Reconciliation Puzzles

Vikas Chauhan, Ari Trachtenberg
{chauhan,trachten}@bu.edu
Boston University Photonics Center
8 St. Mary's Street
Boston, MA 02215

Abstract

We consider the problem of exchanging two similar strings held by different hosts with a minimum amount of communication. We reduce the problem of string reconciliation to a problem of multi-set reconciliation, for which nearly optimal solutions exist. Our approach involves transforming a string into a multi-set of substrings, which are reconciled efficiently and then put back together on a remote host using recent graph-theoretic results. We present an analysis of our algorithm to show that its communication complexity compares favorably to an existing method for string reconciliation.

A version of this article will appear as:

- V. Chauhan and A. Trachtenberg, "Reconciliation puzzles", IEEE Globecom 2004, Dallas, TX.

I. Introduction

We address the problem of reconciling similar strings held by two distinct hosts. Formally, we consider two distinct hosts A and B each with a string σ_A and σ_B, respectively, derived from the same alphabet Ξ. The *string reconciliation* problem is thus for host A to determine σ_B and for host B to determine σ_A with a minimum communication. In the general case, when the two strings are arbitrary, there is no better deterministic solution than simply transmitting σ_A from host A to host B and doing the comparison locally [11]. Thus, to enable more interesting solutions, we shall assume that the two strings differ by at most d edits, defined as character insertions, deletions, or modifications.

The problem of string reconciliation appears in a number of applications, most prominently file synchronization and pattern recognition. Image reconciliation can be thought of as a two (or more) dimensional generalization of string reconciliation, and, in general, string reconciliation algorithms can be used as a basis for reconciling various types of structured data.

We next provide a brief overview of the CPIsync algorithm for set and multi-set[1] reconciliation. Thereafter, we provide a high-level description of our algorithm for string reconciliation, which is based on CPISync. In Section II we survey some of the relevant existing literature. Thereafter in Sections III and IV we provide theoretical underpinnings and implementational details of our algorithms. Finally, we analyze our algorithm in Section V and present experimental results in Section VI.

A. Overview of the CPISync Protocol [1]

The Characteristic Polynomial Interpolation Synchronization protocol [CPISync] is based on an algebraic solution to the problem of reconciling two sets of information S_A and S_B held on two distinct hosts A and B. In the model used by CPISync, a set $S = \{x_1, x_2, x_3, x_4, ..., x_n\}$ is represented by a characteristic polynomial

$$\chi_S(Z) = (Z - x_1)(Z - x_2)(Z - x_3)(Z - x_4)...(Z - x_n)$$

[1] Recall that a multi-set is an unordered collection of elements, some of which may be duplicated.

The key observation of the protocol is that for two sets S_A and S_B,

$$\frac{\chi_{S_A}(Z)}{\chi_{S_B}(Z)} = \frac{\chi_{\Delta_A}(Z)}{\chi_{\Delta_B}(Z)}, \tag{1}$$

where Δ_A is the set difference $S_A \setminus S_B$, and $\Delta_B = S_B \setminus S_A$. This is because terms common to both sets cancel out in the numerator and denominator. Thus, the rational function in Eq. (1) can be uniquely interpolated from m samples, if there are at most m differences between synchronizing sets.

The CPISync protocol can be described as follows:

- Hosts A and B evaluate their characteristic polynomials on m sample points (over a chosen finite field). Host A sends its evaluations to host B.
- The evaluations are combined to compute m sample points of the rational function $\frac{\chi_{S_A}(Z)}{\chi_{S_B}(Z)}$, which are interpolated to determine $\frac{\chi_{\Delta_A}(Z)}{\chi_{\Delta_B}(Z)}$.
- The numerator and denominator of the interpolated function are factored to determine the differences between S_A and S_B.

In the probabilistic version of the algorithm, if host B uses CPISync for set reconciliation, the amount of communication needed reconcile two sets is bounded above by

$$2(b+1)m + b + bm_A + m + k \text{ bits.}$$

In this expression, b is the size (in bits) of each element, m is the symmetric difference between the sets, $m_A = |\Delta_A|$ and $m_B = |\Delta_B|$ are the two components of this symmetric difference ($m = m_A + m_B$) and k is a confidence parameter corresponding to the probability of success of CPISync. Notice that this protocol generalizes nicely to multi-set reconciliation.

