Summary - Efficiently Decoding Strings From Their Shingles

Bowen Song[1]
June 18, 2018

Abstract— **This report is a part of an independent study for set and string reconciliation problems in distributed systems. The paper [1] studies the problem of reconstructing string from a set of un-ordered substrings known as shingles. There may exist multiple strings decodable from the same set of shingles. The paper provides algorithms to determine if a string can get recovered as a unique choice from its shingle set and to fix a none-uniquely decodable shingle set.**

I. Introduction

The paper proposes an online streaming algorithm to determine if a shingle set with two characters per shingle is uniquely decodable (UD). The algorithm delivers a linear time and space complexity and a constant pre-processing time.

The paper compares its protocol with two preexisting methods of determining unique decodability of a shingle set. The first method is exhaustively finding proof of other existing strings from a shingle set through transformations where a string is tested against transposition and rotation. This method is rather inefficient since the worst case for successfully verifying a uniquely decodable shingle set reaches computation complexity of $\Theta(n^4)$. The second method is based on building deterministic and non-deterministic finite automatons, DFA and NFA. While a DFA provides an efficient online testing procedure with an example such as [2], the space complexity could potentially be substantial. The NFA, on the other hand, would require a significant computational cost.

II. Algorithm Overview

The proposed online streaming algorithm determines whether a string is uniquely recoverable if broken down into a set of shingles, character-based bigrams. The algorithm evaluates the original string by going through each character of the string once and stores the outcome of each increment in a directed bigram graph. First, the algorithm creates a directed bigram graph with only vertices based on shingles that has no edges. The algorithm, then, links these vertices by advancing character by character from the string to evaluate if the string is UD. Each vertex has two boolean properties of whether the vertex belongs to a cycle and if it is previously visited. Each edge connecting two vertices contains a value of occurrence of the vertices relation. The string is considered not UD if an incremental character belongs to an existing cycle in the string with a different prefix or an incremental character is seen before and has a different prefix.

[1]B. Song is with Department of Electrical and Computer Engineering, Boston University, Boston MA, `sbowen@bu.edu`

III. Adapting To String Reconciliation Protocol

The problem with directly using this online streaming algorithm in a string reconciliation context is that the algorithm evaluates the string on whether it is UD rather than the shingle set. However, in a two-way shingle-based string reconciliation protocol where two strings from Alice and Bob are broken into shingles and reconciled as two sets; the shingle sets are no longer the shingles from the original UD strings. A set of shingles from a UD string after some changes would still be none-UD which defeats the purpose of the verification. Therefore, this algorithm needs some adaptation to function correctly.

In a shingle-based string reconciliation protocol, host Alice and Bob both hold a string similar to the other and break their strings into shingles of q length. The bigram algorithm can be easily adapted into q-gram, q characters shingle, by dividing a q-gram before its last character into two parts. Using a set reconciliation protocol [3], Alice and Bob both hold the reconciled set of shingles. The adaptation to a string reconciliation protocol is to merge none-UD causing shingles, merging bigram to q-gram, until the set is UD. The merging protocol may produce uneven length shingles, but would not affect the decoding process. The merging protocol merges by combining two bigram vertices into one. After necessary merges, both Alice and Bob sort their set in canonical order and exchange indices. The last step is to backtrack a Eulerian cycle which is guaranteed by this algorithm to obtain a unique reconciled string.

IV. Performance Metric

Complexity	Time	Space	Communication
Online	$O(n)$	$O(\|\Sigma\|)$	N/A
Merging	N/A	N/A	$O(n\,lg(n))$

n = Character length of a sting
$|\Sigma|$ = number of elements in an Alphabet

TABLE I
Performance Metric

The online streaming algorithm can determine if a string is UD with absolute confidence. However, adapting the algorithm into a shingle-based string reconciliation protocol raises concerns about the increasing communication complexity and communication rounds. The merging protocol requires an exchange of indices in a two-way reconciliation or one additional round for a one-way scheme. This increase in rounds of communication may be a hold back for systems that are less interactive. In addition, the message size for the merging algorithm has the worst case of $O(n\,lg(n))$ for exchanging indices which would be dominating the communication complexity in a string reconciliation protocol such as [4].

V. Conclusion

This paper provides a way to verify whether a string is uniquely decodable once broken down into a set of shingles. The algorithm provides a merging technique to amend a none-decodable shingle set. While it increases some communication and computation costs, the algorithm guarantees a unique shingle set decoding. This uniqueness would reduce the computational complexity of string reconciliation since it is heavily driven by the amount of Eulerian cycles hidden in the shingle set [4]. The algorithm is efficient and capable of online streaming. The merging adaptation to string reconciliation protocol is potentially less efficient and is only applicable to full reconciliation without tolerating any small differences like the protocol in [5] since exchanging indices requires both hosts holding the same set of shingles.

References

[1] A. Kontorovich and A. Trachtenberg, "Efficiently decoding strings from their shingles," *arXiv preprint arXiv:1204.3293*, 2012.

[2] Q. Li and H. Xie, "Finite automata for testing composition-based reconstructibility of sequences," *Journal of Computer and System Sciences*, vol. 74, no. 5, pp. 870–874, 2008.

[3] Y. Minsky and A. Trachtenberg, "Practical set reconciliation," in *40th Annual Allerton Conference on Communication, Control, and Computing*, vol. 248, 2002.

[4] S. Agarwal, V. Chauhan, and A. Trachtenberg, "Bandwidth efficient string reconciliation using puzzles," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1217–1225, 2006.

[5] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, "Robust set reconciliation," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 2014, pp. 135–146.