Article

Fast and efficient entropy coding architectures for massive data compression

Francesc Aulí-Llinàs [1]

[1] Department of Information and Communications Engineering, Universitat Autònoma de Barcelona; francesc.auli@uab.cat

Abstract: The compression of data is fundamental to alleviate costs of transmitting and storing massive data sets employed in myriad fields of our society. Most compression systems employ an entropy coder in their coding pipeline to remove the redundancy of coded symbols. The entropy coding stage needs to be efficient to yield high compression ratios, and fast to process large amounts of data rapidly. Despite their widespread use, entropy coders are commonly assessed for some particular scenario or coding system. This work provides a general framework to assess and optimize different entropy coders. First, the paper describes three main families of entropy coders, namely, those based on variable to variable length codes (V2VLC), arithmetic coding (AC) and tabled asymmetric numeral systems (tANS). Then, a low-complexity architecture for the most representative coder(s) of each family is presented, more precisely, a general version of V2VLC, the MQ, M and a fixed-length version of AC and two different implementation of tANS. These coders are evaluated under different coding conditions in terms of compression efficiency and computational throughput. The results obtained suggest that V2VLC and tANS achieve the highest compression ratios for most coding rates and that the AC coder that uses fixed-length codewords attains the highest throughput. The experimental evaluation discloses the advantages and shortcomings of each entropy coding scheme, providing insights that may help to select this stage in forthcoming compression systems.

Keywords: entropy coding; variable to variable length codes; arithmetic coding; asymmetric numeral systems

1. Introduction

Our society is immersed in a flow of data that supports all kinds of services and facilities such as online TV and radio, social networks, medical and remote sensing applications, or information systems, among others. The data employed in these applications are of different nature: from text and audio, to images and videos, strands of DNA, or environmental indicators, with a long etcetera. In many scenarios, these data are transmitted and/or stored for a fixed period of time or indefinitely. Despite enhancements on networks and storage devices, the amount of information globally generated increases so rapidly that only a small part can be saved [1,2]. *Data compression* is the solution to relieve the Internet traffic congestion and the storage necessities of data centers.

The compression of information has been a field of study for more than a half century. Since C. Shannon established the bases of information theory [3], the problem of how to reduce the amount of bits to store an original message has been a relevant topic of study [4–6]. Depending on the data type and their purposes, the compression regime may be lossy or lossless. Image, video and audio, for example, often use lossy regimes because the introduction of some distortion in the coding process does not disturb a human observer and achieves higher compression ratios [6]. Lossless regimes, on the other hand, recover the original message losslessly but achieve lower compression ratios. Also depending on the type and purposes of the data, the compression system may use different techniques. There are many systems specifically devised for particular types of data. Image and video

Citation: Aulí-Llinàs, F. Fast and efficient entropy coding architectures for massive data compression. *Technologies* **2023**, *1*, 0.
https://doi.org/

Received:
Accepted:
Published:

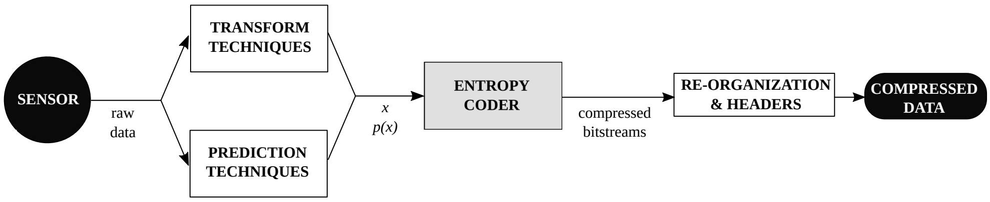


Figure 1. Stages of a conventional compression system.

compression capture samples that are transformed several times to reduce their visual redundancy [7–10]. Contrarily, compression of DNA often relies on a reference sequence to predict only the dissimilarities between the reference and the source [11]. There are universal methods like Lempel–Ziv–Welch [12,13] that code any kind of data, although they do not achieve the high compression ratios that specifically-devised systems yield. In recent years, deep-learning techniques have been spread in many compression schemes to enhance transformation and prediction techniques, obtaining competitive results in many fields [14–17].

Regardless of the coding system and the regime employed, most compression schemes rely on a coding stage called *entropy coding* to reduce the amount of information needed to represent the original data. See in Figure 1 that this stage is commonly situated just after the transformation and/or prediction stages. These stages prepare the data for the entropy coder producing binary symbols x with a corresponding probability $p(x)$. In general, these symbols are (transformed to) binary, so $x = \{0, 1\}$ is assumed in the following. The estimated or real [18,19] probability $p(x)$ depends on the amount of redundancy found in the original data. Adjacent pixels in an image often have similar colors, for instance, so their binary representation can be predicted with a high probability. Both x and $p(x)$ are fed to the entropy coder, which produces a compact representation of these symbols attaining compression. As the Shannon's theory of entropy dictates, the higher the probability of a symbol the lower its entropy, so higher compression ratios can be obtained. The main purpose of entropy coders is to attain coding efficiency close to the entropy of the original message while spending low computational resources, so large sets of data can be processed rapidly and efficiently.

Arguably, there are three main families of entropy coders. The first employs techniques that map one (or some) source symbols to codewords of different length. Such techniques exploit the repetitiveness of some symbols to represent them with a short codeword. The most complete theoretical model of such techniques is *variable to variable length codes (V2VLC)* [20,21]. The first entropy coding technique proposed in the literature, namely Huffman coding [22], uses a similar technique that maps each symbol to a codeword of variable length. Other techniques similar to V2VLC are Golomb-Rice coding [23,24], Tunstall codes [25,26], or Khodak codes [27], among others [20,28], which have been adopted in many scenarios [29–32]. The second main family of entropy coders utilize a technique called *arithmetic coding (AC)* [33]. The main idea is to divide a numeric interval in subintervals of varying size depending on the probability of the source symbols. The coding of any number within the latest interval commonly requires fewer bits than the original message and allows the decoder to reverse the procedure. Arithmetic coding has been widely spread and employed in many fields and standards [34–37] and there exist many variations and architectures [38–43]. The latest family of entropy coders are based on *asymmetric numeral systems (ANS)*, which is a technique introduced in the last decade [44]. ANS divides the set of natural numbers in groups that have a size depending on the probability of the symbols. The coding of the original message then traverses these groups so that symbols with higher probabilities employ the groups of largest size. The decoder reverses the path from the last to the first group, recovering the original symbols. There are different variants of ANS such as the range ANS or the uniform binary ANS, though the tabled ANS (tANS) is the most popular [45–47] since it can operate like a finite-state machine achieving high throughput. tANS has been recently adopted in many compression schemes [11,48,49], so it is employed in this work to represent this family of entropy coders.

