Communication complexity of document exchange

Graham Cormode Mike Paterson Süleyman Cenk Ṣahinalp * Uzi Vishkin †

Abstract

We have two users, A and B, who hold documents x and y respectively. Neither of the users has any information about the other's document. They exchange messages so that B computes x; it may be required that A compute y as well. Our goal is to design communication protocols with the main objective of minimizing the total number of bits they exchange; other objectives are minimizing the number of rounds and the complexity of internal computations. An important notion which determines the efficiency of the protocols is how one measures the distance between x and y. We consider several metrics for measuring this distance, namely the Hamming metric, the Levenshtein metric (edit distance), and a new LZ metric, which is introduced in this paper. We show how to estimate the distance between x and y using a single message of logarithmic size. For each metric, we present the first communication-efficient protocols, which often match the corresponding lower bounds. A consequence of these are error-correcting codes for these error models which correct up to d errors in n characters using $O(d \log n)$ bits. Our most interesting methods use a new *histogram transformation* that we introduce to convert edit distance to L_1 distance.

*Department of Computer Science, University of Warwick, Coventry, UK; *grahamc,msp,cenk@dcs.warwick.ac.uk*
†Dept of Electrical Engineering and UMIACS, University of Maryland, College Park, USA; *vishkin@umiacs.umd.edu*

1 Introduction

This paper focuses on the communication issues related to managing documents shared by physically separated users. Let A and B be two such users holding documents x and y respectively. Neither A nor B has any information about the document held by the other. Their goal is to exchange messages so that B computes x; it may be required that A compute y as well. Our objective is to design efficient communication protocols to achieve this goal. We measure the efficiency of a protocol in terms of the total number of bits communicated, as per Yao's model of communication complexity. We are also interested in minimizing the number of rounds (changes of direction in communication) as well as minimizing the running time of the internal computations performed by A and B.

A trivial but inefficient way of exchanging documents is for A and B to send x and y to each other in full. On the other hand, if A knew y (in addition to x), A could communicate x to B more efficiently by sending a sequence of operations for converting y to x. In this paper we consider protocols in which the set of operations to convert one string into another is pre-determined between the users. Given the set of operations S, the minimum number of operations required to convert x into y defines a respective distance (in most cases a metric) between x and y. This distance between x and y imposes a lower bound on the length of the message that A sends to B.

We consider three different distance measures, namely, the Hamming metric, the Levenshtein metric (edit distance), and a new LZ metric that we introduce in this paper. Computing the Hamming differences between two binary strings x and y is equivalent to computing the binary string z, where $z[i]$, the i^{th} bit of z, is equal to $x[i] \oplus y[i]$, and $\oplus$ denotes the exclusive-or operation. An obvious deterministic lower bound on the communication complexity of computing the bitwise exclusive-or of two binary vectors of size n is n bits. Hence, any protocol that computes the Hamming distance correctly for all pairs of strings x and y will need to communicate $\Omega(n)$ bits. In fact, this lower bound holds even if the protocol computes the Hamming distance with any constant probability, as shown by Pang and El Gamal [PG86]. For more general, "balanced" measures [1] of string distance $d(x, y)$, the first protocols for exchanging documents are given by Orlitsky [Orl91]. These protocols assume a fixed upper bound f on $d(x, y)$ and they require the transmission of $\Omega(f)$ bits. If $d(x, y) > f$, then the protocols result in an error; if $d(x, y) = o(f)$ then we show in this paper that too many bits are transmitted. [2]

Contributions. Our most interesting protocols involve only two rounds of communication. The first round involves estimating $d(x, y)$. Here B computes and sends A a $\log |y|$ bit random signature of y. This signature enables A to compute an estimate $\hat{d}(x, y)$ for $d(x, y)$. For the Hamming metric this signature is computed by hashing a sequence of random bit samples from y of increasing size (see Section 3). For the LZ metric (and the Levenshtein metric) we introduce a new *histogram transformation*, which B applies

[1] Balanced measures include the ones which are symmetric, such as the edit distance.

[2] On a more practical side, Metzner [Met83, Met91], Abdel-Ghaffar and El Abbadi [AGE94] and Barbará and Lipton [BL91] consider the problem of identifying different pages between two copies of an updated file. It is assumed that the differences between pages are aligned with the page boundaries, effectively resulting in Hamming differences. They also assume a known upper bound f on the number of pages that are different, as per the framework in [Orl91]. In particular [AGE94] describes a protocol based on error-correcting codes is given; this protocol sends a single message of $O(f \log n)$ bits to correct up to f differences.

on y before computing its signature (see Section 4). The most interesting property of this transformation is that it converts the LZ (metric) distance to L_1 distance (which is essentially handled like the Hamming distance, see Section 4.1), hence potentially provides the key to solve a number of open problems, most notably in nearest neighbor search [Ind99]. The second round involves the actual exchange of documents via correcting the differences (see Section 5). Here A computes and sends B a deterministic ID for x which is unique among all strings that are in the $\hat{d}(x, y)$ neighborhood of x. By the use of known techniques in hypergraph coloring or error-correcting codes, the size of this ID can be bounded above by $c \log |y| \cdot \hat{d}(x, y)$ bits, where c is a small constant. These protocols work correctly with probability $1 - \epsilon$, a user specified confidence parameter. For Hamming distance, their communication complexity is $c' d(x, y) \log |x|$, where c' is a constant determined by ϵ only. For the LZ metric or the Levenshtein metric, their communication complexity becomes $c' d(x, y) \log^3 |x|$.

It is possible to save a factor of $\log^2 |x/d|$ in the communication complexity of above protocols by spreading the distance estimation to $O(\log d(x, y))$ rounds. In a given round r, A again computes and transmits to B a unique ID for x in its 2^r-neighborhood, consisting of 2^r bits. B computes a candidate $\hat{x}$, which is identical to x if $d(x, y) \leq 2^r$, and sends A a fingerprint of $\hat{x}$ for verification.

