Efficiently decoding strings from their shingles

Aryeh (Leonid) Kontorovich
Email:karyeh@cs.bgu.ac.il
Computer Science
Ben-Gurion University
Beer Sheva, Israel

Ari Trachtenberg
Email:trachten@bu.edu
Electrical & Computer Engineering
Boston University
Boston, MA 02215

Abstract

Determining whether an unordered collection of overlapping substrings (called shingles) can be uniquely decoded into a consistent string is a problem that lies within the foundation of a broad assortment of disciplines ranging from networking and information theory through cryptography and even genetic engineering and linguistics. We present three perspectives on this problem: a graph theoretic framework due to Pevzner, an automata theoretic approach from our previous work, and a new insight that yields a time-optimal streaming algorithm for determining whether a string of n characters over the alphabet Σ can be uniquely decoded from its two-character shingles. Our algorithm achieves an overall time complexity $\Theta(n)$ and space complexity $O(|\Sigma|)$. As an application, we demonstrate how this algorithm can be extended to larger shingles for efficient string reconciliation.

I. Introduction

The problem of efficiently reconstructing a string from a given encoding is fundamental to a broad range of settings. In the information theory world, this is related to the *α-edits* or *string reconciliation* problem [4, 22], wherein two hosts seek to reconcile remote strings that differ in a fixed number of unknown edits, using a minimum amount of communication. A similar problem is faced in cryptography through fuzzy extractors [8], which can be used to match noisy biometric data to encrypted baseline measurements in a secure fashion. Within a biological context, this problem has common roots with the sequencing of DNA from short reads [4] and reconstruction of protein sequences from K-peptides [28]. This idea has even shown up in computational linguistics, where it was used to learn transformations on varying-length sequences [27].

In a simple formal statement of the *unique string decoding problem*, one is given a string $s \in \Sigma^*$ over the alphabet Σ. The string is considered uniquely decodable if there is no other string $s' \in \Sigma^*$ with the same multiset of length 2 substrings (known as bigrams). In the general case, we will be interested in substrings of length $q \geq 2$, which we will call q-grams or *shingles*. In our analysis, we shall assume throughout that alphabet characters can be compared in constant time; otherwise, multiplicative $\log(|\Sigma|)$ terms need to be added where appropriate. Our main result is a $\Theta(n)$ time, $O(|\Sigma|)$ space streaming algorithm for deciding unique decodability. To our knowledge, the best previous algorithm [14] has time complexity $O(n|\Sigma|^3)$ and space complexity $\Theta(|\Sigma|^3)$.

A. Approach

Two principal approaches have been put forth for deciding unique string decodability.

The first is due to Pevzner [25] and Ukkonen [32], who characterized the type of strings that have the same collection of shingles. This approach can be used to generate a simple unique decodability tester whose naive worst-case running time on strings of length n is $\Theta(n^4)$.

The second approach is based on an observation that the set of uniquely decodable strings form a regular language [15]. With this observation, it is possible to produce a deterministic finite state machine on $\exp(\Omega(|\Sigma| \log |\Sigma|))$ states [16] and a non-deterministic one on $O(|\Sigma|^3)$ states [14]. The DFA is prohibitively expensive to construct explicitly, while the NFA may be simulated in time $O(n|\Sigma|^3)$ and space $\Theta(|\Sigma|^3)$.

In this work, we present a streaming, online, linear time algorithm for testing unique decodability of a string. We further show how this algorithm can be extended to provide an efficient protocol for the classic α-edits (or string reconciliation) problem, in which one is tasked with reconciling two remote strings that differ in at most α unknown edits (insertions or deletions) [23]. This approach can be extended into a one-way rateless streaming protocol that reconciles strings an arbitrary edit distance apart.

B. *Outline*

We begin with an overview of related work from the information theory and theoretical computer science communities in Section II, followed by a brief exposition of existing approaches to our core problem in Section III. Our linear-time algorithm for deciding unique decodability, together with a proof of correctness, is described in Section IV. We show in Section V how this algorithm can be generalized for the α-edits problem, and close with concluding remarks and remaining open theoretical questions in Section VI.

II. Related work

A. *Unique decoding*

It was shown in [15] that the collection of strings having a unique reconstruction from the shingles representation is a regular language. Following up, Li and Xie [16] gave an explicit construction of a deterministic finite-state automaton (DFA) recognizing this language. Our work in [14] has demonstrated that there is no DFA of subexponential size for recognizing this language, and instead we have exhibited an equivalent NFA with $\Theta(|\Sigma|^3)$ states.

There has also been work on the probability of a collection of shingles having a unique reconstruction. The authors in [1] show that one can expect a unique decoding for substrings of identically distributed, independent random bits as long as the substrings are roughly logarithmic in the size of the overall decoded string. The work in [9] also provides evidence of a high probability of unique decoding for logarithmically sized substrings, and includes generalizations to non-binary and even non-uniformly random characters for the strings. This is extended in [2] to characterize the number of decodings for a given collection of shingles, and [26] considers decoding from regularly gapped collections of substrings in a DNA sequencing framework. Finally, [21] considers an information-theoretic capacity of the sequencing problem, and presents a greedy algorithm for reconstruction that is asymptotically optimal.

