A Sublinear Algorithm for Weakly Approximating Edit Distance

Tuğkan Batu*
University of Pennsylvania
batu@cis.upenn.edu

Funda Ergün
Case Western Reserve University
afe@eecs.cwru.edu

Joe Kilian
NEC Laboratories America
joe@nec-labs.com

Avner Magen
University of Toronto
avner@cs.toronto.edu

Sofya Raskhodnikova
MIT
sofya@mit.edu

Ronitt Rubinfeld
NEC Laboratories America
ronitt@nec-labs.com

Rahul Sami†
Yale University
sami@cs.yale.edu

ABSTRACT

We show how to determine whether the edit distance between two strings is small in sublinear time. Specifically, we present a test which, given two n-character strings A and B, runs in time $o(n)$ and returns "*CLOSE* " if their edit distance is $O(n^\alpha)$, or "*FAR*" if their edit distance is $\Omega(n)$, where α is a fixed parameter less than 1. Our algorithm for testing the distance works by recursively subdividing the strings into smaller substrings and looking for pairs of substrings in A, B with small edit distance. To do this, we query both strings at random places and use a special technique for "recycling" our samples so that the overall query complexity, as well as the running time, stays low. The test runs in time $\tilde{O}\left(n^{\max\{\frac{\alpha}{2},2\alpha-1\}}\right)$ for any fixed $\alpha < 1$. Our algorithm thus is a first step for trading off accuracy for efficiency for edit distance computation, which is useful when the input data is very long.

We also show a lower bound of $\Omega(n^{\alpha/2})$ on the query complexity of every algorithm that distinguishes pairs of string with edit distance at most n^α from those with edit distance at least $n/6$.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of algorithms and problem complexity

*Supported by ARO DAAD 19-01-1047 and NSF CCR01-05337.
†Supported by ONR grant N00014-01-1-0795.

STOC'03, June 9–11, 2003, San Diego, California, USA.

General Terms

Algorithms, Theory

Keywords

String matching, sublinear algorithms, approximation

1. INTRODUCTION

Let A, B be two strings from a fixed size alphabet Σ. The well known *edit distance* between A and B is defined as the minimum number of character insertions, deletions and substitutions required to transform A to B, or vice versa (it is indeed a distance, and in particular symmetric). This measure of string similarity is widely used in areas such as computational biology, text processing, and web searching. The edit distance is a well studied measure, and can be computed in quadratic time by a dynamic programming algorithm [4]. The task of computing or even approximating edit distance significantly faster has gained a lot of attention, especially in the bioinformatics community, where the regular quadratic algorithm is usually too slow to be used.

Efficient algorithms for the approximation task are crucial for two reasons: first, it is often the case that the strings being compared are quite long (hundreds of millions of characters in a genomic environment); second, it can serve as a part of an *exact* algorithm that works in sublinear amortized time in the following manner. A common task in bioinformatics is, given many pairs of strings, to compute edit distance *only* for pairs of *close* strings. The actual distance between most pairs of strings is irrelevant. Weak approximation algorithms are useful in cheaply eliminating those pairs, and thus allowing an exact algorithm to work on an extremely small fraction of the input pairs.

Our results

We show that if one is willing to accept a weaker quality of approximation, one can solve the edit distance problem surprisingly quickly. In fact, we give (to our knowledge) the first sublinear time algorithm for approximating the edit

distance. In particular, we show that one can distinguish pairs of strings of length n which have edit distance n^α from those which have edit distance $\Omega(n)$ in $\tilde{O}(n^{\alpha/2})$ time for $\alpha \leq 2/3$. For any constant $1 > \alpha > 2/3$, our running time is $\tilde{O}(n^{2\alpha-1})$, which is still sublinear. At the core of our algorithm is a sublinear-time procedure which constructs a useful implicit representation of all locations in A at which there are approximate copies of a specific substring of B; this procedure may be of independent interest.

Finally, we prove that every algorithm that distinguishes pairs of strings with edit distance n^α from strings with edit distance at least $n/6$ requires $\Omega(n^{\alpha/2})$ queries. To achieve this, we show the same lower bound for every algorithm that distinguishes a pair (A, B) of random strings from a pair (A, B) where A is random and B is a right shift of A by t positions for a random $t \in [n^\alpha/2, n^\alpha]$. This implies the lower bound for the edit distance problem, since two random n-bit strings have edit distance at least $n/6$ with high probability.

Related work

The edit distance problem is closely related to the longest common subsequence and sequence alignment problems. Computing the LCS exactly is the same as computing the edit distance, whereas an analogous statement cannot be made for approximate computations. In [MP], Masek and Paterson give an $O(n^2/\log n)$ algorithm for exactly computing the LCS of two strings of size n.[1] This nearly quadratic bound remains intact for the general case, with the costly computations occurring when the LCS is of linear length.

There has been a significant amount of related work on the slightly different problem of approximate string matching, where one would like to find all substrings of the text string of size n which match the pattern string of size m with edit distance at most k (insertions, deletions and substitutions of one character). We mention here a small sample of these works. Landau and Vishkin [3] gave an $O(nk)$ algorithm for this problem. Chang and Lawler [1] consider the case when the text string is random and errors are not too frequent. In this case, they give an algorithm which runs in sublinear expected time, namely $O((n/m)k \log m)$ time for $k < m/\log m + O(1)$. Myers [5] improves on their result but requires linear-time preprocessing on one of the two strings. In general, the first subquadratic time approximate string matching algorithm was given by Sahinalp and Vishkin [6], running in time $O(n \cdot (1 + poly(k) \cdot 1/m \log n))$. Cole and Hariharan [2] improved the running time to $O((nk^4/m) + n + m)$.

To put our work in perspective, we note that when the two strings are assumed to be close, dynamic programming restricted to the relevant fraction of the matrix can be applied, leading to a considerable saving in both time and space. In our setting this translates to an immediate $O(n^{1+\alpha})$ algorithm. Our algorithm is much more efficient, but not surprisingly uses the idea behind this naive saving as one of its components.

