Streaming algorithms for embedding and computing edit distance in the low distance regime*

Diptarka Chakraborty
Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur, India
diptarka@cse.iitk.ac.in

Elazar Goldenberg
Charles University in Prague
Computer Science Institute of Charles University
Prague, Czech Republic
elazargold@gmail.com.

Michal Koucký
Charles University in Prague
Computer Science Institute of Charles University
Prague, Czech Republic
koucky@iuuk.mff.cuni.cz

Abstract

The Hamming and the edit metrics are two common notions of measuring distances between pairs of strings x, y lying in the Boolean hypercube. The edit distance between x and y is defined as the minimum number of character insertion, deletion, and bit flips needed for converting x into y. Whereas, the Hamming distance between x and y is the number of bit flips needed for converting x to y.

In this paper we study a randomized injective embedding of the edit distance into the Hamming distance with a small distortion. We show a randomized embedding with quadratic distortion. Namely, for any x, y satisfying that their edit distance equals k, the Hamming distance between the embedding of x and y is $O(k^2)$ with high probability. This improves over the distortion ratio of $O(\log n \log^* n)$ obtained by Jowhari (2012) for small values of k. Moreover, the embedding output size is linear in the input size and the embedding can be computed using a single pass over the input.

We provide several applications for this embedding. Among our results we provide a one-pass (streaming) algorithm for edit distance running in space $O(s)$ and computing edit distance exactly up-to distance $s^{1/6}$. This algorithm is based on *kernelization* for edit distance that is of independent interest.

*The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 616787. The first author was partially supported by Research-I Foundation, Microsoft Research India Travel Grant and Google India Student Travel Grant. The third author was partly supported by the project 14-10003S of GA ČR.

1 Introduction

The *edit distance* (aka *Levenshtein distance*) [Lev66] is a common distance measure between pairs of strings $x, y \in \{0,1\}^*$. It plays a central role in several domains such as computational biology, speech recognition, text processing and information retrieval. The edit distance between x and y, denoted by $\Delta_e(x,y)$, is defined as the minimum number of character insertion, deletion, and bit flips needed for converting x into y. Another common distance measure between pairs of strings $x, y \in \{0,1\}^n$ is the *Hamming distance*, denoted by $\Delta_H(x,y)$, which is the number of bit flips needed for converting x to y.

From computational perspective, the Hamming distance has many advantages over the edit distance, our current understanding indicates that it is easier to compute and work with (cf. [BI15]). A natural way of bridging the gap between applications requiring the edit distance and algorithms working with the Hamming distance is designing an *embedding* of the edit distance into the Hamming distance. By an embedding of the edit distance into the Hamming distance we mean a mapping f of strings into (possibly different) strings that transforms the edit distance of strings x and y into a related Hamming distance of $f(x)$ and $f(y)$. The *distortion* of such an embedding is the worst case ratio between the Hamming distance of $f(x)$ and $f(y)$ and the edit distance of x and y (assuming the embedding does not shrink distances.) Embeddings with low distortion found many applications: from computing edit distance to sketching it, e.g., [BYJKK04, OR07, AO09].

For many algorithmic applications a weaker concept—*randomized embedding*—suffices. A randomized embedding is a randomized mapping from strings to strings so that for any x and y, with high probability over the randomness r, the ratio between the Hamming distance of $f(x,r)$ and $f(y,r)$ and the edit distance of x and y is small (again assuming that f does not shrink distances.) This is the type of embedding we consider in this paper. Our main result provides a linear-time embedding with quadratic distortion between input edit distance and output Hamming distance. The distortion does not depend on the length of the strings. A notable feature of our embedding is that it processes the input in one-way manner using little or no space, it is a streaming algorithm.

We provide several applications of this embedding. We design a streaming algorithm for computing the *exact* edit distance between a pair of strings provided that their edit distance is small ($o(n^{1/6})$). An additional key ingredient of our algorithm is the *kernelization* of the input strings for edit distance. This is of independent interest.

Two additional applications are in the context of communication complexity. A well-studied one is *Document Exchange* problem [CPSV00], where two communicating parties Alice and Bob hold two input strings x and y and based on the message transmitted by Alice, Bob's task is to decide whether $\Delta_e(x,y) > k$ and if $\Delta_e(x,y) \leq k$ then to report x correctly. Another important problem related to embedding is to decide edit distance using a *sketching* protocol. In this problem, given two strings x and y, we would like to compute *sketches* $s(x)$ and $s(y)$ or in other words a mapping $s : \{0,1\}^n \to \{0,1\}^t$ such that t is much smaller compared to n and our objective is to decide whether $\Delta_e(x,y) > k$ or not having access to only $s(x)$ and $s(y)$. One can also view the same problem as a two-party public-coin *simultaneous* communication protocol [KN97], where Alice holds x and Bob holds y and both of them are only allowed to send one message to a third referee whose job is to decide whether $\Delta_e(x,y) > k$ or not depending on the messages he receives from Alice and Bob. We elaborate on our results next.

1.1 Our Main Results

Our main result shows the existence of a randomized mapping f from edit distance into Hamming distance that at most squares the edit distance. Our result applies to the shared randomness model. Namely, we show that for every pair of strings $(x, y) \in \{0,1\}^n$ having a small edit distance the Hamming distance between the encoded strings is small, provided that the encoding was done using the same sequence of random bits.[1] Formally:

Theorem 1.1. *For any integer $n > 0$, there is $\ell = O(\log n)$ and a function $f : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^{3n}$ such that for any pair of strings $x, y \in \{0,1\}^n$:*

$$\frac{1}{2} \cdot \Delta_e(x,y) \le \Delta_H(f(x,r), f(y,r)) \le O((\Delta_e(x,y))^2)$$

*with probability at least $2/3$ over the choice of the random string $r \in \{0,1\}^\ell$. Furthermore, f can be computed in a streaming fashion in (*non-uniform*) log-space using one pass over the input string x.*

We can also give a uniform version of this theorem by allowing a little bit more random bits.

Theorem 1.2. *There is an algorithm A computing a function $f : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^{3n}$ where $\ell = O(\log^2 n)$ such that for any pair of strings $x, y \in \{0,1\}^n$:*

$$\frac{1}{2} \cdot \Delta_e(x,y) \le \Delta_H(f(x,r), f(y,r)) \le O((\Delta_e(x,y))^2)$$

with probability at least $2/3$ over the choice of the random string $r \in \{0,1\}^\ell$. The algorithm A on input x and r works in a streaming fashion in log-space using one pass over its input x and random access to r. On a word RAM with word size $O(\log n)$, A can be implemented so that it uses only constant amount of working memory, i.e., $O(1)$ words, and uses $O(1)$ amortized time per output bit ($O(\log n)$ per output bit in the worst-case.)

In both of the above theorems, with probability at least $1 - 1/n^{\Omega(1)}$ over the choice of random r, x can be recovered from $f(x,r)$. Indeed, the algorithm computing f knows which is the case by the end of its computation. Hence, the function f in both of the theorems can be made one-to-one by appending r at the end of the output and also appending either 0^n or x depending on whether x can be recovered or not. (One would require an extra pass over x to append it.) This answers positively Problem 59 of Dortmund Workshop on Algorithms for Data Streams 2012 [DW1] in the randomized case whether there is an embedding which maps strings of constant edit distance into strings of Hamming distance $o(\log n)$.

We can specify the trade-off between the Hamming distance and its probability: for any positive $c \in \mathbb{R}$, the probability that $\Delta_H(f(x,r), f(y,r)) \le c \cdot \Delta_e(x,y)^2$ is at least $1 - \frac{12}{\sqrt{c}} - O(\frac{1}{n})$ (extra $O(1/n)$ term comes from the error incurred due to Nisan's PRG discussed in Section 5) On the other hand $\Delta_H(f(x,r), f(y,r)) \ge \Delta_e(x,y)/2$ happens with probability $1 - 1/n^{\Omega(1)}$.

One may naturally wonder what is the distribution of the resulting Hamming distance. It very much depends on the two strings x and y. For example if y is obtained from x by flipping the first k bits then with high probability the Hamming distance of $f(x,r)$ and $f(y,r)$ is $O(k)$. On the other hand, if y is obtained from a *random* x by flipping each n/k-th bit (where $n \ge k^3$) then with high

[1] Our result can easily be extended for strings lying in a larger alphabet (See Section 6).

probability the Hamming distance of $f(x, r)$ and $f(y, r)$ will be $\Omega(k^2)$. Interestingly though, for any two distinct fixed x and y, the expectation of $\Delta_H(f(x, r), f(y, r))$ is $\Theta(n)$ so the distribution is heavy tailed. For many applications this is not any problem as in many applications one can abort when the Hamming distance is large and retry with a new random seed.

Applications One of the important contributions of this paper is providing an algorithm efficient in terms of running time as well as space that computes the edit distance between two strings with a promise that the edit distance is "small". Moreover, our algorithm can be implemented in a streaming fashion using only a single pass over the input. Let us first state our result formally.

Theorem 1.3. *There is a probabilistic algorithm that on input $x, y \in \{0,1\}^n$ and an integer s with probability at least $1 - 1/n$ outputs $k = \Delta_e(x, y)$ if $k < s^{1/6}$, and a series of k edit operations transforming x into y. With the remaining probability the algorithm outputs 'I DO NOT KNOW'. The algorithm accesses x and y in one-way manner, runs in time $O(n + s \log n)$ and uses space $O(s)$.*

To the best of our knowledge this is the first algorithm capable of computing the *exact* edit distance of two strings in a streaming fashion provided the distance is not too large $o(n^{1/6})$. Furthermore, it can output the sequence of associated edit operations in sub-linear space. No such linear-time algorithm existed thus far. If we allow the algorithm $O(\log \log k)$ passes over the input, we do not have to provide the parameter s to the algorithm (Corollary 7.10).

Other applications of the randomized embedding considered in this paper include the document exchange problem, design of sketching algorithm to solve *gap edit distance problem*, and *approximate nearest neighbor search*. For the document exchange problem we first apply our embedding and then use the document exchange protocol under Hamming distance, for which an efficient communication protocol is known [PL07]. As a consequence, we achieve $O(k^2 \log n)$ bound on the number of bits transmitted for document exchange protocol under edit metric. Moreover, in our protocol, running time of each party will be $O(n \log n + k^2 \log n)$.

In a similar fashion, our main embedding result provides a randomized sketching algorithm for k vs. ck^2 gap edit distance problem for some constant c and a randomized algorithm for approximate nearest neighbor search problem that will return a point within the distance of $O(k)$ times that of the closest one under edit distance metric.