B. *Edit distance*

The problem of determining the minimum number of edits (insertions or deletions) required to transform one string into another has a long history in the literature [6, 11]. Orlitsky [22] shows that the amount of communication $C_{\hat{\alpha}}(x,y)$ *necessary* to reconcile two strings x and y (of lengths $|x|$ and $|y|$ respectively) that are known to be at most $\hat{\alpha}$-edits apart is at most

$$C_{\hat{\alpha}}(x,y) \leq f(y) + 3\log f(y) + \log\hat{\alpha} + 13,$$

for

$$\log\left(\binom{|y|+\hat{\alpha}}{\hat{\alpha}}\right) \leq \lceil f(y)\rceil \leq \log\left(\binom{|y|+\hat{\alpha}}{\hat{\alpha}}\right) + 3\log(\hat{\alpha}),$$

although he leaves an efficient one-way protocol as an open question.

The literature includes a variety of proposed protocols for this problem. Cormode et al. [7] propose a hash-based approach that requires a known bound $\hat{\alpha}$ on edits between x and y (assuming, without loss of generality, that y is the longer string) and communicates at most

$$4\alpha\log(\frac{2|y|}{\alpha})\log(2\hat{\alpha}) + O\left(\alpha\log|y|\log\frac{\log(|y|)}{\ln\frac{1}{1-\epsilon}}\right) \tag{1}$$

bits to reconcile the strings with probability of failure ϵ.

Orlitsky and Viswanthan [24] propose a interactive protocol that does not need to know the number of edits in advance and requires at most

$$2\alpha \log |y| \left(\log |y| + \log \log |y| + \log(1/\epsilon) + \log \alpha\right)$$

bits of communication.

Other approaches include those of Evfimievski [10] for small edit distances, Suel [30] based on delta-compression, and Tridgell [31] which presents the computationally efficient (but potentially communicationally inefficient) rsync protocol.

C. Reconciliation

Another natural approach to the α-edits problem involves the utilization of a *reconciliation* algorithm, which reconciles remote data with minimum communication.

a) Set reconciliation: The problem of set reconciliation seeks to reconcile two remote sets S_A and S_B of b-bit integers using minimum communication. The approach in [20] involves translating the set elements into an equivalent *characteristic polynomial*, so that the problem of set reconciliation is reduced to an equivalent problem of rational function interpolation, much like in Reed-Solomon decoding [18].

The resulting algorithm requires one message of roughly bm bits of communication and bm^3 computation time to reconcile two sets that differ in m entries. The approach can be improved to expected bm communication and computation through the use of interaction [19] and generalized to multisets and to arbitrary error-correcting codes [12].

b) String reconciliation: A string σ can be transformed into a multiset S through *shingling*, or collecting all contiguous substrings of a given length, including repetitions. For example, shingling the string katana into length 2 shingles produces the multiset:

$$\{\mathsf{at}, \mathsf{an}, \mathsf{ka}, \mathsf{na}, \mathsf{ta}\}\,. \tag{2}$$

As such, in order to reconcile two strings σ_A and σ_B, the protocol STRING-RECON [1] first shingles each string, then reconciles the resulting sets, and then puts the shingles back together into strings in order to complete the reconciliation. It is important to note that if two strings differ by α edits, then they will also differ in $O(\alpha)$ shingles, as long as shingle size is a constant.

The process of combining shingles of length l back into a string involves the construction of a modified de Bruijn graph of the shingles. In this graph, each shingle corresponds to an edge, with weight equal to the number times the shingle occurs in the multiset. The vertices of the graph are all length $l-1$ substrings over the shingling alphabet; in this manner, an edge $e(u, v)$ corresponds to a shingle s if u (resp. v) is a prefix (resp. suffix) of s. A special character \$ used at the beginning and end of the string in order to mark the first and last shingle.

An Eulerian cycle in the modified de Bruijn graph, starting at the first shingle, necessarily corresponds to a string that is consistent with the set of shingles. Unfortunately, there may be a large number of strings consistent with a given shingling, so that well-defined decoding requires either the specification of one cycle of interest or another way to guarantee only one possible cycle.

III. EXISTING APPROACHES

We now describe two existing approaches for determining whether a given string is uniquely decodable.