Our techniques

Our techniques are based on the observation that, if two strings have small edit distance, they will have many almost identical (with small Hamming distance) substrings whose locations in the respective strings are similar. We exploit this property by dividing one of the strings into blocks and determining whether most of these blocks occur in similar locations in the other string. In order to detect substrings that match a block with small Hamming distance, it suffices to randomly sample both the strings. To reduce our query (sample) complexity, we use a procedure that we call a "ruler" that collects a sublinear pool of samples from both strings, and then builds a structure containing all matching pairs of the form *(location in A, block in B)*. In addition, we make use of recursion while subdividing our blocks, which allows us to further improve the complexity of the algorithm. Finally, we "quantize" the locations in the strings output by the matching process for two substrings; that is, we consider only discrete "shifts" between the location of a substring in one string and in the other. This results in many fewer cases to consider (and, more importantly, to store) in terms of where a block of one string is located in the other.

Overview of paper

The rest of this paper is structured as follows: Section 2 defines the edit distance testing problem. In Section 3 we develop a sublinear time algorithm for this problem: we describe our basic techniques in Sections 3.1-3.4; in Section 3.5 we show how to match the blocks in one string to substrings in the other, and in Section 3.6 we present our main algorithm that uses these techniques to estimate the edit distance. In Section 4, we give a lower bound on the query complexity for this problem.

2. PRELIMINARIES

The strings we consider are over a constant size alphabet Σ. For simplicity and without loss of generality, we assume a binary alphabet $\Sigma = \{0, 1\}$, and the input strings (usually denoted A, B) of length n. $A[i]$ refers to the ith character (bit) of string A, and $A[i \ldots j]$ refers to the substring of A delimited by the characters (bits) at positions i and j.

$\mathcal{D}(A, B)$, the *edit distance* between two binary strings A and B, is the minimal number of single bit insertions, deletions, or replacements required to generate B from A, or vice versa.

The edit distance testing problem

We wish to devise an algorithm to distinguish pairs of n-bit strings A, B that are close to each other from pairs that are far from each other in terms of their edit distance. The required behavior from this algorithm on input A, B, and parameters α, C, $0 < \alpha < 1, C > 1$, is as follows.

- if $\mathcal{D}(A, B) \leq n^\alpha$, output *CLOSE* with probability at least 2/3.
- if $\mathcal{D}(A, B) > n/C$, output *FAR* with probability at least 2/3.

The output of the algorithm is unspecified for $n^\alpha < \mathcal{D}(A, B) \leq n/C$. We treat C as a fixed constant and do not analyze the dependence of our algorithm on C.

3. A TEST FOR EDIT DISTANCE

We now describe a recursive algorithm that checks whether the edit distance between two n-bit strings A and B is small ($\leq n^\alpha$) or large ($\Omega(n)$). We arbitrarily designate A to be the *reference string*, against which B is matched. On the highest level, our algorithm is based on the standard divide

[1] In fact, the strings can be of different length; we present the equal-length case for simplicity.

and conquer paradigm: B is is broken up into substrings, which are recursively matched against A. The matching for these local patches is pieced together to form a matching (alignment) for the larger string. However, since it would be too expensive to look at all the subintervals, we randomly sample a small number of them and rely on the statistical properties of these matchings. We then analyze the effect of the statistical uncertainties that arise as a result of the sampling.

We start by discussing the relationship between the edit distance of two strings and the similarity of their substrings.

3.1 Approximate matchings and coordinated matchings

A matching of B against A describes how A can be obtained from B. In particular, it gives an *alignment* between the matching portions of A and B. Consider how a subinterval $I = B[s \ldots e]$ corresponds to A. We may think of I as being matched against a substring $A[s' \ldots e']$; the matching involves a sequence of operations on $A[s' \ldots e']$ that transform it into I. In general, $s \neq s'$; we refer to the quantity $s' - s$ as the *shift* of I. The shift is due to external edits required to match the earlier portions of A and B. We refer to the number of edit operation needed to transform $A[s', e']$ to I as the *internal edit distance.*[2] Note that there may be many possible low-edit matchings of I against A.

We are interested in matchings in which the internal edit distance is a small fraction of the total number of characters being matched.

DEFINITION 1. *An interval $I = B[s \ldots e]$ has a (t, E)* matching *with respect to A if for some interval $A[s' \ldots e']$, $s' = s + t$ and $\mathcal{D}(A[s' \ldots e'], I) \leq E$.*

If $\mathcal{D}(A, B)$ is small, it is apparent that most subintervals in B will have an approximate matching somewhere in A. Further, these matching subintervals must have similar shifts, because a change in the shift value can only arise from an edit (specifically, insert or delete) operation. This leads us to consider *coordinated matchings*:

DEFINITION 2. *Given a collection of intervals $\mathcal{I} = I_1, \ldots, I_k$, we say that $\mathcal{I}$ has a (t, σ, E, D)* coordinated matching *with A if for all but D of the intervals $I_i \in \mathcal{I}$, I_i has a (t_i, E) matching with A, where $|t - t_i| \leq \sigma$.*

We can decompose an interval I of size S into k disjoint contiguous subintervals, $I_1, \ldots, I_k$, each of size $S' = S/k$ (we assume that $k|S$). The existence of a coordinated matching of $\mathcal{I}$ indicates that most of the intervals therein are well matched with similar shifts in A. Lemma 1 says that if these subintervals have a coordinated matching with suitable parameters then I has an approximate matching.

LEMMA 1. *Let $A, I, I_1, \ldots, I_k, S$ and S' be as above. If $I_1, \ldots, I_k$ have a $(t, \sigma, \epsilon S', \delta k)$ coordinated matching with A, then I has a $(t, \beta S)$ approximate matching with A, where*

$$\beta = \left(\frac{2\sigma}{S'} + \epsilon + \delta\right).$$

[2]This is no longer a distance function. The corresponding matching of the *internal edit distance* is also described in the bioinformatics literature as *local alignment*.

PROOF. (Sketch) We construct a matching for I by stitching together the matchings for $I_1, \ldots, I_k$, correcting for gaps, overlaps and unmatching subintervals.

Let $I = B[s \ldots e]$ and $I_i = B[s_i \ldots e_i]$. If I_i has a (t_i, ϵ) approximate matching, we denote by $I'_i = A[s'_i \ldots e'_i]$ the substring of A that is transformed into I_i (choosing arbitrarily if there are multiple matches). I'_i can be transformed into I_i using $\epsilon S'$ edit operations. If I_i does not have a (t_i, ϵ) matching (for $t_i \in [t - \sigma, t + \sigma]$), we define $t_i = t$ and I'_i to be $A[s_i + t \ldots e_i + t]$, i.e., the region obtained by translating the interval $[s_i \ldots e_i]$ by t. We can trivially transform I'_i to I_i using S' edit operations.

