Towards Unified Approximate Pattern Matching for Hamming and L_1 Distance

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Przemysław Uznański
Department of Computer Science, ETH Zürich, Switzerland
przemyslaw.uznanski@inf.ethz.ch

Abstract

Computing the distance between a given pattern of length n and a text of length m is defined as calculating, for every m-substring of the text, the distance between the pattern and the substring. This naturally generalizes the standard notion of exact pattern matching to incorporate dissimilarity score. For both Hamming and L_1 distance only relatively slow $\widetilde{\mathcal{O}}(n\sqrt{m})$ solutions are known for this generalization. This can be overcome by relaxing the question. For Hamming distance, the usual relaxation is to consider the k-bounded variant, where distances exceeding k are reported as ∞, while for L_1 distance asking for a $(1 \pm \varepsilon)$-approximation seems more natural. For k-bounded Hamming distance, Amir et al. [J. Algorithms 2004] showed an $\widetilde{\mathcal{O}}(n\sqrt{k})$ time algorithm, and Clifford et al. [SODA 2016] designed an $\widetilde{\mathcal{O}}((m + k^2) \cdot n/m)$ time solution. We provide a smooth time trade-off between these bounds by exhibiting an $\widetilde{\mathcal{O}}((m + k\sqrt{m}) \cdot n/m)$ time algorithm. We complement the trade-off with a matching conditional lower bound, showing that a significantly faster *combinatorial* algorithm is not possible, unless the combinatorial matrix multiplication conjecture fails. We also exhibit a series of reductions that together allow us to achieve essentially the same complexity for k-bounded L_1 distance. Finally, for $(1 \pm \varepsilon)$-approximate L_1 distance, the running time of the best previously known algorithm of Lipsky and Porat [Algorithmica 2011] was $\widetilde{\mathcal{O}}(\varepsilon^{-2}n)$. We improve this to $\widetilde{\mathcal{O}}(\varepsilon^{-1}n)$, thus essentially matching the complexity of the best known algorithm for $(1 \pm \varepsilon)$-approximate Hamming distance.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases approximate pattern matching, conditional lower bounds, L_1 distance, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.62

1 Introduction

The basic question in algorithms on strings is pattern matching, which asks for reporting (or detecting) occurrences of a given pattern P of length m in a text T of length n. A particularly relevant variant of this fundamental question is approximate pattern matching, where the goal is to detect fragments of the text that are *similar* to the pattern. This can be restated as computing the text-to-pattern distance, defined as the distance between P and every m-substring of T. If both P and T are over an integer alphabet Σ, two most natural distance functions are Hamming and L_1. Abrahamson [1] showed how to compute the text-to-pattern Hamming distance with a clever application of boolean convolution: a single convolution can be used to count matches generated by a particular letter in time close to linear, and by carefully partitioning the letters into frequent and non-frequent the overall running time can be guaranteed to be $\mathcal{O}(n\sqrt{m \log m})$. With a somewhat similar approach, the same complexity can be achieved for L_1 distance [5]. Naturally, one would like to design an $\widetilde{\mathcal{O}}(n)$ time algorithm. However, an unpublished result attributed to Indyk [6] is that any significant improvement for Hamming distance by a *combinatorial* algorithm implies a significant improvement for *combinatorial matrix multiplication*, which is believed to be very hard. Later, a direct reduction from Hamming distance to L_1 distance was shown [14], as well as a reverse reduction [8] together with a full suite of two-way reductions between these two metrics and other functions, linking the corresponding complexities (up to poly-logarithmic factors).

A natural relaxation of computing the text-to-pattern distance is to ask for its $(1 \pm \varepsilon)$-approximation. Karloff [10] showed how to use random $\Sigma \to \{0,1\}$ projections together with boolean convolution to approximate the Hamming distance in $\mathcal{O}(\varepsilon^{-2} n \log^3 m)$. The ε^{-2} dependency was believed to be tight, given the similarly looking lower bound for sketching Hamming distance [16, 9, 4], but Porat and Kopelowitz [11] refuted this by exhibiting a (complicated) $\mathcal{O}(\varepsilon^{-1} n \log \varepsilon^{-1} \log n \log m \log |\Sigma|)$ time algorithm. Later, they presented a much simpler solution with a slightly better complexity of $\mathcal{O}(\varepsilon^{-1} n \log n \log m)$ [12]. For approximating the L_1 distance, it was only known how to achieve $\mathcal{O}(\varepsilon^{-2} n \log m \log |\Sigma|)$ time complexity [15], and breaking the ε^{-2} barrier was open.

Another natural relaxation is to cap the distances at k, that is, return ∞ if the distance exceeds k. We will call this variant the k-bounded distance. For k-bounded Hamming distance, the classical solution by Landau and Vishkin [13] works in $\mathcal{O}(nk)$ time by checking every m-substring with $k{+}1$ constant-time longest common extension queries (also known as "kangaroo jumps"). Later, Amir et al. [2] improved this to $\mathcal{O}(n\sqrt{k \log k})$ using boolean convolution similarly as in the classical algorithm of Abrahamson, and also showed an $\mathcal{O}((k^3 \log k + m) \cdot n/m)$ time algorithm. Later, Clifford et al. [7] introduced a new repertoire of tools allowing them to further improve the latter complexity to $\mathcal{O}((k^2 \log k + m \operatorname{polylog} m) \cdot n/m)$. In particular, this is near linear-time for $k = \mathcal{O}(\sqrt{m})$. At a very high level, the improvement was obtained by partitioning both the pattern and the text into $\mathcal{O}(k)$ subpatterns and subtexts, such that the total number of blocks in their run-length encoding is small. This reduces the original problem to $\mathcal{O}(k^2)$ instances of pattern matching with mismatches on run-length encoded inputs, which can be solved in $\widetilde{\mathcal{O}}(k^2)$ total time, leading to an $\widetilde{\mathcal{O}}((k^2{+}m) \cdot n/m)$ time algorithm. For k-bounded L_1 distance, only an $\mathcal{O}(n\sqrt{k \log k})$ time algorithm was known [3], and designing an almost linear-time algorithm for polynomially large (in m) values of k was open.

Our results. We provide a smooth transition between the $\widetilde{\mathcal{O}}(n\sqrt{k})$ time algorithm of Amir et al. [2] and the $\widetilde{\mathcal{O}}((m + k^2) \cdot n/m)$ solution given by Clifford et al. [7] for k-bounded Hamming distance. This is obtained by reducing k-bounded Hamming distance to $\mathcal{O}(k)$-RLE Hamming distance, in which the run-length encodings of both the pattern and the text consist of $\mathcal{O}(k)$ blocks, and then designing an efficient algorithm for the latter.

