A Brief Summary of Suffix Trees and Their Applications

Joshua Glass

The College of Charelston  
CSCI 230-01

glassja@g.cofc.edu

**Keywords**

Suffix tree; data structures; genomics; data mining;

# INTRODUCTION

Suffix trees have become one of the most common and powerful data structures for solving an ever growing list of problems and challenges in the world of data processing and analysis. Suffix trees have become incredibly popular in fields such as bioinformatics for the ease with which genomes can be indexed and analyzed for patterns and similarities as well as finding applications in fields such as data science for mining and analyzing massive quantities of user generated text for patterns and frequency data. More than forty years of research on suffix trees has even led to the creation of a geometric suffix tree designed for indexing, pattern matching, comparison, and analysis of the 3-dimensional structures of proteins [9].

A suffix tree is a rooted indexed trie of all the distinct suffixes of some input string S (pp. 434-435, 564). They are strongly related to the suffix array which is an ordered array of all the distinct suffixes of S and in fact many proofs exist showing that a suffix tree is equivalent to it’s suffix array plus its LCP array. Because it is a trie it is fairly easy to compress as is evidenced by the plethora of papers on the compression of a suffix tree.

More importantly than the compression of a suffix tree is how efficiently one can build, access and compare entries in a suffix tree. The construction of a suffix tree, with an intelligent algorithm is O(|S|). Traversing a suffix tree in full is O(|S|) and is done from root to node with a depth-first traversal while searching is O(log(|S|)) just as any other tree is but, solving the exact matching problem for some pattern P where |P| is the size of the pattern we’re trying to match. In this case, the time complexity is O(|P| + log(|S|)). While these time complexities are beautiful there are a few drawbacks to suffix trees.

Constructing a suffix tree poses several problems largely due to the size of the tree compared to the size of the input string. While it is easy to accommodate a real world sized input string of 2-8 GB in main memory, the suffix tree for that input can easily be in the range of 50-150+ GB which will rapidly exceed any reasonable main memory size. As long as random disk access is minimized construction of suffix trees can be a realistically fast O(n).

# Indexing Genomes with Suffix Trees [1]

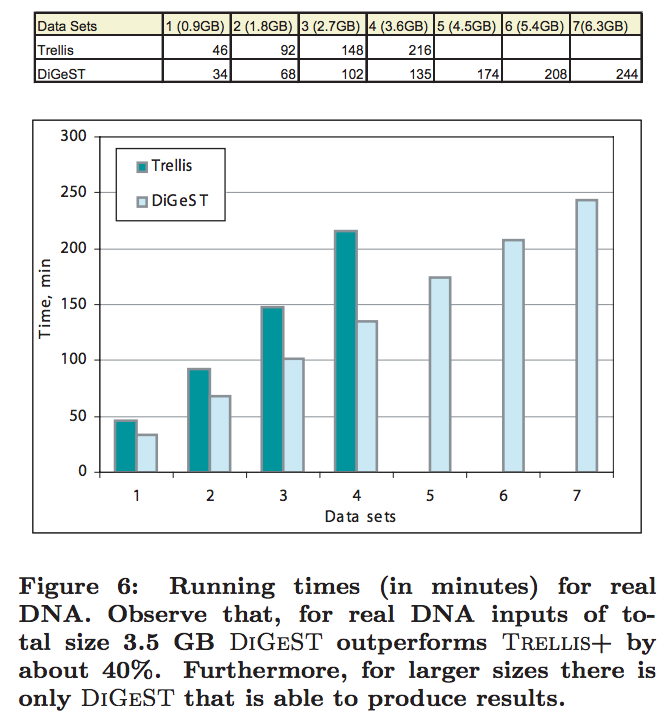
Barsky, Stege, Thomo, and Upton published a hybrid algorithm for the construction of on-disk suffix trees for indexing genome sequences in 2008 in the paper, A New Method for Indexing Genomes Using On-Disk Suffix Trees. Recognizing that even a relatively small input string can lead to a very large suffix tree they devised an algorithm to significantly cut down the number of random accesses to the input and passes over the disk data as well as ensuring the scalability to much larger genomes than was previously feasible [1].

Figure 1: A diagram taken from Barsky, Stege, Thomo, and Upton’s paper on Indexing Genomes with On-Disk Suffix Trees demonstrating the results of their tests running DiGeST against Trellis+ on various input sets [1]

## The Algorithm

The algorithm, called Disk-Based Genomic Suffix Tree (DiGeST) is designed such that only two passes over the data saved to the disk are needed and such that random access to the input string is greatly reduced [1] and accomplishes this using three stages. The first stage consists of a novel method of splitting the original input string into several, roughly equal sized input strings. Once the input has been broken in to several smaller problems the suffixes within each smaller file are sorted using Larsson’s algorithm and a suffix subtree is built for each of the smaller inputs and a prefix is added for efficient merging [1]. Once all of the suffix subtrees have been built they are read into buffers and organized before being sent to the disk.

