ISSN 1746-7659, England, UK
Journal of Information and Computing Science
Vol. 5, No. 3, 2010, *pp*. 193-198

An Improved Bit-level Arithmetic Coding Algorithm

Jianjun Zhang [1, +] and Xingfang Ni [2]

[1]Dept. of Math, Shanghai University, Shanghai 200444, P.R. China
[2]Shanghai Jiao Tong University, Shanghai 200030, P.R. China

(*Received March 16, 2009, accepted October 22, 2009*)

Abstract: Arithmetic coding is the most powerful lossless data compression technique that has attracted much attention in recent years. This paper presents a new implementation of bit-level arithmetic coding using integer additions and shifts. The algorithm has less computational complexity and more flexibility, and thus is very suitable for hardware design. We show that degradation of the proposed algorithm is bounded by 0.2075.

Keywords: Arithmetic coding, multiplication-free.

1. Introduction

Arithmetic coding is the most powerful lossless data compression technique that has attracted much attention in recent years [2-10]. It provides more flexibility and better efficiency than the celebrated Huffman coding. Arithmetic coding completely bypasses the traditional coding paradigm: replace an input symbol with a specific code. Instead, it represents a stream of input symbols by a binary number in the interval [0,1]. This approach totally relaxes the constraint upon Huffman coding: each symbol has to be encoded by an integral number of bits and by at least one bit. Thus its coding results are closer to Shannon's entropy bound [1].

The basic arithmetic coding requires infinite precision operations that are difficult to implement on a fixed precision computer. Therefore, it takes about twenty years to implement a practical arithmetic coding program on a fixed precision computer due to Rissanen and Pasco [5][6]. They proposed a LIFO-form (last-in-first-out) and a FIFO-form (first-in-first-out) of arithmetic coding respectively. These illustrate that arithmetic coding can be implemented on a fixed length computer. Combining the advantage of the above two schemes, Rubin [7] proposed a general implementation of arithmetic coding by using of fixed precision registers. A good tutorial can be found in Witten, Neal and Cleary [8], in which an interesting carry-over technique to avoid bit propagation was introduced.

One drawback of arithmetic coding is its slow speed, because arithmetic coding requires multiplications. Some research have been done to avoid multiplications and to improve the efficiency. Langdon and Rissanen[4] proposed a modified scheme for encoding a binary string by using of shift-and-add. Rissanen and Mohiuddin[6] proposed a multiplication-free algorithm for encoding general string. The method was further developed by Lei[5]. Howard and Vitter[2] described an efficient bit-level implementation that uses table lookup as a fast alterative to arithmetic operations.

To further improve the implementation efficiency of arithmetic coding, we present a new bit-level arithmetic coding by using integer additions and shifts. The algorithm has less computational complexity and more flexibility, and thus is very suitable for hardware and software design.

The paper is organized as follows. In section 2, we briefly review the bit-level arithmetic coding. In section 3, we propose an improved multiplication-free arithmetic coding algorithm which has less computational complexity. In section 4, we analyze the efficiency of the algorithm and show that the degradation of the proposed algorithm is bounded by 0.2075. Finally, computer simulation is given.

2. Review of bit-level arithmetic coding

Let $p(0|s)$ be the probability of 0 according to a given model, where s denotes the previous string. If the encoding interval for string s is $[C(s), C(s)+A(s))$, then the bit-level arithmetic coding algorithm

updates $A(s)$ and $C(s)$ according to the following rule:

$$\begin{aligned} C(s0) &= C(s) & A(s0) &= A(s)\,p(0\,|\,s) \\ C(s1) &= C(s) + A(s0) & A(s1) &= A(s) - A(s0) \end{aligned} \tag{2.1}$$

In practice, $C(s)$ and $A(s)$ are represented by finite bits, and probability $p(0\,|\,s)$ is estimated by the zero occurrence frequency. That is $p(0\,|\,s) \approx n(0\,|\,s)/n(s)$, $p(1\,|\,s) \approx n(1\,|\,s)/n(s)$ where $n(0\,|\,s)$ and $n(1\,|\,s)$ represent the count number of 0 and 1 in string s according to a given model, and $n(s) = n(0\,|\,s) + n(1\,|\,s)$. It is obvious that, arithmetic coding requires multiplications even if we only use integer number. This is expensive and usually has slow speed in both hardware and software implementation. Therefore, in practical, we should avoid to use formula (2.1).

Two typical multiplication-free bit-level arithmetic coding are LR algorithm (London & Rissanen) [4] and RM algorithm (Rissssanen & Mohiouddin) [6].

LR algorithm uses $2^{-k(s)}$ as an approximation to probability $p(0\,|\,s)$ to eliminate the multiplications in (2.1), where $k(s)$ is some integer. As the encoding proceeds, $A(s)$ becomes smaller and smaller. In order to keep as many significant bits as possible, it multiplies $2^{L(s)}$ to $A(s)$ such that $2^{L(s)}A(s)$ lies in [1, 2), where $L(s)$ is some nonnegative number adjusted according to magnitude of $A(s)$. LR algorithm then updates $A(s)$ and $C(s)$ as follows:

$$\begin{aligned} C(s0) &= C(s) & A(s0) &= 2^{-L(s)-k(s)} \\ C(s1) &= C(s) + A(s0) & A(s1) &= A(s) - A(s0) \end{aligned} \tag{2.2}$$

RM algorithm is a simplified form of multiplication-free multi-alphabet arithmetic coding. The mechanism to avoid multiplications is choosing $L(s)$ and $k(s)$ such that $2^{L(s)}A(s)$ and $2^{-k(s)}n(s)$ lie inside [0.75, 1.5) simultaneously. RM algorithm then updates $A(s)$ and $C(s)$ as follows:

$$\begin{aligned} C(s0) &= C(s) & A(s0) &= 2^{-L(s)-k(s)}\,n(0\,|\,s) \\ C(s1) &= C(s) + A(s0) & A(s1) &= A(s) - A(s0) \end{aligned} \tag{2.3}$$

It is obvious that, the above algorithm can be implemented by shifts and additions. However, the operation of choosing $2^{L(s)}$ such that $2^{L(s)}A(s)$ lies in [1, 2], and choosing $L(s)$ and $k(s)$ such that $2^{L(s)}A(s)$ and $2^{-k(s)}n(s)$ lie inside [0.75, 1.5] simultaneously is somewhat time-consuming. On the other hand, in practical implementation, the inherent carry-over problem need to be considered. A wide used technique to avoid carry-over is bit-stuffing proposed by Rissanen [6], but this technique reduces the encoding efficiency in some extent. In [8], an algorithm to avoid bit-stuffing is presented, but the technique in the algorithm is very complicated. In [9], a very simple method to avoid carry-over was presented.

3. An improved bit-level arithmetic coding

Zhao et al in [9] used the similar approximation method as LR and RM algorithm. They directly choose integer number $k(s)$ such that $2^{-k(s)}$ approximates $p(0\,|\,s)$. To deal with the carry-over problem, they presented a very simple and efficient technique.

In the subsequent discussion, we use a similar method to deal with the carry-over problem. To avoid integer multiplication, we take a method that is easy to manipulate. The method combines the encoding and the carry-over technique together. The method has less computational complexity and accordingly runs rapidly.

Improved arithmetic coding algorithm:

1. Initialize C and A;
2. Compute $p0$ and $p1$ according to a given statistical model;
3. Input the next symbol x. If end of file, then output the content of C and terminate;