2016 IEEE 57th Annual Symposium on Foundations of Computer Science

Edit Distance: Sketching, Streaming and Document Exchange *

Djamal Belazzougui
Center for Research on Scientific and Technical Information (CERIST)
Algiers, 16306, Algeria
dbelazzougui@cerist.dz

Qin Zhang †
Computer Science Department
Indiana University Bloomington
Bloomington, IN 47401, USA
qzhangcs@indiana.edu

***Abstract*—We show that in the document exchange problem, where Alice holds $x \in \{0,1\}^n$ and Bob holds $y \in \{0,1\}^n$, Alice can send Bob a message of size $O(K(\log^2 K + \log n))$ bits such that Bob can recover x using the message and his input y if the edit distance between x and y is no more than K, and output "error" otherwise. Both the encoding and decoding can be done in time $\tilde{O}(n + \mathsf{poly}(K))$. This result significantly improves on the previous communication bounds under polynomial encoding/decoding time. We also show that in the referee model, where Alice and Bob hold x and y respectively, they can compute sketches of x and y of sizes $\mathsf{poly}(K \log n)$ bits (the encoding), and send to the referee, who can then compute the edit distance between x and y together with all the edit operations if the edit distance is no more than K, and output "error" otherwise (the decoding). To the best of our knowledge, this is the *first* result for sketching edit distance using $\mathsf{poly}(K \log n)$ bits. Moreover, the encoding phase of our sketching algorithm can be performed by scanning the input string in one pass. Thus our sketching algorithm also implies the *first* streaming algorithm for computing edit distance and all the edits exactly using $\mathsf{poly}(K \log n)$ bits of space.**

***Keywords*-document exchange; sketching; streaming; edit distance**

I. Introduction

In this paper we study the problem of *edit distance*, where given two strings $s, t \in \{0,1\}^n$, we want to compute $ed(s,t)$, which is defined to be the minimum number of insertions, deletions and substitutions to convert s to t. We also want to find all the edit operations. This problem has been studied extensively in the literature due to its numerous applications in bioinformatics (comparing the similarity of DNA sequences), natural language processing (automatic spelling correction), information retrieval, etc. In this paper we are interested in the small distance regime, that is, given a threshold K, we want to output $ed(s,t)$ together with all the edit operations if $ed(s,t) \leq K$, and output "error" otherwise. We will explain shortly that this is the interesting regime for many applications. We consider three different settings:

- *Document Exchange*. We have two parties Alice and Bob, where Alice holds s and Bob holds t. The task is for Alice to compute a message msg based on s (the encoding) and send to Bob, and then Bob needs to recover s using msg and his input t (the decoding). We want to minimize both the message size and the encoding/decoding time.
- *Sketching*. We have Alice and Bob who hold s and t respectively, and a third party called the *referee*, who has no input. The task is for Alice and Bob to compute sketches $sk(s)$ and $sk(t)$ based on their inputs s and t respectively (the encoding), and send them to the referee. The referee then computes $ed(s,t)$ and all the edits using $sk(s)$ and $sk(t)$ (the decoding). The goal is to minimize both the sketch size and the encoding/decoding time.
- *Streaming*. We are allowed to scan string s from left to right once, and then string t from left to right once, using a memory of small size. After that we need to compute $ed(s,t)$ and all the edits using the information retained in the memory. The goal is to minimize the memory space usage and the processing time.

Motivations. Document exchange is a classical problem that has been studied for decades. This problem finds many applications, for example, two versions of the same file are stored in two different machines and need to be synchronized, or we want to restore a file transmitted through a channel with erroneous insertions, deletions and substitutions. It is useful to focus on the small distance regime since one would expect that in the first application the two files will not differ by much, and in the second application the channel will not introduce too many errors. Otherwise we can detect

* The full version of this paper can be found at [1].

† Supported in part by NSF CCF-1525024, IIS-1633215, and IU's Office of the Vice Provost for Research through the FRSP.

0272-5428/16 $31.00 © 2016 IEEE
DOI 10.1109/FOCS.2016.15

the exception and ask the sender to transmit the whole string again, which is a low probability event thus we can afford.

Sketching the edit distance is harder than document exchange because in the decoding phase the referee does not have access to any of the original strings, which, however, also makes the problem more interesting and useful. For example, in the *string similarity join*, which is a fundamental problem in databases, one needs to find all pairs of strings (e.g., genome sequences) in a database that are close (no more than a given threshold K) with respect to edit distance. This is a computationally expensive task. With efficient sketching algorithms we can preprocess in parallel each string to a small size sketch, and then compute the edit distances on those small sketches without losing any accuracy (all of our algorithms aim at exact computations).

Our Results. In this paper we push the frontiers further for all of the three problems. For the convenience of the presentation we use $\tilde{O}(f)$ to denote $f\mathsf{poly}(\log f)$, and further assume that $K \leq n^{1/c}$ for a sufficiently large constant $c > 0$ (thus n absorbs $\mathsf{poly}(K)$ factors). We list our results together with previous results in Table I. Our contribution includes:

1) We have improved the communication cost of the document-exchange problem to $O(K(\log^2 K + \log n))$ bits while maintaining almost linear running time. Note that when $\log K = O(\sqrt{\log n})$, our communication bound matches the information theoretic lower bound $\Omega(K \log n)$.
2) We have obtained a sketch of size $O(K^8 \log^5 n)$ bits for edit distance, which, to the best of our knowledge, is the *first* result for sketching edit distance in size $\mathsf{poly}(K \log n)$. This result answers an open problem in [2], and is in contrast to the lower bound result in [3] which states that any *linear* sketch for edit distance has to be of size $\Omega(n/\alpha)$ even when we want to distinguish $ed(s,t) \geq 2\alpha$ or $ed(s,t) \leq 2$.
3) Our sketching algorithm can be implemented in the standard streaming model. To the best of our knowledge, this is the *first* efficient streaming algorithm for exact edit distance computation. We also show that if we can scan s and t simultaneously in the coordinated fashion, we can compute the edit distance using only $O(K \log n)$ space, which significantly improves on the result in [4].

