Determining the unique decodability of a string in linear time

Jiaxi Jin Email:jin@bu.edu
Electrical & Computer Engineering
Boston University
Boston, MA 02215 **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 common to 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 an efficient streaming algorithm for determining whether a string of n characters over the alphabet Σ can be uniquely decoded from its two-character shingles; our online algorithm achieves an overall time complexity $\Theta(n + |\Sigma|)$ and space complexity $O(|\Sigma|)$. As an application, we demonstrate how this algorithm can be adapted to larger, varying-size shingles for (empirically) 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 information theory, this is related to the α-*edits* or *string reconciliation* problem [3, 20], 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 [7], 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 [3] and reconstruction of protein sequences from K-peptides [25]. This idea has even shown up in computational linguistics, where it was used to learn transformations on varying-length sequences [24].

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.

A. Approach

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

The first is due to Pevzner [23] and Ukkonen [29], 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 [13]. With this observation, it is possible to produce a deterministic finite state machine on $\exp(\Omega(|\Sigma| \log |\Sigma|))$ states [14] and a non-deterministic one on $O(|\Sigma|^3)$ states [12]. 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 from its length 2 substrings; to our knowledge, the best previous algorithm [12] has time complexity $O(n|\Sigma|^3)$ and

space complexity $\Theta(|\Sigma|^3)$. We further show how this algorithm can be extended to arbitrary (and varying) length shingles, thus enabling an (empirically) 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) [21]. This approach can be extended into a one-way rateless streaming protocol that reconciles strings that are 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. Our linear-time algorithm for deciding unique decodability, together with a proof of correctness, is described in Section III. We show in Section IV how this algorithm can be generalized to arbitrary- and varying-length shingles, which have application to the α-edits problem, and close with concluding remarks and remaining open theoretical questions in Section V.

II. Related work

A. 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 [5, 9]. Orlitsky [20] 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 $f(y) \approx \log \left(\binom{|y|+\hat{\alpha}}{\hat{\alpha}} \right)$, 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. [6] 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 $4\alpha \log(\frac{2|y|}{\alpha}) \log(2\hat{\alpha}) + O\left(\alpha \log |y| \log \frac{\log(|y|)}{\ln \frac{1}{1-\epsilon}}\right)$ bits to reconcile the strings with probability of failure ϵ.

Orlitsky and Viswanthan [22] 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 [8] for small edit distances, Suel [27] based on delta-compression, and Tridgell [28] which presents the computationally efficient (but potentially communicationally inefficient) rsync protocol.

B. 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 [18] 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 [16].

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 [17] and generalized to multisets and to arbitrary error-correcting codes [10].

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:

$$\{\mathsf{at}, \mathsf{an}, \mathsf{ka}, \mathsf{na}, \mathsf{ta}\}. \tag{1}$$

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.

C. Unique decoding

In an analysis of approximate string matching, Ukkonen [29] conjectured that two strings with the same shingles are related through two types of string transformations, and

Pevzner [23] proved that this conjecture is true, thus providing a simple but inefficient algorithm for determining the unique decodability of a string, and Motahari et al [19] provided asymptotic bounds on how many shingles are needed to reliably reconstruct a string.

It was later shown in [13] that the collection of strings having a unique reconstruction from the shingles representation is a regular language. Following up, Li and Xie [14] gave an explicit construction of a deterministic finite-state automaton (DFA) recognizing this language. Our work in [12] 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.

III. Efficient online testing

Before describing our main result, we give a conceptually simpler 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. The actual algorithms used in our protocol build on the ideas in Algorithm 1.

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 further use the shorthand "u is UD" to denote that $u \in L_{\text{UNIQ}}$, and the i^{th} character of w is denoted by $w[i]$ and characters i through j by $w[i:j]$.