A. Transformation

In an analysis of approximate string matching, Ukkonen [32] conjectured that two strings with the same shingles are related through two string transformations, for $(q-1)$-grams z_1 and z_2 and arbitrary strings x_i:

- **Transposition** - wherein a string
$$x = x_1 z_1 \mathbf{x_2} z_2 x_3 z_1 \mathbf{x_4} z_2 x_5$$
is transformed into
$$x' = x_1 z_1 \mathbf{x_4} z_2 x_3 z_1 \mathbf{x_2} z_2 x_5.$$
- **Rotation** - wherein a string
$$x = z_1 x_1 z_2 x_2 z_1$$
is transformed into
$$x' = z_2 x_2 z_1 x_1 z_2.$$

Pevzner [25] proved that this conjecture is true, thus providing a simple but inefficient algorithm for determining the unique decodability of a string.

B. Regular languages

A second approach for testing unique decodability is automata theoretic in nature.

1) Preliminaries: We assume a finite alphabet Σ along with a special delimiter character $\$ \notin \Sigma$, and define $\Sigma_\$ = \Sigma \cup \{\$\}$. For $k \geq 1$, the q-gram map Φ takes string $x \in \$\Sigma^*\$$ to a vector $\xi \in \mathbb{N}^{\Sigma_\$^q}$, where $\xi_{i_1,\ldots,i_q} \in \mathbb{N}$ is the number of times the string $i_1 \ldots i_q \in \Sigma^q$ occurred in x as a contiguous subsequence, counting overlaps. Note that, though we focus this section on the *bigram* case when $q = 2$, the results are straightforwardly generalized to the case $q > 2$.

It is easy to see that the bigram map $\Phi : \$\Sigma^*\$ \to \mathbb{N}^{\Sigma_\$^2}$ is not injective; for example, the shingles in (2) imply that $\Phi(\$\mathsf{katana}\$) = \Phi(\$\mathsf{kanata}\$)$. We denote by $L_{\text{UNIQ}} \subseteq \Sigma^*$ the collection of all strings w for which
$$\Phi^{-1}(\Phi(\$w\$)) = \{\$w\$\}$$
and refer to these strings as *uniquely decodable*, meaning that there is exactly one way to reconstruct them from their bigrams. The induced *bigram graph* of a string $w \in \Sigma^*$ is a weighted directed graph $G = (V, E)$, with $V = \Sigma_\$$ and $E = \{e(a,b) : a, b \in \Sigma_\$\}$, where the edge weight $e(a,b) \geq 0$ records the number of times a occurs immediately before b in the string $\$w\$$. Finally, we will denote the omission of a symbol from the alphabet by $\Sigma_{\bar{x}} := \Sigma \setminus \{x\}$ for $x \in \Sigma$.

2) Regularity of obstructions: For $x \in \Sigma$ and $a, b \in \Sigma_{\bar{x}}$, the languages
$$I_{x,a,b} = L\left(\Sigma^* a x \Sigma_{\bar{a}}^* b \Sigma^*\right)$$
and
$$J_{x,a,b} = L\left(\Sigma^* a \Sigma_{\bar{x}}^* b \Sigma^*\right)$$
form the obstruction language
$$K_{x,a,b} = I_{x,a,b} \cap J_{x,a,b},$$
whose elements are called *obstructions* (because they obstruct a unique decoding). The language of all obstructions is thus
$$L_{\text{OBST}} = \bigcup_{x \in \Sigma} \bigcup_{a,b \in \Sigma_{\bar{x}}} K_{x,a,b}. \tag{3}$$

The work in [14] provides a canonical DFA that recognizes $K_{x,a,b}$ with 9 states, regardless of Σ. Over all $x \in \Sigma$ and $a, b \in \Sigma_{\bar{x}}$, there are

$$|\Sigma| \left(|\Sigma| - 1 + (|\Sigma| - 1)(|\Sigma| - 2)\right) \tag{4}$$

distinct obstruction languages, whose union can thus be accepted by an NFA of $\Theta(|\Sigma|^3)$ states.

The main theorem in that work is that the language of obstructions is precisely the complement of the language of uniquely decodable strings.

Theorem 1 ([14]).

$$L_{\text{OBST}} = \Sigma^* \setminus L_{\text{UNIQ}}.$$

The result of Theorem 1 is that the NFA accepting $K_{x,a,b}$'s can be used to test for unique decodability.

IV. Efficient online testing

We now describe our main result: an efficient, online streaming algorithm for determining whether a given string $w \in \Sigma^*$ is uniquely decodable from its bigrams. Algorithm 1 is online in the sense that it needs only constant-time pre-processing, and streaming, in that results for one string can be sub-linearly extended to a superstring.

As a convention, we will use "low" letters a, b, c to denote members of Σ while the "high" letters u, v, w will denote *strings* over Σ. For any $u \in \Sigma^*$, we write $G(u)$ for the bigram graph induced by u, and we shall use the notation $a \to b$ (resp. $a \Rightarrow b$) to mean that there is a directed edge (resp. path) from a to b. We use the shorthand "u is UD" to denote that $u \in L_{\text{UNIQ}}$. The i^{th} character of w is denoted by $w[i]$ and characters i through j by $w[i:j]$.

The following theorem establishes the correctness of Algorithm 1.

Theorem 2. *Algorithm 1 returns TRUE iff its string* $w \in L_{\text{UNIQ}}$.