▶ **Theorem 1.** *There is a deterministic algorithm that outputs k-bounded text-to-pattern Hamming distance in time $\mathcal{O}((m \log^2 m \log |\Sigma| + k\sqrt{m \log m}) \cdot n/m)$.*

The complexity from Theorem 1 matches the previous solutions at the extreme points $k = \mathcal{O}(\sqrt{m})$ and $k = \Omega(m)$, but provides a better trade-off in-between. See Figure 1. Furthermore, we prove that this trade-off is essentially the best possible. More precisely, we complement the algorithm with a matching conditional lower bound, showing that a significantly faster *combinatorial* algorithm is not possible, unless the popular combinatorial matrix multiplication conjecture fails.

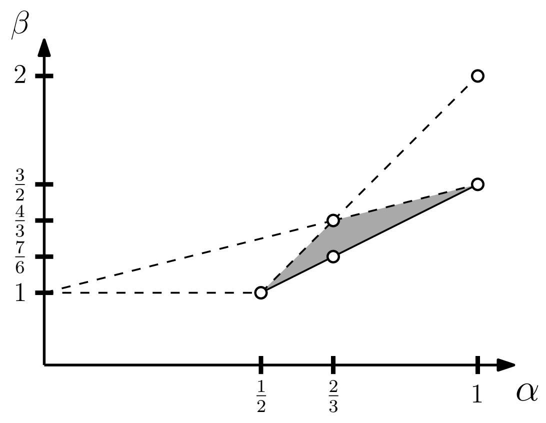


Figure 1 Running time $\mathcal{O}(m^{\beta})$ of algorithm from Theorem 1 on instances with $n = \Theta(m)$ and $k = m^{\alpha}$. Previous algorithms are represented by dashed lines and our algorithm is represented by the solid line. For example, for $k = \Theta(m^{2/3})$ we improve the complexity from $\widetilde{\mathcal{O}}(m^{4/3})$ to $\widetilde{\mathcal{O}}(m^{7/6})$.

▶ **Theorem 2.** *For any positive $\varepsilon, \alpha, \kappa$, such that $\frac{1}{2}\alpha \le \kappa \le \alpha \le 1$ there is no* combinatorial *algorithm solving pattern matching with $k = \Theta(n^{\kappa})$ mismatches in time $\mathcal{O}((k\sqrt{m})^{1-\varepsilon} \cdot n/m)$ for a text of length n and a pattern of length $m = \Theta(n^{\alpha})$, unless the combinatorial matrix multiplication conjecture fails.*

Next, we move to computing L_1 distance. For computing $(1 \pm \varepsilon)$-approximate text-to-pattern L_1 distance we are able to break the quadratic dependency on $1/\varepsilon$ present in the previous algorithms by designing an $\widetilde{\mathcal{O}}(\varepsilon^{-1}n)$ solution, thus making the complexities of the best known algorithms for $(1 \pm \varepsilon)$-approximate text-to-pattern L_1 distance and Hamming distance essentially equal.

▶ **Theorem 3.** *There is a randomized Monte Carlo algorithm that outputs $(1 \pm \varepsilon)$-approximation of L_1 distance in time $\mathcal{O}(\varepsilon^{-1} n \log^3 n \log m)$. The algorithm is correct with high probability.*

For k-bounded L_1 distance we are able to obtain essentially the same trade-off as for k-bounded Hamming distance.

▶ **Theorem 4.** *There is a deterministic algorithm that outputs k-bounded text-to-pattern L_1 distance in time $\mathcal{O}((m \log^3 m + m \log^2 n + k\sqrt{m \log m} \cdot \log^2 n) \cdot n/m)$.*

In fact, instead of designing a new algorithm for k-bounded L_1 distance we exhibit a series of generic reductions that together allow us to reduce computing k-bounded L_1 distance to computing k-bounded Hamming distance.

▶ **Theorem 5.** *Let $T(n, m, k)$ be the complexity for k-bounded text-to-pattern Hamming distance. Then k-bounded L_1 text-to-pattern distance can be computed in time $\mathcal{O}(n \log^3 m + T(m, m, \mathcal{O}(k)) \cdot \log^2 n \cdot n/m)$.*

Thus, if k-bounded Hamming distance can be computed in time $\widetilde{\mathcal{O}}(n + (k\sqrt{m})^{1-\delta} \cdot n/m)$ for some $\delta \ge 0$ then k-bounded L_1 distance can be computed in time $\widetilde{\mathcal{O}}(n + (k\sqrt{m})^{1-\delta} \cdot n/m)$ as well. Together with a reduction from k-RLE to k-bounded Hamming distance this gives us a clear (although not yet complete) picture of connections between k-bounded and k-RLE text-to-pattern distance under Hamming and L_1 distance summarized in Figure 2.

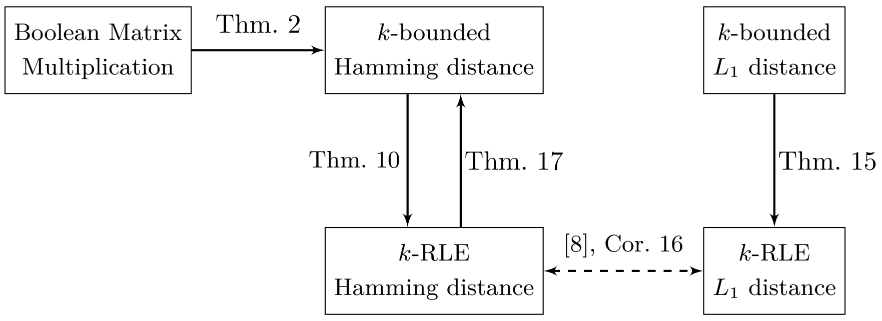


Figure 2 Existing (dashed lines) and new (solid lines) reductions.