Since the algorithm above will be superseded by those in the sequel, we omit a runtime and correctness analysis and, instead, provide a brief informal discussion. As each character of the string is read, the corresponding shingle (i.e. edge) is traced through a modified de Bruijn graph, whose vertices correspond to Σ. The main idea of the algorithm is to track cycles in this graph. As we prove later, there are two ways that a cycle can break unique decodability: if a cycle intrudes on an existing cycle from outside that cycle, or if the current node has two parents that are in the same strongly connected component. Otherwise, the string is uniquely decodable.

IV. String reconciliation

We next present the string reconciliation protocol in [11] as a specific example where a generalization of 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 set reconciliation [18].

A. Definitions

The protocol is fundamentally based on the concept of a *shingling*. Formally, a *shingle* $s = s_1 s_2 \dots s_k$ is simply an element of $\Sigma_{\$}^*$, where \$ is a special delimiter found only at the beginning and end of a string. For two shingles $s = s_1 s_2 \dots s_k$ and $t = t_1 t_2 \dots t_{k'}$, we write $s \overset{l}{\rightsquigarrow} t$ if there is some length $\geq l-1$ 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 $|u| \geq l-1$. We define the

input : string $w \in \Sigma^*$
output: TRUE if $w \in L_{\text{UNIQ}}$ and FALSE otherwise

initialize each $v \in \Sigma$ as **not** having been visited, and each $v \in \Sigma$ as **not** belonging to a cycle;
initialize the graph G with vertex set Σ and no edges;
mark $w[1]$ as having been visited;
for $i := 2$ **to** $|w|$ **do**
  has the node $w[i]$ already been visited? **if** NO **then**
    mark $w[i]$ as having been visited;
  else// node $w[i]$ has already been visited -- thus, it is on a cycle
    does the edge $w[i-1] \to w[i]$ already exist in G? **if** NO **then**
      does $w[i]$ belong to an existing cycle? **if** YES **then**
        // intrusion on an existing cycle
        return FALSE;
      else// creating a new cycle
        label $w[i]$ and all the nodes visited since the previous occurrence of $w[i]$ as belonging to a cycle;
      end
    else
      // the edge $w[i-1] \to w[i]$ already exists in G, stepping along an existing cycle
    end
  end
  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
  if YES **then**
    return FALSE;
  end
  draw the edge $w[i-1] \to w[i]$ in G;
end
return TRUE;

Algorithm 1: Online algorithm for testing unique decodability

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, kata $\overset{3}{\rightsquigarrow}$ tana and kata $\odot_3$ tana = 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 $w = \$||s^1 \odot s^2 \odot \ldots \odot s^t||\$$, where $||$ denotes string concatenation and $s^i \overset{l}{\rightsquigarrow} s^{i+1}$ for all i. If $S = \left\{s^1, \ldots, s^t\right\}$ is a multiset of shingles, we will use $\Phi^{-1}(S) \subseteq \Sigma^*$ 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,

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

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 shingles 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 $w =$ katana with the shingling $I(w) = \{\$\mathsf{k}, \mathsf{ka}, \mathsf{at}, \mathsf{ta}, \mathsf{an}, \mathsf{na}, \mathsf{n}\$\}$. 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.

Protocol 1 transforms a string that is not uniquely decodable into one that is uniquely decodable by merging shingles and suitably modifying Algorithm 1 to handle a heterogeneous collection of arbitrarily-sized shingles. The main new technical challenge in this protocol is embodied in Step 4, in which the protocol must efficiently determine whether the shingles it has are uniquely decodable and, if not, merge shingles (and any metadata) until a uniquely decodable collection of shingles is produced.

1. Split σ into a set S_σ of length l shingles, with the i^{th} shingle of the string denoted s_i. Similarly split τ into S_τ.
2. Reconcile sets S_σ and S_τ.
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_σ^i is not uniquely decodable
   Merge the last two shingles added to S_σ^i.
5. Exchange indices of merged shingles.
6. Uniquely decode S_σ^i and S_τ^i on the remote hosts.

Protocol 1: Reconciliation of remote strings σ and τ.

