Approximating Edit Distance Efficiently

Ziv Bar-Yossef* T. S. Jayram† Robert Krauthgamer† Ravi Kumar†

Abstract

Edit distance has been extensively studied for the past several years. Nevertheless, no linear-time algorithm is known to compute the edit distance between two strings, or even to approximate it to within a modest factor. Furthermore, for various natural algorithmic problems such as low-distortion embeddings into normed spaces, approximate nearest-neighbor schemes, and sketching algorithms, known results for the edit distance are rather weak.

We develop algorithms that solve gap versions of the edit distance problem: given two strings of length n with the promise that their edit distance is either at most k or greater than ℓ, decide which of the two holds.

We present two sketching algorithms for gap versions of edit distance. Our first algorithm solves the k vs. $(kn)^{2/3}$ gap problem, using a constant size sketch. A more involved algorithm solves the stronger k vs. ℓ gap problem, where ℓ can be as small as $O(k^2)$—still with a constant sketch—but works only for strings that are mildly "non-repetitive".

Finally, we develop an $n^{3/7}$-approximation quasi-linear time algorithm for edit distance, improving the previous best factor of $n^{3/4}$ [5]; if the input strings are assumed to be non-repetitive, then the approximation factor can be strengthened to $n^{1/3}$.

1. Introduction

A fundamental measure of similarity between strings is the *edit distance* (aka *Levenshtein distance*), which is the minimum number of character insertions, deletions, and substitutions needed to transform one string to the other. Edit distance is an important primitive with numerous applications in areas like computational biology and genomics, text processing, and web search; see, for instance, the books by Gusfield [9] and Pevzner [22]. Many of these application areas typically deal with large amounts of data—ranging from a moderate number of extremely long strings, as in computational biology, to a large number of moderately long strings, as in text processing and web search—and therefore algorithms for edit distance that are efficient in terms of time and/or space, even with modest approximation guarantees, are highly desirable. We present super-efficient algorithms for approximating the edit distance, focusing on two powerful notions of efficiency that are applicable in dealing with massive data, namely, sketching algorithms and linear-time algorithms.

Edit distance has been extensively studied for the past several years. An easy dynamic programming algorithm computes the edit distance in quadratic time [18, 21, 24] and the algorithm can be made to run in linear space [10]. However, the quadratic time algorithm for computing the edit distance was improved by only a logarithmic factor in [19], and even developing sub-quadratic time algorithms for approximating it within a modest factor has proved to be quite challenging, see [11, Section 6] and [13, Section 8.3.2].

We design very efficient algorithms for the k vs. ℓ gap version of the edit distance problem: given two n-bit input strings with the promise that the edit distance is either at most k or more than ℓ, decide which of the two cases holds. Such algorithms immediately yield approximation algorithms that are as efficient, with the approximation factor directly correlated with the gap between k and ℓ. Specifically, we design sketching algorithms and (quasi)-linear time algorithms for this gap problem. In addition to the inherent theoretical interest in these fundamental algorithmic questions, we believe that our efficient algorithms may find applications (as building blocks) in a multitude of scenarios with voluminous data.

1.1. Sketching algorithms

A sketching algorithm for edit distance consists of two *compression procedures* and a *reconstruction procedure*, which work in concert as follows. The compression procedures produce a fingerprint (*sketch*) from each of the input strings, and the reconstruction procedure uses solely the sketches to approximate the edit distance between the two strings. The key feature is that the sketch of each string is constructed without knowledge of the other string. The

*Department of Electrical Engineering, Technion, Haifa 32000, Israel. Email: zivby@ee.technion.ac.il. This work was done while the author was at the IBM Almaden Research Center.

†IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA. Email: {jayram,robi,ravi}@almaden.ibm.com.

sketches are supposed to retain the minimum amount of information about the strings that is required to subsequently approximate the edit distance. The procedures are allowed to share random coins, and the main measure of complexity is the size of the sketches produced. (In actual applications it is desirable that the procedures be efficient.)

In contrast to Hamming distance, whose sketching complexity is well-understood [17, 8], essentially nothing is known about sketching of edit distance. In part, this is due to the fact that edit distance does not correspond to a normed space. In fact, it is not even known whether the edit distance metric space embeds into some normed space with low distortion [11, 1]. We note that besides being a very basic computational primitive for massive data sets, sketching is also related to (i) approximate nearest neighbor algorithms [14, 17], (ii) protocols that are secure (i.e., leak no information), cf. [8], and (iii) the simultaneous messages communication model with public coins [25].

Results. Our first sketching algorithm solves the k vs. $O((kn)^{2/3})$ gap problem, for any $k \leq \sqrt{n}$. This algorithm is ultra-efficient in terms of sketch size—it is *constant*! This algorithm is extremely appealing in applications where one expects most pairs of strings to be either quite similar or very dissimilar, e.g., duplicate elimination or a preprocessing filter in text corpora or in computational biology.

Our second sketching algorithm can distinguish a smaller gap and still produces a constant-sized sketch, but it is guaranteed to work only if the input strings are "non-repetitive". Specifically, for any $k \leq \sqrt{n}$ and $t \geq 1$, if each of the length kt substrings of the inputs strings does not contain identical length t substrings, then the algorithm solves the k vs. $O(k^2 t)$ gap problem. We note that the study of algorithms for non-repetitive strings is quite standard (cf. [23, 5]) and has often led to comparable algorithms that work for arbitrary strings. Furthermore, input instances for the *Ulam metric*, which is equivalent to the edit distance on strings that consist of distinct characters (e.g., permutations of $\{1, \ldots, n\}$), are non-repetitive with $t = 1$.

Section 2 describes the efficient compression and reconstruction procedures used in these two sketching algorithms.

Techniques. The overall structure of the first sketching algorithm is an embedding of the original edit distance space into a Hamming space of low dimension. This embedding, which may be of independent interest, is achieved in two steps. First, we map each string to the multi-set of all its (overlapping) substrings. Each substring is annotated with a careful "encoding" of its position inside the input string. The encoding is insensitive to small "shifts", and is thus useful in identifying substrings that are matched by an optimal alignment of the two strings. In the second step, we take the characteristic vector of the resulting set of substrings, which lies in a Hamming space of an exponentially high dimension, and embed it in a Hamming space of constant dimension (a la [17]). The dependence on n in the gap in the first algorithm is a consequence of the encoding method for the position of a substring. In essence, for each substring we produce an independent encoding of its position; while this conveniently separates the analysis of different substrings, the outcome is that we fail to identify many matches, even in the presence of just one edit operation.

We overcome this handicap by resorting to a method in which the encodings of the substring positions are correlated. Scanning the input string from left to right, we iteratively locate *anchor* substrings—identical substrings that occur in the two input strings at approximately the same position. We map each string to the set of substrings corresponding to the regions between successive anchors; the anchors are used for encoding the substring positions. As before, the resulting set of substrings is used to obtain an embedding in a Hamming space of constant dimension. Random permutations of small size are used to ensure that anchors are detected with high probability. This places a technical requirement that the input strings cannot have identical substrings within the window where we might be looking for anchors, implying that the algorithm is applicable to non-repetitive strings only.

1.2. Quasi-linear time algorithms

As a first step towards the important goal of approximating edit distance to within a constant factor (in near-linear time), we propose to focus on the best approximation achievable by linear time algorithms. We say that an algorithm provides a ρ-approximation if it produces a number that is at least the edit distance but no more than ρ times the edit distance. Throughout, our time bounds refer to a RAM model with word size $O(\log n)$.

