2015 IEEE 56th Annual Symposium on Foundations of Computer Science

Breaking the Variance: Approximating the Hamming Distance in $1/\epsilon$ Time Per Alignment

Tsvi Kopelowitz*
Department of EECS
University of Michigan
Ann Arbor, MI, USA
kopelot@gmail.com

Ely Porat
Department of Computer Science
Bar-Ilan University
Ramat-Gan, Israel
porately@cs.biu.ac.il

Abstract

The algorithmic tasks of computing the *Hamming distance* between a given pattern of length m and each location in a text of length n is one of the most fundamental algorithmic tasks in string algorithms. Unfortunately, there is evidence that for a text T of size n and a pattern P of size m, one cannot compute the exact Hamming distance for all locations in T in time which is less than $\tilde{O}(n\sqrt{m})$. However, Karloff [30] showed that if one is willing to suffer a $1 \pm \epsilon$ approximation, then it is possible to solve the problem with high probability, in $\tilde{O}(\frac{n}{\epsilon^2})$ time.

Due to related lower bounds for computing the Hamming distance of two strings in the one-way communication complexity model, it is strongly believed that obtaining an algorithm for solving the approximation version cannot be done much faster as a function of $\frac{1}{\epsilon}$. We show here that this belief is *false* by introducing a new $\tilde{O}(\frac{n}{\epsilon})$ time algorithm that succeeds with high probability.

The main idea behind our algorithm, which is common in sparse recovery problems, is to reduce the variance of a specific randomized experiment by (approximately) separating heavy hitters from non-heavy hitters. However, while known sparse recovery techniques work very well on vectors, they do not seem to apply here, where we are dealing with mismatches between *pairs* of characters. We introduce two main algorithmic ingredients. The first is a new sparse recovery method that applies for pair inputs (such as in our setting). The second is a new construction of hash/projection functions, for which have which allows us to count the number of projections that induce mismatches between two characters *exponentially* faster than brute force. We expect that these algorithmic techniques will be of independent interest.

Keywords

Approximate Hamming distance, Stringology, Sparse Recovery.

I. Introduction

One of the most fundamental family of problems in string algorithms is to compute the distance between a given pattern P of length m and each location in given larger text T of length n both, over alphabet Σ, under some string distance metric (See [31], [22], [2], [32], [8], [6], [3], [7], [37], [13], [35], [33], [9], [12], [39], [34], [20], [11], [16], [19], [18], [17], [5], [4], [38]). The most important distance metric in this setting is the *Hamming Distance* of two strings, which is the number of aligned character mismatches between the strings. Let $\texttt{HAM}(X, Y)$ denote the Hamming distance of two strings X and Y. Abrahamson [1] showed an algorithm whose runtime is $\tilde{O}(n\sqrt{m})$. The task of obtaining a faster upper bound seems to be very challenging, and indeed there is a folklore matching conditional lower bound for combinatorial algorithms based on the hardness of combinatorial boolean matrix multiplication (see [15]). However, for constant sized alphabets the runtime can be reduced to $\tilde{O}(n)$ using a constant number of convolution computations (which are implemented via the FFT algorithm) [22].

* Supported by NSF Grants CCF-1217338, CNS-1318294, and CCF-1514383.

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.43

This naturally lead to approximation algorithms for computing the Hamming distance in this setting, which is the problem that we consider here and is defined as follows. Denote $T_j = T[j, \ldots, j+m-1]$. In the *pattern-to-text approximate Hamming distance problem* the input is a parameter $\epsilon > 0$, T, and P. The goal is to compute for all locations $i \in [1, n-m+1]$ a value δ_i such that $(1-\epsilon)\texttt{HAM}(T_i, P) \leq \delta_i \leq (1+\epsilon)\texttt{HAM}(T_i, P)$. For simplicity we assume without loss of generality that Σ is the set of integers $\{1, 2, \ldots, |\Sigma|\}$.

Karloff in [30] utilized the efficiency of the algorithm for constant sized alphabets to introduce a beautiful randomized algorithm for solving the pattern-to-text approximate Hamming distance problem, by utilizing projections of Σ to binary alphabets. Karloff's algorithm runs in $\tilde{O}(\frac{n}{\epsilon^2})$ time, and is correct with high probability.

Communication complexity lower bounds.: One of the downsides of Karloff's algorithm is the dependence on $\frac{1}{\epsilon^2}$. In particular, if one is interested in a one percent approximation guarantee, then this term becomes 10000! However, many believe that beating the runtime of Karloff's algorithm is not possible, mainly since there exist qualitatively related lower bounds for estimating the Hamming distance of two equal length strings (for a single alignment). In particular, Woodruff [40] and later Jayram, Kumar and Sivakumar [28] showed that obtaining a $(1 \pm \epsilon)$ approximation for two strings in the one-way communication complexity model requires sending $\Omega(1/\epsilon^2)$ bits of information. This lower bound implies a lower bound for the sketch size of the Hamming distance and some other streaming problems.

Our results here show that **this intuition is flawed**, by introducing an $\tilde{O}(\frac{n}{\epsilon})$ time algorithm that succeeds with high probability.

The challenge – beating the variance.: The main idea of Karloff's algorithm is to project Σ to a binary alphabet, and compute the Hamming distance for each location of the projected text and the projected pattern. For a given location denote by d the Hamming distance of the pattern at this location. While the projected Hamming distance is expected to be $\frac{d}{2}$, the variance of the projected distance could be as high as $\Omega(d^2)$ (see Section III for a detailed calculation). In order to overcome this high variance Karloff's algorithm makes use of $\tilde{O}(\frac{1}{\epsilon^2})$ projections. More detail is given in Section III.

The first step in obtaining a more efficient algorithm is to somehow reduce the number of projections that an algorithm would use. One line of attack would be to somehow reduce the variance. Indeed, such approaches have been considered in other problems [29], and even for the problem considered here, Atallah, Grigorescu, and Wu in [10] managed to slightly reduce the variance in some cases. They do this by computing the exact contribution to the Hamming distance of the k most frequent characters in the pattern and approximating the contribution of the rest of the characters. This reduces the variance to $O(d \min(m\frac{m}{k}, d)$ for each projection, and by repeating the process k times the variance of the average result becomes $d \min(\frac{m}{k}, d)/k$. This approach is only useful when the Hamming distance is high – at least $\frac{m}{k}$.