We also provide simpler protocols that search the differences between x and y in divide-and-conquer fashion (see Section 6). These protocols make extensive use of fingerprints for searching and comparing substrings of x and y. They require $O(d(x, y) \cdot \log |x|)$ rounds and have a communication complexity of $O(d(x, y) \cdot \log |x|)$; their main advantage is that the computations involved are faster.

2 Preliminaries

We first present the preliminaries and state some lower bounds related to our problems.

2.1 Measuring string distance

The Hamming distance between two strings x and y, denoted $h(x, y)$, is of interest when single characters are altered only. This distance is insufficient to capture more general scenarios. If we are comparing an edited document to an earlier version, a better measure is the edit (or Levenshtein [Lev66]) distance, denoted $e(x, y)$. Here, three basic operations, namely inserting, deleting, and replacing single characters, are each assigned unit cost. The distance between two strings is then the minimal number of edit operations required to transform one into the other. Clearly, $e(x, y) \leq h(x, y)$. Edit distance still does not fully capture the appropriate measure of distance for many circumstances; e.g. a paragraph may be moved within a document, or a long sequence of DNA may be copied. Motivated by analogy with Lempel-Ziv data-compression algorithms, we define a new measure between strings below.

Definition 1 *The LZ distance between strings x and y, denoted $lz_D(x, y)$ is the minimum number of single characters or substrings of y or of the partially built string which are required to produce x.*

e.g. Suppose x = aabcdefgabc and y = aaaaaaaaaaaaaaaa. Then $lz_D(x,y) = 8$ and $lz_D(y,x) = 4$. This can be seen by forming appropriate parsings of the strings: x = (aa)(b)(c)(d)(e)(f)(g)(abc), and y = (aa)(aa)(aaaa)(aaaaaaaa). As $lz_D(x,y) \neq lz_D(y,x)$, the LZ distance does not provide a metric. We now describe a companion distance measure which is similar to LZ distance but provides a metric.

Definition 2 *The LZ metric distance between two strings x and y, denoted $lz_M(x,y)$, is defined as the minimum number of operations of copying a substring, deleting a repeated substring, or inserting, deleting or changing a single character to transform x into y.*

Lemma 1 *The LZ metric distance provides a metric on finite strings from any constant size alphabet.*

Proof. The lemma follows from the observation that any distance defined by assigning unit cost to a number of reversible operations and counting the shortest sequence of such operations as the distance will necessarily define a metric. □

Intuitively, the LZ metric measures the amount of information that must be exchanged for two parties to know each other's string. If an oracle knows both x and y and wishes to broadcast a single message so that someone with y could calculate x and vice-versa, the minimum size of this message is $O(lz_M(x,y)\log(|x|+|y|))$. Exact computation of these two new distances is NP-hard. We are interested in considering the success of our protocols against the measures, and approximating the distance.

Lemma 2 $lz_M(x,y) \leq lz_D(x,y) + lz_D(y,x)$.

Proof. Consider a particular transformation of x into y which extends x to the string xy by successive 'copy' operations at the right hand end, and then removes x using 'deletes' from right to left through x. The number of copies required for this transformation is precisely $lz_D(x,y)$, and the number of deletions is precisely $lz_D(y,x)$. □

Lemma 3 $lz_D(x,y) \leq 2e(x,y)+1$ *and* $lz_M(x,y) \leq e(x,y)$.

Proof. If $e(x,y) = k$ then y can be parsed into $k+1$ substrings from x separated by at most k single characters. The first inequality follows. The second is trivial. □

Remark. Lemma 3 shows that any protocol that is communication efficient in terms of LZ metric distance is also communication efficient in terms of the edit distance as well, up to a factor of 2. In general we do not treat the protocols for edit distance separate from the protocols for LZ metric distance. Some exceptions are made for cases that can be improved by a constant factor.

2.2 Lower bounds on communication

To examine the performance of the proposed protocols, we need a lower bound on the amount of communication required. The obvious lower bound is the number of bits that would be required for B to compute

x given y. If A already knew y in addition to x, A could send a single message containing sufficient information to derive x from y. If, for any y, there are at most m possible strings for x, then $\lceil \log m \rceil$ bits are necessary. [3]

To correct Hamming errors, we must identify the location of each error, and send the correct character for that location. A careful encoding with a finite alphabet Ξ can be achieved using at most $h \log(|\Xi| n/h) + 2h \log \log n$ bits, for $n = |x| = |y|$ and $h = h(x, y)$. Since $h \leq n$, this is $O(n)$ for a constant-sized alphabet.

For the Levenshtein metric, it is possible to use a bit flag to denote whether each edit is an insertion or a change, and use $\log |\Xi|$ bits to code the character concerned. In the case of a deletion, this is represented as a 'change', but using the code of the existing character at that location. Using a similar location coding as before gives this scheme an overall cost of $e \log(|\Xi|(n + 1)/e)$, for $e = e(x, y)$. Here, n denotes the length of the longer string.

For the LZ metric, we again show an approximation to the lower bound using an encoding of the operations. A 'copy' operation requires $\log n$ bits to specify each of the start, length of substring and destination. Provided n is more than $|\Xi|$, for $lz_M(x, y) = l$ we require no more than $3l \log n^3$ bits[4] .

3 Distance Estimation: Hamming Metric

In this section we show how one of the communicating parties, say B, computes an estimate of the Hamming distance between its binary string y and the string x of the other party, A. Denote by n the length of x (and y), and denote by $\hat{h}(x, y)$ the estimate of $h(x, y)$ computed by B. We will need only a single round of communication during which B receives a $O(\log n)$ bit signature of x from A. In Section 5 we describe how this estimate is used by B to compute and send A a unique ID for y, which can be decoded by A. This ID will consist of $O(\hat{h}(x, y) \log n)$ bits hence it is important not to over-estimate $h(x, y)$.