The popularity of the aforementioned entropy coding techniques has changed depending on the trends and necessities of applications. Since entropy coding is in the core of the compression scheme, efficiency and speed are two important features. Before the introduction of ANS, Huffman coding and variants of V2VLC were generally considered the fastest techniques because they use a direct mapping between source symbols and codewords. Nonetheless, they are not the most efficient in terms of compression [50]. Arithmetic coding was preferable in many fields due to its highest compression efficiency although it was criticized since it commonly requires some arithmetic operations to code each symbol, achieving lower computational throughput [51]. Recent works claim that tANS achieves the efficiency of arithmetic coding while spending the computational costs of Huffman coding [44,52]. Although these discussions and claims are well grounded, they are commonly framed for a specific scheme or scenario without considering and evaluating other techniques. Differently from the previously cited references, this work provides a common framework to appraise different entropy coders. It also provides simple software architectures to test and optimize them using different coding conditions. The experimental evaluation discerns the advantages and shortcomings of each family of coders. The result of this evaluation is the main contribution of this work, which may help to select this coding stage in forthcoming compression schemes.

The rest of the paper is organized as follows. Section 2 describes the entropy coders evaluated in this work and proposes a software architecture for each. This section is divided in three subsections, one for each family of entropy coders. Section 2.1 presents a general method for V2VLC that uses pre-computed codes. Arithmetic coding is tackled in Section 2.2 describing two coders widely employed in image and video compression and an arithmetic coder that uses codewords of fixed length. Section 2.3 describes the tANS coding scheme and proposes two architectures for its implementation. All these coders are evaluated in terms of compression efficiency and computational throughput in Section 3, presenting experimental results obtained with different coding conditions. The last section discusses results and provides conclusions.

2. Materials and Methods

2.1. Variable to variable length codes (V2VLC)

Let $m = x_1 x_2 x_3 \ldots x_{|m|}$ be a message composed of a string of symbols, with $|m|$ denoting its length. V2VLC maps sequences of symbols in m to codewords $w_j = y_1 y_2 \ldots y_{|w_j|}$, with $y = \{0, 1\}$. When $p(x = 0)$ is close to 1, the original message contains sequences with many zeroes, so they can be mapped to a codeword of shorter length. The selection of these pairs of sequences-codewords needs and approach that uniquely maps each sequence to a codeword and inversely since otherwise the coding process could not guarantee the recovering of the original message. V2VLC are commonly represented with binary trees like those depicted in Figure 2. Each level in the top tree represents the encoding of a symbol, with left (right) branches being the coding of $x = 0$ ($x = 1$). Leaves represent the end of each sequence and are mapped to a codeword. Codewords are represented through the bottom tree in Figure 2 using the same structure as that in the top tree. Such a representation produces prefix codes [21], so the encoding process generates a unique compressed bitstream.

The determination of the optimal codewords for a fixed tree employs the well-known procedure described by Huffman [22], progressively joining the leaves with lowest probabilities. Each leaf in the top tree of Figure 2 has a probability to occur that can be determined by the probability of the sequence of symbols that it represents as

$$p(l_k) = p(x_i) \cdot p(x_{i+1}) \cdot p(x_{i+2}) \cdot \ldots \cdot p(x_{|l_k|}) \,, \tag{1}$$

with l_k denoting a leaf and $|l_k|$ the length of the sequence (or the depth level of the leaf). The construction of the codewords begins by joining those two l_k with lowest $p(l_k)$. This

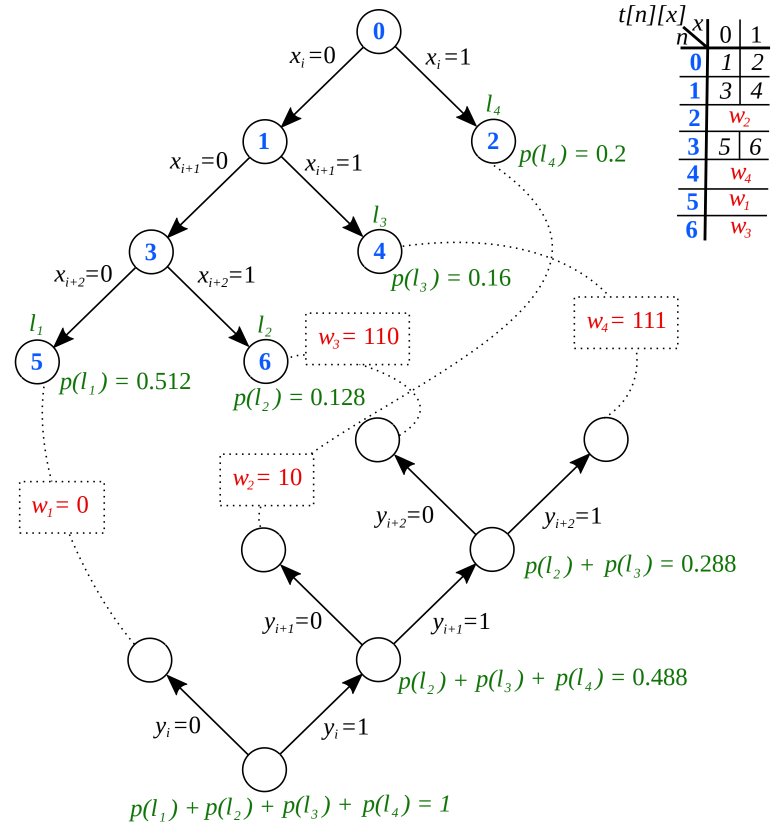


Figure 2. Illustration of V2VLC through binary trees.

procedure is repeated until a single leaf is left, which is the root of the tree containing the codewords. The toy example depicted in Figure 2 uses $p(x = 0) = 0.8$. The first leaves joined by this procedure are l_2 and l_3 (deepest level of the bottom tree in Figure 2), then l_4 and finally l_1. See in this figure that the sequence of original symbols $m' = 000$ is mapped to $w_1 = 0$, coding three symbols with one bit. The *compression efficiency* achieved by such a scheme is determined through the weighted length of the sequences of symbols and the weighted length of the codewords according to

$$E = \frac{\sum_j |w_j| \cdot p(w_j)}{\sum_k |l_k| \cdot p(l_k)} \,. \tag{2}$$

where $p(w_j)$ is the probability of codeword w_j, which is the same to that l_k mapped to this codeword (e.g., $p(w_2) = p(l_4)$ in Figure 2). The above expression divides the average length of the codewords (considering their probability of appearance) by the average length of the sequences (also considering their appearance probability). Otherwise stated, it divides the length of the compressed data by the length of the original data, resulting in the efficiency of the V2VLC scheme. The difference between E and the entropy of the source is called *redundancy* and is determined as

$$R = E - \sum_x p(x) \cdot log_2 \frac{1}{p(x)} \,. \tag{3}$$

R is employed to assess the optimality of the coding scheme.