Overview of the techniques. Our algorithm for k-bounded Hamming distance (and then also L_1 distance), is a refinement of the approach of Clifford et al. [7]. The general idea is to first consider the periodic structure of the pattern. If the pattern is not very periodic, then m-substrings of the text with a small distance to the pattern cannot occur too often. This allows us to filter the possible occurrences with an algorithm for $(1 \pm \varepsilon)$-approximate Hamming distance and then manually verify the remaining possibilities. Otherwise, the problem can be reduced to multiple smaller instances, in which both the pattern and the text are highly compressible, i.e. their run-length encodings consist of only $\mathcal{O}(k)$ blocks. The first new insight is that, instead of many small instances, it is possible to obtain a single instance of $\mathcal{O}(k)$-bounded Hamming distance, in which the run-length encoding of both the pattern and the text consist of $\mathcal{O}(k)$ blocks. The second observation is that, because the pattern and the text are still of length $\mathcal{O}(m)$, it (again) makes sense to partition the letters into frequent and non-frequent, except that now the threshold is defined with respect to the number of blocks of the pattern containing that letter. For non-frequent letters, we produce a compact representation of their contribution by iterating through every block of the pattern and every block of the text. For frequent letters, we essentially uncompress the corresponding fragments and run the classical convolution.

Our algorithm for $(1 \pm \varepsilon)$-approximate L_1 distance is based on generalized weighted mismatches, similarly as in the previous work [15]: given an arbitrary weight function $\sigma : \Sigma \times \Sigma \to \mathbb{Z}$, we output an array S_σ such that $S[i] = \sum_{j=1}^{m} \sigma(t_{i+j}, p_j)$. This can be computed as follows. For every letter $c \in \Sigma$, construct a new text T^c by setting $T^c[i] = 1$ if $t_i = c$ and $T^c[i] = 0$ otherwise. Similarly, construct a new pattern P^c such that $P^c[i] = \sigma(c, p_i)$. Then, $\sigma(t_{i+j}, p_j) = \sum_{c \in \Sigma} T^c[i+j] \cdot P^c[j]$, so S_σ can be computed with $|\Sigma|$ convolutions in $\mathcal{O}(|\Sigma| n \log m)$ time.

To connect the complexities of computing the text-to-pattern Hamming and L_1 distance we use the recently introduced notion of linearity preserving reductions [8] as a formalization of previously existing reductions between metrics (cf. [14]). The main idea is that in order to show a reduction between two pattern-matching problems, one can represent them as a $(+, \diamond)$ convolution and a $(+, \Box)$ convolution, and show how to represent $\Box$ as a linear combination of many copies of $\diamond$.

2 Preliminaries

Distance between strings. Let $X = x_1 x_2 \dots x_n$ and $Y = y_1 y_2 \dots y_n$ be two strings over an integer alphabet $[M]$, for some $M = \text{poly}(n)$. We define their L_1 distance as $L_1(X, Y) = \sum_i |x_i - y_i|$, and their Hamming distance as $\text{Ham}(X, Y) = |\{i : x_i \neq y_i\}|$.

Text-to-pattern distance. The text-to-pattern distance between a text $T = t_1 t_2 \dots t_n$ and a pattern $P = p_1 p_2 \dots p_m$ is defined as an array S such that, for every i, $S[i] = d(T[i+1 \mathinner{.\,.} i+m], P)$. Thus, for L_1 distance $S[i] = \sum_{j=1}^{m} |t_{i+j} - p_j|$, while for Hamming distance $S[i] = |\{j \in \{1, \dots, m\} : t[i+j] \neq p[j]\}|$. Then, $(1 \pm \varepsilon)$-approximate distance is an array S_ε such that, for every i, $(1-\varepsilon) \cdot S[i] \le S_\varepsilon[i] \le (1+\varepsilon) \cdot S[i]$. Finally, k-bounded distance is an array S_k such that, for every i, $S_k[i] = S[i]$ when $S[i] \le k$ and $S_k[i] = \infty$ otherwise. Finally, in k-RLE distance we assume that the run-length encoding of both the pattern and the text consist of $\mathcal{O}(k)$ blocks, that is, they contain only $\mathcal{O}(k)$ maximal runs of identical letters.

Model. We assume the standard word RAM model, in which arithmetic operations on integers from $[M]$ take constant time.

Linearity preserving reductions. Say that we want to show a reduction between binary operators $\diamond$ and $\Box$. In order for such a reduction to be relevant in the convolutional setting, it needs to be of a specific form. Let t be a fixed integer called the size of the reduction, and suppose there exist integer coefficients $\alpha_1, \dots, \alpha_t$, and functions $f_1, \dots, f_t, g_1, \dots, g_t$ such that, for any x and y, there is $x \mathbin{\Box} y = \sum_{\ell=1}^{t} \alpha_\ell \cdot (f_\ell(x) \diamond g_\ell(y))$. Then $(+, \Box)$-convolution of T and P is computed as a linear combination of $(+, \diamond)$-convolutions $f_\ell(T)$ with $g_\ell(P)$, where $f(X) = f(x_1) f(x_2) \dots f(x_n)$ for $X = x_1 x_2 \dots x_n$. That is, a following equality holds $\sum_{i+j=k} x[i] \Box y[j] = \sum_{\ell=1}^{t} \alpha_\ell \cdot \left(\sum_{i+j=k} x[i] \diamond y[j] \right)$.

3 k-bounded Distance

The goal of this section is to prove Theorem 1. We first show how to reduce k-bounded Hamming distance to $\mathcal{O}(n/m)$ instances of $\mathcal{O}(k)$-RLE Hamming distance on inputs of length $\mathcal{O}(m)$. We start with the standard trick of reducing the original problem to $\lceil n/m \rceil$ instances with pattern P of length m and text T of length $2m$ and work with such formulation from now on. Thus, now we need to reduce one such instance to an instance of $\mathcal{O}(k)$-RLE Hamming distance on inputs of length $\mathcal{O}(k)$.

We now highlight the kernelization technique of Clifford et al. [7]. An integer $\pi > 0$ is an x-period of a string $S[1, m]$ if $\mathrm{Ham}(S[\pi+1, m], S[1, m-\pi]) \le x$ (cf. Definition 1 in [7]). Note that compared to the original formulation, we drop the condition that π is minimal from the definition.

▶ **Lemma 6** (Fact 3.1 in [7]). *If the minimal $2x$-period of the pattern is ℓ, then the starting positions of any two occurrences with x mismatches of the pattern are at distance at least ℓ.*

The first step of the algorithm is to determine the minimal $\mathcal{O}(k)$-period of the pattern. More specifically, we run the $(1+\varepsilon)$-approximate algorithm of Karloff [10] with $\varepsilon = 1$ matching the pattern P against itself. Since Karloff's algorithm is convolution-based, it can be adapted to computing mismatches for partial alignments, that is, between suffixes and prefixes of equal length of the same string. This takes $\mathcal{O}(m \log^2 m \log |\Sigma|)$ time and, by looking at the approximate outputs for offsets not larger than k, allows us to distinguish between two cases: (1) every $4k$-period of the pattern is at least k, or (2) there is an $8k$-period of the pattern that is at most k. Then we run the appropriate algorithm as described below.