B. Modifications to Algorithm 1

The string reconciliation protocol described in this section requires the use of a modified form of the unique-decodability algorithm from Section III, one in which shingle sizes may vary in length.

1) Checking Unique Decodability: Algorithm 2 generalizes Algorithm 1 to shingles of length $\geq l$, for fixed l; an analysis of its complexity is provided in Section IV-C2. It is based on the following lemma, which was first proved in [13] for bigrams but is readily extended to arbitrary shinglings.

Lemma 1. *A shingle set S is uniquely decodable iff there is exactly one Eulerian cycle in the corresponding De Bruijn graph $G(S)$ that starts and ends with* \$.

The following theorem establishes the correctness of Algorithm 2.

Theorem 2. *Algorithm 2 returns **true** iff its input set S is uniquely decodable.*

Proof: From Lemma 1 we know that to determine the unique decodability of S is equivalent to determining the existence of a unique Eulerian cycle in G that starts and ends with the special delimiter \$.

Completeness: Given an input set S that makes Algorithm 2 return **true**, what needs to be proved is that $G(S)$ has a unique Eulerian cycle. Assume that after S is processed by Algorithm 2 all the labels in $G(S)$ are fixed; we now restart from \$ along the Eulerian cycle to see if there were any opportunities to diverge from the cycle we found to produce different Eulerian cycle in $G(S)$. During the traversal, there are four cases at any vertex v:

- **case 1**: v is labeled as **NOT IN CYCLE**
- **case 2**: v is labeled as **IN CYCLE** and has exactly one out-going edge;
- **case 3**: v is labeled as **IN CYCLE** and has two out-going edges;
- **case 4**: v is labeled as **IN CYCLE** and has more than two out-going edges;

In case 1, Algorithm 2 only visited v once, meaning that any traversal on G(S) must leave v along the only available edge. In case 2, since v has only one out-going edge, any traverse must leave v along the same edge. In case 3, there are two out-going edges of v. Suppose the traversal leaves v from one of the two edges first, denoted $e1$, and returns to v at some later point in order to traverse the second out-going edge, denoted $e2$. Note that by returning to v for the first time the traversal already forms a cycle, denoted C_{e1}, in which $e1$ is included while $e2$ is not. Were the traversal to leave on $e2$ and return to v again, it would cause an intrusion on C_{e1} and Algorithm 2 would return **false**. Bounded by this, any traversal to v must leave along $e1$ all but the last time, there is no opportunity to diverge from the existed cycle at v. In light of case 3, case 4 is therefore not possible since any path that leaves v along its second out-going edge is not allowed to return.

Soundness: To prove that Algorithm 2 returns **true** if its input set S is uniquely decodable is equivalent to proving that a shingle set S is **not** uniquely decodable if Algorithm 2 returns **false**.

Algorithm 2 only returns **false** when an intrusion on an existing cycle is detected at vertex v_x, at which time we know that: (i) v_x has been marked as **VISITED**, so that the path between the last visit and the current visit forms a cycle. (ii) v_x is already in another cycle including its parent edge, which is necessarily different from the cycle just found in (i), since an intrusion is only detected when stepping onto v_x along an edge other than its recorded parent edge. Since v_x is in two different cycles that both return to v_x, there are at least two different Eulerian cycles on