The protocol is randomized and can result in an under-estimation of $h(x, y)$ with probability no more than a user specified confidence parameter ϵ. Because an under-estimation will result in A computing y erroneously, we would like to keep ϵ small. We demonstrate that the probability of over-estimating $h(x, y)$ by more than a constant factor (which we will specify later) is small as well.

The signature of x is computed in $O(\log n)$ iterations as follows. We assume that two identical random number generators are accessible to both parties A and B. This assumption can be realized by using standard random number generators which are widely available; A may include the seed in the message. In the i^{th} iteration, A uses its random number generator to uniformly and independently pick a sequence of β^i integers from the range $\{1, \ldots, n\}$, denoted by $r_1, \ldots, r_{\beta^i}$ for some $\beta > 1$ which will be specified later. This sequence (which is available to B as well) is then used to obtain the sample $\hat{x}_i = x[r_1], \ldots, x[r_{\beta^i}]$ from x, which is then hashed to $m_i(x) = \bigoplus_{j=1,\ldots,\beta^i} x[r_j]$ as per Lemma 13 of [AMRT96]. The message of

[3] For our distance measurements, such quantities are difficult and cumbersome to evaluate exactly, so instead we shall approximate the quantities from above, by considering the size of a simple encoding of such a message.

[4] This follows from observing that there is no need for the length of any intermediate string to contain more than one copy of any substring of x or y. The total length of such substrings is $O(n^3)$.

A will then be $m(x) = m_1(x), \ldots, m_{\log_\beta n}(x)$ and hence will include a total of $\log_\beta n$ bits.

At the same time B computes $m(y) = m_1(y), \ldots, m_{\log_\beta n}(y)$ by using the same procedure. After receiving $m(x)$, B computes $m(x) \oplus m(y) = m_1(x) \oplus m_1(y), \ldots, m_{\log_\beta n}(x) \oplus m_{\log_\beta n}(y)$. Then, B uses the smallest i for which $m_i(x) \oplus m_i(y) = 1$ to compute an estimate $\hat{h}(x,y)$ for $h(x,y)$. Below, we first provide some tools to analyze the probability distribution of $m(x) \oplus m(y)$ and then show how B computes its estimate.

Observe that $m_i(x) \oplus m_i(y) = \bigoplus_{j=1,\ldots,\beta^i} x[r_j] \oplus y[r_j]$. Therefore $\Pr(m_i(x) \oplus m_i(y) = 1)$ is equal to the probability of getting an odd parity out of β^i random i.i.d. bits, each of which is 1 with probability $p = h(x,y)/n$. This can easily be computed by the following well known lemma, for which we provide a simple new proof.

Lemma 4 *Let $b_1, \ldots, b_k$ be independent Boolean random variables with expectation p.* $\Pr[\sum b_i$ *is even* $] = (1 + (1-2p)^k)/2$.

Proof. Let $e_j = \Pr[\sum b_i = j]$. Then $e_j = \binom{k}{j} (1-p)^{k-j} p^j$. The generating function for e_j can be computed as $E(x) = \sum e_j x^j = (1-p+px)^k$, so $e_0 + e_2 + \ldots = (E(1) + E(-1))/2 = (1 + (1-2p)^k)/2$. $\Box$

Denote $C(k,p) = (1 + (1-2p)^k)/2$. We show that $C(k,p)$ can be tightly bounded above and below as follows.

Lemma 5 *For all $k \geq 2$, and $0 \leq p \leq 1$: (i) $e^{-kp} \leq C(k,p) \leq (1 + e^{-2kp})/2$;*
(ii) if $e^{-\frac{2}{3}kp} \geq \frac{\sqrt{5}-1}{2}$, i.e., $kp \leq 0.721818\ldots$, then $C(k,p) \leq e^{-\frac{2}{3}kp}$

Proof. (i) Since $e^{-kp} = C(k,p) = 1$ for $p = 0$, we can show $e^{-kp} \leq C(k,p)$ by proving $\frac{d}{dp} e^{-kp} \leq \frac{d}{dp} C(k,p)$, i.e., $e^{-kp} \geq (1-2p)^{k-1}$, for $k \geq 2$, or $kp \leq -(k-1)\log(1-2p)$. However, $-(k-1)\log(1-2p) = (k-1)(2p + \frac{(2p)^2}{2} + \ldots) \geq (k-1)2p \geq kp$. For the right-hand inequality, $(1-2p)^k \leq e^{-2pk}$ since $1 - 2p \leq e^{-2p}$.

(ii) We want to show that $(1 + e^{-2kp})/2 \leq e^{-\frac{2}{3}kp}$ under the given inequality. Putting $y = e^{-\frac{2}{3}kp}$, we want the inequality $1 + y^3 - 2y = (1-y)(1-y-y^2) \leq 0$ when $1 \geq y \geq \frac{\sqrt{5}-1}{2}$. This is easily checked. $\Box$

Consider the process of taking the parity of geometrically increasing sequences of random samples of bits. The ratio between successive sample sizes is $\beta > 1$. Suppose we have such a sequence increasing up to size k. Define $Z(k,p,\beta)$, the probability that all k parities are zero, to be $\prod_{i \geq 0}^{k} C(k/\beta^i, p)$.

Lemma 6 $Z(k,p,\beta) \geq e^{-kp/(1-1/\beta)}$*; and* $Z(k,p,\beta) \leq e^{-2kp/3(1-1/\beta)}$ *provided* $e^{-\frac{2}{3}kp} \geq \frac{\sqrt{5}-1}{2}$.