## The Advantages and Accessing of the Suffix Tree

Because the suffix subtrees are small, lexicographically sorted, and technically independent of each other it is trivial to find the appropriate subtree and load it into main memory for analysis purposes. The typical case is to check for an exact pattern match in the input string. This is accomplished using a “blind search” [1] which requires only a single random access to the input string in disk memory per attempted match which results in significant performance increases. In the case that a traversal of the entire suffix tree is required a depth-first traversal is the best way to traverse the tree because each subtree is simply read into main memory and the depth-first traversal is performed and then the next subtree is read in series [1]. The authors ran several tests of DiGeST against Trellis+, the most efficient algorithm prior to the development of DiGeST, and found that DiGeST was in fact much faster and much more efficient than Trellis+ was. 

# HTML Data Extraction with Suffix Trees

Xie, Fang, and Li Li’s paper, Extracting Data Records from Web Using Suffix Tree, demonstrates a unique and novel approach to website data mining via suffix trees as opposed to the various clustering techniques and algorithms that require learning to be be effective. By creating an input sequence from the HTML tags on a website and building a suffix tree from that sequence they are able to analyze the HTML document very efficiently to look for “data record” [2] regions in the HTML.

## The Algorithm

Xie, Fang, and Li Li developed an algorithm called “suffix tree-based data record extracting method (STEM)” [2] with the goal that it could function autonomously without human guidance and still produce accurate, complete, and noiseless data files. They begin by creating a string of HTML tags from the document of interest via the DOM tree of the document and assigning a unique integer to each path in the DOM tree. They then represent the web page as this sequence of numbers and treat repeated regions as data regions [2]. By converting the sequence into a suffix tree they change the problem of finding the data regions into a simple matter of finding a target node in a tree [2] which has a more favorable time complexity than an array. Traversals are done from root to node and therefore have a worst case time complexity of O(log n) where n is the number of nodes and searching for an exact match to a target pattern sequence in a suffix tree is O(|P| + log n) where |P| is the size of the pattern sequence. Xie, Fang and Li Li also implement some more complicated filtering patterns to eliminate nodes that cannot match the target sequence for a variety of reason such as being too short or the node is outside of the reasonable target region which further improves performance [2]. Though because the process of creating the input sequence from web pages and building the suffix tree itself are both O(n) time complexity processes so the entire STEM algorithm still runs in O(n) time overall.

## Advantages Discussion and Data

Xio, Fang, and Li Li ran test of STEM against both a clustering algorithm, TPC, and another algorithm, MDR [2]. The used three metrics to measure STEM against TPC and MDR, how similar the various data region patterns were based against a threshold value, how accurate the data region extraction process was, and how capable the algorithms were of avoiding what they refer to as “noise” [2] where noise is simply extraneous tags and formatting that doesn’t contain any real data. Their results showed that TPC had very poor similarity scores while STEM had consistently high similarity scores [2]. STEM also far exceeded MDR for accuracy of the data region but was not nearly as significantly an improvement over TPC, however when it came to ignoring noise in the HTML documents STEM was considerably more reliable than either TPC or MDR [2].

# Future Work and Conclusion

# Suffix trees have become a staple in the world of data processing and continue to find surprising applications in an incredibly diverse set of fields and problem areas from genomics to music analysis to data compression and even 3-dimensional structure analysis. The number of problems that suffix trees have provided elegant, efficient solutions to is nearly impossible to count but there is still plenty left to be done. While suffix trees do provide an efficient time complexity solution to a host of problems, construction of them continues to pose a challenge because of the sheer size of the inputs and the change in magnitude from input to tree. Jie Lin, Yue Jiang and Don Adjeroh seem to have developed a potential solution with their Virtual Suffix Tree variation of the standard suffix tree [8] though the implementation may still be fairly complicated.

# ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

# REFERENCES

1. Barsky, M., Stege, U., Thomo, A. and Upton, C. 2008. A new method for indexing genomes using on-disk suffix trees. In *Proceedings of the 17th ACM conference on Information and knowledge management* (CIKM '08). ACM, New York, NY, USA, 649-658.
2. Xie, X., Fang, Y. and Li Li, Z. Z. 2012. Extracting data records from web using suffix tree. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics* (MDS '12). ACM, New York, NY, USA, , Article 12 , 8 pages.
3. A. Crauser and P. Ferragina A Theoretical and Experimental Study on the Construction of Suffix Arrays in External Memory. *Algoritmica*, 32(1): 1-35, 2002.
4. P. Ferragina and R. Grossi The string B-tree: a new data structure for string search in external memory and its applications. *J. of the ACM*, 46(2): 1999.
5. Hung Chim, Xiaotie Deng. A new suffix tree similarity measure for document clustering. Proceedings of the 16th international conference on World Wide Web 2007.
6. Woong-Kee Loh, Heejune Ahn. A Storage-Efficient Suffix Tree Construction Algorithm for Human Genome Sequences. IEICE Transactions (IEICET) 94-D(12):2557-2560 (2011).
7. Weiss, M. A. Data Structures and Algorithm Analysis in Java 3rd Ed. Pearson, 2012
8. Lin, J., Jiang, Y. and Adjeroh, D. The Virtual Suffix Tree. *International Journal of Foundations of Computer Science, 20(6), 1109-1133.*
9. Shibuya, T. 2010. Geometric Suffix Tree: Indexing Protein 3-D Structures. J. ACM 57, 3 Article 15, (March 2010), 17 pages. DOI = 10.1145/1706591.1706595 http://doi.acm.org/10.1145/1706591.1706595