2 Our Techniques

Embedding. Our embedding scheme is based on markedly different idea than previously known embeddings of edit distance into Hamming distance. Our basic embedding scheme is based on the idea of coupling of Markov chains known as Kruskal Principle (cf. [LRV09]). The basic randomized scheme works as follows: First we pick (using the random string r) a sequence of random functions $h_1, \ldots, h_{3n} : \{0,1\} \rightarrow \{0,1\}$. We further maintain a pointer i for current position on the input string x, initially set to 1. In time $t \leq 3n$ we append the bit x_i to the output, and increment i by $h_t(x_i)$ (if i exceeds n, we pad the output string with zeros).

This scheme clearly uses a linear number of bits. Using derandomization techniques such as Nisan's pseudo-random generator [Nis90], we can reduce the number of bits to $O(\log^2 n)$ in the uniform case and $O(\log n)$ in the non-uniform case.

To shed some light on why the basic embedding works well, consider two strings $x, y \in \{0,1\}^n$ of edit distance k, and consider the output $f(x,r), f(y,r)$ of the embedding for the same random bits r, i.e., produced using the same hash functions. Observe that as long as the pointer i points to indices in the shared prefix of x and y, the two output strings are equal. In the first iteration when i points to an index such that $x_i \neq y_i$, the output bits become different. Nevertheless, the increment of i is done independently in the embedding of each input. This independent increment is done in each iteration where the output bits are different. The crux of our argument relies on this independent increment to ensure that with high probability $\Delta_H(f(x,r), f(y,r)) \leq ck^2$. This is done by reducing our problem to a certain question regarding random walks on the integer line.

Algorithm of Saha [Sah14]: An idea similar to ours is used in the algorithm of [Sah14] for computing the edit distance of a string of parenthesis of various types from the set of well parenthesized expressions. The main idea of the algorithm presented in [Sah14] is to process the string left to right, push opening parenthesis on a stack and match them against closing parenthesis. Whenever there is a mismatch remove at random either the closing parenthesis or the opening one. This algorithm can be applied also to approximately compute the edit distance of strings by pushing a reverse of one of the strings on the stack and matching the other string against the stack. Whenever there is a mismatch remove at random a symbol either from the top of the stack or from the other string.

In order to approximate the edit distance of two strings at hand it is fairly natural idea to remove the mismatched symbols at random. Our algorithm also builds on this idea. However, while designing an embedding protocol, there is a major technical challenge when we do not have access to both of the strings at the same time and we should remove the mismatched characters. We do not know which one are those. Deleting symbols at random is unlikely to provide a good result. Moreover, we would like the embedding of x to be *simultaneously* close to the embedding of all the strings y, which are relatively close (in edit distance) to x.

Hence, it is not clear that one can carry out the basic idea of mismatch removal in the case of string embedding when we have access only to one of the strings. Indeed, one needs some *oblivious synchronization mechanism* that would guarantee that once the removal process is synchronized it stays synchronized. The existence of such a mechanism is not obvious. Surprisingly, there is an elegant solution to this problem with a good performance.

Our solution to this problem repeats symbols random number of times instead of removing them and we introduce correlations between the random decisions so that our process achieves synchronization. (Indeed, deleting symbols would not work in our setting as we want to preserve the overall information contained in the strings.) Although the underlying mechanism of our algorithm and that of [Sah14] is different, the analysis in [Sah14] eventually reduces to a similar problem about random walks. One of an open problems about the oblivious synchronization is whether one can design such a mechanism with a better performance than ours.

Computing edit distance. Our algorithm for small edit distance consists of two steps. In the first step we produce a *kernel* x' and y' from the input strings x and y such that the edit distance of x' and y' is the same as that of x and y, while the length of x' and y' is related to their edit distance rather than the original input size. Then we run known algorithm [LMS98] on the kernel to compute the edit distance. (This is akin to Fixed Parameter Tractability.)

To compute the kernel, we first find a good candidate *alignment* of x and y using our embedding procedure or Saha's algorithm. By an alignment we mean a mapping of symbols from x to their copies in y resulting from performing edit operations on x. (See Section 7.1 for a precise definition.) The candidate alignment obtained from our algorithm results from $O(k^2)$ edit operations with high probability, where k is the edit distance of x and y. Using this alignment we reduce x and y to x' and y' of length $O(k^6)$ and the same edit distance. Finding the embedding and reducing the size can be done during a single pass over the inputs.

As the size of the kernel is directly related to the quality of the initial alignment the above procedure succeeds only with constant probability. To decrease the error probability we re-run the embedding and kernelization procedures if we obtain a kernel that is too large. To avoid re-reading the whole input we use the over-sized kernel from the previous trial as the input in the next trial.

Further Improvements: Since our earlier version of the paper we have realized that we could obtain a somewhat better deterministic algorithm that uses $O(k^4)$ space and runs in time $O(n+k^4)$. This could be done as follows. We divide the input strings into blocks of size $O(k^4)$. Then apply the deterministic algorithm to find the optimal alignment between the first block of both the input strings [LMS98]. It is guaranteed that the resulted alignment will be of cost at most k and thus it will be at most $2k$ steps away from the desired optimal alignment. Now do the same step for the next block. However this time, instead of considering just the current block for each input, we consider the string resulted by appending kernel computed so far with the current block. Thus the strings under consideration will be of size $O(k^4)$. Again apply the algorithm given in [LMS98] and continue this process. At the end we will output the exact edit distance between the two input strings.

Using a different approach the authors claim to find a deterministic algorithm that uses $O(k^2)$ space and runs in time $O(n+k^2)$. This approach is still in progress and it would be published soon.

2.1 Related Works

2.1.1 Computing and approximating edit distance

The notion of edit distance plays a central role in several domains such as computational biology, speech recognition, text processing and information retrieval. As a consequence, computational problems involving edit distance seek attentions of many researchers. We refer the reader to a survey by Navarro [Nav01] for a comprehensive treatment of this topic. The problem of computing exact edit distance (the *decision* problem) can be solved in $O(n^2)$ time using classical dynamic programming based algorithm [WF74]. Later, Masek and Paterson [MP80] achieved slightly improved bound of $O(n^2/\log n)$ and this is the best known upper bound so far for computing arbitrary edit distance. Backurs and Indyk [BI15] indicates that this bound cannot be improved significantly unless the strong Exponential Time Hypothesis is false. They give a reduction which (implicitly) maps instances of SAT into instances of edit distance with the edit distance close to n. This does not exclude faster algorithms for small edit distance such as the $O(n+k^2)$ time algorithm provided by Landau et al. [LMS98].

So suppose the edit distance between the strings is guaranteed to be bounded by $k \ll n$. Then the running time and the space can be improved. Ukkonen [Ukk85] provided an algorithm that solves the decision problem in time $O(kn)$ and space $O(k)$. To find the optimal alignment (the *search* problem) the algorithm uses $O(n)$ space. Landau et al. [LMS98] solved the decision problem within time $O(n+k^2)$ and $O(n)$ space (by slightly modifying their algorithm the search problem can be solved as well using $O(k^2)$ extra space). In this paper, we provide a randomized algorithm

Table 1: Taxonomy of Algorithms Solving/ Approximating the Edit Distance

Authors	Time	Space	Approximation Ratio	Solving Decision or Search
[WF74]	$O(n^2)$	$O(n)$	Exact	Search
[MP80]	$O(n^2/\log(n))$	$O(n)$	Exact	Search
[LMS98]	$O(n+k^2)$	$O(n)$	Exact	Decision
[LMS98]	$O(n+k^2)$	$O(n+k^2)$	Exact	Search
[LMS98]	$O(n)$	$O(n)$	$\sqrt{n}$	Search
[AKO10]	$n^{1+\epsilon}$ (randomized)	$O(n)$	$(\log n)^{O(1/\epsilon)}$	Decision
[Sah14]	$O(n)$ (randomized)	$O(\log n)$	$O(k)$	Search (streaming)
This paper	$O(n+k^6)$ (randomized)	$O(k^6)$	Exact	Search with a promise (streaming and single pass)
This paper	$O(\epsilon^{-1} n \log\log k + k^{6+\epsilon}\log n)$ (randomized)	$O(k^{6+\epsilon})$	Exact	Search (streaming and $\log\log k$ pass)

for the search problem under the promise that the edit distance is small. Our algorithm runs in time $O(n+k^6)$, uses space of size $O(k^6)$ and it is a single-pass algorithm. We can remove the promise by paying a penalty in the number of passes over the input and slightly worse time and space complexity. Table 1 summarizes the above results.

On the other hand, if we focus on approximating edit distance, we have much better bounds on running time. The exact algorithm given in [LMS98] immediately gives a linear-time $\sqrt{n}$-approximation algorithm. A series of subsequent works improved this approximation factor first to $n^{3/7}$ [BYJKK04], then to $n^{1/3+o(1)}$ [BES06] and later to $2^{\widetilde{O}(\sqrt{\log n})}$ [AO09] while keeping he running time of the algorithm almost linear. Batu et al.[BEK+03] provided an $O(n^{1-\alpha})$-approximation algorithm that runs in time $O(n^{\max\{\frac{\alpha}{2},2\alpha-1\}})$. The approximation factor was further improved to $(\log n)^{O(1/\epsilon)}$, for every $\epsilon > 0$ where the approximation algorithm runs in $n^{1+\epsilon}$ time [AKO10].

The result related to computing edit distance approximately in [AO09] was based on embedding edit distance into Hamming distance [OR07], where authors showed such an embedding with distortion factor $2^{O(\sqrt{\log n \log\log n})}$. In case of embedding of *edit distance with moves* [2] into Hamming metric, the distortion factor is known to be upper bounded by $O(\log n \log^* n)$ [CM02]. Andoni *et al.* [ADG+03] showed a lower bound of 3/2 on distortion factor in case of embedding from edit distance metric to Hamming metric. This lower bound was later strengthened to $(\log n)^{1/2-o(1)}$ [KN05] and then to $\Omega(\log n)$ in [KR06].

2.1.2 Randomized embedding and its implications

Embeddings that we have talked about in the last subsection are all deterministic. If we consider randomized embedding then we already have a much better bound on distortion factor by [Jow12]. Jowhari [Jow12] gave a time-efficient randomized embedding from edit distance metric into Hamming metric with distortion factor $O(\log n \log^* n)$ equipped with a polynomial time *reconstruction procedure*. He raised the question whether it is possible to achieve distortion factor $o(\log n)$ keeping

[2]Similar to $\Delta_e(x,y)$ with addition of a block move operation, where moving a substring of x to another location is considered as a single operation.

both the embedding and the reconstruction procedure time-efficient. In this paper, we answer this question and achieve an upper bound of $O(k)$ on distortion factor which is a significant improvement over the previous result when k is "small" compared to n.