Due to the space constraints, we refer readers to [1] for all the omitted proofs in this extended abstract, and the algorithms for standard streaming and simultaneous streaming.

problem	comm. / size / space	time	ref.
doc-exchange	$O(K \log n)$	$n^{O(K)}$	[5]
	$O(K \log(n/K) \log n)$	$\tilde{O}(n)$	[6]
	$O(K \log^2 n \log^* n)$	$\tilde{O}(n)$	[2]
	$O(K^2 + K \log^2 n)$	$\tilde{O}(n)$	[7]
	$O(K^2 \log n)$	$\tilde{O}(n)$	[4]
	$O(K(\log^2 K + \log n))$	$\tilde{O}(n)$	**new**
sketching	$O(K^8 \log^5 n)$	$\tilde{O}(K^2 n)$ (enc.), $\mathsf{poly}(K \log n)$ (dec.)	**new**
streaming	$O(K^8 \log^5 n)$	$\tilde{O}(K^2 n)$	**new**
simul-streaming	$O(K^6 \log n)$	$\tilde{O}(n)$	[4]
	$O(K \log n)$	$O(n)$	**new**

Table I
Our results for computing edit distance in different models. n is the input size and K is a given upper bound of the edit distance. We have assumed that $K \leq n^{1/c}$ for a sufficiently large constant $c > 0$, and thus n absorbs $\mathsf{poly}(K)$ factors. All the algorithms are randomized except those in [5], [7], and our new simultaneous streaming algorithm, which are deterministic.

Related Work. We list a few closely related works. We refer readers to the full version of the paper for more related work.

Document Exchange. The first algorithm for document exchange was proposed by Orlitsky [5], who gave a communication optimal algorithm but with decoding time exponential in K. Since then, the running time has been improved [8], [6], [2]. Irmak et al. [6] proposed a randomized protocol using an erasure-correcting-code, achieving $O(K \log(n/K) \log n)$ bits of communication and $\tilde{O}(n)$ encoding/decoding time, and (independently) Jowhari [2] gave a randomized protocol using the ESP-tree [9] that achieves $O(K \log^2 n \log^* n)$ bits of communication and $\tilde{O}(n)$ encoding/decoding time. Very recently Chakraborty et al. [4] obtained a protocol with $O(K^2 \log n)$ bits of communication and $\tilde{O}(n)$ encoding/decoding time, by first embedding strings in the edit space to the Hamming space using random walks, and then performing the document exchange in the Hamming space. We note that random walk has also been used for computing the Dyck language edit distance in an earlier paper by Saha [10].

Sketching. While the approximate version has been studied extensively in the literature, little work has been done for sketching edit distance without losing any accuracy. Jowhari [2] gave a sketch of size $\tilde{O}(K \log^2 n)$ bits for a special case of edit distance called the Ulam distance, where the alphabet size is n and each string

has no character repetitions.[1] Andoni et al. [3] showed that if we require the sketch for edit distance to be linear, then its size must be $\Omega(n/\alpha)$ even when we want to distinguish whether the distance is at least 2α or no more than 2.

Streaming. To the best of our knowledge, computing exact edit distance has not been studied in the streaming model. Chakraborty et al. [4] studied this problem in a variant of the streaming model where we can scan the two strings x and y simultaneously in a coordinated fashion, and showed that $O(K^6 \log n)$ bits of space is enough to computing $ed(x, y)$.

Techniques Overview. We now summarize the high level ideas of our algorithms. The parameters used in this overview are just for illustration purposes.

Document Exchange. As mentioned, the document exchange problem can be solved by the algorithm of Irmak et al. [6] (call it the IMS algorithm) using $O(K \log^2 n)$ bits of communication. Intuitively speaking, IMS first converts strings s and t to two binary parse trees $\mathcal{T}_s$ and $\mathcal{T}_t$ respectively, where the root of the tree corresponds to the hash signature of the whole string, the left child of the root corresponds to the hash signature of the first half of the string, and the right child corresponds to that of the second half, and so on. IMS then tries to *synchronize* the two parse trees and for the receiver to identify on $\mathcal{T}_s$ at most K root-leaf paths which lead to the at most K edits. The synchronization is done using error-correcting codes at each level of the tree.

The main idea of our new algorithm is that if we can identify those large common blocks in some optimal alignment between s and t, then we can skip those common blocks and effectively reduce s, t to two strings s', t' of much smaller sizes, say, each consisting of at most K substrings each of size at most K^{99}. Now if we apply the IMS algorithm on s' and t' we only need $O(K \log^2 K^{99}) = O(K \log^2 K)$ bits of communication. The question now is how to identify those large common blocks, which turns out to be quite non-trivial. Note that Alice has to do this step independently in the one-way communication model, and she does not even have a good alignment between s and t.

Our main tool is the embedding result by Chakraborty et al. [4] (denoted by the *CGK embedding*): we can embed binary strings s and t of size n to binary strings s' and t' of size $3n$ independently, such that if $ed(s, t) = k$ $(\leq K)$, then with probability 0.99 we have $\Omega(k) \leq ham(x, y) \leq O(k^2)$, where $ham(\cdot, \cdot)$ denotes the Hamming distance. In the (good) case that after the embedding, the $O(k^2)$ mismatches in s' and t' (in the Hamming space) are distributed into at most K pairs of trunks each of length at most K^{99}, then we can identify those mismatched trunks as follows: We partition s' and t' into blocks of size $100\sqrt{n}$, and then map those blocks back to substrings in s and t (the edit space). We next use an error-correcting code to identify those (up to K) pairs of substrings of s and t that differ, and recurse on those mismatching pairs. By doing this we will have effectively reduced s and t to at most K substrings each of length $100\sqrt{n}$ after the first round. Then after $O(\log \log n)$ recursion rounds we can reduce the length of each of the (at most) K substrings to K^{99}, at which moment we apply the IMS algorithm.

The subtlety comes from the fact that if the strings s and t contain long common periodic substrings of sufficiently short periods, then the $O(k^2)$ mismatches will possibly be distributed into $O(k^2)$ remote locations in s' and t' (note that we cannot recurse on k^2 substrings since that will introduce a factor of k^2 in the message size). More precisely, the random walk used in the CGK embedding may get "stuck" in the common periodic substrings in s' and t', and consequently spread the mismatches to remote locations. We thus need to first carefully remove those long common periodic substrings in s and t (again Alice has to do this independently), and then apply the above scheme to reduce the problem size.

Sketching. We can view an alignment $\mathcal{A}$ between s and t as a bipartite matching, where nodes are characters in s and t, and edges correspond to those aligned pairs (i, j) in $\mathcal{A}$. The matching can naturally be viewed as a group of clusters each consisting of a set of consecutive edges $\{(i, j), (i+1, j+1), \ldots\}$, plus some singleton nodes in between. Now let $sk(\mathcal{A})$ be a sketch of $\mathcal{A}$ containing the first and last edges of each cluster in $\mathcal{A}$ plus all the singleton nodes. Intuitively, if $ed(s, t) \leq K$ then for a good alignment $\mathcal{A}$, the size of $sk(\mathcal{A})$ can be much smaller than that of $\mathcal{A}$.

Given a collection of matchings $\{\mathcal{A}_1, \ldots, \mathcal{A}_\rho\}$, letting $\mathcal{I} = \bigcap_{j \in [\rho]} \mathcal{A}_j$ be the set of common edges of $\mathcal{A}_1, \ldots, \mathcal{A}_\rho$, our main idea is the following: if there exists an optimal alignment that goes through all edges in $\mathcal{I}$, then we can produce an optimal alignment for strings s and t using $\{sk(\mathcal{A}_1), \ldots, sk(\mathcal{A}_\rho)\}$.

