Approximating Text-To-Pattern Distance via Dimensionality Reduction

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

Abstract

Text-to-pattern distance is a fundamental problem in string matching, where given a pattern of length m and a text of length n, over an integer alphabet, we are asked to compute the distance between pattern and the text at every location. The distance function can be e.g. Hamming distance or ℓ_p distance for some parameter $p > 0$. Almost all state-of-the-art exact and approximate algorithms developed in the past ~ 40 years were using FFT as a black-box. In this work we present $\widetilde{\mathcal{O}}(n/\varepsilon^2)$ time algorithms for $(1 \pm \varepsilon)$-approximation of ℓ_2 distances, and $\widetilde{\mathcal{O}}(n/\varepsilon^3)$ algorithm for approximation of Hamming and ℓ_1 distances, all without use of FFT. This is independent to the very recent development by Chan et al. [STOC 2020], where $\mathcal{O}(n/\varepsilon^2)$ algorithm for Hamming distances not using FFT was presented – although their algorithm is much more "combinatorial", our techniques apply to other norms than Hamming.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximate Pattern Matching, ℓ_2 Distance, ℓ_1 Distance, Hamming Distance, Approximation Algorithms, Combinatorial Algorithms

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.29

Related Version A full version of the paper is available at [30], `https://arxiv.org/abs/2002.03459`.

Funding Supported by Polish National Science Centre grant 2019/33/B/ST6/00298.

1 Introduction

Text-to-pattern distance is a generalization of a classical pattern matching by incorporating the notion of similarity (or dissimilarity) between pattern and locations of text. The problem is defined in a following way: for a particular distance function between words (interpreted as vectors), given a pattern of length m and a text of length n, we are asked to output distance between the pattern and every m-substring of the text. Taking e.g. distance to be Hamming distance, we are essentially outputting number of mismatches in a classical pattern matching question (that is, not only detecting exact matches, but also counting how far pattern is to from being located in a text, at every position). Such a formulation, for a constant-size alphabet, was first considered by Fischer and Paterson in [12]. The algorithm of [12] 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 poly(n) size alphabets by Abrahamson in [1, 21] with $\mathcal{O}(n\sqrt{m \log m})$ run-time.

The lack of progress in Hamming text-to-pattern distance complexity sparked interest in searching for relaxations of the problem, with a hope for reaching linear (or almost linear) run-time. There are essentially two takes on this. First consists of approximation algorithms. Until very recently, the fastest known $(1 \pm \varepsilon)$-approximation algorithm for computing the Hamming distances was by Karloff [18]. The algorithm uses 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. Later Kopelowitz and Porat [19] 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 in Kopelowitz and Porat [20], with alternative formulation by Uznański and Studený [28].

Second widely considered way of relaxing exact text-to-pattern distance is to report exactly only the values not exceeding certain threshold value k, the so-called k-mismatch problem. The very first solution to the k-mismatch problem was shown by Landau and Vishkin in [23] working in $\mathcal{O}(nk)$ time, using essentially a very combinatorial approach of taking $\mathcal{O}(1)$ time per mismatch per alignment using LCP queries. This initiated a series of improvements to the complexity, with algorithms of complexity $\mathcal{O}(n\sqrt{k \log k})$ and $\mathcal{O}((k^3 \log k + m) \cdot n/m)$ by Amir et al. in [3], later improved to $\mathcal{O}((k^2 \log k + m \text{ poly} \log m) \cdot n/m)$ by Clifford et al. [8] and finally $\mathcal{O}((m \log^2 m \log |\Sigma| + k\sqrt{m \log m}) \cdot n/m)$ by Gawrychowski and Uznański [13] (and following poly-log improvements by Chan et al. in [5]).

Moving beyond counting mismatches, we consider ℓ_1 distances, where we consider text and pattern over integer alphabet, and distance is sum of position-wise absolute differences. Using techniques similar to Hamming distances, the $\mathcal{O}(n\sqrt{m \log m})$ complexity algorithms were developed by Clifford et al. and Amir et al. in [6, 4] for reporting all ℓ_1 distances. It is a major open problem whether near-linear time algorithm, or even $\mathcal{O}(n^{3/2-\varepsilon})$ time algorithms, are possible for such problems. A conditional lower bound was shown by Clifford in [7], via a reduction from matrix multiplication. This means that existence of combinatorial algorithm with $\mathcal{O}(n^{3/2-\varepsilon})$ run-time solving the problem for Hamming distances implies combinatorial algorithms for Boolean matrix multiplication with $\mathcal{O}(n^{3-\delta})$ run-time, which existence is unlikely. Looking for unconditional bounds, we can state this as a lower-bound of $\Omega(n^{\omega/2})$ for Hamming distances pattern matching, where $2 \le \omega < 2.373$ is the matrix multiplication exponent. Later, complexity of pattern matching under Hamming distance and under ℓ_1 distance was proven to be identical (up to poly-logarithmic terms), see Labib et al. and Lipsky et al. [22, 24].