Results. We design a linear time algorithm that achieve approximation $\rho = n^{3/7}$, which improves to $\rho = n^{1/3}$ if the two strings are non-repetitive. The best approximation factor that could be achieved in quasi-linear time with previous techniques is $n^{3/4}$, by a straightforward application of an algorithm by Cole and Hariharan [5] (see below). These results are described in Section 3.

Techniques. We present a very general framework for taking an approximation for the edit pattern matching and boosting it to a *stronger* approximation for edit distance. Here, *edit pattern matching* is the problem of finding all approximate matches of a pattern of size m in a text of size n, where an approximate match of the pattern is a substring of the text whose edit distance to the pattern is at most k. We demonstrate three instances of this paradigm. First, a simple instantiation of this framework already provides an algorithm that solves the k vs. k^2 gap problem. This implies a $\sqrt{n}$-approximation algorithm for edit dis-

tance, while the approximation provided directly by the edit pattern matching primitive that we rely on is only n. Using a non-trivial edit pattern matching algorithm of Cole and Hariharan [5], our framework yields an enhanced algorithm that solves the k vs. $k^{7/4}$ gap problem, which implies the $n^{3/7}$-approximation claimed above. Under the assumption that the input strings are non-repetitive, the third instantiation solves the k vs. $k^{3/2}$ gap, giving an $n^{1/3}$-approximation.

1.3. Related work

To the best of our knowledge, sketching or quasi-linear time algorithms for gap versions of edit distance have not been explicitly studied before. Yet, some of the previous work can be easily adapted to give such algorithms.

Batu *et al.* [4] developed a sub-linear time algorithm that runs in $O(n^{\max(\alpha/2,2\alpha-1)})$ time and solves the $O(n^\alpha)$ vs. $\Omega(n)$ edit distance gap problem. Their algorithm can be cast as a sketching algorithm. On the one hand, their algorithm applies also for $\alpha > 1/2$, which our algorithm does not handle. On the other hand, their algorithm would use a sketch whose size is far more than constant; e.g., for $k = \sqrt{n}$ their sketch size would be about $n^{1/4}$ compared with our $O(1)$ sketch size (for the same gap problem). Furthermore, their algorithm cannot solve the n^δ vs. $n^{1-\delta}$ gap problem, even for arbitrarily small fixed $\delta > 0$, while we accomplish this for any $\delta \leq 1/5$. We note that their algorithm runs in sub-linear time, while ours does not.

The dynamic programming algorithm can solve the k vs. $k+1$ gap version of edit distance in $O(kn)$ time. An algorithm of Sahinalp and Vishkin [23] for the edit pattern matching problem can be used to solve the k vs. $2k$ gap problem in $O(n+k^8)$ time. A simpler algorithm of Cole and Hariharan [5] for edit pattern matching yields an $O(n+k^4)$ time algorithm for the same gap problem. This leads to the aforementioned $n^{3/4}$-approximation algorithm in linear time. In contrast, we have an algorithm that, for any $k \leq n^{4/7}$, solves the k vs. $k^{7/4}$ gap problem in $\tilde{O}(n)$ time, deriving an $n^{3/7}$-approximation in quasi-linear time.

Other related work includes a near-linear time deterministic algorithm of Cormode and Muthukrishnan [6] for a variant of edit distance called the *block edit distance*, where a block of characters can be moved in a single edit operation. Andoni *et al.* [1] showed that edit distance cannot be embedded into the Hamming space with distortion better than $3/2$; Cormode *et al.* [7, 6] and Muthukrishnan and Sahinalp [20] showed that the block edit distance can be embedded into Hamming space with distortion $O(\log n \log^* n)$. Lack of good sketching algorithms for edit distance is also reflected in a lack of good nearest-neighbor algorithms for edit distance, since efficient sketching primitives are at the heart of many approximate nearest-neighbor algorithms. Recently, Indyk [12] obtained an approximate nearest-neighbor algorithm for edit distance where the data structure size is strongly sub-exponential in n and the query time is asymptotically smaller than the number of database points.

1.4. Preliminaries

The goal of this paper is to design efficient algorithms for the k vs. ℓ gap version of edit distance. k is given as input parameter to the algorithm. The smaller the difference between k and $\ell = \ell(n,k)$, the better the approximation achievable from these algorithms. To simplify the exposition, we make no attempt to optimize constants.

Strings, alignments, and edit distance. We deal with strings over a finite alphabet Σ. For simplicity, most of our results are stated for Boolean strings (i.e., $\Sigma = \{0,1\}$). xy denotes the concatenation of two strings x and y. The empty string is denoted by ϵ. For integers i, j, the interval $[i\,..\,j]$ denotes the set of integers $\{i, \ldots, j\}$ (which is empty if $i > j$); $[i]$ is a shorthand for the interval $[1\,..\,i]$. Let $\mathrm{x} \in \Sigma^n$ be a string of *length* n. For $i \in [n]$, $\mathrm{x}(i)$ is the i-th character of x. Let $\mathrm{x}[i\,..\,j]$ denote the *substring* obtained by projecting x on the positions in the set $[i\,..\,j] \cap [n]$. If this set is empty, then $\mathrm{x}[i\,..\,j] = \epsilon$.

An *edit operation* on a string $\mathrm{x} \in \Sigma^n$ is either an insertion, a deletion, or a substitution of a character of x. We associate with each edit operation a position in the string x: a deletion and a substitution are associated with the position of the character being deleted or substituted, and an insertion is associated with the position of the character before which the new character is inserted (if the character is inserted after the last character of x, then we associate with the insertion the position $n+1$). An *alignment* of two strings $\mathrm{x}, \mathrm{y} \in \Sigma^n$ is a sequence of edit operations on x that transform x into y (we view the operations as operating directly on x and not on the intermediate strings obtained in the transformation). An *optimal alignment* is one that uses a minimum number of edit operations. The *edit distance* between x and y is the length of their optimal alignment. We note the following properties of the edit distance:

1. **Triangle inequality:** for any three strings $\mathrm{x}, \mathrm{y}, \mathrm{z}$, $\mathrm{ED}(\mathrm{x},\mathrm{y}) \leq \mathrm{ED}(\mathrm{x},\mathrm{z}) + \mathrm{ED}(\mathrm{z},\mathrm{y})$.
2. **Splitting inequality:** for strings x and y of lengths n and m, respectively, and any integers i, j, $\mathrm{ED}(\mathrm{x},\mathrm{y}) \leq \mathrm{ED}(\mathrm{x}[1\,..\,i],\mathrm{y}[1\,..\,j]) + \mathrm{ED}(\mathrm{x}[i+1\,..\,n],\mathrm{y}[j+1\,..\,m])$.