Jowhari's motivation behind giving such efficient randomized embedding was to design a one-way protocol that solves document exchange problem under edit distance metric. Before looking at the result by Jowhari [Jow12], let us first shed some light on what was known earlier about this problem. If we focus only on one-way communication protocols, then by using a simple strategy (e.g., see [Jow12]) involving random hash functions, we can achieve an upper bound of $O(k \log n)$ on the total number of bits transmitted and the same bound is also known for deterministic protocol [Orl91]. Unfortunately, both the protocols incur Bob's running time exponential in k. Jowhari [Jow12] studied this problem with the restriction that both Alice and Bob run $poly(k, n)$-time algorithms and gave a protocol with a $O(k \log^2 n \log^* n)$ bound on the number of bits transmitted. He actually showed that any efficient randomized embedding with a reconstruction procedure which is efficient as well, can be used to design a randomized protocol for this problem. As a consequence, our embedding result translates into a protocol with a $O(k^2 \log n)$ bound on the number of bits transmitted.

Another problem that we consider in this paper is the design of sketching algorithm to decide edit distance. For Hamming distance, efficient sketching algorithm is known [KOR98, BYJKK04] that solves the k vs. $(1+\epsilon)k$ *gap Hamming distance* problem with constant size sketches. Building on that result, Bar-Yossef *et al.* [BYJKK04] gave a computation of constant size sketch that can distinguish the two cases when edit distance is at most k and when that is at least $(kn)^{2/3}$, for $k \leq \sqrt{n}$. Later, improvement on distortion factor of embedding [OR07] results in solution to k vs. $2^{O(\sqrt{\log n \log \log n})} \cdot k$ gap edit distance problem. Our embedding result can be used to solve k vs. ck^2 gap edit distance problem with high probability for some constant c using constant size sketches.

Organization of the Paper: In Section 3, we restate some of the basic facts about random walks on the integer line. In Section 4, we provide the main randomized embedding algorithm and also show the bound on distortion factor of that embedding. Then in Section 5, we discuss how to reduce the number of random bits used in the embedding algorithm which implies Theorem 1.1 and Theorem 1.2. We dedicate Section 7.1 and Section 7.2 to the discussion of one of the main applications of our embedding algorithm which is computing edit distance between a pair of strings with a guarantee that the distance is small. In Section 8, we provide several other application of our main embedding algorithm in the domain of communication complexity.

3 Preliminaries

In the rest of the paper, we refer to a random walk on a line as a random walk on the integer line where in each step with probability 1/2 we stay at the same place, with probability 1/4 we step to the left, and otherwise we step to the right. For parameters $t \in \mathbb{N}, \ell \in \mathbb{Z}$, by $q(t, \ell)$, we denote the probability that a random walk on a line starting at the origin reaches the point ℓ at time t for the first time (for convenience we set $q(0, 0) = 1$) and by $p(t, \ell)$, we denote the probability that a random walk on a line starting at the origin reaches the point ℓ within time t. Note that $p(t, \ell) = \sum_{i=0}^{t} q(i, \ell)$. Now a few basic facts about the functions $p(t, \ell)$ and $q(t, \ell)$ are mentioned below.

Observation 3.1. *Let $t \in \mathbb{N}$ then:*

1. *For all* $\ell < 0$ *it holds that* $p(t,\ell) \le p(t,\ell+1)$*, and for all* $\ell > 0$*,* $p(t,\ell) \le p(t,\ell-1)$*,*
2. *For all* $\ell \neq 0$ *it holds that* $q(t,\ell) = \frac{1}{4}q(t-1,\ell-1) + \frac{1}{2}q(t-1,\ell) + \frac{1}{4}q(t-1,\ell+1)$*,*
3. *For all* $\ell > 1$ *it holds that* $q(t,\ell) = \sum_{j<t} q(t-j,\ell-1)q(j,1)$*.*

The following is a well known fact about random walks that can be found e.g. in [LPW06, Theorem 2.17].

Proposition 3.2 (Folklore)**.** *For any* $k, \ell \in \mathbb{N}$ *it holds that:*

$$\sum_{t=0}^{\ell} q(t,k) \ge 1 - \frac{12k}{\sqrt{\ell}}.$$

In particular, $\sum_{t=0}^{1296k^2} q(t,k) \ge \frac{2}{3}$*.*

4 The Basic Embedding

In this section we present our basic embedding and in the subsequent section we show how to reduce the number of its random bits to prove our main theorems. The pseudo-code for the embedding is given in Algorithm 1.

Algorithm 1 Basic Embedding Function f

```
Input  : $x \in \{0,1\}^n$, and a random string $r \in \{0,1\}^{6n}$
Output: $f(x,r) \in \{0,1\}^{3n}$
Interpret $r$ as a description of $h_1, \ldots, h_{3n} : \{0,1\} \to \{0,1\}$.
Initialization: $i = 1$, $Output = \lambda$;
for $j = 1, 2, \ldots, 3n$ do
    if $i \le n$ then
        $Output = Output \odot x_i$, where the operation $\odot$ denotes concatenation;
        $i = i + h_j(x_i)$;
    end
    else
        $Output = Output \odot 0$;
    end
end
Set $f(x,r) = Output$.
```

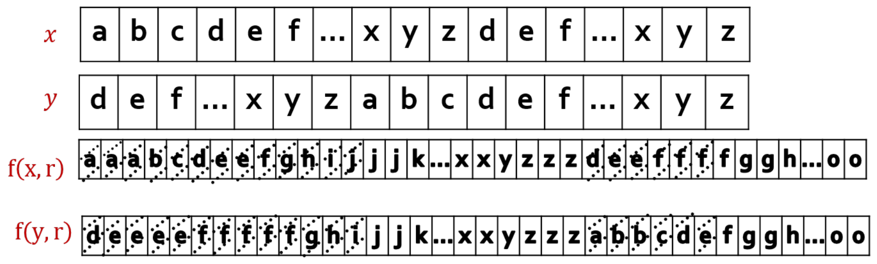
Let us illustrate our embedding applied on the strings: $x = abc, \ldots, xyz, def, \ldots, xyz$ and $y = def, \ldots, xyz, abc, \ldots, xyz$ while using the same sequence r as random string.

As one can see, upon each block of edit changes, the output strings $f(x,r)$ and $f(y,r)$ become different, till they "synchronize" again. In the sequel, we justify this kind of behavior and show that the synchronization is rapid.

We summarize here the main properties of our basic embedding:

Theorem 4.1. *The mapping* $f : \{0,1\}^n \times \{0,1\}^{6n} \to \{0,1\}^{3n}$ *computed by Algorithm 1 satisfies the following conditions:*

Figure 1: Runtime example of the embedding algorithm:


x
a b c d e f ... x y z d e f ... x y z
y
d e f ... x y z a b c d e f ... x y z
f(x, r)
a a a b c d e e f g h i j j j k ... x x y z z z d e e f f f f g g h ... o o
f(y, r)
d e e e e f f f f f g h i j j k ... x x y z z z a b b c d e f g g h ... o o

1. *For every $x \in \{0,1\}^n$, given $f(x,r)$ and r, it is possible to decode back x with probability $1 - \exp(-\Omega(n))$.*

2. *For every $x, y \in \{0,1\}^n$, $\Delta_e(x,y)/2 \le \Delta_H(f(x,r), f(y,r))$ with probability at least $1 - \exp(-\Omega(n))$.*

3. *For every positive constant c and every $x, y \in \{0,1\}^n$, $\Delta_H(f(x,r), f(y,r)) \le c \cdot (\Delta_e(x,y))^2$ with probability at least $1 - \frac{12}{\sqrt{c}}$.*

Moreover, both the mapping f and its decoding (given $f(x,r)$ and r) take linear time and can be performed in a streaming fashion.

Let us justify the properties of our basic embedding from Theorem 4.1. It is clear from the algorithm that $|f(x,r)| = 3n$. For the first property observe that we can recover x from $f(x,r)$ and r provided that $i = n+1$ at the end of the run of the algorithm. Since at each iteration of the algorithm, i is incremented with probability 1/2, the probability that during $3n$ rounds it does not reach $n+1$ can be bounded by $2^{-\Omega(n)}$ by the Chernoff bound. So unless this low probability event happens we can recover x from $f(x,r)$ and r.

Proving the second property is straightforward. Indeed, let $k = \Delta_e(x,y)$. We claim that $k/2 \le \Delta_H(f(x,r), f(y,r))$ whenever the algorithm ends with $i = n+1$ on both x and y. In such a case x can be obtained from $f(x,r)$ by removing all the bits where $h_j(f(x,r)_j) = 0$. Similarly for y. Hence, y differs from x only in the part which is obtained from the portion of $f(y,r)$ which bit-wise differs from $f(x,r)$. If $\ell = \Delta_H(f(x,r), f(y,r))$ then we need to apply at most ℓ edit operations on x to obtain all the y except for at most the last ℓ bits of y (in the case when they are all 0). So except for an event that happens with exponentially small probability $\Delta_e(x,y) \le 2 \cdot \Delta_H(f(x,r), f(y,r))$.

The rest of this section is devoted for the proof of Property 3. We will need the following main technical lemma. Together with Proposition 3.2 it implies the theorem.

Lemma 4.2. *Let $x, y \in \{0,1\}^n$ be of edit distance $\Delta_e(x,y) = k$. Let $q(t,k)$ be the probability that a random walk on the integer line starting from the origin visits the point k at time t for the first time. Then for any $\ell > 0$, $\Pr[\Delta_H(f(x,r), f(y,r)) \le \ell] \ge \sum_{t=0}^{\ell} q(t,k)$ where the probability is over the choice of r.*

Consider two strings $x, y \in \{0,1\}^n$ such that $\Delta_e(x,y) = k$. We will analyze the behavior of the embedding function on these two strings. We are interested in the Hamming distance of the output of the function. Let $y = x^{(0)}, x^{(1)}, \dots, x^{(k)} = x$ be a series of strings such that $\Delta_e(x^{(\ell-1)}, x^{(\ell)}) = 1$. Such strings exist by our assumption on the edit distance of x and y. Let i_ℓ be the first index on which $x^{(\ell-1)}$ and $x^{(\ell)}$ differ. To ease the notation assume $i_1 < \dots < i_k$ (we will remove the assumption in the end of the proof). For a fixed value of $h_1, \ldots, h_{3n}$ we define a function $i_x : [3n] \to [n+1]$ such that $i_x(j)$ is the value of i in the j-th iteration of Algorithm 1 applied on x.