Once again, existence of such lower-bound spurs interest in approximation algorithm for ℓ_1 distances. Lipsky and Porat [25] gave a deterministic algorithm with a run time of $\mathcal{O}(\frac{n}{\varepsilon^2} \log m \log U)$, while later Gawrychowski and Uznański [13] have improved the complexity to a (randomized) $\mathcal{O}(\frac{n}{\varepsilon} \log^2 n \log m \log U)$, where U is the maximal integer value on the input. Later [28] has shown that such complexity is in fact achievable (up to poly-log factors) with a deterministic solution.

Considering other norms, we mention following results. First, that for any $p > 0$ there is ℓ_p distance $(1 \pm \varepsilon)$-approximated algorithm running in $\widetilde{\mathcal{O}}(n/\varepsilon)$ time by [28]. More importantly, for specific case of $p = 2$ (or more generally, constant, positive even integer values of p) the exact problem reduces to computation of convolution, as observed by [25].

Text-to-pattern distance via convolution

Consider the case of computing ℓ_2 distances. We are computing output array $O[1 \mathinner{..} n-m+1]$ such that $O[i] = \sum_j (T[i+j] - P[j])^2$. However, this is equivalent to computing, for every i simultaneously, the value of $\sum_j T[i+j]^2 + \sum_j P[j]^2 - 2\sum_j T[i+j]P[j]$. While the terms $\sum_j T[i+j]^2$ and $\sum_j P[j]^2$ can be easily precomputed in $\mathcal{O}(n)$ time, we observe (following [25]) that $\sum_j T[i+j]P[j]$ is essentially a convolution. Indeed, let P^R denote reverse string to P. Then

$$\sum_j T[i+j]P[j] = \sum_j T[i+j]P^R[m+1-j] = \sum_{j+k=m+1+i} T[j]P'[k] = (T \circ P^R)[m+1+i].$$

Since $T \circ P^R$ can be computed efficiently this provides a very strong tool in constructing text-to-pattern distance algorithms. Almost all of the discussed results use convolution as a black-box. For example, by appropriate binary encoding we can compute using a single convolution the number of Hamming mismatches generated by a single letter $c \in \Sigma$, which is a crucial observation leading to computation of exact Hamming distances in $\mathcal{O}(n\sqrt{n \log n})$ time. Other results rely on projecting large alphabets into smaller ones, e.g. [18, 20, 28].

Convolution over integers is computed by FFT in $\mathcal{O}(n \log n)$ time. This requires actual embedding of integers into field, e.g. $\mathbb{F}_p$ or $\mathbb{C}$. This comes at a cost, if e.g. we were to consider text-to-pattern distance over (non-integer) alphabets that admit only field operations, e.g. matrices or geometric points. Convolution can be computed using "simpler" set of operations, that is just with ring operations in e.g. $\mathbb{Z}_p$ using Toom-Cook multiplication [29], which is a generalization of famous divide-and-conquer Karatsuba's algorithm [17]. This however comes at a cost, with Toom-Cook algorithm taking $\mathcal{O}(n2^{\sqrt{2 \log n}} \log n)$ time, and increased complexity of the algorithm.

Computing convolution comes with another string attached – it is inefficient to compute/sketch in the streaming setting. All of the efficient streaming text-to-pattern distance algorithms [5, 8, 9, 10, 26, 14, 27] use some form of sketching and are actually avoiding convolution computation. The reason for this is that convolution does not admit efficient sketching schemes other than with additive error, that is any algorithm based on convolution is supposed to make the same error of estimation in small and large distance regime.

Our results

We present approximation algorithm for computing the ℓ_2 text-to-pattern distance in $\widetilde{\mathcal{O}}(n/\varepsilon^2)$ time, where $\widetilde{\mathcal{O}}$ hides poly $\log n$ terms. Our algorithm is convolution-avoiding, and in fact it uses mostly additions and subtractions in its core part (some non-ring operations are necessary for output-scaling and hashing). We thus claim our algorithm to be more "combinatorial", in the sense that it does not rely on field embedding and FFT computation. Our algorithm is also first non-trivial algorithm for text-to-pattern distance computation with other norms (than Hamming, which was presented recently in [5]).