No small $4k$-period. We again run Karloff's algorithm with $\varepsilon = 1$, but now we match the pattern with the text. We look for positions i where the 2-approximate algorithm reports at most k mismatches, meaning that $\mathrm{Ham}(P, T[i, i+m-1]) \le 2k$. By Lemma 6, there

$T' = \texttt{hokuspokusopensezame}$

h	s	u	e	z
o	p	s	n	a
k	o	o	s	m
u	k	p	e	e

s	u	e	z	#
p	s	n	a	#
o	o	s	m	#
k	p	e	e	#

$P = \texttt{abracadabrab}$

a	c	b	$	$
b	a	r	$	$
r	d	a	$	$
a	a	b	$	$

$$T^\star = \texttt{hsuez opsna koosm ukpee suez\# psna\# oosm\# kpee\#}$$
$$P^\star = \texttt{acb\$\$ bar\$\$ rda\$\$ aab\$\$}$$

Figure 3 Example of rearranging the text and the pattern with $\ell = 4$.

are $\mathcal{O}(m/k)$ such positions, and we can safely discard all the others. Then, we test every such position using the "kangaroo jumps" technique of Landau and Vishkin [13], using $\mathcal{O}(k)$ constant-time operations per position, in total $\mathcal{O}(m)$ time.

Small $8k$-period. Let $\ell \le k$ be any $8k$-period of the pattern. For a string S and $1 \le i \le \ell$, let $\{S\}_{\ell,i} = S[i]S[i+\ell]S[i+2\ell]\dots$ up until end of S. We denote by $\{S\}_\ell$ an ℓ-encoding of S, that is the string $\{S\}_{\ell,1}\{S\}_{\ell,2}\cdots\{S\}_{\ell,\ell-1},\{S\}_{\ell,\ell}$. Let $\mathrm{runs}(S)$ be the number of runs in S. Denote $\mathrm{runs}_\ell(S) = \sum_{i=1}^{\ell} \mathrm{runs}(\{S\}_{\ell,i})$, and observe that it upperbounds the number of runs in $\{S\}_\ell$.

▶ **Lemma 7** (Lemma 6.1 in [7]). *If P has an $8k$-period ℓ for $\ell \le k$, then $\mathrm{runs}_\ell(P) \le 9k$.*

We proceed with the kernelization argument. Let T_L be the longest suffix of $T[1,m]$ such that $\mathrm{runs}_\ell(T_L) \le 11k$. Similarly, let T_R be the longest prefix of $T[m+1,2m]$ such that $\mathrm{runs}_\ell(T_R) \le 11k$. Let $T' = T_L T_R$. Obviously, $\mathrm{runs}_\ell(T') \le 22k$.

▶ **Lemma 8** (Lemma 6.2 in [7]). *Every $T[i, i+m-1]$ that is an occurrence of P with k mismatches is fully contained in T'.*

Thus we see that k-mismatch pattern matching is reduced to a kernel where the ℓ-encoding of both the text and the pattern have few runs, that is, they both compress well with RLE.

From now on assume that both T' and P are of lengths divisible by ℓ. If this is not the case, we can pad them separately with at most $\ell - 1 < k$ characters each, not changing the complexity of our solution. Let m_1 and m_2 be integers such that $m_1 \cdot \ell = |T'|$ and $m_2 \cdot \ell = |P|$, $m_1 \ge m_2$.

We rearrange both P and T' to take advantage of their regular structure. That is, we define $T^\star = \{T'\}_\ell\{T''\}_\ell$, where $T'' = T'[\ell+1, m_1 \cdot \ell]\ \#^\ell$. Observe that $T^\star$ is a string of length $2m_1 \cdot \ell$, composed first of m_1 blocks of the form $T'[i]T'[i+\ell]\dots T'[i+(m_1-1)\ell]$ for $1 \le i \le \ell$, and then of m_1 blocks of the form $T'[i+\ell]\dots T'[i+(m_1-1)\ell]\ \#$. Similarly, we define $P^\star = \{P\ \$^{(m_1-m_2)\ell}\}_\ell$. Again we observe that $P^\star$ is the string of length $m_1 \cdot \ell$, composed of blocks of the form $P[i]P[i+\ell]\dots P[i+(m_2-1)\ell]\ \$^{m_1-m_2}$ for $1 \le i \le \ell$. An example of this reduction is presented in Figure 3.

Next we show that $T^\star$ and $P^\star$ preserve the Hamming distance between P and any m-substring of T' in the following sense:

▶ **Lemma 9.** *For any integer $0 \le \alpha \le (m_1 - m_2)\ell$, let $x = \lfloor \alpha/\ell \rfloor$ and $y = \alpha \bmod \ell$. Let $\beta = x + y \cdot m_1$. Then*

$$\mathrm{Ham}(T'[\alpha+1, \alpha+m_2\cdot\ell], P) = \mathrm{Ham}(T^\star[\beta+1, \beta+m_1\cdot\ell], P^\star) - (m_1-m_2)\cdot\ell.$$

Proof. Observe that

$$\mathrm{Ham}(T'[\alpha+1,\alpha+m_2\cdot\ell],P) = \sum_{i=0}^{m_2-1}\sum_{j=1}^{\ell}\delta(T'[x\ell+y+i\ell+j],P[i\ell+j]), \tag{1}$$

where δ is indicator of character inequality. Observe that $P[i\ell+j]) = P^\star[i+j\cdot m_1]$, for $1\le j\le \ell-y$ there is $T'[x\ell+y+i\ell+j] = T^\star[(x+i)+(y+j)m_1]$, and for $\ell-y<j\le\ell$ there is $T'[x\ell+y+i\ell+j] = T''[(x+i)\ell+(y+j-\ell)] = T^\star[(x+i)+(y+j-\ell)m_1+\ell m_1] = T^\star[(x+i)+(y+j)m_1]$. Additionally, for $m_2\le i<m_1$, $P^\star[i+j\cdot m_1] = \$$, which always generates a mismatch with any character in $T^\star$. Thus

$$(1) = \sum_{i=0}^{m_2-1}\sum_{j=1}^{\ell}\delta(T^\star[(x+i)+(y+j)m_1],P^\star[i+j\cdot m_1]) =$$
$$= -(m_1-m_2)\ell + \sum_{i=0}^{m_1-1}\sum_{j=1}^{\ell}\delta(T^\star[(x+i)+(y+j)m_1],P^\star[i+j\cdot m_1]). \qquad ◀$$