The main difficulty of V2VLC is to find a low complexity algorithm that minimizes the redundancy for a given $p(x)$. This is an open problem in the field tackled in different ways [21,28,53,54]. However, to find optimal V2VLC is not part of the compression procedure, which can use codes determined a priori. This work utilizes pre-computed codes created with trees of 16 leaves or less employing a brute-force approach to find the optimal V2VLC scheme. The encoding procedure is described in Algorithm 1. This procedure is

Algorithm 1 V2VLC
Parameters: x bit to code
Initialization: $n \leftarrow 0$

1: **if** $t[n] \neq w$ **then**
2:     $n \leftarrow t[n][x]$
3: **else**
4:     emitCodeword($t[n][x]$)
5:     $n \leftarrow 0$
6: **end if**

Algorithm 2 emitCodeword
Parameters: w codeword to emit
Initialization: $T \leftarrow 0, b \leftarrow 8$

for $i \in [|w| - 1, 0]$ **do**
    $T \leftarrow (T \ll 1)$ OR $((w \gg i)$ AND $1)$
    if $b > 0$ **then**
        $b \leftarrow b - 1$
    else
        writeByte(T)
        $T \leftarrow 0$
        $b \leftarrow 8$
    end if
end for

called for each symbol of the message from a loop that is not included in this (and following) algorithms. It uses the table depicted in the top-right corner of Figure 2 denoted by $t[n][x]$. Each row in this table is a node of the tree. The first and second columns contain the next node when $x = 0$ or $x = 1$, respectively, except when reaching a leaf, in which case a codeword is emitted. As seen in Algorithm 1, $n = 0$ at the beginning of the process and then the encoding of each symbol simply updates n (in line 2) except when emitting a codeword. In this case, the codeword is emitted through the procedure described in Algorithm 2 and n is reset. The procedure that emits the codeword uses variable T to store a byte that is filled with the bits of the codeword and written to the disk (or transmitted) when necessary. Decoding inverses the procedure using a table constructed with the tree of codewords (not shown). Note that these procedures almost do not require arithmetic operations but only access to memory positions.

2.2. *Arithmetic coding (AC)*

Differently from V2VLC, the output of conventional arithmetic coders is a very long codeword. Figure 3 depicts an example of the interval division procedure carried out by arithmetic coding. It typically begins with interval $I = (0, 1)$, which is split in $I' = (0, p(x = 0)]$ and $I'' = (p(x = 0), 1)$ to code the first symbol. If $x_1 = 0$, I' is further employed to code following symbols, whereas $x_1 = 1$ keeps I''. In practice, the division of the interval uses hardware registers of at most 64 bits, so the interval is computed progressively. I is commonly represented as $I = [L, U)$ with L and U being the lower and upper bound of the interval, respectively. L and U are initialized to 0 and to the largest integer available, respectively. The binary representation of L and U are completely different at the beginning of coding but, as the interval is subsequently partitioned in $I''' = [L', U')$, some bits in the leftmost part of the binary representation of L' and U' become equal. This happens because the interval becomes smaller in each new partition, with L' and U' being closer. These bits do not change in further partitions so they can be emitted as a segment of the codeword before the end of coding. Once they are emitted, the remaining bits in L' and U' are shifted to the left as many positions as bits have been emitted. The emission of these bits that partially belongs to the codeword is a procedure called *renormalization*.

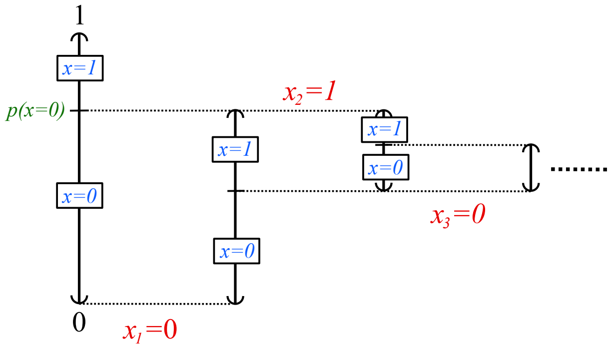


Figure 3. Illustration of the interval division carried out by arithmetic coding.

Algorithm 3 ACFLW

Parameters: x bit to code, P probability

Initialization: $L \leftarrow 0, S \leftarrow 2^{\mathcal{W}} - 1$

if $x = 0$ **then**
  $S \leftarrow (S \cdot P) \gg \mathcal{B}$
else
  $q \leftarrow ((S \cdot P) \gg \mathcal{B}) + 1$
  $L \leftarrow L + q$
  $S \leftarrow S - q$
end if
if $S = 0$ **then**
  emitCodeword(L)
  $L \leftarrow 0$
  $S \leftarrow 2^{\mathcal{W}} - 1$
end if

Three arithmetic coders are evaluated in the following due to its widespread use and popularity. The MQ coder [55] is a descendant of the Q coder [56]. It is used in JPEG [29], JBIG2 [57] and JPEG2000 [34] standards due to its high efficiency and low computational complexity. It incorporates many computational optimizations. It is not detailed herein since it has been thoroughly described in the literature (see [58] for a comprehensive description). The M coder [59] employs lookup tables and a reduced range of interval sizes. Variants of such coder are employed in popular video standards such as H.264/AVC [35] and H.265/HEVC [36] (see [59] for a review).

The main particularity of the third arithmetic coder evaluated is that it obviates renormalization. Renormalization is useful to employ all bits of the integer registers during the coding process but it spends significant computational resources since it is intensively executed. The method proposed in [51,60–64] eliminates the use of renormalization by employing arithmetic coding with fixed-length codewords (ACFLW). It splits intervals as previously described but it does emit partial segments of the final codeword. Instead, when the interval size is 0, it dispatches a codeword and begins with a new one. This may cause an efficiency loss when the interval size is small and $p(x)$ is high because the interval is split with poor precision. Nonetheless, it is shown in [51] that intervals of moderate size penalize efficiency only slightly. Our implementation uses intervals of a size of $\mathcal{W} = 32$ bits as it is recommended in [51]. ACFLW uses variables L and S to represent the lower bound of the interval and its size, respectively. At the beginning of the coding process $L = 0$ and $S = 2^{\mathcal{W}} - 1$. The coding of $x = 0$ requires the following operation

$$S \leftarrow (S \cdot P) \gg \mathcal{B}\,, \tag{4}$$

with $\gg$ being a bit shift operation to the right and P denoting the probability $p(x = 0)$ expressed in the range $[0, 2^{\mathcal{B}} - 1]$ (i.e., $P = \lfloor p(x = 0) \cdot 2^{\mathcal{B}} \rfloor$ with $\lfloor \cdot \rfloor$ being the floor operation). $\mathcal{B}$ is the number of bits to express the symbol's probability. Our implementation

Z	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
x=0		4	5		6	7		8	9		10	11		12	13		14	15		16	17	
x=1				5			6			7			8			9			10			11

K = 001001

Z	X_0	X_1	D
1			
2		8	
3		11	
4	6		
5	7		
6	9		(0,4)
7	10		(0,5)
8			(1,2)
9			(0,6)
10			(0,7)
11			(1,3)

