entropy

IMPACT FACTOR 2.738

Indexed in: PubMed

Article

The Compression Optimality of Asymmetric Numeral Systems

Josef Pieprzyk, Jarek Duda, Marcin Pawłowski, Seyit Camtepe, Arash Mahboubi and Paweł Morawiecki

MDPI

https://doi.org/10.3390/e25040672

entropy

MDPI

Article

The Compression Optimality of Asymmetric Numeral Systems

Josef Pieprzyk [1,2,*], Jarek Duda [3], Marcin Pawłowski [3], Seyit Camtepe [2], Arash Mahboubi [4] and Paweł Morawiecki [1]

1 Institute of Computer Science, Polish Academy of Sciences, 01-248 Warsaw, Poland
2 Data61, CSIRO, Sydney, NSW 2122, Australia
3 Institute of Computer Science and Computer Mathematics, Jagiellonian University, 30-348 Cracow, Poland
4 School of Computing and Mathematics, Charles Sturt University, Port Macquarie, NSW 2444, Australia
* Correspondence: josef.pieprzyk@gmail.com

Abstract: Source coding has a rich and long history. However, a recent explosion of multimedia Internet applications (such as teleconferencing and video streaming, for instance) renews interest in fast compression that also squeezes out as much redundancy as possible. In 2009 Jarek Duda invented his asymmetric numeral system (ANS). Apart from having a beautiful mathematical structure, it is very efficient and offers compression with a very low coding redundancy. ANS works well for any symbol source statistics, and it has become a preferred compression algorithm in the IT industry. However, designing an ANS instance requires a random selection of its symbol spread function. Consequently, each ANS instance offers compression with a slightly different compression ratio. The paper investigates the compression optimality of ANS. It shows that ANS is optimal for any symbol sources whose probability distribution is described by natural powers of 1/2. We use Markov chains to calculate ANS state probabilities. This allows us to precisely determine the ANS compression rate. We present two algorithms for finding ANS instances with a high compression ratio. The first explores state probability approximations in order to choose ANS instances with better compression ratios. The second algorithm is a probabilistic one. It finds ANS instances whose compression ratios can be made as close to the best ratio as required. This is done at the expense of the number θ of internal random "coin" tosses. The algorithm complexity is $\mathcal{O}(\theta L^3)$, where L is the number of ANS states. The complexity can be reduced to $\mathcal{O}(\theta L \log_2 L)$ if we use a fast matrix inversion. If the algorithm is implemented on a quantum computer, its complexity becomes $\mathcal{O}(\theta(\log_2 L)^3)$.

Keywords: entropy coding; source coding; lossless compression; ANS

check for updates

Citation: Pieprzyk, J.; Duda, J.; Pawłowski, M.; Camtepe, S.; Mahboubi, A.; Morawiecki, P. The Compression Optimality of Asymmetric Numeral Systems. *Entropy* **2023**, *25*, 672. https://doi.org/10.3390/e25040672

Academic Editor: T. Aaron Gulliver

Received: 27 February 2023
Revised: 13 April 2023
Accepted: 15 April 2023
Published: 17 April 2023

1. Introduction

The increasing popularity of working from home has dramatically intensified Internet traffic. To cope with heavy communication traffic, there are essentially two options. The first option involves an upgrade of the Internet network. This is, however, very expensive and not always available. The second option is much cheaper and employs compression of transmitted symbols. It also makes sense as typical multimedia communication is highly redundant. Source coding has a long history, and it can be traced back to Shannon [1] and Huffman [2]. The well-known Huffman code is the first compression algorithm that works very well for symbol sources, whose statistics follow the natural powers of 1/2. Unfortunately, Internet traffic sources almost never have such simple symbol statistics.

Compression algorithms can be divided into two broad categories: lossy and lossless. Lossy compression does not allow users to recover original data but is widely used for multimedia data such as voice, music, video and picture, where a loss of a few least significant bits does not matter. Ahmed et al. [3] have introduced a lossy compression that applies discrete cosine transform (DCT). The algorithm is now used to compress images (JPEG [4] and HEIF formats), video (MPEG and AVC formats) and music (such as MP3 and MP4 [5,6]). In contrast to lossy compression, a lossless one guarantees the recovery of original data at the receiver side. The lossless compression family includes Huffman code [2] and its variants (see [7] for example), arithmetic coding (AC) [8–10], Lempel–Ziv compression and its variants [11–13], prediction by partial matching (PPM) [14], run-length encoding (RLE) [15] and, last but not least, asymmetric numeral systems (ANS) [16]. Recent developments in quantum technology bring us closer to the construction of real-life quantum computers [17]. A natural question is whether or not compression of quantum data is feasible [18,19]. The question has been answered in the affirmative in the work [20], which shows how to compress an ensemble of qubits into exponentially fewer qubits. An implementation of quantum compression on IBM quantum computers is reported in the work [21].

ANS introduced by Duda in [16] offers a very versatile compression tool. It allows the compression of symbols that occur with an arbitrary probability distribution (statistics). ANS is also very fast in both hardware and software. Currently, ANS is the preferred compression algorithm in the IT industry. It has been adopted by Facebook, Apple, Google, Dropbox, Microsoft, and Pixar, to name a few main IT companies (for details see https://en.wikipedia.org/wiki/Asymmetric_numeral_systems, accessed on 13 April 2023). ANS can be seen as a finite state machine (FSM) that starts from an initial state, absorbs symbols from an input source one by one and squeezes out binary encodings. The heart of an ANS algorithm is its *symbol spread function* (or simply *symbol spread* for short) that assigns FSM states to symbols. The assignment is completely arbitrary as long as each symbol s is assigned a number L_s of states such that $p_s \approx L_s/L$, where p_s is probability of the symbol s and $L = 2^R$ is the total number of states (R is a parameter that can be chosen to obtain an acceptable approximation of p_s by L_s/L).

Consequently, there are two closely related problems while designing ANS. The first, called *quantisation*, requires from the designer to approximate a symbol probability distribution $\mathcal{P} = \{p_s | s \in \mathbb{S}\}$ by its approximation $\mathcal{Q} = \{q_s = L_s/L | s \in \mathbb{S}\}$, where $\mathbb{S}$ is the set of all symbols (also called an alphabet). It is expected that ANS implementation for $\mathcal{Q}$ achieves a compression ratio that is as close as possible to the symbol source entropy. The second problem is the selection of a symbol spread for fixed ANS parameters. It turns out that some symbol spreads are better than others. Again, an obvious goal is to choose them in such a way that the average encoding length is as small as possible and close to the symbol source entropy.

Motivation. Designers of ANS have to strike a balance between efficiency and compression quality. The first choice that has to be made is how closely symbol probabilities p_s need to be approximated by $\frac{L_s}{L}$. Clearly, the bigger the number of states ($L = 2^R$), the better approximation and better compression. Unfortunately, a large L slows down compression. It turns out that the selection of a symbol spread has an impact on the quality of compression. For some applications, this impact cannot be ignored. This is true when ANS is applied to build a pseudorandom bit generator. In this case, the required property of ANS is minimal coding redundancy. Despite the growing popularity of ANS, there is no proof of its optimality for arbitrary symbol statistics. This work does not aim to prove optimality, but rather, it aims to develop tools that allow us to compare the compression quality of different ANS instances. It means that we are able to adaptively modify ANS in such a way that every modification provides a compression ratio gain (coding redundancy reduction). The following issues are the main drivers behind this work:

- Investigating symbol quantisation and its impact on compression quality.
- Understanding the impact of a chosen symbol spread on the ANS compression ratio.
- Designing an algorithm that allows us to build ANS instances that maximise compression ratio or equivalently minimises coding redundancy.

Contributions. They are as follows:

- The application of Markov chains to calculate ANS compression ratios. Note that Markov chains have been used in ANS analysis previously. The work [22] applies them for analysis of ANS-based encryption while the paper [23] uses Markov chains to prove the asymptotic optimality of ANS.
- Designing "good" ANS instances whose state probabilities follow the approximation $\log_2 e/x$, where x is an ANS state.

- A randomised algorithm that permits the building of ANS, whose compression ratio is close or alternatively equal to the best possible. The algorithm uses a pseudorandom number generator (PRNG) as a random "coin".
- An improvement of the Duda–Niemiec ANS cryptosystem that selects at random ANS instances with best compression ratios.

The rest of the work is organised as follows. Section 2 puts the work in the context of the known source coding algorithms. Section 3 describes the ANS compression and its algorithms. Section 4 studies the optimality of ANS. Section 5 shows how Markov chains can be used to calculate ANS state equilibrium probabilities and, consequently, average lengths of ANS encodings. Section 6 presents an algorithm that produces ANS instances whose state probabilities follow the approximation $\log_2 e/x$. Section 7 describes an algorithm that permits obtaining the best (or close to it) compression ratio. Section 8 suggests an alternative to the Duda–Niemiec ANS encryption. The alternative encryption called cryptographic ANS allows us to design an ANS secret instance whose compression ratio is close to the best one. Section 9 presents the results of our experiments and finally, Section 10 concludes our work.