We see that it is enough to find all occurrences of $P^\star$ in $T^\star$ with $(k+(m_1-m_2)\cdot\ell)$ mismatches, where $k+(m_1-m_2)\cdot\ell\le 2k$, $|P^\star| = |T'|\le m$ and $|T^\star| = 2|T'|\le 2m$. Additionally, $\mathrm{runs}(P^\star)\le 9k+\ell\le 10k$ and $\mathrm{runs}(T^\star)\le 22k+\ell\le 23k$. This gives us the following theorem.

▶ **Theorem 10.** *k-bounded text-to-pattern Hamming distance reduces in $\mathcal{O}(n\log^3 m)$ time to $\mathcal{O}(n/m)$ instances of $\mathcal{O}(k)$-RLE text-to-pattern Hamming distance on inputs of length $\mathcal{O}(m)$.*

Now we describe how to solve an instance of $\mathcal{O}(k)$-RLE Hamming distance.

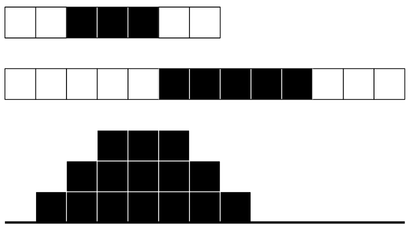
▶ **Lemma 11.** *There is a deterministic algorithm that outputs k-RLE Hamming text-to-pattern distance on inputs of length $\mathcal{O}(m)$ in time $\mathcal{O}(m+k\sqrt{m\log m})$.*

Proof. Consider a letter $c\in\Sigma$. For a string S, we denote by $\mathrm{runs}(S,c)$ the number of runs in S consisting of occurrences of c. Fix a parameter t. Call a letter c such that $\mathrm{runs}(P^\star,c)>t$ a heavy letter, and otherwise call it light. Now we describe how to count the number of mismatches for each type of letters. This is reminiscent to a trick originally used by Abrahmson [1] and later refined by Amir et al. [2].

Heavy letters. For every heavy letter c separately we use a convolution scheme. Since both $P^\star$ and $T^\star$ are of size $\mathcal{O}(m)$, this takes time $\mathcal{O}(m\log m)$ per every such letter. Since $\sum_{c\in\Sigma}\mathrm{runs}(P^\star,c) = \mathrm{runs}(P^\star)\le 10k$, there are $\mathcal{O}(k/t)$ heavy letters, making the total time $\mathcal{O}(mk/t\cdot\log m)$.

Light letters. First, we preprocess $P^\star$, and for every light letter c we compute a list of runs consisting of occurrences of c. Our goal is to compute the array $A[0,|T^\star|-|P^\star|]$, where $A[i]$ counts the number of matching occurrences of light letters in $T^\star[i+1,i+|P^\star|]$ and $P^\star$.

We scan $T^\star$, and for every run of a particular light letter, we iterate through the precomputed list of runs of this letter in $P^\star$. Observe that, given a run of the same letter in $P^\star$ and in $T^\star$, denoted $T^\star[u,v]$ and $P^\star[y,z]$, respectively, their corresponding matches can be seen as a piecewise linear function. More precisely, for all integers $u\le i\le v$ and $y\le j\le z$, we need to increase $A[i-j]$ by one. To see that we can process pair of runs in constant time, we work with discrete derivative, instead of original arrays.



$$DA = \boxed{1}\boxed{0}\boxed{0}\boxed{-1}$$

$$DB = \boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{-1}$$

$$D^2(A \cdot B) = \boxed{1}\boxed{0}\boxed{0}\boxed{-1}\boxed{0}\boxed{-1}\boxed{0}\boxed{0}\boxed{1}$$

Figure 4 Left: a run in the pattern and a run in the text (both represented by black boxes) consisting of the same character and a histogram of the matches they generate. Right: first derivatives of the indicator arrays and second derivative of the match array, without padding zeroes.

Given a sequence F, we define its discrete derivative DF as follows: $(DF)[i] = F[i] - F[i-1]$. Correspondingly, if we consider a generating function $F(x) = \sum_i F[i]x^i$, then $(DF)(x) = F(x) \cdot (1-x)$ (for convenience, we assume that arrays are indexed from $-\infty$ to ∞).

Now consider indicator sequences $T_{u,v}[i] = \mathbf{1}(u \le i \le v)$ and $P_{y,z}[j] = \mathbf{1}(-z \le j \le -y)$. To perform the update, we set $A[i+j] \mathrel{+}= T_{u,v}[i] \cdot P_{y,z}[j]$ for all i, j, or simpler using generating functions:

$$A(x) \mathrel{+}= T_{u,v}(x) \cdot P_{y,z}(x), \tag{2}$$

where $T_{u,v}(x) = \sum_{i=u}^{v} x^i$ and $P_{y,z}(x) = \sum_{j=y}^{z} x^{-j}$. However, we observe that $DT_{u,v}$ and $DP_{y,z}$ have particularly simple forms: $DT_{u,v}(x) = x^u - x^{v+1}$ and $DP_{y,z}(x) = x^{-z} - x^{-y+1}$. Thus it is easier to maintain second derivative of A, and (2) becomes:

$$D^2A(x) \mathrel{+}= x^{u-z} - x^{v-z+1} - x^{u-y+1} + x^{v-y+2}.$$

All in all, we can maintain D^2A in constant time per pair of runs, or in $\mathcal{O}(k \cdot t)$ total time, since every list of runs is of length at most t, and there are at most $23k$ runs in $T^\star$. Additionally, in $\mathcal{O}(m)$ time we can compute $A[0]$ and $A[1]$, allowing us to recover all other $A[i]$s from the formula $A[i] = (D^2A)[i] + 2A[i-1] - A[i-2]$.

Setting $t = \sqrt{m \log m}$ gives the total running time $\mathcal{O}(k\sqrt{m \log m})$ in both cases as claimed. ◀

Combining the reduction from Theorem 10 with Lemma 11 gives us Theorem 1.