Input: *Ordered* shingle set $S = \{s_1, s_2, s_3, ..., s_n\}$ constructed from shingling string w with minimum shingle length l;
Output: ***true*** if S is uniquely decodable and **false** otherwise;
1 initialize the graph $G(S)$ with vertex set V, each $v_i \in V$ represents the length $l-1$ prefix of s_i, $v_i = v_j$ if s_i and s_j have the same prefix;
2 initialize each $v \in V$ as **UNVISITED**;
3 initialize each $v \in V$ as **NOT IN CYCLE**;
4 initialize each $\Psi(v)$ as empty;
5 **for** $i \leftarrow 1$ ***to*** $|S|$ **do**
6 **case** ***1:*** v_i ***is UNVISITED***
7 mark v_i as **VISITED**;
8 **endsw**
9 **case** ***2:*** v_i ***is NOT IN CYCLE***
10 $j \leftarrow i$;
11 **repeat**
12 **if** v_j ***is NOT IN CYCLE*** **then**
13 label v_j as **IN CYCLE**;
14 $\Psi(v_j) \leftarrow s_{j-1}$;
15 **end**
16 $j \leftarrow j - 1$;
17 **until** $v_j = v_i$;
18 **endsw**
19 **case** ***3:*** v_i ***is IN CYCLE***
20 **if** $s_{i-1} = \Psi(v_i)$ **then** /* stepping along an existing cycle */
21 do nothing;
22 **else** /* intruding an cycle from a different edge */
23 **return** ***false***
24 **end**
25 **endsw**
26 **end**
27 **return** ***true***

Algorithm 2: Checking the unique decodability of a shingle set

G(S) can be found based on which cycle is visited first. Lemma 1 tells us S is not uniquely decodable if $G(S)$ has two Eulerian cycles, and *Soundness* is proved. ■

2) Patching Unique Decodability: In cases where an unique decoding of a shingle set does not exist, Algorithm 3 provides method of merging some of the shingles in order to produce uniquely decodable shingle set that decodes to the same string. We call the checking and (potential) merging process *patching* the unique decodability of a shingle set. Algorithm 3 executes in almost the same way as Algorithm 2 to check the unique decodability of the input shingle set. We only change the boolean label **INCYCLE** in Algorithm 2 to a counter $\Phi(v)$, which keeps track of how many cycles (not necessarily distinct) that include vertex v have been detected at the time. If the input shingle set fails a unique-decodability check, Algorithm 3 makes use of Procedure **deCycle** and its Sub-Procedure **mergePrevious** in order to recover the unique decodability property for the working shingle set.

Procedure deCycle is called at line 27 of Algorithm 3, and its function is to delete one cycle at v_i by merging all the edges backwards from current to just before the last occurrence of v_i. As a sub-procedure of deCycle, mergePrevious is called when one edge (s_{k-1}) needs to be merged with its previous edge (s_{k-2}), with different decisions being made at each merge, depending on the state of vertex v_k.

Theorem 3. *The shingle set S' returned by Algorithm 3 is uniquely decodable.*

Lines 1 to 25 work in the same way as in Algorithm 2, and therefore when Algorithm 3 reaches Line 26, *UD*=**false** iff the shingle set seen so far is **NOT** uniquely decodable; the rest of the proof is developed with the

Input: *Ordered* shingle set $S = \{s_1, s_2, s_3, ..., s_n\}$ constructed from shingling string w with minimum shingle length l;
Output: Shingle set S' that decodes uniquely to w;

initialize the graph $G(S)$ with vertex set V, each $v_i \in V$ represents the length $l-1$ prefix of s_i, $v_i = v_j$ if s_i and s_j have the same prefix;
initialize each $v \in V$ as **UNVISITED**, each $\Phi(v) = 0$, each $\Psi(v)$ as **null**;
initialize *UD*, the boolean flag indicating unique decodability, to be **true**;
$i \leftarrow 1$;
while $i \leq |S|$ **do**
    case ***1:*** v_i *is* ***UNVISITED***
        mark v_i as **VISITED**;
    endsw
    case ***2:*** v_i *is* ***VISITED*** *and* $\Phi(v_i) = 0$
        $j \leftarrow i$;
        repeat
            if $\Phi(v_j) = 0$ **then**
                $\Psi(v_j) \leftarrow s_{j-1}$;
            end
            $\Phi(v_j) \leftarrow \Phi(v_j) + 1$;
            $j \leftarrow j - 1$;
        until $v_j = v_i$;
    endsw
    case ***3:*** v_i *is* ***VISITED*** *and* $\Phi(v_i) > 0$
        if $s_{i-1} = \Psi(v_i)$ **then** `/* stepping along an existing cycle */`
            do nothing;
        else `/* intruding an cycle from a different edge */`
            UD=**false**;
        end
    endsw
    if *UD*=***false*** **then**
        (S, G, i) $\leftarrow$ deCycle(S, G, i) `/* delete one cycle at` v_i `*/`;
        UD$\leftarrow$**true**;
    end
end
$i \leftarrow i + 1$; **return** S