We transform the interval $A[s + t \ldots e'_k]$ into I as follows. If $s'_1 = s + t$, then we simply transform I'_1 into I_1 using the same edit operations as for that matching. If $s'_1 < s + t$, then the $s + t - s'_1$ first characters of I' are missing from $A[s+t \ldots e_k]$; we add these to the beginning of $A[s+t \ldots e'_k]$ using $s+t-s'_1$ insert operations, and then proceed as before. Similarly, if $s'_1 > s + t$, we trim the first $s'_1 - s - t$ characters from $A[s + t \ldots e'_k]$ and proceed as before. We are left with the remaining portion of $A[s + t \ldots e'_k]$ ($A[e'_1 + 1 \ldots e'_k]$), which must be transformed into the remaining portion of I ($B[e_1 + 1 \ldots e_k]$).

To complete the transformation, we transform I'_i into I_i, for $i = 2, \ldots, k$, in the same manner, yielding I. At each stage, we trim or add to the remaining string so that I'_i is a prefix, and then perform the transformation from I'_i to I_i.

It remains to compute the number of edits required by this transformation. The number of edits required to transform I'_i to I_i, for all i for which there are is an approximate match, is at most $\epsilon S' k$. For at most δk intervals, I'_i and I_i don't have a good match; the trivial transformation costs at most $S' \cdot \delta k = \delta S' k$.

We must also account for the $|s'_i - (s + t)|$ edit operations required prior to transforming I'_1 and the $|s'_i - (e'_{i-1}) + 1|$ editing operations required to align the remaining string prior to transforming I_i, for $i > 1$. By the definition of a coordinated matching, $|s'_i - (s + t)| \leq \sigma$. Since $s'_i = s_i + t_i$ we can write,

$$\begin{aligned} |s'_i &- (e'_{i-1} + 1)| \\ &= |(s_i + t_i) - (s'_{i-1} + (e'_{i-1} - s'_{i-1}) + 1)| \\ &= |(s_i + t_i) - (s_{i-1} + t_{i-1} + (e'_{i-1} - s'_{i-1}) + 1)| \\ &\leq |t_i - t_{i-1}| + |(e'_{i-1} - s'_{i-1} + 1) - (s_i - s_{i-1})|. \end{aligned}$$

It follows from the definition of a coordinated matching that $|t_i - t_{i-1}| \leq 2\sigma$. The latter term is simply the absolute difference between the length of I_{i-1} ($s_i - s_{i-1}$) and the length of $|I'_{i-1}|$ ($e'_{i-1} - s'_{i-1}) + 1$). If I'_{i-1} can be transformed into I_{i-1} using $\epsilon S'$ edit operations, this difference cannot be more than $\epsilon S'$, and if no such matching exists, then I'_{i-1} will have the same length as I_{i-1} by definition. Thus, at most $2\sigma + \epsilon S'$ operation are required per interval, giving at most $2\sigma k + \epsilon S' k$ operations in all.

Recalling that $S = S'k$, we have at most $\left(\frac{2\sigma}{S'} + \epsilon + \delta\right)S$ edits required, implying the lemma.

□

Lemma 2 shows that if a good matching for an interval exists then there must be a coordinated matching among its subintervals.

LEMMA 2. *Let $A, I, \mathcal{I} = I_1, \ldots, I_k, S$ and S' be as above. Let $c > 1$ and $S > cE$. If I has a (t, E) matching with A*

then $\mathcal{I}$ has a $(t, E, cE/k, k/c)$ coordinated matching with A.

PROOF. (Sketch) Let $I = B[s \ldots e]$. We consider the matching from $A[s+t \ldots q]$ to I that has edit distance E. We for each I_i consider the smallest interval I_i' of A containing all the characters that are matched to characters in I_i. If no such characters exist, we no not assign I_i. We claim that this correspondence induces a $(t, E, cE/k, k/c)$ coordinated matching.

First, we note that the I_i's are disjoint, since our edit operations do not change the ordering of the characters of A that are matched. The edit operations transforming $A[s+t \ldots q]$ into I transform each I_i' into I_i. Each edit operation affects only one (I_i', I_i) pair or unassigned I_i - either it deletes a character from at most one I_i' or it inserts a character into exactly one I_i. Hence, the sum of edit distances between I_i' and I along with the sum of edits assigned to unassigned I_i is at most E. If more than k/c had edit distance greater than cE/k, or were unassigned (with an edit cost of $S' > cE/k$), this would cause the sum to be greater than E, a contradiction.

It remains to show that the translation (shift) t_i between each I_i' and I_i satisfies $|t_i - t| \leq E$. Now, consider what happens if we edit $A[s+t \ldots q]$ to obtain I, first by deleting the unmatched characters, one by one, and then inserting the new characters, one by one. At each step, we can recompute the matchings and hence the shifts for each (I_i', I_i) pair. Each operation can change any t_i by at most 1 each way. However, at the end of this process, when $A[s+t \ldots q] = I$, $t_i = t$, so the original values of t_i could not be more than E away from t. □

A special case of this lemma is the existence of coordinated matchings where the intervals have no internal edit distance at all.

LEMMA 3. *Let $A, I, I_1, \ldots, I_k, S$ and S' be as above. If I has a (t, E) matching with A, and $k \geq E$, then $I_1, \ldots, I_k$ have a $(t, E, 0, E)$ coordinated matching with A.*

DISCUSSION: We use Lemmata 1 and 2 in concert to detect good matchings. Suppose string A and B have a good matching. We break B up into subintervals and use Lemma 2 to argue that these intervals have a good coordinated matching. In the next section we show how to efficiently detect a good coordinated matching. Once detected, we use Lemma 1 to infer the existence of a good matching between A and B. The properties of the inferred matching will be far weaker than the one that actually exists, but sufficiently good to distinguish between the two cases we consider. To obtain the strongest result, we apply this technique recursively, taking care that the degradation in the guarantee does not grow too large.