Proof: Observe that $u \notin L_{\text{UNIQ}}$ implies $u\sigma \notin L_{\text{UNIQ}}$ for all $\sigma \in \Sigma$ (in fact, $\Sigma^* \setminus L_{\text{UNIQ}}$ is a two-sided ideal under concatenation). Thus, as soon as a non-UD prefix is observed, we know that the entire string is not UD.

Our algorithm can conclude that the prefix is not UD in two places: at line 11 and line 20. Line 11 handles an intrusion upon an existing cycle. Formally, this means that the prefix $u = w[1:i-1]$ may be expressed as $u = va_1a_2 \ldots a_kv'$ where $v, v' \in \Sigma^*$, $(a_j)_{1 \le j \le k} \in \Sigma$, $a_1 = a_k$ and the current character $x = w[i]$ is equal to some a_j, for $1 \le j \le k$. Thus the string $w[1:i]$ has at least two distinct decodings, among which are $va_1a_2 \ldots a_kv'x$ and $va_1v'xa_{j+1}a_{j+1} \ldots a_ka_2 \ldots a_j$.

Line 20 handles the case of *communicating* parents a and b (one of them possibly a self-parent), by which we mean that they are in the same strongly connected component. Note that the mere existence of a node with two communicating parents is insufficient to disqualify a string, as the example $w = axbxa$ shows. However, the condition in the loop has us visiting a node $x = w[i]$ that *already* has two communicating parents. First, let us dispense with the case where $x \in \{a, b\}$ — say, $x = b$ without loss of generality. Since $a \to x$, $x \to x$ and $x \Rightarrow a$, the self-loop at x can be taken after the first visit to x or after a later visit, creating an ambiguity in the decoding. Thus, we will take $x \notin \{a, b\}$ and assume without loss of generality that the first occurrence of a in $w[1:i-1]$ occurs before the first occurrence of b. We claim that $w[1:i]$ must be of the form $(\Sigma \setminus \{a, b, x\})^*ax(\Sigma \setminus \{a, x\})^*bx(\Sigma \setminus \{a\})^*ax$. Indeed, a must occur twice in $w[1:i-1]$ (since it occurs before b but $b \Rightarrow a$) and the second occurrence of a must be after the last occurrence of b (for otherwise b's directed path to a would intrude upon an existing cycle and render the string non-UD earlier on). Immediately following a's first occurrence is x, followed by some string containing neither a (whose second occurrence will be at $w[i-1]$) nor x (for otherwise the edge $b \to x$ will intrude on an existing cycle and disqualify the string earlier on). Then bx

```
input  : string $w \in \Sigma^*$
output: TRUE if $w \in L_{\text{UNIQ}}$ and FALSE otherwise
1  initialize each $v \in \Sigma$ as not having been visited;
2  initialize each $v \in \Sigma$ as not belonging to a cycle;
3  initialize the graph $G$ with vertex set $\Sigma$ and no edges;
4  mark $w[1]$ as having been visited;
5  for $i := 2$ to $|w|$ do
6      has the node $w[i]$ already been visited? if NO then
7          mark $w[i]$ as having been visited;
8      else// node $w[i]$ has already been visited -- thus, it is on a
       cycle
9          does the edge $w[i-1] \to w[i]$ already exist in $G$? if NO then
10             does $w[i]$ belong to an existing cycle? if YES then
                   // intrusion on an existing cycle
11                 return FALSE;
12             else// creating a new cycle
13                 label $w[i]$ and all the nodes visited since the previous  occurrence of $w[i]$ as
                   belonging to a cycle;
14             end
15         else
               // the edge $w[i-1] \to w[i]$ already exists in $G$
               // stepping along an existing cycle
16         end
17     end
18     are there two distinct nodes $a, b \in G$ such that $a \to w[i], b \to w[i]$ and $a, b$ belong to the
       same strongly connected component of $G$?
       // the possibility $a = w[i]$ is not excluded
19     if YES then
20         return FALSE;
21     end
22     draw the edge $w[i-1] \to w[i]$ in $G$;
23 end
24 return TRUE;
```

Algorithm 1: Online algorithm for testing unique decodability

occurs for the first time and is immediately followed a string not containing a and followed by ax. Define i_1, i_2, i_3 to be the indices of the first, second and last occurrences of x in $w[1:i]$, respectively, and put $v_1 = w[1:i_1-1]$, $v_2 = w[i_1+1:i_2-1]$, $v_3 = w[i_2+1:i_3-1]$. Observe that $w[1:i] = v_1xv_2xv_3x$ and $w' \equiv v_1xv_3xv_2x$ have the same bigram encoding. Note also that necessarily $v_2 \neq v_3$, since the former does not contain a and the latter does. This shows that the prefix $w[1:i]$ is not UD.

Having shown that whenever our algorithm disqualifies a string it is indeed not UD (*completeness*), we now show that any string that survives at the loop's termination is in fact UD (*soundness*).