▶ **Theorem 1.** *Text-to-pattern ℓ_2 distances can be approximated by an algorithm using only basic arithmetic operations and not using convolution. The approximation is $1 \pm \varepsilon$ multiplicative with high probability, computed in $\mathcal{O}(\frac{n \log^3 n}{\varepsilon^2})$ time.*

This mirrors the recent development of [5] where a combinatorial algorithm for Hamming distances was presented with $\mathcal{O}(n/\varepsilon^2)$ run-time. However, our techniques are general enough so that we can construct algorithm for ℓ_1 norm (and Hamming), however with $\widetilde{\mathcal{O}}(n/\varepsilon^3)$ run-time.

▶ **Theorem 2.** *Text-to-pattern Hamming distances can be approximated by an algorithm using only basic arithmetic operations and not using convolution. The approximation is $1 \pm \varepsilon$ multiplicative with high probability, computed in $\mathcal{O}(\frac{n \log^4 n}{\varepsilon^3})$ time.*

▶ **Theorem 3.** *Text-to-pattern ℓ_1 distances over alphabet $[u]$ for some constant $u = poly(n)$ can be approximated by an algorithm using only basic arithmetic operations and not using convolution. The approximation is $1 \pm \varepsilon$ multiplicative with high probability, computed in $\mathcal{O}(\frac{n \log^2 n(\log^2 n + \log^4 u)}{\varepsilon^3})$ time.*

We present two novel techniques, to our knowledge never used previously in this setting. First, we show that a "mild" dimensionality reduction (linear map reducing from dimension $2d$ to d, while preserving ℓ_2 norm) can be used to repeatedly compress word, and produce sketches for its every m-subword. Second, we show an approximate embedding of ℓ_1 space into ℓ_2^2, that can be efficiently computed. We believe our techniques are of independent interest, both to stringology and general algorithmic communities.

2 Definitions and preliminaries

Distance between strings

Let $X = x_1x_2 \dots x_n$ and $Y = y_1y_2 \dots y_n$ be two strings. We define their ℓ_2 distance as

$$\|X - Y\| = \left(\sum_i |x_i - y_i|^2\right)^{1/2}.$$

More generally, for any $p > 0$, we define their ℓ_p distance as

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

Particularly, the ℓ_1 distance is known as the *Manhattan distance*. By a slight abuse of notation, we define the ℓ_0 (Hamming distance) to be

$$\|X - Y\|_0 = \sum_i |x_i - y_i|^0 = |\{i : x_i \neq y_i\}|,$$

where $x^0 = 1$ when $x \neq 0$ and $0^0 = 0$.

Text-to-pattern distance

For text $T = t_1t_2 \dots t_n$ and pattern $P = p_1p_2 \dots p_m$, the text-to-pattern d-distance is defined as an array S_d such that, for every i, $S_d[i] = d(T[i+1 \,..\, i+m], P)$. Thus, for ℓ_p distance $S_{\ell_p}[i] = \left(\sum_{j=1}^m |t_{i+j} - p_j|^p\right)^{1/p}$, while for Hamming distance $S_{\mathrm{HAM}}[i] = |\{j : t_{i+j} \neq p_j\}|$. Then $(1 \pm \varepsilon)$-approximated distance is defined as an array S_ε such that, for every i, $(1-\varepsilon) \cdot S_d[i] \leq S_\varepsilon[i] \leq (1+\varepsilon) \cdot S_d[i]$.

3 Sketching via dimensionality reduction

Sketching is a tool in algorithm design, where a large object is summarized succinctly, so that some particular property is approximately preserved and some predefined operations/queries are still supported. Our interest lies on sketches that preserve ℓ_2 distances, for which we use the standard tools from dimensionality reduction.

▶ **Theorem 4** (Johnson-Lindenstrauss [15]). *Let $P \subseteq \mathbb{R}^m$ be of size m. Then for some $d = \mathcal{O}(\frac{\log m}{\varepsilon^2})$ there is linear map $A \in \mathbb{R}^{d \times m}$ such that*

$$\forall_{x,y \in P} \|Ax - Ay\| = (1 \pm \varepsilon)\|x - y\|.$$

A map that preserves ℓ_2 distances is useful. Our goal is to construct a linear map such that we can apply the map to P and to every m-substring of T simultaneously and computationally efficiently. For this, we need to actually use constructive version of Johnson-Lindenstrauss lemma.

▶ **Theorem 5** (Achlioptas [2]). *Consider a probability distribution $\mathcal{D}$ over matrices $\mathbb{R}^{m\times d}$ defined as follow so that each matrix entry is either -1 or 1 independently and uniformly at random. Then for any $x \in \mathbb{R}^m$ there is*

$$\Pr_{A\sim\mathcal{D}}\left(\frac{1}{\sqrt{d}}\|Ax\| = (1\pm\varepsilon)\|x\|\right) \geq 1-\delta$$

if only $d = \mathcal{O}(\frac{\log \delta^{-1}}{\varepsilon^2})$ is large enough.