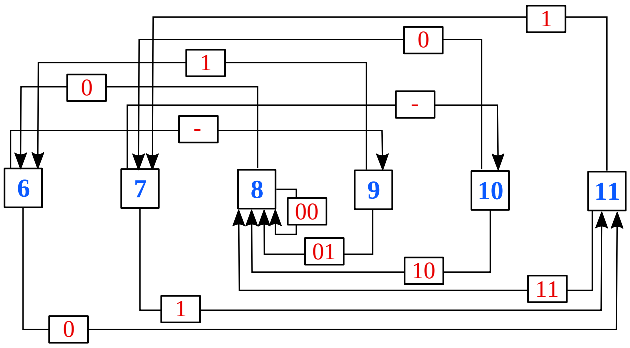


Figure 4. Illustration of tANS via asymmetric groups of numbers (top), a tabled automaton (bottom-left) and a state machine (bottom-right).

uses $\mathcal{B} = 15$ since it provides high precision requiring few computational resources [51]. The coding of $x = 1$ requires the following operations

$$
\begin{aligned}
S &\leftarrow S - ((S \cdot P) \gg \mathcal{B}) - 1\,, \\
L &\leftarrow L + ((S \cdot P) \gg \mathcal{B}) + 1\,.
\end{aligned} \tag{5}
$$

These operations employ an integer multiplication to split the interval. Such an operation requires a single clock cycle in modern CPUs, so it does not penalize throughput significantly. Algorithm 3 details the procedure to encode symbols. The procedure to emit the codeword is the same as that in Algorithm 2 (with L being the codeword). Decoding uses a similar procedure (not shown). The compression efficiency of arithmetic coders can not be determined a priori like with V2VLC schemes but it has to be appraised experimentally. The next section proposes a series of tests that assess their performance compared to the other coders.

2.3. Tabled Asymmetric Numeral Systems (tANS)

tANS represents the message with a state denoted by Z that is progressively increased during the encoding of symbols. Coding requires a pre-computed table. For a probability distribution of $p(x = 0) = 2/3$, for instance, this table is like that shown in the top of Figure 4. The first row of the table represents the current state Z, whereas the second and third rows are the next Z when coding $x = 0$ and $x = 1$, respectively. The cells filled in the second and third rows have the distribution of $p(x)$, creating asymmetric groups of numbers. Z can be set to any position of the table at the beginning of the encoding procedure. If the next symbol to encode is $x = 0$, the procedure then advances to that column of the table indicated in the second row. If the symbol is $x = 1$, the procedure is the same but using the third row of the table. The top table in Figure 4 depicts an example (in orange) in which the message $m'' = 001$ is encoded. State Z is initialized at $Z = 5$ and then transitions to $Z = 7$ because the first symbol is $x = 0$ and this is the column in which the second row of the table has a 5 too. The next symbol is also $x = 0$, so the state is transitioned to $Z = 10$. Since the last symbol is $x = 1$, the state 10 is found in the third row of the table at the column in which $Z = 23$. 23 is the codeword emitted to the decoder. The decoding process starts with the last state (i.e., the emitted codeword) and reverses the procedure. Differently to the entropy coders previously described, decoding the message begins with the latest symbol coded and goes backwards as if they were put

Algorithm 4 tANS
Parameters: x bit to code
Initialization: $Z \leftarrow |k|, Tr \leftarrow 0, b \leftarrow 7$

1: **while** $Z > |k_x| \cdot 2 - 1$ **do**
2: $\quad Tr \leftarrow Tr$ OR $((Z \text{ AND } 1) \ll b)$
3: $\quad Z \leftarrow Z \gg 1$
4: $\quad$ **if** $b = 0$ **then**
5: $\qquad$ writeByte(Tr)
6: $\qquad Tr \leftarrow 0$
7: $\qquad b \leftarrow 7$
8: $\quad$ **else**
9: $\qquad b \leftarrow b - 1$
10: $\quad$ **end if**
11: **end while**
12: $Z \leftarrow X_x[Z]$

in a stack. The key to achieve compression is that coding symbols with higher probabilities advances Z more slowly than coding symbols with lower probabilities, so the final state can be represented with fewer bits than the original message. In the extreme case of $p(x = 0) \approx 1$, for instance, a final state $Z = 10$ may represent the coding of a message with 10 consecutive 0s but requiring only 4 bits (since $10 = 1010_{(2)}$).

tANS can not use an infinite number of states in practice, so Z is represented with a fixed number of bits. To this end, the top table depicted in Figure 4 is transformed to a finite-state machine, with the coding of each symbol being state transitions. There exist many different automatons for each distribution [46], so a key K that represents a unique scheme needs to be chosen first. A suitable key for the distribution of the above example might be $K = 001001$ since it strictly respects $p(x = 0) = 2/3$. The table shown in Figure 4(bottom-left) is generated with this key. The first column of this table is Z. Although the range of Z is $Z \in [1, 11]$, only those rows from $|K|$ to $|K| \cdot 2 - 1$ belong to the automaton (depicted in gray in the figure). The construction of this table begins filling the rows of the fourth column from state $Z = 6$ (i..e, $|K|$) to $Z = 11$ (i.e., $|K| \cdot 2 - 1$), which contains the decoding tuple D. The first element of the tuple is filled with the symbols of the key in the same order. The second element of the tuple is the first empty cell for that symbol in the second or third columns of the table. These columns contain the state transitions for symbols $x = 0$ and $x = 1$, respectively. They are denoted by X_0 and X_1. X_x is only filled from $|K_x|$ to $|K_x| \cdot 2 - 1$, with $|K_x|$ denoting the number of 0s or 1s in K. Following our example, the second element in tuple D for $Z = 6$ is 4 since $|K_0| = 4$. The cell X_0 for the row $Z = 4$ is then filled with 6 since this is the state from which it comes. This process is repeated for each state resulting in the table depicted in Figure 4.

The table generated with key K aids the coding of symbols. Coding x removes as many least significant bits of the binary representation of the current Z until $Z \in [|K_x|, |K_x| \cdot 2 - 1]$. These removed bits are emitted by the coder forming the compressed bitstream. The next state of the automaton is that given in column X_x. This process is automated via the state-machine depicted in the bottom-right part of Figure 4, which is generated with the bottom-left table of Figure 4. Transitions in the upper part of this automaton represent the coding of $x = 0$ whereas those in the lower part represent $x = 1$. The emission of bits in each transition, if necessary, is depicted in the middle of each arrow. Similarly to V2VLC, the selection of the key that achieves lowest redundancy uses a full search approach since this process is carried out before coding. Some strategies to accelerate the selection of K can be found in [46].