We prove this claim by induction on the prefix length. Our inductive hypothesis is that the prefix $u = w[1:i-1]$ is UD. We read the next character $x = w[i]$. Clearly, if $u \in L_{\text{UNIQ}}$ and x does not occur in u then $ux \in L_{\text{UNIQ}}$. As such, we consider the case where x does occur somewhere in u. If the edge $w[i-1] \to x$ does not currently exist in the bigram graph $G(u)$, then we may assume that x occurs exactly once in u, as, otherwise, it would already be marked as belonging to a cycle, disqualifying u.

Thus, $u = vxv'$ where $v, v' \in \Sigma_{\bar{x}}^*$. Furthermore, our assumption that $u \in L_{\text{UNIQ}}$ implies that v and v' cannot have any letters in common, for then x would be on a cycle upon which the new edge $w[i-1] \to x$ would intrude. Finally, observe that if two UD strings have no characters in common, then their concatenation is also UD. Thus, $ux \in L_{\text{UNIQ}}$.

It remains to consider the case where the edge $w[i-1] \to x$ already exists in $G(u)$. Although we are stepping along an existing cycle and not creating a new one, this transition may render the string non-UD, as the example $w[1:i] = axbxbax$ shows. Since $u = w[1:i-1]$ is UD, there can be at most two distinct a, b such that $a \to x$ and $b \to x$ (the existence of 3 or more distinct nodes pointing to x is easily seen to render u non-UD; see [15, Theorem 9] for an analogous fact regarding 3 or more children). The case of a single $a \to x$ is trivial, so suppose that $a \to x$ and $b \to x$, but a and b are not in the same strongly connected component. There is no loss of generality in assuming that b is reachable from a but not vice versa. In this case, the only valid decoding of $w[1:i]$ is of the form $vaxv'bx$ where $v \in (\Sigma \setminus \{x, b\})^*$ and $v' \in (\Sigma \setminus \{a\})^*$. ■

A. Runtime analysis

Algorithm 1 can be implemented in time $\Theta(n)$ on strings of n characters over an alphabet Σ, with the aid of several simple data structures. We account for the running time:

- Lines 01-04. This is simple initialization. It can be accomplished explicitly in $\Theta(|\Sigma|)$ time for our data structures delineated hereafter, or in constant time with a sparse representation.
- Lines 06-08. We use a simple array to keep track of which vertices have been seen, a constant time cost for each string character.
- Line 9. The key observation here is that the graph is necessarily sparse, since any node with more than two parents or children necessarily renders the graph not uniquely decodable [15]. As such, the graph can be stored as an adjacency list so that this line represents a constant time operation for each string character.
- Lines 10-19. We maintain a stack onto which vertices are pushed in the order that they are visited. When a vertex is visited a second time, we pop all vertices off the stack until we revisit the original node, marking all popped vertices as being within an existing cycle. Each character of w will be, at worst, pushed and popped from the stack once, resulting in an aggregated running time of $\Theta(n)$ for this step.
- Line 21. To determine whether two vertices a, b are in the same strongly connected component, we record the first and last index in w at which a occurs in i_a and j_a, respectively, and do the same for b. The vertices a and b belong to the same connected component if and only if $[i_a, j_a] \cap [i_b, j_b] \neq \emptyset$. This check is a constant-time operation per character.

V. String reconciliation

We next present the string reconciliation protocol in [13] as a specific example where our online unique decodability algorithm is applicable. This specific protocol is a refinement of a shingling approach in [1], and is further based on a transformation to an instance of the set reconciliation [20].

A. Definitions

The protocol is fundamentally based on the concept of a *shingling*. Formally, a *shingle* $s = s_1 s_2 \ldots s_k$ is simply an element of $\Sigma_{\$}^*$. For two shingles $s = s_1 s_2 \ldots s_k$ and $t = t_1 t_2 \ldots t_\ell$, we write $s \overset{l}{\rightsquigarrow} t$ if there is some length $\geq l$ suffix u of s that is also a prefix of t, or, more precisely, if we can rewrite $s = s'u$ and $t = ut'$ for strings s', t' and $|u| \geq l$. We define the *non-overlapping concatenation* $s \odot_l t$ (or just $s \odot t$ in context) as the concatenation $s'ut'$, where $s = s'u$, $t = ut'$ and $|u| = l - 1$. For example, $\mathsf{kata} \overset{3}{\rightsquigarrow} \mathsf{tana}$ and $\mathsf{kata} \odot_3 \mathsf{tana} = \mathsf{katana}$.

