An efficient chaotic source coding scheme with variable-length blocks*

Lin Qiu-Zhen(林秋镇)[a)†], Wong Kwok-Wo(黄国和)[a)], and Chen Jian-Yong(陈剑勇)[b)]

[a)] *Department of Electronic Engineering, City University of Hong Kong, Hong Kong* 999077, *China*

[b)] *Department of Computer Science and Technology, Shenzhen University, Shenzhen* 518060, *China*

(Received 24 November 2010; revised manuscript received 27 January 2011)

An efficient chaotic source coding scheme operating on variable-length blocks is proposed. With the source message represented by a trajectory in the state space of a chaotic system, data compression is achieved when the dynamical system is adapted to the probability distribution of the source symbols. For infinite-precision computation, the theoretical compression performance of this chaotic coding approach attains that of optimal entropy coding. In finite-precision implementation, it can be realized by encoding variable-length blocks using a piecewise linear chaotic map within the precision of register length. In the decoding process, the bit shift in the register can track the synchronization of the initial value and the corresponding block. Therefore, all the variable-length blocks are decoded correctly. Simulation results show that the proposed scheme performs well with high efficiency and minor compression loss when compared with traditional entropy coding.

Keywords: chaos, compression, source coding, finite-precision implementation

PACS: 05.45.–a, 05.45.Gg, 05.45.Vx **DOI:** 10.1088/1674-1056/20/7/070501

1. Introduction

In recent years, the application of chaos theory in communications has received much attention. Since the first report of using chaotic signals to carry information published in 1993,[1] chaos theory has shown its potential in many fields of communications, such as cryptography and channel coding.[2−6] The properties of chaotic systems, such as ergodicity, sensitive dependence on initial conditions and good pseudo-randomness, are favourable for secure and multiple access systems. However, the use of chaotic systems for source coding or data compression has received little attention.

The idea of using chaotic systems for data compression was first proposed in Ref. [7], in which the Bernoulli shift map was adapted to the probability distribution of the source messages. It can be considered as a method for finding the best initial condition for iterating a chaotic map in order to generate a symbolic sequence representing the source message. The theoretical compression performance of this chaotic coding scheme can reach that of the optimal entropy compression. In Ref. [8], source coding and chaotic systems were related to each other by treating messages emitted by independent and identically distributed sources as symbol sequences of a chaotic nonlinear dynamical system known as the generalized Luroth series (GLS). It is proven that the GLS achieves Shannon's entropy bound and is a generalization of the arithmetic coding.[8] However, the conclusions in Refs. [7] and [8] are obtained under the assumption of infinite-precision computation. When finite-precision computation is used in practical cases, the encoding interval enlargement approach in the arithmetic coding cannot be adopted directly in the chaotic coding. The reason is that the arithmetic coding employs the first-in-first-out (FIFO) mechanism. The symbols are decoded in the same order as that they are encoded in and so the encoding interval enlargement can be traced in the decoder.[9] On the contrary, the chaotic coding adopts the reverse interval mapping mechanism in encoding. The last symbol encoded is the first symbol decoded. Therefore, the encoding interval enlargement will affect the previous encoded symbols.

Another efficient approach to the arithmetic coding with finite-precision computation is the connected-index finite-length arithmetic coding. It splits the

*Project supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (Grant No. CityU 123009).

†Corresponding author. E-mail: qiuzhlin@student.cityu.edu.hk

© 2011 Chinese Physical Society and IOP Publishing Ltd

information about the last non-fitting symbol into the current and the subsequent codewords.[10] This approach can be further improved by integrating a probability-fitting algorithm.[11] For the practical application of chaotic coding scheme, Hardy and Sabatta[12] implemented three different techniques, namely, reverse interval mapping, variable-bit-length encoding and perturbation encoding, using a certain type of one-dimensional chaotic map. Although the chaotic coding can be implemented using double-precision data type, the compression performance is still unsatisfactory. In Ref. [13], we proposed a simultaneous arithmetic coding and encryption scheme using chaotic maps, in which the message sequence was divided into fixed-length blocks and a forbidden symbol was used to separate the ciphertext of each block. The computation precision problem of the chaotic coding is tackled, but it costs much computational time in high-precision calculation. Besides, the compression performance is lowered by the requirement for inserting forbidden symbols.

