

Note that at any time we store $\mathcal{O}(n/b)$ sketches of the blocks, so the algorithm uses $\mathcal{O}(dn/b + \log d\sigma n)$ space in total. $\blacktriangleleft$

▶ **Lemma 42.** *Given a text T, there is a streaming algorithm that for every position i outputs the Manhattan suffix sketch of a word $T[b \cdot k + 1, i]$, where k is the smallest integer such that $i - b \cdot k \leq n$. The algorithm takes $\mathcal{O}(d \cdot (n/b) + \log^2 \sigma)$ space, and $\mathcal{O}(d(1 + n/b^2) \cdot \log^3(bd\sigma))$ time per character.*

Proof. The proof mirrors the proof of Lemma 36, and we describe the key elements. We fix the random coefficients $\alpha_1, \ldots, \alpha_{\lceil n/b \rceil}$ and the hash function h from Definition 40. As previously, we do not store the coefficients α_i explicitly, but generate them using a hash

function drawn at random from a 4-wise independent family. The matrix $\mathcal{R}$ is already defined by h, with the following parameters: it requires $\mathcal{O}(\log^2(bd\sigma))$ bits of seed, and range-sum queries are answered in time $\mathcal{O}(\log^3(bd\sigma))$.

In the sketching of blocks, we proceed in the same manner, except that when a new character $T[i]$ of a block B_k arrives, we compute and add $\sum_{0\leq j<T[i]}[M[1, i\cdot b\sigma+j], \ldots, M[d, i\cdot b\sigma + j]]^T = \alpha_i \cdot [g(b\sigma, b\sigma + T[i]; \xi), g(2b\sigma, 2b\sigma + T[i]; \xi), \ldots, g(d \cdot b\sigma, d \cdot b\sigma + T[i]; \xi)]^T$ to the sketch, which takes $\mathcal{O}(d \cdot \log^3(bd\sigma))$ time ($\log^3(bd\sigma)$ times slower as the corresponding step in Lemma 36).

Consider now a block $B_{k+\lceil n/b\rceil}$. When a new character $T[i]$ arrives, we update the suffix sketch by adding $\alpha_{\lceil n/b\rceil} \cdot [g(b\sigma, b\sigma + T[i]; \xi), g(2b\sigma, 2b\sigma + T[i]; \xi), \ldots, g(d \cdot b\sigma, d \cdot b\sigma + T[i]; \xi)]^T$ to it.

All of the operations are $\mathcal{O}(\log^3(bd\sigma))$ time slower than the corresponding steps in Lemma 36, and the memory complexity is increased by the seed size $\mathcal{O}(\log^2(bd\sigma))$ term ($\log^2 b$ and $\log^2 d$ terms get absorbed). ◀