Digital Signal Processing 53 (2016) 1–10

Contents lists available at ScienceDirect

Digital Signal Processing

www.elsevier.com/locate/dsp

Chain code compression using string transformation techniques

Borut Žalik, Domen Mongus, Krista Rizman Žalik, Niko Lukač *

University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova 17, SI-2000 Maribor, Slovenia

ARTICLE INFO

Article history:
Available online 17 March 2016

Keywords:
Chain codes
Lossless compression
Burrows–Wheeler transform
Move-to-front transform

ABSTRACT

This paper considers the suitability of string transformation techniques for lossless chain codes' compression. The more popular chain codes are compressed including the Freeman chain code in four and eight directions, the vertex chain code, the three orthogonal chain code, and the normalised directional chain code. A testing environment consisting of the constant 0-symbol Run-Length Encoding (RLE_0^L), Move-To-Front Transformation (MTFT), and Burrows–Wheeler Transform (BWT) is proposed in order to develop a more suitable configuration of these techniques for each type of the considered chain code. Finally, a simple yet efficient entropy coding is proposed consisting of MTFT, followed by the chain code symbols' binarisation and the run-length encoding. PAQ8L compressor is also an option that can be considered in the final compression stage. Comparisons were done between the state-of-the-art including the Universal Chain Code Compression algorithm, Move-To-Front based algorithm, and an algorithm, based on the Markov model. Interesting conclusions were obtained from the experiments: the sequential uses of MTFT, RLE_0^L, and BWT are reasonable only in the cases of shorter chain codes' alphabets as with the vertex chain code and the three orthogonal chain code. For the remaining chain codes, BWT alone provided the best results. The experiments confirm that the proposed approach is comparable against other lossless chain code compression methods, while in total achieving higher compression rates.

1. Introduction

Efficient representation of information has remained a challenge since the earliest days of computing. During those times, Freeman [1] invented a method (known as a chain code) for representing the borders of rasterised geometric shapes. Since then the chain codes have become the popular representation methods within various scientific and engineering disciplines [2–11].

The chain code consists of a small number of instructions, which determine the boundary of a rasterised shape. The original Freeman chain code (in the continuation denoted as F8) contains 8 symbols describing a pixel's neighbourhood with 8-connectivity. This connectivity is coded with 8 codes from the alphabet $\Sigma_{F8} = \{0, 1, 2, 3, 4, 5, 6, 7\}$. Also a pixel's 4-connectivity neighbourhood can be used leading to the Freeman chain code in four directions (F4) having the alphabet $\Sigma_{F4} = \{0, 1, 2, 3\}$. Instead of pixels, the sequence of boundary edges can be described as proposed by Nunes et al. [12]. The alphabet of their Differential Chain Code (DCC) contains only three elements, $\Sigma_{DCC} = \{R, L, S\}$, where R, L, and S stand for right, left, and straight, respectively. Another chain code with only three symbols in the alphabet, $\Sigma_{VCC} = \{1, 2, 3\}$, is the Vertex Chain Code (VCC) proposed by Bribiesca [13]. The elements of VCC represent the number of a shape's boundary pixels meeting within the considered raster vertex. In 2005 Sánchez-Cruz and Rodríguez-Dagnino [14] introduced another three-symbol chain code known as the Three OrThogonal (3OT) chain code with alphabet $\Sigma_{3OT} = \{0, 1, 2\}$. Its codes are determined as follows:

- if the current coding direction is the same as the coding direction of its predecessor, the code is 0;
- if the current coding direction is equal to its first predecessor coding direction, which is different than the direction of its predecessor, the code is 1;
- otherwise the code is 2.

Graphical representations of the chain codes can be found at many places in the literature (e.g. [15–17]). Although chain codes provide compact representations of shapes' boundaries, some redundancies still remain. This is the reason for developing the domain-specific chain code compression methods. In 1997 Nunes et al. proposed a near-lossless method with Huffman codes on their differential chain code DCC [12]. Liu and Žalik [15] derived the Directional Difference Chain Code (DDCC) by encoding the angular differences of F8 by Huffman codes. A few years later, Liu et al. [16] proposed three simple compression methods for VCC:

* Corresponding author. Fax: +386 2 220 7272.
E-mail address: niko.lukac@um.si (N. Lukač).
URL: http://gemma.feri.um.si/ (N. Lukač).