B. Reconciliation puzzles

The idea behind our approach to reconciliation is to divide structured data into multi-sets of puzzle pieces. These multi-sets are reconciled using the techniques in [7] and then put together, like a puzzle, to form the original data. The key to this approach is that the algorithm described in [7] does not depend on the cardinality of the reconciling multi-sets, but rather on the number of differences between these multi-sets. Thus, we can form many puzzle pieces so long as the differences between strings translate to a proportional number of differences in puzzle pieces. This approach essentially enables us to break down the ordered data (strings) into unordered data (pieces) without losing the ability to reconstruct the ordered data from the pieces. In this paper, we deal with strings, but this approach can be applied to data that have been organized in two-dimensions too.

C. Strings

In the case of strings, (puzzle) pieces are composed of proximate characters according to masks. Formally, a *mask* is a binary array; applying a mask to a string involves computing a dot product of the mask with a substring of the string. In other words, applying a mask is equivalent to placing the mask over the string, beginning at a certain character, and reading off all characters corresponding to 1 bits in the mask, thus producing one *piece*. To divide a string into pieces, one simply applies the mask at all shifts ($i.e.$ starting at each character) in the string. The following example demonstrates concretely how a string might be broken up into puzzle pieces.

Consider the string 01010010011 under the mask 111, which has length $l_m = 3$. We artificially provide the string with anchors (*i.e.,* characters not in the string alphabet) at the beginning and end of the string, in this case the character "$". The resulting string would be $\$01010010011\$$ and the puzzle pieces from the string would be:

$$\{\$01, 010, 101, 010, 100, 001, 010, 100, 001, 011, 11\$\}.$$

To reconcile two different strings, one would first break up both strings into multi-sets of pieces, and then reconcile these multi-sets using the algorithm described in [7]. Once the multi-sets are reconciled, each host knows what pieces that other host has and tries to decode these pieces into a string.

As mentioned earlier, the key observation is that, though there are many pieces, edit changes will only affect a small number of pieces corresponding to masks that are applied within a small vicinity of the changes. The analysis will be presented in the next section.

On the encoding side there are several conflicting optimization criteria:

- Mask length: The mask length determines the number of pieces affected by an insertion or deletion edit, and should be made as small as possible.
- Unique decoding: If the mask is too small, then the puzzle pieces do not uniquely determine the encoded string. For example, the pieces from the above example can be decoded to one of the two possible strings:
  1) $\$01001001011\$$
  2) $\$01010010011\$$

II. Previous Work

Orlitsky [8] presented some information theoretic bounds on the amount of communication needed for exchanging documents modeled as random variables with a known joint distribution. He also proposed an efficient one-way protocol for reconciling documents that differ by at most α string edits. In Orlitsky's model, host P_X holds string X which differs from the string Y held by P_Y by at most α edits. It is shown that host P_X requires at least

$$\alpha \log |X| - o(\alpha \log |X|) \text{ bits}$$

of communication to communicate string X to host P_Y. Orlitsky and Viswanathan [9] also present an efficient protocol for transmitting a file X to another host with a file Y, an edit distance k from X. This protocol succeeds with probability $1 - \epsilon$ and requires the communication of at most

$$2k \log |X| \left(\log |X| + \log \log |X| + \log \left(\frac{1}{\epsilon} \right) + \log k \right) \text{ bits.}$$

Cormode, Paterson, Ṣahinalp and Vishkin [2] have also proposed protocols that minimize communication complexity in reconciling similar documents and have come up with bounds on these values based on the Hamming, LZ and Levenshtein metrics. In their protocols, the amount of communication needed to correct an edit distance of d between two strings is upper bounded by

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

where n is the length of the string, $\hat{d}$ is the bound on the edit distance and p_c is the desired probability of success. They also show how to identify different pages between two copies of an updated file using error-correcting codes, a problem addressed earlier by Barbara and Lipton and also by Abdel-Ghaffar and El Abbadi using Reed-Solomon codes [10].

Recently, Evfimievski [3] has also presented a probabilistic algorithm for communicating an edited binary string over a communication network with arbitrarily low probability of error. The upper bound for the communication complexity in [3] is

$$1000k^2 \log |y| (16 + \log |x| + 5 \log |y| + \log(\epsilon^{-1})) \text{ bits,} \tag{2}$$

where k is the edit distance, $|x|$ and $|y|$ are the lengths of the strings and ϵ is the error probability.