However our goal here is to obtain an even faster algorithm, and so we devise a new method for reducing the variance.

A. Our results and techniques.

Our main result is the first significant improvement on this problem in the last over 20 years. We present a new randomized algorithm that solves the pattern-to-text approximate Hamming distance with high probability (at least $1 - n^{-\Omega(1)}$) that runs in worst-case $\tilde{O}(\frac{n}{\epsilon})$ time. This is summarized in the following Theorem.

Theorem 1. *There exists an algorithm that with high probability solves the pattern-to-text approximate Hamming distance problem and runs in $O(\frac{n}{\epsilon} \log \frac{1}{\epsilon} \log n \log m \log |\Sigma|)$ time.*

Furthermore, we introduce two exciting novel techniques in our algorithms.

Intuition.: The main intuition (which may not reveal the technical challenges) is to reduce the variance of the estimation produced by a single projection, by removing pairs of characters with a large contribution to the Hamming distance. In particular, for Hamming distance d we would like to remove pairs that contribute at least ϵd. Such pairs are called *heavy hitter pairs* and there are at most $\frac{1}{\epsilon}$ such pairs at each location. Unfortunately, it is not clear how to detect the heavy hitter pairs for each location within the time bounds that we are aiming for.

A common method for quickly approximating the heavy hitters in a given vector is sparse recovery [24], [25], [36], [23], [14], [21], [26], [27]. The idea is to approximate the heavy hitters of a vector, while suffering from some extra noise in the form of some additional points in the vector, but not more than $O(\frac{1}{\epsilon})$ points. However, since in our setting the heavy hitters are pairs of elements (as opposed to single elements in a vector), there are structural constraints that make known sparse recovery techniques irrelevant here.

Approximating heavy hitter pairs.: To overcome these structural constraints, we introduce an algorithm that utilizes a *specially constructed small set of projections*. We roughly show that with high probability the L_2 distance of the approximate heavy hitter pairs and the actual heavy hitter pairs is small. This in turn allows us to obtain a linear time algorithm which estimates the Hamming distance with variance at most $O(\epsilon d^2)$, and so $\tilde{O}(\frac{1}{\epsilon})$ repetitions suffice. This construction can be found in Section V.

Computing all of the projections quickly.: While the approximation of the heavy hitter pairs implies that the number of repetitions can be low, the algorithm that we use will still need to project the $O(\frac{1}{\epsilon})$ pairs in the approximation with each of the $\tilde{O}(\frac{1}{\epsilon})$ projections. Doing this directly will cost $\tilde{O}(\frac{1}{\epsilon^2})$ time per location which is too high. To overcome this, we make use of the way in which our algorithm uses the projected outcomes of the approximating pairs, by constructing a specially tailored set of projections. This construction may be of independent interest, and is detailed in Section VI. Given k we construct k hash functions $h_1, ...h_k$ into the binary alphabet, such that given $x \neq y$ we can count for how many functions h_i we have $h_i(x) = h_i(y)$ in $O(\log k)$ time, rather than $O(k)$ time.

II. Preliminaries

For a given location j, consider the alignment of T_j with P. This alignment naturally defines an alignment matrix $D = D_j = \{d_{u,v}\}_{u,v\in\Sigma}$, such that for $u \neq v$ we have $d_{u,v} = |\{0 \leq i \leq m-1 : T[j+i] = u \wedge P[i] = v\}|$ and 0 otherwise. In words, $d_{u,v}$ is the contribution of the pair (u, v) to the Hamming distance of T_j and P. Clearly, the Hamming distance is $d = \sum_{u,v\in\Sigma} d_{u,v}$. We emphasize that computing or representing the alignment matrix explicitly is too costly in our setting.

Local versus global operations.: The operations that our algorithm performs during the computation of the Hamming distance at some location j can be partitioned into two types. The first type are *local* operations which are independent of the computations performed for other locations in T. The second type are *global* operations, which in order to be done efficiently may consider the alignments at other locations in T. In particular, all of the global operations in our algorithm can be reduced to computing the number of times that a 1 in a projection of T_j to a binary alphabet aligns with a 1 in a projection of P to a binary alphabet (the projection function is not required to be the same for T_j and P). Such a computation will make use of the following Theorem.

Theorem 2. *Given a binary text T of size n and a binary pattern P of size m, there exists an $O(n \log m)$ time algorithm that computes for all locations i in T the number of times that a 1 in T_i is aligned with a 1 in P.*

The algorithm for Theorem 2 is implemented via a single convolution (using the FFT) in $O(n \log m)$ time, and so can charge an $O(\log m)$ time cost to each location for each global operation. We emphasize that the efficiency of the algorithm in Theorem 2 is only relevant when the projections to binary alphabets are the same for all locations j, which indeed will be the case in our algorithm.

Simplifying assumptions.: With the goal of easing the presentation of our algorithm, we focus on estimating the Hamming distance between T_j and P, and show that the number of global and local operations that our algorithm performs for this location is $\tilde{O}(1/\epsilon)$. In particular, we will use D throughout to refer to the alignment matrix D_j, omitting the subscript j. We also emphasize that since we are interested in algorithms that succeed with high probability (at least $1 - \frac{1}{n^{\Theta(1)}}$) then it suffices to show that with high probability the algorithm succeeds at location j. Furthermore, since we are ignoring poly–log factors, we will only show that the algorithm succeeds with probability which is strictly larger than $\frac{1}{2}$ by at least some constant. The median of the estimations of $\Theta(\log n)$ independent executions of the algorithm guarantees success with high probability.

KARLOFF(T_j, P, ϵ)

1 $k \leftarrow O(\frac{1}{\epsilon^2})$
2 construct a p.w.i set of 4-wise independent hash functions $h_1, h_2, \ldots, h_k : \Sigma \to \{0,1\}$.
3 **for** $i = 1$ to k
4 **do** compute $x_i = \texttt{HAM}(h_i(T_j), h_i(P))$.
5 $X^* = 2\frac{\sum_{i=1}^{k} x_i}{k}$
6 return X^*

Figure 1. Karloff's Algorithm.