For an interval $[i\,..\,j]$, where $1 \leq i \leq j \leq n+1$, we say that an edit operation *belongs* to the interval $[i\,..\,j]$, if it is associated with one of the positions in the interval. Given an alignment τ of x and y, for each interval $[i\,..\,j]$, we define $\mathrm{ins}_\tau(i\,..\,j)$, $\mathrm{del}_\tau(i\,..\,j)$, and $\mathrm{sub}_\tau(i\,..\,j)$ to be the number of insertions, deletions, and substitutions, respectively, that

belong to the interval $[i\,..\,j]$. We define the *shift* at $[i\,..\,j]$ to be $\mathrm{sh}_\tau(i\,..\,j) = \mathrm{ins}_\tau(i\,..\,j) - \mathrm{del}_\tau(i\,..\,j)$; the shift of a position $i \in [n+1]$ is defined as $\mathrm{sh}_\tau(i) = \mathrm{sh}_\tau(1\,..\,i)$; we also define $\mathrm{sh}_\tau(0) = 0$. The *induced alignment* of τ on an interval $[i\,..\,j]$ is the subsequence of edit operations in τ that belong to $[i\,..\,j]$. We denote by $\mathrm{ed}_\tau(i\,..\,j)$ the size of the induced alignment. We note the following property of induced alignments:

Proposition 1.1. *For any alignment τ of* x *and* y *and for all $i \le j$,* $\mathrm{ed}_\tau(i\,..\,j) \ge \mathrm{ED}_\tau(\mathrm{x}[i\,..\,j], \mathrm{y}[i + \mathrm{sh}_\tau(i-1)\,..\,j + \mathrm{sh}_\tau(j)])$.

Definition 1.2 (Non-repetitive strings). A string $\mathrm{x} \in \{0,1\}^n$ is called (t,ℓ)*-non-repetitive*, if for any interval $[i\,..\,j]$ of size ℓ, the ℓ substrings of x of length t whose left endpoints are in this interval are distinct.

The sketching model. A sketching algorithm is best viewed as a two-party public-coin simultaneous messages communication complexity protocol. In this model three players, Alice, Bob, and a referee, jointly compute a two-argument function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$. Alice is given $x \in \mathcal{X}$ and Bob is given $y \in \mathcal{Y}$. Based on her input and based on randomness that is shared with Bob, Alice prepares a "sketch" $s_A(x)$ and sends it to the referee; similarly, Bob sends a sketch $s_B(y)$ to the referee. The referee uses the two sketches (and the shared randomness) to compute the value of the function $f(x,y)$. The main measure of cost of a sketching algorithm is the length of the sketches $s_A(x)$ and $s_B(y)$ on the worst-case choice of inputs x, y.

Throughout, we seek algorithms whose error probability is some small constant, say 1/3. As usual, this error can be reduced to any value $0 < \delta < 1$, using $O(\log(1/\delta))$ simultaneous repetitions.

In many applications, it is desirable that the three players are efficient (in time, space, etc.) We will say that a sketching algorithm is $t(n)$*-efficient*, if the running time of each of the three players is $O(t(n))$, where n is the size of the player's input (x for Alice, y for Bob, and $(s_A(x), s_B(y))$ for the referee).

2. Sketching algorithms for edit distance

Overview. In this section we describe our two sketching algorithms for solving gap edit distance problems. The underlying principle in both algorithms is the same: the two input strings have a small edit distance if and only if they share many sufficiently long substrings occurring at nearly the same position in both strings, and hence, the number of mismatching substrings provides an estimate of the edit distance. More formally, both algorithms map the inputs x and y into sets T_x and T_y, respectively; these sets consist of pairs of the form (γ, i), where γ is a sufficiently long substring and i is a special "encoding" of the position at which the substring begins. The encoding scheme has the property that nearby positions are likely to share the same encoding. A pair $(\gamma, i) \in T_x \cap T_y$ represents substrings of x and of y that *match*, i.e., they are identical (in terms of contents) and they occur at nearby positions in x and in y. A pair $(\gamma, i) \in (T_x \setminus T_y) \cup (T_y \setminus T_x)$ represents a substring that cannot be matched using a small number of edit operations. This gives rise to a natural reduction from the task of estimating edit distance between x and y to that of estimating the Hamming distance between the characteristic vectors u and v of T_x and T_y, respectively. (Recall that the Hamming distance between two strings $\mathrm{x}, \mathrm{y} \in \{0,1\}^n$ is defined as $\mathrm{HD}(\mathrm{x},\mathrm{y}) \stackrel{\mathrm{def}}{=} |\{i \in [n] : \mathrm{x}(i) \neq \mathrm{y}(i)\}|$.) The great advantage of the Hamming distance is that it can be approximated using constant-size sketches, as shown by Kushilevitz, Ostrovsky and Rabani [17].

The realizations of the above idea in the two algorithms are quite different, mainly due to the implementation of the "position encoding". The first algorithm works for arbitrary input strings. In this algorithm, T_x and T_y consist of all the (overlapping) substrings of a suitable length $B = B(n,k)$ of x and y, respectively. (Recall that n is the length of the input strings and k is the gap parameter.) The position of each substring is encoded by rounding the position down to the nearest multiple of an appropriately chosen integer $D = D(n,k)$. A tradeoff between B and D implies that the best choice of parameters is $B = \Theta(n^{2/3}/k^{1/3})$ and $D = n/B$, which results in an algorithm that can solve the k vs. kB gap edit distance problem.

The second algorithm, which works for mildly non-repetitive strings, introduces a more sophisticated "position encoding" method, based on selecting a set of "anchors" from x and from y in a coordinated way. Anchors are substrings that are unique within a certain window and appear in both x and y in that window. Suppose x and y have an alignment that uses only a small number of edit operations. Then, a sufficiently short substring chosen at random from any sufficiently long window in x is unlikely to contain any edit operation, and thus has to be matched with a corresponding substring in y within the same window. This pair of substrings form anchors. The key idea is that the coordinated selection of anchors can be done without Alice and Bob communicating with each other, but rather by using the shared random coins. Once this is done, the anchors induce a natural partitioning of x and y into disjoint substrings. T_x and T_y then consist of these substrings, with the position of each substring being encoded by the number of anchors that precede it. This technique solves much smaller (i.e. stronger) gap edit distance problems, in which the gap is independent of n.

A technical obstacle in both algorithms is that the Ham-

ming distance instances to which the problem is reduced are exponentially long. While this still leads to constant size sketches, the running time needed to produce these sketches may be prohibitive. We observe that the Hamming distance instances produced above are always of Hamming weight at most n. We introduce below a sketching method that approximates the Hamming distance within the same guarantees as [17], but runs in time proportional to the Hamming weight of the strings. This scheme may be of independent interest. Due to lack of space, the proof is defered to the full version of the paper.

Lemma 2.1. *For any $\varepsilon > 0$ and $k = k(n)$, there is an efficient sketching algorithm that solves the k vs. $(1+\varepsilon)k$ gap Hamming distance problem in binary strings of length n, with a sketch of size $O(1/\varepsilon^2)$. If the set of non-zero coordinates of each input string can be computed in time t, then Alice and Bob run in $O(\varepsilon^{-3} t \log n)$ time.*

Note that the running time of Alice and Bob in the KOR algorithm [17] is $O(\varepsilon^{-2} n)$.

2.1. Sketching algorithm for arbitrary strings

Theorem 2.2. *For any $0 \leq k < \sqrt{n}$, there exists a quasi-linear time sketching algorithm that solves the k vs. $\Omega((kn)^{2/3})$ gap edit distance problem using sketches of size $O(1)$.*

Proof. The algorithm follows the general scheme described in the overview above. We are thus left to formally describe how the sets T_x and T_y are constructed. For simplicity of exposition, we assume n and k are powers of 2 with an exponent that is a multiple of 3. We describe now how Alice creates the set T_x. Bob's algorithm is analogous. Let $B = n^{2/3}/(2k^{1/3})$ and let $D = n/B$. For each position $i \in [n]$, let $\text{DIV}(i) \stackrel{\text{def}}{=} \lfloor i/D \rfloor$ (which is proportional to the largest multiple of D that is at most i). T_x is the set of pairs $(\mathrm{x}[i \,..\, i+B-1], \text{DIV}(i))$ for $i = 1, \ldots, n-B+1$.

