Efficient determination of the unique decodability of a string

Arnold Filtser*, Jiaxi Jin†, Aryeh Kontorovich* and Ari Trachtenberg†
* Computer Science, Ben-Gurion University, Beer Sheva, Israel; supported by the Frankel Center
† 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 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 a motivating 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 many settings. In information theory, this is related to the α-*edits* or *string reconciliation* problem [1], 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 [2], 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 [4].

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 [5] and Ukkonen [6], 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 [7]. With this observation, it is possible to produce a deterministic finite state machine on $\exp(\Omega(|\Sigma|\log|\Sigma|))$ states [8]. and a non-deterministic one on $O(|\Sigma|^3)$ states [9]. 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 [9] 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).

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, as is a motivating application to the α-edits problem. We close with concluding remarks and remaining open theoretical questions in Section IV.

II. Related work

A. Edit distance

Orlitsky [1] 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 (i.e. insertions or deletions) 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 other solutions, such as hash-based approaches [10, 11], an interactive protocol [12], and a protocol based on delta-compression [13].

B. Reconciliation

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 [14] involves translating the set elements into an equivalent *characteristic polynomial*, thus reducing set reconciliation into an equivalent problem of rational function interpolation. The resulting algorithm requires one message of roughly bm bits and bm^3 computation to reconcile two sets that differ in m entries,

though this can be made expected bm communication and computation through interaction [15].

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:

$$\{\mathsf{at}, \mathsf{an}, \mathsf{ka}, \mathsf{na}, \mathsf{ta}\}. \quad (1)$$

In order to reconcile two strings σ_A and σ_B, the protocol STRING-RECON [16] first shingles each string, then reconciles the resulting sets, and then puts the shingles back together into strings in order to complete the reconciliation.

The process of recombining shingles of length l into a string involves the construction of a modified de Bruijn graph. In this graph, each shingle corresponds to an edge, with weight equal to the number times the shingle occurs in the string. 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 delimiter \$ is used to mark the beginning and end of the string.

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.

C. Unique decoding

Ukkonen [6] conjectured that two strings with the same shingles are related through two types of string transformations, and Pevzner [5] proved this true. Motahari et al [17] provided asymptotic bounds on how many shingles are needed to reconstruct a string.

It was later shown in [7] that the collection of strings having a unique reconstruction from the shingles representation is a regular language, and Li and Xie [8] gave an explicit construction of a deterministic finite-state automaton (DFA) recognizing this language.

III. String reconciliation

Our string reconciliation protocol in [18], which is a refinement of the shingling approach in [16] based upon a transformation to an instance of set reconciliation [14], serves as a clear motivation for our main results, Algorithms 1 and 2.

A. Definitions

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 we can rewrite $s = s'u$ and $t = ut'$ for strings s', t' and $|u| \geq l-1$. 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 $s^1 \overset{l}{\rightsquigarrow} s^2 \overset{l}{\rightsquigarrow} \dots \overset{l}{\rightsquigarrow} s^t$ is said to *represent* the word $w \in \Sigma^*$ if $w = \$||s^1 \odot s^2 \odot \dots \odot s^t||\$$, where $||$ denotes string concatenation and $s^i \overset{l}{\rightsquigarrow} s^{i+1}$ for all i. If $S = \{s^1, \dots, s^t\}$ is a multiset of shingles, we use $\Gamma(S) \subseteq \Sigma^*$ to denote the collection of all words represented by S. We refer to the members of $\Gamma(S)$ as the *decodings* of S, and say that S is uniquely decodable if $|\Gamma(S)| = 1$. A *shingling* I of a word $w = w_1 \dots w_t \in \Sigma^*$ is a set of shingles of w that represents w. We say that I is an uniquely decodable shingling of w if $|\Gamma(I(w))| = 1$.

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.