2. Arithmetic Coding versus Asymmetric Numeral Systems

Two variants of ANS (tabled tANS and range rANS) are often used as a direct replacement not only for Huffman coding to improve compression ratio but also for arithmetic coding (AC [8–10]) to improve speed. In contrast to Huffman coding (which can handle whole bits only), AC and ANS can process fractions of bits. This requires an additional buffer to store bit fractions and accumulate them until getting a whole bit, which is next released into the output bitstream. AC applies a buffer that stores a range represented by two numbers whose length depends on the probability of encoded symbols. On the other hand, ANS maintains a buffer with a single natural number, whose length reflects both the probability of the encoded symbol and the previous contents of the buffer. In other words, compared to AC, ANS needs a single number only to store the current state.

For both AC and ANS, there are two possible implementation options. The first applies basic arithmetic operations (addition, multiplication and division) performed on large states very close to Shannon entropy. This option has a lower memory cost and is convenient for vectorisation in CPU/GPU architectures. More importantly, it allows for the modification of symbol probabilities on the fly. This option is used in both AC and rANS. The second option translates compression into FSM. It avoids arithmetic operations (especially multiplication) and is recommended for fixed symbol probability distributions. Importantly, it allows for joint compression and encryption. This option is used by quasi-AC, M coder, and tANS.

2.1. Arithmetics: AC versus rANS

A classical AC for binary alphabet (bitwise) uses one multiplication per symbol and two for a large alphabet. In contrast, rANS uses one multiplication per symbol no matter how big the alphabet is. This is true for the rANS decoder, as its encoder requires integer division, which is notoriously expensive on most CPU architectures. However, in many applications, decoding speed is more important than encoding speed. One of the fundamental differences between AC and ANS is encoding and decoding orders. For AC, the orders are the same, while for ANS, decoding is executed in reverse. This means that AC is advantageous for some applications, such as streaming. On the other hand, the ANS reverse decoding order enables methods such as the alias method (https://fgiesen.wordpress.com/2014/02/18/rans-with-static-probability-distributions, accessed on 13 April 2023) and the BB-ANS method of Townsend et al. [24], which are not possible with AC. rANS uses a state determined by a single natural number that makes low-level optimisation and vectorisation possible. In contrast, AC has to keep two numbers that define a state and a current range.

In practice, rANS is used for relatively large alphabets ranging from 2^8 to 2^{16} symbols, whose probability distribution needs a regular update. For static probability distribution, rANS applies alphabets with 256 symbols. It also copes very well with much larger alphabets with, say, 2^{25} symbols [25]. An obvious advantage of a large alphabet used in rANS is a reduction of the required number of steps. Consider two alphabets: one with two and the other with 256 symbols. The number of steps for the larger alphabet can then be reduced by the factor of 8, and there is no need for binary conversion of states. A drawback of rANS with large alphabets is the growing size of tables and an increase in the number of variables. A typical compromise to cope with the dilemma is to deploy byte-oriented arithmetics.

In contrast, AC is usually applied for binary alphabets [26,27]. As we have already pointed out, this creates a significant computational overhead as symbols of large alphabets need to be represented in binary. Fortunately, AC can also work directly with large alphabets. This is referred to as range coding. Note that the implementation of both AC and rANS requires a similar number of operations (with the exception that rANS needs a single multiplication per symbol while AC needs two). However, the main advantage of rANS over AC is a shorter state. A state for AC is determined by two numbers, while a state for rANS consists of a single integer. Moreover, rANS is easier to parallelise due to the fact that the decoder goes through the exact same states as the encoder (albeit in reverse order) [28]. Consequently, rANS low-level implementations can be easily optimised and vectorised (SIMD, GPU).

2.2. FSM Representation: Quasi-AC, M Coders and tANS

It turns out that the most expensive arithmetic operation applied in compression is multiplication. To improve efficiency, the idea is to eliminate multiplication altogether. This can be done for a majority of popular entropy coders with a relatively small number of states. As an entropy coder can be seen as FSM, it is enough to determine its state-transition table. An encoding table accepts a pair (symbol, current state) and outputs (binary encoding, next state). Clearly, there must exist a corresponding decoding table that reverses encoding. Quasi-AC is an example of an FSM representation for AC (https://kuscholarworks.ku.edu/bitstream/handle/1808/7210/HoV93.qtfull.pdf;sequence=1, accessed on 13 April 2023). There is also a very popular 64-state M coder (https://iphome.hhi.de/marpe/mcoder.htm, accessed on 13 April 2023), which is the core of the Context Adaptive Binary Arithmetic Coder (CABAC). The coder is used in MPEG video compressors. As far as we know, all FSM variants of AC are restricted to binary alphabets. Theoretically, larger alphabets can also be converted into their binary equivalents. However, the number of states would be much larger than for ANS as each AC state consists of a pair of numbers (the range). In contrast, thanks to single-number states, ANS can be easily converted into its FSM equivalent for a large alphabet. Table-based ANS (tANS) usually uses from 4 to 8 times more states than the alphabet size. It means that for an alphabet containing 256 symbols, tANS needs 2048 states. This tANS variant is used in a popular FSE implementation of the Zstandard compressor and it rebuilds encoding/decoding tables for every new symbol frame (30 kB in length).

2.3. LIFO Handling and Statistical Modelling

ANS works according to the last-in-first-out (LIFO) principle. This technical difficulty is solved by encoding symbols backwards and storing a final state. Then decoding can proceed forward from the final state. If an initial encoding state is fixed and known to a decoder, then the decoder can verify decompression correctness. This way, we obtain a popular checksum mechanism for free. For simple statistical models (like Markov), we can directly encode backwards and use a context of future-forward decoding. For more sophisticated models (like a popular adaptive update of the probability distribution) we can use an additional buffer to store symbol statistics. An encoder first makes a statistical analysis of incoming symbols in their natural (forward) order. It stores the observed probability distribution of symbols in the buffer. Knowing probability distribution, the encoder can next design an appropriate instance of ANS and share it with the decoder. Finally, the encoder processes symbols backwards. The decoder recovers symbols in their natural order. Note that the decoder needs to know where a binary frame terminates. There are essentially the following two options: in the first, we can fix the length of a binary frame; in the second option, we can use a sentinel value that is chosen from a set of low-probability symbols (that do not occur in the current symbol frame).

2.4. Industry Status

For AC, there are implementations that achieve compression speeds from ∼50 (alphabet with two symbols) to ∼150 MB/s/core (alphabet with 256 symbols). On the other hand, consider ANS: its compression speed ranges from ∼500 (for FSE implementation of tANS) to ∼1500 MB/s/core (for vectorised SIMD rANS) (see https://github.com/jkbonfield/rans_static, https://sites.google.com/site/powturbo/entropy-coder, accessed on 13 April 2023) . For GPU, there are available vectorised rANS implementations (e.g., Facebook or NVidia) reaching speeds of hundreds GB/s (see https://github.com/facebookresearch/dietgpu, https://developer.nvidia.com/blog/latest-releases-and-resources-feb-3-10/#software-releases, accessed on 13 April 2023). To our best knowledge, we are not aware of such fast implementations for AC. There are also available FPGA implementations processing one symbol per cycle [29]. As a consequence, in recent years, ANS has started to dominate the compression market for various applications. Notable examples include compression for:

- General purpose—Facebook Zstandard (tANS) [30] has become probably the most popular replacement of the gzip algorithm (built-in Linux kernel),
- Genetic data—almost the default is CRAM (rANS) citecram, which is a part of the SAMtools library,
- 3D data (meshes, point clouds)—the Google Draco 3D compressor (rANS) used by Pixar,
- Images—JPEG XL (rANS) [31] is recommended by Adobe and often provides the best benchmark performance (https://cloudinary.com/blog/contemplating-codec-comparisons, accessed on 13 April 2023).

The exception is video compression, which is still dominated by AC. This is due to the following two factors: the first one concerns intellectual property and patent issues in relation to the legacy compression algorithms (https://en.wikipedia.org/wiki/Asymmetric_numeral_systems, accessed on 13 April 2023); the second factor is backward encoding required by ANS, which complicates the implementation of compression.

3. Asymmetric Numeral Systems

Here we do not describe the ideas behind ANS design. Instead, we refer the reader to original papers by Duda [16,32] and the ANS Wikipedia page, whose URL address is given in the previous section. A friendly introduction to ANS can be found in the work [22]. The ANS algorithm suite includes initialisation, compression (encoding), and decompression (decoding). Algorithm 1 shows initialisation— it also serves as a reference for basic ANS notation.