Algorithm 3: Patching the unique decodability of a shingle set

Input: S: shingle set; G: de Bruijn graph of S; i, index number of current vertex
Output: modified input (S, G, i), with updated state Ψ and Φ to reflect cycle deletion

$k \leftarrow i$;
repeat
    $k \leftarrow k - 1$;
    (S, G) $\leftarrow$ mergePrevious(S, G, k); `/* merge` s_k `with` s_{k-1} `*/`;
until $v_k = v_i$;
delete s_k to s_{i-1} from S;
$i \leftarrow k - 1$;
return *(S,G,i)*

Procedure deCycle(S, G, i), deleting cycle by merging edges backwards from v_i until $\Psi(v_i)$ is merged once

Input: S: shingle set; G: de Bruijn graph of S; k, index number of current vertex
Output: modified input (S, G)

if $\Phi(v_k) = 0$ **then** /* v_k is not in cycle */
  mark v_k as **UNVISITED**;
else if $\Phi(v_k) = 1$ **then** /* v_k is in exactly one cycle, this merge will break the cycle */
  $j \leftarrow k$;
  repeat
    $\Phi(v_j) \leftarrow \Phi(v_j) - 1$;
    if $\Phi(v_j) = 0$ **then**
      $\Psi(v_j) \leftarrow$ **null**;
    end
    $j \leftarrow j - 1$;
  until $v_j = v_k$;
else /* v_k is in more than one indistinct cycles, this merge will reduce the number of cycles by 1 */
  $\Phi(v_k) \leftarrow \Phi(v_k) - 1$
end
Append the l-th to the last character of s_k to s_{k-1};
return *(S,G)*

Procedure mergePrevious(S, G, k), merging s_k with s_{k-1} and maintaing relevant metadata

aid of Lemma 4.

Lemma 4. *When UD=**false** at Line 26 of Algorithm 3 for some index i, then*

- *1) when it next reaches Line 29, $\Phi(v_i)$ will be reduced by one, and v_i is involved in one fewer cycles;*
- *2) the next iteration of while loop (from Line 5) will restart at v_i;*
- *3) by the next time UD=**true** at Line 26 of Algorithm 3, the intruded cycle will be broken.*

The proof of Lemma 4 has been omitted due to space considerations.

Proof of Theorem 3: From Theorem 2, we know that Algorithm 3 takes a shingle set as input, and detects whether it is uniquely decodable, more precisely, whether there is an intrusion on existing cycle(s) at the time. By Lemma 4, if *decode*=**false**, Algorithm 3 repeatedly breaks the intruded cycle(s) and restarts the check at the same vertex, until all the intruded cycles that the current vertex is involved are broken, at which point *UD*=**true**. In essence, if the input set is it is not uniquely decodable, Algorithm 3 "patches" it by merging some of the shingles together, and such merging cannot increase the number of decoding needed to reconstruct the string, for all the merges are designed to take place on existing cycles and therefore cannot introduce new cycles during the patching process. After the patching, Algorithm 3 always exits with *UD*=**true**, indicating that it always returns an uniquely decodable set. ■

C. Analysis

1) Data Structures: We can use an *array* and a double-linked *list* to store the vertex information. A two-dimensional array can be used to store the state information of all vertices. The rows of this array are indexed by vertex number, for each row representing some vertex v, it contains the state information of v such as the **VISITED** boolean, and values $\Phi(v)$ and $\Psi(v)$. The total number of rows of the array is $|\Sigma|^l$, and the number of non-zero rows is at $n - l - 1$ (excluding shingles that contain the delimiter), in practice, it is common that $n \ll |\Sigma|^l$.