4 Lower Bound for k-bounded Hamming Distance

In this section, we present a conditional lower bound for computing k-bounded Hamming distance. The main idea expands upon the proof attributed to Indyk [6], except that we use rectangular matrices instead of square, and use the padding accordingly. We pad using the same character in both text and pattern, increasing the number of mismatches only by a factor of 2.

Recall the combinatorial matrix multiplication conjecture stating that, for any $\varepsilon > 0$, there is no *combinatorial*[1] algorithm for multiplying two $n \times n$ boolean matrices working in time $\mathcal{O}(n^{3-\varepsilon})$. The following formulation is equivalent to this conjecture.

[1] It is not clear what does combinatorial mean *precisely*. However, FFT and so boolean convolution often used in algorithms on strings are considered not to be combinatorial.

▶ **Conjecture 12** (Combinatorial matrix multiplication)**.** *For any $\alpha, \beta, \gamma, \varepsilon > 0$, there is no combinatorial algorithm for multiplying an $n^\alpha \times n^\beta$ matrix with an $n^\beta \times n^\gamma$ matrix in time $\mathcal{O}(n^{\alpha+\beta+\gamma-\varepsilon})$.*

The equivalence can be seen by simply cutting the matrices into square blocks (in one direction) or in rectangular blocks (in the other direction).

Now, consider two boolean matrices, A of dimension $M' \times N$ and B of dimension $N \times M$, for $M' \geq M \geq N$. We encode A as a text T by writing down its elements row-by-row and adding some padding. Namely:

$$T = \#^{M^2} r_1 \ \#^{M-N+1} \ r_2 \ \#^{M-N+1} \ \ldots \ \#^{M-N+1} \ r_{M'} \#^{M^2}$$

where $r_i = r_{i,1} \ldots r_{i,N}$ and $r_{i,j} = 0$ when $A_{i,j} = 0$ and $r_{i,j} = j$ when $A_{i,j} = 1$. Similarly, we encode B as a pattern P by writing down its elements column-by-column, except that here the padding is shorter by one character:

$$P = c_1 \ \#^{M-N} \ c_2 \ \#^{M-N} \ \ldots \ \#^{M-N} \ c_M$$

where $c_j = c_{1,j} \ldots c_{N,j}$ and $c_{i,j} = 0'$ when $B_{i,j} = 0$ and $c_{i,j} = i$ when $B_{i,j} = 1$.

Observe that, since we encode 0s from A and B using different symbols, and encoding of 1s is position-dependent, r_i and c_j will generate a match only if they are perfectly aligned and there is k such that $r_{i,k} = c_{k,j}$, or equivalently $A_{i,k} = B_{k,j} = 1$. Since each block (encoded row plus following padding) is either of length $N+1$ for rows or N for columns, there will be at most one aligned row-column pair for each pattern-text alignment.

The total number of mismatches, for each alignment, is at most $2NM$ (since there are at most MN non-# text characters that are aligned with pattern, and at most MN non-# pattern characters). We can determine whether any given entry of $A \cdot B$ is a 1, since if so the number of mismatches for the corresponding alignment is decreased by 1.

We have $|T| = \Theta(M'M)$ and $|P| = \Theta(M^2)$. By setting $M = \sqrt{m}$, $M' = \frac{n}{\sqrt{m}}$ and $N = \frac{k}{\sqrt{m}}$ we obtain Theorem 2.

If we denote by $\omega(\alpha, \beta, \gamma)$ the exponent of fastest algorithm to multiply a matrix of dimension $n^\alpha \times n^\beta$ with a matrix of dimension $n^\beta \times n^\gamma$, we also have:

▶ **Corollary 13.** *For any positive $\varepsilon, \alpha, \kappa$, such that $\frac{1}{2}\alpha \leq \kappa \leq \alpha \leq 1$ there is no algorithm solving pattern matching with $\Theta(n^\kappa)$ mismatches in time $\mathcal{O}(n^{\omega(2-\alpha, 2\kappa-\alpha, \alpha)/2-\varepsilon})$ for a text of length n and a pattern of length $\Theta(n^\alpha)$.*

5 $(1 \pm \varepsilon)$-approximation of L_1 Distance

In this section we prove Theorem 3. We use a procedure `generalized_weighted_matching` $(T, P,$ `score`$)$ that computes, for a text $T = t_1 t_2 \ldots t_n$ and a pattern $P = p_1 p_2 \ldots p_m$ and an arbitrary weight function $\sigma : \Sigma \times \Sigma \to \mathbb{Z}$, the array S_σ such that $S[i] = \sum_{j=1}^{m} \sigma(t_{i+j}, p_j)$ in in $\mathcal{O}(|\Sigma| n \log m)$ time.

Let $\delta = \frac{\varepsilon}{24 \cdot (3 + \log M)} = \Theta(\varepsilon / \log n)$, and let b be the smallest positive integer such that $2^b \geq 1/\delta$. We claim that with such parameters, Algorithm 1 outputs the desired $(1 \pm \varepsilon)$-approximation in the claimed time. Let S_ε be its output.

▶ **Theorem 14.** *For any i, $S[i] \cdot (1 - \varepsilon) \leq S_\varepsilon[i] \leq S[i] \cdot (1 + \varepsilon)$ with probability at least $2/3$.*

Proof. Consider first $x = x_a$ and $y = y_b$, two characters of the input. We analyze how well Algorithm 1 approximates $|x - y| = \operatorname{sgn}(x - y) \cdot (x - y)$ in the consecutive calls of `generalized_weighted_matching`. First, fix value of Δ and consider the binary representations of $x' = x + \Delta$ and $y' = y + \Delta$. More precisely, let $x' = \sum_i 2^i \cdot \alpha_i$ and $y' = \sum_i 2^i \cdot \beta_i$ for