Let d_j measure the difference between deleted and inserted bits between x and y that were seen till iteration j, i.e. d_0 is initialized to 0, and d_j is defined recursively as follows: whenever j is such that $i_x(j) \notin \{i_1, \ldots, i_k\}$ then $d_j = d_{j-1}$. Otherwise, suppose $i_x(j) = i_\ell$, then $d_j = d_{j-1} - 1$ when $x^{(\ell+1)}$ is obtained by deletion from $x^{(\ell)}$, it is d_{j-1} when it was a bit flip and $d_{j-1} + 1$ when it was a insertion. The main observation is as follows:

Observation 4.3. *Let $j \in [3n]$ be such that $i_x(j) \in [i_\ell, i_{\ell+1})$, then:*

1. *If $i_x(j) = i_y(j) + d_j$, then $x_{i_x(j)} = y_{i_y(j)}$ so $f(x,r)_j = f(y,r)_j$.*
2. *Moreover if $i_x(j) = i_y(j) + d_j$ and for every $j' \geq j$ if $i_x(j') < i_{\ell+1}$, then $i_x(j') = i_y(j') + d_j$. Overall, for every $j' \geq j$ satisfying $i_x(j') < i_{\ell+1}$ it holds that $f(x,r)_{j'} = f(y,r)_{j'}$.*

The first item follows easily by the definition of d_j and $x^{(0)}, \ldots, x^{(k)}$. As for the second item, observe that as long as $i_x(j') < i_{\ell+1}$ the increment of $i_x(j'), i_y(j')$ is identical as $x_{i_x(j')} = y_{i_y(j')}$, so in particular $h_{j'}(x_{i_x(j')}) = h_{j'}(y_{i_y(j')})$.

To bound $\Delta_H(f(x,r), f(y,r))$ we define the following randomized process which is induced by the behavior of the algorithm on x and y. The process consists of a particle moving randomly on the integer line and a goalspot. In the beginning both the particle and the goalspot are located at the origin. The process lasts $3n$ units of time.

The goal spot is moved according to d_j, i.e. whenever $i_x(j)$ hits an index i_ℓ, then the goalpost is moved according to the following rule:

- If $x^{(\ell)}$ is obtained from $x^{(\ell-1)}$ by a bit flip then the goalspot remains in its place.
- If $x^{(\ell)}$ is obtained from $x^{(\ell-1)}$ by a bit insertion then the goalspot shifts one step to the right.
- If $x^{(\ell)}$ is obtained from $x^{(\ell-1)}$ by a bit deletion then the goalspot shifts one step to the left.

The particle moves according to the following rule: If j is such that $f(x,r)_j = f(y,r)_j$ then the particle is idle. Otherwise, it makes a step according to $h_j(x_{i_x(j)}) - h_j(y_{i_y(j)})$. Clearly, $\Delta_H(f(x,r), f(y,r))$ equals the number of steps in which the particle is not idle (including the steps it remains in its place). Observe that whenever $f(x,r)_j \neq f(y,r)_j$ then $x_{i_x(j)} \neq y_{i_y(j)}$. Therefore, in such a case, since h_j is random, the particle shifts to the left/right with probability 1/4 and stays in its place with probability 1/2.

Let $j \in [3n]$ satisfying $i_\ell \leq i_x(j) < i_{\ell+1}$. The goalpost position in iteration j measures the difference between deleted and inserted bits in the series $x^{(1)}, \ldots, x^{(\ell)}$, namely it equals d_j. The particle position measures the difference between the increments of $i_x(j)$ and $i_y(j)$. Therefore, if the particle reaches the goalspot position then by Observation 4.3 it holds $x_{i_x(j)} = y_{i_y(j)}$. And by the second item of Observation 4.3 this implies that the particle would stay idle till the next iteration in which $i_x(j) = i_{\ell+1}$. Therefore, we need to analyze how many steps the particle performs after it becomes non-idle till it reaches the goalspot again. For this sake we define a new process that is easier to analyze:

The Kennel Struggle: Let us consider a process involving a dog, a cat and a kennel. All the involving entities are located on the integer line. In the beginning the kennel is located on the origin and so are the dog and the cat. The dog would like the cat to step out of the kennel. To this end the dog can perform one of the following actions:

The dog can bark, forcing the cat to perform a random step (defined shortly). Alternatively, the dog can move with kennel one step towards his preferred side. Whenever the cat is in the kennel the dog must perform an action. If the cat is not in the kennel, the dog may perform an action or stay idle. The dog's decision is based only on the cat position. Upon performing k actions the dog gives up and vanishes so the kennel is empty from then on (where $k = \Delta_e(x, y)$).

The cat, upon each dog barking, or whenever she is not at the kennel performs a random step: she steps to the left/right with probability 1/4 and stays in its place with probability 1/2. If the cat finds the kennel empty, then she happily stays there and the game is over.

It can be easily seen that for each configuration of the particle and goalpost process, there is a strategy for the dog such that: The distribution of the cat steps equals to the distribution of the particle moves, with the little change that we do not stop the particle and goalpost process after $3n$ steps (in the kennel struggle we skip the idle steps). Observe that if we do not end the particle and goalpost process after $3n$ steps the number of steps made by the particle is just larger. Therefore, an upper bound on the number of cat steps under any dog strategy, translates into an upper bound on the number of particle non-idle steps, which in turn bounds the Hamming distance of x and y.

Fix a strategy S for the dog, and denote by $p_S(\ell)$ the probability that after at most ℓ steps the cat reaches an empty kennel, provided that the dog acts according to S. The following claim implies Lemma 4.2.

Lemma 4.4. *Let $k \in \mathbb{N}$. Consider the kennel struggle, where the dog performs k actions. For every strategy S of the dog, $p_S(\ell) \geq \sum_{t=0}^{\ell} q(t, k)$.*

Proof. The lemma is a consequence of the following claim.

Claim 4.5. *The following dog's strategy $\mathcal{S}$ minimizes $p_{\mathcal{S}}(t)$ for every t: Wait till the cat reaches the kennel and then push the kennel to the right.*

Let us conclude the proof using the claim. Consider the dog's strategy given by Claim 4.5. In this strategy the probability $p_{\mathcal{S}}(\ell)$ is given by:

$$\sum_{t_1,\dots,t_k | t_1+\dots+t_k \leq \ell} (q(t_1, 1) \cdots q(t_k, 1)) = \sum_{t=0}^{\ell} \sum_{t_1,\dots,t_k | t_1+\dots+t_k = t} (q(t_1, 1) \cdots q(t_k, 1)). \tag{1}$$

On the other hand, the inner sum in (1) equals the probability that a random walk on line starting at the origin reaches place k at time t. To see that, observe that the last event can be phrased as follows: First compute the probability that the walk reaches place 1 in t_1 steps. Conditioned on that compute the probability that it reaches 2 in t_2 steps, and so on. Clearly, these events are independent. In order to get a total number of t steps we require $t_1 + \cdots + t_2 = t$.

Let us put things together. If the dog acts according to the strategy given by Claim 4.5 (which minimizes $p_{\mathcal{S}}(\ell)$ for every ℓ), then for all values of $\ell > 0$: $p_{\mathcal{S}}(\ell) = \sum_{t=0}^{\ell} q(t, k)$. The lemma follows. □

Proof of Claim 4.5. Consider the best strategy for the dog. That is, a series of decisions conditioned on the current position of the cat. We first argue that all of the actions performed by the dog must be pushing kennel one step away from the cat. Initially the distance between the cat and the kennel is 0. Now let us just consider a series of movements performed by the cat. Then we consider a strategy by the dog consisting of only one single action. We claim that for any value of $t \in \mathbb{N}$, the probability that the cat reaches an empty kennel within t steps, i.e., the quantity $p_S(t)$, is minimized when the dog pushes the kennel further from the cat. Let us first recall that for parameters $t \in \mathbb{N}, \ell \in \mathbb{Z}$, by $q(t,\ell)$, we denote the probability that a random walk on a line starting at the origin reaches the point ℓ at time t for the first time and by $p(t,\ell)$, we denote the probability that a random walk on a line starting at the origin reaches the point ℓ within time t.

Now if dog chooses to bark, then the value of the quantity $p_S(t)$ will be

$$\frac{1}{4}\sum_{i=0}^{t-1} q(i,-1) + \frac{1}{2}\sum_{i=0}^{t-1} q(i,0) + \frac{1}{4}\sum_{i=0}^{t-1} q(i,1) = \frac{1}{2} + \frac{1}{2}\sum_{i=0}^{t-1} q(i,1) = \frac{1}{2} + \frac{1}{2}p(t-1,1)$$

where the first equality follows from the symmetric nature of the function $q(\cdot,\cdot)$ and thus we get $q(i,1) = q(i,-1)$.

On the other hand, if the dog chooses to push the kennel further from the cat, then the value of the quantity $p_S(t)$ will be

$$\frac{1}{4}\sum_{i=0}^{t-1} q(i,0) + \frac{1}{2}\sum_{i=0}^{t-1} q(i,1) + \frac{1}{4}\sum_{i=0}^{t-1} q(i,2) = \frac{1}{4}p(t-1,0) + \frac{1}{2}p(t-1,1) + \frac{1}{4}p(t-1,2) \le \frac{1}{2} + \frac{1}{2}p(t-1,1).$$

So we can conclude that the first action taken by the dog that minimizes the probability of the cat reaching an empty kennel within t steps, is pushing the kennel one step away from the cat.

Now we iteratively add actions to the dog's strategy such that the probability that the cat reaches an empty kennel within t steps is minimized. Now we use inductive argument to prove that given the first $d \in \mathbb{N}$ actions by the dog are all pushing the kennel one step further from the cat, the $(d+1)$-th action also must be pushing the kennel one step away from the cat. As the first $d \in \mathbb{N}$ actions by the dog are all pushing the kennel one step further from the cat, just before $(d+1)$-th action by the dog, the distance of the cat from the kennel must be d. Now let us just consider a series of movements performed by the cat. Then we add a new action in the strategy set S for the dog. We claim that the quantity $p_S(t)$ is minimized when the dog pushes the kennel further from the cat. Now consider the following:

- If the dog chooses to bark and let us denote the corresponding strategy as S_1, then: for $d \ge 1$, $p_{S_1}(t) = \frac{1}{4}\sum_{i=0}^{t-1} q(i,d-1) + \frac{1}{2}\sum_{i=0}^{t-1} q(i,d) + \frac{1}{4}\sum_{i=0}^{t-1} q(i,d+1) = \sum_{i=0}^{t-1} q(i+1,d) = p(t,d)$.
- If the dog chooses to push the kennel towards the cat and let us denote the corresponding strategy as S_2, then:
  - for $d = 1$, $p_{S_2}(t) = \frac{1}{4}\sum_{i=0}^{t-1} q(i,1) + \frac{1}{2}\sum_{i=0}^{t-1} q(i,0) + \frac{1}{4}\sum_{i=0}^{t-1} q(i,-1) = \frac{1}{2} + \frac{1}{2}\sum_{i=0}^{t-1} q(i,1) = \frac{1}{2} + \frac{1}{2}p(t-1,1) \ge p(t-1,1) \ge p(t,2)$,
  - for $d \ge 2$, $p_{S_2}(t) = \frac{1}{4}\sum_{i=0}^{t-1} q(i,d-2) + \frac{1}{2}\sum_{i=0}^{t-1} q(i,d-1) + \frac{1}{4}\sum_{i=0}^{t-1} q(i,d) = \sum_{i=0}^{t-1} q(i+1,d-1) = p(t,d-1)$.
- If the dog chooses to push the kennel further from the cat and let us denote the corresponding strategy as S_3, then: for $d \ge 1$, $p_{S_3}(t) = \frac{1}{4}\sum_{i=0}^{t-1} q(i,d) + \frac{1}{2}\sum_{i=0}^{t-1} q(i,d+1) + \frac{1}{4}\sum_{i=0}^{t-1} q(i,d+2) = \sum_{i=0}^{t-1} q(i+1,d+1) = p(t,d+1)$.

