

Augmenting Suffix Trees, with Applications

Yossi Matias¹ *, S. Muthukrishnan² **, Süleyman Cenk Şahinalp³ ***, and
Jacob Ziv⁴ †

¹ Tel-Aviv University, and Bell Labs, Murray Hill

² Bell Labs, Murray Hill

³ University of Warwick and University of Pennsylvania

⁴ Technion

Abstract. Information retrieval and data compression are the two main application areas where the rich theory of string algorithmics plays a fundamental role. In this paper, we consider one algorithmic problem from each of these areas and present highly efficient (linear or near linear time) algorithms for both problems. Our algorithms rely on augmenting the *suffix tree*, a fundamental data structure in string algorithmics. The augmentations are nontrivial and they form the technical crux of this paper. In particular, they consist of adding extra edges to suffix trees, resulting in Directed Acyclic Graphs (DAGs). Our algorithms construct these “suffix DAGs” and manipulate them to solve the two problems efficiently.

1 Introduction

In this paper, we consider two algorithmic problems, one from the area of Data Compression and the other from Information Retrieval. Our main results are highly efficient (linear or near linear time) algorithms for these problems. All our algorithms rely on the *suffix tree* [McC76], a versatile data structure in combinatorial pattern matching. Suffix trees, with suitably simple augmentations, have found numerous applications in string processing [Gus98,CR94]. In our applications too, we augment the suffix tree with extra edges and additional information. In what follows, we describe the two problems and provide some background information, before presenting our results.

* Department of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel; and Bell Labs, Murray Hill, NJ, 07974, USA; matias@math.tau.ac.il. Partly supported by Alon Fellowship.

** Bell Labs, Murray Hill, NJ, 07974, USA; muthu@research.bell-labs.com.

*** Department of Computer Science, University of Warwick, Coventry, CV4-7AL, UK; and Center for BioInformatics, University of Pennsylvania, Philadelphia, PA, 19146, USA; cenk@dc.s.warwick.ac.uk. Partly supported by ESPRIT LTR Project no. 20244 - ALCOM IT.

† Department of Electrical Engineering, Technion, Haifa 32000, Israel; jz@ee.technion.ac.il.

1.1 Problems and Background

We consider the *document listing problem* of interest in Information Retrieval and the *HYZ compression problem* from context-based Data Compression.

The Document Listing Problem. We are given a set of documents $T = \{T_1, \dots, T_k\}$ for preprocessing. Given a query pattern P , the problem is to output a list of all the documents that contain P as a substring.

This is different from the standard query model where we are required to output all the *occurrences* of P . The standard problem can be solved in time proportional to the number of occurrences of P in T using a suffix tree. In contrast, our goal in solving the document listing problem is to generate the output with a running time that depends on the number of documents that contain P . Clearly, the latter may be substantially smaller than the former if P occurs multiple times in the documents.

A related query is where we are required to merely report the *number* of documents that contain P . An algorithm that solves this problem in $O(|P|)$ time is given in [Hui92], which is based on data structures for computing lowest common ancestor (LCA) queries.

The document listing problem is of great interest in information retrieval and has independently been formulated in many scenarios (see pages 124-125 in [Gus98] for a “morbid” application), for example in discovering gene homologies [Gus98].

The HYZ Compression Problem. Formally the (α, β) -HYZ compression problem is as follows. We are given a binary string T of length t . We are asked to replace disjoint *blocks* (substrings) of size β with desirably shorter codewords. The codewords are selected in a way that it would be possible for a corresponding decompression algorithm to compute the original T out of the string of codewords. This is done as follows. Say the first $i - 1$ such blocks have been compressed. To compute the codeword c_j for block j , we first determine its *context*. The context of a block $T[i : l]$ is the longest substring $T[k : i - 1]$, $k < i$, of size at most α such that $T[k : l]$ occurs earlier in T . The codeword c_j is the ordered pair $\langle \gamma, \lambda \rangle$ where γ is the length of the context of block j and λ is rank of block j with respect to the context, according to some predetermined ordering. For instance, one can use the lexicographic ordering of all distinct substrings of size exactly β that follow any previous occurrence of the context of block j in the string. The (α, β) -HYZ compression scheme is based on the intuition that similar symbols in data appear in similar contexts. At the high level, it achieves compression by sorting context-symbol pairs in lexicographic order, and encoding each symbol according to its context and its rank. Thus it is a *context-based* scheme.

The (α, β) -HYZ compression problem has been proposed recently in [Yok96] and [HZ95, HZ98]. The case considered in [Yok96] is one where $\beta = O(1)$ and α is unbounded. During the execution of this algorithm, the average length of a codeword for representing a β -sized block is shown to approach the *conditional entropy* for the block, $H(C)$, within an additive term of $c_1 \log H(C) + c_2$ for