The Hamming distance sketch of the vectors u and v (recall these are the charateristic vectors of T_x and T_y, respectively) is tuned to determine whether $\text{HD}(\mathrm{u}, \mathrm{v}) \leq 4kB$ or $\text{HD}(\mathrm{u}, \mathrm{v}) > 8kB$ with (large) constant probability of error. The referee, upon receiving the sketches from Alice and Bob, decides that $\text{ED}(\mathrm{x}, \mathrm{y}) \leq k$ if he finds that $\text{HD}(\mathrm{u}, \mathrm{v}) \leq 4kB$. Otherwise, he decides that $\text{ED}(\mathrm{x}, \mathrm{y}) \geq 13(kn)^{2/3}$.

The algorithm's correctness follows immediately from Lemmas 2.3 and 2.4 below, using the sketching algorithm for Hamming distance from Lemma 2.1. □

Lemma 2.3. *If $\text{ED}(\mathrm{x}, \mathrm{y}) \leq k$, then $\text{HD}(\mathrm{u}, \mathrm{v}) \leq 4kB$.*

Proof. Fix any alignment τ of x and y of length at most k. For each $i = 1, \ldots, n-B+1$, let $\alpha_i = \mathrm{x}[i \,..\, i+B-1]$. We call a substring α_i "bad", if $\text{ed}_\tau(i \,..\, i+B-1) > 0$. (See Section 1.4 for definition.) All the substrings that are not bad are called "good". By Proposition 1.1, for any good substring α_i there is a "companion" substring $\beta_i = \mathrm{y}[(i + \text{sh}_\tau(i-1)) \,..\, (i+B-1+\text{sh}_\tau(i+B+1))]$ in y, so that $\alpha_i = \beta_i$.

Recall that coordinates of u, v are associated with pairs of the form (γ, j), where γ is a bitstring of length B and j is an integer between 0 and $\frac{n}{D} - 1$. Let us upper bound the number of coordinates in which we have 1 in u but 0 in v. Each such coordinate (γ, j) corresponds to a unique $i \in [n-B+1]$ such that $\alpha_i = \gamma$ and $\text{DIV}(i) = j$. Furthermore, it must be the case that either (1) α_i is a bad substring; or (2) α_i is a good substring, but its companion string $\beta_i = \mathrm{y}[i' \,..\, i'+B-1]$ is such that $\text{DIV}(i') \neq j$.

It therefore suffices to upper bound the number of positions i in which (1) and (2) are satisfied. Clearly, the number of bad substrings α_i is at most kB, because every edit operation is contained in at most B different substrings. A good substring α_i can have a companion β_i with $\text{DIV}(i + \text{sh}_\tau(i-1)) \neq \text{DIV}(i)$ only if i belongs to an interval $[tD - k \,..\, tD + k - 1]$ "centered" at some multiple tD of D, because by definition $-k \leq \text{sh}_\tau(i-1) \leq k$ (recall that τ consists of at most k edit operations). Hence, the total number of such positions i is at most $2k \cdot n/D$. A more careful analysis slightly improves this bound to k per interval. Indeed, suppose (2) happens for two values $i_1, i_2 \in [tD - k \,..\, tD + k - 1]$ with $i_1 < tD \leq i_2$ (otherwise we are done); then $i_1 + \text{sh}_\tau(i_1 - 1) \geq tD$ and $i_2 + \text{sh}_\tau(i_2 - 1) < tD$, hence $\text{sh}_\tau(i_1 - 1) - \text{sh}_\tau(i_2 - 1) > (tD - i_1) + (i_2 - tD) = i_2 - i_1$, and since the lefthand side is clearly at most k, the size of the interval $[i_2 \,..\, i_1]$ is upper bounded by k.

We conclude that (2) is satisfied at most $k \cdot n/D = kB$ times, and therefore the number of coordinates in which u is 1 and v is 0 is at most $2kB$. The number of coordinates where v is 1 and u is 0 is bounded similarly, which gives $\text{HD}(\mathrm{u}, \mathrm{v}) \leq 4kB$. □

Lemma 2.4. *If $\text{ED}(\mathrm{x}, \mathrm{y}) \geq 13(kn)^{2/3}$, then $\text{HD}(\mathbf{u}, \mathbf{v}) \geq 8kB$.*

Proof. Assume for contradiction that $\text{HD}(\mathbf{u}, \mathbf{v}) < 8kB$. We will show that it implies $\text{ED}(\mathrm{x}, \mathrm{y}) < 13(kn)^{2/3}$.

For each $j = 1, \ldots, n-B+1$, let $\alpha_j = \mathrm{x}[j \,..\, j+B-1]$. We call a substring α_j "good", if there exists a "companion" substring $\beta_{j'} = \mathrm{y}[j' \,..\, j'+B-1]$ such that $\alpha_j = \beta_{j'}$ and $\text{DIV}(j) = \text{DIV}(j')$. Otherwise, α_j is called "bad". If α_j is bad, then the coordinate corresponding to the pair $(\gamma_j, \text{DIV}(j))$ has value 1 in $\mathbf{u}$ and 0 in $\mathbf{v}$. Since $\text{HD}(\mathbf{u}, \mathbf{v}) < 8kB$, the number of bad strings is less than $8kB$.

We use the good substrings to align x and y, by iteratively extending an alignment of prefixes of x and y. The initial alignment is trivial since both prefixes are the empty string.

Assume now we already aligned the first $j-1$ bits of x and of y, and let us extend the alignment to a longer prefix. If the substring α_j is bad, we simply extend the current alignment by one bit, paying one edit operation for the substitution of $x(j)$ with $y(j)$. If $\alpha_j = \mathrm{x}[j\,..\,j+B-1]$ is good, we extend the alignment by B bits, using its companion string $\beta_{j'} = \mathrm{y}[j'\,..\,j'+B-1]$ as much as possible. Observe that we can align $\mathrm{x}[j\,..\,j+B-1]$ with $\mathrm{y}[j\,..\,j+B-1]$ using at most $2|j-j'|$ edit operations. If $j' \geq j$, we transform $\mathrm{x}[j\,..\,j+B-1]$ into $\mathrm{y}[j\,..\,j+B-1]$) by inserting before its beginning the first $j'-j$ bits of $\mathrm{y}[j\,..\,j+B-1]$ and deleting from it the last $j'-j$ bits. If $j' < j$, the operations are analogous. In either case, we pay at most $2|j'-j|$ edit operations. The key point is that $\textsc{div}(j) = \textsc{div}(j')$ and hence $|j-j'| < D$.

Finally, once we get to $j \geq n-B+1$, i.e., we aligned more than $n-B$ bits, we just pay $n-(j-1)$ edit operations to substitute the $n-(j-1)$ last characters of x with those of y.