The implementation of such a coding scheme may consider two different architectures. The first is embodied in Algorithm 4. This procedure removes bits from Z and emits them until Z is in the range $[|K_x|, |K_x| \cdot 2 - 1]$ (lines 2 and 3). The next state is set employing X_x in the last line of the algorithm. The operations from line 4 to 10 write a byte to the disk when it is filled, similarly to the procedure described in Algorithm 2. The

Algorithm 5 tANSAuto
Parameters: x bit to code
Initialization: $Z \leftarrow |k|, Tr \leftarrow 0, b \leftarrow 7$

for $i \in [|W[Z][x]| - 1, 0]$ **do**
    $Tr \leftarrow Tr \ll 1$
    $Tr \leftarrow Tr$ OR $((W[Z][x] \gg i)$ AND $1)$
    if $b = 0$ **then**
        writeByte(Tr)
        $Tr \leftarrow 0$
        $b \leftarrow 7$
    else
        $b \leftarrow b - 1$
    end if
end for
$Z \leftarrow S[Z][x]$

second architecture proposed for tANS is detailed in Algorithm 5. Instead of computing the next state by progressively removing bits from Z, this architecture stores the transitions and emitted bits of the automaton in tables constructed a priori. Table $S[Z][x]$ stores the transition to the next state for the current Z and symbol coded. Table $W[Z][x]$ contains the bits emitted when coding x in the state Z. These tables can be constructed using the automaton of Figure 4. The first three lines in Algorithm 5 emit the bits for the state transition and write a full byte in disk when necessary. The last line of the algorithm updates Z. Decoding uses similar algorithms for both architectures.

Note that all entropy coders described above recover the original message losslessly. This is a characteristic of entropy coding, but it does not entail the compression system to use a lossless regime too. Lossy regimes commonly introduce distortion in the transformation or prediction stages.

3. Results

3.1. Data and metrics

The data employed in the following tests are produced artificially given a probability distribution. The symbols are generated assuming independence and identical distribution. The range of the probability distribution evaluated is $p(x = 0) = [0.5, 1)$ because same results are obtained for probabilities biased toward $x = 1$. The probability is fed directly to the coder, disregarding the estimation mechanisms that some coders use. This provides a common framework for all coders. Also, the same artificially generated data are employed for all coders, with sequences of 2^{28} symbols. All coders are programmed in Java and tests are executed with an Intel Core i7-3770 @ 3.40 GHz. Except when otherwise stated, the V2VLC scheme employed in the tests uses trees of 16 leaves and the tANS scheme uses an automaton with 16 states. Both are set to the same number of leaves/states so that the tables employed by such coders have similar size. As seen in the experiments below, using 16 leaves or states achieves near optimal compression efficiency. Compression results are reported via the redundancy achieved by the coder (as defined in Equation 3), whereas computational throughput is evaluated in terms of mega symbols coded per second (MS/s).

3.2. Tests

The first test evaluates compression efficiency. Figure 5 depicts the results for all coders and the full range of probabilities. The vertical axis of the figure is the redundancy produced by the coder, reported in bits per symbol (bps). The horizontal axis reports the probability distribution. The efficiency achieved by tANS (Algorithm 4) and tANSAuto (Algorithm 5) is the same, so only the first is depicted. The results reported in this figure indicate that the MQ coder penalizes the coding efficiency when the probability is low, especially at $p(x = 0) \approx 0.62$. V2VLC and tANS achieve an efficiency that is very close

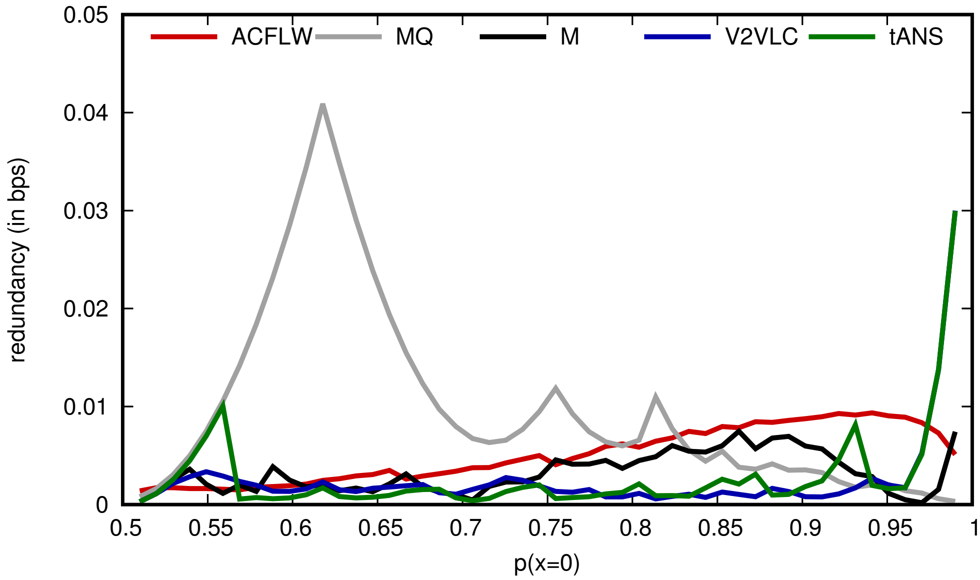


Figure 5. Compression efficiency evaluation of all coders.

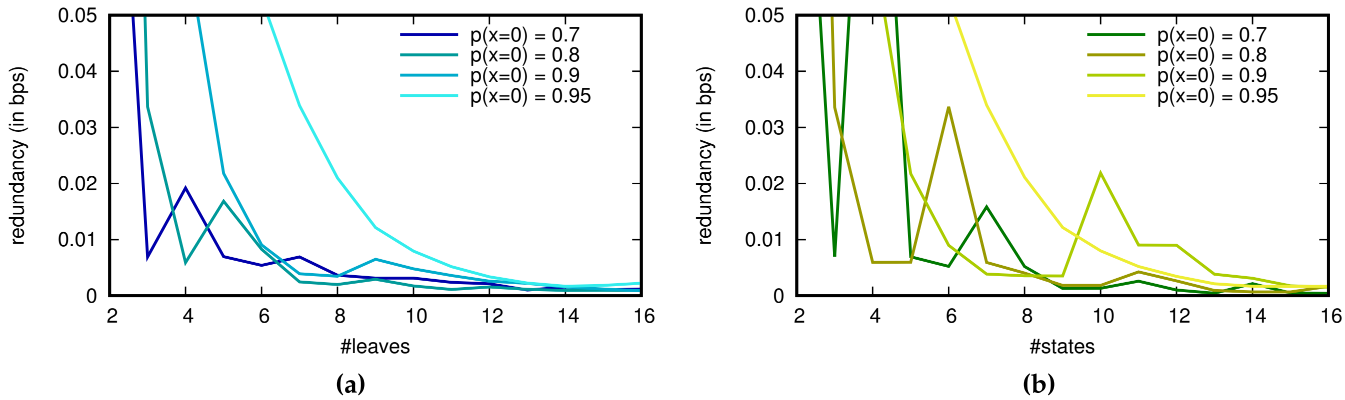
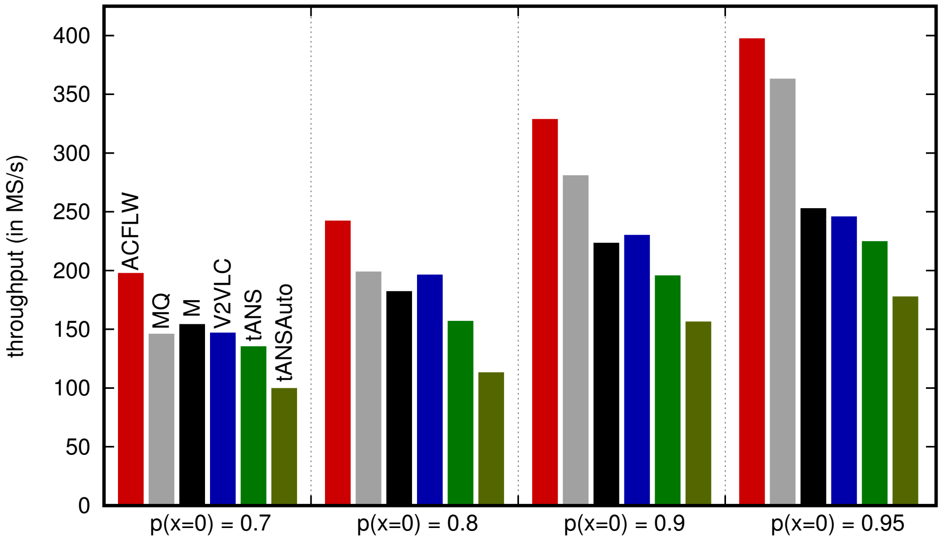