Proof. Immediate from the definition of $Z(k,p,\beta)$ by using the preceding lemma. $\Box$

The nature of our protocol is such that it *fails* if B *underestimates* p, i.e., if this first sample of odd parity (at k) occurs later than we would expect. For any $\epsilon > 0$, B will make a cautious estimate $\hat{p}$ $(= \hat{h}(x,y)/n)$

by choosing $\hat{p}$ such that $Z(k/\beta, \hat{p}, \beta) \leq \epsilon$, since the probability of not getting odd parity before k is at most ϵ if $p \geq \hat{p}$. We will satisfy $Z(k/\beta, \hat{p}, \beta) \leq \epsilon$ by taking $e^{\frac{-2k\hat{p}}{3(\beta-1)}} = \epsilon$, i.e. $\hat{p} = \frac{3(\beta-1)\ln 1/\epsilon}{2k}$, and ensuring $e^{\frac{-2k\hat{p}}{3}} \geq \frac{\sqrt{5}-1}{2}$. The latter holds if we choose β so that $1 < \beta \leq 1 + \frac{1}{\ln 1/\epsilon}\ln(\frac{\sqrt{5}-1}{2})$.

As a result, the protocol estimates $h(x, y)$ to be $\hat{h}(x, y) = \hat{p} \cdot n = n \cdot \frac{3(\beta-1)\ln 1/\epsilon}{2k}$, and ensures that the probability of an underestimation is at most ϵ.

Lemma 7 *In the protocol* $\Pr(\hat{p} \geq (1+\alpha)p) < 1/2$*, i.e., the probability of overestimation of p by a factor of* $1+\alpha$ *is small, if* $\alpha \geq \frac{3\beta \ln 1/\epsilon}{2\ln 2} - 1$.

Proof. The protocol's estimate for p is $\hat{p} = \frac{3(\beta-1)\ln(1/\epsilon)}{2k}$. Let $k' = \frac{3(\beta-1)\ln(1/\epsilon)}{2p}$. Then $\Pr(\hat{p} \geq (1+\alpha)p)$ is equal to $\Pr(\frac{3(\beta-1)\ln(1/\epsilon)}{2k}) \geq \frac{3(1+\alpha)(\beta-1)\ln(1/\epsilon)}{2k'})$. But this is equal to $\Pr(k' \geq (1+\alpha)k) = Z(k'/(1+\alpha), p, \beta)$. As $Z(k'/(1+\alpha), p, \beta) \geq \exp\left(\frac{k'p}{(1+\alpha)(1-1/\beta)}\right)$, if we plug in the value of k', we obtain $1 - Z(k'/(1+\alpha), p, \beta) \leq 1 - e^{\frac{-3\beta\ln(1/\epsilon)}{2(1+\alpha)}}$. If we set the right hand side to be less than $1/2$, then $\ln 2 \geq \frac{3\beta\ln(1/\epsilon)}{2(1+\alpha)}$ and hence $\alpha \geq \frac{3\beta\ln(1/\epsilon)}{2\ln 2} - 1$. $\square$

The communication complexity of the distance estimation protocol is $O(\log_\beta n)$, which is $O(\log(1/\epsilon)\log n)$, and involves a single round of communication. By increasing the number of rounds to $\log_\beta \hat{h}(x, y)$, the communication complexity of this protocol can be reduced to $\log_\beta \hat{h}(x, y)$ which is bounded above by $\log_\beta(1+\alpha)h(x, y)$ with probability greater than $1/2$.

Generalization to L_1 metric It is not hard to generalize the above protocol for estimating the L_1 distance between integer vectors; in fact the L_1 distance is identical to Hamming distance for binary vectors. For the sake of brevity we leave the proof of the following lemma to the full paper.

Lemma 8 *B can compute an estimate for $||x, y||_1$ by using a random signature of x sent by A which consists of $\log n$ bits. The estimate will be less than $||x, y||_1$ with user-specified probability ϵ and will be no more than $(1+\alpha)||x, y||_1$ with probability at least $1/2$ for $\alpha \geq \frac{3\beta\ln(1/\epsilon)}{2\ln 2} - 1$.*

4 Distance Estimation: LZ Metric

Our protocol for estimating $lz_M(x, y)$ is based on a transformation that we introduce which converts the LZ metric to an L_1 metric. Before we give the details of the transformation, we first focus on a simpler metric that we call the limited LZ metric and provide a transformation to convert the limited LZ metric to L_1 metric. This will provide the intuition behind the more general transformation which we describe later.

4.1 Converting Limited LZ Distance to L_1 Distance

Definition 3 *Let x and y be two binary strings of length $n = 2^k$. Define the* limited LZ distance *between x and y, denoted $ltd(x, y)$, as the minimum number of the following "edit" operations applied on x to obtain*

y: (1) changing any given bit of x, (2) swapping any two non-overlapping substrings $x[i : i+k-1]$ and $x[j : j+k-1]$ for which the least significant $\lceil \log(k-1) \rceil$ bits of the binary representations of i and j are the same. We call such substrings aligned. *Notice that if we apply on y the reversal of the operations applied on x, we obtain x back; so $ltd(x,y) = ltd(y,x)$, and hence $ltd(\cdot,\cdot)$ is a metric.*

Given a binary string z of size $n = 2^k$, let the 2^i-histogram of z, denoted $lT_{2^i}(z)$, be the integer vector of size 2^{2^i}, whose j^{th} dimension, denoted $lT_{2^i}(z)[j]$, is defined to be the number of substrings $z[l2^i + 1 : (l+1)2^i]$ (for $l = 0, \ldots, n/2^i - 1$) which are equal to the binary representation of j. (E.g. let $z = 1110$; to compute $lT_2(z)$ we consider only the substrings $z[1:2] = 11$ and $z[3:4] = 10$ and deduce that $lT_2(z)[00] = 0$, $lT_2(z)[01] = 0$, $lT_2(z)[10] = 1$, $lT_2(z)[11] = 1$ and hence $lT_2(z) = (0,0,1,1)$.)