The second equality in each case except the case corresponding to $d = 1$ for the strategy S_2 is obtained by Observation 3.1 Item 2. The second equality in the case corresponding to $d = 1$ for the strategy S_2 follows from the symmetric nature of the function $q(\cdot,\cdot)$ and thus we get $q(i,1) = q(i,-1)$

and the last inequality in this case is followed from a simple fact that to visit point 2 within time t, it has to visit point 1 within time $t-1$. Now by Item 1 of Observation 3.1, it follows that for $d \geq 1$, $p_{S_3}(t)$ is not larger than $p_{S_1}(t)$ or $p_{S_2}(t)$. So we can conclude that the $(d+1)$-th action taken by the dog that minimizes the probability of the cat reaching an empty kennel within t steps, is pushing the kennel one step away from the cat.

Next we divide the dog's strategy into intervals, where each interval lasts until the cat reaches the kennel. Observe first that the distribution on the number of steps made by the cat in each interval is independent on the other intervals. Therefore in order to conclude the claim we show that in each interval separately we can replace the dog action by waiting till the cat reaches the kennel and then pushing it to the right.

Fix such an interval. Now we show that if we replace the last action by waiting until the cat reaches the kennel, and then pushing it to the right, the distribution on the cat steps does not change. Let d' be the distance of the cat from the kennel before the dog pushes the kennel. The probability that the cat reaches the kennel in some t' steps is given by: $q(t', d'+1)$. If the dog instead waits until the cat reaches the kennel and then pushes the kennel to the right. Then probability that the cat reaches (again) the kennel in t' steps is given by: $\sum_{i=1}^{t'} q(i, d')q(t-i, 1)$ which, by Observation 3.1 Item 3, equals $q(t', d'+1)$, the claim follows. In such a way, it can be shown that each kennel push in this interval, can be be replaced by waiting till the cat reaches the kennel and then pushing the kennel to the right. □

Remark: Let us come back to the assumption that $i_1 < i_2 < \ldots, < i_k$: Observe that if for some subsequence $i_{j_1}, \ldots, i_{j_m}$ we have an equality, then this translates to several actions of the dog preformed at the same time unit (counted as m operations). By the same arguments applied before if the dog performs m multiple actions at the same unit time, then the following strategy would maximize the total number of steps made by the cat: Push the kennel m steps further from the cat. It is easy to see that the number of steps made by the cat is equivalent under the following two strategies: (i) push the kennel m steps further from the cat and (ii) wait till the cat reaches the kennel and then push it m times. Now the previous argument applies.

5 Reducing the randomness

In this section we show how to reduce the number of random bits used by Algorithm 1 to derive Theorems 1.1 and 1.2.

An easy observation is that for the Boolean case, one can save a half of the random bits needed for our basic embedding function given by Algorithm 1. We may replace the current condition for incrementing i that $h_j(x_i) = 1$ by the condition $r_j = x_i$. As one can verify in our analysis of the third property in Theorem 4.1 this would not substantially affect the property of the algorithm. It would actually improve the bound on the resulting Hamming distance by a factor of roughly 2 because the induced random walk would be non-lazy. To obtain more substantial savings we will use tools from derandomization.

By standard probabilistic method similar to [Gol01, Proposition 3.2.3], we argue that there exists a subset of $\{0,1\}^{6n}$ of substantially *small* size (of size at most $2n^{c'}$), from which sampling r is "almost" as good as sampling it uniformly from $\{0,1\}^{6n}$. Thus by hard-wiring R inside the algorithm and sampling r from R, we get the desired non-uniform algorithm promised in Theorem 1.1, details below.

By choosing c appropriately and for large enough n, our basic embedding function f has the property that with probability at least 3/4, for random $r \in \{0,1\}^{6n}$

$$\frac{1}{2} \cdot \Delta_e(x,y) \leq \Delta_H(f(x,r), f(y,r)) \leq c \cdot (\Delta_e(x,y))^2.$$

Now take a random subset $R \subseteq \{0,1\}^{6n}$ of the size of the smallest power of two $\geq n^{c'}$, for some suitable constant $c' > 0$. Fix $x, y \in \{0,1\}^n$. By Chernoff bound, the probability that

$$\left| \Pr_{r \in \{0,1\}^{6n}} \left[\frac{1}{2} \cdot \Delta_e(x,y) \leq \Delta_H(f(x,r), f(y,r)) \leq c \cdot (\Delta_e(x,y))^2 \right] \right.$$
$$\left. - \Pr_{r \in R} \left[\frac{1}{2} \cdot \Delta_e(x,y) \leq \Delta_H(f(x,r), f(y,r)) \leq c \cdot (\Delta_e(x,y))^2 \right] \right| > \frac{1}{n}$$

over the random choice of R is at most 2^{-2n}. Hence, by the union bound over all x and y, there is a set R of the required size such that for any $x, y \in \{0,1\}^n$,

$$\left| \Pr_{r \in \{0,1\}^{6n}} \left[\frac{1}{2} \cdot \Delta_e(x,y) \leq \Delta_H(f(x,r), f(y,r)) \leq c \cdot (\Delta_e(x,y))^2 \right] \right.$$
$$\left. - \Pr_{r \in R} \left[\frac{1}{2} \cdot \Delta_e(x,y) \leq \Delta_H(f(x,r), f(y,r)) \leq c \cdot (\Delta_e(x,y))^2 \right] \right| \leq \frac{1}{n}.$$

Thus instead of sampling r from the whole universe $\{0,1\}^{6n}$ we can sample r from R without affecting the probability of small distortion by more than $1/n$. Since R is of size at most $2n^{c'}$, we need only $\log|R| = \lceil c' \log n \rceil$ random bits to sample a random element of R. The non-uniform algorithm for the embedding function of Theorem 1.1 has R hard-wired as a table. On input x and $s \in \{0,1\}^{\log|R|}$ the algorithm simulates Algorithm 1 on x and the s-th string in R. By properties of R we know that such an algorithm satisfies the conclusion of Theorem 1.1.

The above algorithm has an optimal seed length but it has the disadvantage of storing a large table of non-uniformly selected strings (the subset R). To get rid of the table we will use Nisan's pseudo-random generator [Nis90].

Nisan's pseudo-random generator $G_{k,w}$ is a function that takes a seed s of length w and k pair-wise independent hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^w \to \{0,1\}^w$ and outputs a string $r \in \{0,1\}^{w2^k}$ defined recursively as follows:

$$G_{0,w}(s) = s$$
$$G_{k,w}(s, h_1, h_2, \ldots, h_k) = G_{k-1,w}(s, h_1, \ldots, h_{k-1}) \odot G_{k-1,w}(h_k(s), h_1, \ldots, h_{k-1})$$

Nisan proved that his generator satisfies the following property.

Theorem 5.1 ([Nis90]). *For an arbitrary constant $c_0 > 0$, let A be an algorithm that uses work space of size at most $c_0 \log n$ and runs in time at most n^{c_0} with a two-way access to its input string x and a one-way access to a random string r. There is a constant $c_1 > 0$ such that*

$$\left| \Pr_r[A(x,r) \text{ accepts}] - \Pr_{r'}[A(x,r') \text{ accepts}] \right| < \frac{1}{n^2}$$

where r is taken uniformly at random, and r' is taken according to the distribution induced by $G_{c_0 \log n, w}(s, h_1, \ldots, h_{c_0 \log n})$ where $w = c_1 \log n$, $s \in \{0,1\}^w$ is taken uniformly at random and each h_i is sampled independently from an ensemble of pair-wise independent hash functions.

There are ensembles of pair-wise independent hash functions mapping w bits into w bits where each function is identified by a binary string of length $O(w)$. Nisan [Nis90] gives several such examples and there are many others. In particular, the ensemble given by Dietzfelbinger [Die96] can be evaluated on word RAM with word size $O(w)$ using $O(1)$ multiplications and bit operations.

We would like to apply Theorem 5.1 on Algorithm 1. However, Theorem 5.1 applies only for decision algorithms. Therefore we define the following algorithm A:

Algorithm 2 Hamming Distance Test

```
Input  : $x, y \in \{0,1\}^n$, $k \in \{1, \ldots, 3n\}$, and a random string $r \in \{0,1\}^{6n}$
Output: Accept iff: $\Delta_H(f(x,r), f(y,r)) = k$
Compute $\Delta_H(f(x,r), f(y,r))$ for the basic embedding function $f$ by simultaneously computing
  $f(x,r)$ and $f(y,r)$ while counting the Hamming distance of $f(x,r)$ and $f(y,r)$;
Accept if $\Delta_H(f(x,r), f(y,r)) = k$;
```

Given the properties of our Algorithm 1 for the basic embedding function, it is clear that A processes its input in logarithmic space using one-way access to its random string r. Hence, we can apply Theorem 5.1 on algorithm A. That implies that the distributions of the Hamming distance $\Delta_H(f(x,r), f(y,r))$ on a random string r and a random string r' sampled according to Nisan's pseudo-random generator are close in ℓ_∞ distance.

Hence, instead of providing Algorithm 1 with completely random string we will provide it with a seed of length $O(\log n)$ and a sequence $O(\log n)$ of hash functions that will be expanded by the Nisan's pseudo-random generator into a full pseudo-random string r'. This r' is used in place of r to compute $f(x, r')$. Since each hash function can be specified using $O(\log n)$ bits this algorithm will require only $O(\log^2 n)$ random bits in total.

Furthermore, it is clear from the description of the Nisan's pseudo-random generator, that each bit of r' can be obtained by evaluating at most $O(\log n)$ hash functions. When computing r' bit by bit we only need to evaluate $O(1)$ hash function on average. Thus when using Dietzfelbinger's hash functions on a RAM with word size $O(\log n)$ we can compute $f(x, r')$ in a streaming fashion spending only $O(1)$ operations per output bit on average and $O(\log n)$ in the worst-case. This proves Theorem 1.2.

