

Suffix Sorting via Shannon-Fano-Elias Codes*

Don Adjeroh and Fei Nan

Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506

The suffix sorting problem is to construct the suffix array for an input sequence. Given a sequence $T[0 \dots n-1]$ of size $n = |T|$, with symbols from a fixed alphabet Σ , ($|\Sigma| \leq n$), the suffix array provides a compact representation of all the suffixes of T in a lexicographic order. Traditionally, the suffix array is often constructed by first building the suffix tree for T , and then performing an inorder traversal of the suffix tree. The direct suffix sorting problem is to construct the suffix array of T directly *without* using the suffix tree data structure. Manber and Myers[3] were the first to propose suffix arrays as a new and conceptually simple data structure for online string searching. They suggest an $O(n \log n)$ algorithm to construct the suffix array with three to five times less space than the traditional suffix tree. While algorithms for linear time, linear space direct suffix sorting have been proposed [1, 2], the actual constant in the linear space is still a major concern, especially given that applications of suffix trees and suffix arrays (such as in whole-genome sequence analysis) often involve huge data sets. Puglisi *et al*[4] provide a comparison of different recently proposed linear time algorithms for suffix sorting.

We propose two algorithms for the direct suffix sorting problem. The first is a simple algorithm that runs in an $O(n)$ average time and space complexity, but with a worst case complexity in $O(n \log n)$ time and $O(n)$ space. The second algorithm improves the first algorithm to $O(n)$ time and space in the worst case. The improved algorithm requires only $7n$ bytes of storage, including the n bytes for the original string, and the $4n$ bytes for the suffix array. We take a general divide and conquer approach: Divide the sequence into two groups; Construct the suffix array for the first group; Construct the suffix array for the second group, based on the sorted suffix from the first; Merge the suffix arrays from the two groups to form the suffix array for the parent sequence; Perform the above steps recursively to construct the complete suffix array for the entire sequence. Given a string of length n , our algorithm runs in $O(n)$ worst case time and space.

Our algorithm differs from previous approaches in the use of a simple partitioning step, and how it exploits this simple partitioning scheme for conflict resolution, using the notion of conflict trees. The basis of our improved algorithm is an extension of Shannon-Fano-Elias codes used in information theory. The space requirement for the proposed algorithm is $7n$ bytes, including the n bytes required to store the original string. This is a significant improvement when compared with the $13n$ bytes required by the KS algorithm [1]. The method is also unique in its use of Shannon-Fano-Elias codes in efficient suffix sorting. To our knowledge, this is the first time information-theoretic methods have been used as the basis for solving the suffix sorting problem.

References

- [1] J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006.
- [2] P. Ko and A. Aluru. Space-efficient linear time construction of suffix arrays. *J. Discrete Algorithms*, 3(2-4):143–156, 2005.
- [3] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Computing*, 22(5):935–948, 1993.
- [4] S. J. Puglisi, W. F. Smyth, and A. Turpin. A taxonomy of suffix array construction algorithms. *ACM Computing Surveys*, 39(2):1–31, 2007.

*Partially supported by a DOE CAREER award: DE-FG02-02ER25541, and an NSF ITR grant: IIS-0228370.