We now again make use of the CGK embedding, which can be thought of as a random walk running on two strings s and t of size n in the edit space,

[1] In this paper when considering edit distance we always assume binary alphabet, but our results can be easily carried to non-binary alphabets as long as the alphabet size is no more than $\mathsf{poly}(n)$. We note that the embedding result by Chakraborty et al. can be applied to strings with non-binary alphabets (see [4] for details).

and producing two strings s' and t' of size $3n$ in the Hamming space such that $ham(s', t') = \mathsf{poly}(K \log n)$ with high probability. Each random walk consists of a set of states $\{(p_1, q_1), \ldots, (p_m, q_m)\}$ $(p_j, q_j \in [n])$, which naturally corresponds to an alignment between s and t. We say a random walk passes a pair (u, v) if there exists some $j \in [m]$ such that $(p_j, q_j) = (u, v)$. Our key observation is that given $\rho = K^2 \log n$ random walks according to the CGK embedding, for any pair (u, v) with $s[u] = t[v]$, if all the ρ random walks pass (u, v), then (u, v) must be part of a particular optimal alignment (see Lemma 13). We thus only need to compute the sketches of the alignments corresponding to those random walks, each of which corresponds to the differences between s' and t' in the Hamming space, for which efficient sketching algorithms have already been obtained. Moreover, the size of each sketch can be bounded by $\mathsf{poly}(K \log n)$ using the fact that $ham(s', t') = \mathsf{poly}(K \log n)$. The last step is to map these differences in the Hamming space back to the edit space for computing an optimal alignment, which requires us to add to the sketches some position-aware structures to assist the reverse mapping. The whole sketch consists of $\rho = K^2 \log n$ sub-sketches each of size $\mathsf{poly}(K \log n)$, and is thus of size $\mathsf{poly}(K \log n)$.

Streaming. Our algorithm for the standard streaming model follows directly from our sketching algorithm, since the encoding phase of our sketching algorithm can be performed using a one-pass scan on the input string. Our result in the simultaneous streaming model is obtained by implementing the classic dynamic programming algorithm for edit distance in a space and time efficient way, more precisely, by only trying to compute those must-know cells in the alignment matrix.

Notations. We use n as the input size, and K as the threshold under which we can compute $ed(s, t)$ and the edit operations successfully.

Denote $[i..j] = \{i, i+1, \ldots, j\}$ and $[n] = [1..n]$. When we write $x[i..j]$ for a string x we mean the substring $(x[i], \ldots, x[j])$. We use "$\circ$" for string concatenation. All logs are base-2 unless noted otherwise.

II. Preliminaries

In this section we present some tools and previous results that we need in our algorithms.

The CGK Embedding. A basic procedure in our algorithms is the embedding from edit space to Hamming space introduced in [4]. The embedding is parameterized with a random string $r \in \{0,1\}^{6n}$, and maps $s \in \{0,1\}^n$ to $s' \in \{0,1\}^{3n}$. We use two counters i and j both initialized to 1. Counter i points to s and counter j points to s'. The algorithm proceeds in steps $j = 1, 2, \ldots$. At step j it does:

1) $s'[j] \leftarrow s[i]$.
2) If $r[(2j-1) + s[i]] = 1$, then $i \leftarrow i + 1$. Stop when $i = n + 1$.
3) $j \leftarrow j + 1$.

At the end, if $j < 3n$, then we append $(3n - j)$ '0's to s to make it a string of length $3n$. In words, at each step j the algorithm reads a bit from s indexed by i and copies it to the output string s' at position j. We then decide whether to increment the index i using the $((2j-1) + s[i])$-th bit of string r.

We are interested in comparing the Hamming distance between strings s' and t' produced by the embeddings on $s \in \{0,1\}^n$ and $t \in \{0,1\}^n$ respectively. Let i_0 be a counter for s, i_1 be a counter for t, and j be a counter denoting the current time step. At time step j, the bit $s[i_0]$ is copied to $s'[j]$, and the bit $t[i_1]$ is copied to $t'[j]$. Then one of the following four cases will happen: (1) neither i_0 nor i_1 increments; (2) only i_0 increments; (3) only i_1 increments; (4) both i_0 and i_1 increment. Note that if $s[i_0] = t[i_1]$, then only first and last cases can happen, and the bits copied into s' and t' are the same. Otherwise if $s[i_0] \neq t[i_1]$, then the bits copied into s' and t' are different. Each of the four cases happens with probability $1/4$ depending on the random string r. We have the following definitions.

Definition 1 (State and Progress Step). *We call the sequence of (i_0, i_1) the* states *of the random walk. We say that we have a* progress step *if $s[i_0] \neq t[i_1]$ and at least one of i_0 and i_1 increments.*

We can model the evolution of the "shift" $d = (i_0 - i_1)$ as (a different) random walk on the integer line, where at each time step, if $s[i_0] \neq t[i_1]$ then d stays the same with probability $1/2$, decrements by 1 with probability $1/4$, and increments by 1 with probability $1/4$, otherwise if $s[i_0] = t[i_1]$ then d always stays the same. We can focus on the cases when one of i_0 and i_1 increments, and view the change of d as a *simple* random walk on the integer line, where at each step the shift d increments or decrements with equal probability.

The following lemma has been shown in [4] by using the properties of simple random walks.

Lemma 1. *Let (p_0, q_0) be a state of a random walk $\mathcal{W}$ according to the CGK embedding in which $d = ed(s[p_0..n], t[q_0..n]) \geq 1$, then with probability $1 - O(1/\sqrt{\ell})$, $\mathcal{W}$ reaches within ℓ progress steps a state (p_1, q_1) with $ed(s[p_1..n], t[q_1..n]) \leq d - 1$.*

The IMS Algorithm. As mentioned, we will use the

IMS algorithm proposed in [6]. In fact, we can slightly improve the original IMS algorithm in [6] to get the following result.

Theorem 2. *There exists an algorithm for document exchange having communication cost $O(K(\log K + \log\log n)\log n))$, running time $\tilde{O}(n)$, and success probability $1 - 1/\mathsf{poly}(K \log n)$, where n is the input size and K is the distance upper bound.*

Periods and Random Walk. When performing the CGK embedding, common periods in the strings may "slow down" the random walk, and consequently distribute the mismatches into remote locations (which is undesirable for our algorithm for document exchange; see the techniques overview in the introduction). On the other hand, if two strings do not share long periodic substrings of small periods, then the random walk induced by the embedding will make steady progress.