It remains to bound the total cost of this alignment. Since we can encounter each bad substring at most once, we pay a total of at most $8kB$ edit operations for all the steps involving a bad substring. Similarly, we pay at most B edit operations for the final stage. All the remaining operations use good strings. Each such step pays at most $2D$ operations each time, but aligns B bits, and hence there are at most n/B such steps. We conclude that

$$\mathrm{ED}(\mathrm{x},\mathrm{y}) \leq 8kB + B + 2D \cdot \frac{n}{B} < 13(kn)^{2/3}. \quad \square$$

2.2. Sketching algorithm for non-repetitive strings

Theorem 2.5. *For any $1 \leq t < n$ and for any $1 \leq k < O(\sqrt{n/t})$, there exists a polynomial-time efficient sketching algorithm that solves the k vs. $\Omega(tk^2)$ gap edit distance problem for (t, tk)-non-repetitive strings using sketches of size $O(1)$.*

Proof. Again, the algorithm uses the general framework described in the overview. We are left to specify how the sets T_x and T_y are constructed. Let $\mathrm{x},\mathrm{y} \in \{0,1\}^n$ be two (t, tk)-non-repetitive input strings (see Section 1.4). Alice creates the set T_x as follows; Bob's algorithm is similar. First, she uses the shared randomness to compute a Karp–Rabin fingerprint [16] of size $O(\log n)$ for every substring of x of length t. This can be done in $O(n)$ time. We let $f(\cdot)$ denote the chosen fingerprint function. Let $\lambda > 0$ be a sufficiently large constant that will be determined later.

Next, Alice selects a sequence of disjoint substrings $\alpha_1, \ldots, \alpha_{r_x}$ of x, called "anchors", iteratively as follows. She maintains a sliding window of length $W \stackrel{\text{def}}{=} \lambda tk$ over her string. Let c denote the left endpoint of the sliding window; initially, c is set to 1. At the i-th step, Alice considers the W substrings of length t whose starting position lies in the interval $[c+W\,..\,c+2W-1]$. For $j = 1, \ldots, W$, let $s_{i,j} = \mathrm{x}[c+j+W-1\,..\,c+j+W+t-2]$ be the j-th substring. Using the shared randomness, Alice picks a random permutation Π_i on the space $\{0,1\}^{O(\log n)}$, and sets the anchor α_i to be a substring $s_{i,\ell}$ whose fingerprint is minimal according to Π_i, i.e.,

$$\Pi_i(f(s_{i,\ell})) = \min\{\Pi_i(f(s_{i,1})), \ldots, \Pi_i(f(s_{i,W}))\}.$$

She then slides the window by setting c to the position immediately following the anchor, i.e., $c \leftarrow c+\ell+W-1+t$. If this new value of c is at most $n-(2W+t)$, Alice starts a new iteration. Otherwise, she stops, letting r_x be the number of anchors she collected.

For $i \in [r_x]$, let ϕ_i be the substring starting at the position immediately after the last character of anchor α_{i-1} and ending at the last character of α_i. For this definition to make sense for $i = 1$, define α_0 to be the empty string, and consider it as if it is located at position 0, hence ϕ_1 starts at position 1. Finally, T_x is the set of pairs (ϕ_i, i) for all $i \in [r_x]$.

Bob constructs T_y analogously, by choosing anchors $\beta_1, \ldots, \beta_{r_y}$ using the same random permutations Π_i. The Hamming distance sketch for the strings u, v (the incidence vectors of T_x, T_y) is tuned to solve the $3k$ vs. $6k$ gap Hamming distance problem with probability of error at most $1/12$. The referee, upon receiving the two sketches, decides that $\mathrm{ED}(\mathrm{x},\mathrm{y}) \leq k$ if he finds that $\mathrm{HD}(\mathrm{u},\mathrm{v}) \leq 3k$, and decides that $\mathrm{ED}(\mathrm{x},\mathrm{y}) > \Omega(tk^2)$ otherwise.

The algorithm's correctness follows immediately from Lemmas 2.6 and 2.8 below, using the sketching algorithm for Hamming distance from Lemma 2.1. $\square$

Lemma 2.6. *If $\mathrm{ED}(\mathrm{x},\mathrm{y}) \leq k$, then $\Pr[\mathrm{HD}(\mathrm{u},\mathrm{v}) \leq 3k] \geq 5/6$.*

Proof. Fix any alignment τ of x and y that uses at most k edit operations. We will say that two substrings $\mathrm{x}[i\,..\,j]$ and $\mathrm{y}[(i+\mathrm{sh}_\tau(i-1))\,..\,(j+\mathrm{sh}_\tau(j))]$ are "perfectly matched" by the alignment, if $\mathrm{ed}_\tau(i\,..\,j) = 0$. We slightly abused notation here by using in this definition not only the "contents" of the two substrings, but also their position in x, y. By Proposition 1.1, perfectly matched substrings must be identical. Note that the probability that any two of the $2n$ Karp–Rabin fingerprints computed by Alice and Bob collide is $o(1)$. It therefore suffices to assume that there are no collisions and prove that the statement in the lemma holds with probability $6/7$.

Letting $r = \min\{r_x, r_y\}$, we aim to show that with high probability, for all $i \in [r]$ the anchors α_i and β_i are perfectly matched. For $i \in r_x$, let c_i be Alice's value of c at the end of iteration i, and let $c_0 = 1$ be the initial value of c. It follows that $\alpha_i = \mathrm{x}[c_i - t, c_i - 1]$. Let d_i be similarly for Bob, hence $\beta_i = \mathrm{y}[d_i - t, d_i - 1]$.

The key ingredient is the "inductive" step provided by the next claim. For $i \geq 1$, let $\mathcal{E}_i$ be the event that α_i and β_i do not perfectly match and $i \leq r$. For consistency, let $\mathcal{E}_0$ be the event $\alpha_0 \neq \beta_0$ (which is empty by definition). Let $m_i = \mathrm{ed}_\tau(c_i \,..\, (c_i + 2W + t - 2))$.

Claim 2.7. *Then for every $i \geq 0$,*

$$\Pr[\mathcal{E}_{i+1} | \bar{\mathcal{E}}_i] \leq 4tm_{i+1}/W.$$

Proof. Fix $i \geq 0$. We may assume $i \leq r$, as otherwise we're done. Suppose $\bar{\mathcal{E}}_i$ holds. If $i > 0$, then $\alpha_i = \mathrm{x}[c_{i-1} - t \,..\, c_i - 1]$ and $\beta_i = \mathrm{y}[d_i - t \,..\, d_i - 1]$ are perfectly matched, hence position $c_i - 1$ in x is aligned with position $d_i - 1$ in y, i.e., $d_i - 1 = c_i - 1 + \mathrm{sh}_\tau(c_i - 1)$.

Let A be the set of length t substrings of x whose first character is in the interval $[(c_i + W) \,..\, (c_i + 2W - 1)]$. Similarly, let B be the set of length t substrings of y whose first character is in the interval $[(d_i + W) \,..\, (d_i + 2W - 1)]$. Since x and y are non-repetitive, $|A| = |B| = W$.

Let $A' \subseteq A$ be the substrings in A that are perfectly matched with substrings in B. Similarly, let $B' \subseteq$ be the substrings in B that are perfectly matched with substrings in A. Since perfectly matched substrings are identical, $A' = B' \subseteq A \cap B$. We will upper bound $|A \setminus A'|$.

First, we argue that at most tm_i substrings in A are not perfectly matched at all (i.e., to any substring in y). Indeed, such substrings must contain an edit operation, and belong to the interval $[(c_i + W) \,..\, (c_i + 2W + t - 2)]$ in x, but this interval contains only m_i edit operations, and each operation appears in at most t substrings in A. Next, we argue that at most m_i substrings in A are perfectly matched to a substring in y that is not in B. Indeed, position $c_i - 1$ in x is aligned with position $d_i - 1$ in y (for $i = 0$ we have instead $c_0 = 1 = d_0$), and since there are at most m_i insert operations in $\mathrm{x}[c_i \,..\, (c_i + 2W - 1)]$, only the m_i substrings in A with largest starting point might fall into this category. Combining the two, we have that $|A \setminus A'| \leq (t + 1)m_i$.