Computing such dimension-reduction naively takes $\mathcal{O}(md)$ time. However better constructions are possible.

▶ **Theorem 6** (Sparse JL, c.f. [11, 16]). *There is probability distribution $\mathcal{S}$ over matrices of dimension $d \times m$ with elements from $\{-1, 0, 1\}$, for large enough $d = \mathcal{O}(\frac{\log \delta^{-1}}{\varepsilon^2})$, such that each column has only $s = \mathcal{O}(d\varepsilon)$ non-zero elements and for any vector $x \in \mathbb{R}^m$ there is*

$$\Pr_{A\sim\mathcal{S}}\left(\frac{1}{\sqrt{s}}\|Ax\| = (1\pm\varepsilon)\|x\|\right) \geq 1-\delta.$$

Such matrices can be easily drawn from the distribution by selecting the s positions in each column independently at random and then filling them uniformly at random with $\{-1, 1\}$. The advantage of this is that single dimensionality reduction operation is computed in $\mathcal{O}(sm)$ time which is ε^{-1} factor faster than for dense matrices.

We now state the take-away from this section, which is our main technical tool to be used in the following.

▶ **Corollary 7.** *For $d = \mathcal{O}(\frac{\log n}{\varepsilon^2})$ large enough there is a probability distribution $\mathcal{F}$ of linear maps $\varphi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ such that:*

1. *$\varphi(x, y) = A_0x + A_1y$ can be evaluated in $\mathcal{O}(d^2\varepsilon) = \mathcal{O}(\frac{\log^2 n}{\varepsilon^3})$ time,*
2. *$\Pr_{\varphi\sim\mathcal{F}}\left(\|\varphi(x, y)\|^2 = (1 \pm \varepsilon)(\|x\|^2 + \|y\|^2)\right) = 1 - n^{-\Omega(1)}$,*
3. *both A_0 and A_1 are $\{-\frac{1}{\sqrt{s}}, 0, \frac{1}{\sqrt{s}}\}$-matrices where $s = \mathcal{O}(d\varepsilon)$ is the sparsity of each column of A_0 and A_1.*

4 Algorithm for ℓ_2 distances.

We first use Corollary 7 to construct dimensionality reduction with guarantees similar to Johnson-Lindenstrauss (reducing dimension n to dimension $\widetilde{\mathcal{O}}(\varepsilon^{-2})$). In the following we assume that $d = \mathcal{O}(\frac{\log n}{\varepsilon^2})$ is large enough. We show a procedure which assumes that m is divisible by d, and denote $s = \frac{m}{d}$. We assume s is a power of two, and if the case is otherwise, we can always pad input with enough zeroes at the end (we can do this, since extra zeroes have no effect on the output of linear map). We also denote $k = \log_2 s$.
We then have the following

▶ **Theorem 8.** *Given input $x \in \mathbb{R}^m$, and $\varepsilon \leq \frac{1}{k}$, procedure* SingleSketch *outputs $v \in \mathbb{R}^d$ such that*

$$\|v\| = (1 \pm \mathcal{O}(k\varepsilon))\|x\|$$

with high probability, in time $\mathcal{O}(\frac{m \log n}{\varepsilon})$. The map $x \to v$ is linear.

Proof. We first bound the stretch. Denote by

$$\alpha_i = \sum_j \|v_j^{(i)}\|^2.$$

Algorithm 1 At each level i, we partition its vectors into 2^{k-i} pairs, and compress each pair using φ_i producing vectors for level $i+1$.

```
1: Input: $x \in \mathbb{R}^m$.
2: Output: $v \in \mathbb{R}^d$.
3: procedure SingleSketch($x$)
4:     Pick $k$ fully independent maps $\varphi_1, \ldots, \varphi_k$ as in Corollary 7.
5:     Partition input $x = (x_1, \ldots, x_m)$ into $s$ vectors $v_1^{(0)}, \ldots, v_s^{(0)}$ where $v_i^{(0)} \leftarrow (x_{d\cdot(i-1)+1}, \ldots, x_{d\cdot i})$.
6:     for $i \leftarrow 1 .. k$ do
7:         for $j \leftarrow 1 .. 2^{k-i}$ do
8:             $v_j^{(i)} \leftarrow \varphi_i(v_{2j-1}^{(i-1)}, v_{2j}^{(i-1)})$
9:     return $v = v_1^{(k)}$.
```

