Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing and Management 41 (2005) 829–841
INFORMATION PROCESSING & MANAGEMENT
www.elsevier.com/locate/infoproman

Pattern matching in Huffman encoded texts ☆

Shmuel T. Klein [a,*], Dana Shapira [b,1]

[a] Department of Math. & CS, Bar Ilan University, Ramat-Gan 52900, Israel
[b] Department of Computer Science, Brandeis University, Waltham, MA 02254, USA

Received 25 April 2003; accepted 25 August 2003
Available online 25 March 2004

Abstract

For a given text which has been encoded by a static Huffman code, the possibility of locating a given pattern directly in the compressed text is investigated. The main problem is one of synchronization, as an occurrence of the encoded pattern in the encoded text does not necessarily correspond to an occurrence of the pattern in the text. A simple algorithm is suggested which reduces the number of erroneously declared matches. The probability of such false matches is analyzed and empirically tested.

Keywords: Data compression; Huffman codes; Pattern matching; Compressed matching

1. Introduction

The general approach for looking for a pattern in a file that is stored in its compressed form, is first decompressing and then applying one of the known pattern matching algorithms to the decoded file. In many cases, however, in particular on the Internet, files are stored in their original form, for if they were compressed, the host computer would have to provide either memory space for each user in order to store the decoded file, or appropriate software to support on the fly decoding and matching. Both requirements are not reasonable, as many users can simultaneously quest the same information reservoir which will either demand a large quantity of free memory, or put a great burden on the host CPU. Another possibility is transferring the compressed files to the personal computer of the user, and then decoding the files. However, we then assume that the user

☆ This is an extended version of a paper that has been presented at the *Data Compression Conference DCC'01*, Snowbird, Utah, 2001 and appeared in its Proceedings, pp. 449–458.

* Corresponding author. Tel.: +972-3-531-8865; fax: +972-3-736-0498.
E-mail addresses: tomi@cs.biu.ac.il (S.T. Klein), shapird@cs.brandeis.edu (D. Shapira).
[1] Tel.: +781-736-2707; fax: +781-736-2741.

0306-4573/$ - see front matter
doi:10.1016/j.ipm.2003.08.008

knows the exact location of the information she or he is looking for; if this is not the case, much unneeded information will be transferred.

There is therefore a need to develop methods for directly searching within a compressed file. This so-called *compressed matching problem* has been introduced by Amir and Benson (1992), and has recently got a lot of attention (Amir, Benson, & Farach, 1996; Gąsieniec & Rytter, 1999; Farach & Thorup, 1995; Kärkkäinen, Navarro, & Ukkonen, 2000; Kida, Takeda, Shinohara, & Arikawa, 1999; DeMoura, Navarro, Ziviani, & Baeza-Yates, 1998; Navarro & Raffinot, 1999; Shibata, Kida, Takeda, Shinohara, & Arikawa, 2000). It is a variant of the classical pattern matching problem, in which one is given a pattern P and a (usually much larger) text T, and one tries to locate the first or all occurrences of P in T. In the compressed version of this problem, the text is supposed to be stored in some compressed form.

For complementary encoding and decoding functions $\mathscr{E}$ and $\mathscr{D}$, that is, functions such that for any text T, one gets $\mathscr{D}(\mathscr{E}(T)) = T$, our aim is to search for $\mathscr{E}(P)$ in $\mathscr{E}(T)$, rather than the usual approach which searches for the pattern P in the decompressed text $\mathscr{D}(\mathscr{E}(T))$. A necessary condition is then that the pattern P should be encoded in the same way throughout the text, which is not the case for arithmetic coding and for dynamic methods such as adaptive Huffman coding. The various Lempel–Ziv variants are also dynamic methods, but for them compressed matching is possible: all of the fragments of the pattern P appear in the compressed text, though not necessarily contiguously and not necessarily in the same order as in the pattern, since parts of the compressed text are pointers to an external dictionary or to previous locations in the given text itself. Much of the previous work on compressed pattern matching concentrates on Lempel–Ziv encodings. A different approach is not to adhere to a known compression scheme, but to devise a new one that is specially adapted to allow efficient searches directly in the compressed file (Manber, 1997; Klein & Shapira, 2000).

Fukamachi, Shinohara, and Takeda (1992) propose a pattern matching algorithm for Huffman encoded strings, based on the Aho–Corasick algorithm. In order to reduce the processing time due to bit per bit state transitions, they use a special code in which the lengths of the codewords are multiples of four bits and present an algorithm for pattern matching in this kind of compressed files. Shibata, Matsumoto, Takeda, Shinohara, and Arikawa (2000) present an efficient realization of pattern matching for Huffman encoded text, substituting t consecutive state transitions of the original machine by a single one. When t is a multiple of 4, this results in a speedup. Takeda et al. (2002) build a pattern matching machine by embedding a DFA that recognizes a set of codewords into an ordinary Aho–Corasick machine, and then make it run over a text byte after byte. Their technique can handle any prefix code including Huffman codes.

DeMoura, Navarro, Ziviani, and Baeza-Yates (2000) propose a compression scheme that uses a word based byte oriented Huffman encoding. The first bit of each byte is used to mark the beginning of a word. Exact and approximate pattern matching can be done on the encoded text without decoding. Their algorithm runs twice as fast as `agrep`, but compression performance is slightly hurt. Moreover, the compression method is not applicable to texts like DNA sequences, which cannot be segmented into words.

In the present work, we are interested in searching within the original Huffman encoded text without any modification. We concentrate on static Huffman coding, for which the problem might at first sight seem trivial. It is, however, not always straightforward, since an instance of $\mathscr{E}(P)$ in the compressed text is not necessarily the encoding of an instance of P in the original text T, and