Lemma 3. *Suppose that we have $w = s[i..i+m] = t[j..j+m]$ and that the substring w has no substring of length $\ell < m$ with period at most θ. Then, the random walk induced by the CGK embeddings of s and t will have the following property: suppose at a given step the random walk is in the state (p, q) such that $|(p-i) - (q-j)| \in [1..\theta]$, $p \in [i..i+m-\ell]$ and $q \in [j..j+m-\ell]$, then there will be a progress step for some pair (u_0, u_1) with $u_0 \in [p, p+\ell-1]$ and $u_1 = u_0 + (q-p)$.*

Error-Correcting. We consider the following problem: Alice holds a vector a and Bob holds a vector b, where a and b are of length u over an alphabet of size σ. Let k be a given threshold. Bob knows that the differences between a and b fall into a set S of $\lambda \geq k$ coordinates, but Alice does not know the set S. The task is for Alice to send a message msg to Bob so that Bob can recover a exactly based on msg and his input string b if $ham(a, b) \leq k$, and output "error" otherwise.

Lemma 4. *There exists an algorithm for the above problem having communication cost $O(\log\log u + k(\log\sigma + \log\lambda + \log(1/p))$, running time $\tilde{O}(u + k\lambda)$, and success probability $1 - p$.*

Sketching Hamming Distance. We will use the result for sketching Hamming distance as a building block in our sketching algorithm. In particular we will use the one in [11], and state it for a general alphabet.

Lemma 5 ([11])**.** *There exists a sketching algorithm which, given a vector of length n over an alphabet of size $\sigma = O(\mathsf{poly}(n))$ and a threshold k, outputs a vector of length $O(k \log n)$, such that given the sketches of two vectors a and b, one can recover with probability $(1 - 1/\mathsf{poly}(n))$ the coordinates (indices and values) where they differ in time $O(k \log n)$ if $ham(a, b) \leq k$, and outputs "error" otherwise. The sketching process can be done in the one-pass streaming fashion and runs in $O(\log n)$ time per element.*

III. Document Exchange

In this section we prove the following theorem.

Theorem 6. *There exists an algorithm for document exchange having communication cost $O(K(\log^2 K + \log n))$, running time $\tilde{O}(n)$, and success probability $1 - 1/\mathsf{poly}(K \log n)$, where n is the input size and K is the distance upper bound.*

A. The Algorithm

Let $k = ed(s, t)$ be the edit distance between s and t that we want to compute. Our algorithm for document exchange consists of two stages. After the first stage we effectively reduce the problem to a much smaller size, more precisely, to that on two strings each consisting of at most $k \leq K$ substrings each of size at most $(K2^{\sqrt{\log n}})^{O(1)}$, while preserving the edit distance. We will then in the second stage run the IMS algorithm on the reduced problem to compute the distance and all the edits. We thus only describe the first stage of the algorithm.

The first stage consists of $L = O(\log\log n)$ levels, each of which consists of two phases. These levels and phases are executed in synchrony between Alice and Bob, but the whole communication is still one-way. We now describe the two phases at each level $\ell \in [L]$. We will use the following parameters, which are known to both parties before running the algorithm. Let c_1, c_2 be two sufficiently large constants.

- $b_\ell = \sqrt{n_\ell} 2^{c_1(\log K + \sqrt{\log n})}$: block size in the first phase.
- $b'_\ell = \sqrt{n_\ell} 2^{(c_1+c_2)(\log K + \sqrt{\log n})}$: block size in the second phase.
- $\theta_\ell = b_\ell/2$: upper bound of period length.

Let n_ℓ be an upper bound of the effective size of the problem at the beginning of the ℓ-th level. $n_1 = n = |s| = |t|$, and $n_\ell = K b'_{\ell-1}$ $(\ell \geq 2)$.

Phase I: Alice partitions her string x into blocks of size b_ℓ, except that the first block is of a uniformly random size $\Delta_\ell \in [b_\ell/2, b_\ell]$, and the last block is of size in the range $[1..b_\ell]$. Denote these blocks by $B_1, \ldots, B_m$. Alice sends Δ_ℓ to Bob.

Next, Alice creates a vector $U_\ell = (e_1, \ldots, e_m)$ where $e_j = (\chi_j, w_j)$ contains the following information of the j-th block B_j: If B_j is part of a periodic substring with period length at most θ_ℓ, then she sets $\chi_j = 1$ and

w_j to the period length;[2] otherwise she sets $\chi_j = 0$ and $w_j = 0$. Alice then sends to Bob a redundancy of U_ℓ (denoted as $sk(U_\ell)$) that allows to recover up to $2K$ errors using the scheme in Lemma 4 (setting $p = 1/\mathsf{poly}(K \log n)$).

Bob maintains a vector $\tilde{x}$ based on his decoding of Alice's string x in the previous $(\ell - 1)$ levels such that with high probability $\tilde{x}$ only differs from x at no more than n_ℓ coordinates (those that Bob hasn't recovered), for which Bob marks '$\perp$'. ($\tilde{x}$ is initialized to an all-'$\perp$' vector of length n at the beginning of the algorithm.) After receiving the offset Δ_ℓ he does the same partitioning on both $\tilde{x}$ and y, getting $(P_1, \ldots, P_m)$ and $(Q_1, \ldots, Q_m)$ respectively. He then examines each block P_j. If P_j or the θ_ℓ positions in $\tilde{x}$ preceding P_j contain a '$\perp$', then he builds $e'_j = (\chi'_j, w'_j)$ in the same way as $e_j = (\chi_j, w_j)$ by looking at Q_j's context in y; otherwise he builds $e'_j = (\chi'_j, w'_j)$ in the same way as $e_j = (\chi_j, w_j)$ by looking at P_j's context in $\tilde{x}$. Let $U'_\ell = (e'_1, \ldots, e'_m)$. We will show in the analysis that Bob can recover Alice's vector U_ℓ using $sk(U_\ell)$ and his vector U'_ℓ with high probability.

Finally, for each maximal sequence of consecutive blocks contained in the same periodic substring $\mathtt{str}$ of x, and denoted by $B_1 \circ B_2 \cdots \circ B_{z-1} \circ B_z$, Alice removes the longest prefix $\mathtt{pre}$ of $B_2 \circ \cdots \circ B_{z-1}$ (i.e., excluding the first block B_1 and the last block B_z) such that $|\mathtt{pre}|$ is a multiple of w where w is the length of the period of $\mathtt{str}$[3]. Bob, after decoding Alice's vector U_ℓ, does the same thing, that is, he removes in his input string y and the maintained string $\tilde{x}$ those characters of the same indices that Alice removes.

Phase II: Alice maps her string x into the Hamming space using the CGK embedding (see Section II), getting a string x', and then partitions x' into blocks of size b'_ℓ, except that the first block is of a uniformly random size $\Delta'_\ell \in [b'_\ell/2, b'_\ell]$, and the last block is of size in the range $[1..b'_\ell]$. She then maps these blocks back to the edit space, getting a partition of the original string x. Denote these blocks by $A_1, \ldots, A_d$. Alice sends Δ'_ℓ to Bob, and Bob does the same partitioning to his string y, getting $A'_1, \ldots, A'_d$.