A compression algorithm accepts a symbol sequence $\mathbf{s}$ and outputs a bitstream $\mathbf{b}$. In most cases, the probability distribution of symbols is unknown so it has to be calculated. This is done by pushing symbols one by one to a stack and counting their occurrences. When the last symbol s_ℓ is processed, the symbol statistics are known, and compression can start. It means that compression starts from the last symbol s_ℓ. Algorithm 2 describes the ANS compression steps.

```
Algorithm 1: ANS Initialisation
  Input: a symbols source $\mathbb{S}$, its symbol probability distribution $p : \mathbb{S} \to [0,1]$ and a
         parameter $R \in \mathbb{N}^+$.
  Output: instantiation of encoding and decoding functions:
  •  $C(s,x)$ and $k_s(x)$;
  •  $D(x)$ and $k(x)$.
  Steps: proceed as follows:
  •  Calculate the number of states $L = 2^R$;
  •  Determine the set of states $\mathbb{I} = \{L, \ldots, 2L-1\}$;
  •  Compute integer $L_s \approx Lp_s$, where $p_s$ is probability of $s$; $s \in \mathbb{S}$;
  •  Choose symbol spread function $\bar{s} : \mathbb{I} \to \mathbb{S}$, where $\mathbb{L}_s = \{x : \bar{s}(x) = s\}$ denotes
     a set of states assigned to the symbol $s$ and $L_s = |\mathbb{L}_s|$;
  •  Establish coding function $C(s,y) = x$ for $y \in \{L_s, \ldots, 2L_s - 1\}$,
     which assigns states $x \in \mathbb{L}_s$ according to symbol spread function;
  •  Compute $k_s(x) = \lfloor \log_2(x/L_s) \rfloor$ for $x \in \mathbb{I}$ and $s \in \mathbb{S}$. It gives
     the number of output bits per symbol;
  •  Construct decoding function $D(x) = (s,y)$, which for $x \in \mathbb{I}$, assigns
     its unique symbol (given by the symbol spread function) and the
     integer $y$, where $L_s \le y \le 2L_s - 1$. Note that $D(x) = C^{-1}(x)$;
  •  Calculate $k(x) = R - \lfloor \log_2(y) \rfloor$, which determines the number of bits
     that need to be read out from the bitstream;
```

```
Algorithm 2: ANS Encoding
  Input: a symbol sequence $\mathbf{s} = (s_1, s_2, \ldots, s_\ell) \in \mathbb{S}^*$ and an initial state $x = x_\ell \in \mathbb{I}$, where
         $\ell = |\mathbf{s}|$.
  Output: a bitstream $\mathbf{b} = (b_1|b_2|\ldots|b_\ell) \in \{0,1\}^*$, where $|b_i| = k_{s_i}(x_i)$ and $x_i$ is state in $i$-th
          step.
  Steps: for $i = \ell, \ell-1, \ldots, 2, 1$ do
      $s := s_i$;
      $k = k_s(x) = \lfloor \log_2(x/L_s) \rfloor$;
      $b_i = x \mod 2^k$;
      $x := C(s, \lfloor x/2^k \rfloor)$, where / stands for integer division;
  store the final state $x_0 = x$;
```

Note that the bitstream is created by the concatenation of individual encodings. This is denoted by $\mathbf{b} = (b_1|b_2|\ldots|b_\ell)$, where b_i is a binary encoding (sequence of k bits); $i = 1, \ldots, \ell$. Decompression steps are shown in Algorithm 3. Note that $LSB(\mathbf{b})_k$ and $MSB(\mathbf{b})_k$ stand for the k least and most significant bits of $\mathbf{b}$, respectively.

```
Algorithm 3: ANS Decoding
  Input: a bitstream $\mathbf{b} \in \{0,1\}^*$ and the final state $x = x_0 \in \mathbb{I}$.
  Output: symbol sequence $\mathbf{s} \in \mathbb{S}^*$.
  Steps: while $\mathbf{b} \neq \emptyset$ do
      $(s,y) = D(x)$;
      $k = k(x) = R - \lfloor \log_2(y) \rfloor$;
      $b = MSB(\mathbf{b})_k$;
      $\mathbf{b} := LSB(\mathbf{b})_{|\mathbf{b}|-k}$;
      $x := 2^k y + b$;
```

ANS Example

Given a symbol alphabet $\mathbb{S} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, where $p_{\mathtt{a}} = \frac{3}{16}$, $p_{\mathtt{b}} = \frac{5}{16}$, $p_{\mathtt{c}} = \frac{8}{16}$ and the parameter $R = 4$. The number of states is $L = 2^R = 16$ and the state set equals to $\mathbb{I} = \{16, 17, \ldots, 31\}$. A symbol spread $\bar{s} : \mathbb{I} \to \mathbb{S}$ can be chosen at random as long as (1) the number of elements in $\mathbb{L}_i$ equals to L_i and (2) the chosen states in $\mathbb{L}_i$ are in the increasing order. So we assume that

$$\bar{s}(x) = \begin{cases} \texttt{a} & \text{if } x \in \{18, 22, 25\} = \mathbb{L}_{\texttt{a}} \\ \texttt{b} & \text{if } x \in \{19, 20, 23, 26, 28\} = \mathbb{L}_{\texttt{b}} \\ \texttt{c} & \text{if } x \in \{16, 17, 21, 24, 27, 29, 30, 31\} = \mathbb{L}_{\texttt{c}} \end{cases} \tag{1}$$

$$\Updownarrow$$

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
c	c	a	b	b	c	a	b	c	a	b	c	b	c	c	c

where $L_{\texttt{a}} = 3$, $L_{\texttt{b}} = 5$ and $L_{\texttt{c}} = 8$. The encoding table $\mathbb{C}(x_i, s_i) = (x_{i+1}, b_i) \overset{def}{\equiv} \binom{x_{i+1}}{b_i}$ is shown in Table 1.

Table 1. ANS for 16 states (symbol probabilities $\{3/16, 5/16, 8/16\}$).

$s_i \backslash x_i$	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
a	$\binom{22}{00}$	$\binom{22}{01}$	$\binom{22}{10}$	$\binom{22}{11}$	$\binom{25}{00}$	$\binom{25}{01}$	$\binom{25}{10}$	$\binom{25}{11}$	$\binom{18}{000}$	$\binom{18}{001}$	$\binom{18}{010}$	$\binom{18}{011}$	$\binom{18}{100}$	$\binom{18}{101}$	$\binom{18}{110}$	$\binom{18}{111}$
b	$\binom{26}{0}$	$\binom{26}{1}$	$\binom{28}{0}$	$\binom{28}{1}$	$\binom{19}{00}$	$\binom{19}{01}$	$\binom{19}{10}$	$\binom{19}{11}$	$\binom{20}{00}$	$\binom{20}{01}$	$\binom{20}{10}$	$\binom{20}{11}$	$\binom{23}{00}$	$\binom{23}{01}$	$\binom{23}{10}$	$\binom{23}{11}$
c	$\binom{16}{0}$	$\binom{16}{1}$	$\binom{17}{0}$	$\binom{17}{1}$	$\binom{21}{0}$	$\binom{21}{1}$	$\binom{24}{0}$	$\binom{24}{1}$	$\binom{27}{0}$	$\binom{27}{1}$	$\binom{29}{0}$	$\binom{29}{1}$	$\binom{30}{0}$	$\binom{30}{1}$	$\binom{31}{0}$	$\binom{31}{1}$

This example is used throughout the work to illustrate our considerations.

4. Optimality of ANS—Bounds and Limits

There are two main factors that determine how well ANS compresses symbol sequences: (1) compression quality of the underlying ANS encoder and (2) the length of auxiliary data needed by a decoder to recover symbols from bitstream. Auxiliary data include symbol statistics, the ANS spread function, a chosen sentinel value (or the length of bitstream), and the final ANS state. Note that auxiliary data introduces a loss of compression quality. The loss is significant for relatively short symbol sequences, but it becomes negligible for very long files. In this section, our considerations are focused on ANS compression quality only. We ignore auxiliary data.

Given an ANS instance designed for a memoryless symbol alphabet with its probability distribution $p = \{p_s; s \in \mathbb{S}\}$ and the parameter R. The set of all ANS states is $\mathbb{I} = \{2^R, \ldots, 2^{R+1} - 1\}$. ANS is optimal if the average length of the binary encoding is equal to the alphabet entropy or

$$\frac{1}{\ell} \sum_{i=1}^{\ell} |b_i| = H(\mathbb{S}) = \sum_{s \in \mathbb{S}} p_s \log_2 p_s^{-1}, \tag{2}$$

where a sequence $\mathbf{s} = (s_1, s_2, \ldots, s_\ell)$ consists of ℓ symbols. The sequence $\mathbf{s}$ is also called a symbol frame. The same symbols of the frame can be grouped so we get

$$\frac{1}{\ell} \sum_{i=1}^{\ell} |b_i| = \frac{1}{\ell} \sum_{s \in \mathbb{S}} \sum_{s \in \mathbf{s}} |b_s| = \sum_{s \in \mathbb{S}} \frac{\sum_{s \in \mathbf{s}} |b_s|}{\ell}. \tag{3}$$

Note that $\frac{1}{\ell} \sum_{s \in \mathbf{s}} |b_s| = p_s \log_2 p_s^{-1}$. So we have proven the following lemma.