3.2 Detecting coordinated matchings via sampling

The crux of our algorithm is to approximately detect a coordinated matching with a very few queries. Given a set of intervals $\mathcal{I} = I_1, \ldots, I_k$, we wish to determine for which t, $\mathcal{I}$ has a (t, σ, E, D) coordinated matching. We actually accomplish an approximate version of this task: For all t, if $\mathcal{I}$ has a (t, σ, E, D) coordinated matching we with high probability detect that it has a $(t, \sigma, E, D + \epsilon k)$ coordinated matching, for any constant $\epsilon > 0$.

To implement our detection routine, COORD-MATCHES, we assume for now that we have a subroutine, MATCHES(A, I, E), that determines for which t, I has a (t, E) matching with A. We will later implement MATCHES recursively using COORD-MATCHES. Our actual subroutines only approximate this behavior; we later adapt our technique to accommodate this approximation.

A key observation is that a randomly selected set of $O(\log n)$ subintervals will approximate the behavior of the entire set. We use the following simple consequence of the Chernoff bound.

LEMMA 4. *For any positive ϵ and c, there exists d such that the following is true. Suppose that a randomly chosen element of a set S has some property Z with probability p. If we uniformly sample (with replacement) $d \log n$ elements from S, the fraction p' of these samples with property Z satisfies $p - \epsilon/2 \leq p' \leq p + \epsilon/2$ with probability $1 - 1/n^c$.*

We give the sampling procedure COORD-MATCHES in Figure 1. The parameters $A, \mathcal{I}$ and σ are as in Definition 2. The parameters ϵ and c control the accuracy and reliability of the estimate, as analyzed in Lemma 5.

LEMMA 5. *With probability $1 - 1/n^{c-1}$ over the random coins of* COORD-MATCHES, *the output T of* COORD-MATCHES *$(A, \mathcal{I}, \sigma, E, D, \epsilon, c)$ has the following two properties:*

1. *If $\mathcal{I}$ has a (t, σ, E, D) coordinated matching then $t \in T$.*
2. *If $t \in T$ then $\mathcal{I}$ has a $(t, \sigma, E, D + \epsilon k)$ coordinated matching.*

PROOF. For any t, if at most D intervals I_i do not have a (t_i, E) matching where $|t_i - t| \leq \sigma$ then by Lemma 4, with probability at least $1 - 1/n^c$ at most a $D/k + \epsilon/2$ fraction of the I_{i_j}s do not have such a matching, in which case $t \in T$. Similarly, if more than $D + \epsilon k$ intervals do not have a (t_i, E)-matching with $|t_i - t| \leq \sigma$, then with probability at least $1 - 1/n^c$ at least a $(D/k + \epsilon) - \epsilon/2$ fraction of the I_{i_j}s do not have such a matching, in which case $t \notin T$. Thus, for both types of errors, the probability of making a mistake is thus at most $1/n^c$. Since there are at most n possible errors possible (for each t the number of non-matches can be either too big or too small, but not both), the lemma follows from the union bound. □

3.3 Quantizing shifts

Our MATCHES and COORD-MATCHES algorithms may conceivably give an output set T consisting of n elements. While not affecting the query complexity, this by itself is more time than we wish to take. Further, observe that for detecting strings with the edit distance of at most n^α, we may restrict the allowed shifts to $[-n^\alpha \ldots + n^\alpha]$. However, to achieve a $o(n^\alpha)$ running time, we must further restrict the set of possible outputs. We do this by specifying a quantization parameter, Q, which governs the precision of the output.

DEFINITION 3. *The Q-*quantization *of t, denoted $t[Q]$, is the unique value Qk (k an integer) satisfying $-Q/2 < t - Qk \leq Q/2$. The Q-quantization of a set consists of the Q-quantization of its values. If an interval has a (t, E)*

COORD-MATCHES$(A, \mathcal{I}, \sigma, E, D, \epsilon, c)$

1. Let d be as in Lemma 4 for the given ϵ and c, and $l = d \log n$. Choose $i_1, \ldots, i_l$ uniformly and independently from $[1 \ldots k]$.
2. For each I_{i_j}, compute $T_j = \text{MATCHES}(A, I_{i_j}, E)$.
3. Return $T = \text{MERGE}(T_1, \ldots, T_l, \sigma, \Delta)$, where $\Delta = (D/k + \epsilon/2)l$ and MERGE is defined below.

MERGE$(T_1, \ldots, T_l, \sigma, \Delta)$

1. Return the set T, where $t \in T$ iff $T_j \cap [t - \sigma \ldots t + \sigma] = \emptyset$ for at most Δ sets T_j.

Figure 1: Sampling algorithm for (approximately) finding coordinated matches.

matching with A, we say that it has an $(t[Q], E)$ quantized matching *with A. We say that a set of intervals, $\mathcal{I}$ has a (t, σ, E, D)* quantized coordinated matching *with A if $t = t[Q]$ and for all but D of the intervals $I_i \in \mathcal{I}$, I_i has a (t_i, E) quantized matching with A, where $|t - t_i| \leq \sigma$. We define* QMERGE *as the Q-quantization of the output of* MERGE.

The key observation to make is that coordinated matchings already allow for some "wiggle room," in the shifts allowed for the intervals. Adding moderate amounts of quantization doesn't change this wiggle room significantly. Proposition 1 quantifies this relationship; its proof follows immediately from the fact that quantization only alters a number by at most $Q/2$.

PROPOSITION 1. *If interval $\mathcal{I}$ has a (t, σ, E, D) coordinated matching with respect to A then it has a $(t[Q], \sigma + Q/2, E, D)$ quantized coordinated matching with A. If $\mathcal{I}$ has a (t, σ, E, D) quantized coordinated matching with A then it has a $(t, \sigma + Q/2, E, D)$ coordinated matching with A.*

Intuitively, if we don't make the quantization factor too large then we can make qualitatively the same inferences using quantized shifts as we can using unquantized shifts.

3.4 Recursively using coordinated matches

Our COORD-MATCHES algorithm makes calls to MATCHES, which has to find good matches for individual intervals; we now describe how the MATCHES procedure is implemented. Using Lemma 1, we can detect a good match for an interval I by breaking I into subintervals, detecting good coordinated matchings for these intervals, and inferring the existence of good matches for I. That is, we call COORD-MATCHES using a suitable decomposition of I and using suitable error tolerances. While running COORD-MATCHES, we make calls to MATCHES on a subset of these subintervals, which are approximated via a call to COORD-MATCHES on a suitable decomposition of these subintervals, and so on. This process yields a multi-stage algorithm in which matches found in a given stage are used to generate matches in the earlier stage.