Next, Alice creates a vector $V_\ell = (g_1, \ldots, g_d)$ where $g_j = (h(A_j), r_j, \mathcal{E}_j)$ contains the following information on the j-th block A_j: $h : \{0,1\}^* \to \left[(Kn_i)^{\Theta(1)}\right]$ is a Karp-Rabin hash signature of A_j; r_j is the length of A_j; and $\mathcal{E}_j = 1$ if the first character of A_j is shared with the last character of A_{j-1} (this could happen due to the copy operations in the CGK embedding; $\mathcal{E}_j$ is needed for Bob to identify the boundaries of the unmatched substrings in x accurately), and $\mathcal{E}_j = 0$ otherwise. Alice then sends to Bob a redundancy of V_ℓ (denoted as $sk(V_\ell)$) that allows to recover up to K errors using the scheme in Lemma 4 (setting $p = 1/\mathsf{poly}(K \log n)$). Bob does the same for $(A'_1, \ldots, A'_d)$, getting a vector $V'_\ell = (g'_1, \ldots, g'_d)$. We will show in the analysis that Bob can recover Alice's vector V_ℓ using $sk(V_\ell)$ and his vector V'_ℓ with high probability. Bob then updates the string $\tilde{x}$ by replacing the '$\perp$'s with actual contents for those matched blocks, and at the next level he will do the decoding recursively on those unmatched blocks (i.e., those still marked with '$\perp$') whose sizes sum up to no more than $n_{\ell+1} = Kb'_\ell$.

The first stage concludes when the length of blocks in the second phase becomes $b'_\ell \le (K2^{\sqrt{\log n}})^{10(c_1+c_2)}$, from where Alice and Bob apply IMS directly to compute the edit distance and all the edits. The message Alice sends to Bob in the first stage includes the offsets $\{\Delta_\ell, \Delta'_\ell \mid \ell \in [L]\}$ and the redundancies $\{sk(U_\ell), sk(V_\ell) \mid \ell \in [L]\}$. Note that Alice can compute these independently using parameters b_ℓ, b'_ℓ and θ_ℓ at each level $\ell \in [L]$. In the second stage, Alice sends Bob the IMS sketch on her string x, but omits all the top-levels in the IMS sketch at which the block sizes are larger than b'_L, since Bob does *not* need to do the recovery at those levels given the first stage. At the end, Bob can recover Alice's input string s by adding back the removed periods at all levels.

B. The Analysis

Correctness. We focus on the case $k = ed(s,t) \le K$; otherwise if $k > K$ then Bob can detect it during the decoding (in particular, various recoveries using Lemma 4 will report "error") and output "error" with probability $1 - 1/\mathsf{poly}(K \log n)$.

The following lemma (whose proof is deffered to the full version) shows that the period-removal step in Phase I at each level preserves the edit distance.

Lemma 7. *Given two strings $s = ppp$ and t of the same length, letting $\pi = |p| \le ed(s,t)$ be the length of the period of s, the edit distance between $s' = pp$ and $t' = t[1..\pi] \circ t[2\pi + 1..3\pi]$ is at most $ed(s,t)$.*

The following two lemmas show that the redundancies sent by Alice in the two phases are sufficient for Bob to recover Alice's vectors U_ℓ and V_ℓ.

Lemma 8. *At each level ℓ in Phase I, let x and y be the strings held by Alice and Bob respectively.*

[2]We say that substring $s[i..j]$ is part of periodic substring of s with period $w_j > 0$ iff $s[i..j] = s[i - w_j..j - w_j]$ and w_j is the smallest number with this property.

[3]Formally two consecutive blocks B_j and B_{j+1} are contained in the same periodic substring iff $w_j = w_{j+1}$.

Suppose that $ed(x,y) \le K$, then with probability $1 - 1/\mathsf{poly}(K \log n)$, Bob can recover Alice's vector U_ℓ using his vector U'_ℓ and Alice's message $sk(U_\ell)$. Moreover, after the period-removal step the edit distance between the two resulting strings x' and y' does not increase, that is, $ed(x',y') \le ed(x,y)$.

Proof: For the first part of the lemma, we just need to show that U_ℓ and U'_ℓ differ in at most $2K$ pairs (e_j, e'_j). Note that we only need to look at those e'_j built from Q_j in y, since otherwise if e'_j is built from P_j in $\tilde{x}$ then we always have $e_j = e'_j$. We thus only need to consider at most $n_\ell/\theta_\ell = 2n_\ell/b_\ell$ pairs (e_j, e'_j).

We call a pair of block (B_j, Q_j) *good* if there is no edit in both B_j and Q_j as well as their preceding θ_ℓ characters in x and y respectively; we call the pair *bad* otherwise. Since $ed(x,y) \le K$, there are at most $2K$ bad pairs. We call a pair j *periodic* if $\chi_j = 1$ or $\chi'_j = 1$ (i.e., at least one of B_j or Q_j is part of a periodic subtring in s or t with period length at most θ_ℓ), and *non-periodic* otherwise. Clearly for a good and non-periodic pair j we have $(\chi_j, w_j) = (\chi'_j, w'_j) = (0,0)$.

We now show that for a good pair j, if $\chi_j = 1$ or $\chi'_j = 1$, then with probability $1 - O(K/b_\ell)$ we have $e_j = e'_j$. We prove only for the case when $\chi_j = 1$; the proof for the other case is similar. The observation is that we have a random shift $\Delta_\ell \in [b_\ell/2, b_\ell]$ in the block partition, and in any optimal alignment the indices of two matching characters in x and y can differ by at most $ed(x,y) \le K$, thus for a good block B_j that is part of a periodic substring of x of period length at most $\theta_\ell = b_\ell/2$, with probability $1 - O(K/b_\ell)$, Q_j is also part of a substring with the same period. By a union bound on at most $2n_\ell/b_\ell$ pairs (e_j, e'_j), we have with probability $1-(2n_\ell/b_\ell)\cdot(K/b_\ell) \ge 1-1/\mathsf{poly}(K \log n)$, that for all good and periodic pairs (e_j, e'_j), $e_j = e'_j$. The first part of the lemma follows.

The second part of the lemma follows directly from Lemma 7. Note that when removing periods in each periodic substring `str` we have kept the first and the last blocks that are contained in `str`, and thus Bob can recover those periods that have been removed. ∎

Lemma 9. *At each level ℓ in Phase II, let x and y be the strings held by Alice and Bob respectively. Suppose that $ed(x,y) \le K$, then with probability $1 - 1/\mathsf{poly}(K \log n)$, Bob can recover Alice's vector V_ℓ using his vector V'_ℓ and Alice's message $sk(V_\ell)$.*

Proof: We again just need to show that V_ℓ and V'_ℓ differ on at most K pairs (g_j, g'_j).