Definition 4 *Let $lT^c_{2^i}(z)$ be the concatenation of all $lT_{2^j}(z)$ for $j = 0, \ldots, i$. Then we define the* limited histogram transformation *of z to be $lT^c_{2^k}(z)$ and denote it by $lT(z)$.*

Definition 5 *We define the* limited parse tree of a string *z of size 2^k as the complete binary tree of depth k whose nodes are labeled with substrings of z. The root of the tree is labeled with z itself. The labels of the children of any given internal node with label w are $w[1 : |w|/2]$ and $w[|w|/2 + 1 : |w|]$.*

Theorem 9 $ltd(x,y) \leq ||lT(x), lT(y)||_1 < 8k \cdot ltd(x,y)$

Proof. (i) $||lT(x), lT(y)||_1 \leq 8k \cdot ltd(x,y)$: We show that each operation affects fewer than $8k$ components of $||lT(x), lT(y)||_1$. Changing one bit of x changes the value of two components of $lT_{2^i}(x)$ for all $0 \leq i \leq k$. Consider a swap of two substrings length $L > 1$, and let $l = \lfloor \log L \rfloor$. In the limited parse tree, we count the nodes which cover part of a substring to be swapped and part of the string which remains static. Each such node correponds to the change of at most two components of $lT(x)$ (since one count can increase by one at the cost of another). The number of such nodes is upper bounded by $2(k+l) - 1$ (count k from the left side and l from the right side of both substrings), which shows that each swap affects less than $8k$ components of $lT(x)$.

(ii) $ltd(x,y) \leq ||lT(x), lT(y)||_1$: We describe a procedure that obtains y by applying at most $||lT(x), lT(y)||_1$ operations on x. We need the following definition.

Definition 6 *Given binary strings z and y, we define the* marking *of the limited parse tree of z and $lT(z)$ with respect to y as follows. Each node in the (limited parse) tree of z corresponds to a non-zero entry in $lT(z)$. Consider the left-to-right BFS scan of the tree. The following nodes of the tree as well as the corresponding entries of $lT(z)$ are marked. Let w be the label of the node visited during the BFS scan. Let $lT(z)[i_w]$ be the entry of $lT(z)$ that corresponds to w and let m be the number of times $lT(z)[i_w]$ has already been marked. If $lT(z)[i_w] - m > lT(y)[i_w]$ and the parent of the node w is marked, then both the node and the entry $lT(z)[i_w]$ are marked.*

Note that the number of nodes marked in a marking of z relative to y is exactly $\frac{1}{2}||lT(x), lT(y)||_1$.

The procedure for converting x to y has $k+1$ iterations. Let $x_0 = x$ and $x_1, x_2, \ldots, x_{k+1}$ be the updated versions of x after the corresponding iterations; clearly $x_{k+1} = y$. In iteration j, we apply at most $||lT_{2^j}(x_j), lT_{2^j}(y)||_1$ operations on x_j to obtain x_{j+1}. We show that these operations guarantee the following: (i) $||lT_{2^l}(x_{j+1}), lT_{2^l}(y)||_1 = 0$ for all $l \leq j$, and (ii) $||lT_{2^l}(x_{j+1}), lT_{2^l}(y)||_1 \leq ||lT_{2^l}(x_j), lT_{2^l}(y)||_1$ for all $l > j$.

Our inductive assumption at iteration j is that $lT_{2^{j-1}}(x_j)[i] = lT_{2^{j-1}}(y)[i]$ for all $i > 0$. To make $lT_{2^j}(x_{j+1})[i] = lT_{2^j}(y)[i]$ for all i, and satisfy guarantee (i), we will swap the following substrings of x_j which are all of size 2^{j-1}.

Pick any i such that $lT_{2^j}(x_j)[i] > lT_{2^j}(y)[i]$. Observe that this entry should be marked. Find a corresponding node in the (limited parse) tree of x_j which is marked; we are guaranteed that such a node exists. Let the label of this node be w. If $|w| = 1$, then change its single bit (in a single operation). If $|w| > 1$, then consider its "half" substrings $w[1 : |w|/2]$ and $w[|w|/2 + 1 : |w|]$. If w is the only marked node at its level in the tree, then by swapping its two half substrings in a single move we can make a new string in which no strings are marked at that level. Otherwise with at most 2 swaps with half substrings of other marked nodes we can form a new substring in place of w which is not marked. This therefore decreases $||lT_{2^j}(x_j), lT_{2^j}(y)||_1$ by two, and does not create any new marked nodes. Such swaps will always exist, from our inductive assumption. We continue this procedure until no such i remains.

The proof follows from the observations that (1) only marked nodes are involved in the swaps and (2) all ancestors of marked nodes are also marked. Hence each step of this procedure must decrease the L_1 distance between the limited histogram transformations of the obtained string and the target string. $\square$

Remark. The limited histogram transformation of a binary string of size n is an $O(2^n)$ dimensional integer vector which is very sparse - only $O(n)$ of the entries are non-zero. To decrease the size of the signature, we start with a very large sample (of size $O(2^n/poly(n))$ and again increase the sample size by a factor of β in subsequent iterations. This will not affect the communication complexity of the protocols or the confidence bounds provided.

4.2 Converting LZ Distance to L_1 Distance

The procedure for converting the LZ distance to L_1 distance is based on a generalization of the limited histogram transformation by removing the restriction that only the aligned substrings can be swapped. The main tool which enables us to obtain this (general) histogram transformation is the locally consistent parsing technique, LCP [SV94].

Given a string z of size n, its histogram transformation includes the histograms of the *level-i cores* of z for all $i = 0, \ldots, \log n$, computed by LCP [5]. The cores are special substrings that are determined by the size of the alphabet which satisfy the following important property: The number of level-i cores of a given string z is upper bounded by $|z|/2^i$ and is lower bounded by $|z|/3^i - 2(\log^* |z| + 3)$ [SV94]. This enables

[5]Compare this to the limited histogram transformation which comprises of the histograms of length 2^i substrings of the input, for $i = 0, \ldots, \log n$.

us to define the level-i-histogram of z, denoted $T_i(z)$, as the integer vector of size $O((3\log^* n)^i)$, whose j^{th} dimension, denoted $T_i(z)[j]$, is defined to be the number of *all* level-i-cores that are equal to the binary representation of i. Now we define the *histogram transformation of* z as the concatenation of all $T_j(z)$ for $j = 0, \ldots, \log n$, and denote it by $T(z)$. Note that we do not have the restriction of alignment between strings, hence we can accomodate not only aligned swaps but also unaligned swaps, insertions and deletions. This leads us to the following theorem whose proof is left to the full paper.