6 Non-binary alphabets

Our results carry over directly to larger alphabets of constant size. For alphabets Σ_n where the size of Σ_n depends on n we assume that the symbols are binary encoded by strings of length $\log |\Sigma_n|$. Our basic embedding given by Algorithm 1 only needs that each $h_1, \ldots, h_{3n}$ is a pair-wise independent hash function from Σ_n to $\{0,1\}$. Such a hash function is obtained for example by selecting a random vector $r \in \{0,1\}^{\log |\Sigma_n|}$ and a bit b, and taking the inner product of the binary encoding of an alphabet symbol with r and adding b over GF_2. Hence, one needs only $1 + \log |\Sigma_n|$ bits to specify each hash function.

Thus Theorem 4.1 will use $n \cdot (1 + \log |\Sigma_n|)$ random bits, Theorem 1.1 will have $\ell = O(\log n + \log \log |\Sigma_n|)$ and Theorem 1.2 will have $\ell = O((\log n + \log \log |\Sigma_n|)^2)$.

7 Computing the Edit Distance

In the following three sections we develop our streaming algorithm for computing edit distance. The algorithm is based on first extracting a kernel from its input that preserves the edit distance and then processing the kernel using a known off-line algorithm for edit distance. The next section builds the necessary combinatorial tools while the subsequent sections provide the algorithmic tools.

7.1 Kernelization: Information Theoretic Perspective

In this section we present the main combinatorial tools to produce a kernel from two strings x and y. An *alignment of two strings* $x, y \in \{0,1\}^n$ is a function $a : [n] \to [n] \cup \{\mathbf{S}, \mathbf{D}\}$ that is increasing, i.e., for all $i < j$, if $a(i), a(j) \in [n]$ then $a(i) < a(j)$, and that maps symbols of x to corresponding symbols of y, i.e., for all i, if $a(i) \in [n]$ then $x_i = y_{a(i)}$. The alignment represents edit operations that produce y from x: x_i is deleted when $a(i) = \mathbf{D}$, it is substituted by another symbol when $a(i) = \mathbf{S}$ and it is preserved and mapped to $a(i)$-th symbol of y otherwise. (Insertions are encoded in an alignment only implicitly and there might be an ambiguity as to where precisely a substituted symbol is mapped.) The *cost* of an alignment is the cost of the associated edit operations, i.e., $c(a) = 2|a^{-1}(\mathbf{D})| + |a^{-1}(\mathbf{S})|$. (The number of insertions equals the number of deletions.) Clearly, the edit distance of x and y is the cheapest cost of an alignment of x and y. For an alignment a we let its *displacement* be $d(a) = \max_{i, a(i) \in [n]} |a(i) - i|$. Clearly, $d(a) \le c(a)$. For $i \le j \in [n]$, the *block* $x_{i,\dots,j}$ is the substring $x_i x_{i+1} \cdots x_j$. We say that *a preserves the block* $x_{i,\dots,j}$ if for all $\ell \in [i, j)$, $a(\ell + 1) = a(\ell) + 1$. Note, a block $x_{i,\dots,\ell}$ might not be preserved under a even though $x_{i,\dots,j}$ and $x_{j+1,\dots,\ell}$ are preserved for some j as there might be an insertion between the two blocks.

An integer ℓ is a *period* of a string w if there exist $p \in \{0,1\}^\ell$ and an integer r such that w is a prefix of p^r. A string is *periodic* if its minimum period is less than a half of its length.

The following lemma shows whenever two strings share a long periodic substring, then a deflation of some of the periods in both strings preserves the edit distance between the original strings.

Lemma 7.1 (Deflation)**.** *Let $x, y \in \{0,1\}^n$. Let $x = uwv$ and $y = u'wv'$ for some strings u, w, v, u', v'. Let K and k be integers such that $\Delta_e(x, y) \le k$ and $||u| - |u'|| \le K$. Let ℓ be the minimal period of w and $p \in \{0,1\}^\ell, r > 0$ be such that $w = p^r$. Let $t = 2K + 3k + (\ell + 2) \cdot (k + 1)$. If $|w| \ge t$ then for all r' such that $r' \ge t/\ell$, $\Delta_e(x, y) = \Delta_e(up^{r'}v, u'p^{r'}v')$.*

Proof. Let a be an alignment of x and y of the minimum cost. The only symbols from x's copy of w that can be mapped outside of y's copy of w are the first and last $K + d(a) \le K + k$ symbols of x's w. Thus at least $|w| - 2(K + k)$ symbols of x's w are mapped to $y's$ w or are deleted or substituted. Hence, there must be a block in x's w of length at least $(|w| - (2K + 3k))/(k + 1) \ge \ell + 2$ that is preserved under a and mapped within y's w. Pick an inner sub-block of length ℓ of the preserved block so the sub-block does not contain the first and last symbol of the preserved block. We can remove the sub-block from x and its image from y and shift the alignment to get an alignment of $up^{r-1}v$ and $u'p^{r-1}v'$ of the same cost. (The removal does not affect periodicity of w as we are removing a block of size ℓ.) Hence, $\Delta_e(up^{r-1}v, u'p^{r-1}v') \le \Delta_e(x, y)$. Similarly, instead of removing the sub-block, insert its copy immediately after the sub-block in x and after its image in y. Again, extending the alignment yields, $\Delta_e(up^{r+1}v, u'p^{r+1}v') \le \Delta_e(x, y)$. As this holds for all $r \ge t/\ell$ the edit distance is always the same. ☐

The following property of periodic strings is well known (see Proposition 2.1 in [CR94]).

Proposition 7.2. *Let $w, u, v \in \{0,1\}^*$ be such that $wu = vw$. Then $w = p^r u'$ for some strings p, u' and an integer r such that $|u'| \le |p| \le |u|$.*

The next lemma deals with strings which are far from being periodic. A string w is called *(t,ℓ)-periodic free* if no block w of length more than t is periodic with a period smaller or equal to ℓ. The lemma shows that if x, y share a long periodic free substring, then we can shrink the size of the shared part of x and y while preserving the edit distance.

Lemma 7.3 (Shrinkage)**.** *Let $x, y \in \{0,1\}^n$. Let $x = uwv$ and $y = u'wv'$ for some strings u, w, v, u', v'. Let K, k and t be integers such that $\Delta_e(x,y) \le k$, $\big||u| - |u'|\big| \le K$, and assume w is $(t, K+k)$-periodic free. Let $s = K + 2k + (k+1)\cdot(t+1)$. For any $s' \ge s$, if $|w| \ge 2s'$ and $w' = w_{1,\dots,s'} w_{|w|+1-s',\dots,|w|}$ then $\Delta_e(x,y) = \Delta_e(uw'v, u'w'v')$.*

Proof. Let a be an alignment of x and y of the minimum cost. Assume $|w| \ge 2s$. By the same argument as in Lemma 7.1 there is a preserved sub-block $w_{i,\dots,j}$ of $w_{1,\dots,s}$ of length at least $(s-K-2k)/(k+1) = t+1$ that is mapped by a within y's copy of w. The image of $w_{i,\dots,j}$ must overlap $w_{i,\dots,j}$ in y in at least $j-i+1-(K+k)$ symbols. This implies that either $w_{i,\dots,j}$ is periodic with a period at most $K+k$ by Proposition 7.2 or $w_{i,\dots,j}$ is mapped to itself in y. The former possibility is excluded by the assumption that w is $(t, K+k)$-periodic free so $w_{i,\dots,j}$ is mapped to itself in y. Similarly, there must be a sub-block $w_{i',\dots,j'}$ within the last s symbols of w that is mapped to itself in y's copy of w. Hence, all the symbols between $w_{i,\dots,j}$ and $w_{i',\dots,j'}$ must also be mapped to itself as a is of minimal cost. Thus, the whole $w_{i,\dots,j'}$ is preserved by a and $\Delta_e(x,y) \ge \Delta_e(uw'v, u'w'v')$. On the other hand, if we take the minimum cost alignment a' of $uw'v$ and $u'w'v'$ then by the same argument on these words the block $w_{s',s'+1} = w_{s'} w_{|w|+1-s'}$ must be preserved by a' and mapped to itself in y. So the alignment a' can be extended into the same cost alignment of x and y giving $\Delta_e(x,y) \le \Delta_e(uw'v, u'w'v')$. The lemma follows. □

7.2 Kernelization: Algorithmic Perspective

Let K, k be some chosen parameters. We provide an algorithm that given two strings x, y, where $\Delta_e(x,y) \le k$, and an alignment a of cost at most K, computes a pair of strings x', y' (*kernel*) of length $O(K^2k^2)$ so that $\Delta_e(x',y') = \Delta_e(x,y)$. The algorithm is given next. In this section we are not concerned with the performance of the algorithm. We will focus on the performance only in Section 7.3.

Algorithm 3 Kernelization(x, y, a, K, k)

```
Input  : $x, y \in \{0,1\}^n$, an alignment $a$ of $x$ and $y$, integers $K, k \ge 5$ such that $\Delta_e(x,y) \le k$ and
         $c(a) \le K$
Output: $x', y' \in \{0,1\}^{O(K^2k^2)}$ such that $\Delta_e(x',y') = \Delta_e(x,y)$.
Decompose $x = u_0 w_1 u_1 \cdots w_\ell u_\ell$, where $\ell \le K+1$, each $w_i$ is a maximal preserved block of $x$ under
  $a$, and $u_i \in \{0,1\}^*$.
Decompose $y = v_0 w_1 v_1 \cdots w_\ell v_\ell$, where $v_i \in \{0,1\}^*$.
For each $i$ let $w'_i = \text{Shrink}(\text{Deflate}(w_i, 3(K+k)k, K+k), 5(K+k)k^2)$.
Output $x' = u_0 w'_1 u_1 \cdots w'_\ell u_\ell$ and $y' = v_0 w'_1 v_1 \cdots w'_\ell v_\ell$.
```

Algorithm 4 Deflate(w, t, ℓ)

```
Input  : $w \in \{0,1\}^n$, integers $\ell \geq 1$, $t > 4\ell$
Output: Substring $w'$ of $w$ such that $w'$ is $(t,\ell)$-periodic free.
Initialization: $Output = \lambda$;
while $w$ is non-empty do
    if $w = p^r v$ for some maximum $r \geq t/|p|$ and some $p, v \in \{0,1\}^*$, $|p| \leq \ell$ then
        $Output = Output \odot p^{\lceil t/|p| \rceil}$;
        $w = v$;
    end
    else
        $Output = Output \odot w_1$;
        $w = w_{2,\dots,|w|}$
    end
end
Output $Output$.
```

Algorithm 5 Shrink(w, s)

```
Input  : $w \in \{0,1\}^n$, an integer $s \geq 1$
Output: $w_{1,\dots,s} w_{|w|+1-s,\dots,|w|}$ if $|w| \geq 2s$ and $w$ otherwise.
```

Proposition 7.4. *For $K \geq k \geq 5$ the procedure* Kernelization(x, y, a, K, k) *outputs x', y' such that $\Delta_e(x', y') = \Delta_e(x, y)$ and $|x'| = |y'| \leq 12K(K+k)k^2$.*

Notice that one can easily annotate the kernel, i.e., strings x' and y', so that given an alignment of x' and y' we can determine the alignment of x and y. One just needs to know which parts of x' and y' were deflated and by how much and how many symbols were removed by Shrink. From this information one can also directly infer the edit operations that take x into y for the price of the alignment.