Let x' and y' be two strings obtained by performing the CGK embedding on x and y respectively. Let $\mathcal{W}$ be the random walk corresponding to the embedding. Recall that starting from a state (p_0, q_0) where a progress step happens, by Lemma 1 we have that with probability $1 - O(1/\sqrt{\gamma})$, after at most γ progress steps $\mathcal{W}$ will reach a state (p_1, q_1) such that $ed(x[p_1..n], y[q_1..n]) \le ed(x[p_0..n], y[q_0..n]) - 1$. We call such a sequence of walk steps a *progress phase*. Since $ed(x,y) \le K$, the total number of progress phases is upper bounded by K. By a union bound, with probability $1-O(K/\sqrt{\gamma})$, each of the at most K progress phases "consumes" at most γ progress steps. Note that for all indices j between two progress phases, we have $x'[j] = y'[j]$, that is, the two substrings of x' and y' are perfectly matched.

The key observation is that after the period-removal process in Phase I, by Lemma 3 we will have a "break" after passing at most two blocks (less than $4b_\ell$ characters when counting the periods crossing the two block boundaries) allowing for at least one progress step to execute. Therefore the total number of pairs of coordinates in x' and y' that are involved in one of the at most K progress phases can be bounded by $K \cdot \gamma \cdot 4b_\ell$ with probability $1 - O(K/\sqrt{\gamma})$. Setting $\gamma = (K \log n)^{c_2/2}$. By our choices of parameters,

$$b'_\ell \ge 2^{c_2(\log K + \sqrt{\log n})} b_\ell \ge (K \cdot \gamma \cdot 4b_\ell) \cdot (K \log n)^{\Theta(1)}.$$

Also recall that there is a random shift $\Delta'_\ell \in [b'_\ell/2, b'_\ell]$ at the beginning of the block partition in Phase II. We thus have with probability at least $1 - \big(O(K/(K \log n)^{c_2/4}) + (K \cdot \gamma \cdot 4b_\ell)/(b'_\ell/2)\big) \ge 1 - 1/\mathsf{poly}(K \log n)$ that at most K pairs (A_j, A'_j) differ, where $(A_1, \ldots, A_d)$ is the block partition of x in Phase II, and $(A'_1, \ldots, A'_d)$ is that of y. The lemma follows. ∎

The correctness of the algorithm follows from Lemma 8 and Lemma 9. Note that the first stage will finish in at most $O(\log \log n)$ levels since $\log n_\ell$ decreases at each level by a factor of $\log n_\ell / \log n_{\ell+1} = \log n_\ell / \log(Kb'_\ell) \ge \log n_\ell \Big/ \Big(\log\Big(\sqrt{n_\ell} 2^{(c_1+c_2)(\log K + \sqrt{\log n})}\Big)\Big) \ge 1.5$, where in the last inequality we used the fact that $n_\ell \ge (K 2^{\sqrt{\log n}})^{10(c_1+c_2)}$ (otherwise we go to the second stage and apply the IMS algorithm). The overall success probability is at least $1 - (1/\mathsf{poly}(K \log n) + 1/\mathsf{poly}(K \log n)) \cdot O(\log \log n) - 1/\mathsf{poly}(K \log n) \ge 1 - 1/\mathsf{poly}(K \log n)$, where the last term on the left hand side counts the error probability of the IMS algorithm (Theorem 2).

Complexities. We now analyze the communication cost of our algorithm; the running time of the algorithm is pretty traightforward, and we leave it to the full version of this paper.

In the first stage, at each level ℓ, the communication cost is dominated by the size of $sk(U_\ell)$ and $sk(V_\ell)$, both of which can be bounded by $O(K \log n_\ell)$ by Lemma 4 (where $\lambda \leq n_\ell$). Since $\log n_\ell$ decreases by a constant factor at each level, the total size of the message in the first stage is bounded by $O(K \log n)$. In the second stage, the total number of levels in the IMS sketch is bounded by $\log b'_L = O(\log K + \sqrt{\log n})$, and the sketch size per level is $O(K(\log K + \log\log n))$ (these are the same as Theorem 2 except that the number of levels has been reduced from $\log n$ to $\log b'_L$). Thus the total size of the IMS sketch is bounded by $O(\log K + \sqrt{\log n}) \cdot O(K(\log K + \log\log n)) = O(K(\log^2 K + \log n))$. Summing up, the total communication is bounded by $O(K(\log^2 K + \log n))$.

IV. Sketching

In this section we show the following theorem.

Theorem 10. *There exists a sketching algorithm for computing edit distance and all the edit operations having sketch size $O(K^8 \log^5 n)$, encoding time $\tilde{O}(K^2 n)$, decoding time $\mathsf{poly}(K \log n)$, and success probability 0.9, where n is the input size and K is the distance upper bound. When the distance is above the upper bound K, the decoding algorithm outputs "error" with probability $1 - 1/\mathsf{poly}(n)$.*

We first introduce a concept called *effective alignment*, and then show that a set of effective alignments satisfying a certain property can be used to construct an optimal alignment between the two strings.

Definition 2 (Effective Alignment). *Given two strings $s, t \in \{0,1\}^n$, we define an effective alignment between s and t as a triplet (G, g_s, g_t), where*

- *$G = (V_s, V_t, E)$ is a bipartite graph where nodes $V_s = \{1, \ldots, n\}$ and $V_t = \{1, \ldots, n\}$ correspond to indices of characters in s and t respectively, and if $(i,j) \in E$ then $s[i] = t[j]$. Moreover, edges in E are non-crossing, that is, for every pair of edges (i,j) and (i',j'), we have $i < i'$ iff $j < j'$.*
- *g_s is a partial function defined on the set of singletons (unmatched nodes) $U_s \subseteq V_s$; for each $i \in U_s$, define $g_s(i) = s[i]$. Similarly, g_t is a partial function defined on the set of singletons $U_t \subseteq V_t$; for each $j \in U_t$, define $g_t(i) = t[i]$.*

Intuitively, an effective alignment can be seen as a *summary* of an alignment after removing the information of those matched nodes. The following lemma gives the main idea of our sketching algorithm.

Lemma 11. *We can compute an optimal alignment for s and t using a set of effective alignments under the promise that there exists an optimal alignment going through all edges that are common to all effective alignments.*

Proof: Let $\mathcal{I}$ be the set of edges that are common to all effective alignments. We will show that we can reconstruct using these effective alignments all characters $s[i]$'s and $t[j]$'s for which i, j are not adjacent to any edges in $\mathcal{I}$.