Inspired by Refs. [12] and [13], we propose an efficient chaotic source coding scheme with variable-length blocks to fit the finite-precision computation. Making use of a piecewise linear chaotic map, the proposed scheme generates compressed bits to a message block once the number of output bits exceeds a threshold. The number of compressed bits is smaller than the registered length, and an initial value is obtained for each block. In the decoding process, the bit shift operation in the register can track the synchronization of the initial value and the corresponding block. This means that the bit shift will transform the current initial value to the next one for decoding the next block. Therefore, all the variable-length blocks are guaranteed to be decoded correctly. In order to evaluate the actual compression performance, the proposed scheme is implemented with C++ language to compress 18 standard test files.[14] Simulations show that our scheme is slightly faster than the fast 32-bit precision arithmetic codes[15] with insignificant loss in compression performance.

The rest of this paper is organized as follows. In Section 2, the concept of chaotic source coding using a piecewise linear chaotic map is illustrated. The details of the proposed scheme are described in Section 3. Its performance is reported in Section 4, whereas conclusions are drawn in Section 5.

2. Chaotic source coding scheme

In this section, the concept of encoding a binary message using a piecewise linear chaotic map is illustrated. The map is defined as

$$f(x) = \begin{cases} x/p, & 0 \le x < p, \\ (x-p)/(1-p), & p \le x < 1, \end{cases} \tag{1}$$

where p is the point of partition. Suppose that there is a binary message sequence $M = 1001000101$. The probabilities of occurrence of symbols 0 and 1 are 0.6 and 0.4 respectively. By setting the parameter p in Eq. (1) as 0.6, the interval [0, 1) is split into two partitions: [0, 0.6) represents symbol 0, while [0.6, 1) corresponds to symbol 1, as plotted in Fig. 1.

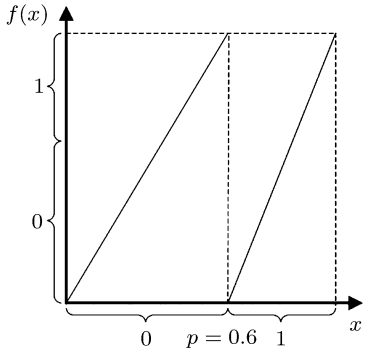


Fig. 1. Piecewise linear chaotic map with parameter $p = 0.6$ for encoding binary message.

The compression is performed with the reverse interval mapping approach. The reverse function of the above piecewise linear chaotic map is given by

$$f^{-1}(I) = \begin{cases} 0.6 \times I, & \text{symbol} = \text{“0”}, \\ 0.6 + 0.4 \times I, & \text{symbol} = \text{“1”}, \end{cases} \tag{2}$$

where I is the current encoding interval, which is initially set as [0, 1). Starting from the last symbol “1”, the mapping is from the interval [0, 1) to [0.6, 1) according to Eq. (2). The second last symbol is “0” and so the interval [0.6, 1) is mapped to [0.36, 0.6). The third last symbol is “1”, then the mapping is from the interval [0.36, 0.6) to [0.744, 0.84). The remaining symbols are encoded in the same way until the first symbol is encountered. Then the final encoding interval is obtained, in which the initial value for iterating the piecewise linear chaotic map lies. Any real value within the final encoding interval can be chosen and is represented in binary format as the compressed sequence. In the decoding process, the compressed binary sequence is first converted into a real value, which is then used as the initial value for Eq. (1) with

$p = 0.6$. In each iteration of the chaotic map, the partition that the output falls in determines the reconstructed symbol. The original binary sequence M can then be recovered by collecting the reconstructed symbols as shown in Fig. 2, where the x coordinate is the counter for the decoded symbols and the y coordinate is the iteration value using Eq. (1) with $p = 0.6$. In Fig. 2, the corresponding lower and upper bounds are obtained in the encoding procedure by using the reverse interval mapping. It can be observed that the iteration value will lie between the lower and the upper bounds in each iteration if the initial point is chosen within the final encoding interval. This method is proven to be Shannon optimal under the infinite-precision computation.[7,8]