Algorithm 1: $(1 \pm \varepsilon)$-approximation of text-to-pattern L_1 distance.

```
Input: Integer strings $T$ and $P$.
Output: Score vector $S_\varepsilon$.
def score($x, y$):
    $x_0 \leftarrow x \bmod 2$
    $y_0 \leftarrow y \bmod 2$
    if $x_0 = y_0$ then
        return $0$
    else if $\operatorname{sgn}(x - y) = \operatorname{sgn}(x_0 - y_0)$ then
        return $1$
    else
        return $-1$

def approximate($T, P$):
    $\Delta \leftarrow$ u.a.r. integer from 0 to $2^{\lceil \log M \rceil} - 1$
    $T' \leftarrow T + \Delta$
    $P' \leftarrow P + \Delta$
    $S_\varepsilon \leftarrow [0 \dots 0]$
    for $i \leftarrow 0$ to $\lceil \log M \rceil$ do
        $T'' \leftarrow \lfloor T'/2^i \rfloor \bmod 2^b$
        $P'' \leftarrow \lfloor P'/2^i \rfloor \bmod 2^b$
        $S \leftarrow$ generalized_weighted_matching($T'', P''$, score)
        $S_\varepsilon \leftarrow S_\varepsilon + S \cdot 2^i$
    return $S_\varepsilon$
```

some $\alpha_i, \beta_i \in \{0, 1\}$. Algorithm 1 in essence estimates $|x - y| = \operatorname{sgn}(x-y) \sum_i 2^i(\alpha_i - \beta_i)$ with $C = \sum_i 2^i \gamma_i$ where $\gamma_i \in \{-1, 0, 1\}$ is the estimation of a contribution of $(\alpha_i - \beta_i) \cdot \operatorname{sgn}(x - y)$ to $(x' - y')$ and depends only on values of $\alpha_j - \beta_j \in \{-1, 0, 1\}$ for $i \le j < i + b$ in the following way:

- If, for every $i \le j < i + b$ we have $\alpha_j = \beta_j$, then $\gamma_i = 0$.
- Otherwise, let j' be the largest j such that $i \le j < i+b$ and $\alpha_j \neq \beta_j$. If $\alpha_{j'} - \beta_{j'} = 1$, then the local estimation is that $x' > y'$ and so $\gamma_i = \alpha_i - \beta_i$, and otherwise $\gamma_i = -1 \cdot (\alpha_i - \beta_i)$.

Consider $c = \max\{i : \alpha_i \neq \beta_i\}$ and $d = \max\{i : 2^i \le (x' - y')\}$, that is c is the position of the highest bit on which x' and y' differ, and d is the position of the highest bit of $x' - y'$. In general, $c \ge d$, and we say that pair x', y' is *t-bad*, if $c - d = t$.

We first observe that for a x, y pair to be at least t-bad, a following condition must be met: $\lfloor x'/2^{d+t} \rfloor \neq \lfloor y'/2^{d+t} \rfloor$. Since Δ is chosen uniformly at random from a large enough range of integers, there is

$$\sum_{\tau \ge t} \Pr(x', y' \text{ is } \tau\text{-bad} \mid x, y) \le |x - y|/2^{d+t} \le 2^{-t+1}.$$

We also observe following: for any pair x', y', in C, all the coefficients $\gamma_c, \gamma_{c-1}, \dots, \gamma_{c-b+1}$ are computed correctly, since for any j such that $c \ge j \ge c - b + 1$ there is $j' = c$, and then $\gamma_j = (\alpha_j - \beta_j) \cdot \operatorname{sgn}(x - y)$. Therefore

$$\big|C - |x' - y'|\,\big| = \left| \sum_{i \le c-b} 2^i(\gamma_i - (\alpha_i - \beta_i) \cdot \operatorname{sgn}(x - y)) \right| \le \sum_{i \le c-b} 2 \cdot 2^i < 2 \cdot 2^{c-b+1}.$$

If a pair x', y' is t-bad, it immediately follows that the absolute error of estimation is at most $2^{c-b+2} = 2^{d+t-b+2} \le |x' - y'| 2^{t+2}\delta$.

We now estimate expected error in estimation based on choice of Δ. If a particular pair x, y is t-bad, then $t \le 1 + \lceil \log M \rceil$. Using the previous observations, we have

$$\mathbb{E}\Big[\big|C - |x' - y'|\big| \;\Big|\; x, y\Big] = \sum_t \Pr(x', y' \text{ is } t\text{-bad} \mid x, y) \cdot \mathbb{E}\Big[\big|C - |x' - y'|\big| \;\Big|\; x', y' \text{ is } t\text{-bad}\Big]$$
$$\le \sum_{t=0}^{1+\lceil \log M \rceil} 2^{-t+1}|x - y| 2^{t+2}\delta = (3 + \log M) 8\delta |x - y| = \frac{\varepsilon}{3}|x - y|.$$

By linearity of expectation $\mathbb{E}\Big[\big|S_\varepsilon[i] - S[i]\big|\Big] \le \frac{\varepsilon}{3} S[i]$, and by Markov's inequality the claim follows. ◀

Now, a standard amplification technique applies: it is enough to repeat Algorithm 1 independently p times and take the median value from $S_\varepsilon^{(1)}[i], S_\varepsilon^{(2)}[i], \dots, S_\varepsilon^{(p)}[i]$ as the final estimate $\widehat{S}_\varepsilon[i]$. Taking $p = \Theta(\log n)$ to be large enough makes the final estimate good with high probability, and by the union bound whole $\widehat{S}_\varepsilon$ is a good estimate of S.

The complexity of Algorithm 1 is dominated by `generalized_weighted_matching` being invoked $\mathcal{O}(\log n)$ times on alphabet of size $2^b = \Theta(\varepsilon^{-1} \log n)$. Each such invocation takes $\mathcal{O}(2^b n \log m) = \mathcal{O}(\varepsilon^{-1} n \log n \log m)$, and Algorithm 1 takes $\mathcal{O}(\varepsilon^{-1} n \log^2 n \log m)$ time and the total time for computing $(1 \pm \epsilon)$-approximation is $\mathcal{O}(\varepsilon^{-1} n \log^3 n \log m)$.

6 Reductions

In this section we design a series of reductions to complete the picture from Figure 2. We start with an L_1 version of Theorem 10.

▶ **Theorem 15.** *k-bounded text-to-pattern L_1 distance problem reduces in $\mathcal{O}(n \log^3 m)$ time to $\mathcal{O}(n/m)$ instances of $\mathcal{O}(k)$-RLE text-to-pattern L_1 distance on inputs of length $\mathcal{O}(m)$, where both the pattern and the text might contain wildcards.*

Proof. As in the proof of Theorem 10, we can assume that the length of T is $2m$. We observe that if the L_1 distance of an m-substring of T is at most k, so must be its Hamming distance. Therefore, we can use exactly the same filtering approach. In the case of no small $4k$-period, after filtering m-substrings with more than $2k$ mismatches we are left with $\mathcal{O}(m/k)$ possibilities, which are then verified using the "kangaroo jumps" technique of Landau and Vishkin [13]. The only required modification is that now for every found mismatch we calculate the corresponding increase in the L_1 distance. In the case of a small $8k$-period, we use the same transformation, except that all special characters become identical wildcards $*$ with L_1 distance 0 to every letter. This preserves the L_1 distance by the same argument. ◀

Then, instead of designing an algorithm for computing k-RLE L_1 distance, we apply the following reduction.