Protocol 1 [18] transforms a string that is not uniquely decodable into one that is uniquely decodable by merging shingles. The main new technical challenge in this protocol is embodied in Step 4, in which the protocol must efficiently determine whether its shingles are uniquely decodable and, if not, merge shingles (and any metadata) until a uniquely decodable collection of shingles is produced.

B. Unique decodability

The string reconciliation protocol described in this section requires the use of an algorithm that tests whether a given set of possibly different-length shingles admits a unique decoding, and this is accomplished by Algorithm 1.

1) Checking Unique Decodability: The correctness of Algorithm 1 rests upon Lemma 1, originally proved in [7] for bigrams but readily extended to larger shingles, and Theorem 2.

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

Theorem 2. *Algorithm 1 returns **true** iff its input set S is uniquely decodable.*

Proof: From Lemma 1 we know that unique decodability of S is equivalent to having a unique Eulerian cycle in G starting and ending with \$.

Completeness: Given an input set S that makes Algorithm 1 return **true**, what needs to be proved is that $G(S)$ has a unique Eulerian cycle. Assume that after S is processed by Algorithm 1 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**;

Algorithm 1: Checking the unique decodability of a shingle set

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 ∈ 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 ∈ V as UNVISITED;
3  initialize each v ∈ V as NOT IN CYCLE;
4  initialize each Ψ(v) as empty;
5  for i ← 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 ← i;
11         repeat
12             if v_j is NOT IN CYCLE then
13                 label v_j as IN CYCLE;
14                 Ψ(v_j) ← s_{j−1};
15             end
16             j ← j − 1;
17         until v_j = v_i;
18     endsw
19     case 3: v_i is IN CYCLE
20         if s_{i−1} = Ψ(v_i) then
21             do nothing;
22         else
23             return false
24         end
25     endsw
26 end
27 return true
```

- **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 1 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 1 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.

Soundness: Algorithm 1 only returns **false** when detecting an intrusion on an existing cycle 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, at least two different Eulerian cycles on G(S) exist so, by Lemma 1, S is not uniquely decodable. ■

2) Patching Unique Decodability: In cases where a unique decoding of a shingle set does not exist, Algorithm 2 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 2 executes in almost the same way as Algorithm 1 to check the unique decodability of the input shingle set. We only change the boolean label **INCYCLE** in Algorithm 1 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 2 makes use of Procedure **deCycle** and Sub-Procedure **mergePrevious** to recover the unique decodability property for the working shingle set.

Procedure deCycle is called at line 27 of Algorithm 2, 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 2 is uniquely decodable.*

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

Lemma 4. *When UD=**false** at Line 26 of Algorithm 2 for some index i, then*

1) *when it next sees 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 2, the intruded cycle will be broken.*

C. Analysis

Theorem 5. *Algorithm 1 requires $\Theta(|\Sigma|)$ preprocess time and $\Theta(n)$ on-line time for constant shingle length. Algorithm 2 has linear time complexity $\Theta(n+|\Sigma|)$ running on string w of length n.*

Algorithm 2: Patching the unique decodability of a shingle set.

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' decoding uniquely to w;

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**, each $\Phi(v) = 0$, each $\Psi(v)$ as **null**;