III. Encoding and decoding algorithms

We now describe the algorithms used to encode and decode strings using the above mentioned concept of masks. Let us say that we have two hosts A and B which have the same copy of string σ initially, and that $\sigma = 01101010011$. Host A makes some changes to σ and the modified string is $\sigma' = 01010011$. Host B makes a change to σ and let us say that the modified string is $\sigma'' = 01010010011$. The problem now is to make each host know the other host's copy of the string.

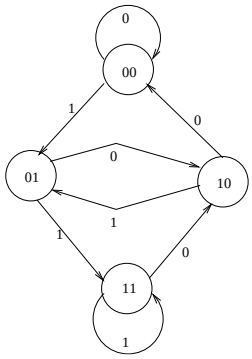

Fig. 1. The de Bruijn digraph $G_3(\{0,1\})$

Note that we assume that neither host knows the original string that they have in common. This corresponds to the most general application scenario. It occurs, for example, if the hosts were given their strings by a third party, or if the strings were derived by some foreign process that generates related output.

We will encode the strings at each host to make a multi-set of pieces and synchronize these sets using CPISync [7]. Once we know the multi-sets corresponding to the other host's string, we use a decoding algorithm to put these pieces back together into the original string. We will have to be careful to ensure that the pieces can be put together in only one meaningful way.

A. Encoding

For purposes of illustration, let us use a mask $m = 111$ for the encoding and decoding. This mask will be communicated to the other host while synchronizing. At host A, $\sigma' = 01010011$ will first be padded on both sides with an anhor (*i.e.,* "$") and then encoded as $S_A = \{\$01, 010, 101, 010, 100, 001, 011, 11\$\}$. At host B, σ'' will be encoded as $\sigma_B = \{\$01, 010, 101, 010, 100, 001, 010,$
$100, 001, 011, 11\$\}$. The two sets will be synchronized using a version of CPISync, resulting in communication roughly linear in the number of differences between the sets. In addition to the set reconciliation communication, both sets would also be sending each other the mask used for encoding, and another number corresponding to the decoding, which we will discuss later. In practice, we hash each piece together with the number of copies of the piece in the multi-set, and synchronize these resulting hash sets.

B. Decoding

1) The de Bruijn Digraph: The *de Bruijn* digraph $G_{l_m}(\Xi)$ for an alphabet Ξ and a length l_m contains $|\Xi|^{l_m-1}$ vertices, each corresponding to an length $(l_m - 1)$ string over the alphabet. There will be an edge from vertex v_i to v_j labeled l_{ij} if the string associated with v_j contains the last $l_m - 2$ characters of v_i followed by l_{ij}. Thus, each edge (v_i, v_j) represents a string defined by the label of v_i followed by l_{ij}. The de Bruijn digraph $G_3(\{0,1\})$ is shown in Fig. 1

2) Modified de Bruijn Digraph: The problem of determining the original string from a multi-set of pieces is equivalent to finding the correct Eulerian path in a modified *de Bruijn* digraph [10]. The following steps transform a de Bruijn digraph $G_{l_m}(\Xi)$ to a *modified de Bruijn* digraph for a particular multi-set of pieces for a string drawn from alphabet Ξ and encoded with a mask of length l_m:

- Each edge represents an l_m length string drawn from Ξ. Parallel edges are added to the digraph for each occurrence of a particular piece in the multi-set.
- Edges which represent strings not in the multi-set are deleted.
- Vertices with degree zero are deleted.

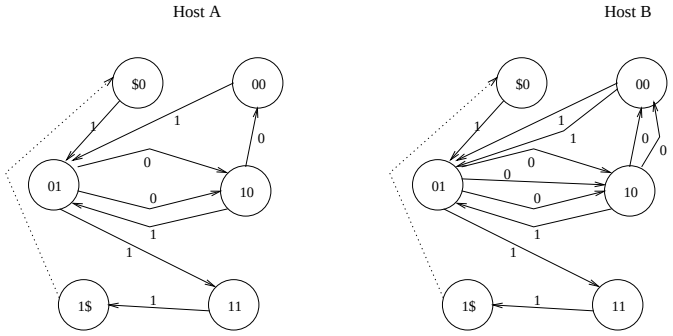

Fig. 2. The modified de Bruijn digraphs for the strings on hosts A and B.