Figure 6. Compression efficiency evaluation depending on the number of: **(a)** leaves of the V2VLC scheme and **(b)** states of the tANS automaton.

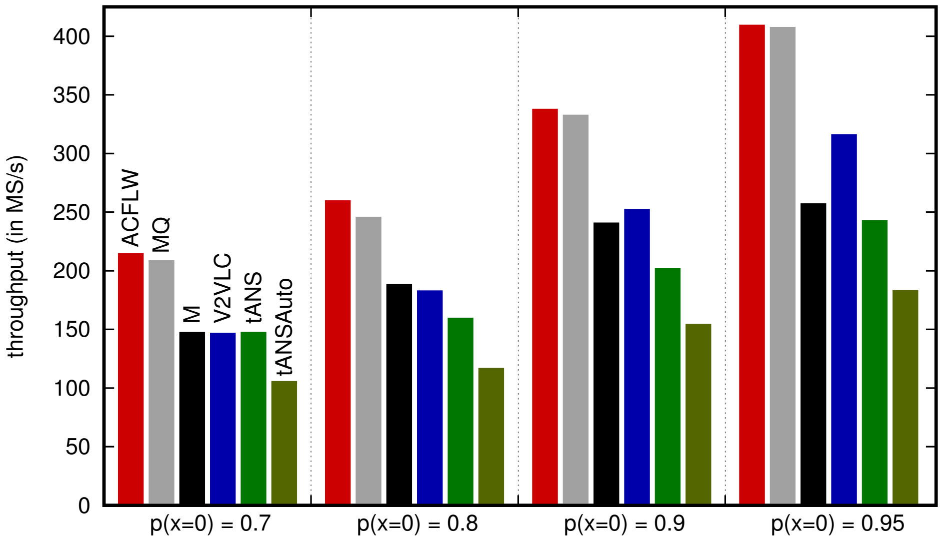
to the entropy for most probabilities, followed by the M coder and ACFLW. These coders yield a redundancy of less than 0.01 bps for most probabilities, suggesting that they are highly efficient.

The second test appraises the coding efficiency of V2VLC and tANS depending on the number of leaves and states, respectively. Figure 6 depicts the redundancy on the vertical axis and the number of leaves/states employed by the coder in the horizontal axis. Only a representative set of probabilities are depicted in the figure, though results hold for the rest. The redundancy achieved by the V2VLC scheme (Figure 6(a)) decreases smoothly as more leaves are employed, regardless of $p(x)$. As seen in the Figure, the use of 16 leaves is enough to achieve competitive performance. The tANS automaton (Figure 6(b)) obtains redundancy results that increase and decrease depending on the number of states, except when using a high $p(x)$. These irregularities are caused because $p(x)$ does not fit well for some number of states, reducing the efficiency of the coder. 16 states seems to be enough to obtain near-optimal efficiency.

The third test analyzes computational throughput. Figure 7 reports the obtained results for all coders when encoding and decoding. Again, only a significant set of probabilities is depicted in the figure, though results hold for the rest. The figure indicates that higher probabilities lead to higher throughput. This is because a higher $p(x)$ obtains higher compression efficiency, requiring the emission of fewer bits and so accelerating



(a)



(b)

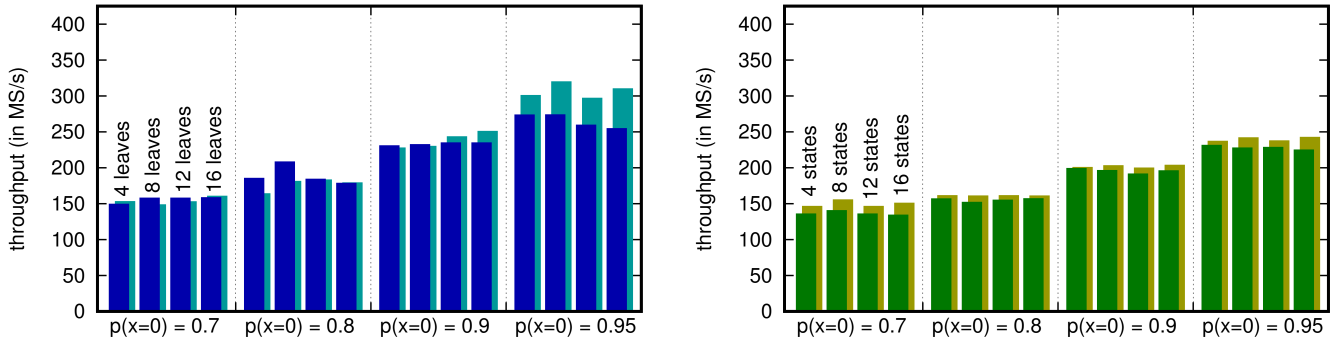
Figure 7. Computational throughput evaluation of all coders when **(a)** encoding and **(b)** decoding.

the coding process. Regardless of the probability, ACFLW obtains the highest throughput followed by the MQ coder for most probabilities. The V2VLC and both architectures of tANS attain lower throughput, with tANSAuto being the slowest. The results also suggest that decoding is generally faster than encoding, which is a common feature of all entropy coders because decoding requires slightly simpler operations.

The last test evaluates the computational throughput achieved by the V2VLC scheme and tANS depending on the number of leaves/states. Figure 8 reports the results obtained, which suggest that the number of leaves/states does not significantly affect the throughput achieved. This holds for both the encoding and decoding process.

4. Discussion

Entropy coding is in the core of most compression systems and it has to be chosen and implemented carefully to obtain high compression efficiency while using few computational resources. The techniques employed by each family of entropy coders use different mechanisms to attain compression, so comparison requires a common framework. This paper presents software architectures for the most representative coder(s) of each



(a) **(b)**

Figure 8. Computational throughput evaluation depending on the number of: **(a)** leaves of the V2VLC scheme and **(b)** states in the tANS automaton. Columns in the front (back) are for the encoding (decoding) process.