APPROX-HAMM-DISTANCE(T_j, P, ϵ)

1 $k \leftarrow \frac{8b}{\epsilon}$
2 Construct-$D'(T_j, P, \epsilon)$
3 construct a p.w.i set of 4-wise independent hash functions $h_1, h_2, \ldots, h_k : \Sigma \to \{0,1\}$.
4 $X^* = \sum_{i=1}^{k} \left(\texttt{HAM}(h_i(T_j), h_i(P)) + \frac{1}{2} \sum_{\substack{u,v \\ h_i(u)=h_i(v)}} d'_{u,v} - \frac{1}{2} \sum_{\substack{u,v \\ h_i(u)\neq h_i(v)}} d'_{u,v} \right) / (k/2)$
5 return X^*

Figure 2. The new Algorithm.

III. KARLOFF'S ALGORITHM

Karloff in [30] presented an algorithm for solving pattern-to-text approximate Hamming distance that runs in $\tilde{O}(\frac{n}{\epsilon^2})$ time, and is correct with high probability. We present an overview of (a simplified version of) Karloff's algorithm as understanding it is helpful for the setup of the new algorithm presented here. The pseudo-code for the algorithm is given in Figure 1.

The only global operation is to compute the Hamming distance in line 4, which happens $O(\frac{1}{\epsilon^2})$ times for a total of $O(\frac{1}{\epsilon^2} \log m)$ time. The rest of the operations are all local and cost $O(\frac{1}{\epsilon^2})$ time. For any x_i as computed in line 4, the expected value of x_i is $E[x_i] = \frac{d}{2}$, and since each hash function is 4-wise independent, the variance is

$$
\begin{aligned}
V[x_i] &= \sum_{u<v} V\left[\alpha_{u,v}(d_{u,v} + d_{v,u})\right] = \sum_{u<v} E\left[\left(\alpha_{u,v}(d_{u,v} + d_{v,u}) - E\left[(\alpha_{u,v})(d_{u,v} + d_{v,u})\right]\right)^2\right] \\
&= \sum_{u<v} E\left[\left(\alpha_{u,v}(d_{u,v} + d_{v,u}) - \frac{1}{2}(d_{u,v} + d_{v,u})\right)^2\right] \\
&= \sum_{u<v} \Pr\left[\alpha_{u,v} = 1\right] \cdot \left(\frac{1}{2}(d_{u,v} + d_{v,u})\right)^2 + \Pr\left[\alpha_{u,v} = 0\right] \cdot \left(-\frac{1}{2}(d_{u,v} + d_{v,u})\right)^2 \\
&= \sum_{u<v} \frac{1}{4}(d_{u,v} + d_{v,u})^2 = \sum_{\substack{u<v \\ d_{u,v}<\epsilon d \\ d_{v,u}<\epsilon d}} \frac{(d_{u,v} + d_{v,u})^2}{4} + \sum_{\substack{u<v \\ d_{u,v}\geq\epsilon d \vee d_{v,u}\geq\epsilon d}} \frac{(d_{u,v} + d_{v,u})^2}{4} \\
&< \frac{\epsilon d^2}{4} + \frac{1}{4} \sum_{\substack{u<v \\ d_{u,v}\geq\epsilon d \vee d_{v,u}\geq\epsilon d}} (d_{u,v} + d_{v,u})^2. \qquad (1)
\end{aligned}
$$

The challenge here is that the variance can be very large – up to roughly $\Omega(d^2)$, and so in order to overcome it Karloff suggested using $O(\frac{1}{\epsilon^2})$ pair-wise independent projections to the binary alphabet, and then by applying the

Chebyshev inequality the probability that the average of the projected distances is an acceptable approximation of d is large enough.

IV. New Algorithm

The reason why Karloff used $O(\frac{1}{\epsilon^2})$ different projections was because the variance of a single projection is high. As can be seen in Equation 1, the high variance is due to the at most $\frac{1}{\epsilon}$ entries in the alignment matrix D that are larger than ϵd. Such entries are called *heavy hitters*. If we could somehow separate those heavy hitters from the rest of D, and, say, compute their values directly, then the remaining non-heavy hitter entries would have a low variance ($O(\epsilon d^2)$) in which case $O(\frac{1}{\epsilon})$ projections would suffice.

However, it is not clear how to find these heavy hitters efficiently. So instead, we use a different strategy. We say that a matrix $D' = \{d'_{u,v}\}_{u,v\in\Sigma}$, is a *sparse approximate matrix* if:

1) $\sum_{u,v\in\Sigma}(d_{u,v} - d'_{u,v})^2 \leq b\epsilon d^2$ for constant $b = \frac{2^{13}+2^{12}+1}{2^{14}} < 0.752$.
2) The number of non-zero entries in D' is at most $\frac{3}{\epsilon}$.

Notice that the matrix in which all entries that correspond to a heavy hitter pair (u, v) have the value $d_{u,v}$ and all other entries are zero is a sparse approximate matrix, which matches the intuition described above. As we shall show, the only properties of such a matrix that we require are the ones which define a sparse approximate matrix.

Our algorithm will make use of sparse approximate matrices by constructing a (possibly different) matrix for each location in T. However, as explained in Section II the discussion here focuses on only one location j, and so we make use of only one sparse approximate matrix D'. We emphasize that representing D' explicitly is too costly since it is too large. Instead, we use an implicit representation of D' by considering only the non-zero entries. In Section V we show an algorithm that with high probability will construct an implicitly represented matrix D' with the desired properties in $\tilde{O}(\frac{1}{\epsilon})$ time, by proving the following lemma.