Proof. By Lemma 7.1, removing repetitions of a period $\leq K + k$ that span beyond $3(K+k)k \geq 2K + 3k + (K+k+2)\cdot(k+1)$ symbols does not change the edit distance of x and y. Hence, deflation of each w_i preserves the edit distance. Once deflated, each w_i is $(3(K+k)k, K+k)$-periodic free. Hence, by Lemma 7.3 we can shrink each deflated w_i to its borders of size $5(K+k)k^2 \geq K + 2k + (k+1)\cdot(4(K+k)k+1)$ while preserving the edit distance. For the size of x', there are at most $K+1$ different w'_i and each of them is of size at most $10(K+k)k^2$. ☐

Remark A useful observation can be made from the previous proof. If on strings x, y and $K \geq k \geq 5$ the procedure Kernelization(x, y, a, K, k) outputs x', y' such that $\Delta_e(x', y') \leq k$ then $\Delta_e(x, y) = \Delta_e(x', y') \leq k$. Put in contrapositive, for x, y of distance $\Delta_e(x, y) > k$, the Kernelization(x, y, a, K, k) produces x', y' such that $\Delta_e(x', y') > k$. Given that the procedure shrinks the strings this is not automatic nevertheless it is true.

It is clear that the kernelization procedure can be implemented in polynomial time with random access to a, x and y. We claim next that it can be implemented in linear time with one-way (streaming) access to a, x and y. Furthermore, we show that in one pass we can find a candidate alignment and a kernelization based on that alignment. This will be the content of the next section.

7.3 Kernelization in a Streaming Fashion

In this section we describe a streaming algorithm to compute the edit distance of x and y. First we describe how to compute a small kernel of x and y. We start with a streaming algorithm based on Saha's algorithm [Sah14] or our embedding procedure that obtains a candidate alignment a:

CandidateAlignment(x, y): We maintain two pointers i_x, i_y to the input strings x, y which are both initialized to 1 and we repeat the following: If $x_{i_x} = y_{i_y}$, then the strings are aligned so we set $a(i_x) = i_y$ and increment both i_x and i_y by 1. Otherwise, with probability $1/2$ we set $a(i_x) = \mathbf{D}$ and increment only i_x by 1, and with the remaining probability we increment only i_y by 1. When $i_y = n + 1$ or $i_y = n + 1$ we stop. If $i_y = n + 1$ and $i_x < n$ we set $a(i_x), ..., a(n) = \mathbf{D}$.

Lemma 7.5. *Let $x, y \in \{0,1\}^n$ and let $k = \Delta_e(x, y)$. Let $a =$ CandidateAlignment(x, y). Then with probability at least $2/3$, $c(a) < ck^2$, where c is the big-O constant in the distortion factor of the embedding in Theorem 1.1.*

The correctness of Lemma 7.5 follows directly from the correctness and properties of the embedding function that uses $r_j = x_i$ as the rule for incrementing i which was introduced in Section 5. It also follows from the correctness of Saha's algorithm. We will also use the following lemma in the analysis:

Lemma 7.6. *There exists a linear time algorithm that on input $w \in \{0,1\}^n$, computes the shortest p such that w can be decomposed as $w = p^r u$ for some prefix u of p.*

The lemma is proven by reducing the problem to the task of string matching: That is, given a text *text* and a pattern *pat*, verify whether *pat* occurs in *text*. If it does then output the first index in which it appears. The later task can be solved in linear time using [KJP77] algorithm. See [CR94] for details.

The Deflate and Shrink procedure (with parameters (K, k)) can be implemented simultaneously in one-way fashion using the procedure described next.

StreamingKernelization(K, k): Set $t = 3(K + k)k, \ell = K + k$ and $s = 5(K + k)k^2$. We maintain an output buffer B_{out} of size $2s$ whose last s bits are cyclic (i.e. the k-th output symbol is stored in location $k - 1$ if $k \leq 2s$ and $s + (k \mod s)$ otherwise). Now, whenever the alignment procedure encounters a beginning of a new preserved block w (i.e, when $x_{i_x} = y_{i_y}$), it proceeds as follows:

Take the next $t/2$ unprocessed symbols from w, call that word z. Use the linear-time algorithm whose existence is implied by Lemma 7.6 to find the smallest p such that $|p| \leq \ell$ and $z = p^r u$, for some prefix u of p. If no such p exists we store z in B_{out} and continue processing the input. If there is such a p, we check how many times p appeared previous to z and how many times it will appear in the input starting from z. The t previously output symbols that could possibly contain p are still in our output buffer so we use them to check the former count and we will process the input further to count the subsequent repetitions of p. This can be easily done in time linear in the number of processed symbols. We output the deflated number of repetitions of p into B_{out} taking into account the repetitions already in the output. Proceed with the rest of w. When w is fully processed then by rearranging the content of B_{out}, the value of Shrink(Deflate$(w, t, \ell), s)$ can be extracted.

The key point is that whenever w contains a period p that spans beyond t symbols, then while processing w in blocks of size $t/2$ there must be a block containing only periods of p. As we reach this block we will correctly deflate p's repetitions. When we reached the end of each preserved block w, then the buffer B_{out} contains the first s bits of the deflated w and the last bits of w stored in a cyclic shift. So after rearranging the buffer B_{out} indeed stores Shrink(Deflate($w, 3(K+k)k, K+k$), $5(K+k)k^2$).[3]

Overall, we can compute Deflate and Shrink during the same pass over x and y that computes their candidate alignment. For this end, we use buffers of size t and $2s$. Now we are ready to state our first streaming algorithm, that attains success probability 2/3.

Theorem 7.7. *There is a probabilistic algorithm that on input $x, y \in \{0,1\}^n$ and an integer k, such that $\Delta_e(x,y) \leq k$, with probability at least 2/3 outputs $m = \Delta_e(x,y)$, and a series of m edit operations transforming x into y. The algorithm accesses x and y in one-way manner, runs in time $O(n+k^6)$ and uses space $O(k^6)$.*

Proof. Let c be the constant term that appears in Lemma 7.5. We invoke CandidateAlignment(x,y) procedure combined with the procedure StreamingKernelization($K = ck^2, k$). By Lemma 7.5 the alignment a obtained by CandidateAlignment satisfies $c(a) < K$ with probability at least 2/3. Suppose that this is indeed the case, then for any preserved block w found by a we have: $x = uwv, y = u'wv'$ with $||u| - |u'|| \leq K$. Hence, we can apply Proposition 7.4 and obtain that the output of the procedure StreamingKernelization is x', y' such that $\Delta_e(x', y') = \Delta_e(x,y)$ and $|x'| = |y'| \leq 12K(K+k)k^2 = O(k^6)$. Now apply the algorithm of [LMS98] that runs in time $(|x'| + k^2) = O(k^6 + k^2)$ and uses space $O(|x|) = O(k^6)$. The proof follows. □

Note that there is nothing special about the alignment obtained by our embedding. We can take any candidate alignment and apply the kernelization process in order to shrink the input size.

Theorem 7.8 (Streaming Kernelization)**.** *There is an algorithm that on input $x, y \in \{0,1\}^n$, an alignment a of x and y, and parameters $K \geq k \geq 5$ satisfying $d(a) \leq K$ and $\Delta_e(x,y) \leq k$ outputs strings x', y' of length $O(K^2k^2)$ such that $\Delta_e(x,y) = \Delta_e(x', y')$, where the algorithm accesses a, x and y in one-way manner, runs in time $O(n + K^2k^2)$ and uses space $O(K^2k^2)$.*

One can substantially improve the error probability to obtain the following result, which is just a restatement of Theorem 1.3.

Theorem 7.9 (Streaming Edit Distance)**.** *There is a probabilistic algorithm that on input $x, y \in \{0,1\}^n$ and an integer s accesses x and y in one-way manner, runs in time $O(n + s\log n)$, uses space $O(s)$ and if $\Delta_e(x,y) \leq s^{1/6}$ then with probability at least $1 - (1/n)$ correctly outputs $\Delta_e(x,y)$. In all the remaining cases the algorithm outputs 'I DO NOT KNOW'.*

Proof. We provide a sketch of the proof. Let s be given, set $k = s^{1/6}$. Our goal is to compute a kernel of size $c_1k^6 = c_1s$, for some constant $c_1 > 0$, for each of the strings. Assume for a moment that we have a multiple access to the inputs. Then by running our CandidateAlignment procedure repeatedly if necessary, we can find an alignment of cost $\leq c_2k^2$, where $c_2 > 0$ is some constant to be fixed later. By Theorem 1.1, the probability that a particular run of CandidateAlignment procedure fails to produce an alignment of cost $\leq c_2k^2$ is constant. Thus the probability that we would need in total more than $O(\log n)$ runs of the CandidateAlignment procedure for the input strings can be made less than $1/n$ by a suitable choice of constants. Hence, with probability at least $1 - 1/n$ we can produce a cheap alignment in time $O(n\log n)$.

[3]There is a negligible difference in the output of the streaming deflation algorithm and the deflation Algorithm 4 as we ignore a partial suffix of p occuring before the first repetition of p. One could modify the streaming deflation accordingly.