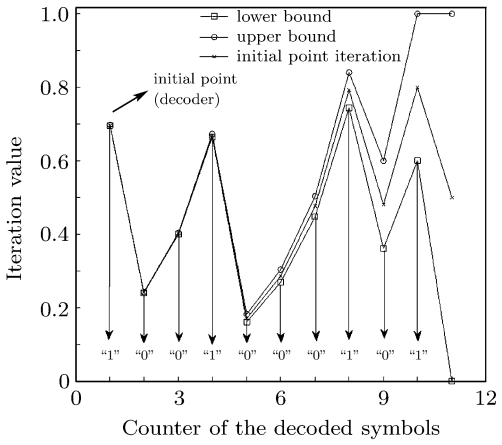


Fig. 2. Chaotic map iteration for decoding the corresponding message M.

3. Chaotic source coding with finite precision computation

In this section, we introduce an interesting chaotic source coding scheme for compressing a source file with multiple symbols using a finite-precision computation. Assume that the source file composed of m symbols is represented by $s_1, s_2, ..., s_m$ and has n distinct symbols $a_1, a_2, ..., a_n$ with probabilities of occurrence $p_1, p_2, ..., p_n$, respectively, where $\sum_{i=1}^{n} p_i = 1$. Based on the probability distribution of the source file, a piecewise linear chaotic map is constructed as follows:

$$f(x) = \begin{cases} x/p_1, & x \in I_1, \\ (x - p_1)/p_2, & x \in I_2, \\ \quad \vdots & \\ \left(x - \sum_{i=1}^{n-1} p_i\right)/p_n, & x \in I_n, \end{cases}$$

where

$$\begin{cases} I_1 = [0, p_1), \\ I_i = \left[\sum_{j=1}^{i-1} p_j, \sum_{j=1}^{i} p_j\right), \quad i = 2, 3, ..., n. \end{cases} \tag{3}$$

The intervals I_i $(i = 1, 2, ..., n)$ map to the distinct symbols a_i $(i = 1, 2, ..., n)$, as shown in Fig. 3.

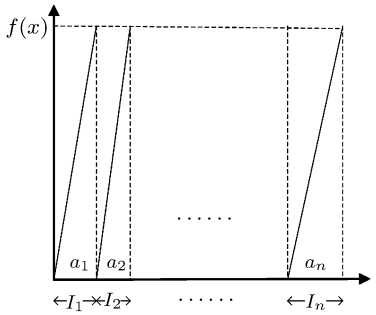


Fig. 3. Piecewise linear chaotic map for multiple symbols.

In the reverse interval mapping, the following mapping function is used:

$$f^{-1}(I) = \begin{cases} p_1 \times I, & \text{symbol} = \text{“}a_1\text{”}, \\ \sum_{i=1}^{1} p_i + p_2 \times I, & \text{symbol} = \text{“}a_2\text{”}, \\ \quad \vdots & \\ \sum_{i=1}^{n-1} p_i + p_n \times I, & \text{symbol} = \text{“}a_n\text{”}, \end{cases} \tag{4}$$

where I is the encoding interval and is initially set as [0, 1).

A block diagram of the proposed scheme is shown in Fig. 4, where the source file with m symbols is divided into K blocks with different size k_i $(i = 1, 2, \ldots, K$ and $\sum_{i=1}^{K} k_i = m)$. In the encoding process, our scheme encodes the variable-length blocks with the piecewise linear chaotic map based on the entropy information and gives an initial value v_i for the i-th block. In decoding, the bit shift in the register can track the synchronization between each initial value v_i and the corresponding i-th block. This means that the bit shift will transform the current initial value v_i to a new one v_{i+1} for decoding the next block. Before the details of the proposed scheme are described in the following sections, it is necessary to define the following parameters. The register length is L; the maximal entropy among all the distinct symbols is $E_{\max}$: $E_{\max} = \max\{-\log_2(p_i), i = 1, 2, ..., n\}$; the number of bit shift $B(k, s_t)$ for encoding or decoding k consecutive symbols from s_t $(t = 1, 2, \ldots, m)$ is

given by

$$B(k, s_t) = \sum_{j=0}^{k-1} -\log_2 p(s_{t+j}) + \delta, \tag{5}$$

where δ is a pre-defined small positive value to prevent the calculation from overflowing and $B(0, s_t)$ is initially set to be δ; The threshold of output bits is T: $L/2 \le T < L - 4 - E_{\max}$ with the requirement $E_{\max} < L/2 - 4$, where the constraint is to prevent the calculation from overflowing and guarantee the correct decoding as described in Section 3.3.