Lemma 3. *There exists an algorithm that with high probability computes a sparse approximate matrix D' for location j in T such that the number of global and local operations performed by the algorithm is $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log n\log m\log|\Sigma|)$.*

Given Lemma 3, the pseudo-code of the algorithm is given in Figure 2. We will now bound the expected value and variance of X^*. Fix h_i. Let $\alpha_{u,v} = 1$ if $h_i(u) \neq h_i(v)$ and 0 otherwise, and let $x_i = \texttt{HAM}(h_i(T_j), h_i(P)) + \frac{1}{2}\sum_{h_i(u)=h_i(v)} d'_{u,v} - \frac{1}{2}\sum_{h_i(u)\neq h_i(v)} d'_{u,v}$. Notice that $E[\frac{1}{2} - \alpha_{u,v}] = 0$. Recall that each function h_i is 4-wise independent. Then the expected value and variance of x_i, under the random choice of h_i, is $E[x_i] = \frac{d}{2}$ and

$$
\begin{aligned}
V[x_i] &= \sum_{u<v} V\left[\alpha_{u,v}(d_{u,v}+d_{v,u}) + (\frac{1}{2}-\alpha_{u,v})(d'_{u,v}+d'_{v,u})\right] \\
&= \sum_{u<v} E\left[\left(\alpha_{u,v}(d_{u,v}+d_{v,u}) + (\frac{1}{2}-\alpha_{u,v})(d'_{u,v}+d'_{v,u}) - E\left[(\alpha_{u,v})(d_{u,v}+d_{v,u}) + (\frac{1}{2}-\alpha_{u,v})(d'_{u,v}+d'_{v,u})\right]\right)^2\right] \\
&= \sum_{u<v} E\left[\left(\alpha_{u,v}(d_{u,v}+d_{v,u}) + (\frac{1}{2}-\alpha_{u,v})(d'_{u,v}+d'_{v,u}) - \frac{1}{2}(d_{u,v}+d_{v,u})\right)^2\right] \\
&= \sum_{u<v}\left[\Pr[\alpha_{u,v}=1]\cdot\left(\frac{1}{2}(d_{u,v}+d_{v,u}) - \frac{1}{2}(d'_{u,v}+d'_{v,u})\right)^2 + \Pr[\alpha_{u,v}=0]\cdot\left(\frac{1}{2}(d'_{u,v}+d'_{v,u}) - \frac{1}{2}(d_{u,v}+d_{v,u})\right)^2\right] \\
&= \sum_{u<v}\left[\frac{1}{2}\left(\frac{1}{2}(d_{u,v}+d_{v,u}) - \frac{1}{2}(d'_{u,v}+d'_{v,u})\right)^2 + \frac{1}{2}\left(\frac{1}{2}(d'_{u,v}+d'_{v,u}) - \frac{1}{2}(d_{u,v}+d_{v,u})\right)^2\right] \\
&= \sum_{u<v}\frac{1}{4}\left((d_{u,v}-d'_{u,v}) + (d_{v,u}-d'_{v,u})\right)^2 \leq \sum_{u<v}\frac{1}{4}(2(d_{u,v}-d'_{u,v})^2 + 2(d_{v,u}-d'_{v,u})^2) \\
&= \sum_{u,v}\frac{1}{2}(d_{u,v}-d'_{u,v})^2 < \frac{0.752}{2}\epsilon d^2 = 0.376\epsilon d^2.
\end{aligned}
$$

Thus, given (an implicit) D', we obtain an estimation of d that has low variance. Recall that the hash functions are pair-wise independent among themselves. Therefore, $E[X^*] = d$ and $V[X^*] = V[\sum_{i=1}^{k} x_i/(k/2)] = \frac{4}{k^2}\sum_{i=1}^{k} V[x_i] = \frac{4V[x_i]}{k} = \frac{4b\epsilon d^2}{8b/\epsilon} = \frac{\epsilon^2 d^2}{2}$. Therefore, by Chebyshev's inequality, $\Pr[|X^* - E[X^*]| > \epsilon d] < \frac{1}{2}$.

While D' can be used to decrease the variance of the estimation, there is still a bottleneck in the runtime of the algorithm from computing $\frac{1}{2}\sum_{h_i(u)=h_i(v)} d'_{u,v} - \frac{1}{2}\sum_{h_i(u)\neq h_i(v)} d'_{u,v}$ for all of the $O(\frac{1}{\epsilon})$ projections. This can be done directly in $O(\frac{1}{\epsilon^2})$ time, which is too costly for our goals (since this computation would need to be repeated $O(n)$ times). We overcome this challenge by introducing a special construction of $O(\frac{1}{\epsilon})$ projections to a binary alphabet, where each projection is 4-wise-independent, and the projection functions are pair-wise-independent among themselves. The special construction will have the property that computing

$$\sum_{i=1}^{k}\left(\frac{1}{2}\sum_{h_i(u)=h_i(v)} d'_{u,v} - \frac{1}{2}\sum_{h_i(u)\neq h_i(v)} d'_{u,v}\right)$$

can be done in $O(\frac{1}{\epsilon})$ time. This construction is summarized in the following lemma, which is proven in Section VI.

Lemma 4. *There exists an algorithm for constructing $h_1, \ldots, h_k$ in line 3 of Approx-Hamming-Distance, so that executing line 4 of Approx-Hamming-Distance takes $O(\frac{1}{\epsilon}\log k)$ time.*

Thus, the total runtime of the algorithm is $\tilde{O}(\frac{1}{\epsilon})$ time per location, for a total of $\tilde{O}(\frac{n}{\epsilon})$ time for all locations.

V. Computing D'

Intuition.: In order to construct a sparse approximate matrix D' we would intuitively like to estimate the at most $\frac{1}{\epsilon}$ heavy hitters of D which are the entries of D that are at least ϵd. One idea for obtaining this estimation is to project the alphabet Σ to a smaller alphabet, with the hopes that heavy hitters before the projection can be established from the heavy hitters after the projection. However, this specific task seems out of grasp, since the projections introduce too much noise. To overcome this challenge, we use several specially constructed projections so that together with the proper algorithm we are able to estimate $O(\frac{1}{\epsilon})$ entries of D, which will suffice in order to bound $\sum_{u,v\in\Sigma}(d_{u,v} - d'_{u,v})^2$. We emphasize that while our algorithm may in fact not estimate all of the heavy hitters, this bound is still obtained with high probability.

The projections.: We consider $O(\log\frac{1}{\epsilon})$ 4-wise independent projections as follows[1]. The ith projection, for $i = 0, \ldots, \log\frac{1}{\epsilon}$, is defined by two projection functions $\tau_i : \Sigma \to [\ell_i]$ and $\pi_i : \Sigma \to [r_i]$, where $\ell_i = 32 \cdot 2^i$ and $r_i = \frac{32}{2^i\epsilon}$. Notice that for all i we have $\ell_i \cdot r_i = \frac{32^2}{\epsilon}$. We assume without loss of generality that $\frac{32^2}{\epsilon}$ is a power of 2. For each i, the text T_j is projected with τ_i while the pattern P is projected with π_i. We then compute for each pair $(x, y) \in [\ell_i] \times [r_i]$ the number of times that x in the projected text is aligned with y in the projected pattern using a single global convolution. Since the number of such pairs is $O(\frac{1}{\epsilon})$ this can be computed in $O(\frac{\log m}{\epsilon})$ time for each i via global operations, for a total of $O(\frac{\log m}{\epsilon}\log\frac{1}{\epsilon})$ for all i.