Both Algorithm 2 and 3 take an *ordered* shingle set as input, and we use a doubly-linked *list* to store all these input shingles in order of occurrence. The i-th element of the list will be denoted L_i, and by design, $L_i = s_i$.

2) Runtime Analysis: Algorithm 2:

Theorem 5. *Algorithm 2 has $\Theta(|\Sigma|)$ preprocessing time complexity and $\Theta(n)$ on-line time complexity.*

Proof: We list the detailed run time analysis as below.

- Lines 1-4. Initialization of De Bruijn graph G and its vertex set V, can be accomplished in constant time with sparse storage, with the two-dimensional array implementation described in Section IV-C1. Note that for G, only vertices need to be stored in the array while edges are essentially the input shingles, which are already kept in another list.
- Lines 6-8. Since the array containing the state information of vertices has constant time access, the time cost of this step is constant.
- Lines 9-18. All the input vertices are kept in an order *list* (see Section IV-C1), and the iteration at lines 11-17 can then be accomplished by scanning backwards through the list. Since l is constant, an operation like comparison, searching or reading/writing can be done in constant time.
- Lines 19-25. Comparing shingles of length l takes constant time, again because l is constant.

■

3) Runtime Analysis: Algorithm 3:

Theorem 6. *Algorithm 3 has linear time complexity* $\Theta(n + |\Sigma|)$ *running on string* w *of length* n*.*

Proof: Details are as followed:

- Lines 1-29. Though more metadata need to be maintained compared to Algorithm 2, all the operations can still be accomplished in constant time.
- Procedure deCycle. The cost of merging two shingles with length-l overlap is constant, because l is assumed constant. In the worst-case, all the shingles are merged together and the output shingle set S is uniquely decodable by itself, the sequence of $n-l+1$ merges takes $\Theta(n)$ time in *aggregation*. Therefore, the *amortized* cost per call of procedure deCycle is $\Theta(n)/n$.

■

D. 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 needed to be merged for transforming a set S into uniquely decodable. The work in [1] provides some hints in estimating the "safe" length of shingling random bit-strings into uniquely decodable set without additional merges. Specifically, for length-n Bernoulli string (strings of n random bits in which each bit is 0 with probability $p > 0.5$), it can expected that each node in the corresponding De Bruijn graph of length l shingles to have only one outgoing edge if $l \geq n + 1 + \dfrac{W(-\ln(p)p^{-n})}{\ln p} = O(\log n)$, where $W(\cdot)$ is the Lambert W function [4]. This suggests that the minimum length of shingling needs to be sized logarithmically with the string length in order to avoid high-frequency merges.

Thus, when the two strings are composed of random iid bits, then, under the appropriate choice of l, 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. Empirical evidence suggests that this length is tight, in the sense that decreasing it a little produces significant number of possible decodings in the corresponding shingle set, and a similar effect has been observed for strings generated from simple Markov processes and natural English-language text.

E. 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 [18]. 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 [2, 15, 26], and the two rateless streams can be combined by considering them inputs to yet a third rateless encoding.

V. 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. Finally, it is possible that context-dependent shingling, as in [30].

Acknowledgments

We would like to thank Arnold Filtzer and Omrit Naftali for their helpful suggestions and improvements of this work.

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] 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.

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

[4] 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.

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

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

[7] 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.

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

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

[10] 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.

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

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

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

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

[15] 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.

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

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

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

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

[20] 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.

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

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

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

[24] 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.

[25] 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.
[26] A. Shokrollahi. Raptor codes. *Information Theory, IEEE Transactions on*, 52(6):2551 –2567, june 2006.
[27] 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.
[28] A. Tridgell. *Efficient algorithms for sorting and synchronization*. PhD thesis, The Australian National University, 2000.
[29] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191 – 211, 1992.
[30] Hao Yan, U. Irmak, and T. Suel. Algorithms for low-latency remote file synchronization. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 156 –160, april 2008.