At each stage, we match smaller intervals, and require that the matches have smaller internal edit distances. Eventually, we seek (t, E) matches in which $E < 1$ (hence, E might as well be 0). But note that if an interval I has a $(t, 0)$ matching with respect to A, then A must contain interval I unchanged except for a translation or *shift* by t positions. In this case, we compute the set of allowable t values directly, using the algorithm SHIFTS described below; this forms the final stage of the recursion. We now turn to the description of this algorithm.

3.5 Finding approximate block shifts via ruler procedure

This subsection describes an algorithm to efficiently find substrings in A that approximately match a block (interval) in B. This procedure is at the core of our edit distance testing algorithm.

The approximate matching problem is as follows (for simplicity, we will drop the term 'approximate'). Given a block $I = B[s \ldots e]$ of length $b = e - s + 1$ in B; and a constant $c_2 > 1$, find all indices s' such that $A[s' \ldots (s' + b - 1)]$ *matches* I, in the sense that the two substrings have Hamming-distance at most b/c_2. Note that, if $\mathcal{D}(A, B) < n^\alpha$, it is enough to consider $s' \in [s - n^\alpha, s + n^\alpha]$.

Thus, we now need to solve the following. Given a string I of length b, and a string $A' = A[(s - n^\alpha) \ldots (s + n^\alpha + b - 1)]$, we want to find all *shifts* t of I within A', such that $A'[t + 1 \ldots t + b]$ matches I. That is, we want to find all length b substrings of A' with Hamming distance at most b/c_2 from I. Naively, we can randomly sample $O(\log n)$ indices i to determine (with high probability) if the substring $A'[(t+1) \ldots (t+b)]$ matches I, for a given t, and try all $2n^\alpha$ possible shifts t. This requires $\Omega(n^\alpha)$ queries to A. Below we reduce the number of queries by a "ruler" procedure.

Suppose we would like to compare pairs of characters $A'[i], I[j]$ such that some pairs $A'[i], I[j]$ are checked for every $i - j$ from 0 to $u = 2n^\alpha$. Here is how to achieve this with $\sqrt{u}$ queries to each string, provided that $b >> \sqrt{u}$: In A', character positions divisible by $\sqrt{u}$ are queried: $A'[\sqrt{u}], A'[2\sqrt{u}], \ldots, A'[u]$. In I, $\sqrt{u}$ consecutive positions are queried: $I[1 \ldots \sqrt{u}]$. Intuitively, queries to A' act as "centimeter" marks on the ruler, and queries to I act as "millimeter" marks. For every $t = 0, 1, \ldots, u$, there is a pair of queried positions $A'[i], I[j]$ with $i - j = t$. Let $\text{cen} = \lfloor t/\sqrt{u} \rfloor$ and $\text{mil} = t \bmod \sqrt{u}$. Then $A[\text{cen} \cdot \sqrt{u}]$ and $B[\sqrt{u} - \text{mil}]$ are queried positions exactly distance t apart.

We can extend this idea to test whether the entire block matches with shift l, using the random sampling idea mentioned earlier: Pick $l = \Theta(\log n)$ numbers $m_1, m_2, \ldots, m_l$ randomly from the range $[0, b - \sqrt{u}]$. For each tick mark on the ruler, construct a *fingerprint* by querying at l offsets instead of just 1; for example, in A', the fingerprint of the centimeter mark $\sqrt{u}$ is the sequence of l bits

$$(A'[\sqrt{u} + m_1], A'[\sqrt{u} + m_2], \ldots, A'[\sqrt{u} + m_l]).$$

Now, we can detect with high probability whether the block matches with shift t by comparing the fingerprints of *cen* and *mil* as defined above.

Up to this point we have assumed $b >> \sqrt{u}$. We use the same idea when $b \leq \sqrt{u}$; the only difference is that

we need to make the ruler asymmetric. In this case, we can have only $O(b)$ millimeter tick marks, and so we need $\Omega(u/b)$ centimeter tick marks. Thus, in general we can find all matching shifts l by using $O(\max\{\sqrt{u}, u/b\}\log n)$ queries.

Efficient implementation of the ruler

We now describe a data structure that allows us to efficiently execute the ruler procedure. Recall that we want to detect when a tick mark i in A' has the same fingerprint as a tick mark j in I. To do this, we maintain a binary search tree, with a leaf corresponding to each fingerprint f encountered thus far. (In practice, a hash tree may be better, but it does not change the asymptotic performance.) Each leaf contains pointers to two linked lists: the A-list contains indices i (in A') that resulted in fingerprint f, and the B-list contains indices j (in I) that yielded f. It takes $O(\log n)$ time per tick mark, and thus $O(\max\{\sqrt{u}, u/b\}\log n)$ time overall, to build up this data structure.

When all tick marks have been processed, the data structure contains an implicit representation of all shifts t such that I matches $A'[t+1\ldots t+b]$, in the following sense: for each fingerprint f, every combination of an index i from f's A-list and j from f's B-list describe a matching shift $t = j - i$. However, it is still potentially expensive to go from this to an *explicit* list of all matching t values. The problem is simply that there may be $\Omega(u)$ such values.