For a fixed l, the sequence of shingles $s^1 \overset{l}{\rightsquigarrow} s^2 \overset{l}{\rightsquigarrow} \ldots \overset{l}{\rightsquigarrow} s^t$ is said to *represent* the word $w \in \Sigma^*$ if $w = s^1 \odot s^2 \odot \ldots \odot s^t$ and $s^i \overset{l}{\rightsquigarrow} s^{i+1}$ for all i. If $S = \{s^1, \ldots, s^t\}$ is a multiset of shingles, we will use

```
1. Split $\sigma$ into a set $S_\sigma$ of length $l$ shingles, with the $i^{\text{th}}$ shingle of the string denoted $s_i$. Similarly
   split $\tau$ into $S_\tau$.
2. Reconcile sets $S_\sigma$ and $S_\tau$.
3. The first host sets $S_\sigma^0 \longleftarrow \{s_0\}$.
4. For $i$ from 1 to $|\sigma| - l + 1$ do
       $S_\sigma^i \longleftarrow S_\sigma^{i-1} \cup \{s_i\}$
       While $S_\sigma^i$ is not uniquely decodable
           Merge the last two shingles added to $S_\sigma^i$.
5. Exchange indices of merged shingles.
6. Uniquely decode $S_\sigma^i$ and $S_\tau^i$ on the remote hosts.
```

Protocol 1: Reconciliation of remote strings σ and τ.

$\Phi^{-1}(S) \subseteq \Sigma^*$ to denote the collection of all words represented by S. More formally, define $\Pi = \Pi(S)$ to be the set of all permutations on $t = |S|$ elements with the property that $s^{\pi(i)} \overset{l}{\rightsquigarrow} s^{\pi(i+1)}$ for all i. Then $\Phi^{-1}(S)$ is

$$\left\{ w \in \Sigma^* : \$w\$ = s^{\pi(1)} \odot s^{\pi(2)} \odot \ldots \odot s^{\pi(t)}, \pi \in \Pi \right\}.$$

We refer to the members of $\Phi^{-1}(S)$ as the *decodings* of S, and say that S is uniquely decodable if $|\Phi^{-1}(S)| = 1$.

A *shingling* I of a word $w = w_1 \ldots w_t \in \Sigma^*$ is a set of substrings of $w$ that represents $w$. We say that I is an uniquely decodable shingling of $w$ if $|\Phi^{-1}(I(w))| = 1$.

As a simple example, consider the string $w = \mathsf{katana}$ with the shingling $I(w) = \{\mathsf{\$k}, \mathsf{ka}, \mathsf{at}, \mathsf{ta}, \mathsf{an}, \mathsf{na}, \mathsf{n\$}\}$. As we saw in Section III-B, for l=2, I can be alternately decoded into kanata and is thus not uniquely decodable. However, if the second and third shingles are merged into ata, that the shingling becomes $\{\mathsf{\$k}, \mathsf{ka}, \mathsf{ata}, \mathsf{an}, \mathsf{na}, \mathsf{n\$}\}$, and then there is exactly one decoding: katana.

B. Elaboration

Protocol 1 transforms a string that is not uniquely decodable into one that is by merging shingles. Several important details of Protocol 1 require explanation and proof of correctness.

1) Steps 1 and 2: The first two steps of the protocol derive from the base protocol described in Section II-C. Note that l is an implementation parameter.

2) Step 3: The expression S_σ^i represents the multiset of shingles that have been seen so far. It is modified, by combining shingles as necessary in the subsequent steps, in order to ensure unique decodability. If full reconciliation is desired (i.e. both hosts know the other host's string, as opposed to just one host having this knowledge) then Steps 3 and 4 are similarly run on the remote host with set S_τ^i.

3) Step 4: In merging two shingles s_a and s_b, we are simply computing the non-overlapping concatenation $s_a := s_a \odot s_b$, as defined earlier. Since the shingles are contiguous and based on an initial length l shingling, we know necessarily that $s_a \overset{l}{\rightsquigarrow} s_b$. Furthermore, it is clear that such merging will always, eventually, lead to a decodable set of shingles because, at worst, the protocol results in just one shingle representing the entire string, which is necessarily uniquely decodable.

The main challenge of this step is in checking whether a given set of shingles is uniquely decodable. This can be done in an online manner with two extensions to our algorithm in IV.

a) Extension to q-grams: First, Algorithm 1 needs to be extended to shingles of length $q > 2$, rather than just bigrams. This can be accomplished by considering w_i to be the length $q - 1$ prefix of the i^{th} shingle of the input string; for $q = 2$, we have the existing case that w_i is the i^{th} character of the string.

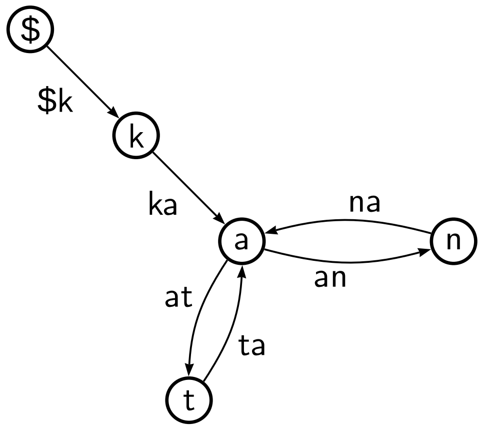

Fig. 1. A de Bruijn graph corresponding to the substring $katana.