http://dx.doi.org/10.1016/j.dsp.2016.03.002
1051-2004/

E_VCC (Extended VCC), V_VCC (Variable VCC), and C_VCC (Compressible VCC). In order to achieve a more compact code, a modified (M_3OT) chain code was proposed by Sánchez-Cruz et al. [18]. They introduced additional symbols 3, 4, and 5 to encode frequent combinations of 3OT symbols 0 and 1. Instead of static Huffman codes, arithmetic coding was later applied for 3OT and DDCC [19]. However, the arithmetic coding as a part of chain code compression was firstly introduced in 2007 [20]. A context tree for describing the Freeman chain codes' context model followed by arithmetic coding had been proposed in [21] for compressing contour lines. In this case the geometric shapes are not necessarily closed. One of the more efficient methods was proposed by Alcaraz-Corona and Rodríguez-Dagnino [22]. Actually, the method was designed for bi-level image compression. At first, image objects are described by the chain codes. After that the symbol dependences are determined in order to calculate the conditional probability of a Markov model. The selection between various Markov orders is done by Bayesian information criterion – BIC. Finally, the Markov order with the minimal BIC is selected and used by the arithmetic coder. However, the number of probability combinations in the Markov model grows exponentially, which is computationally demanding, and even more importantly, it requires memory space to store the information about the used symbol dependences' probabilities. This is why the authors use a small Markov order (i.e. up to 5). A quasi-lossless chain code compression method was proposed in [23], where the less frequent angular differences (i.e. 135° and 180°) in the DDCC were replaced by the more frequent ones. Žalik and Lukač [17] developed a new lossless chain code compression approach for the more popular chain codes (F8, F4, VCC, 3OT, and NAD – a normalisation of DDC). In order to reduce the information entropy, the Move-To-Front transform was applied followed by the adaptive RLE. Very recently a Universal Chain Code Compression (UCCC) algorithm has been proposed in [24]. The chain codes have been binarised and then compressed, regardless of the chain code type, by a combination of three modes: RLE, LZ77 and COPY. The equivalence between the chain codes was formally proven in [25]. The chain codes can also be used in 3D [26,27].

This paper introduces a new lossless chain code compression approach, which combines different string transformation techniques. A testing environment was set-up consisting of Move-To-Front Transform (MTFT), constant 0-symbol Run-Length Encoding (RLE_0^L) (where runs of L 0-symbols are replaced by a new symbol L), and Burrows–Wheeler Transform (BWT). In the next section, these techniques are briefly explained. In Section 3 the structure of the testing environment is given together with the entropy coder. Section 4 contains the conducted experiments using 24 benchmark shapes, where the comparisons were performed using various lossless chain code algorithms. The last section concludes the paper.

2. String transformation methods

This section briefly introduces the string transformation techniques used in this paper: Move-To-Front Transform (MTFT), Burrows–Wheeler Transform (BWT), and two variations of the Run-Length Encoding (RLE).

2.1. Move-to-front transform

MTFT [28,29] is a technique originally developed for memory paging and more efficient access to the elements of a list [30,31]. The elements, which have been accessed recently are located nearer to the top of the list. In data compression, MTFT is frequently used during the pre-processing stage, as it may reduce the information entropy [17]. Let $\Sigma_S = \{\sigma_0, \sigma_1, \ldots, \sigma_{k-1}\}$ be the alphabet consisting of k symbols, and $S = \{\sigma_i\}, \sigma_i \in \Sigma_S, i \in [0, n-1]$ the sequence of $n = |S|$ input symbols. The list T contains an arrangement of all $\sigma_i \in \Sigma_S$.

```
begin
    Initialization;                 \\fill T with {σ_i} ∈ Σ
    for i = 0 to n-1 do
    begin
        index = 0;
        while T[index] ≠ σ_i do
            index = index + 1;
        O[i] =index;                \\send index to output
        for k = index to 1 do       \\move all element upward
            T[k] = T[k-1];
        T[0] = σ_i;                 \\current symbol is moved-to-front
    endfor;
end;
```

Pseudo-code 1. Move-To-Front transform.

MTFT incrementally takes symbol σ_i from S, finds its position (*index*) within T, stores the *index* in the output array O, moves σ_i to the first position in T and shift all others elements from T up to *index* for one position (see Pseudo-code 1 for details).

Let us consider an example. $\Sigma_S = \{a, b, c\}$, $S = [c, a, a, a, a, a, a, b, a, b, a, b, a]$ and $T = [c, b, a]$. Running MTFT results in $O = [0, 2, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1]$. As can be seen, the sequence of the same symbol results in runs of 0's, whilst the sequence of alternating symbols produces runs of 1's.

In the next example, a less favourable case is shown: $S = [c, a, b, a, a, b, a, c, a, b, c, b, a]$ gives $O = [0, 2, 2, 1, 0, 1, 1, 2, 1, 2, 2, 1, 2]$, where runs of the same symbols do not exist. Such streams can be rearranged in, hopefully, a more suitable form for compression by the Burrows–Wheeler Transform, which is described next.

2.2. Burrows–Wheeler transform

Burrows–Wheeler Transform (BWT) is one of the more fascinating algorithms in computer science, being discovered in 1978 and published in 1994 [32]. A detailed explanation with various applications can be found in [33,34]. BWT transforms the input string S to the output string O such that symbols with a similar context are grouped closely together. It works over three steps:

- generate n rows by left-shifting S n times;
- lexicographically sort the rows;
- read the characters from the last column, which represent BWT.

The position of the initial string S should be stored in order to reconstruct the input string S from $BWT(S)$. A short example follows, where $S = [cabcabcd]$:

stage 1: shifting	stage 2: sorting	stage 3: obtaining BWT
cabcabcd	abcabcdc	c
abcabcdc	abcdcabc	c
bcabcdca	bcabcdca	a
cabcdcab	bcdcabca	a
abcdcabc	cabcabcd ← 4	d
bcdcabca	cabcdcab	b
cdcabcab	cdcabcab	b
dcabcabc	dcabcabc	c

BWT(S) = [ccaadbbc] and the *BWT-index* for the reconstruction is 4. As can be seen, after BWT the same character tends to form runs and therefore, BWT(S) is more compressible. The procedure for the inverse BWT can be found in the literature (e.g. [33]). The main problem of BWT is its time complexity. A naive imple-