Intuitively, we would like to claim that for a pair of characters $u \neq v$ the number of times that $\tau_i(u)$ in the projected text aligns with $\pi_i(v)$ in the projected pattern is a close enough estimation of $d_{u,v}$. However, if we allow τ_i and π_i to be any random projection functions then we will be introducing new mismatches via the projection, since it is likely that for a projected pair (x, y) where $x \neq y$ there exists some character $\sigma \in \Sigma$ such that $(x, y) = (\pi_i(\sigma), \tau_i(\sigma))$, and the task of distinguishing projections of matching pairs from projections of mismatching pairs seems to be too difficult within the allowed time. One method for overcoming this problem would be to only consider the number of mismatches for projected pairs as long as they are not of the form $(\pi_i(\sigma), \tau_i(\sigma))$ for any $\sigma \in \Sigma$. Then we would hope that repeating the entire process with enough choices of π_i and τ_i will guarantee that all of the appropriate pairs of different characters are projected enough times to a pair

[1]Notice that these projections are the same for all locations in T and not just for location j. This enables the use of Theorem 2 for global operations in our setting.

CONSTRUCT-$D'(T_j, P, \epsilon)$

1 implicitly initialize all $d'_{u,v} \leftarrow \infty$
2 **for** $i = 0$ to $\log \frac{1}{\epsilon}$
3 **do** $\ell_i \leftarrow 32 \cdot 2^i$
4 $r_i \leftarrow \frac{32}{2^i \epsilon}$
5 **repeat** $\Theta(\log n)$ times
6 **if** $\ell_i \geq r_i$
7 **then** pick a random 4-wise independent projection $\tau_i : \Sigma \rightarrow [\ell_i]$
8 $\pi_i(\cdot) = \tau_i(\cdot) \mod r_i$
9 **else** pick a random 4-wise independent projection $\pi_i : \Sigma \rightarrow [r_i]$
10 $\tau_i(\cdot) = \pi_i(\cdot) \mod \ell_i$
11 **for** every pair $(x, y) \in [\ell_i] \times [r_i]$ where no (σ, σ) is in the preimage of (x, y)
12 **do** $c_{x,y} \leftarrow$ number of times that x in $\tau_i(T_j)$ is aligned with y in $\pi_i(P)$
13 **if** there exists (u, v) in preimage of (x, y) where $d_{u,v} > \frac{c_{x,y}}{2}$
14 **then** $d'_{u,v} \leftarrow \min(d'_{u,v}, c_{x,y})$
15 implicitly set all $d'_{u,v} = \infty$ to $d'_{u,v} \leftarrow 0$
16 **for** every $d'_{u,v} > 0$ such that $d'_{u,v}$ is not one of the $\frac{3}{\epsilon}$ largest values in D'
17 **do** $d'_{u,v} \leftarrow 0$

Figure 3. Constructing D'

that is not of this form. However, for each $\sigma \in \Sigma$, $(\pi_i(\sigma), \tau_i(\sigma)$ is a uniformly random point in $[l_i] \times [r_i]$, which is a universe of size $O(\frac{1}{\epsilon})$. Given $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ random points in this universe with high probability each point appears at least once (this is the coupon collector problem). If $|\Sigma| = \Omega(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ then with high probability we will fail to recover any pair in the preimage.

Instead, we do the following. Assume that $\ell_i \geq r_i$. The other case of $\ell_i < r_i$ is dealt with by reversing the roles of π_i and τ_i. We first pick a random τ_i, and then set $\pi_i(\sigma) = \tau_i(\sigma) \mod r_i$. Recall that both τ_i and r_i are powers of 2. The following lemma bounds the probability that a pair the projection of a given pair of different characters (u, v) is discarded.

Lemma 5. *If $r_i \leq \ell_i$, then for a given pair of different characters (u, v) the probability that this pair is projected to a pair of the form $(\tau_i(\sigma), \pi_i(\sigma))$ for some $\sigma \in \Sigma$ is at most $\frac{1}{32}$.*

Proof:

$$\begin{aligned}\Pr[\exists_{\sigma \in \Sigma} : \tau_i(u) = \tau_i(\sigma) \wedge \pi_i(v) = \pi_i(\sigma)] &\leq \Pr[\exists_{\sigma \in \Sigma} : \pi_i(u) = \pi_i(\sigma) \wedge \pi_i(v) = \pi_i(\sigma)] \\ &\leq \Pr[\pi_i(u) = \pi_i(v)] = \frac{1}{r_i} \leq \frac{1}{32}\end{aligned}$$

■

Therefore, repeating the entire process with $O(\log n)$ choices of π_i and τ_i will guarantee that with high probability every pair of different characters is projected $\Omega(\log n)$ times to a pair that is not of this form (we will need this $\Omega(\log n)$ repetition later).

The Algorithm.: The pseudo-code for the algorithm is given in Figure 3. In lines 6-10 we create the projection functions, and then in lines 11-12 we compute $c_{x,y}$ for each projected pair (x, y) that is not of the form $(\pi_i(\sigma), \tau_i(\sigma))$ for any $\sigma \in \Sigma$, where $c_{x,y}$ is the exact number of alignments of this projected pair. We then use a bit-tester scheme using error-correcting codes (see [23]) in line 13 in order to establish if there is a pair

(u, v) in the preimage of (x, y) that contributed more than half of $c_{x,y}$. If so, then in line 14 we use $c_{x,y}$ to estimate $d_{u,v}$, unless a smaller estimator was encountered before. Finally, the algorithm filters away all but the largest $\frac{3}{\epsilon}$ entries in D', thereby guaranteeing that D' is sparse enough.