Recall that Alice and Bob choose their anchors from A and B, respectively, using a min-wise permutation (of the fingerprints). Since there are no collisions among the fingerprints, the minimum among the fingerprints is attained uniquely. Consider the string in $A \cup B$ whose fingerprint attains the minimum according to the permutation Π_i used by Alice and Bob. Noting that $|A| = |B|$ and $A' = B'$ implies $|A \setminus A'| = |B \setminus B'|$, we get that the probability this minimum string does not belong to $A' = B'$ is at most

$$\frac{|(A \setminus A') \cup (B \setminus B')|}{|A \cup B|} \leq \frac{2|A \setminus A'|}{|A|} \leq \frac{4tm_i}{W}.$$

The claim follows by observing that if the minimum string belongs to $A' = B' \subseteq A \cap B$ then Alice's and Bob's anchors are equal, $\alpha_{i+1} = \beta_{i+1}$, and since the substrings in A and the substrings in B are distinct, this means that the two anchors are perfectly matched and $\mathcal{E}_{i+1}$ does not occur. □

The anchor selection process fails, if at some iteration $i \leq r$, the anchors α_i and β_i do not perfectly match. WLOG, let i be the first such iteration. Necessarily, $i > 0$, because the anchors α_0, β_0 trivially match. Thus, if the process fails, there is some $i > 0$ so that the event $\mathcal{E}_i \cap \bar{\mathcal{E}}_{i-1}$ holds. Therefore, by the union bound, the probability of failure is at most

$$\Pr[\cup_{i \geq 1}(\mathcal{E}_i \cap \bar{\mathcal{E}}_{i-1})] \leq \sum_{i \geq 1} \Pr[\mathcal{E}_i | \bar{\mathcal{E}}_{i-1}] \leq \frac{4t}{W} \sum_{i=1}^{r} m_i.$$

Any position $i \in [n]$ is contained in at most two intervals of the form $[(c_i + W) \,..\, (c_i + 2W + t - 2)]$, simply because every iteration increases c by at least $W + t > \frac{1}{2}(2W + t - 1)$. Therefore, $\sum_{i=1}^{r} m_i \leq 2k$, implying that the above probability is at most $8tk/W$. Choosing a constant $\lambda \geq 56$, this probability is at most $1/7$.

Assume then that all the first r anchors are perfectly matched. Let $\phi_1, \ldots, \phi_{r_x}$ be the substrings used to create T_x and let $\psi_1, \ldots, \psi_{r_y}$ be the substrings used to create T_y. It is easy to verify that for all $i \in [r]$, since the anchors before ϕ_i and ψ_i perfectly match and also the anchors after ϕ_i and ψ_i perfectly match, the only way for $\phi_i \neq \psi_i$ is that ϕ_i contains edit operations. Since the substrings ϕ_i are disjoint, this can happen for at most k strings ϕ_i, and hence also for at most k strings ψ_i, contributing at most $2k$ to $\mathrm{HD}(\mathrm{u}, \mathrm{v})$. It easy to verify that by our definition of r_x and r_y, if α_r and β_r perfectly match, then $\max\{r_x, r_y\} \leq r + 1$. Thus, the extra substring in x or in y can contribute an additional one to $\mathrm{HD}(\mathrm{u}, \mathrm{v})$. We conclude that $\mathrm{HD}(\mathrm{u}, \mathrm{v}) \leq 2k+1 < 3k$. □

Lemma 2.8. *If* $\mathrm{HD}(\mathrm{u}, \mathrm{v}) \leq 6k$*, then* $\mathrm{ED}(\mathrm{x}, \mathrm{y}) \leq O(tk^2)$.

Proof. Let $\phi_1, \ldots, \phi_{r_x}$ be the substrings Alice used to create u and let $\psi_1, \ldots, \psi_{r_y}$ be similarly for v. Let $r = \max\{r_x, r_y\}$. For $i = r_x + 1, \ldots, r$ let $\phi_i = \epsilon$ be the empty string and similarly for $i = r_y + 1, \ldots, r$ let $\psi_i = \epsilon$. Since $\mathrm{HD}(\mathrm{u}, \mathrm{v}) \leq 6k$, we know that there are at most $6k$ values $i \in [r]$ for which $\phi_i \neq \psi_i$. For every such i we have $\mathrm{ED}(\phi_i, \psi_i) \leq 2W + t$, since the length of ϕ_i and of ψ_i is less than $2W + t$. For the remaining i's, with $\phi_i = \psi_i$, clearly $\mathrm{ED}(\phi_i, \psi_i) = 0$. Recall that the strings ϕ_i form a partition of x, except possibly for the last $2W + t$ or less characters, and similarly ψ_i for y. Therefore, we get as desired

$$\mathrm{ED}(\mathrm{x}, \mathrm{y}) \leq \sum_{i=1}^{r} \mathrm{ED}(\phi_i, \psi_i) \leq (6k+1) \cdot (2W+t) = O(tk^2)$$

by using the Splitting inequality (Section 1.4). □

3. Algorithms for approximating the edit distance

Overview. In this section, we develop quasi-linear time algorithms for edit distance gap problems. The *edit graph* G_E is a well-known representation of the edit distance by means of a directed graph (cf. [9]). In essence, a source-to-sink shortest path in G_E is equivalent to the natural dynamic programming algorithm. We will define a graph G which can be viewed as a lossy compression of G_E—the shortest path in G provides an approximation to the edit distance. Each edge in G will correspond to edit distance between substrings, unlike in G_E where each edge corresponds to at most a single edit operation. The advantage of G is its structure that allows to speed up the shortest path computation by handling multiple edges simultaneously. The latter turns out to be essentially an instance of the edit pattern matching problem.

The graph G is defined as follows. Let B be a parameter that will determine the size of substrings used in the algorithm; assume that B divides n. Each vertex in G corresponds to a pair (i, s) where $i = jB$, for some $j \in [0\,..\,n/B]$ and $s \in [-k\,..\,k]$; this vertex is closely related to the edit distance between the substrings $\mathrm{x}[1\,..\,i]$ and $\mathrm{y}[1\,..\,i+s]$ (s denotes the amount by which we extend/diminish y with respect to x). There is a directed edge e from (i', s') to (i, s) if and only if either (1) $i' = i$ and $|s' - s| = 1$, or (2) $i' = i - B$ and $s' = s$. The edge e has an associated weight $w(e)$ which equals 1 if $i' = i$ and $|s' - s| = 1$. For the other case when $i' = i - B$ and $s' = s$, we will allow some flexibility in setting the value of $w(e)$. In particular, given an approximation parameter c, then $w(e)$ can be any value such that $w(e)/c \leq \mathrm{ED}(\mathrm{x}[i' + 1\,..\,i], \mathrm{y}[i' + 1 + s\,..\,i + s]) \leq w(e)$. We will deal with the issue of computing such weights during the development of our algorithms.

For any path P in G, let the weight $w(P)$ of the path P equal the sum of the weights of the edges in P. Let T equal the weight of the shortest path from $(0, 0)$ to $(n, 0)$. The following two lemmas show that the value of T can be used to solve the k vs. ℓ edit distance gap problem for a suitable $\ell = \ell(k, c)$.

Lemma 3.1. $T \geq \mathrm{ED}(\mathrm{x}, \mathrm{y})$.