The described procedure might need several passes over the input strings when the procedure CandidateAlignment fails to output a cheap alignment and thus also takes time $O(n \log n)$. To mitigate this, we run the kernelization procedure on-line fed by the output of the CandidateAlignment procedure, and reading x and y in sync with the CandidateAlignment procedure. As we are computing the kernel, once the cost of the current alignment reaches c_2k^2, we stop the CandidateAlignment and kernelization procedures and we re-run them both with fresh randomness. The partial kernel produced so far is of size at most c_1s. At this point we do not have access to the part of x and y that we had already turned into the partial kernel. However, this partial kernel concatenated together with the rest of the portions of x and y has the same edit distance as the original strings. Note that each time we encounter a failure due to an alignment of cost more than c_2k^2, it is guaranteed that $||u| + |u'|| \leq c_2k^2 + k$ (as the candidate alignment may shift the strings to one direction, and the optimal to the opposite direction), where $x = uwv, y = u'wv'$ for any preserved block w found by that alignment. Let us recall that we set $K = ck^2$ in the proof of Theorem 7.7. Thus we take $c_2 < c$ such that $||u| + |u'|| \leq K$ and hence we can use Lemma 7.1 and Lemma 7.3 to argue that the partial kernel together with the remaining input preserves edit distance. So instead of re-running the CandidateAlignment and kernelization procedures on x and y, we re-run them on the stored kernel and then continue with the remaining part of strings still in the input. This can be repeated until we produce a kernel of size at most c_1s for the whole input string or we fail the whole algorithm due to too many repetitions. The fact that the partial kernel concatenated with the remaining portions of the strings has the same edit distance as the original strings can be seen directly from the kernelization algorithm as this corresponds to setting some of the w'_i to w_i in both of the output strings.

Now let us analyze the time and space requirement of the above stated implementation of our algorithm. The space used by the algorithm is clearly $O(s)$. For the bound on time requirement, observe that in the above implementation, we only need to process each bit of x and y only once and during the run of the algorithm, intermediate kernel of length at most c_1s corresponding to each input string needs to be processed at most $O(\log n)$ times. Thus total time required is bounded by $O(n + s \log n)$.

If the algorithm succeeds in producing a kernel x', y' of x and y respectively each of size $O(s)$, we run the edit distance algorithm of [LMS98] on x' and y'. Since $k^2 \leq s$ this algorithm will run in time $O(s)$ and will use space $O(s)$. If the algorithm determines that the edit distance of x' and y' is at most k then we output the edit distance otherwise we output 'I DO NOT KNOW'. □

If we allow the algorithm multiple passes over the input instead of supplying the parameter s we can try a sequence $s_1, s_2, \ldots$ of s, where $s_1 = 2$ and $s_i = \lceil s_{i-1}^{1+\epsilon} \rceil$. Since $\log(1+\epsilon) \geq \epsilon/2$ for a non-negative $\epsilon \leq 1$, this gives the following corollary.

Corollary 7.10. *For every $\epsilon > 0$, there is a probabilistic algorithm that on input x, y computes $k = \Delta_e(x, y)$ with probability at least $1 - 1/n$, runs in time $O(\epsilon^{-1} n \log\log k + k^{6+\epsilon} \log n)$, space $O(k^{6+\epsilon})$ and makes at most $\log\log k / \log(1 + \frac{\epsilon}{6})$ passes over x and y.*

8 Other Applications of the Main Result in Communication Complexity

As a consequence of Theorem 1.1, we achieve a better bound on number of bits transmitted in a one-way protocol solving the document exchange problem. The most obvious protocol in this regard is the following [Jow12]: Alice and Bob first compute $f(x, r)$ and $f(y, r)$, where r is the shared randomness and then they run the one-way protocol for document exchange problem under Hamming metric. We use the following lemma from [PL07] which provides an efficient sketching algorithm in case of Hamming distance.

Lemma 8.1 ([PL07])**.** *For two strings $x, y \in \{0, 1\}^n$ such that $\Delta_H(x, y) \leq h$, there exists a randomized algorithm that maintains sketches of size $O(h \log n)$ and using sketches $s_h(x)$ and $s_h(y)$, it outputs all the tuples $\{(x_i, y_i)\}$ where $x_i \neq y_i$ with probability at least $(1 - 1/n)$ in $O(h \log n)$ time. Construction of sketch $s_h(x)$ can be done in $O(n \log n)$ time and space in one pass.*

Now if $\Delta_e(x, y) \leq k$, Bob will learn $f(x, r)$ and then using decoding algorithm he can get back x. After having x, Bob can decide $\Delta_e(x, y) \leq k$ in $O(n + k^2)$ time using the algorithm by [LMS98]. This idea leads us to the following corollary.

Corollary 8.2. *In the two-party communication model, there is a randomized one-way protocol that solves document exchange problem with high probability while transmitting only $O(k^2 \log n)$ bits. The running time of each party will be $O(n \log n + k^2 \log n)$.*

Another straightforward but important application of Theorem 1.1 is that it provides us a randomized sketching algorithm for k vs. ck^2 gap edit distance problem for some constant c. For this purpose, we need the following lemma from [BYJKK04].

Lemma 8.3 ([BYJKK04])**.** *For any $\epsilon > 0$ and k, k vs. $(1 + \epsilon)k$ gap Hamming distance problem can be solved using an efficient sketching algorithm that maintains sketches of size $O(1/\epsilon^2)$ and if the set of non-zero coordinates of each input string can be computed in time t, then running time of Alice and Bob will be bounded by $O(\epsilon^{-3} t \log n)$.*

Now given two input strings x and y, we can first use embedding f of Theorem 1.1 and then apply the above lemma to get the following corollary.

Corollary 8.4. *There exists a $c \in \mathbb{N}$ such that for any k, there is a randomized sketching algorithm that solves k vs. ck^2 gap edit distance problem with high probability using sketches of size $O(1)$ attaining an upper bound of $O(n \log n)$ on Alice and Bob's running time.*

Among other implications of embedding, one interesting problem is to design *approximate nearest neighbor search* algorithms which is defined as given a database of m points, we have to pre-process such that given a query point, it would be possible to efficiently find a database point close to the query point. For Hamming metric, a search algorithm is known [IM98] that retrieves a database point which is at most $(1 + \epsilon)$ times far from the closest one. Together with that, our embedding result implies a randomized algorithm that will return a point (under edit distance metric) within the distance of $O(k)$ times that of the closest one.

Acknowledgments

The authors thank anonymous reviewers for pointing out [Sah14, DW1] and other useful comments.

References

[ADG$^+$03] Alexandr Andoni, Michel Deza, Anupam Gupta, Piotr Indyk, and Sofya Raskhodnikova, *Lower bounds for embedding edit distance into normed spaces.*, SODA, ACM/SIAM, 2003, pp. 523–526.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak, *Polylogarithmic approximation for edit distance and the asymmetric query complexity*, 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA, 2010, pp. 377–386.

[AO09] Alexandr Andoni and Krzysztof Onak, *Approximating edit distance in near-linear time*, Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '09, ACM, 2009, pp. 199–204.

[BEK$^+$03] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami, *A sublinear algorithm for weakly approximating edit distance*, Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '03, ACM, 2003, pp. 316–324.

[BES06] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp, *Oblivious string embeddings and edit distance approximations*, Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm (Philadelphia, PA, USA), SODA '06, Society for Industrial and Applied Mathematics, 2006, pp. 792–801.

[BI15] Arturs Backurs and Piotr Indyk, *Edit distance cannot be computed in strongly subquadratic time (unless seth is false)*, Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (New York, NY, USA), STOC '15, ACM, 2015, pp. 51–58.

[BYJKK04] Z. Bar-Yossef, T.S. Jayram, R. Krauthgamer, and R. Kumar, *Approximating edit distance efficiently*, Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on, Oct 2004, pp. 550–559.

[CM02] Graham Cormode and S. Muthukrishnan, *The string edit distance matching problem with moves*, Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA., 2002, pp. 667–676.

[CPSV00] Graham Cormode, Mike Paterson, Süleyman Cenk Sahinalp, and Uzi Vishkin, *Communication complexity of document exchange*, Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (Philadelphia, PA, USA), SODA '00, Society for Industrial and Applied Mathematics, 2000, pp. 197–206.

[CR94] Maxime Crochemore and Wojciech Rytter, *Text algorithms*, Oxford University Press, 1994.

[Die96] Martin Dietzfelbinger, *Universal hashing and k-wise independent random variables via integer arithmetic without primes*, STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings, 1996, pp. 569–580.

[DW1] *Dortmund workshop on algorithms for data streams 2012.*

[Gol01] Oded Goldreich, *The foundations of cryptography - volume 1, basic techniques*, Cambridge University Press, 2001.

[IM98] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '98, ACM, 1998, pp. 604–613.

[Jow12] Hossein Jowhari, *Efficient communication protocols for deciding edit distance*, Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings, 2012, pp. 648–658.

[KJP77] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt, *Fast pattern matching in strings*, SIAM J. Comput. **6** (1977), no. 2, 323–350.

[KN97] Eyal Kushilevitz and Noam Nisan, *Communication complexity*, Cambridge University Press, New York, NY, USA, 1997.

[KN05] Subhash Khot and Assaf Naor, *Nonembeddability theorems via fourier analysis*, 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings, 2005, pp. 101–112.

[KOR98] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani, *Efficient search for approximate nearest neighbor in high dimensional spaces*, Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '98, ACM, 1998, pp. 614–623.

[KR06] Robert Krauthgamer and Yuval Rabani, *Improved lower bounds for embeddings into* L_1, Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006, 2006, pp. 1010–1017.

[Lev66] VI Levenshtein, *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*, Soviet Physics Doklady **10** (1966), 707.

[LMS98] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt, *Incremental string comparison*, SIAM J. Comput. **27** (1998), no. 2, 557–582.

[LPW06] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer, *Markov chains and mixing times*, American Mathematical Society, 2006.

[LRV09] Jeffrey C. Lagarias, Eric Rains, and Robert J. Vanderbei, *The kruskal count.*, pp. 371–391, Berlin: Springer, 2009.

[MP80] William J. Masek and Michael S. Paterson, *A faster algorithm computing string edit distances*, Journal of Computer and System Sciences **20** (1980), no. 1, 18 – 31.

[Nav01] Gonzalo Navarro, *A guided tour to approximate string matching*, ACM Comput. Surv. **33** (2001), no. 1, 31–88.

[Nis90] N. Nisan, *Pseudorandom generators for space-bounded computations*, Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '90, ACM, 1990, pp. 204–212.

[OR07] Rafail Ostrovsky and Yuval Rabani, *Low distortion embeddings for edit distance*, J. ACM **54** (2007), no. 5, 23+.

[Orl91] Alon Orlitsky, *Interactive communication: Balanced distributions, correlated files, and average-case complexity*, FOCS, IEEE Computer Society, 1991, pp. 228–238.

[PL07] Ely Porat and Ohad Lipsky, *Improved sketching of hamming distance with error correcting.*, CPM (Bin Ma and Kaizhong Zhang, eds.), Lecture Notes in Computer Science, vol. 4580, Springer, 2007, pp. 173–182.

[Sah14] Barna Saha, *The dyck language edit distance problem in near-linear time*, 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, 2014, pp. 611–620.

[Ukk85] Esko Ukkonen, *Algorithms for approximate string matching*, Inf. Control **64** (1985), no. 1-3, 100–118.

[WF74] Robert A. Wagner and Michael J. Fischer, *The string-to-string correction problem*, J. ACM **21** (1974), no. 1, 168–173.