Projected-noise.: Consider a pair (u, v). For a given π_i and τ_i let

$$\sum_{(u',v')\neq(u,v):\tau_i(u)=\tau_i(u')\wedge\pi_i(v)=\pi_i(v')} d_{u',v'}$$

be the *projected-noise* for (u, v). We would like to bound the amount of projected-noise for a pair (u, v), since if it is less than $d_{u,v}$ then the *bit-tester* in line 13 will identify the pair (u, v) from the projected noise.In the following analysis we focus on a specific choice of i, τ_i, π_i, and a pair (u, v). The analysis is partitioned into two cases. In the first case we consider pairs (u', v') of the form either (u, v') or (u', v). In the second case we consider the remaining possible forms.

The first case.: We focus on pairs of the form either (u, v') or (u', v). Let $w(u) = \sum_{v'\in\Sigma} d_{u,v'}$, and let $w(v) = \sum_{u'\in\Sigma} d_{u',v}$. Notice that for any pair (u, v') the probability that $\pi_i(v) = \pi_i(v')$ is $\frac{1}{r_i}$. Similarly, for any pair (u', v) the probability that $\tau_i(u) = \tau_i(u')$ is $\frac{1}{\ell_i}$. Therefore, $E[\sum_{\substack{v'\neq v\\ \pi_i(v')=\pi_i(v)}} d_{u,v'}] = \frac{w(u)}{r_i}$ and $E[\sum_{\substack{u'\neq u\\ \pi_i(u')=\pi_i(u)}} d_{u',v}] = \frac{w(v)}{\ell_i}$. Thus the expected amount of projected noise from pairs of the form (u, v') or (u', v) is $\frac{w(u)}{r_i} + \frac{w(v)}{\ell_i}$. Recall that $\ell_i \cdot r_i = \frac{32^2}{\epsilon}$. This expected projected noise is minimized when $\ell_i = \sqrt{\frac{\epsilon w(u)}{32^2 w(v)}}$, and will then be $\frac{w(u)}{r_i} + \frac{w(v)}{\ell_i} = \frac{\sqrt{\epsilon w(u)w(v)}}{16}$. However, our algorithm is restricted to values of ℓ_i and r_i that are powers of 2. Within these limited options for ℓ_i and r_i the new minimized expected amount of projected noise is at most twice the minimum over all options for ℓ_i and r_i, and hence the expected amount of noise due to this type of pair is at most $\frac{\sqrt{\epsilon w(u)w(v)}}{8}$.

The second case.: We now focus on the remaining cases. Recall that the projections are 4-wise independent. Furthermore, recall that our algorithm ignores projected pairs of the form $(\tau_i(\sigma), \pi_i(\sigma))$ for any $\sigma \in \Sigma$. Therefore, we can ignore the case $u = v'$ since otherwise if $\tau_i(u) = \tau_i(u')$ and $\pi_i(v) = \pi_i(v')$ then $(\tau_i(u), \pi_i(v)) = (\tau_i(v'), \pi_i(v'))$ and the algorithm skips this projected pair. Similarly, we can ignore the case $u' = v$. Therefore, the only case that remains is that all four of u, u', v, v' are distinct. Since the projections are 4-wise independent, it must be that

$$\Pr[\tau_i(u) = \tau_i(u') \wedge \pi_i(v) = \pi_i(v')] = \frac{1}{\ell_i} \cdot \frac{1}{r_i} = \frac{\epsilon}{2^{10}}.$$

Thus the expected amount of such noise due to this type of pair is at most $\frac{\epsilon d}{2^{10}}$.

Bounding the noise.: Now, via Markov's inequality, with probability at least $\frac{1}{2}$ the amount of noise on (u, v) is at most $\frac{\sqrt{\epsilon w(u)w(v)}}{4} + \frac{\epsilon d}{2^9}$. Since each pair (u, v) is not projected into a pair of the form $\pi_i(\sigma), \tau_i(\sigma))$ for any $\sigma \in \Sigma$ at least $\Omega(\log n)$ times, and we pick the smallest estimation out of all choices of projections, then with high probability the total amount of noise on (u, v) is at most $\frac{\sqrt{\epsilon w(u)w(v)}}{4} + \frac{\epsilon d}{2^9}$.

The amount of noise on (u, v) can have two types of estimations for values of $d_{u,v}$. The first type of estimation comes from the case in which the noise on (u, v) is smaller than $d_{u,v}$. In this case the bit-tester will find (u, v) and so $d'_{u,v} \leq d_{u,v} + \frac{\sqrt{\epsilon w(u)w(v)}}{4} + \frac{\epsilon d}{2^9}$. In other words, this estimation overestimates the contribution of $d_{u,v}$ by at most the amount of noise on (u, v). The second type of estimation comes from the case in which the noise on (u, v) is at least $d_{u,v}$ and so $d'_{u,v} = 0$. This case will only concern us if (u, v) is a heavy hitter pair (in which case we may give this pair an estimation of 0). This case implies that the bit-tester failed to find (u, v), and so $\frac{\sqrt{\epsilon w(u)w(v)}}{4} + \frac{\epsilon d}{2^9} \geq d_{u,v}$, or $d'_{u,v} = 0 \geq d_{u,v} - \frac{\sqrt{\epsilon w(u)w(v)}}{4} + \frac{\epsilon d}{2^9}$.

Filtering.: The last part of the algorithm for constructing D' filters all but the largest $\frac{3}{\epsilon}$ estimators. We will now show that this filtering does not affect the estimation of any heavy hitter pair. If the estimator in D' for a heavy hitter pair was 0 prior to the filtering, then the estimator after the filtering is clearly unchanged. What

needs to be proven is that if the estimator in D' for a heavy hitter was larger than 0 prior to the filtering, then with high probability that estimator will be one of the $\frac{3}{\epsilon}$ largest entries, and so it remains unchanged due to the filtering. For the rest of the discussion here we assume that the values in D' are the values prior to the filtering.

By a simple counting argument, there are at most $\frac{2}{\epsilon}$ entries in D with a value that is at least $\frac{\epsilon}{2}d$. The following lemma will help us complete the proof.