Naturally,

$$\alpha_0 = \sum_j \|v_j^{(0)}\|^2 = \sum_{j=1}^{s} (x_{d\cdot(j-1)+1}^2 + \ldots + x_{d\cdot j}^2) = \sum_{j=1}^{n} x_j^2 = \|x\|^2.$$

Moreover, by Corollary 7

$$\alpha_i = \sum_{j=1}^{2^{k-i}} \|v_j^{(i)}\|^2 = \sum_{j=1}^{2^{k-i}} (1 \pm \varepsilon)(\|v_{2j-1}^{(i-1)}\|^2 + \|v_{2j}^{(i-1)}\|^2)$$
$$= (1 \pm \varepsilon) \sum_{j=1}^{2^{k-i+1}} \|v_j^{(i-1)}\|^2 = (1 \pm \varepsilon)\alpha_{i-1}$$

We could apply Corollary 7 at this step since for any usage of map φ_i, its inputs are independent from actual choice of φ_i (e.g. are result of processing x and $\varphi_1, \ldots, \varphi_{i-1}$). Then we have $\|v\|^2 = \alpha_k = (1 \pm \varepsilon)^k \alpha_0 = (1 \pm \varepsilon)^k \|x\|^2$. Since $\varepsilon \le \frac{1}{k}$, the claimed bound follows.

We then observe that the map is linear, since every building step of the map is linear. The total number of times we apply one of $\varphi_1, \ldots, \varphi_k$ is $\mathcal{O}(m/d)$, so the total run-time is $\mathcal{O}(\frac{m}{d} d^2 \varepsilon)$. ◀

We then extend the algorithm to a scenario where for an input word (vector) $x \in \mathbb{R}^n$ we compute the same dimensionality reduction for all m-subwords of x that start at all the positions divisible by d. In the following we assume that d divides n, and denote $t = \frac{n-m}{d} + 1$ to be the number of such m-subwords. If its not the case, input can be padded with enough zeroes at the end.

▶ **Theorem 9.** *Given input $x \in \mathbb{R}^n$, denote by $y_1, \ldots, y_t \in \mathbb{R}^m$ vectors such that $y_i = (x_{1+(i-1)d}, \ldots, x_{m+(i-1)d})$. For $\varepsilon \le \frac{1}{k}$ procedure AllSketch outputs $v_1, \ldots, v_t \in \mathbb{R}^d$ such that*

$$\|v_j\| = (1 \pm \mathcal{O}(k\varepsilon))\|y_j\|$$

with high probability, in time $\mathcal{O}(\frac{n \log^2 n}{\varepsilon})$. Moreover, the map $y_i \to v_i$ is linear and identical to map from Theorem 8.

Algorithm 2

```
1: Input: $x \in \mathbb{R}^n$.
2: Output: $v_1, \dots, v_t \in \mathbb{R}^d$ for $t = \frac{n-m}{d} + 1$.
3: procedure AllSketch($x$)
4:     Let $\varphi_1, \dots, \varphi_k$ be $k$ fully independent maps used in procedure SingleSketch.
5:     Partition input $x = (x_1, \dots, x_n)$ into $n/d$ vectors $v_1^{(0)}, \dots, v_{\frac{n}{d}}^{(0)}$ where $v_i^{(0)} \leftarrow (x_{d\cdot(i-1)+1}, \dots, x_{d\cdot i})$.
6:     for $i \leftarrow 1 \,..\, k$ do
7:         for $j \leftarrow 1 \,..\, (\frac{n}{d} - 2^i + 1)$ do
8:             $v_j^{(i)} \leftarrow \varphi_i(v_j^{(i-1)}, v_{j+2^{i-1}}^{(i-1)})$
9:     return $v_1^{(k)}, \dots, v_t^{(k)}$.
```

Proof. The proof follows from inductive observation that $\|v_j^{(i)}\|^2 = (1 \pm \varepsilon)^i (\|v_j^{(0)}\|^2 + \dots \|v_{j+2^i-1}^{(0)}\|^2)$, which results in

$$\begin{aligned}\|v_j\|^2 &= (1 \pm \varepsilon)^k \sum_{i=1}^{s} \|v_{j+i}^{(0)}\|^2\\ &= (1 \pm \varepsilon)^k \sum_{i=1}^{m} \|x_{i+(j-1)d}\|^2\\ &= (1 \pm \varepsilon)^k \|y_j\|^2.\end{aligned}$$

The rest of the proof follows reasoning from Theorem 8. ◀

▶ **Theorem 1.** *Text-to-pattern ℓ_2 distances can be approximated by an algorithm using only basic arithmetic operations and not using convolution. The approximation is $1 \pm \varepsilon$ multiplicative with high probability, computed in $\mathcal{O}(\frac{n \log^3 n}{\varepsilon^2})$ time.*

Proof. First, we note that for simplicity we compute $(\ell_2)^2$ distances since they are additive when taken under concatenation of inputs (unlike ℓ_2), that is $\|x \circ y - u \circ v\|^2 = \|x-u\|^2 + \|y-v\|^2$ for equal length x, u and equal length y, v.