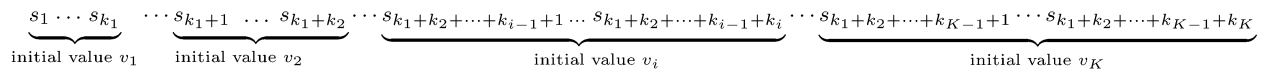


Fig. 4. A block diagram of the proposed scheme.

3.1. Encoding procedures

Step 1: Scan the source file to find out the probability of occurrence for each distinct symbol. Write the file length and the statistic model into a compressed file as the header. Based on Eq. (4), construct the corresponding reverse interval mapping function using the statistical model. Initialize the encoding interval I as [0, 1). Set both the block number i and the symbol pointer t to be 1.

Step 2: Read k_i consecutive symbols from s_t until all the symbols in the source file have been encoded or the following conditions are satisfied:

$$B(k_i - 1, s_t) < T, \quad B(k_i, s_t) \ge T. \tag{6}$$

Step 3: Encode $s_t, s_{t+1}, \ldots, s_{t+k_i-1}$ with the constructed reverse interval mapping function and find the final encoding interval $[I_{\text{low}}, I_{\text{up}})$. Set $I_{\text{low,pre}} = I_{\text{low}}$, $I_{\text{up,pre}} = I_{\text{up}}$, $r = \lceil B(k_i, s_t) \rceil$, $t = t + k_i$ and $i = i + 1$, where the function $\lceil x \rceil$ returns the smallest integer greater than or equal to x.

Step 4: If all the symbols in the source file have been encoded, go to Step 7. Otherwise, go to Step 5.

Step 5: Reset the encoding interval as [0, 1) and read k_i consecutive symbols from s_t until all the symbols of source file have been encoded or the conditions stated in expression (6) are satisfied.

Step 6: Encode $s_t, s_{t+1}, \ldots, s_{t+k_i-1}$ with the constructed reverse interval mapping function and find the final encoding interval $[I_{\text{low}}, I_{\text{up}})$. From Fig. 4, the initial value v_{i-1} can be represented as a binary value $0.b_1b_2\cdots b_rb_{r+1}\cdots b_L$ as the register length is L, in which the binary sequences $b_1b_2\cdots b_r$ and $b_{r+1}\cdots b_L$ are obtained according to the previous final encoding interval $[I_{\text{low,pre}}, I_{\text{up,pre}}]$ and the current final encoding interval $[I_{\text{low}}, I_{\text{up}}]$, respectively. In the proposed scheme, the binary sequence $b_{r+1}\cdots b_L$ is estimated based on the first $L-r$ bits of I_{low}, and the binary sequence $b_1b_2\cdots b_r$ can be found based on the condition that $0.b_1b_2\cdots b_rb_{r+1}\cdots b_L$ lies in the reduced previous final encoding interval $[I_{\text{low,pre}}, I_{\text{up,pre}} - 2^{-L})$. As the actual initial value v_{i-1} is not 2^{-L} larger than our estimated value $0.b_1b_2\cdots b_rb_{r+1}\cdots b_L$, the reduced previous final encoding interval ensures that the actual initial value v_{i-1} locates exactly within $[I_{\text{low,pre}}, I_{\text{up,pre}})$. Then write binary bits $b_1b_2\cdots b_r$ into the compressed file and the previous consecutive k_{i-1} symbols are compressed. Reset $I_{\text{low,pre}} = I_{\text{low}}$, $I_{\text{up,pre}} = I_{\text{up}}$, $r = \lceil B(k_i, s_t) \rceil$, $t = t + k_i$ and $i = i + 1$. Go to Step 4.