Table 1. Summary of the obtained experimental results. $\Downarrow$, $\approx$ and $\Uparrow$ indicate low, medium and high performance, respectively.

	low rates		**medium rates**		**high rates**	
	coding efficiency	comput. through.	coding efficiency	comput. through.	coding efficiency	comput. through.
V2VLC	$\approx$	$\approx$	$\Uparrow$	$\approx$	$\Uparrow$	$\approx$
MQ	$\Downarrow$	$\approx$	$\approx$	$\Uparrow$	$\Uparrow$	$\Uparrow$
M	$\Uparrow$	$\approx$	$\approx$	$\approx$	$\Downarrow$	$\approx$
ACFLW	$\Uparrow$	$\Uparrow$	$\approx$	$\Uparrow$	$\Downarrow$	$\Uparrow$
tANS	$\approx$	$\approx$	$\Uparrow$	$\Downarrow$	$\Uparrow$	$\approx$
tANSAuto	$\approx$	$\Downarrow$	$\Uparrow$	$\Downarrow$	$\Uparrow$	$\Downarrow$

family and evaluates them in terms of efficiency and throughput. Table 1 summarizes the results obtained in the experimental tests depicting the coding efficiency and computational throughput of each coder at low, medium and high rates. These results suggest that when coding efficiency is the most important aspect of the system, V2VLC, tANS, or the M coder are the best options. ACFLW or the MQ coder seem to be the fastest despite the use of some arithmetic operations to code symbols. For the two architectures of tANS, the one that re-computes the state for each coded symbol (instead of using pre-computed tables) achieves higher throughput. Both for V2VLC and tANS, using more leaves/states significantly reduce the redundancy of the system and slightly improves throughput. Future research may adapt and appraise the presented coders in dedicated hardware architectures such as commodity GPUs or ASICs, which may help to further accelerate the compression process.

Funding: This research was funded by the Spanish Ministry of Science, Innovation and Universities (MICIU) and by the European Regional Development Fund (FEDER) under Grant PID2021-125258OB-I00 and by the Catalan Government under Grant SGR2021-00643.

Data Availability Statement: The data employed in this work are artificially generated. Most of the sources of our implementation are freely available at https://deic.uab.cat/~francesc.

Conflicts of Interest: The author declares no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

V2VLC	Variable to variable length codes
AC	Arithmetic coding
ACFLW	Arithmetic coding with fixed-length codewords
ANS	Asymmetric numeral systems
tANS	Tabled asymmetric numeral systems

References

1. Taylor, P. Worldwide data created. Technical report, Statista, 2022. URL: https://www.statista.com/statistics/871513/worldwide-data-created/.
2. Berisha, B.; Mëziu, E.; Shabani, I. Big Data Analytics in Cloud Computing: An overview. *SPRINGER Journal of Cloud Computing* **2022**, *11*.
3. Shannon, C.E. A mathematical theory of communication. *Bell System Technical Journal* **1948**, *27*, 379–423.
4. Storer, J. *Data compression: methods and theory*; Computer Science Press, 1988.
5. Salomon, D.; Motta, G. *Handbook of Data Compression*; Springer: New York, NY 10013 USA, 2009.
6. Sayood, K. *Introduction to Data Compression*; Elsevier, 2019.
7. Xiong, Z.; Wu, X.; Cheng, S.; Hua, J. Lossy-to-Lossless Compression of Medical Volumetric Data Using Three-Dimensional Integer Wavelet Transforms. *IEEE Trans. Med. Imag.* **2003**, *22*, 459–470.
8. Xiong, R.; Xu, J.; Wu, F.; Li, S. Barbell-Lifting Based 3-D Wavelet Coding Scheme. *IEEE Trans. Circuits Syst. Video Technol.* **2007**, *17*, 1256–1269.
9. Auli-Llinas, F.; Marcellin, M.W.; Serra-Sagrista, J.; Bartrina-Rapesta, J. Lossy-to-Lossless 3D Image Coding through Prior Coefficient Lookup Tables. *ELSEVIER Information Sciences* **2013**, *239*, 266–282.
10. Auli-Llinas, F. Entropy-based evaluation of context models for wavelet-transformed images. *IEEE Trans. Image Process.* **2015**, *24*, 57–67.
11. Fritz, M.H.Y.; Leinonen, R.; Cochrane, G.; Birney, E. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research* **2011**, *5*, 734–740.
12. Ziv, J.; Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory* **1978**, *24*, 530–536.
13. Welch. A Technique for High-Performance Data Compression. *Computer* **1984**, *17*, 8–19.
14. Leung, M.K.K.; Delong, A.; Alipanahi, B.; Frey, B.J. Machine Learning in Genomic Medicine: A Review of Computational Problems and Data Sets. *Proc. IEEE* **2016**, *104*, 176–197.
15. Wang, T.; Li, F.; Qiao, X.; Cosman, P.C. Low-Complexity Error Resilient HEVC Video Coding: A Deep Learning Approach. *IEEE Trans. Image Process.* **2021**, *30*, 1245–1260.
16. Chen, Z.; Gu, S.; Lu, G.; Xu, D. Exploiting Intra-Slice and Inter-Slice Redundancy for Learning-Based Lossless Volumetric Image Compression. *IEEE Trans. Image Process.* **2022**, *31*, 1697–1707.
17. Wei, Z.; Niu, B.; Xiao, H.; He, Y. Isolated Points Prediction via Deep Neural Network on Point Cloud Lossless Geometry Compression. *IEEE Trans. Circuits Syst. Video Technol.* **2023**, *33*, 407–420.
18. Auli-Llinas, F. Stationary probability model for bitplane image coding through local average of wavelet coefficients. *IEEE Trans. Image Process.* **2011**, *20*, 2153–2165.
19. Auli-Llinas, F.; Marcellin, M.W. Stationary Probability Model for Microscopic Parallelism in JPEG2000. *IEEE Trans. Multimedia* **2014**, *16*, 960–970.
20. Gallager, R. Variations on a theme by Huffman. *IEEE Trans. Inf. Theory* **1978**, *24*, 668–674.
21. Fabris, F. Variable-length-to-variable length source coding: a greedy step-by-step algorithm. *IEEE Trans. Inf. Theory* **1992**, *38*, 1609–1617.
22. Huffman, D. A method for the construction of minimum redundancy codes. *Proc. IRE* **1952**, *40*, 1098–1101.
23. Golomb, S. Run-length encodings. *IEEE Trans. Inf. Theory* **1966**, *12*, 399–401.
24. Rice, R.; Plaunt, J. Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data. *IEEE Trans. Commun.* **1971**, *19*, 889–897.
25. Savari, S.A.; Gallager, R.G. Generalized Tunstall codes for sources with memory. *IEEE Trans. Inf. Theory* **1997**, *43*, 658–668.
26. Drmota, M.; Reznik, Y.A.; Szpankowski, W. Tunstall Code, Khodak Variations, and Random Walks. *IEEE Trans. Inf. Theory* **2010**, *56*, 2928–2937.
27. Bugeaud, Y.; Drmota, M.; Szpankowski, W. On the Construction of (Explicit) Khodak's Code and Its Analysis. *IEEE Trans. Inf. Theory* **2008**, *54*, 5073–5086.
28. Szpankowski, W. Asymptotic Average Redundancy of Huffman (and Other) Block Codes. *IEEE Trans. Inf. Theory* **2000**, *46*, 2434–2443.
29. International Organization for Standardization / International Electrotechnical Commission. *Digital compression and coding for continuous-tone still images*; Number ISO/IEC 10918-1, 1992.
30. International Organization for Standardization / International Electrotechnical Commission. *Coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s. Part 3: audio*; Number ISO/IEC 11172-3, 1995.