We then assume w.l.o.g. that n is divisible by d. We then observe that contribution of any fragment of pattern to distance at every text location can be computed naively in $\mathcal{O}(c \cdot n)$ time where c is fragment length. We are thus safe to discard any suffix of pattern of length $\mathcal{O}(d)$ as this time is absorbed in total computation time. So we fix $h = \mathcal{O}(\log n/\varepsilon)$ and assume w.l.o.g. that $m' = m - 2h$ is divisible by d.

We denote by $\varepsilon' = \Omega(\varepsilon/\log n)$ such value that guarantees $(1 \pm \varepsilon)$-approximation in Theorem 8 and Theorem 9. First, assume for simplicity that $\frac{m'}{d}$ is a power of two. We then consider $P_0, \dots, P_h$, the $(h+1)$ distinct m'-substrings of P, and for each we run procedure SingleSketch on each of them, so by Theorem 8 we compute their sketches in total $\mathcal{O}(\frac{m \log n}{\varepsilon'} h)$ time. Similarly, for text T we run AllSketch $\frac{d}{h}$ times to compute sketches of all m'-substrings of T starting at positions $1, h+1, 2h+1, \dots$. By Theorem 9 this takes $\mathcal{O}(\frac{n \log^2 n}{\varepsilon'} \cdot \frac{d}{h})$ time. Both steps take thus $\mathcal{O}(\frac{n \log^3 n}{\varepsilon^2})$ time, and maps used to compute sketches in both steps are linear.

We now observe that for any starting position t, the substring $T[t \,..\, (t+m'-1)]$ can be partitioned into $T_1 = T[t \,..\, t_1]$, $T_2 = T[t_1+1 \,..\, t_2]$ and $T_3 = T[t_2+1 \,..\, (t+m'-1)]$, where length of T_1 and T_3 is at most $2h$, length of T_3 is m' and t_1 and t_2 are multiples of h. We then compute the distances between corresponding fragments of T and P as follows (where we consider corresponding partitioning of P into P_1, P_2 and P_3): computing $\|T_1 - P_1\|^2$ and $\|T_3 - P_3\|^2$ takes $\mathcal{O}(h)$ each ($\mathcal{O}(nh)$ in total for all alignments), while $(1 \pm \varepsilon)$ approximating $\|T_2 - P_2\|^2$ follows from pre-computed sketches.

We now discuss the general case when $\frac{m'}{d}$ is not a power of two. However we then observe that m' can be represented as $m' = d(2^{i_1} + \ldots + 2^{i_s})$ where $s \leq \log n$. And so the necessary computation require actually querying s different sketches for fragments of length $d \cdot 2^{i_\ell}$. To avoid unnecessary $\mathcal{O}(\log n)$ overhead in time (and repeating running the preprocessing steps $\log n$ times for many various lengths of fragments) we observe that all the necessary sketches are already computed as temporary values in procedures SingleSketch and AllSketch. ◀

5 Hamming and ℓ_1 distances.

We now briefly discuss how to use our framework for approximating other norms. We first recall the classical result by [18].

▶ **Lemma 10** ([18]). *Let $d = \mathcal{O}(\log n/\varepsilon^2)$ be large enough. Consider $\mu : \Sigma \to \{0,1\}^d$ where each $\varphi(c)$ is chosen uniformly and independently at random. Then*

$$\forall_{c_1 \neq c_2} \|\mu(c_1) - \mu(c_2)\|^2 = (1 \pm \varepsilon) \cdot \frac{d}{2}$$

with high probability.

We note that we assumed that the dimension $\mathcal{O}(\log n/\varepsilon^2)$ of map μ matches value of $d = \mathcal{O}(\log n/\varepsilon^2)$ from dimensionality-reductions in previous section. This can be easily ensured w.l.o.g. as we can always either pad with extra zeroes each image of μ mapping, or add extra null coordinates to dimensionality reduction. Extending the mapping from letters to words, that is for $w = c_1 \ldots c_k \in \Sigma^*$ denote $\mu(w) = \mu(c_1) \ldots \mu(c_k)$, we have a corollary:

▶ **Corollary 11.** *For μ as in Lemma 10, and any two words $u, v \in \Sigma^n$, there is*

$$\|\mu(u) - \mu(v)\|^2 = (1 \pm \varepsilon) \cdot \frac{d}{2} \|u - v\|_0$$

with high probability.