Step 7: Find a binary sequence $b_1b_2\cdots b_r$, where $0.b_1b_2\cdots b_r$ lies in $[I_{\text{low,pre}}, I_{\text{up,pre}})$. Write the binary bits $b_1b_2\cdots b_r$ into the compressed file and the source file has been compressed completely.

3.2. Decoding procedures

Step 1: Read the header of the compressed file to obtain the distinct symbols and their corresponding probabilities. Based on Eq. (3), construct the piecewise linear chaotic map according to the statistics information. Initialize both the block number i and the symbol pointer t as 1.

Step 2: Read L bits from the compressed file and put them into the register as represented by $b_1b_2\cdots b_L$.

Step 3: Convert the binary sequence $b_1b_2\cdots b_L$ into a real value $0.b_1b_2\cdots b_L$ and iterate the constructed piecewise linear chaotic map from this value. Then decode k_i consecutive symbols until the number of the decoded symbols is equal to the length of the source file or the conditions in expression (6) are fulfilled.

Step 4: If the number of the decoded symbols is equal to the length of the source file, the decoding procedure is completed. Otherwise, left shift the register for $\lceil B(k_i, s_t) \rceil$ bits and load $\lceil B(k_i, s_t) \rceil$ more bits from the compressed file to the register. Set $i = i + 1$, $t = t + k_i$ and go to Step 3.

3.3. Correctness of decoding

In order to guarantee the correctness of the decoding, the bit shift operations in the decoding process should ensure that each initial value $0.b_1b_2\cdots b_L$ obtained after the bit shift has lain in the final encoding interval $[I_{\rm low},\ I_{\rm up}]$. The following remarks is conducible to the analysis.

Remark 1: As $B(k,s_t)$ is a strictly increasing function of k, the encoding and the decoding of k consecutive symbols stop when the conditions stated in expression (6) are satisfied.

As $B(k+1,s_t)-B(k,s_t)=-\log_2 p(s_{t+k})>0$, it is obvious that $B(k,s_t)$ is a strictly increasing function of k. When T is pre-defined as not smaller than δ, it is greater than or equal to $B(0,s_t),\forall t\in[1,m]$. Therefore, with the increase of k, there exists a certain value of k that satisfies the inequalities in expression (6). It should be noticed that in the previous sections, different values of k in the i-th block are identified with k_i.

Remark 2: With the binary sequence $b_{r+1}\cdots b_L$ estimated based on the first $L-r$ bits of $I_{\rm low}$, it is possible to find a binary sequence $b_1b_2\cdots b_r$, where $0.b_1b_2\cdots b_rb_{r+1}\cdots b_L$ lies in the previous reduced final coding interval $[I_{\rm low,pre},\ I_{\rm up,pre}-2^{-L})$ in Step 6 of the encoding process.

Remark 2 is valid only if

$$-\log_2(I_{\rm up,pre}-I_{\rm low,pre}-2^{-L})\le r.$$

As

$$\begin{aligned} r &= \lceil B(k,s_t)\rceil \\ &= \lceil -\log_2(I_{\rm up,pre}-I_{\rm low,pre})+\delta\rceil \\ &\le \lceil T+E_{\max}\rceil \\ &\le L-4, \end{aligned}$$

we have

$$\begin{aligned} -\log_2(I_{\rm up,pre}-I_{\rm low,pre}) &< L-4, \\ (I_{\rm up,pre}-I_{\rm low,pre}) &> 2^{-(L-4)}. \end{aligned}$$

Therefore, we obtain $-\log_2(I_{\rm up,pre}-I_{\rm low,pre}-2^{-L})<-\log_2((I_{\rm up,pre}-I_{\rm low,pre})\times(1-2^{-4}))\approx-\log_2(I_{\rm up,pre}-I_{\rm low,pre})+0.0931$. When δ is larger than 0.0931, e.g., $\delta=0.1$, the inequality $-\log_2(I_{\rm up,pre}-I_{\rm low,pre}-2^{-L})\le r$ is satisfied.