- Two vertices and two edges corresponding to the first and last pieces of the encoded string are added.
- An artificial vertex v is added to the graph. Directed edges are added from the last piece of the encoded string to v and from v to the first piece of the encoded string in order to effect the desired Eulerian cycle.

There is a one-to-one correspondence between the edges in this graph and the pieces in the multi-set except for the two edges attached to the artifical vertex. The modified de Bruijn digraphs for the strings σ' and σ'' (padded on both sides with anchors) are shown in Fig. 2.

3) Modified BEST theorem: Theorem 1, a modification of the well-known BEST theorem, provides the number of Eulerian cycles in a modified de Bruijn digraph. The Theorem and its proof were presented in [6], and we restate it here, with an explanation of notation, for completeness.

It is easy to see that in order for a graph to be Eulerian (*i.e.,* have an Eulerian cycle), in must be the case that the in-degree $d_{in}(i)$ of any vertex v_i equals its out degree $d_{in}(i)$. We may thus define

$$d_i \hat{=} d_{in}(i) = d_{out}(i).$$

We can form a diagonal matrix M from the degrees of the n vertices of the graph

$$M = diag(d_1, d_2, d_3, ..., d_n)$$

that, when put together with the adjacency matrix $A = [a_{i,j}]$ of the graph, produces the Kirchoff matrix C defined to be

$$C = M - A.$$

The Kirchoff matrix has the interesting property that its elements along any row or column sum to zero. Also, for an $n \times n$ Kirchoff matrix, the determinants of all the $(n-1) \times (n-1)$ cofactors are equal and we denote their value by Δ.

Theorem 1 ([6])

For a general Eulerian graph, the total number of Eulerian cycles R is given by

$$R = \frac{\Delta \Pi_i (d_i - 1)!}{\Pi_{ij} a_{ij}!}.$$

4) Decoding algorithm: Once we have the pieces corresponding to a string, we can make them into modified de Bruijn digraph as described Section III-B.2. The graph will generally have more than one Eulerian cycle and, thus, it would be possible to put the pieces together into more than one string. As such, we can use the algorithm in [4] to enumerate all the Eulerian cycles for a given modified de Bruijn digraph. The algorithm (modified for dealing with digraphs) begins with identifying a vertex in the digraph with degree $\neq 1$. It then uses the splitting lemma [5] to split the edges at this vertex, creating a graph for each of the splittings. The algorithm proceeds recursively until all the graphs have no vertex with degree > 1. At that point, each resulting graph corresponds to an Eulerian cycle. The algorithm can easily be modified to deal with loops by not splitting on a self-edge. The algorithm can be used to sequentially enumerate the Eulerian cycles of a graphy by determining a total order (e.g. lexicographic) by which to choose the vertex with degree $\neq 1$ at each step of the recursion.

IV. The STRING-RECON Protocol

We now have the necessary tools to describe our string reconciliation protocol (STRING-RECON).

Consider two hosts A and B holding strings x and y respectively. Then A determines y and B determines x as follows:

- Host A transforms x into a multi-set of pieces MS_A using a (predetermined) mask of length l_m and constructs a modified de Bruijn digraph from the pieces. Host B transforms y into a multi-set of pieces MS_B using the same mask of length l_m and constructs a modified de Bruijn digraph from the pieces.
- A and B determine the index of the desired decoding in the sequential enumeration of all Eulerian cycles in the graph. Thus, n_x corresponds to the index of the Eulerian cycle producing string x, and similarly n_y corresponds to y.
- A and B transform multi-sets MS_A and MS_B to sets with unique elements S_A and S_B by concatenating each element in the set with the number of times it occurs in the multi-set and hashing the resulting string. The sets S_A and S_B contain the resulting hashes.
- The CPISync algorithm [6] is executed to reconcile the sets S_A and S_B. In addition, A sends n_x to B and B sends n_y to A. At the end of this step, both A and B know S_A, S_B, n_x and n_y. Host A then sends elements corresponding to hashes in the set $S_A \setminus S_B$ to B and B sends elements corresponding to the hashes in the set $S_B \setminus S_A$ to A.
- A and B construct the multi-sets MS_B and MS_A respectively from the information obtained in the previous step. The modified de Bruijn digraphs corresponding to these multi-sets are generated.
- The decoding algorithm is applied by A and B to determine y and x respectively.