This allows us to estimate Hamming distance between words from ℓ_2^2 distance between the respective embeddings, which are of length $\mathcal{O}(\frac{n \log n}{\varepsilon^2})$.

▶ **Theorem 2.** *Text-to-pattern Hamming distances can be approximated by an algorithm using only basic arithmetic operations and not using convolution. The approximation is $1 \pm \varepsilon$ multiplicative with high probability, computed in $\mathcal{O}(\frac{n \log^4 n}{\varepsilon^3})$ time.*

Proof. By Corollary 11 it is enough to estimate the ℓ_2^2 text-to-pattern distance between embedded words $\mu(P)$ and $\mu(T)$ at starting positions $1, d+1, 2d+1, \ldots$. We use procedure SingleSketch to compute sketch of $\mu(P)$, and procedure AllSketch to compute sketch of every (dm)-substring of $\varphi(T)$ starting at positions $1, d+1, 2d+1, \ldots$. Former takes $\mathcal{O}(\frac{n \log^2 n}{\varepsilon^2 \varepsilon'})$ time, and latter takes $\mathcal{O}(\frac{n \log^3 n}{\varepsilon^2 \varepsilon'})$ time, where we set $\varepsilon' = \Omega(\varepsilon/k)$ so that error from sketching accumulates to $1 \pm \mathcal{O}(\varepsilon)$ in total. All in all this gives $\mathcal{O}(\frac{n \log^4 n}{\varepsilon^3})$ time algorithm. ◀

We now proceed to ℓ_1 distances. Our goal is to construct a mapping $f : [u] \to \{0,1\}^d$ that embeds ℓ_1 into ℓ_2^2 approximately. That is, we require $\forall_{a,b \in [u]} |a - b| \sim (1 \pm \varepsilon)\|f(a) - f(b)\|^2$ where $\sim$ hides constant factors. The existence of such map can be easily shown: (i) Take exact map $f_1 : [u] \to \{0,1\}^u$ defined as $f_1(a) = 1^a 0^{u-a}$, (ii) Take any ℓ_2 dimensionality-reduction map $f_2 : \{0,1\}^u \to \{0,1\}^d$, (iii) set $f = f_2 \circ f_1$. However, our goal is to compute such f faster than in time proportional to universe size u. We do it by running first a preprocessing phase, and then a fast computation procedure.

Algorithm 3

```
1: procedure PREPROCESS($u$)
2:     Pick $\log(u/d)$ fully independent maps $\varphi'_1, \dots, \varphi'_{\log(u/d)}$ as in Corollary 7.
3:     $s_0 \leftarrow (1, 1, \dots, 1) \in \mathbb{R}^d$.
4:     for $i \leftarrow 1$ .. $\log(u/d)$ do
5:         $s_i \leftarrow \varphi'_i(s_{i-1}, s_{i-1})$
6: procedure PROJECT($x \in [u]$, $c$)
7:     if $c = 0$ then
8:         return $(\underbrace{1, 1, \dots, 1}_{x}, \underbrace{0, \dots, 0}_{d-x})$
9:     else if $x < \frac{1}{2} d \cdot 2^c$ then
10:        return $\varphi'_c(\textsc{Project}(x, c-1), (0, \dots, 0))$
11:    else
12:        return $\varphi'_c(s_{c-1}, \textsc{Project}(x - \frac{1}{2} d \cdot 2^c, c-1))$
```

▶ **Lemma 12.** *$\psi : x \to \textsc{Project}(x, \log(u/d))$ represents a linear map $[u] \to \mathbb{R}^d$ that embeds approximately ℓ_1 to ℓ_2^2, that is*

$$|x - y| = (1 \pm \mathcal{O}(\varepsilon \log u)) \|\psi(x) - \psi(y)\|^2$$

with high probability. Moreover, ψ takes $\mathcal{O}(\frac{\log^2 n \log u}{\varepsilon^3})$ time to evaluate.

Proof. Let us define informally $\pi_i = \varphi'_i(\varphi'_{i-1}(\dots,\dots), \varphi'_{i-1}(\dots,\dots))$ to be unfolded version of φ', that is a linear map $\mathbb{R}^{d \cdot 2^i} \to \mathbb{R}^d$. Formally $\pi_0 = \mathrm{id}$, and for $x = (x_1, \dots, x_{d \cdot 2^i})$, defining

$$\pi_i((x_1, \dots, x_{d \cdot 2^i})) = \varphi'_i(\pi_{i-1}(x_{\text{left}}), \pi_{i-1}(x_{\text{right}})),$$

where $x_{\text{left}} = (x_1, \dots, x_{d \cdot 2^{i-1}})$, $x_{\text{right}} = (x_{d \cdot 2^{i-1}+1}, \dots, x_{d \cdot 2^i})$.