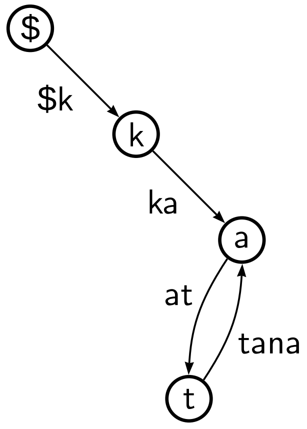

Fig. 2. A uniquely-decodable modified de Bruijn graph corresponding to the substring $katana.

In this model, the input alphabet is enlarged to Σ^{q-1} and edges correspond to shingles. Note that this extension works even with the mixed-length shingles which Protocol 1 produces.

b) Extension for shingle merging: When shingles are merged, we are effectively combining two edges $e_1 = (v_1, v_2)$ and $e_2 = (v_2, v_3)$ into their transitive closure $e_3 = (v_1, v_3)$. This is demonstrated when Figure 1, which is not uniquely decodable, is transformed into Figure 2, which is uniquely decodable because shingles ta,an, and na have been merged into their transitive closure tana.

Such a transitive closure can be implemented in Algorithm 1 by patching steps 11 and 20 so as to reverse one step of the broader iteration, and add the transitive closure edge instead, instead of returning FALSE as in the current implementation.

4) Step 5: Each host needs to know which shingles were merged on the other host in order to produce a uniquely decodable multiset of shingles. To exchange this information, we first canonically order all shingles, and then note that each merge involves at least one shingle of length l and another (possibly composite) shingle of length $\geq l$. As such, a merge is fully specified by sending the index of the length l shingle, and the index of one of the shingles that comprises the composite shingle.

5) Step 6: The resulting collection of shingles can only be decoded in only one way, which can be provided by any efficient algorithm for generating an Eulerian cycle through the graph (e.g., the algorithm implied in [15, Theorem 11] can be implemented in linear time).

C. Communication Complexity

Only Steps 2 and 5 in Protocol 1 transmit data. For two strings of length n differing in α edits, Step 2 will require $O(\alpha\ l^2)$ bits of communication for the implementation parameter l. Step 5 will require between 0 and $2n\log(n-l+1)$ communication, depending on the decodability of the string.

More precisely, the communication efficiency of the protocol relies upon having as few merge operations as possible, since, at worst, *every* shingle is merged in Step 5, requiring $2n\log n$ bits of communication for a shingle set of size n. In the best case, no shingles are merged and the communication complexity of the protocol is directly related to the edit distance between reconciled strings. The shingle size l thus represents a tradeoff between communication spent on set reconciliation and communication spent on merge identification.

Though it is hard to give precise bounds on the number of shingles that are merged in this step, the work in [1] provides some guidance for random strings. Specifically, for strings of n random bits, in which each bit is 0 with probability $p > 0.5$, then we can expect each node in the de Bruijn graph of length l shingles to have only one outgoing edge (implying unique decodability) if

$$l \leq n + 1 + \frac{W\left(-\ln(p)p^{-n}\right)}{\ln p}, \tag{5}$$

where $W(\cdot)$ is the Lambert W function [5]. When n goes to infinity, then (5) is $O(\log(n))$, meaning that logarithmically sized shingles should avoid communicationally expensive merges.

Thus, when the two strings are composed of random iid bits, then, under the appropriate choice of l from (5), we can expect that no merging is needed giving an overall communication complexity that is $O\left(\alpha\log^2(n)\right)$, for large n.

D. Rateless approach

Observe that Protocol 1 communicates two types of data: (i) set reconciliation data from step 2, and (ii) merged shingle indices in step 5.

The set reconciliation data can be ratelessly streamed for reconciling strings with arbitrary edit distance by using a simple modification of the protocol in [20]. Specifically, a characteristic polynomial

$$\chi_{S_\sigma}(Z) = (Z - s_1)(Z - s_2)(Z - s_3)\cdots(Z - s_{|S_\sigma|})$$

of the shingles $s_i \in S_\sigma$ is computed and its evaluations at points in an appropriately sized finite field are provided to the decoder, which similarly computes evaluations of its own characteristic polynomial. The rational function representing the division of the two polynomials can be determined from any Δ sample points, if the two shingle sets differ in at most Δ shingles (an additional k verification points can be added to probabilistic check the result).

The merged shingle indices, which can be determined independently of the reconciliation, can be encoded with any standard rateless code [3, 17, 29], and the two rateless streams can be combined by considering them inputs to yet a third rateless encoding.

VI. Conclusion

We have provided a linear-time algorithm for determining whether a given string is uniquely decodable from its bigrams. Our algorithm is online, in that it needs only constant-time pre-processing, and streaming, in that results for one string can be sub-linearly extended to a superstring. We have also shown how this algorithm can be incorporated into an existing protocol for string reconciliation, though the space of applications potentially extends further to networking, cryptography, and genetic engineering.