Lemma 6. *With high probability, there are at most $\frac{1}{\epsilon}$ entries in D with value less than $\frac{\epsilon}{2}$ that are estimated in D' with value at least ϵd.*

Proof: Recall that there are two cases that contribute noise. With high probability the first case contributes $\frac{\sqrt{\epsilon w(u)w(v)}}{4}$ and the second case contributes $\frac{\epsilon d}{2^9}$. Let S be the set of pairs for which the noise from the first case is more than $\frac{\epsilon d}{4}$, which is required (but not sufficient) in order for their estimation to be at least ϵd. Then

$$|S|\left(\frac{\epsilon d}{4}\right)^2 \leq \sum_{(u,v)\in S}\left(\frac{\sqrt{\epsilon w(u)w(v)}}{4}\right)^2 = \sum_{(u,v)\in S}\frac{\epsilon w(u)w(v)}{16} = \frac{\epsilon}{16}\sum_{u,v\in\Sigma} w(u)w(v) = \frac{\epsilon d^2}{16}.$$

Therefore, $|S| \leq \frac{1}{\epsilon}$. ∎

Corollary V.1. *Prior to filtering there are at most $\frac{3}{\epsilon}$ entries in D' with value at least ϵd is at most.*

Finally, notice that a heavy hitter pair (u, v) such that $d'_{u,v} > 0$ must have $d'_{u,v} \geq \epsilon d$, and so this entry cannot be filtered away.

Bounding the variance.: Notice that there are at most $\frac{3}{\epsilon}$ non-zero entries in D' and at most $\frac{1}{\epsilon}$ heavy hitters, and recall that some heavy hitter pairs may be estimated in D' with 0. Finally, with high probability we have

$$\begin{aligned}
\sum_{u,v\in\Sigma}(d_{u,v}-d'_{u,v})^2 &= \sum_{\substack{u,v\in\Sigma\\ d_{u,v}<\epsilon d\\ d'_{u,v}=0}}(d_{u,v})^2 + \sum_{\substack{u,v\in\Sigma\\ d_{u,v}\geq\epsilon d\vee d'_{u,v}\neq 0}}(d_{u,v}-d'_{u,v})^2\\
&\leq \frac{\epsilon d^2}{2} + \sum_{\substack{u,v\in\Sigma\\ d_{u,v}\geq\epsilon d\vee d'_{u,v}\neq 0}}\left(\frac{\sqrt{\epsilon\cdot w(u)\cdot w(v)}}{4}+\frac{\epsilon d}{2^9}\right)^2\\
&\leq \frac{\epsilon d^2}{2} + \sum_{\substack{u,v\in\Sigma\\ d_{u,v}\geq\epsilon d\vee d'_{u,v}\neq 0}}\left(\frac{\sqrt{\epsilon\cdot w(u)\cdot w(v)}}{2}\right)^2 + \sum_{\substack{u,v\in\Sigma\\ d_{u,v}\geq\epsilon d\vee d'_{u,v}\neq 0}}\left(\frac{\epsilon d}{2^8}\right)^2\\
&\leq \frac{\epsilon d^2}{2} + \left(\sum_{\substack{u,v\in\Sigma\\ d_{u,v}\geq\epsilon d\vee d'_{u,v}\neq 0}}\frac{\epsilon\cdot w(u)\cdot w(v)}{4}\right) + \frac{4\epsilon d^2}{2^{16}}\\
&\leq \frac{\epsilon d^2}{2} + \frac{\epsilon d^2}{4} + \frac{\epsilon d^2}{2^{14}} \leq \frac{2^{13}+2^{12}+1}{2^{14}}\epsilon d^2 = b\epsilon d^2
\end{aligned}$$

VI. Constructing the Projections

We now turn our focus towards constructing the projections in line 4 from Figure 2, so that the computation in line 5 will take $\tilde{O}(\frac{1}{\epsilon})$ time. In particular, we will show that computing

$$\sum_{i=1}^{k}\left(\frac{1}{2}\sum_{\substack{u,v\\ h_i(u)=h_i(v)}} d'_{u,v} - \frac{1}{2}\sum_{\substack{u,v\\ h_i(u)\neq h_i(v)}} d'_{u,v}\right)$$

can be done in $\tilde{O}(\frac{1}{\epsilon}\log k)$ time.

To start off, notice that it suffices to know for each $d'_{u,v} > 0$ the number $\beta_{u,v} = |\{1 \leq i \leq k : h_i(u) = h_i(v)\}|$, since

$$\sum_{i=1}^{k}\left(\sum_{\substack{u,v\\ h_i(u)=h_i(v)}} d'_{u,v} - \sum_{\substack{u,v\\ h_i(u)\neq h_i(v)}} d'_{u,v}\right) = \sum_{d'_{u,v}>0}(\beta_{u,v} - (k-\beta_{u,v}))d'_{u,v}.$$

Once we show how to construct the hash functions $h_1, \ldots, h_k$ so that we can compute $\beta_{u,v}$ for $d'_{u,v} \neq 0$ in $O(\log k)$ time, then we are done.

The construction.: Assume without loss of generality that k is a power of 2. Consider $2\log k$ base hash functions $f_1, \ldots f_{2\log k} : \Sigma \to [0,1]$ where each base hash function is picked independently from a 4-wise independent family of hash functions. We partition the base hash functions to $\log k$ pairs, and consider all possible combinations of picking 1 function from each pair. Each combination of $\log k$ functions defines a different projection hash function h_i by considering the xor of the outputs of the $\log k$ functions. The number of projection hash functions is exactly the number of combinations of base hash functions in our construction, which is $(2)^{\log k} = k$, as required.

We now argue that each projection hash function h_i is 4-wise independent, and that the projection hash functions are pair-wise independent among themselves. Since h_i is the xor of $\log k$ 4-wise independent base functions, h_i is 4-wise independent as well. Moreover, for each $h_i \neq h_j$, there must be at least one pair of base functions in which h_i uses one base function and h_j uses the other base function. Since these base functions are independent, h_i and h_j must be independent as well.

Computing $\beta_{u,v}$.: Set (u,v). Build a balanced binary tree over the $\log k$ pairs of base functions and compute $f_1(u), f_2(u), \ldots f_{2\log k}(u)$ and $f_1(v), f_2(v), \ldots f_{2\log k}(v)$. For each node w in the balanced binary tree let t_w be the number of pairs of base functions at the leaves of the subtree of w. For each such t_w pairs of base functions, we consider all 2^{t_w} combinations of picking one base function from each pair. Each such combination defines a projection function local to w by taking the xor of the outputs of the base hash functions for that combination. Let e_w be the number of such local projections for which the projection on u and the projection on v are equal, while d_w is the number of such projections for which the projection on u and the projection on v are not equal. If ℓ and r are the left and right children of w, respectively, then $e_w = e_\ell \cdot e_r + d_\ell \cdot d_r$ and $d_w = e_\ell \cdot d_r + d_\ell \cdot e_r$. Finally, since the root of the balanced binary tree $root$ covers all of the $\log k$ pairs of base functions, then $\beta_{u,v} = e_{root}$. Thus using a direct bottom up approach we can compute $\beta_{u,v}$ in $O(\log k)$ time.