3 initialize *UD*, the boolean flag indicating unique decodability, to be **true**;
4 $i \leftarrow 1$;
5 **while** $i \leq |S|$ **do**
6 **case** ***1:*** v_i *is* ***UNVISITED***
7 mark v_i as **VISITED**;
8 **endsw**
9 **case** ***2:*** v_i *is* ***VISITED*** *and* $\Phi(v_i) = 0$
10 $j \leftarrow i$;
11 **repeat**
12 **if** $\Phi(v_j) = 0$ **then**
13 $\Psi(v_j) \leftarrow s_{j-1}$;
14 **end**
15 $\Phi(v_j) \leftarrow \Phi(v_j) + 1$;
16 $j \leftarrow j - 1$;
17 **until** $v_j = v_i$;
18 **endsw**
19 **case** ***3:*** v_i *is* ***VISITED*** *and* $\Phi(v_i) > 0$
20 **if** $s_{i-1} = \Psi(v_i)$ **then**
21 do nothing;
22 **else**
23 *UD*=**false**;
24 **end**
25 **endsw**
26 **if** *UD*=***false*** **then**
27 (S, G, i) $\leftarrow$ deCycle(S, G, i);
28 *UD*$\leftarrow$**true**;
29 **end**
30 **end**
31 $i \leftarrow i + 1$; **return** S

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

1 $k \leftarrow i$;
2 **repeat**
3 $k \leftarrow k - 1$;
4 (S, G) $\leftarrow$ mergePrevious(S, G, k);
5 **until** $v_k = v_i$;
6 delete s_k to s_{i-1} from S;
7 $i \leftarrow k - 1$;
8 **return** *(S,G,i)*

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

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

1 **if** $\Phi(v_k) = 0$ **then**
2 mark v_k as **UNVISITED**;
3 **else if** $\Phi(v_k) = 1$ **then**
4 $j \leftarrow k$;
5 **repeat**
6 $\Phi(v_j) \leftarrow \Phi(v_j) - 1$;
7 **if** $\Phi(v_j) = 0$ **then**
8 $\Psi(v_j) \leftarrow$ **null**;
9 **end**
10 $j \leftarrow j - 1$;
11 **until** $v_j = v_k$;
12 **else**
13 $\Phi(v_k) \leftarrow \Phi(v_k) - 1$
14 **end**
15 Append the l-th to the last character of s_k to s_{k-1};
16 **return** *(S,G)*

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 a two-dimensional array.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 ordered *list*, and the iteration at lines 11-17 can then be accomplished by scanning backwards through the list.
- Lines 19-25. Comparing shingles of length l takes constant time, again because l is constant.

■

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 [14]. 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. Precise bounds on the number of shingles that need to be merged when transforming a set S into a uniquely decodable are difficult to obtain. The results in [16] suggest that, at least in the model of iid input sequences of length n, a "safe" shingle length is $O(\log n)$. Inspired by the techniques in [19], our result below sharpens the analysis in [16].

Theorem 6. *If the input string w is drawn uniformly at random from Σ^n and shingled into length ℓ shingles, then the expected number of calls to procedure deCycle in Algorithm 2 is at most $\binom{n-\ell+1}{2}|\Sigma|^{-\ell}$.*

Note that this expectation is less than 1 if $\ell \geq \log_{|\Sigma|} n^2 + 2$.

Proof: Put $s = |\Sigma|$ and let $I_{i,j}$ be the 0-1 indicator variable of the event $w[i : i+\ell-2] = w[j : j+\ell-2]$, that is, that the length-ℓ substrings starting at i and j, respectively, are identical. Note that the number of calls is upper bounded by $\sum_{i<j} I_{i,j}$, since each call is triggered by some pair of identical length $\ell-1$ substrings, corresponding to a revisited vertex in the de Bruijn graph.

We claim that for all $1 \leq i < j \leq n-\ell$, $\mathbb{E}[I_{i,j}] = \Pr[I_{i,j} = 1] = s^{-\ell}$. To prove this, let us define the index sets $A, B, C \subseteq [n]$ as follows: $A = \{i, i+1, \ldots, i+\ell-2\}$, $B = \{j, j+1, \ldots, j+\ell-2\}$, and $C = A \cap B$, with $w[A]$, $w[B]$, $w[C]$ being the substrings of characters at the corresponding indices. We will consider the cases $C = \emptyset$ and $C \neq \emptyset$ separately. In the first case, the result derives from the independence of $w[A]$ and $w[B]$, each of length ℓ. To see the second case, observe that the characters of $w[A \setminus C]$ are independent and completely determine the remaining characters in $w[C]$ and $w[B]$. As such, $\Pr[I_{i,j} = 1] = \frac{s^{\ell-c}}{s^{2\ell-c}} = s^{-\ell}$.