31. International Organization for Standardization / International Electrotechnical Commission. *JPEG-LS lossless and near-lossless compression for continuous-tone still images*; Number ISO/IEC 14495-1, 1999.
32. Internet Engineering Task Force. *Free Lossless Audio Codec (FLAC)*; Number IETF draft-ietf-cellar-flac-09, 2023.
33. Rissanen, J. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development* **1976**, *20*, 198–203.
34. International Organization for Standardization / International Electrotechnical Commission. *Information technology - JPEG 2000 image coding system - Part 1: Core coding system*; Number ISO/IEC 15444-1, 2000.
35. International Telecommunication Union. *Advanced video coding for generic audiovisual services*; Number ITU H.264, 2005.
36. International Telecommunication Union. *High Efficiency Video Coding Standard*; Number ITU H.265, 2013.
37. International Telecommunication Union. *Versatile video coding*; Number ITU H.266, 2022.
38. Pinho, A.J. Adaptive Context-Based Arithmetic Coding of Arbitrary Contour Maps. *IEEE Signal Process. Lett.* **2001**, *8*, 4–6.
39. Hong, D.; Eleftheriadis, A. Memory-Efficient Semi-Quasi Renormalization for Arithmetic Coding. *IEEE Trans. Circuits Syst. Video Technol.* **2007**, *17*, 106–110.
40. Grangetto, M.; Magli, E.; Olmo, G. Distributed Arithmetic Coding for the Slepian-Wolf Problem. *IEEE Trans. Signal Process.* **2009**, *57*, 2245–2257.
41. Hu, W.; Wen, J.; Wu, W.; Han, Y.; Yang, S.; Villasenor, J. Highly Scalable Parallel Arithmetic Coding on Multi-Core Processors Using LDPC Codes. *IEEE Trans. Commun.* **2012**, *60*, 289–294.
42. Auli-Llinas, F.; Enfedaque, P.; Moure, J.C.; Sanchez, V. Bitplane Image Coding with Parallel Coefficient Processing. *IEEE Trans. Image Process.* **2016**, *25*, 209–219.
43. Bartrina-Rapesta, J.; Blanes, I.; Auli-Llinas, F.; Serra-Sagrista, J.; Sanchez, V.; Marcellin, M. A lighweight contextual arithmetic coder for on-board remote sensing data compression. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 4825–4835.
44. Duda, J. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. *arXiv:1311.2540* **2013**.
45. Duda, J.; Niemiec, M. Lightweight compression with encryption based on Asymmetric Numeral Systems. *arXiv:1612.04662* **2016**.
46. Blanes, I.; Hernández-Cabronero, M.; Serra-Sagristà, J.; Marcellin, M. Redundancy and Optimization of tANS Entropy Encoders. *IEEE Trans. Multimedia* **2021**, *23*, 4341–4350.
47. Duda, J.; Niemiec, M. Lightweight Compression with Encryption Based on Asymmetric Numeral Systems. *ACM Int. J. Appl. Math. Comput. Sci.* **2023**, *33*, 45–55.
48. *Zstandard Compression and the application/zstd Media Type*; Number IETF 8878, 2021.
49. International Organization for Standardization / International Electrotechnical Commission. *JPEG XL image coding system — Part 1: Core coding system*; Number ISO/IEC 18181-1, 2022.
50. Witten, I.; Neal, R.; Cleary, J. Arithmetic coding for data compression. *Communications of ACM* **1987**, *30*, 520–540.
51. Auli-Llinas, F. Context-adaptive Binary Arithmetic Coding with Fixed-length Codewords. *IEEE Trans. Multimedia* **2015**, *17*, 1385–1390.
52. Hsieh, P.A.; Wu, J.L. A Review of the Asymmetric Numeral System and Its Applications to Digital Images. *MDPI Entropy* **2022**, *24*, 375.
53. Baer, M.B. Redundancy-Related Bounds for Generalized Huffman Codes. *IEEE Trans. Inf. Theory* **2011**, *57*, 2278–2290.
54. Kirchhoffer, H.; Marpe, D.; Schwarz, H.; Wiegand, T. Properties and Design of Variable-to-Variable Length Codes. *ACM Trans. Multimedia Comput. Commun. Appl.* **2018**, *14*, 1–19.
55. Rabbani, M.; Joshi, R. An overview of the JPEG 2000 still image compression standard. *ELSEVIER Signal Processing: Image Communication* **2002**, *17*, 3–48.
56. Slattery, M.; Mitchell, J. The Qx-coder. *IBM Journal of Research and Development* **1998**, *42*, 767–784.
57. International Organization for Standardization / International Electrotechnical Commission. *Information technology - Lossy/lossless coding of bi-level images*; Number ISO/IEC 14492, 2001.
58. Taubman, D.S.; Marcellin, M.W. *JPEG2000 Image compression fundamentals, standards and practice*; Kluwer Academic Publishers: Norwell, Massachusetts 02061 USA, 2002.
59. Marpe, D.; Schwarz, H.; Wiegand, T. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. *IEEE Trans. Circuits Syst. Video Technol.* **2003**, *13*, 620–636.
60. Boncelet, C.G. Block Arithmetic Coding for Source Compression. *IEEE Trans. Inf. Theory* **1993**, *39*, 1546–1554.
61. Teuhola, J.; Raita, T. Arithmetic Coding into Fixed-Length Codewords. *IEEE Trans. Inf. Theory* **1994**, *40*, 219–223.
62. Chan, D.Y.; Yang, J.F.; Chen, S.Y. Efficient Connected-Index Finite-Length Arithmetic Codes. *IEEE Trans. Circuits Syst. Video Technol.* **2001**, *11*, 581–593.
63. Xie, Y.; Wolf, W.; Lekatsas, H. Code Compression Using Variable-to-fixed Coding Based on Arithmetic Coding. In Proceedings of the Proc. IEEE Data Compression Conference, 2003, pp. 382–391.
64. Reavy, M.D.; Boncelet, C.G. An Algorithm for Compression of Bilevel Images. *IEEE Trans. Image Process.* **2001**, *10*, 669–676.