REFERENCES

[1] K. Abrahamson. Generalized string matching. In *SIAM J. Computing 16 (6)*, page 10391051, 1987.

[2] A. Amir, O. Lipsky, E. Porat, and J. Umanski. Approximate matching in the l1 metric. In *CPM*, pages 91–103, 2005.

[3] Amihood Amir, Yonatan Aumann, Gary Benson, Avivit Levy, Ohad Lipsky, Ely Porat, Steven Skiena, and Uzi Vishne. Pattern matching with address errors: rearrangement distances. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 1221–1229, 2006.

[4] Amihood Amir, Yonatan Aumann, Piotr Indyk, Avivit Levy, and Ely Porat. Efficient computations of l_1 and l_{infinity} rearrangement distances. In *String Processing and Information Retrieval, 14th International Symposium, SPIRE 2007, Santiago, Chile, October 29-31, 2007, Proceedings*, pages 39–49, 2007.

[5] Amihood Amir, Yonatan Aumann, Oren Kapah, Avivit Levy, and Ely Porat. Approximate string matching with address bit errors. In *Combinatorial Pattern Matching, 19th Annual Symposium, CPM 2008, Pisa, Italy, June 18-20, 2008, Proceedings*, pages 118–129, 2008.

[6] Amihood Amir, Estrella Eisenberg, and Ely Porat. Swap and mismatch edit distance. In *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, pages 16–27, 2004.

[7] Amihood Amir, Tzvika Hartman, Oren Kapah, Avivit Levy, and Ely Porat. On the cost of interchange rearrangement in strings. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 99–110, 2007.

[8] Amihood Amir, Moshe Lewenstein, and Ely Porat. Approximate swapped matching. In *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS 2000 New Delhi, India, December 13-15, 2000, Proceedings.*, pages 302–311, 2000.

[9] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386, 2010.

[10] Mikhail J. Atallah, Elena Grigorescu, and Yi Wu. A lower-variance randomized algorithm for approximate string matching. *Inf. Process. Lett.*, 113(18):690–692, 2013.

[11] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Accepted to 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

[12] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 550–559, 2004.

[13] Ayelet Butman, Noa Lewenstein, Benny Porat, and Ely Porat. Jump-matching with errors. In *String Processing and Information Retrieval, 14th International Symposium, SPIRE 2007, Santiago, Chile, October 29-31, 2007, Proceedings*, pages 98–106, 2007.

[14] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 693–703, 2002.

[15] Raphael 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 August 2015.

[16] Raphaël Clifford, Klim Efremenko, Benny Porat, Ely Porat, and Amir Rothschild. Mismatch sampling. *Information and Computation*, 214:112–118, 2012.

[17] Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. pages 778–784, 2009.

[18] Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. Pattern matching with don't cares and few errors. *Journal of Computer System Science*, 76(2):115–124, 2010.

[19] Raphaël Clifford and Ely Porat. A filtering algorithm for *k*-mismatch with don't care s. *Information Processing Letters*, 110(22):1021–1025, 2010.

[20] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 667–676, 2002.

[21] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*, pages 29–38, 2004.

[22] M.J. Fischer and M.S. Paterson. String matching and other products. r.m. karp (ed.), complexity of computation. In *SIAMAMS Proceedings, vol. 7,*, page 113125, 1974.

[23] A. C. Gilbert, Y. Li, E. Porat, and M. J. Strauss. Approximate sparse recovery: optimizing time and measurements. In *STOC*, pages 475–484, 2010.

[24] A. C. Gilbert, Y. Li, E. Porat, and M. J. Strauss. For-all sparse recovery in near-optimal time. In *ICALP (1)*, pages 538–550, 2014.

[25] A. C. Gilbert, H. Q. Ngo, E. Porat, A. Rudra, and M. J. Strauss. L2/l2-foreach sparse recovery with low risk. In *ICALP (1)*, pages 461–472, 2013.

[26] Anna C. Gilbert, Martin J. Strauss, Joel A. Tropp, and Roman Vershynin. One sketch for all: fast algorithms for compressed sensing. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 237–246, 2007.

[27] Piotr Indyk and Milan Ruzic. Near-optimal sparse recovery in the L1 norm. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 199–207, 2008.

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

[29] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 745–754, 2011.

[30] H. Karloff. Fast algorithms for approximately counting mismatches. In *Inf. Process. Lett. 48 (2)*, pages 53–60, 1993.

[31] Vladimir Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. In *Probl. Inf. Transmission 1*, page 817, 1965.

[32] O. Lipsky and E. Porat. Approximated pattern matching with the l1, l2 and linfinit metrics. In *SPIRE*, pages 212–223, 2008.

[33] R. Lowrance and R. A. Wagner. An extension of the string-to-string correction problem. *J. of the ACM*, pages 177–183, 1975.

[34] Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 315–323, 2009.

[35] Benny Porat, Ely Porat, and Asaf Zur. Pattern matching with pair correlation distance. In *String Processing and Information Retrieval, 15th International Symposium, SPIRE 2008, Melbourne, Australia, November 10-12, 2008. Proceedings*, pages 249–256, 2008.

[36] E. Porat and M. J. Strauss. Sublinear time, measurement-optimal, sparse recovery for all. In *SODA*, pages 1215–1227, 2012.

[37] Ely Porat and Klim Efremenko. Approximating general metric distances between a pattern and a text. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 419–427, 2008.

[38] Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, pages 173–182, 2007.

[39] Ariel Shiftan and Ely Porat. Set intersection and sequence matching. In *String Processing and Information Retrieval, 16th International Symposium, SPIRE 2009, Saariselkä, Finland, August 25-27, 2009, Proceedings*, pages 285–294, 2009.

[40] David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 167–175, 2004.