Lemma 1. *Given an alphabet $\mathbb{S}$ whose probability distribution is $p = \{p_s; s \in \mathbb{S}\}$ and ANS for $\mathbb{S}$. Then ANS is optimal if and only if*

$$\frac{1}{\ell} \sum_{s \in \mathbf{s}} |b_s| = p_s \log_2 p_s^{-1} \textit{ for all } s \in \mathbb{S}, \tag{4}$$

where **s** *is a symbol sequence, which repeats every symbol* $\ell \cdot p_s$ *times.*

A caveat—strictly speaking Equation (3) holds when probabilities p_s are natural powers of $1/2$. In other cases, the left-hand side is a rational number while the right-hand side is an irrational one. Consequently, Equation (3) is an approximation determined by the applied calculation precision.

Let us have a closer look at how ANS encodes symbols. According to Algorithm 2, a symbol s is encoded into $b_i = x \pmod{2^k}$, where $k_s = \lfloor \log_2(x/L_s) \rfloor$ and $x \in \{2^R, \ldots, 2^{R+1}-1\}$. The length k_s of encoding depends on the state x. The shortest encoding is when $x = 2^R$ and the longest when $x = 2^{R+1}-1$. It means that

$$\lfloor \log_2(2^R/L_s) \rfloor \leq k_s \leq \lfloor \log_2((2^{R+1}-1)/L_s) \rfloor.$$

As $L_s = 2^R p_s$, we get $\lfloor \log_2 p_s^{-1} \rfloor \leq k_s \leq \lfloor \log_2 p_s^{-1}(2 - \frac{1}{2^R}) \rfloor$. Consider a general case when $2^{-(i+1)} < p_s < 2^{-i}$. Then $k_s \in \{i, i+1\}$. If $p_s = 2^{-i}$, then the inequality

$$\lfloor \log_2 2^i \rfloor \leq k_s \leq \lfloor \log_2 2^i (2 - \frac{1}{2^R}) \rfloor$$

points to a single encoding length $k_s = i$. The above leads us to the following conclusion.

Lemma 2. *Given ANS described by Algorithm 2. Then a symbol s is encoded into a binary string of the length k_s, where*

- $k_s = i$ *if* $p_s = 2^{-i}$,
- $k_s \in \{i, i+1\}$ *if* $2^{-(i+1)} < p_s < 2^{-i}$. *This includes an interesting case when* $2^{-1} < p_s < 2^0$ *with* $k_s \in \{0, 1\}$*—some symbols are encoded into void bits* $\varnothing$.

Consider a general case of a symbol alphabet $\mathbb{S}$ with an arbitrary probability distribution $p = \{p_s; s \in \mathbb{S}\}$ (not necessarily the natural powers of $1/2$). Given an ANS instance designed for the distribution p and the set of states $\mathbb{I} = \{2^R, \ldots, 2^{R+1}-1\}$. We start from an observation that the average length (the expectation) of binary encoding of a single symbol $s \in \mathbb{S}$ into a binary encoding b_s is equal to

$$\mathbb{E}(b_s) = \frac{1}{R} \sum_{x \in \mathbb{I}} p(x) k_s(x) = \frac{1}{R} \sum_{x \in \mathbb{I}} p(x) \lfloor \log_2(x/L_s) \rfloor, \tag{5}$$

where a state probability distribution is $\mathcal{P} = \{p(x); x \in \mathbb{I}\}$. Note that optimal encoding of $s \in \mathbb{S}$ occurs when $|b_s| = \log_2 p_s^{-1}$ and does not (directly) depend on the encoding function C. It, however, has a direct impact on state probabilities. Assume that we have found an ANS instance which is optimal. This means that each symbol is compressed into its binary encodings whose expected length equal to the symbol entropy. This implies that the following system of equations must hold:

$$\frac{1}{R} \sum_{x \in \mathbb{I}} p(x) k_s(x) = \log_2 p_s^{-1} \text{ for } s \in \mathbb{S}. \tag{6}$$

Consequently, we have proven the lemma presented below.

Lemma 3. *Given an ANS instance designed for the symbol probability distribution* $\{p_s; s \in \mathbb{S}\}$ *and the set of states* $\mathbb{I} = \{2^R, \ldots, 2^{R+1}-1\}$. *Then ANS is optimal if its state probabilities satisfy relations given by Equation (6).*

Let us discuss briefly the consequences of Lemma 3.

- In general, ANS allows us to attain close to optimal compression. However, finding an appropriate coding function C seems to be a challenge, especially for a large enough R, where enumeration of all possible C is not possible. There is also a possibility that there is no such C.

- The system given by Equation (6) is underdetermined so its solutions create a linear subspace. To see that this is true, it is enough to observe that for a given symbol, the symbol spread assigns more than one ANS state. Consequently, we have $|\mathbb{S}|$ equations in 2^R variables.
- The work [33] shows how to change the state probabilities so they follow a uniform distribution using a pseudorandom bit generator (PRBG). This interference has been necessary to provide confidentiality and integrity of a compressed stream. Unfortunately, the price to pay is a loss of compression ratio. This approach can be applied here. It would involve a design of PRBG with fine control of probability distribution, which could be a very difficult task. Additionally, running PRBG would be much more expensive than a single call to C.
- An optimistic conclusion is that ANS has no structural weaknesses. Whether or not it attains close to optimal compression ratio depends on the coding function C or alternatively on the symbol spread.

It is important to make clear the difference between the optimal and the best compression ratios. Given an instance of ANS. We can exhaustively run through all possible symbol spreads and identify the one whose compression ratio is the best. This obviously does not guarantee that such ANS is optimal.

5. State Probabilities and Markov Chains

ANS can be looked at as FSM, whose internal state $x \in \mathbb{I}$ changes during compression. The behaviour of ANS states is probabilistic and can be characterised by state probability distribution $\{p(x); x \in \mathbb{I}\}$. For a given symbol $s \in \mathbb{S}$, the average encoding length of s is $\kappa(s) = \sum_{x\in\mathbb{I}} k_s(x)p(x)$, where $k_s(x)$ is the length of an encoding assigned to s when ANS is in the state x (see Algorithm 2). The average length of ANS encodings is $\kappa = \sum_{s\in\mathbb{S}} p_s\kappa(s)$. When we deal with an optimal ANS, then $\kappa = H(\mathbb{S})$, which also means that $\kappa(s) = H(s) = \log_2 p_s^{-1}$ for all $s \in \mathbb{S}$. Typically, compression quality is characterised by a ratio between the length of a symbol sequence and the length of the corresponding bitstream. A better measure from our point of view is coding redundancy, which is defined as $\Delta H = \kappa - H(\mathbb{S})$. The measure has the following advantages:

- Easy identification of an optimal ANS instance when $\Delta H = 0$;
- Quick comparison of two ANS instances—a better ANS has a smaller ΔH;
- Fast calculation of the length of a redundant part of bitstream, which is $\ell \cdot \Delta H$ bits, where ℓ is the number of symbols in the input sequence.

To determine ΔH for an ANS instance, it is necessary to calculate probability distribution $\{p(x); x \in \mathbb{I}\}$. It depends on a (random) selection of symbol spread. Note that ANS state transitions are probabilistic and can be described by a matrix. If one of the matrix eigenvalues is equal to $\lambda = 1$, then the corresponding eigenvector points to an equilibrium probability of the related Markov chain [23,34–36]. Note that the ANS Markov chain is in equilibrium when state probabilities do not change after processing single symbols.

Algorithm 4 shows steps for the construction of a system of linear equations, whose solution gives an equilibrium probability distribution $\mathcal{P}_{eq} = \{p(x); x \in \mathbb{I}\}$. The encoding table $\mathbb{C}(s, x) = x'$ shows transition of a state x to the next state x' while encoding/compressing a symbol s.

```
Algorithm 4: Equilibrium of ANS States
  Input: ANS encoding table $\mathbb{E}(s,x)$ and symbol probability distribution $\{p_s; s \in \mathbb{S}\}$.
  Output: probability distribution $\mathcal{P}_{eq} = \{p(x); x \in \mathbb{I}\}$ of ANS Markov chain equilibrium.
  Steps:
  •  initialise $2^R \times 2^R$ matrix $M$ to all zeros except the main diagonal
     $M[i,i] = -1$ for $i = 0, \dots, 2^R - 1$ and the last row, where $M[2^R - 1, j] = 1$ for $j = 1, \dots, 2^R$;
  •  create a vector $B$ with $2^R$ entries with all zero entries except
     the last entry equal to 1;
  for $x \in \{2^R, \dots, 2^{R+1} - 2\}$ and $s \in \mathbb{S}$ do
      find $\mathbb{E}(s,x) = x'$;
      $M[x' \bmod 2^R, x \bmod 2^R] := M[x' \bmod 2^R, x \bmod 2^R] + p_s$;
  • solve $M \cdot X = B$ using Gaussian elimination;
  • return $X \to \mathcal{P}_{eq} = \{p(x); x \in \mathbb{I}\}$;
```

Example 1. *Consider the ANS instance from Table 1. According to the above algorithm, we obtain a matrix M as follows:*

$$M = \begin{bmatrix}
-\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} \\
0 & 0 & 0 & -1 & \frac{5}{16} & \frac{5}{16} & \frac{5}{16} & \frac{5}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & \frac{5}{16} & \frac{5}{16} & \frac{5}{16} & \frac{5}{16} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{3}{16} & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & \frac{5}{16} & \frac{5}{16} & \frac{5}{16} & \frac{5}{16} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & \frac{3}{16} & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{5}{16} & \frac{5}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{5}{16} & \frac{5}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & -1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

The vector $B = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]$. The equilibrium probabilities $(p(16), \dots, p(31))$ are

$$\left(\frac{367}{4590}, \frac{367}{4590}, \frac{1933}{24480}, \frac{1189}{14688}, \frac{991}{14688}, \frac{991}{14688}, \frac{367}{6120}, \frac{157}{2448}, \frac{1519}{24480}, \frac{1189}{24480}, \frac{367}{7344}, \frac{677}{12240}, \frac{367}{7344}, \frac{1933}{36720}, \frac{157}{3060}, \frac{157}{3060} \right).$$

Now it is easy to calculate both the average encoding length $\kappa = 1.4790168845$ and alphabet entropy $H(\mathbb{S}) = 1.4772170014$. ANS leaves a coding redundancy $\Delta H = \kappa - H(\mathbb{S}) = 0.0017998831$ bits per symbol.