V. Analysis

Suppose hosts A and B initially have the same copy of a string σ of length n and use the same all-one mask of length l_m to split it up into pieces according to the previously described method. The following lemmas bound the number of differences Δ_{AB} that can occur between the piece multi-sets of A and B in the face of certain edits.

Lemma 1

If only d insertions occur on the copy of σ held by one of the hosts, then the number of differences Δ_{AB} between resulting piece multi-sets is bounded by

$$d + 2(l_m - 1) \leq \Delta_{AB} \leq min((2l_m - 1)d, 2(n - l_m + 1) + d).$$

Proof: It is assumed that the insertions occur at a distance of at least l_m from the beginning and the end of the string, for the more general case. The best case would be achieved when the d insertions occur at a single location. In this case, the host that has the augmented string would have $d + l_m - 1$ different pieces. The other host will have $l_m - 1$ different pieces, totalling up to $d + 2(l_m - 1)$ differences. The worst case would occur when the insertions are spaced a distance l_m apart. One such insertion would cause a difference of $2l_m - 1$ pieces. d such insertions would cause a difference of $d(2l_m - 1)$ pieces. When the insertions (so spaced) span the entire string, all the pieces are different on both the strings and thus the difference becomes $2(n - l_m + 1) + d$. ■

Deletion on one host can be considered as insertion on the other host and thus the bounds for d deletions are trivially

$$d + 2(l_m - 1) \leq \Delta_{AB} \leq min((2l_m - 1)d, 2(n + d - l_m + 1) + d).$$

Lemma 2

If only d replacements occur on the copy of σ held by one of the hosts, then

$$2(d + l_m - 1) \leq \Delta_{AB} \leq min(2dl_m, 2(n - l_m + 1)).$$

Proof: It is assumed that the replacements occur at a distance of at least l_m from the beginning and the end of the string, for the more general case. The best case would be achieved when the replacements are contiguous in the string. In such a case, there

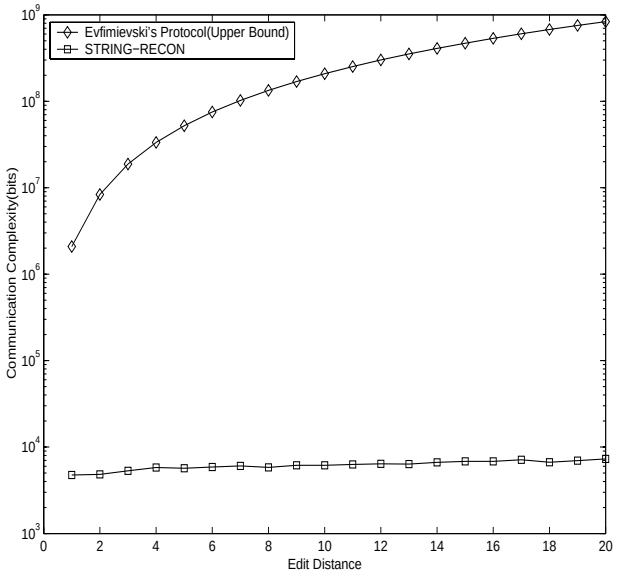

Fig. 3. Upper bound on the communication complexity for STRING-RECON compared to the theoretical bound given by Evfimievski [3]

will be $(d + l_m - 1)$ differences on each host with the other, thus making the total $2(d + l_m - 1)$. The worst case would occur when the replacements are spaced l_m apart. In this case each replacement would introduce $2l_m$ differences, and d replacements would introduce $2dl_m$ differences. When all the pieces are different, then the number of differences is $2(n - l_m + 1)$. ■

Thus, the upper bound on the number of symmetrical differences between the multi-sets on hosts A and B when either d insertions/deletions or replacements are carried out can always be expressed as

$$\Delta_{AB} \leq \alpha d + \beta$$

where α and β are functions of n and l_m only. Thus, the number of elements in the multi-sets affected by such edits is linear in the number of edits performed.