Several interesting open questions remain. For one, it is natural to ask whether the proposed online algorithm can be extended for testing the existence of 2, 3, ... or k decodings. It is also interesting to provide sharper bounds for the numbers of merged shingles in Protocol 1 under different random string models, as this could help determine the correct choice for initial shingling size l, in addition to tightening bounds on the communication complexity of the protocol.

REFERENCES

[1] Sachin Agarwal, Vikas Chauhan, and Ari Trachtenberg. Bandwidth efficient string reconciliation using puzzles. *IEEE Trans. Parallel Distrib. Syst.*, 17(11):1217–1225, 2006.

[2] Richard Arratia, Béla Bollobás, Don Coppersmith, and Gregory B. Sorkin. Euler circuits and dna sequencing by hybridization. *Discrete Applied Mathematics*, 104(1 - 3):63 – 96, 2000.

[3] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. *Proceedings of ACM SIGCOMM '98*, pages 56–67, September 1998.

[4] Mark Chaisson, Pavel A. Pevzner, and Haixu Tang. Fragment assembly with short reads. *Bioinformatics*, 20(13):2067–2074, 2004.

[5] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W function. *Adv. Comput. Math.*, 5(4):329–359, 1996.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C.F. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[7] Graham Cormode, Mike Paterson, Süleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *SODA*, pages 197–206, 2000.

[8] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[9] Martin Dyer, Alan Frieze, and Stephen Suen. The probability of unique solutions of sequencing by hybridization. *Journal of Computational Biology*, 1(2):105–110, Summer 1994.

[10] A.V. Evfimievski. A probabilistic algorithm for updating files over a communication link. *Theoretical Computer Science*, pages 191–199, 2000.

[11] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

[12] M. Karpovsky, L. Levitin, and A. Trachtenberg. Data verification and reconciliation with generalized error-control codes. *39th Annual Allerton Conference on Communication, Control, and Computing*, October 2001.

[13] Aryeh (Leonid) Kontorovich and Ari Trachtenberg. String reconciliation with unknown edit distance. Presented in part at ITA 2012. Also submitted elsewhere.

[14] Aryeh (Leonid) Kontorovich and Ari Trachtenberg. Unique decodability for string reconciliation. submitted.

[15] Leonid Kontorovich. Uniquely decodable n-gram embeddings. *Theor. Comput. Sci.*, 329(1-3):271–284, 2004.

[16] Qiang Li and Huimin Xie. Finite automata for testing composition-based reconstructibility of sequences. *J. Comput. Syst. Sci.*, 74(5):870–874, 2008.

[17] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. *Proceedings of the 29th ACM Symposium on Theory of Computation*, 1997.

[18] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, New York, 1977.

[19] Y. Minsky and A. Trachtenberg. Scalable set reconciliation. In *Proc. 40-th Allerton Conference on Comm., Control, and Computing*, Monticello, IL., October 2002.

[20] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Trans. on Info. Theory*, September 2003.

[21] Abolfazl Motahari, Guy Bresler, and David Tse. Information theory of dna sequencing.

[22] A. Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 228–238, 1991.

[23] A. Orlitsky. Interactive communication of balanced distributions and correlated files. *SIAM Journal on Discrete Mathematics*, 6(4):548–564, November 1993.

[24] A. Orlitsky and K. Viswanathan. Practical algorithms for interactive communication. In *IEEE International Symposium on Info. Theory*, June 2001.

[25] P. Pevzner. Dna physical mapping and alternating eulerian cycles in colored graphs. *Algorithmica*, 13:77–105, 1995. 10.1007/BF01188582.

[26] Franco P. Preparata and Eli Upfal. Sequencing-by-hybridization at the information-theory bound: An optimal algorithm. *Journal of Computational Biology*, 7(3-4):621–630, August 2000.

[27] D. E. Rumelhart and J. L. McClelland. On learning past tenses of english verbs. In *Parallel Distributed Processing: Vol 2: Psychological and Biological Models*, pages 216–271. MIT press, 1986.

[28] Xiaoli Shi, Huimin Xie, Shuyu Zhang, and Bailin Hao. Decomposition and reconstruction of protein sequences: The problem of uniqueness and factorizable language. *Journal of the Korean Physical Society*, 50(1I):118–123, 2007.

[29] A. Shokrollahi. Raptor codes. *Information Theory, IEEE Transactions on*, 52(6):2551 –2567, june 2006.

[30] Torsten Suel, Patrick Noel, and Dimitre Trendafilov. Improved file synchronization techniques for maintaining large replicated collections over slow networks. In *ICDE*, pages 153–164, 2004.

[31] A. Tridgell. *Efficient algorithms for sorting and synchronization*. PhD thesis, The Australian National University, 2000.

[32] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191 – 211, 1992.