Remark 3: The i-th bit shift will transfer the current initial value v_i to the next one v_{i+1}.

Remark 1 implies that the number of bit shift in the decoding process is the same as that of compressed bits for the k_i consecutive symbols, if the initial value v_i lies in the i-th final encoding interval. According to Remark 2, each initial value v_i converted from L bits falls exactly in the i-th final encoding interval. Therefore, the current k_i consecutive symbols will be decoded correctly and the bit shift will synchronize the initial value for each block in both encoding and decoding. This proves that the i-th bit shift will transfer the current initial value v_i to the next one v_{i+1}.

3.4. Compression performance

As the proposed scheme outputs the compressed bits and resets the initial encoding interval as [0, 1) once the number of bit shift exceeds the threshold T, it will induce compression loss when the final number of bit shift is not an integer. Besides, the pre-defined parameter δ also affects the compression performance. In our scheme, T compressed bits will introduce $(1+\delta)$ redundant bits in the worst case. For the source file composed of m symbols, each represented by 8 bits, the best compression ratio ($B_{\rm r}$) is calculated using the source entropy, and is given by

$$B_{\rm r}=\frac{8}{\sum_{i=1}^{n}p_i\log p_i}. \tag{7}$$

The maximal total number of redundant bits in the compressed file is $m(1+\delta)/(B_{\rm r}T)$. Therefore, compared with $B_{\rm r}$, the upper bound of the loss in the compression ratio (C) is

$$C=B_{\rm r}\times\frac{1+\delta}{T+1+\delta}. \tag{8}$$

Therefore, it is suggested to choose a large value for T so as to reduce the compression loss.

4. Simulations

The proposed scheme is implemented with C++ to compress 18 standard test files from the Calgary Corpus.[14] The compression performance can be found in Table 1, where the compression ratio ($C_{\rm r}$), the compression time ($T_{\rm c}$) and the decompression time ($T_{\rm d}$) are listed. Note that the compression ratio is calculated as

$$C_{\rm r}=\frac{\text{length of original file}}{\text{length of compressed file}}, \tag{9}$$

where the length of the compressed file includes that of the header information. For comparison purpose, the best compression ratio ($B_{\rm r}$) and the compression performance using the fast 32-bit precision arithmetic coding[15] are also included. The values of L, T and δ are set as 48, 32 and 0.1, respectively. All the simulations are performed in a personal computer with an i5 2.8 GHz CPU and 3GB memory.

Table 1. Comparison between the proposed scheme and fast 32-bit precision arithmetic coding.

File	Size(byte)	B_r	Our scheme			32-bit fast AC		
			$C_{\rm r}$	$T_{\rm c}$ /ms	$T_{\rm d}$/ms	$C_{\rm r,AC}$	$T_{\rm c,AC}$ /ms	$T_{\rm d,AC}$ /ms
Paper5	11,954	1.62	1.53	0.766	0.922	1.56	0.797	1.016
Paper4	13,286	1.70	1.62	0.797	0.969	1.65	0.797	1.047
Obj1	21,504	1.34	1.26	0.953	1.172	1.28	1.032	1.359
Paper6	38,105	1.60	1.55	1.141	1.578	1.58	1.281	1.828
Progc	39,611	1.54	1.50	1.203	1.625	1.52	1.328	1.89
Paper3	46,526	1.71	1.67	1.234	1.782	1.70	1.407	2.016
Progp	49,379	1.64	1.60	1.328	1.843	1.63	1.515	2.172
Paper1	53,161	1.61	1.56	1.375	1.953	1.59	1.579	2.329
Progl	71,646	1.68	1.64	1.563	2.437	1.67	1.875	2.718
Paper2	82,199	1.74	1.70	1.609	2.703	1.73	2.016	3.125
Trans	93,695	1.45	1.41	1.781	2.859	1.44	2.297	3.688
Geo	102,400	1.42	1.38	1.937	3.109	1.40	2.453	3.75
Bib	111,261	1.54	1.51	1.953	3.25	1.53	2.578	4.047
Obj2	246,814	1.28	1.25	3.75	6.843	1.27	5.218	8.735
News	377,109	1.54	1.51	5.016	11.531	1.54	7.11	12.297
Pic	513,216	6.61	6.45	4.407	11.453	6.50	7.109	9.828
Book2	610,856	1.67	1.64	10.36	19.125	1.67	10.719	18.813
Book1	768,771	1.77	1.74	11.828	22.359	1.77	13.046	22.813

