Journal of Discrete Algorithms ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A space efficient direct access data structure ☆

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

[a] Computer Science Department, Bar Ilan University, Ramat Gan 52900, Israel
[b] Computer Science Department, Ariel University, Ariel 40700, Israel

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Wavelet trees
Direct access
Skeleton Huffman tree
Rank/select data structures

ABSTRACT

Given a file T, we suggest a data structure based on pruning a Huffman shaped Wavelet tree (WT) according to the underlying skeleton Huffman tree that enables direct access to the i-th element of T. This pruned WT is especially designed to support faster random access and save memory storage, at the price of less effective rank and select operations, as compared to the original Huffman shaped WT. The savings are significant only if the underlying alphabet is large enough. We give empirical evidence that when memory storage is of main concern, our suggested data structure generally outperforms other direct access techniques such as those due to Külekci, DACs and sampling, with a slowdown as compared to DACs and fixed length encoding.

1. Introduction

Research in Lossless Data Compression was originally concerned with finding a good balance between the competing efficiency criteria of compressibility of the input, processing time and additional auxiliary storage for the involved data structures. Working directly with compressed data is now a popular research topic, including not only classical text but also various useful data structures, and with a wide range of possible applications. A common operation used in text processing is *Random Access*, which enables direct access to any element of the encoded text. Efficient random access to an encoded file may lead to effective range decoding, in which only a portion of the file bounded by two indices has to be decompressed, and may also improve *parallel decoding*, as different threads will decode disjoint ranges of the file in parallel. If the text is encoded by using some standard fixed length codes (FLC), such as ASCII, random access to the ith codeword is straightforward for any i. However, FLC are wasteful from the storage point of view, and have therefore been replaced in many applications by variable length codes. This may improve the compression performance, but at the price of losing some features. For example, with variable length codes (VLC), random access becomes more involved, because the beginning position of the ith codeword is the sum of the lengths of all the preceding ones. In this paper we are interested in data structures supporting random access in efficient time, while using a compact representation.

A Wavelet tree (WT), suggested by Grossi et al. [12], is a data structure which reorders the bits of the compressed file into an alternative form, thereby enabling direct access, as well as other efficient operations. Wavelet trees can be defined for any prefix code, and the tree structure associated with this code is inherited by the WT. The internal nodes of the WT are annotated with bitmaps. The root of the WT holds the bitmap obtained by concatenating the *first* bit of each of

☆ This is an extended version of a paper that has been presented at the Prague Stringology Conference (PSC'15) in 2015, and appeared in its Proceedings, 67–77, and a paper that has been presented at the Data Compression Conference (DCC'16) in 2016, and appeared in its Proceedings, 63–72.

* Corresponding author.
E-mail addresses: gilad.baruch@gmail.com (G. Baruch), tomi@cs.biu.ac.il (S.T. Klein), shapird@ariel.ac.il (D. Shapira).

http://dx.doi.org/10.1016/j.jda.2016.12.001
1570-8667/

the sequence of codewords in the order they appear in the compressed text. The left and right children of the root hold, respectively, the bitmaps obtained by concatenating, again in the given order, the *second* bit of each of the codewords starting with 0, respectively with 1. This process is repeated similarly on the next levels: the grand-children of the root hold the bitmaps obtained by concatenating the *third* bit of the sequence of codewords starting, respectively, with 00, 01, 10 or 11, if they exist at all, etc.

Various manipulations on the bitmaps of the WT are based on fast implementations of operations known as rank and select. These are defined for any bit vector B and bit $b \in \{0, 1\}$ as:

$\mathsf{rank}_b(B, i)$ – **number** of occurrences of b up to and including position i; and
$\mathsf{select}_b(B, i)$ – **position** of the ith occurrence of b in B.

Efficient implementations for rank and select are due to Jacobson [13], Raman et al. [23], Okanohara and Sadakane [22], Barbay et al. [1] and Navarro and Providel [21], to list only a few. Wavelet trees can be seen as extensions of rank and select operations to a general alphabet.