For convenience we add two dummy edges $(0,0), (n+1, n+1)$ to all the effective alignments to form the boundaries. We can view the edges in $\mathcal{I}$ as a group of clusters $\mathcal{C}_1, \ldots, \mathcal{C}_\kappa$ (counting from left to right) each of which consists of a set of consecutive matching edges $\{(i,j), (i+1, j+1), \ldots\}$, plus some singleton nodes between these clusters. Now consider a particular effective alignment $\mathcal{A}$ and an $\ell \in [\kappa - 1]$. Let $(a_1, b_1), \ldots, (a_z, b_z)$ be the set of edges in $\mathcal{A}$ that lie between $\mathcal{C}_\ell$ and $\mathcal{C}_{\ell+1}$. By the definition of the effective alignment we can learn directly from $\mathcal{A}$ all the singletons in $\mathcal{A}$ that lie between $\mathcal{C}_\ell$ and $\mathcal{C}_{\ell+1}$. It remains to show that we can recover the characters in s and t that correspond to the nodes $a_1, \ldots, a_z$ and $b_1, \ldots, b_z$. For convenience we will identify nodes and their corresponding characters in the strings.

Let us consider edges $(a_1, b_1), \ldots, (a_z, b_z)$ one by one from left to right. Note that for each $x \in [z]$, we just need to recover one of $s[a_x]$ and $t[b_x]$ because they are equal. Since (a_x, b_x) is not in $\mathcal{I}$, we know that there exists another effective alignment $\mathcal{A}'$ which does not contain (a_x, b_x). We have the following cases:

1) a_x is a singleton in $\mathcal{A}'$. In this case we can recover $s[a_x]$ directly from $\mathcal{A}'$.
2) a_x is connected to a node u in $\mathcal{A}'$ such that $u < b_x$. This case is again easy since $t[u]$ has already been recovered, and thus we just need to set $s[a_x] = t[u]$.
3) a_x is connected to a node u in $\mathcal{A}'$ such that $u > b_x$. In this case b_x is either a singleton or is connected to a node $v < a_x$ in $\mathcal{A}'$. In the former case we can recover $t[b_x]$ directly from $\mathcal{A}'$, and in the latter case since we have already recovered $s[v]$, we can just set $t[b_x] = s[v]$.

We thus have shown that we can recover all nodes that are not adjacent to any edges in $\mathcal{I}$. Since by the promise that there exists an optimal alignment containing the edges that are common to all effective alignments (i.e., $\mathcal{I}$), we can construct such an optimal alignment by aligning characters in s and t in the gaps between clusters $\mathcal{C}_1, \ldots, \mathcal{C}_\kappa \subseteq \mathcal{I}$ in the optimal way. ∎

The rest of our task is to design sketches $sk(s)$ and $sk(t)$ for s and t respectively so that using $sk(s)$ and $sk(t)$ we can extract a set of effective alignments satisfying the promise in Lemma 11. Intuitively, the size of $sk(s)$ and $sk(t)$ can be small if (1) the information contained in each effective alignment is small, and (2) the number of effective alignments needed is small. Our plan is to construct $\rho = \mathsf{poly}(K \log n)$ effective alignments $\mathcal{A}_1, \ldots, \mathcal{A}_\rho$, each of which only contains $\mathsf{poly}(K \log n)$ singletons, such that there is an optimal alignment going through all edges in $\mathcal{I} = \bigcap_{j \in [\rho]} \mathcal{A}_j$. Note that we can compress the information of consecutive edges in each $\mathcal{A}_j$ by just writing down the first and the last edges, whose total number is at most twice the number of singletons. We thus can bound the sketch size by $\mathsf{poly}(K \log n)$. In the rest of this section we show how to carry out this plan.

We again make use of the random walk in the CGK embedding. Recall that a random walk on the two strings s and t can be represented as $\mathcal{W} = ((p_1, q_1), \ldots, (p_m, q_m))$ where (p_j, q_j) $(p_j, q_j \in [n])$ are states of $\mathcal{W}$. $\mathcal{W}$ naturally corresponds an alignment (not necessarily optimal) between s and t. More precisely, $\mathcal{W}$ corresponds to the alignment $\mathcal{A}$ constructed greedily by adding states $(p_m, q_m), \ldots, (p_1, q_1)$ as edges one by one whenever $s[p_j] = t[q_j]$.

Set $\rho = c_\rho K^2 \log n$ for a large enough constant c_ρ, and $N = c_\rho^4 K^6 \log^2 n$. We say that a random walk *passes* a pair (u, v) if there exists some $j \in [m]$ such that $(p_j, q_j) = (u, v)$. We call a random walk $\mathcal{W}$ *good* if the total number of progress steps in $\mathcal{W}$ is at most N (recall that we have a progress step at state (p_j, q_j) only if $s[p_j] \neq t[q_j]$). We can show the following (proof in the full version).

Claim 12. *The probability that all the ρ random walks are good is at least* 0.99.

Definition 3 (Anchor). *Given ρ random walks generated according to the CGK embedding, we say that a pair (u, v) $(u, v \in [n])$ is an* anchor *if $s[u] = t[v]$, and all the ρ random walks pass (u, v).*

The following lemma indicates that the alignments corresponding to a set of ρ random walks can be used (after compression) as a set of effective alignments satisfying the promise in Lemma 11. The proof of the lemma is technical and we leave it to the full version. We have included the high level idea of the proof below.

Lemma 13. *With probability $1 - 1/n^2$, there is an optimal alignment going through all anchors.*

Proof idea: We say an alignment $\mathcal{O}$ passes (or goes through) a pair (u, v) if (u, v) is an edge in $\mathcal{O}$. We choose a particular optimal alignment $\mathcal{O}$ we call the *greedy* alignment, and show that $\mathcal{O}$ passes all anchors with high probability. The high level idea is that suppose on the contrary that $\mathcal{O}$ does not pass an anchor (u, v), then we can find a matching $\mathcal{M}$ in the left neighborhood of (u, v) which may "mislead" a random walk; that is, with a non-trivial probability the random walk will "follow" $\mathcal{M}$ and consequently miss (u, v). We then have that with high probability at least one of the ρ walks will miss (u, v), which means that (u, v) is not an anchor, a contradiction.