If we need to know each individual t value precisely, there is no way to avoid this problem. To get around this, the algorithm described in Section 3.6 only uses *quantized* shift values, *i.e.*, values of t rounded to multiples of some integer Q. Reporting distinct multiples of Q for which some t matched, reduces the worst-case size of the output list to $\Omega(u/Q)$. It is easy to take advantage of this reduction with our data-structure: first prune the B-list so that it never has two j values that get rounded off to the same multiple of Q, and then, if $Q > g$, also prune the A-list such that it never has two i values that get rounded off to the same multiple of Q. The final algorithm is shown in Figure 2. The following theorem summarizes the performance of this algorithm:

THEOREM 1. *Procedure* SHIFTS *finds all quantized shifts t of interval I in A', with high probability. It runs in time $O(\max\{\sqrt{u}, u/b, u/Q\}\log n)$, where $u = |A'| - b$.*

PROOF. If t is a shift corresponding to an exact match, then the preceding discussion of the ruler shows that the corresponding quantized shift value will be found. If t is a shift corresponding to a Hamming distance of greater than $2b/c_2$, then at least b/c_2 of the Hamming errors must occur after the first g characters of I. Hence, any one m_i will find a mismatch with probability at least $1/c_2$. Setting $d = 2$, the probability of the fingerprints matching for t is then at most $\frac{1}{n^2}$. There are less than n possible shift values, and so the probability of finding *any* incorrect shift t is at most $\frac{1}{n}$. Higher values of d can be used to obtain error bounds of at most $\frac{1}{n^c}$ for any constant c, hence the high probability result. The running time bound follows by taking the sum of the time to construct the implicit representation of all shifts, and the time to produce the output. □

3.6 The edit distance testing algorithm

We now have all the tools we need to build our algorithm for testing edit distance. The algorithm is shown in Figure 3. The top-level procedure is a routine DECIDE that takes as input the two strings A and B, and the parameter α. DECIDE calls MATCHES to search for a match of B in A with edit distance at most n^α; if such a match is found, *CLOSE* is output, otherwise *FAR* is output. MATCHES is a recursive procedure, recursing through the procedure COORD-MATCHES. The recursion terminates when the required internal edit distance in each block is less than 1; in this case, MATCHES uses SHIFTS to directly find the matches.

Depth of recursion:

At each level of the recursive decomposition, the size of the interval input to MATCHES goes down by a factor of $\Omega(n^{\alpha-1})$. Thus for any constant $\alpha < 1$, there is a constant number r of levels of recursion required to reach a state in which the intervals have size $O(n^{1-\alpha})$; at this point, $E < 1$ and hence SHIFTS will be called, terminating the recursion.

We assign a *height* to each invocation of procedure MATCHES as follows: the final invocation that calls SHIFTS has height 0, the level above that has height 1, and so on till we get that the height of the top-level invocation of MATCHES is r. We also define the height of an invocation of COORD-MATCHES to be the height of the MATCHES procedure that invoked it.

Correctness of the algorithm

We need to show that with a suitable choice of constants c_1 and ϵ (perhaps dependent on α), procedure DECIDE correctly solves the edit distance testing problem. We first prove that if $\mathcal{D}(A,B) \le n^\alpha$, the algorithm outputs *CLOSE*.

LEMMA 6. *If $\mathcal{D}(A,B) \le n^\alpha$, the algorithm outputs CLOSE with high probability, for any parameter values $\epsilon < 1$ and $c_1 > 1$.*

PROOF. (Sketch) Note that it is sufficient to prove that if there is a (quantized) (t,E) matching of B with respect to A, the top-level invocation of MATCHES will find it, with high probability. We prove this statement by induction on the height h of the invocation of MATCHES. For $h = 0$, this is true because of the correctness of the SHIFTS procedure. Assuming it is true for height $(h-1)$, we show that it is true for height h as well: Let MATCHES(A, I, E) be a height h invocation, and suppose that I has a (t,E) matching with respect to A. Then, by Lemma 2, there exists a coordinated matching with the parameters specified in Step (2c) of MATCHES. Then, using a variant of Lemma 4, we see that most of the sampled intervals have some match with translation t_i close to t. Now, consider the recursive calls to MATCHES made by COORD-MATCHES. By the inductive assumption, they will report these matches t_i with high probability. Now, using Lemma 5, we know that they will find these translations t_i, and hence COORD-MATCHES will report t among its output T. Hence, the translation t is the output of the level h MATCHES, with high probability. □

It remains to show that the algorithm outputs *FAR* with high probability when $\mathcal{D}(A,B) > n/C$, for an appropriate choice of constants.

LEMMA 7. *There exist values for the constants ϵ and c_1 (dependent only on α and C), such that if the algorithm outputs CLOSE (with high probability), then $\mathcal{D}(A,B) < n/C$.*

SHIFTS(A', I, Q, c_2)

/ Find all shifts of I in A' with Hamming distance $< 2|I|/c_2$, quantized in multiples of Q */*

1. Let $b = |I|$, $u = |A'| - b$, and $g = \min\{b/c_2, \sqrt{u}\}$.
2. Let $l = d \cdot \log n/(-\log(1 - 1/c_2))$, for some constant $d > 2$. Choose integers $m_1, m_2, \ldots, m_l$ independently and uniformly at random in $[0, b - g]$.
3. Initialize the fingerprint search tree.
4. For $i = g, 2g, \ldots, u$ do
   - Compute fingerprint $f(i) = (A'[i + m_1], \ldots, A'[i + m_l])$.
   - Locate $f(i)$ in the search tree, creating a new leaf if necessary.
   - Add i to the A-list for $f(i)$.
5. For $j = 1, 2, \ldots, g$ do
   - Compute fingerprint $f(j) = (I[j + m_1], \ldots, I[j + m_l])$.
   - Locate $f(j)$ in the search tree, creating a new leaf if necessary.
   - Add j to the B-list for $f(j)$.
6. *Quantizing*: For each fingerprint f, scan the B-list for f and round each j value to the nearest multiple of Q, deleting repeated values; if $Q > g$, also scan the A-list for f and round each i value to the nearest multiple of Q, deleting repeated values.
7. For each fingerprint f, each i in f's A-list, and each (rounded) j in f's B-list, output $t = i - j$.

Figure 2: "Ruler" procedure for finding approximate block shifts.

DECIDE(A, B, α, C)

0. Choose sufficiently small ϵ, and sufficiently large c_1 (for the given α and C).
1. Choose quantization parameter $Q = \epsilon \cdot \min\{n^{1-\alpha}, n^{\alpha/2}\}$.
2. Compute $T =$ MATCHES(A, B, n^α).
3. If T is nonempty, then output *CLOSE*, else output *FAR*.

MATCHES(A, I, E)

1. If $E < 1$, use SHIFTS to compute T.
2. If $E \geq 1$,
   - 2a. Set $k = \min\{\epsilon n^{1-\alpha}, 2c_1 E\}$.
   - 2b. Decompose I into a set $\mathcal{I}$ of contiguous disjoint intervals of size $|I|/k$.
   - 2c. Compute $T =$ COORD-MATCHES$(A, \mathcal{I}, E, c_1 E/k, k/c_1)$.
3. Return T.