Lemma 3

If Host B uses CPISync for set reconciliation, the amount of communication(in bits) needed by STRING-RECON to reconcile two binary strings which differ in e edits and which have been encoded with a mask of length l_m is upper bounded by

$$\begin{aligned} \textit{COMM} \quad \leq \quad & 2(b+1)m + b + bm_A + m + k + \\ & (m_A + m_B)l_m + \log(R_1 R_2). \end{aligned}$$

where the symbols have the same meaning as in Section I-A and R_1 and R_2 are the total number of Eulerian cycles for the modified de Bruijn digraphs at host A and B respectively.

Recall that the symmetric difference between the sets on the two hosts was bounded in Section V.

Proof: The number of bits transmitted using the probabilistic algorithm in [7] is bounded above by $2(b+1)m + b + bm_A + m + k$. In addition, we need to send the actual pieces corresponding to the hashes that we reconciled ($m_A l_m$, $m_B l_m$ bits) and the index of the actual Eulerian cycle corresponding to each host (upper bounded by $\log(R_1) + \log(R_2)$ bits). ■

VI. Experiments and Results

Simulations were carried out to determine the communication complexity of STRING-RECON as compared to the theoretical upper bound by Evfimievski (no lower bound or experimental data were provided in Evfimievski's work). Host A's string, x, was generated as a uniformly random binary string of length 2000. Thereafter, 100 strings y were constructed at a given edit

distance (in this case only insertions and deletions were used) from x. In our simulations, we used a 32-bit hash function, took the confidence parameter $k = 1$, and used a mask length of 5 bits.

Fig. 3 shows our upper bound on the communication complexity for string-reconciliation as compared to that in [3]. For our computation, we use the bound given in Lemma 3, averaged over 100 strings of the same edit distance. Since the probability of error for CPISync is upper bounded by $m\left(\frac{|S_A|+|S_B|-1}{2^b}\right)^k$, the probability of error of our experimental implementation is at most 9.38×10^{-7}. This is well smaller than the probability of error $\epsilon = 0.001$ we use when computing Evfimievski's bound using Eq. 2. Note that Evfimievski's protocol in [3] is only a one-way reconciliation, and we have adapted it to make a fair comparison to STRING-RECON, which performs a full reconciliation of the strings.

VII. Conclusions

We have presented a protocol for reconciling two similar strings on separate hosts which has a low communication complexity. The key to our protocol is to transform the string reconciliation problem into a set reconciliation problem, for which nearly optimal solutions are known. We have also presented analysis and experimental results for reconciliation of binary strings, although our protocol applies to strings over any alphabet. Our communication upper bound significantly improves the known upper bound for an alternative string reconciliation protocol.

It would be interesting to analyze the natural generalizations of this algorithm to image and other data-structure reconciliation, and to compare our protocol to currently deployed protocols, such as *rsync*, in practical environments.

References

[1] Sachin Agarwal, David Starobinski, and Ari Trachtenberg. On the scalability of data synchronization protocols for PDAs and mobile devices. 16(4):22–28, July/August 2002.

[2] Graham Cormode, Mike Paterson, Suleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *Symposium on Discrete Algorithms*, pages 197–206, 2000.

[3] Alexandre V. Evfimievski. A probabilistic algorithm for updating files over a communication link. In *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 300–305, 1998.

[4] Herbert Fleischner. All-trails algorithm. In *Eulerian Graphs and Related Topics, Part 1*, volume 2. Elsevier Science Publishers B.V., 1991.

[5] Herbert Fleischner. All-trails algorithm. In *Eulerian Graphs and Related Topics, Part 1*, volume 1. Elsevier Science Publishers B.V., 1991.

[6] Bailin Hao, Huilin Xie, and Shuyu Zhang. Compositional representation of protein sequences and the number of eulerian loops. *arXiv:physics/0103028 v1*, 1, 2001.

[7] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. In *International Symposium on Information Theory*, page 232, 2001.

[8] Alon Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In *IEEE Symposium on Foundations of Computer Science*, pages 228–238, 1991.

[9] Alon Orlitsky and Krishnamurthy Viswanathan. Practical protocols for interactive communication. page 115, 2001.

[10] S. S. Skiena and G. Sundaram. Reconstructing strings from substrings. *Journal Of Computational Biology*, 2:333–353, 1995.

[11] A. C. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.