The expected number of calls is thus upper bounded by $\mathbb{E}\left[\sum_{i<j} I_{i,j}\right] = \binom{n-\ell+1}{2} s^{-\ell}$. ∎

We can combine Theorem 6 with the communication complexity analysis to get an upper bound on communication for Algorithm 2.

Corollary 7. *Consider two strings drawn uniformly at random from Σ^n that differ by α edits. The expected communication needed to reconcile these strings using Algorithm 2 is at most*

$$\Theta\left(\alpha \log_s^2(n)\right),$$

using shingles of length $\ell = 3\log_s(n)$ and $s = |\Sigma|$.

Proof: The analysis at the beginning of the section indicates a communication complexity of $\alpha\ell^2 + m\log(n-\ell+1)$, for m merges of length ℓ shingles. Replacing m by the expectation in Theorem 6 times the string length n, and length ℓ as in the statement produces a bound of $\alpha l^2 + \left(\frac{n^2}{s^l}\right) log(n-l+1)$ $\leq$ $9\alpha log_s^2(n) + \frac{log(n-l+1)}{n}$, which asymptotically converges to the result. ∎

IV. 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 preprocessing, 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] A. Orlitsky, "Interactive communication: Balanced distributions, correlated files, and average-case complexity." in *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, 1991, pp. 228–238.

[2] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM J. Comp.*, vol. 38, no. 1, pp. 97–139, 2008.

[3] M. Chaisson, P. A. Pevzner, and H. Tang, "Fragment assembly with short reads," *Bioinformatics*, vol. 20, no. 13, pp. 2067–2074, 2004.

[4] X. Shi, H. Xie, S. Zhang, and B. Hao, "Decomposition and reconstruction of protein sequences: The problem of uniqueness and factorizable language." *Journal of the Korean Physical Society*, vol. 50, no. 1I, pp. 118–123, 2007.

[5] P. Pevzner, "DNA physical mapping and alternating Eulerian cycles in colored graphs," *Algorithmica*, vol. 13, pp. 77–105, 1995.

[6] E. Ukkonen, "Approximate string-matching with q-grams and maximal matches," *Theoretical Computer Science*, vol. 92, no. 1, pp. 191 – 211, 1992.

[7] L. Kontorovich, "Uniquely decodable n-gram embeddings," *Theor. Comput. Sci.*, vol. 329, no. 1-3, pp. 271–284, 2004.

[8] Q. Li and H. Xie, "Finite automata for testing composition-based reconstructibility of sequences," *J. Comput. Syst. Sci.*, vol. 74, no. 5, pp. 870–874, 2008.

[9] A. L. Kontorovich and A. Trachtenberg, "Unique decodability for string reconciliation," submitted. [Online]. Available: http://arxiv.org/abs/1111.6431

[10] G. Cormode, M. Paterson, S. Sahinalp, and U. Vishkin, "Communication complexity of document exchange," in *SODA*, 2000, pp. 197–206.

[11] A. Tridgell, "Efficient algorithms for sorting and synchronization," Ph.D. dissertation, Australian National University, 2000.

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

[13] T. Suel, P. Noel, and D. Trendafilov, "Improved file synchronization techniques for maintaining large replicated collections over slow networks," in *ICDE*, 2004, pp. 153–164.

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

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

[16] S. Agarwal, V. Chauhan, and A. Trachtenberg, "Bandwidth efficient string reconciliation using puzzles," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 11, pp. 1217–1225, 2006.

[17] A. Motahari, G. Bresler, and D. Tse, "Information theory of dna sequencing." [Online]. Available: http://arxiv.org/abs/1203.6233v2

[18] A. L. Kontorovich and A. Trachtenberg, "String reconciliation with unknown edit distance," presented in part at ITA 2012. Also submitted elsewhere.

[19] M. Dyer, A. Frieze, and S. Suen, "The probability of unique solutions of sequencing by hybridization," *Journal of Computational Biology*, vol. 1, no. 2, pp. 105–110, Summer 1994.