Figure 3: The Edit Distance Testing Algorithm

PROOF. (Sketch, of Lemma 7) First, as the quantization is at most ϵ times the size of the smallest interval size in the recursion, it alters the constants but not the qualitative matching results. Second, as long as the underlying SHIFTS algorithm and interval sampling procedures took sufficiently many samples ($O(\log n)$), the effects of their imprecision could be reduced to ϵ amounts as well. For the rest of the discussion, we ignore these issues.

The other potential source of error is in inferring the existence of a matching from a coordinated matching, at each level of the recursion. We use Lemma 1 to bound this error; this involves a careful analysis of the β factors that arise at each level of the recursion.

We consider the values of S', σ, E, and (D/k) in each invocation of COORD-MATCHES. Let $\lambda = \epsilon \cdot n^{1-\alpha}$. In each of the first $r-2$ levels, MATCHES subdivides the interval into λ intervals. At height r (the top level), we have

$$S' = \frac{n}{\lambda}, \sigma = n^{\alpha}, E = n^{\alpha} \cdot \frac{c_1}{\lambda}, (D/k) = \frac{1}{c_1}$$

Now, let us write β_h for the factor β in Lemma 1, corresponding to the values of S', σ, E, and (D/k) at height h. We have

$$\beta_r = \frac{2n^{\alpha}\lambda}{n} + \epsilon + \frac{1}{c_1} = 3\epsilon + \frac{1}{c_1}$$

Proceeding in this manner, we find at level 2,

$$S' = \frac{n}{\lambda^{r-1}}, \sigma = n^{\alpha} \cdot (\frac{c_1}{\lambda})^{r-2}, E = n^{\alpha} \cdot (\frac{c_1}{\lambda})^{r-1}, (D/k) = \frac{1}{c_1}$$

$$\begin{aligned} \beta_2 &= \frac{2n^{\alpha}c_1^{r-2}\lambda}{n} + \epsilon + \frac{1}{c_1} \\ &= 2c_1^{r-2} \cdot \epsilon + \epsilon + \frac{1}{c_1} \end{aligned}$$

At this level, we must have $2c_1E < \lambda$, or else the recursion would not terminate at level r. Hence, the pattern changes here; However, observing that σ reduces by a factor of λ/c_1 and S' reduces by a factor $2c_1E < \lambda$, σ/S' can increase by at most another factor of c_1, and so we have

$$\beta_1 \leq 2c_1^{r-1} \cdot \epsilon + \epsilon + \frac{1}{c_1}$$

Note that as r is fixed (by α), we can pick values of c_1 and ϵ to achieve β_1 as small as we wish: we first select a suitable value of c_1, and then select ϵ based on the chosen c_1. In particular, we can pick values such that $\beta_1 < \frac{1}{rC}$. Further, observe that $\beta_r, \ldots, \beta_2 < \beta_1$.

Now, suppose algorithm DECIDE outputs *CLOSE*. This means that at each level of the recursion, we have a coordinated matching. Consider an instance of COORD-MATCHES at height 1, and apply Lemma 1. This says that the corresponding interval at height 2 has a matching with additional edit distance at most β_1 times the length of the interval. Adding up the edits over all intervals in a level, and over all levels of recursion, we see that the total edit distance of A from B is less than $\beta_1 n.r < \frac{n}{C}$. □

Combining Lemma 6 and 7, we get the following theorem:

THEOREM 2. *For any fixed $\alpha < 1$, we can choose constants ϵ and c_1 such that procedure* DECIDE *solves the edit distance testing problem with high probability.*

3.7 Running time analysis

In this section, we provide the running time analysis of our algorithms. The analysis is based on three cases, depending on the value of α.

Case (i): $\alpha < 1/2$

In this case, there will only be one level of recursion. At the top level, B will be broken into intervals of size $O(n^{\alpha})$; the expected number of edits per interval is less than 1, and so in the next level the SHIFTS procedure will be used to find the matches of these intervals. Thus, there are $d \log n$ calls to SHIFTS; for the specified Q, each call takes $O(n^{\alpha/2} \log n)$ time. In addition, there is one call to QMERGE, which takes $O(n^{\alpha/2} \log n)$ time, thus giving us a total running time of $O(n^{\alpha/2} \log^2 n)$.

Case (ii): $1/2 < \alpha < 2/3$

When $1/2 < \alpha < 2/3$, there will be two levels of recursion. At the top level, we break B into intervals of size $c_1 n^{\alpha}$. In the second level, each selected interval is further broken into subintervals of size $n^{\alpha/2}$. Finally, we find matches for these subintervals using SHIFTS. Thus, there are $O(log^2 n)$ calls to SHIFTS; again, each call takes $O(n^{\alpha/2} \log n)$ time. There are also $O(\log n)$ calls to QMERGE. All together, the running time is $O(n^{\alpha/2} \log^3 n)$.

Case (iii): $\alpha > 2/3$

We now consider the general case, when there are $r > 2$ levels of recursion. We note that there two sources of degradation in this recursive algorithm. First, we incur a time and query overhead of $(\log n)^{r+O(1)}$, because of the random sampling at each level of recursion. The second, more significant degradation comes from the SHIFTS procedure at the final stage of the recursion. Each invocation of SHIFTS has to find all (quantized) shifts in the range $[-n^{\alpha} \ldots n^{\alpha}]$ of a block of size $O(n^{1-\alpha})$. However, for $\alpha > 2/3$, we have $n^{1-\alpha} < \sqrt{n^{\alpha}}$, and so the ruler used in the SHIFTS procedure has to be asymmetric. As a result, the running time of each invocation is $O(n^{2\alpha-1} \log n)$. Thus, the edit distance testing algorithm has an overall running time of $\tilde{O}(n^{2\alpha-1})$.

4. A QUERY COMPLEXITY LOWER BOUND FOR THE EDIT DISTANCE PROBLEM

This section proves the following lower bound for the edit distance problem defined in Section 2:

THEOREM 3. *Any algorithm for the edit distance problem requires $\Omega(n^{\alpha/2})$ queries.*

In fact, we show $\Omega(n^{\alpha/2})$ lower bound for the possibly easier problem of distinguishing a pair (A, B) of random strings from a pair (A, B) where A is random and B is a right shift of A by t positions for a random $t \in [n^{\alpha}/2, n^{\alpha}]$. Since two random strings have a linear edit distance, Theorem 3 follows.