We now observe that $s_i = \pi_i((\underbrace{1, \dots, 1}_{2^i d}))$ and then (by induction)

$$\textsc{Project}(x, i) = \pi_i((\underbrace{1, 1, \dots, 1}_{x}, \underbrace{0, \dots, 0}_{2^i d - x})).$$

Inductively, each iteration $1, .., \log(u/d)$ results in extra multiplicative $(1 \pm \varepsilon)$ distortion. Computation time is dominated by applications of $\varphi'_1, \dots, \varphi'_{\log(u/d)}$, both in the preprocessing time and the evaluation time. Since each linear map φ'_i is applied in time $\mathcal{O}(\frac{\log^2 n}{\varepsilon^3})$, the time complexity bound follows. ◀

▶ **Theorem 3.** *Text-to-pattern ℓ_1 distances over alphabet $[u]$ for some constant $u = poly(n)$ can be approximated by an algorithm using only basic arithmetic operations and not using convolution. The approximation is $1 \pm \varepsilon$ multiplicative with high probability, computed in $\mathcal{O}(\frac{n \log^2 n (\log^2 n + \log^4 u)}{\varepsilon^3})$ time.*

Proof. We use Lemma 12 to reduce the problem to estimating ℓ_2^2 text-to-pattern distance between $\psi(P)$ and $\psi(T)$ at starting positions $1, d+1, 2d+1, \ldots$. We use procedure SingleSketch to compute sketch of $\mu(P)$, and procedure AllSketch to compute sketch of every (dm)-substring of $\varphi(T)$ starting at selected positions. Denote by $\varepsilon' = \Omega(\varepsilon/k)$ the stretch constant in procedures SingleSketch and AllSketch, and by $\varepsilon'' = \Omega(\varepsilon/\log u)$ the stretch constant in procedures Project and Preprocess. The total run-time of AllSketch is then $\mathcal{O}(\frac{n\log^3 n}{\varepsilon^2\varepsilon'}) = \mathcal{O}(\frac{n\log^4 n}{\varepsilon^3})$ and total run-time of computing $\psi(T)$ and $\psi(P)$ is $\mathcal{O}(\frac{n\log^2 n\log u}{(\varepsilon'')^3}) = \mathcal{O}(\frac{n\log^2 n\log^4 u}{\varepsilon^3})$. ◀

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 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.

4 Amihood Amir, Ohad Lipsky, Ely Porat, and Julia Umanski. Approximate matching in the L_1 metric. In *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings*, pages 91–103, 2005. doi:10.1007/11496656_9.

5 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern hamming distances. In *STOC*, 2020 (to appear).

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

7 Raphaël Clifford. Matrix multiplication and pattern matching under Hamming norm. http://www.cs.bris.ac.uk/Research/Algorithms/events/BAD09/BAD09/Talks/BAD09-Hammingnotes.pdf. Retrieved March 2017.

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

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

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

11 Michael B. Cohen, T. S. Jayram, and Jelani Nelson. Simple analyses of the sparse johnson-lindenstrauss transform. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 15:1–15:9, 2018. doi:10.4230/OASIcs.SOSA.2018.15.

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

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

14 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *ICALP*, pages 65:1–65:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

15 William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space, 1984.

16 Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, 2014. doi:10.1145/2559902.

17 Anatolii Karatsuba. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, pages 595–596, 1963.

18 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.

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

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

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

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

23 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986. `doi:10.1016/0304-3975(86)90178-7`.

24 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`.

25 Ohad Lipsky and Ely Porat. Approximate pattern matching with the L_1, L_2 and L_∞ metrics. *Algorithmica*, 60(2):335–348, 2011. `doi:10.1007/s00453-009-9345-9`.

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

27 Tatiana Starikovskaya, Michal Svagerka, and Przemysław Uznański. L_p pattern matching in a stream. *CoRR*, abs/1907.04405, 2019. `arXiv:1907.04405`.

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

29 AL Toom. The complexity of a scheme of functional elements simulating the multiplication of integers. In *Doklady Akademii Nauk*, volume 150(3), pages 496–498. Russian Academy of Sciences, 1963.

30 Przemysław Uznański. Approximating text-to-pattern distance via dimensionality reduction. *CoRR*, abs/2002.03459, 2020. `arXiv:2002.03459`.