Pattern Matching in Compressed Texts

(Preliminary Version)

S. Rao Kosaraju
Department of Computer Science
The Johns Hopkins University
Baltimore, Maryland 21218-2694

Abstract

We consider the problem of pattern matching when the text is in compressed form. As in Amir, Benson and Farach, we assume that the text is compressed by the Lempel-Ziv-Welch scheme. If the compressed text is of length n and the pattern is of length of m, our basic compression algorithm runs in $O(n + m \sqrt{m} \log m)$ steps, as against Amir, et al's bound of $O(n + m^2)$ steps. We extend the basic algorithm into another that achieves, for any $k \geq 1$, $O(nk + m^{1+\frac{1}{k}} \log m)$ steps.

1 Introduction

The need for storing and processing large amounts of data is well recognized. The recent initiatives on digital libraries are a direct result of the need to handle explosive amounts of data. Thus techniques for compressing texts and processing compressed texts are of paramount importance. In this paper we consider the problem of searching for a pattern in a compressed text. As in [2], we assume that the text is compressed by Lempel-Ziv-Welch scheme and this compressed text is of length n. Note that the real text can be significantly longer than n. We first develop, in section 2, algorithms for several problems that appear to be of independent interest. Finally the pattern matching algorithms are given in section 3. We assume that for any set $A$, $|A|$ denotes the number of elements in $A$; for any tree $A$, $|A|$ denotes the number of nodes in $A$; for any string z, $|z|$ denotes the length of z.

1Supported by NSF Grants CCR9107293 and CCR9508545
2 Preliminary Algorithms

We first review suffix trees for strings, and then develop algorithms for several problems.

2.1 Suffix Trees

For any input $X = x_1x_2...x_n$ and $1 \leq i \leq n$, let $\text{suffix}_i$, or the $i^{th}$ suffix, be $x_i...x_n$ where $\$ is a special symbol not in the alphabet of $X$, and $\text{suffix}_{n+1}$ be $. Let $\Sigma_X$ be the compact trie for suffixes 1, 2, ..., $n + 1$. For $X = aabbaab$, $\Sigma_X$ is shown in Figure 1.

Note that each edge is labeled by a substring of $X$. In the actual implementation, each label is represented by two pointers into $X$. The parent and the grandparent of any node $u$ are denoted by $\text{parent}(u)$ and $\text{gparent}(u)$, respectively. The head of any edge $(u, v)$ is the node $v$. In addition to the nodes of the tree, it is convenient to be able to refer to each position (locus) within a label. The string for a locus, $u$, is the concatenation of the labels on the path from the root to that locus, and is denoted by $\sigma_u$. If $u$ is a node then we say that $\sigma_u$ occurs explicitly; otherwise it occurs implicitly. The locus for $\sigma_u$ is $u$. Thus in Figure 1, $aab$ occurs explicitly and its locus is $v$; $aa$ occurs implicitly and its locus is in the edge ($\text{parent}(v), v$). The node of any locus $u$ is $u$ itself if $u$ is