The results in Table 1 show that the compression ratio of our scheme is slightly (0.03–0.16) lower than the $B_{\rm r}$. This is because the header information is considered and the encoding interval is re-initialized each time when the compressed bits are generated. In general, the gap between $C_{\rm r}$ and $B_{\rm r}$ decreases with the increase of the document size, as the header information occupies only a smaller proportion of the compressed file. The compression performance of the fast 32-bit arithmetic coding with a static model is 0.02–0.05 better than ours.

Besides the compression performance, the execution time of our scheme is compared with that of the traditional arithmetic coding. The timing data show that our scheme is even faster than the fast 32-bit arithmetic coding in both compression and decompression except for the pic and the book2 files. This is because the fast 32-bit arithmetic coding needs to enlarge the encoding interval each time the register emits the compressed bits, but ours only resets the encoding interval after a large number of bits have been emitted. Besides, in practical implementation, the smallest real number greater than the lower bound is chosen within the precision of bit shift as the compressed bits. Therefore, it is only necessary to calculate the lower bound in the reverse mapping. In the decoding process, the map iterates from the initial value to reconstruct the source symbols. As a result, the proposed scheme can be implemented efficiently. The compression speed of the proposed scheme lies between 14.88 MB/s and 111.06 MB/s, while the decompression speed falls between 12.36 MB/s and 42.73 MB/s. The results show that the proposed chaotic source coding scheme is practical with a high operating efficiency.

5. Conclusion

An efficient chaotic source coding scheme is proposed, in which the variable-length blocks of the source file are encoded within the precision of the register length. The bit shift tracks the synchronization between each initial value and the corresponding block. Simulation results justify the efficiency for practical applications with minor compression loss when compared with those obtained using the source entropy. In our future work, the chaotic coding of variable-length blocks will be adopted to enhance the efficiency of our previous work.[13]

References

[1] Hayes S, Grebogi C and Ott E 1993 *Phys. Rev. Lett.* **70** 3031

[2] Escribano F J, Kozic S, López L, Sanjuán M A F and Hasler M 2009 *IEEE Trans. Commun.* **57** 597

[3] Li H J and Zhang J S 2010 *Chin. Phys.* B **19** 050508

[4] Wong K W and Yuen C H 2008 *IEEE Trans. Circuits Syst. II, Exp. Briefs* **55** 1193

[5] Wang F L 2010 *Chin. Phys.* B **19** 090505

[6] Liu S B, Sun J, Xu Z Q and Liu J S 2009 *Chin. Phys.* B **18** 5219

[7] Luca M B, Serbanescu A, Azou S and Burel G 2004 *IEEE-Communications* Bucharest, Romania, June 3–5, 2004

[8] Nagaraj N, Vaidya P G and Bhat K G 2009 *Commun. Nonlinear Sci. Numer. Simulat.* **14** 1013

[9] Witten I H, Neal R M and Cleary J G 1987 *Commun. ACM.* **30** 520

[10] Chan D Y and Ku C Y 2005 *Appl. Math. Comput.* **167** 976

[11] Chan D Y, Yang J F and Chen S Y 2001 *IEEE Trans. Circuits Syst. Video Technol.* **11** 581

[12] Hardy Y and Sabatta D 2007 *Phys. Lett.* A **366** 575

[13] Wong K W, Lin Q Z and Chen J Y 2010 *IEEE Trans. Circuits Syst. II, Exp. Briefs* **57** 146

[14] Files are available at: ftp://ftp.cpsc.ucalgary.ca/pub/-projects/text.compression.corpus

[15] Said A 2004 *Hewlett-Packard Laboratories Report*, HPL-2004-76, Palo Alto, CA. Available at: http://www.hpl.-hp.com/techreports/2004/HPL-2004-76.pdf.