LEMMA 8. *With probability at least $1 - o(1)$, two random n-bit strings have edit distance $\geq n/6$.*

PROOF. (of Lemma 8) It is enough to show that for a fixed n-bit string X, the fraction of strings within edit distance $n/6$ of X is $o(1)$. A string that is at most d away from

X, is obtained by choosing d locations, and for each of the locations, picking one the deletion, replacement, insertion of a new bit, or no-edit operation. Thus, the number of strings within edit distance d from X is at most $\binom{n}{d} \cdot 5^d$. Substituting $d = n/6$ and recalling that $\binom{n}{\beta n} \leq 2^{H_2(\beta)n}$ where $H_2(p) = -p\log p - (1-p)\log(1-p)$, we get that the number of strings obtained from X with at most $n/6$ edit operations is at most

$$\binom{n}{n/6} 5^{n/6} \leq 2^{(H_2(1/6)+1/2)n} = o(2^n).$$

□

We define two distributions $\mathcal{F}$ and $\mathcal{C}$ on pairs of strings. Let $\mathcal{F}$ be a distribution on pairs of random n-bit strings. Let $\mathcal{C}$ be a distribution on pairs of n-bit strings (A, B) where A is random and B is a right shift of A by t positions for a random $t \in [n^\alpha/2, n^\alpha]$. By lemma 8, with high probability, $\mathcal{F}$ produces a pair of strings with edit distance at least $n/6$ while $\mathcal{C}$ is over pairs of strings with edit distance at most n^α. The following lemma shows that any low complexity algorithm cannot distinguish these two distributions.

LEMMA 9. *Fix $q < \frac{1}{2}n^{\alpha/2}$. Then for every q-query algorithm $\mathcal{A}$,*

$$\left| \Pr_{x \leftarrow \mathcal{F}}[\mathcal{A}(x) = 1] - \Pr_{x \leftarrow \mathcal{C}}[\mathcal{A}(x) = 1] \right| \leq \frac{1}{2}.$$

PROOF. Let $\mathcal{A}$ be a q-query algorithm that has access to two n-bit strings. Namely, $\mathcal{A}$ is a (possibly probabilistic) mapping from query-answer histories

$$[(i_1, s_1, a_1), \ldots, (i_h, s_h, a_h)]$$

to (i_{h+1}, s_{h+1}) for $h < q$, and to $\{\mathrm{C}LOSE, FAR\}$ for $h = q$. A query of $\mathcal{A}$ is in the form (i_h, s_h), where $1 \leq i_h \leq n$ and $s_h \in \{\text{'A'},\text{'B'}\}$, to denote the i_h^{th} location of the string s_h. An answer a_h is the corresponding bit.

Let $H_\mathcal{F}$ denote the distribution on query-answer histories of length q of $\mathcal{A}$ on inputs selected from $\mathcal{F}$. Define $H_\mathcal{C}$ similarly. It is enough to show that $||H_\mathcal{F}, H_\mathcal{C}||$ (the statistical difference between $H_\mathcal{F}$ and $H_\mathcal{C}$) is $o(1)$.

A query (i_h, s_h) (under input distribution $\mathcal{C}$) is called *revealing* when the queried bit was already queried in the other string, i.e., if $s_h =$'A' and the algorithm has already queried $(i_h + t,$'B') or if $s_h =$'B' and the algorithm has already queried $(i_h - t,$'A'). If $\mathcal{A}$ ever asks a revealing query, we assume it outputs "CLOSE". This assumption only makes $\mathcal{A}$'s job easier.

We assume, without loss of generality, that $\mathcal{A}$ does not repeat queries. Under this assumption, each answer under $\mathcal{F}$ is always a random bit. Conditioned on the event that $\mathcal{A}$ does not make a revealing query, each answer under $\mathcal{C}$ is also a random bit. By showing an upper bound on the probability that $\mathcal{A}$ makes a revealing query in its computation, we upper bound $||H_\mathcal{F}, H_\mathcal{C}||$.

Consider queries of $\mathcal{A}$ under $\mathcal{F}$. After h queries, none of which are revealing, pairs of bits $(A[i], B[j])$ for at most $h^2/4$ offsets $|i-j|$ have been queried. So, there are at least $(n^\alpha/2) - h^2/4$ shifts that have not been checked. At most h of these remaining shifts can be checked by the new query. Thus, at this point, the probability of a revealing query is at most $4h/(2n^\alpha - h^2)$. Therefore, the probability that a revealing query is made in any sequence of $q < \frac{1}{2}n^{\alpha/2}$ queries is at most

$$\begin{aligned}
\sum_{h=1}^{q} \frac{4h}{2n^\alpha - h^2} &< \sum_{h=1}^{q} \frac{4h}{2n^\alpha - \frac{1}{4}n^\alpha} \\
&\leq \frac{32}{7} n^{-\alpha} \sum_{h=1}^{q} h \\
&\leq \frac{32}{7} n^{-\alpha} \cdot \frac{1}{8} n^\alpha < \frac{1}{2}
\end{aligned}$$

□

Theorem 3 follows from Lemmas 8 and 9.

5. REFERENCES

[1] W. Chang and E. Lawler. Approximate string matching in sublinear expected time. In *Proceedings of the 31st IEEE Annual Symposium on Foundations of Computer Science*, pages 116–124, Saint Louis, Missouri, 1990. IEEE Computer Society Press.

[2] R. Cole and R. Hariharan. Approximate string matching: A simpler faster algorithm. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 463–472, San Francisco, California, Jan. 1998.

[3] G. M. Landau and U. Vishkin. Introducing efficient parallelism into approximate string matching and a new serial algorithm. In *Proceedings of the Eighteenth annual ACM Symposium on Theory of Computing*, pages 220–230, Berkeley, California, May 1986. ACM Press, New York.

[4] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20:18–31, 1980.

[5] E. W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4/5):345–374, Oct./Nov. 1994.

[6] S. C. Sahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. In *37th Annual Symposium on Foundations of Computer Science*, pages 320–328, Burlington, Vermont, Oct. 1996. IEEE Computer Society Press.