Proof (Sketch). We first claim that $w(P) \geq \mathrm{ED}(\mathrm{x}[1\,..\,i], \mathrm{y}[1\,..\,i + s])$ for any path P from $(0, 0)$ to (i, s). In particular, we show that $\mathrm{x}[1\,..\,i]$ can be transformed to $\mathrm{y}[1\,..\,i + s]$ by a sequence of edit operations corresponding to the sequence of edges in P; the cost of each operation is at most the weight of the corresponding edge. The details are given in the full version of the paper. □

Lemma 3.2. *If* $\mathrm{ED}(\mathrm{x}, \mathrm{y}) \leq k$*, then* $T \leq (2c + 2)k$.

Proof. Consider an optimal alignment τ using at most k edit operations on x. This implies that $|\mathrm{sh}_\tau(i)| \leq k$ for every i. We claim that for every i, there is a path from $(0, 0)$ to $(i, \mathrm{sh}_\tau(i))$ of weight at most $(2c+1) \cdot \mathrm{ed}_\tau(1\,..\,i)$. Applying this claim with $i = n$, we obtain a path from $(0, 0)$ to $(n, \mathrm{sh}_\tau(n))$ whose weight is at most $(2c+1) \cdot \mathrm{ed}_\tau(1\,..\,n) \leq (2c + 1)k$. Extending this path to $(n, 0)$ using an additional weight of at most k, it follows that $T \leq (2c + 2)k$, as required.

It remains to prove the claim, which we will prove by induction on the legal values of i. For $i = 0$, the claim is trivial since $\mathrm{sh}_\tau(0) = 0$. Assume the claim is true for i and let's show it is true for $i + B$. To ease the presentation, let $r = \mathrm{sh}_\tau(i)$ and let $s = \mathrm{sh}_\tau(i + B)$. By the induction hypothesis, there is a path P' from $(0, 0)$ to (i, r) such that $w(P') \leq (2c + 1) \cdot \mathrm{ed}_\tau(1\,..\,i)$.

Now define the path P'' from (i, r) to $(i + B, s)$ by first traversing the edge e from (i, r) to $(i + B, r)$ and then using the path from $(i + B, r)$ to $(i + B, s)$. To bound $w(P'') = w(e) + |r - s|$, we introduce some notation. Let $\alpha = \mathrm{x}[i+1\,..\,i+B]$, $\beta = \mathrm{y}[i+1+r\,..\,i+B+r]$ and $\gamma = \mathrm{y}[i+1+r\,..\,i+B+s]$. By definition, $w(e) \leq c \cdot \mathrm{ED}(\alpha, \beta)$. Using the triangle inequality, $\mathrm{ED}(\alpha, \beta) \leq \mathrm{ED}(\alpha, \gamma) + \mathrm{ED}(\gamma, \beta)$. Observe that $\mathrm{ED}(\alpha, \gamma) \leq \mathrm{ed}_\tau(i + 1\,..\,i + B)$ via Proposition 1.1. Since one of the strings γ, β is a prefix of the other, we have $\mathrm{ED}(\gamma, \beta) \leq |r - s|$. Putting these observations together gives $w(P'') \leq c \cdot \mathrm{ed}_\tau(i+1\,..\,i+B) + (c+1)|r - s|$. Since $|r - s| = |\mathrm{sh}_\tau(i) - \mathrm{sh}_\tau(i+B)| \leq \mathrm{ed}_\tau(i+1\,..\,i+B)$, it follows that $w(P'') \leq (2c + 1) \cdot \mathrm{ed}_\tau(i + 1\,..\,i + B)$.

Let P denote the concatenation of P' with P''. By the derivation above, $w(P) = w(P') + w(P'') \leq (2c + 1) \cdot [\mathrm{ed}_\tau(1\,..\,i) + \mathrm{ed}_\tau(i+1\,..\,i+B)] = (2c+1) \cdot \mathrm{ed}_\tau(1\,..\,i+B)$, so P satisfies the induction step for $i + B$. This completes the proof of the lemma. □

It remains to show how to compute the shortest path in G from $(0, 0)$ to $(n, 0)$ efficiently. Fix an i and consider the set of edges from (i, s) to $(i + B, s)$ for all s. These represent the approximate edit distances between $\mathrm{x}[i+1\,..\,i+B]$ and every substring of $\mathrm{y}[i + 1 - k\,..\,i + B + k]$ of length B. If we can somehow simultaneously compute all these weights efficiently, then it is conceivable that the shortest path algorithm can also be implemented efficiently. This is formalized as a separate problem below:

Definition 3.3 (Edit pattern matching). Given a pattern string P of length p and a text string T of length $t \geq p$, the *$c(p, t)$-edit pattern matching problem*, for some $c = c(p, t) \geq 1$, is to produce numbers $d_1, d_2, \ldots, d_{t-p+1}$ such that $d_i/c \leq \mathrm{ED}(\mathrm{P}, \mathrm{T}[i\,..\,i + p - 1]) \leq d_i$ for all i.

Theorem 3.4. *Suppose there is an algorithm that can solve the $c(p, t)$-edit pattern matching problem in time* $\mathrm{TIME}(p, t)$*. Then, given two strings* x *and* y *of length* n*, and*

the corresponding graph G with parameter B, the shortest path in the graph G can be used to solve the k versus $(2c(B, B + 2k) + 2)k$ edit distance gap problem in time $O((k + \text{TIME}(B, B + 2k))n/B)$

Proof (Sketch). The correctness follows from Lemmas 3.1 and 3.2. Our implementation of the shortest path algorithm proceeds in stages where the i-th stage computes the distance $T(i, s)$ from $(0, 0)$ to (i, s) simultaneously for all s. The key idea is to reduce this problem to computing single-source shortest paths on a graph with $O(k)$ edges. Assume that $T(i - B, s)$ has been computed for all values of s. We will show how to compute $T(i, s)$ for all s in time $O(k + \text{TIME}(B, B + 2k))$; the claim on the overall running time of the algorithm follows easily. Note that any shortest path to (i, s) consists of a shortest path from $(0, 0)$ to $(i-B, s')$, for some s', followed by the edge from $(i-B, s')$ to (i, s'), and then followed by the path from (i, s') to (i, s). Consider the following graph H of at most $2k + 2$ nodes with a start node u and a node v_s for every $s \in [-k, k]$. There is an edge between v_s and v_r with weight 1 if and only if $|s-r| = 1$; there is an edge from u to v_s with weight $T(i - B, s) + w((i - B, s), (i, s))$. This graph can be constructed in time $O(k+\text{TIME}(B, B+2k))$. It can be verified that the shortest path from u to v_s equals $T(i, s)$. This can be implemented using Dijkstra's shortest path algorithm in time $O(k \log k)$. A direct implementation is also possible by sorting the edges from u to v_s in non-decreasing order of weight; the values $T(i, s)$ can be calculated by carefully eliminating the edges, each one in $O(1)$ time. □

As an application of the theorem, suppose we run a pattern matching algorithm and output $d_i = 0$ if $\mathrm{P} = \mathrm{T}[i\,..\,i + p - 1]$ and $d_i = p$ otherwise; thus, $c(p, t) = p$. By precomputing the Karp–Rabin fingerprints of all blocks of length B in x and y in time $O(n)$, we obtain an algorithm for edit pattern matching that runs in time $O(k)$.