▶ **Corollary 16** (Theorem 2.1, Theorem 2.2 and Lemma A.1 in [8]). *For any $M \ge 0$, there is a linearity preserving reduction from L_1 distance between integers from $[M]$ to $\mathcal{O}(\log^2 M)$ instances of Hamming distance. There is a converse reduction from Hamming distance to $\mathcal{O}(1)$ instances of L_1 distance. Both reductions allow for wildcards in the input and output wildcard-less instances.*

By inspecting the proof of the above reduction we see that it does not create any new runs, that is, allows us to reduce k-RLE L_1 distance to $\mathcal{O}(\log^2 M)$ instances of k-RLE Hamming distance. Therefore, together with Theorem 15 and Lemma 11, we obtain an $\mathcal{O}((m + k\sqrt{m \log m}) \cdot \log^2 n + n \log^3 m)$ time algorithm for k-bounded L_1 distance as claimed in Theorem 4.

To complete the picture, we show a reduction from k-RLE Hamming distance to $2k$-bounded Hamming distance, that is, a converse to Theorem 10.

▶ **Theorem 17.** *k-RLE text-to-pattern Hamming distance on inputs of length $\mathcal{O}(m)$ reduces to $\mathcal{O}(1)$ instances of $2k$-bounded Hamming distance on inputs of length $\mathcal{O}(m)$.*

Proof. We proceed similarly as in Lemma 11. We observe that it is enough to compute the second discrete derivative of the output array S, that is D^2S defined as $(D^2S)[i] = S[i+2] - 2S[i+1] + S[i]$, since D^2S and two initial values of S (computed naively in time $\mathcal{O}(m)$) are enough to recover S. For any two blocks $t_u t_{u+1} \dots t_{v-1} t_v$ and $p_y p_{y+1} \dots p_{z-1} p_z$ of the same letter, D^2S needs to be updated in only 4 places, that is $D^2S[u-z] += 1$, $D^2S[v-z+1] -= 1$, $D^2S[u-y+1] -= 1$ and $D^2S[v-y+2]$. We now explain how to deal with the first kind of updates, with the other three being implemented similarly.

We first reduce the problem to *k-sparse text-to-pattern Hamming distance*, where the text and the pattern are of length $\mathcal{O}(m)$ and have each at most k regular characters, with every other character being wildcard $*$ (special character having distance 0 to every other character). We construct sparse instance as follows: for every position t_u in T that starts a block, we set $T_{\text{sparse}}[u] = t_u$, and similarly in pattern for a position p_z (that ends a block), we set $P_{\text{sparse}}[z] = p_z$. Observe, that if $t_u \neq p_z$, then in the answer there is $S_{\text{sparse}}[u-z] += 1$, and if $t_u = p_z$ then S_{sparse} remains unchanged. That is, the answer counts mismatches, while we want to count matches. To invert the answer, we create T_{bin} such that $T_{\text{bin}}[i] = 1$ iff $T_{\text{sparse}}[i] \neq *$ and $T_{\text{bin}}[i] = 0$ otherwise, and P_{bin} in an analogous manner. Convolving T_{bin} and P_{bin} gives us, for every alignment, the total number of non-special text characters aligned with non-special pattern characters, and we obtain the answer with a single subtraction.

To reduce from k-sparse instances of Hamming distance to $2k$-bounded Hamming distance, we follow an analogous reduction from Lemma A.1 in [8] that reduces Hamming distance on $\mathbb{N}^+ + \{*\}$ to Hamming distance on $\mathbb{N}$. First, create a new text T_1 such that $T_1[i] = T[i]$ iff $T[i] \neq *$ and $T_1[i] = 0$ iff $T[i] = *$, and similarly to obtain P_1. Second, create a new text T_2 such that $T_2[i] = 1$ iff $T_2[i] \neq *$ and $T_2[i] = 0$ iff $T[i] = *$, and similarly to obtain P_2. Now we observe that $\text{Ham}(T[i], P[j]) = \text{Ham}(T_1[i], P_1[j]) - \text{Ham}(T_2[i], P_2[j])$, thus it is enough to compute exact Hamming text-to-pattern distances on these two instances and subtract them. However, we observe that in both of them, there are in total at most $2k$ characters different than 0, thus $2k$-bounded Hamming distance works just as fine. ◀

7 Conclusion and Open Problems

Showing a reduction from either *bounded RLE* L_1 distance, or *sparse* L_1 distance to k-approximated L_1 distance would suffice in completing a converse to Theorem 5, and prove that complexity of k-bounded L_1 and Hamming distances is the same, up to poly-logarithmic factors.

References

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

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

3 Amihood Amir, Ohad Lipsky, Ely Porat, and Julia Umanski. Approximate matching in the L_1 metric. In *CPM*, pages 91–103, 2005.

4 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM J. Comput.*, 41(5):1299–1317, 2012.

5 Peter Clifford, Raphaël Clifford, and Costas S. Iliopoulos. Faster algorithms for δ,γ-matching and related problems. In *CPM*, pages 68–78, 2005.

6 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`, 2009. Retrieved March 2017.

7 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *SODA*, pages 2039–2052, 2016.

8 Daniel Graf, Karim Labib, and Przemyslaw Uznański. Hamming distance completeness and sparse matrix multiplication. *CoRR*, abs/1711.03887, 2017. `arXiv:1711.03887`.

9 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

10 Howard J. Karloff. Fast algorithms for approximately counting mismatches. *Inf. Process. Lett.*, 48(2):53–60, 1993.

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

12 Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern hamming distance. In *SOSA*, pages 10:1–10:5, 2018.

13 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986.

14 Ohad Lipsky and Ely Porat. L_1 pattern matching lower bound. *Inf. Process. Lett.*, 105(4):141–143, 2008.

15 Ohad Lipsky and Ely Porat. Approximate pattern matching with the L_1, L_2 and L_∞ metrics. *Algorithmica*, 60(2):335–348, 2011.

16 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA*, pages 167–175, 2004.