Let us change the symbol spread by swapping states 25 with 28 so we have the following symbol spread

$$\bar{s}(x) = \frac{16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25\ 26\ 27\ 28\ 29\ 30\ 31}{\texttt{c c a b b c a b c b b c a c c c}}.$$

The encoding function $C(s,y)$ is:

$s\backslash y$	3	4	5	6	7	8	9	10	11	12	13	14	15
a	18	22	28	–	–	–	–	–	–	–	–	–	–
b	–	–	19	20	23	25	26	–	–	–	–	–	–
c	–	–	–	–	–	16	17	21	24	27	29	30	31

The encoding table is given in Table 2.

Table 2. ANS for 16 states with state swap (symbol probabilities $\{3/16, 5/16, 8/16\}$).

$s_i \backslash x_i$	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
a	$\binom{22}{00}$	$\binom{22}{01}$	$\binom{22}{10}$	$\binom{22}{11}$	$\binom{28}{00}$	$\binom{28}{01}$	$\binom{28}{10}$	$\binom{28}{11}$	$\binom{18}{000}$	$\binom{18}{001}$	$\binom{18}{010}$	$\binom{18}{011}$	$\binom{18}{100}$	$\binom{18}{101}$	$\binom{18}{110}$	$\binom{18}{111}$
b	$\binom{25}{0}$	$\binom{25}{1}$	$\binom{26}{0}$	$\binom{26}{1}$	$\binom{19}{00}$	$\binom{19}{01}$	$\binom{19}{10}$	$\binom{19}{11}$	$\binom{20}{00}$	$\binom{20}{01}$	$\binom{20}{10}$	$\binom{20}{11}$	$\binom{23}{00}$	$\binom{23}{01}$	$\binom{23}{10}$	$\binom{23}{11}$
c	$\binom{16}{0}$	$\binom{16}{1}$	$\binom{17}{0}$	$\binom{17}{1}$	$\binom{21}{0}$	$\binom{21}{1}$	$\binom{24}{0}$	$\binom{24}{1}$	$\binom{27}{0}$	$\binom{27}{1}$	$\binom{29}{0}$	$\binom{29}{1}$	$\binom{30}{0}$	$\binom{30}{1}$	$\binom{31}{0}$	$\binom{31}{1}$

After running Algorithm 4, we obtain equilibrium probabilities $(p_{16}, \dots, p_{31})$ *as follows:*

$$\left(\frac{3071}{38400}, \frac{3071}{38400}, \frac{8077}{102400}, \frac{4981}{61440}, \frac{4177}{61440}, \frac{4177}{61440}, \frac{3071}{51200}, \frac{65}{1024},\right.$$
$$\left.\frac{6321}{102400}, \frac{3071}{61440}, \frac{3071}{61440}, \frac{17159}{307200}, \frac{4981}{102400}, \frac{5419}{102400}, \frac{13}{256}, \frac{13}{256}\right).$$

The average encoding length $\kappa = 1.4789314193$. *This ANS instance leaves coding redundancy* $\Delta H = \kappa - H(\mathbb{S}) = 0.0017144179$ *bits per symbol. Clearly, it offers better compression than the first variant.*

The following remarks summarise the above discussion:

- The extensive practical experience and asymptotic optimality of ANS (see [23]) justifies the conclusion that ANS offers a very close-to-optimal compression when the parameter R is big enough so $L_s = 2^R p_s$ for all $s \in \mathbb{S}$. In general, however, for arbitrary symbol statistics, ANS tends to leave coding redundancy $\Delta H \neq 0$.
- In some applications where ANS compression is being used heavily (for instance, video/teleconference streaming), it makes sense to optimise ANS so ΔH is as small as possible. Note that the smaller ΔH, the more random the bitstreams. This may increase the security level of systems that use compression (for instance, joint compression and encryption).
- It is possible to find the best ANS instance by an exhaustive search through all distinct symbol spreads. For $L = 2^R$ states, we need to search through $\frac{L!}{\prod_{s \in \mathbb{S}} L_s!}$ ANS instances. This is doable for a relatively small number of states. For ANS in our example, the search space includes $\frac{16!}{8! \cdot 5! \cdot 3!} = 720720$ instances. For a bigger L (say, above 50), an exhaustive search is intractable.

6. Tuning ANS Symbol Spreads

For the compression algorithms such as Zstandard and LZFSE, a typical symbol alphabet contains $n = 256$ elements. To obtain a meaningful approximation of the alphabet statistics, we need a bigger number L of states. A rough evaluation of compression ratio penalty given in [37] tells us that choosing $L = 2n$ incurs a entropy loss of $\Delta H = 0.01$ bits/symbol; if $L = 4n$, then $\Delta H = 0.003$ and if $L = 8n$, then $\Delta H = 0.0006$. This confirms an obvious intuition that the bigger number of states L, the better approximation and, consequently, compression ratio. However, there is a “sweet spot” for L, where its further increase slows compression algorithm but without a noticeable compression ratio gain.

Apart from approximation accuracy of symbol probabilities so $p_s \approx L_s/L$, the symbol spread $\bar{s} : \mathbb{I} \to \mathbb{S}$ has an impact on compression ratio. Intuitively, one would expect that symbol spread is chosen uniformly at random from all $\frac{L!}{\prod_{s \in \mathbb{S}} L_s!}$ possibilities. It is easy to notice that they grow exponentially with L. An important observation is that the probability distribution of ANS states during compression is not uniform. In fact, a state $x \in \mathbb{I}$ occurs with probability that can be approximated as $p(x) \approx \log_2(e)/x$. Note that this is beneficial for a compression ratio, as smaller states (with shorter encodings) are preferred over larger ones (with longer encodings). The natural probability bias of states has an impact on the compression ratio, making some symbol spreads better than the others. Let us take a closer look at how to choose symbol spread so it maintains the natural bias.

Recall that for a given symbol $s \in \mathbb{S}$ and state $x \in \mathbb{I}$, the encoding algorithm (see Algorithm 2) calculates $k = k_s = \lfloor \log_2 \frac{x}{L_s} \rfloor$, extracts k least significant bits of the state as the encoding of s and finds the next state $x' = C(s, \lfloor \frac{x}{2^k} \rfloor)$, where $C(s, y)$ is equivalent to a symbol spread $\bar{s}$ and $x' \in \mathbb{L}_s$. Consider properties of coding function $C(s, y)$ that are used in our discussion.

Fact 1. *Given a symbol s and coding function $C(s, \lfloor \frac{x}{2^k} \rfloor)$. Then the collection of states $\mathbb{I}$ is divided into L_s state intervals $\mathbb{I}_i$, where*

- *Each interval $\mathbb{I}_i$ consists of all consecutive states that share the same value $\lfloor \frac{x}{2^k} \rfloor$ for all $x \in \mathbb{I}_i$,*
- *Coding function assigns the same state $x' = C(s, \lfloor \frac{x}{2^k} \rfloor)$ for $x \in \mathbb{I}_i$ and $x' \in \mathbb{L}_s$,*
- *The cardinality of $\mathbb{I}_i$ is 2^k, where $k = k_s = \lfloor \log_2 \frac{x}{L_s} \rfloor$ and $\mathbb{I} = \cup_{i=1,\ldots,L_s} \mathbb{I}_i$.*

Example 2. *Take into account ANS described in Table 1. For the symbol* `a`*, the collection of states $\mathbb{I} = \{16, \ldots, 31\}$ splits into $L_{\mathtt{a}} = 3$ intervals as follows:*

$\mathbb{I}_1 = \{16, 17, 18, 19\}$	$\mathbb{I}_2 = \{20, 21, 22, 23\}$	$\mathbb{I}_3 = \{24, 25, 26, 27, 28, 29, 30, 31\}$
$C(s, \lfloor \frac{x}{4} \rfloor) = 22$; $x \in \mathbb{I}_1$	$C(s, \lfloor \frac{x}{4} \rfloor) = 25$; $x \in \mathbb{I}_2$	$C(s, \lfloor \frac{x}{8} \rfloor) = 18$; $x \in \mathbb{I}_3$

Note that cardinalities of $\mathbb{I}_i$ are powers of 2 or $|\mathbb{I}_1| = |\mathbb{I}_2| = 4$ and $|\mathbb{I}_3| = 8$. As the encoding function $C(s, \cdot)$ is constant in the interval $\mathbb{I}_i$, it makes sense to use a shorthand $C(s, \mathbb{I}_i)$ instead of $C(s, \lfloor \frac{x}{2^k} \rfloor)$ for $x \in \mathbb{I}_i$.