Theorem 3.5. *There is an algorithm for the k vs. $(2B + 2)k$ edit distance gap problem that runs in time $O(kn/B + n)$. In particular, there is a quasi-linear-time algorithm to distinguish between k and $O(k^2)$.*

For the second application, we apply the algorithm of Cole and Hariharan [5] for edit pattern matching. Here, given a parameter k, the goal is to output for each $i \in [1\,..\,t-p+1]$ whether there is a substring $\mathrm{T}[i\,..\,j]$, for some j, such that $\mathrm{ED}(\mathrm{P}, \mathrm{T}[i\,..\,j])$ is at most k. The algorithm in [5] runs in time $O(k^4 \cdot t/p + t + p)$. Their algorithm can be easily modified to obtain a quasi-linear time algorithm for edit pattern matching whose approximation parameter is $c = p^{3/4}$. Applying Theorem 3.4 with $B = k$, we get:

Theorem 3.6. *The k versus $k^{7/4}$ edit distance gap problem can be solved in quasi-linear-time.*

For non-repetitive strings, we can get a stronger $\sqrt{p}$-approximation algorithm for the edit pattern matching problem that runs in quasi-linear-time. Details are given in the full version of the paper. Now Theorem 3.4 (with $B = k$) implies the following:

Theorem 3.7. *The k vs. $k^{3/2}$ edit distance gap problem can be solved in quasi-linear-time if at least one of the pair of input strings is $(k, O(\sqrt{k}))$-non-repetitive.*

It is easy to see that Theorems 3.6 and 3.7 yield approximation algorithms for edit distance with factors $n^{3/7}$ and $n^{1/3}$, respectively.

4. Discussion

We turn our attention to lower bounds for edit distance in the sketching model. Lower bounds on sketch size are usually obtained via randomized communication complexity lower bounds in the public-coin simultaneous messages model [25]. A communication model that is closely related to the simultaneous messages model is the *one-way* model. In the terminology of Section 2, the one-way model is the same as the simultaneous one, except that Bob himself acts as the referee. For a Boolean function f, let $R^{\|}(f)$ (resp., $R^{\rightarrow}(f)$) denote the randomized simultaneous (resp., one-way) communication of f. By definition, $R^{\|}(f) \geq R^{\rightarrow}(f)$. In fact, most known lower bounds for sketching algorithms (i.e., randomized simultaneous model) hold also for the one-way model; the only known exception is the generalized addressing function [2, 3]. There are no general purpose lower bound techniques for the simultaneous messages model with public coins. For the remainder of the section, we use $\mathrm{ED}_{k,\ell}$ (resp., $\mathrm{HD}_{k,\ell}$) to denote the k vs. ℓ gap version of edit (resp., Hamming) distance problem.

In the one-way model, it is straightforward to obtain lower bounds for edit distance by exploiting its connection to the Hamming distance. In particular, we can show that for $k \leq n^{1/2}/2$, $R^{\rightarrow}(\mathrm{ED}_{k,k+1}) \geq R^{\rightarrow}(\mathrm{HD}_{k,k+1}) = \Omega(k)$. Indeed, we reduce Hamming distance to edit distance; letting $\sigma = 0^k$, Alice transforms her input to $\mathrm{x}' = x_1\sigma x_2 \cdots \sigma x_n$ and Bob transforms his input to $\mathrm{y}' = y_1\sigma y_2 \cdots \sigma y_n$. It is easy to see that if $\mathrm{HD}(\mathrm{x}, \mathrm{y}) \leq k$ then $\mathrm{ED}(\mathrm{x}', \mathrm{y}') \leq k$, so it remains to show $\mathrm{HD}(\mathrm{x}, \mathrm{y}) > k$ implies $\mathrm{ED}(\mathrm{x}', \mathrm{y}') > k$. Assume for contradiction there exists an alignment of x′, y′ with at most k edit operations. For each index i with $x_i \neq y_i$, at least one of x_i, y_i is not 0; let's call it z_i. Since $\mathrm{HD}(\mathrm{x}, \mathrm{y}) > k$, there are at least $k + 1$ such indices i, so at least one of them must involve no edit operation, i.e., match a character in the other string. But then the positions of z_i and of its matching character must differ by at least $k + 1$, which cannot happen if the alignment has at most k edit operations. The lower bound follows since $R^{\rightarrow}(\mathrm{HD}_{k,k+1}) = \Omega(k)$ (cf. [15]).

On the other hand, in the one-way model there is an $O(k \log n)$ upper bound for $\mathrm{ED}_{k,k+1}$ that nearly matches this $\Omega(k)$ lower bound. The basic idea is to use hashing; we omit the details in this version.

This state of affairs indicates that proving sketching lower bounds for edit distance may be quite hard. First, strong sketching lower bounds for $\mathrm{ED}_{k,k+1}$ require proving lower bounds that go beyond lower bounds in the one-way model. Second, the above approach does not go beyond the hardness of Hamming distance. For instance, it says nothing about $\mathrm{ED}_{k,2k}$, simply because $\mathrm{HD}_{k,2k}$ can be solved using a constant size sketch. At the moment, we do not know of an $\omega(1)$ sketching lower bound for $\mathrm{ED}_{k,2k}$, or more generally, of any randomized (one-way or simultaneous) communication lower bound for edit distance that exceeds its Hamming distance counterpart.

Acknowledgments

We are indebted to D. Sivakumar for his critical help in early stages of this work. We thank Tugkan Batu, Funda Ergün, Venkatesan Guruswami, and Ronitt Rubinfeld for helpful discussions.

References

[1] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *Proc. 14th SODA*, pages 523–526, 2003.

[2] L. Babai, P. Kimmel, and S. V. Lokam. Simultaneous messages vs. communication. In *Proc. 12th STACS*, volume 900, pages 361–372, 1995.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. Information theory methods in communication complexity. In *Proc. 17th CCC*, pages 93–102, 2002.

[4] T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami. A sublinear algorithm for weakly approximating edit distance. In *Proc. 35th STOC*, pages 316–324, 2003.

[5] R. Cole and R. Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM Journal on Computing*, 31(6):1761–1782, 2002.

[6] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proc. 13th SODA*, pages 667–676, 2002.

[7] G. Cormode, M. Paterson, S. C. Sahinalp, and U. Vishkin. Communication complexity of document exchange. In *Proc. 11th SODA*, pages 197–206, 2000.

[8] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. In *Proc. 28th ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 927–938. Springer, 2001.

[9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[10] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of ACM*, 18(6):341–343, 1975.

[11] P. Indyk. Algorithmic applications of low-distortion embeddings. In *Proc. 42nd FOCS*, pages 10–33, 2001.

[12] P. Indyk. Approximate nearest neighbor under edit distance via product metrics. In *Proc. 15th SODA*, pages 646–650, 2004.

[13] P. Indyk and J. Matousek. Low-distortion embeddings of finite metric spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 15, pages 177–196. CRC Press, 2nd edition, 2004.

[14] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th STOC*, pages 604–613, 1998.

[15] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(5):545–557, 1992.

[16] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[17] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[18] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Dokl.*, 10:707–710, 1965.

[19] W. J. Masek and M. S. Paterson. A faster algorithm for computing string edit distance. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[20] S. Muthukrishnan and S. C. Sahinalp. Approximate nearest neighbors and sequence comparisons with block operations. In *Proc. 32nd STOC*, pages 416–424, 2000.

[21] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[22] P. Pevzner. *Computational Molecular Biology*. Elsevier Science Ltd., 2003.

[23] S. C. Sahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. In *Proc. 37th FOCS*, pages 320–328, 1996.

[24] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1), 1974.

[25] A. C.-C. Yao. Some complexity questions related to distributive computing. In *Proc. 11th STOC*, pages 209–213, 1979.