Theorem 10 $c' \cdot lz_M(x,y) \leq ||T(x), T(y)||_1 \leq c'' \cdot \log^2 n \cdot lz_M(x,y)$ *for constants* c' *and* c''.

5 Correcting Differences

The work here takes a similar approach to that of Orlitsky [Orl91], and proves differently a similar theoretic result, that for certain relations the single message cost is at most twice the optimal. By considering a more restricted class of function, we are able to show a concrete protocol which achieves this.

Let $d(x,y)$ be a metric over a set X defined by the transitive closure of a symmetric relation R which is assigned unit cost – that is, $d(x,y)$ is the minimum n such that $R^n(x,y)$. Suppose that there are two individuals, A who holds $x \in X$ and B who holds $y \in X$, such that $d(x,y) \leq l$ for some known l. The range of d is the natural numbers, and will refer to this as the distance.

Let G_l be the undirected graph whose vertices are X and whose edges are $\{(x,y) : d(x,y) \leq l\}$.

Lemma 11 $\deg(G_{2l}) \leq (\deg(G_l))^2$.

Proof. $\deg(G_{2l})$ corresponds to the greatest number of vertices at a distance $\leq 2l$ from any particular vertex, x. Since the metric is defined by unit cost operations, these vertices will be those which are at most l from all vertices at most l from x. Since from each vertex there are at most $\deg(G_l)$ vertices at distance at most l, the lemma follows. □

Following the model of Orlitsky, we shall employ a hypergraph H whose vertices are the members of X and whose hyperedges are $e_y = \{x : d(x,y) \leq l\}$. Both parties can calculate this hypergraph independently and without communication (if some parameter of the graph is not implicit, A can prepend this information to the message with small additional cost). They can then use a deterministic coloring scheme to color the hypergraph with $\chi(H)$ colors. A's message is the color of x. By the coloring scheme and the specification of the problem, B knows that x must be amongst the vertices in e_y, and can use the color sent by A to identify exactly which it is. We can place upper and lower bounds on the single message complexity by considering this scheme. Since we must be able to distinguish x among the vertices in the hyperedge x, at least $\log(\deg(H))$ bits must be sent overall. To send the color of x, we need $\lceil \log \chi(H) \rceil$ bits [KN97]. So $\log(\deg(H)) \leq S \leq \lceil \log(\chi(H)) \rceil$.

Lemma 12 $\chi(H) \leq \deg(G_{2l})$.

Proof. $\chi(H)$ is the number of colors required to color the vertices of H such that any two vertices which are both at most distance l from any point x are colored differently. Consider y and y' such that $d(x,y) \leq l, d(x,y') \leq l$. Since d is a metric, $d(y,y') \leq d(y,x) + d(x,y') = d(x,y) + d(x,y') \leq 2l$.

Any coloring which ensures that any two points within distance $2l$ have different colors is therefore a coloring of the hypergraph H. A coloring of G_{2l} achieves this, showing that $\chi(H) \leq \chi(G_{2l})$.

On the assumption that G_{2l} is not complete, is connected and has degree of three or more (which will be satisfied by the examples in which we are interested), then $\chi(G_{2l}) \leq \deg(G_{2l})$, and so the lemma follows. □

Let S be the number of bits necessary in a single message to allow B to calculate x (the single message cost).

Theorem 13 $\log(\deg(G_l)) \leq S \leq 2\log(\deg(G_l))$.

Proof. It is clear that $\deg(G_l) = \deg(H)$; in both cases the degree of each vertex corresponds to the number of vertices a distance at most l away. We combine this with Lemma 12 and Lemma 11 into the fact that $\log(\deg(H)) \leq S \leq \lceil\log(\chi(H))\rceil$ to prove the theorem. □

Thus, under certain conditions, a single message can be at worst twice the lower bound on its length for any number of messages. Note that our upper bound is related to the degree of a graph which we can calculate, and algorithms exist which will color a graph with this number of colors; hence it is achievable in practice. The constraints placed upon the metric d may seem restrictive, so we show how our earlier distances (2.1) fit the restrictions.

Definition 7 *A protocol between two communicating parties A and B is non-trivial if, under certain restrictions, it enables A to transmit a given string x using fewer bits than $|x|\log|\Xi|$, where $|\Xi|$ is the alphabet size.*

Hamming distance is a metric on the space of strings of length n over an alphabet Ξ, and is induced by defining a symmetric relation on strings which differ in a single character. For $l < n/2$, G_{2l} is not complete, and for any non-trivial n, G_{2l} will have degree higher than two: $\deg(G_l) = \sum_{i=1}^{l} \left(\begin{array}{c} n \\ i \end{array}\right) (|\Xi| - 1)^i$. There is no simple closed form for this expression, so instead we consider the simple binary encoding for up to l Hamming differences described in Section 2.2. This uses $l\log((|\Xi|-1)(n/l)) + o(l\log\log(n/l))$ bits, hence we have the result that up to l Hamming errors can be corrected using a single message with asymptotic cost at most $2l\log((|\Xi| - 1)(n/l))$.

Edit distance is defined by charging unit cost to insertions, deletions or alterations of a character. Again, for $l < n/2$ (n denotes the length of the longer string), G_{2l} is not complete and will have degree more than two. Also, there is no concise bound on the degree of G_l, but we can use the binary encoding (Section 2.2) for an upper bound on $\log(\deg(G_l))$. We can encode up to e edit operations using at most $e(\log_2(|\Xi|(n+1)/e))$ bits, so $2\log\deg(G_l) \leq 2l\log(|\Xi|(n+1)/l)$ which is non-trivial for $l < n/2$.