Assume that we know probabilities $p(x)$ of states of $x \in \mathbb{I}$, then the following conclusion can be derived from Fact 1.

Corollary 1. *Given an ANS instance defined by Algorithms 1–3. Then for each symbol $s \in \mathbb{S}$, state probabilities have to satisfy the following relations:*

$$p_s \sum_{x \in \mathbb{I}_i} p(x) = p(C(s, \mathbb{I}_i)); \quad i = 1, \ldots, L_s. \tag{7}$$

Recall that those state probabilities can be approximated by $p(x) \approx \log_2(e)/x$. As Equation (7) requires summing up probabilities of consecutive states, we need the following fact.

Fact 2. *Given an initial part of the harmonic series $(1 + \frac{1}{2} + \ldots + \frac{1}{r})$, then it can be approximated as shown below*

$$\sum_{i=1}^{r} \frac{1}{i} = 1 + \frac{1}{2} + \ldots + \frac{1}{r} \approx \ln(r) + \gamma, \tag{8}$$

where the constant $\gamma \approx 0.577$. It is easy to obtain an approximation of the series $(\frac{1}{r} + \ldots + \frac{1}{r+\alpha})$, which is $\approx \ln(r + \alpha) - \ln(r - 1) = \ln(\frac{r+\alpha}{r-1})$, where $\alpha \in \mathbb{N}^+$.

Now, we are ready to find out a preferred state for our symbol spread. Let us take into account Equation (7). Using the above established facts and our assumed approximation $p(x) \approx \log_2(e)/x$, the left-hand side of the equation becomes

$$p_s \sum_{x \in \mathbb{I}_i} p(x) = p_s \log_2(e) \sum_{x \in \mathbb{I}_i} \frac{1}{x} = p_s \log_2(e) \ln \frac{r + \alpha - 1}{r - 1},$$

where r is the first state in $\mathbb{I}_i$ and $(r + \alpha - 1)$—the last and $\alpha = 2^k$. As we have assumed that the state $C(s, \mathbb{I}_i) \approx \log_2(e)\frac{1}{y}$, where y points the preferred state that needs to be included into $\mathbb{L}_s$, we obtain

$$p_s \log_2(e) \ln \frac{r + \alpha - 1}{r - 1} = \log_2(e) \frac{1}{y}.$$

This brings us to the following conclusion.

Corollary 2. *Given ANS as defined above, then a preferred state y for $\mathbb{I}_i$ determined for a symbol s is*

$$y = \left(p_s \ln \frac{r+\alpha-1}{r-1}\right)^{-1}, \tag{9}$$

where $\mathbb{I}_i = [r, \ldots, r+\alpha-1]$ and $\lfloor y \rceil$ an integer closest to y and it is added to $\mathbb{L}_s$.

Algorithm 5 shows an algorithm for the calculation of symbol spread with preferred states.

Algorithm 5: Symbol Spread Tuning

```
Input: ANS number of states $L = 2^R$ and symbol probability distribution $\{p_s; s \in \mathbb{S}\}$.
Output: symbol spread $\bar{s}$ determined by $\mathbb{L}_s$ for $s \in \mathbb{S}$.
Steps: initialise $\mathbb{L}_s = \varnothing$ for $s \in \mathbb{S}$;
for $s \in \mathbb{S}$ do
    • Compute $L_s = L \cdot p_s$;
    • Split $\mathbb{I}$ into $\mathbb{I}_i$; where $|\mathbb{I}_i| = \lfloor \log_2 \frac{x}{L_s} \rfloor$, $x \in \mathbb{I}_i$ and
      $i = 1, \ldots L_s$;
    for $i = 1, \ldots, L_s$ do
        • find $y = \left(p_s \ln \frac{r+\alpha-1}{r-1}\right)^{-1}$,
          where $\mathbb{I}_i = [r, \ldots, r+\alpha-1]$;
        • $\mathbb{L}_s := \mathbb{L}_s \cup \lfloor y \rceil$;
• Remove collisions from $\mathbb{L}_s$ so $\mathbb{L}_s \cap \mathbb{L}_{s'} = \varnothing$ for $s \neq s'$;
• Return $\bar{s}$ or equivalently $\mathbb{L}_s$ for $s \in \mathbb{S}$;
```

Let us consider the following example.

Example 3. *Take ANS from Section 3 with $L = 16$ states and three symbols $\mathbb{S} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ that occur with probabilities $3/16$, $5/16$ and $8/16$, respectively. For the symbol* a*, the set of states $\mathbb{I}$ splits into $L_{\mathtt{a}} = 16p_{\mathtt{a}} = 3$ subsets: $\mathbb{I}_1 = \{16, 17, 18, 19\}$, $\mathbb{I}_2 = \{20, 21, 22, 23\}$ and $\mathbb{I}_3 = \{24, \ldots, 31\}$. Let us compute the preferred state y for $\mathbb{I}_1$. It is $y = \left(p_s \ln \frac{r+\alpha-1}{r-1}\right)^{-1} \approx 22.56$, where $r = 16$ and $r + \alpha - 1 = 19$. For $\mathbb{I}_2$, we obtain $y \approx 27.91$. For $\mathbb{I}_3$, we obtain $y \approx 17.86$. After rounding to the closest integers, $\mathbb{L}_{\mathtt{a}} = \{18, 23, 28\}$. In similar way, we calculate $\mathbb{L}_{\mathtt{b}} = \{17, 20, 23, 26, 29\}$ and $\mathbb{L}_{\mathtt{c}} = \{16, 18, 20, 22, 24, 26, 28, 30\}$. Clearly, there are a few collisions, for example, the state 18 belongs to both $\mathbb{L}_{\mathtt{a}}$ and $\mathbb{L}_{\mathtt{c}}$. They need to be removed. We accept $\mathbb{L}_{\mathtt{c}}$. The colliding states in other sets are replaced by their closest neighbours, which are free. This obviously makes sense as neighbour states share similar probabilities. The final symbol spread is as follows*

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
c	b	c	a	c	b	c	b	c	a	c	b	c	b	c	a

We can compute the average encoding length by computing equilibrium probabilities, which for the above symbol spread, is $\kappa = \frac{3619}{2448} \approx 1.4783$. It turns out that this is the best compression ratio, as argued in Section 7. In contrast, the average lengths of symbol spreads in Example 1 are 1.4790 and 1.4789.

Algorithm 5 may produce many symbol spreads with slightly different compression ratios. Preferred positions need to be rounded to the closest integers. Additionally, there may be collisions in sets $\mathbb{L}_s$ ($s \in \mathbb{S}$) that have to be removed. Intuitively, we are searching for a symbol spread that is the best match for preferred positions. In other words, we need to introduce an appropriate distance to measure the match.

Definition 1. *Given ANS with a symbol spread $\mathbb{L}_{s\in\mathbb{S}}$ and a collection of preferred positions $\mathcal{L}_{s\in\mathbb{S}}$ calculated according to Equation (9). Then the distance between them is computed as*

$$d(\mathbb{L}_{s\in\mathbb{S}}, \mathcal{L}_{s\in\mathbb{S}}) = \sum_{s\in\mathbb{S}} \sum_{x\in\mathbb{L}_s} |x-y|, \tag{10}$$

where y is the preferred position that is taken by x.

Finding the best match for a calculated $\mathcal{L}_{s\in\mathbb{S}}$ is equivalent to identification of a symbols spread $\mathbb{L}^*_{s\in\mathbb{S}}$ such that

$$d(\mathbb{L}^*_{s\in\mathbb{S}}, \mathcal{L}_{s\in\mathbb{S}}) = \min_{\mathbb{L}_{s\in\mathbb{S}}} d(\mathbb{L}_{s\in\mathbb{S}}, \mathcal{L}_{s\in\mathbb{S}}), \tag{11}$$

where the symbol spread $\mathbb{L}_{s\in\mathbb{S}}$ runs through all possibilities.

Algorithm 6 illustrates a simple and heuristic algorithm for finding $\mathbb{L}^*_{s\in\mathbb{S}}$.