We now give some ideas on how to construct $\mathcal{M}$ and then show that it will mislead a random walk. Define the left neighborhood of (u, v) in which we are interested to be $(s[u - z..u], t[v - z..v])$ such that $s[u] = t[v], \ldots, s[u - z] = t[v - z]$ but $s[u - z - 1] \neq t[v - z - 1]$. By exploring the properties of the greedy alignment $\mathcal{O}$, we can find a matching $\mathcal{M} \subseteq \mathcal{O}$ in the left neighborhood of (u, v) consisting of a set of clusters (consecutive edges), each of which is periodic[4] with a small period. Moreover, there are a small number of singleton nodes between those clusters. Our key observation is that once a random walk $\mathcal{W}$ enters a cluster, its shift (i.e., $|p - q|$ for a walk state (p, q)) will be changed by at most the length of the period of that cluster. We thus can show that the maximum shift change of $\mathcal{W}$ after entering the left neighborhood of (u, v) can be bounded by roughly k^2 $(k = ed(s, t))$. Now if $\mathcal{O}$ does not pass (u, v), then we can show that with a constant probability the first state (p, q) of $\mathcal{W}$ after entering the neighborhood does not align with (u, v) (i.e. $|p - q| \neq |u - v|$). We next show that with probability at least (roughly) $1/(100k^2)$, during the whole walk in the neighborhood, the shift of $\mathcal{W}$ will not be equal to $|u - v|$, and consequently $\mathcal{W}$ will miss (u, v).

The Sketch. We now show how to design sketches for s and t from which we can extract the ρ effective alignments corresponding to the ρ random walks. For recovering each of the ρ effective alignments, we prepare a pair of structures we call the *hierarchical* structure and the *content* structure, as follows. Let $B = 4 \log n$ be a parameter denoting a basic block size.

The hierarchical structure P. Let $s' \in \{0, 1\}^{3n}$ be the image of $s \in \{0, 1\}^n$ after the CGK embedding. W.l.o.g. assume $3n/B$ is a power of 2 (otherwise we can pad 0s to the image s'). We build a binary tree of depth $L = \log(3n/B)$ on top of s', whose leaves

[4]That is, the two substrings of the cluster in s and t are both periodic, and of the same period length.

(at level 1) correspond to a partition of s' into blocks of size B, and internal nodes at level ℓ correspond to substrings of s' of size $2^\ell B$ (i.e., the concatenation of the blocks corresponding to all the leaves in its subtree). For each level $\ell \in [L]$, setting $d_\ell = 3n/(2^\ell B)$, we create a vector $V_\ell = ((h_1, \eta_1), \ldots, (h_{d_\ell}, \eta_{d_\ell}))$ where h_j is a hash signature of the pre-image (in s) of the substring in s' corresponding to the j-th node at level ℓ, and η_j is the length of the pre-image. We then build a sketch P_ℓ of V_ℓ that allows to recover up to N errors using the scheme in Lemma 5. Let $P = (P_1, \ldots, P_L)$. The size of P is $O(L \cdot N \log n) = O(N \log^2 n)$ bits by Lemma 5.

The content structure Q. Again let s and s' be repsectively the original string and the string after the embedding. We partition s' into blocks of size B, and create a vector $U = (x_1, \ldots, x_{3n/B})$ where x_j is the pre-image (in s) of the j-th block of s'. We then build a sketch Q of U that allows to recover up to N errors using the scheme in Lemma 5. The size of Q is $O(N(B + \log n)) = O(N \log n)$ bits by Lemma 5.

The final sketch $sk(s)$ for s consists of ρ independent copies of (P, Q). Clearly the size of $sk(s)$ is bounded by $\rho \cdot O(N \log^2 n) = O(K^8 \log^5 n)$. Similarly, the sketch $sk(t)$ for t consists of ρ independent copies (P', Q'), each of which is constructed in the same way as (P, Q) (but for string t). The time for computing the sketch is bounded by $\tilde{O}(\rho \cdot n) = \tilde{O}(K^2 n)$.

Now we show that we can extract an effective alignment for s and t from (P, Q) and (P', Q'). We again focus on the case when $k = ed(s, t) \leq K$. Otherwise if $k > K$ then various recoveries using Lemma 5 in the decoding will report error with probability $1 - 1/\mathsf{poly}(n)$. The proof of the following lemma can be found in the full version.

Lemma 14. *Given (P, Q) and (P', Q'), with probability at least 0.98, we can extract an effective alignment $\mathcal{A}$ for s and t corresponding to a random walk $\mathcal{W}$ according to the CGK embedding such that for any edge $(u, v) \in \mathcal{A}$ there exists some state $(p, q) \in \mathcal{W}$ such that $(p, q) = (u, v)$.*

Lemma 13 and Lemma 14 give the following.

Corollary 15. *With probability 0.97, we can extract ρ effective alignments from $sk(s)$ and $sk(t)$ such that there is an optimal alignment between s and t going through all edges that are common to all of the ρ alignments.*

The next lemma (proof in the full version) finishes the proof of Theorem 10. We comment that the fact that the decoding time can be reduced to $\mathsf{poly}(K \log n)$ is very useful in the distributed/parallel computation models where the decoding phase is performed in a central server which collects sketches produced from a number of machines.

Lemma 16. *With probability 0.97, we can extract ρ effective alignments from $sk(s)$ and $sk(t)$ and use them to construct an optimal alignment between s and t. The running time of the construction is $\mathsf{poly}(K \log n)$.*

ACKNOWLEDGEMENTS

The authors would like to thank Hossein Jowhari for helpful discussions at the early stage of this work. The second author would like to thank Funda Ergun, Cenk Sahinalp, Dirk Van Gucht and Erfan Sadeqi Azer for helpful discussions and comments.

REFERENCES

[1] D. Belazzougui and Q. Zhang, "Edit distance: Sketching, streaming and document exchange," *arXiv preprint arXiv:1607.04200*, 2016.

[2] H. Jowhari, "Efficient communication protocols for deciding edit distance," in *20th ESA*, 2012, pp. 648–658.

[3] A. Andoni, A. Goldberger, A. McGregor, and E. Porat, "Homomorphic fingerprints under misalignments: sketching edit and shift distances," in *45th ACM STOC*, 2013, pp. 931–940.

[4] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *48th ACM STOC*, 2016, pp. 712–725.

[5] A. Orlitsky, "Interactive communication: Balanced distributions, correlated files, and average-case complexity," in *32nd FOCS*, 1991, pp. 228–238.

[6] U. Irmak, S. Mihaylov, and T. Suel, "Improved single-round protocols for remote file synchronization," in *24th IEEE INFOCOM*, 2005, pp. 1665–1676.

[7] D. Belazzougui, "Efficient deterministic single round document exchange for edit distance," *arXiv preprint arXiv:1511.09229*, 2015.

[8] G. Cormode, M. Paterson, S. C. Sahinalp, and U. Vishkin, "Communication complexity of document exchange," in *11th ACM-SIAM SODA*, 2000, pp. 197–206.

[9] G. Cormode and S. Muthukrishnan, "The string edit distance matching problem with moves," *ACM Trans. Algorithms*, vol. 3, no. 1, 2007.

[10] B. Saha, "The dyck language edit distance problem in near-linear time," in *55th FOCS*, 2014, pp. 611–620.

[11] E. Porat and O. Lipsky, "Improved sketching of hamming distance with error correcting," in *18th CPM*. Springer, 2007, pp. 173–182.