LZ metric distance also conforms to the requirements. Using the simple encoding from Section 2.2 we can achieve a single message cost of $18l \log n$ bits, non-trivial for $l \leq \frac{n \log |\Xi|}{18 \log n}$.

Multi-round protocols. It is possible to have multiple round adaptations of these protocols. In the first round the users run the protocol with the assumption that $d(x, y)$ is upper bounded by some small constant. They then use a hash function over their strings to see if they agree. If not, they double the distance, and repeat the protocol, until they agree. This requires at most $\log n$ rounds and but gives a more precise estimate of $d(x, y)$. For the LZ metric and the Levenshtein metric this amounts to a saving of a $\log^2 |x|$ factor in communication complexity.

Remark. While these protocols are communication efficient, they are computationally expensive. For the Hamming distance, an efficient coloring can be achieved by Reed-Solomon codes (see [AGE94]). Since any one-round protocol corresponds to an error-correcting code, an efficient coloring scheme for edit distance or the LZ metric would lead to error-correcting codes for these systems. We have shown concrete methods to construct such codes; it is open whether these codes could be calculated more efficiently.

6 Computationally efficient protocols

We first consider the following abstracted problem: given a string of length n, locate h distinguished characters by using queries of the form whether such a character exists in any specified substring. This can be done by dividing the string into two equal size substrings and querying them. If a substring is free of distinguished characters, then we do not consider it further. Otherwise it is further divided into two each of which is queried inductively. An upper bound on the number of queries is $2h \log(2n/h)$, from considering a worst case even distribution of distinguished characters. Since each query gives us one bit of information, here we are receiving $2h \log(n/h) + O(h)$ bits of information, which is asymptotically only twice the minimum. We will now apply this abstract problem to our various distance measures.

Hamming distance. The problem of locating the Hamming errors, given an upper bound on the number of errors, is similar to the problem of group testing (also observed by Madej [Mad89]). We wish to group samples and perform a test which will return either 'all the same' or 'at least one mismatch'. The differences are that we have an ordering of the samples, given by their location in the string, and that we have to contend with the problem of false negatives We use negative to mean that the test indicates no errors; false negatives are an inevitable consequence of using fewer than n bits of communication. Our tests will be based on fingerprinting; i.e., if a substring in one string hashes to the same value as the corresponding substring in the other, then we take this as evidence that the two substrings match. Note that there is no danger of false positives; if a test leads us to believe that the substrings are different, then there is zero probability that they actually match.

The following scheme is a straightforward way to locate and correct differences, and is similar to that proposed in [Met83] Mapping our terminology onto that of the above problem, the locations of differences between the two strings are our distinguished characters. The test we perform involves communication: in each round, one party will send the value of a hash function for the substrings in question, and the other will calculate the value for the corresponding substrings of their string. If they agree, then there is (with high probability) no difference between the two substrings; otherwise there is (with certainty) some discrepancy between them. The reply to the message is a bitmap indicating the success or failure of each test. We must choose our functions so that the overall probability of success of the procedure is high.

If we use linear hashes relative to a random base B, then the probability of two arbitrarily chosen sequences having the same hash value is $1/B = 1 - P$. If we treat each test as independent, then we want, say $P^{2h(\log 2n/h)} = p_c$.

Lemma 14 *For* $B > 1,\ B > \frac{1}{\ln(\frac{B}{B-1})} > B - 1$.

Proof. $\ln(\frac{B}{B-1}) = -\ln(1 - \frac{1}{B}) > \frac{1}{B}$ and $\ln(\frac{B}{B-1}) = \ln(1 + \frac{1}{B-1}) < \frac{1}{B-1}$. □

Using Lemma 14 allows us to choose B to be at least $\lceil \frac{2h(\log 2n/h)}{\ln 1/p_c} \rceil$. We choose B using $\hat{h}$, our estimate of the Hamming distance, which gives us the following bound on the total number of bits sent in hashes: $2h \log(2n/h) \log B = 2h(\log 2n/h) \log \left(\frac{2\hat{h} \log(2n/\hat{h})}{\ln 1/p_c} \right)$. This protocol is *non-trivial* for $h = O(n/\log n \log\log n)$.

The overall cost is a factor of $O(\log n)$ above the optimal; however, most of this cost comes from the size of the hash values sent. With regard to the number of hash values sent, we have shown it is about twice the optimal of any possible scheme which uses hash functions to give a binary answer to a question. The number of rounds is logarithmic in the length of the string, as we can progress one level of the limited parse tree (5)in each round. Depending on the choice of the hash function, it may be possible to halve the amount of information sent: if linear hashes are sent then the hash value for the second half of a string can be calculated from the hash value of the first half and the whole string. Otherwise, if hashes are independent, then stored values can be used to check that the procedure has succeeded. This applies to all of the hierarchical schemes presented in this section.

Edit Distance. We next translate the above approach to deal the edit distance. A similar scheme is proposed but not analyzed in [SBB90].

It is not immediately apparent how to map the edit distance problem onto the abstract problem presented above. However, observe that the way that differences were located was by a process of elimination: identical substrings were found, until all that remained were disparate substrings. We may use the same kind of hierarchical approach to identify common fragments. It is no longer the case that substrings are aligned between the strings; however, we know that matching substrings will be offset by at most the edit distance.

The party B receiving the hash values must therefore do more work to identify them with a substring. B has a bound d on the edit distance; if substrings are unaltered by the editing process, then they will be found at a displacement of no more than d from their location in A's string. So B calculates hashes of substrings of the appropriate length at all displacements left and right from the corresponding position in its string, testing each one to see if it agrees with the sent hash. If they do agree, it is assumed that they match, and so B now knows the substring at this location in A.

We consider an optimal edit sequence from x to y and how it affects the limited parse tree representing the splitting of x. Deletions and replacements affect single characters at leaf nodes. An insertion of any number of consecutive characters between two adjacent characters is considered to occur at the internal node which is the lowest common ancestor of the pair. The protocol must traverse this tree, computing hashes on every node whose subtree contains any of these edit operations. We observe that the worst case is exactly as before – if (almost) all errors are deletions or alterations (since we have to descend further down the tree to discover these). Certainly, the worst case number of hashes sent will be the same, $2d \log(2n/d)$.