In this paper we suggest a pruning method based on pruning a Huffman tree shaped WT according to the underlying *skeleton* Huffman tree [14]. This pruned WT is especially designed in order to support faster random access and save memory storage, at the price of less effective rank and select operations, as compared to the original Huffman shaped WTs. The general idea is to apply some pruning strategy on the internal nodes of the WTs, so that the overhead of the additional storage, used by the data structures for processing the stored bitmaps, is reduced. Moreover, the average path lengths corresponding to the codewords is also decreased, and so is also the average time spent for traversing the paths from the root to the desired leaf, which is the basic processing component used to evaluate random access. We present experimental results comparing our method to the state of art, showing that skeleton tree based WTs are superior to Huffman shaped WTs, which suggests that if direct access based operations are done much more often than rank and select, the former trees may be a better choice. In addition, the space savings by our compact data structure outperforms other direct access based solutions used in practice, such as DACs and fixed length coding, while the penalty in processing time is reasonable.

Our paper is organized as follows. Section 2 discusses previous research dealing with random access to files encoded using variable length codes. Section 3 deals with random access to Huffman encoded files, using WTs especially adapted to Huffman compressed files. Section 4 improves the self-indexing data structure by pruning the WT using a skeleton Huffman tree. Section 5 evaluates the savings induced by the proposed data structure. Section 6 further improves the overhead storage by pruning the Wavelet tree even further by means of a reduced skeleton tree. Section 7 then compares our suggested data structures to the state of art, and presents some experiments demonstrating that our data structures are competitive. Finally, Section 8 concludes.

2. Related work

The choice of a code will be guided by the intended application and expected properties. Thus in many situations, FLC are used despite their storage inefficiency, because of the simplicity of processing encoded data. In other cases, although FLC are possible, it is preferable to use VLC for storage efficiency when storage is of main concern. This paper focuses on data structures supporting random access in efficient time, while using a compact representation of the underlying data.

Two famous VLC codes are due to P. Elias [6], who developed universal codes for encoding positive integers, known as Elias-γ and Elias-δ codes. To encode an integer $x \geq 1$ using Elias-γ, its standard binary representation without leading zeros, $B(x)$, is used. The length of $B(x)$ is $\lfloor \log x \rfloor + 1$ bits. $B(x)$ without its leading 1-bit is preceded by the unary encoding of its length (the unary code of the integers $\{1, 2, 3, \ldots\}$ is composed of the codewords {1, 01, 001,...}). Thus, to represent a number x, Elias-γ uses $2\lfloor \log_2(x) \rfloor + 1$ bits. For example, the number 23 is encoded as 00001 0111 (the space, here and below, is inserted for clarity), since $B(23) = 10111$.

The Elias-γ code is used as a building block for the Elias-δ code. To represent an integer x, Elias-δ precedes $B(x)$ without its leading 1-bit by the Elias-γ encoding of its length, for a total of $\lfloor \log_2(x) \rfloor + 2\lfloor \log_2(\lfloor \log_2(x) \rfloor + 1 \rfloor + 1$ bits. For example, the number 23 is encoded as 00101 0111, as the Elias-γ encoding of 5, the number of bits in $B(23)$, is 00101. Elias-δ is asymptotically shorter than Elias-γ, however, for the first numbers, Elias-γ codewords are shorter, so for many distributions, especially when the first probabilities are significantly larger than the last ones, Elias-γ might give better compression performance.

Another VLC which serves as an alternative to Elias codes is based the on Fibonacci numbers as follows. The Fibonacci sequence is defined as

$$F(0) = 1, \quad F(1) = 2, \qquad \text{and} \qquad F(n) = F(n-1) + F(n-2) \quad \text{for} \quad n \geq 2.$$

Any integer B can be represented by a binary string of length r, $c_r c_{r-1} \cdots c_0$, such that $B = \sum_{i=0}^{r} c_i F(i)$. Constructing a unique representation for a given integer x is done by finding the largest Fibonacci number $F(r)$ smaller or equal to x, and then continuing recursively with $B - F(r)$. For example, $31 = 21 + 8 + 2$, so its binary Fibonacci representation would be:

21 13 8 5 3 2 1
1 0 1 0 0 1 0

Please cite this article in press as: G. Baruch et al., A space efficient direct access data structure, J. Discret. Algorithms (2016), http://dx.doi.org/10.1016/j.jda.2016.12.001