Algorithm 6: Finding Symbol Spread $\mathbb{L}^*_{s\in\mathbb{S}}$

```
Input: ANS number of states $L = 2^R$, symbol probability distribution $\{p_s; s \in \mathbb{S}\}$ and $\mathcal{L}_{s\in\mathbb{S}}$
Output: symbol spread $\bar{s}$ determined by $\mathbb{L}^*_s$ for $s \in \mathbb{S}$
Steps:
•  Create a table with three rows and $L = 2^R$ columns;
•  Put all consecutive states (i.e., $(L, L+1, \ldots, 2L-1)$ in
   increasing order in the first row;
•  Insert all numbers from $\mathcal{L}_{s\in\mathbb{S}}$ in increasing order in the second row together with their
   symbol labels in the third row;
•  Read out all states from the first row that correspond to appropriate symbol labels (in the
   third row);
•  Return $\bar{s}$ or equivalently $\mathbb{L}^*_s$ for $s \in \mathbb{S}$;
```

Example 4. *Consider again the ANS from Section 3 with $L = 16$ states and three symbols $\mathbb{S} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ that occur with probabilities $3/16$, $5/16$, and $8/16$, respectively. We follow Algorithm 6 and obtain the following table:*

x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
y	16.7	16.9	17.8	17.9	19.9	19.9	21.9	22.5	23.1	23.9	25.5	25.9	27.9	27.9	28.7	29.9
s	b	c	a	c	b	c	c	a	b	c	b	c	a	c	b	c

After the calculation of equilibrium probabilities, we obtain the average encoding length, which is $\kappa = \frac{230755}{156048} \approx 1.4787$.

The following observations are relevant:

- The algorithm for tuning symbol spreads is very inexpensive and can be easily applied for ANS with a large number of states (bigger than 2^{10}). It gives a good compression ratio.
- Equation (9) gives a rational number that needs to be rounded up or down. Additionally, preferred states pointed by it are likely to collide. Algorithm 5 computes a symbol spread, which follows the preferred positions.
- Equation (9) indicates that preferred states are likely to be uniformly distributed over $\mathbb{I}$.
- It seems that finding $\mathbb{L}^*_{s\in\mathbb{S}}$ such that $d(\mathbb{L}^*_{s\in\mathbb{S}}, \mathcal{L}_{s\in\mathbb{S}})$ attains minimum does not guarantee the best compression ratio. However, it results in an ANS instance with a "good" compression ratio. This could be a starting point to continue searching for ANS with a better compression ratio.

7. Optimisation of ANS

7.1. Case Study

Consider a toy instance of ANS with three symbols that occur with probabilities $\{3/16, 5/16, 8/16\}$ and 16 states (i.e., $R = 4$). We have implemented software in PARI/GP (PARI is a free software developed by Henri Cohen. PARI stands for Pascal ARIthmetic and GP—Great Programmable calculator.) that searches through all possible instances of ANS symbol spreads (there are $\frac{16!}{8!\cdot 5!\cdot 3!} = 720,720$ such instances). For each spread, we have calculated equilibrium probabilities for the corresponding Markov chain. This allows us to compute the average length of a symbol (in bits/symbol). The results are shown in Figure 1.

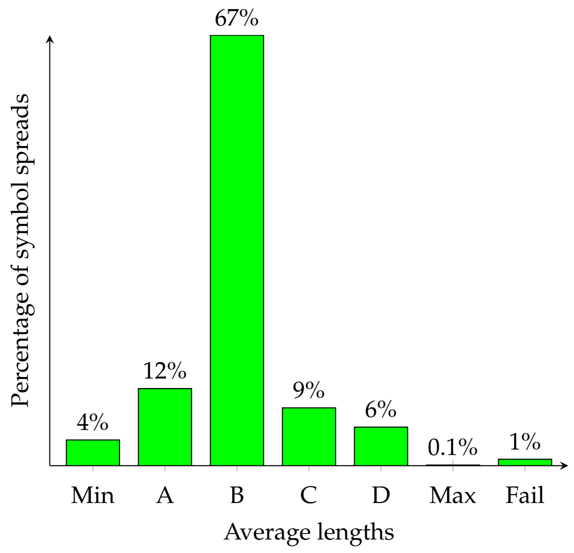

Figure 1. Distribution of average lengths of ANS encodings for the toy ANS. Note that $H(\mathbb{S}) \approx 1.477$ Legend: $A = \langle Min, 1.48]$, $B = \langle 1.48, 1.49]$, $C = \langle 1.49, 1.5]$, $D = \langle 1.5, Max\rangle$, $Min = \frac{3619}{2448} \approx 1.478$ and $Max = \frac{97}{64} \approx 1.515$.

Unfortunately, there is a small fraction of ANS instances whose equilibrium probabilities are impossible to establish as the corresponding system is linearly dependent (its rank is 15). An example of symbol spread with the shortest average encoding length is:

$$\bar{s}(x) = \begin{cases} \mathtt{a} & \text{if } x \in \{16, 24, 25\} = \mathbb{L}_\mathtt{a} \\ \mathtt{b} & \text{if } x \in \{17, 20, 21, 26, 27\} = \mathbb{L}_\mathtt{b} \\ \mathtt{c} & \text{if } x \in \{18, 19, 22, 23, 28, 29, 30, 31\} = \mathbb{L}_\mathtt{c} \end{cases}$$

In contrast, the following symbol spread function has the longest average encoding length:

$$\bar{s}(x) = \begin{cases} \mathtt{a} & \text{if } x \in \{24, 25, 26\} = \mathbb{L}_\mathtt{a} \\ \mathtt{b} & \text{if } x \in \{27, 28, 29, 30, 31\} = \mathbb{L}_\mathtt{b} \\ \mathtt{c} & \text{if } x \in \{16, 17, 18, 19, 20, 21, 22, 23\} = \mathbb{L}_\mathtt{c} \end{cases}$$

Clearly, a careful designer of ANS is likely to use a pseudorandom number generator to select symbol spreads. There is better than $1/2$ probability that such an instance has the average encoding length somewhere in the interval $\langle 1.48, 1.49]$. On the other hand, a careless designer is likely to fall into a trap and choose one of the worst symbol spreads.

7.2. Optimal ANS for Fixed ANS Parameters

The idea is to start from a random symbol spread. Then, we continue swapping pairs of ANS states. After each swap, we calculate the coding redundancy of a new ANS instance. If its redundancy is smaller than the old instance redundancy, then we keep the change. Otherwise, we select a new pair of states for the next swap. Details are given in Algorithm 7.

Algorithm 7: Search for Optimal ANS

```
Input: symbol probability distribution $\{p_s; s \in \mathbb{S}\}$ and parameter $R$ such that $L_s = p_s 2^R$ for
       all $s$
Output: symbol spread $\bar{s}$ or encoding table $\mathbb{E}(s,x)$ of ANS with the smallest coding
        redundancy
Steps:
• Initialise symbol spread $\bar{s}$ using PRBG;
• Determine the corresponding $\mathbb{E}(s,x)$ and calculate
  its redundancy $\Delta H$;
• Assume a required minimum redundancy threshold $T$;
while $\Delta H \geq T$ do
    for $x = 2^R, \ldots, 2^{R+1} - 1$ do
        $y \leftarrow PRBG(\mathbb{I})$, i.e., random selection of state from $\mathbb{I}$;
        if $\bar{s}(x) \neq \bar{s}(y)$ then
            • Calculate equilibrium probabilities for ANS with
              swapped states;
            • Determine average encoding length and
              $\Delta H_{temp}$;
        if $\Delta H_{temp} < \Delta H$ then
            • Update $\bar{s}$ by swapping $x$ and $y$;
• Return $\bar{s}$ and $\Delta H$;
```

We have implemented the algorithm in the PARI/GP environment. Our experiment is executed for the 16-state ANS and three symbols that occur with probabilities $\{3/16, 5/16, 8/16\}$. As the algorithm is probabilistic, its behaviour varies depending on specific coin tosses that determine state swaps. Our starting symbol spread is a spread with the longest average encoding length, i.e., $\{24, 25, 26\}$, $\{27, 28, 29, 30, 31\}$, $\{16, 17, 18, 19, 20, 21, 22, 23\}$). This is the worst case.

The algorithm continues to swap state pairs until the average length equals the minimum $\frac{3619}{2448}$ (the best compression ratio). We have run the algorithm 10^5 times. The results are presented in Table 3. The main efficiency measure is the total number of swaps of ANS state pairs that is required in order to achieve the best compression ratio. Note that each state swap forces the algorithm to redesign ANS. As the algorithm uses PRBG, the number of state swaps varies. The algorithm works very well and successfully achieves the best compression ratio every time. In Table 3, we introduce "good swaps". It means that any good swap produces an ANS instance whose average encoding length gets smaller. This also means that in the worst case, we need only 19 swaps to produce the optimal ANS. Optimality here is understood as the minimum coding redundancy.

Table 3. Number of swaps when Searching for Optimal ANS Instances with 16 states and 3 symbols.