We must compare each hash sent with up to $2d + 1$ others in the worst case. As before, we want the probability of an error to be no more than a constant, which gives $\left(\frac{B-1}{B}\right)^{2d(2d+1)\log(2n/d)} \geq p_c$. Using Lemma 14 gives $B \geq \frac{2d(2d+1)\ \log\ \frac{2n}{d}}{\ln\ 1/p_c}$. The number of bits sent is less than $2d \log(2n/d) \log B$, which is

$$2d \log \frac{2n}{d} \left(2 \log(2d) + \log \frac{\log(2n/d)}{\ln 1/p_c}\right) \leq 4d \log\,(2n/d) \log(2d) + O(d \log n \log \log n)$$

. This expression is $O(d \log(n/d) \log d)$, asymptotically $O(\log d)$ above the optimal. If we have no bound on d, we can use the fact that between two strings of length m and n, the edit distance is at most $\max(m, n)$. Assuming $m \leq n$ gives a cost of at most $4d \log\,(2n/d) \log 2n$; this is *non-trivial* for error rates which are $O(n/(\log n \log \log n))$.

LZ distance. Consider a variation of the above abstract problem. Instead of h distinguished characters, suppose that there are l "dividers" which are positioned between characters, such that there is at most one divider between any two adjacent characters. This problem is reducible to the distinguished character problem: there are $n - 1$ locations where dividers can be placed. If our queries are whether there are any dividers within a substring, then clearly the same protocol will suffice, with the same cost.

Now consider a notional optimal parsing of a string x relative to a string y with $lz_D(x, y) = l$. This means that x is parsed into l pieces, each piece of which is a substring present in y or earlier in x, or a single character. The number of hashes necessary for B who holds y to identify x is that needed to locate the $l - 1$ "dividers" which separate the substrings. This is $2(l-1)\log(2\frac{n-1}{l-1}) < 2l \log(2n/l)$. A modified hierarchical protocol achieves this using many more rounds. Rather than descending the tree in parallel, the protocol performs a depth-first search.

The protocol proceeds as follows. B maintains a dictionary of 'resolved' hashes, containing the mapping from hash values to each substring of length 2^i. This is initialized with all possible hashes over y, and is augmented with information about x as it is learned. [6] On receiving a pair of hashes, B tries to use this dictionary to identify them. B indicates to A how far along x he can identify. A, who maintains a record of the progress of B, sends two hash values on strings made by splitting the first remaining unresolved substring into two equal pieces.

[6] So if B identifies a sub-string of length 16, then not only is the mapping from its hash to the substring added, but so are the 9 substrings of length 8, the 13 substrings of length 4 and the 15 of length 2.

In this protocol, we do many more comparisons between hash values. We will therefore need a larger base over which to compute hashes in order to ensure that the chance of a hash collision is still only constant for the whole process. We only compare hashes for substrings of the same length. If we have two strings each of length n then there are fewer than $2n$ substrings of any given length. So there are at most $2n(2n-1)/2 < 2n^2$ possible pairwise comparisons. Invoking Lemma 14 again leads us to choose $B = \lceil \frac{2n^2}{\ln 1/p_c} \rceil$. Therefore the overall cost in bits is bounded above by $2l \log(2n/l) \log B = 4l \log(2n/l) \log \frac{2n}{\ln 1/p_c}$ which is $O(\log n)$ above the optimal. The number of rounds involved is at most $l \log(n/l)$.

If, instead of the full LZ metric, we only imagine the unknown string to be parsed into substrings of the other string and single characters, then it is straightforward to show a protocol which is $O(\log n)$ above optimal cost under this measure and only uses $\log n$ rounds. In the case where we expect the files to share many substrings, then this approach would be appropriate.

Acknowledgments

The third author thanks Funda Ergun for all the valuable discussions and her help in writing this paper.

References

[AGE94] Khaled A. S. Abdel-Ghaffar and Amr El Abbadi. An optimal strategy for comparing file copies. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):87–93, January 1994.

[AMRT96] Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup. Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{(}\log n/\log\log n))$ is necessary and sufficient. In *37th Annual Symposium on Foundations of Computer Science*, pages 441–450, Burlington, Vermont, 14–16 October 1996. IEEE.

[BL91] Daniel Barbara and Richard J. Lipton. A class of randomized strategies for low-cost comparison of file copies. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):160–170, April 1991.

[Ind99] Piotr Indyk. Private communication. 1999.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[Mad89] T. Madej. An application of group testing to the file comparison problem. In *9th International Conference on Distributed Computing Systems*, pages 237–245, Washington, D.C., USA, June 1989. IEEE Computer Society Press.

[Met83] J. J. Metzner. A parity structure for large remotely located replicated data files. *IEEE Transactions on Computers*, 32(8):727–730, August 1983.

[Met91] J. J. Metzner. Efficient replicated remote file comparison. *IEEE Transactions on Computers*, 40(5):651–659, May 1991.

[Orl91] A. Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In IEEE, editor, *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 228–238, San Juan, Porto Rico, October 1991. IEEE Computer Society Press.

[PG86] King F. Pang and Abbas El Gamal. Communication complexity of computing the Hamming distance. *SIAM Journal on Computing*, 15(4):932–947, 1986.

[SBB90] T. Schwarz, R. W. Bowdidge, and W. A. Burkhard. Low-cost comparisons of file copies. In *10th International Conference on Distributed Computing Systems*, pages 196–202, Paris (France), May–June 1990. IEEE, IEEE Computer Society Press.

[SV94] S. C. Sahinalp and U. Vishkin. Symmetry breaking for suffix tree construction. In *Proceedings of the 26th Annual Symposium on the Theory of Computing*, pages 300–309, New York, May 1994. ACM Press.