Average #	Min #	Max #	Min # of Good Swaps	Max # of Good Swaps
≈ 24	4	223	4	19

The complexity of Algorithm 7 is $\mathcal{O}(\theta L^3)$, where θ is the number of iterations of the main loop of the algorithm. The most expensive part is the Gaussian elimination needed to find equilibrium probabilities. It takes $\mathcal{O}(L^3)$ steps. We assume that we need θ swaps to obtain a required redundancy with high probability. In other words, the algorithm becomes very expensive when the number of states L is bigger than 2^{10}. This is unfortunate, however, the good news is that a system of linear equations defining equilibrium probabilities is sparse. Consider a simple geometric symbol probability distribution $\{1/2, 1/4, \ldots, 1/1024, 1/1024\}$. Assume that ANS has $L = 2^{10}$ states. A simple calculation indicates that the matrix M in the relation $M \cdot X = B$ for the Markov equilibrium has ≈99% zeros. It means that we can speed up the search for the optimal ANS in the following way:

- Use specialised algorithms for Gaussian elimination that targets sparse systems. There are some mathematical packages (such as MatLab, for example) that include such algorithms;
- Apply inversion of sparse matrices (such as the algorithm from [38], whose complexity is $\mathcal{O}(L^{2.21})$). First we solve $M \cdot X = B$ by computing M^{-1}. This allows us to find X. Now we swap two states that belong to different symbols. Now we need to solve $\tilde{M} \cdot \tilde{X} = B$, where $\tilde{M}$ is a system that describes equilibrium after the swap. We build $\tilde{M}^{-1}$. Now we translate $M \cdot X = B$ by first $I \cdot M \cdot X = B$ and then $\tilde{M}\tilde{M}^{-1}M \cdot X = B$, which produces $\tilde{M} \cdot \tilde{X} = B$, where $\tilde{X} = \tilde{M}^{-1}M \cdot X$. In other words, the solution $\tilde{X}$ can be obtained from X by multiplying it by $\tilde{M}^{-1}M$.

Further efficiency improvements can be achieved by taking a closer look at swapping operations. For the sake of argument, consider a swap of two states and their matrices M and $\tilde{M}$ that describe their Markov chain probabilities before and after the swap, respectively. There are essentially two distinct cases when the swap is:

- Simple, i.e., only two rows of M are affected by the swap. The matrix entries outside the main diagonal are swapped, but entries on the main diagonal need to be handled with care as they do not change if their values are -1. For instance, take ANS from Table 1. Let us swap the states 25 and 26. The rows of M before swap are

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$25 \rightarrow$	0	0	0	0	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	0	-1	0	0	0	0	0	0
$26 \rightarrow$	$\frac{5}{16}$	$\frac{5}{16}$	0	0	0	0	0	0	0	0	-1	0	0	0	0	0

The rows after the state swap are

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$25 \rightarrow$	$\frac{5}{16}$	$\frac{5}{16}$	0	0	0	0	0	0	0	-1	0	0	0	0	0	0
$26 \rightarrow$	0	0	0	0	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	0	0	-1	0	0	0	0	0

The matrix $\tilde{M}$ after swap can be obtained by $\tilde{M} = M\Delta$, where Δ is a sparse matrix that translates M into $\tilde{M}$. As argued above, the equilibrium solution is $\tilde{X} = \Delta^{-1}X$, where $\Delta^{-1} = \tilde{M}^{-1}M$. We still need to calculate $\tilde{M}^{-1}$; however, it is possible to recycle a part of the computations performed while computing M^{-1}. This is possible as M and $\tilde{M}$ share the same entries (except the two rows that correspond to the state swap),

- Complex, i.e., more than two rows need to be modified. This occurs when a swap causes a cascade of swaps that are needed to restore the increasing order in their respective $\mathbb{L}_s$. For example, consider ANS from Table 1 and swap the states 22 and 26.

$$\begin{array}{l} \mathbb{L}_1 = \{18,22,25\} \\ \mathbb{L}_2 = \{19,20,23,26,28\} \end{array}$$
$$\big\downarrow 22 \leftrightarrow 26$$
$$\boxed{\begin{array}{l} \{18,26,25\} \\ \{19,20,23,22,28\} \\ \textit{invalid ANS} \end{array}}$$
$$\Big\downarrow \begin{array}{l} 25 \leftrightarrow 26 \\ 22 \leftrightarrow 23 \end{array}$$
$$\begin{array}{l} \{18,25,26\} \\ \{19,20,22,23,28\} \end{array}$$

To obtain a valid ANS instance, we need two extra swaps. The calculation of $\tilde{M}^{-1}$ can still be supported by the computations for M^{-1} but the matrices M and $\tilde{M}$ differ on more than two rows.

7.3. Optimalisation with Quantisation

So far we have assumed that $L_s = p_s \cdot L$; $s \in \mathbb{S}$ or at least L_s/L is a very close approximation of p_s. However, this is not always true. In practice, there are two issues that need to be dealt with.

- The first is the fact that $p_s \cdot L$ may have two "good" approximations when $\alpha_s < p_s L < (\alpha_s + 1)$, where $\alpha_s \in \mathbb{N}^+$. So, we can choose $L_s \in \{\alpha_s, \alpha_s + 1\}$. This occurs more frequently when the number L is relatively small.
- The second issue happens when there is a tail of symbols whose probabilities are small enough so $p_s \cdot L < 1$. It means that symbols have to be assigned to single states $L_s = 1$. This is equivalent to an artificial increase of symbol probabilities of the tail to $1/L$. Note that $\sum_{s\in\mathbb{S}} p_s = 1$, which is equivalent to $\sum_{s\in\mathbb{S}} L_s = L$. Consequently, the number of states in the symbol spread for other symbols has to be reduced.

The research question in hand (also called quantisation) is defined as follows. Given that a symbol probability distribution $\mathcal{S} = \{p_s | s \in \mathbb{S}\}$, we have to ask: How do we design ANS so its compression is optimal (or close to it), where ANS is built for the symbol probability distribution $\mathcal{Q} = \{q_s | s \in \mathbb{S}\}$, where $\mathcal{Q}$ approximates $\mathcal{S}$, where $L_s = q_s \cdot L \in \mathbb{N}^+$, $q_s \approx p_s$ and $n = |\mathbb{S}|$?

Let us consider the following algorithms.

- Exhaustive search through all possible quantised ANS, where an ANS instance is determined by a selection of $L_s \leftarrow \{\alpha_s, \alpha_s + 1\}$, where $s \in \mathbb{S}$. For each selection, we run Algorithm 7. Finally, we choose the ANS instance with the best compression ratio. Unfortunately, the complexity of the algorithm is exponential or more precisely, $\mathcal{O}(2^n \theta L^3)$. A possible tail of t symbols reduces complexity as all t symbols are assigned a single state and the next t low probability symbols are assigned α_s states. Higher $L_s = \alpha_s + 1$ are ignored in order to compensate states that have been taken by the tail symbols. Note also that some of the selections for L_s must be rejected if $\sum_{s\in\mathbb{S}} L_s \neq L$.
- Best-fit quantised ANS. The idea is to start from symbols from the tail, where L_s has to be 1. Symbols with high probabilities are assigned $L_s = \alpha_s + 1$, while symbols with low probabilities $L_s = \alpha_s$. Symbols with mid-range probabilities are randomly assigned $L_s \in \{\alpha_s, \alpha_s + 1\}$. The intuition behind the algorithm is the fact that selection of higher L_s reduces the average length of encodings and vice versa. Now we can run Algorithm 7 to find the optimal (or close to it) ANS.

To recap our discussion, we make the following remarks:

- Finding ANS with the optimal compression ratio is of prime concern to anyone who would like to either maximise communication bandwidth or remove as much redundancy as possible from bitstreams. We model the asymptotic behaviour of ANS by Markov chains. The calculated equilibrium probabilities allow us to precisely determine the average length of binary encodings and, consequently, the ANS compression ratio.
- Search for the best ANS can be done in two steps. (1) We run Algorithm either 5 or 6 that tunes symbol spread using an approximation of state probabilities. The algorithm is very efficient, and its complexity is $\mathcal{O}(L)$. Unfortunately, it does not guarantee that the calculated ANS instance is optimal/best. (2) Next, we execute Algorithm 7. Its initial symbol spread has been calculated in the previous step. The randomised algorithm usually attains the best (or close to it) ANS in practice (see experiments in Section 9).
- Algorithm 7 can be sped up by using a specialised sparse matrix inversion algorithm together with reusing computations from previous inversions. This allows us to find a close-to best ANS for the number L of states in the range $[2^{10}, 2^{12}]$. The range is the most used in practice.
- The number of iterations θ contributes to the complexity of Algorithm 7. In general, the bigger θ, the higher the probability that the algorithm produces the best ANS. In fact, $\theta = \alpha L$ is enough to obtain ANS with a close to the best compression ratio with high probability, where α is a small integer (say 2 or 3) —see our experiments in Section 9.

8. Cryptographic ANS

Duda and Niemiec [39] have proposed a randomised ANS, where its symbols spread is chosen at random. To make it practical, the authors suggest replacing truly random coin tosses with a pseudorandom number generator (PRNG), which is controlled by a relatively short random seed/key. The two communicating parties can agree on a common secret key K. Both sender and receiver use it to select their symbol spread using PRNG controlled by K. The sender can build an appropriate encoding table, while the receiver—the matching decoding table. Consequently, the parties can use compression as encryption. Although such encryption does not provide a high degree of protection (especially against integrity and statistics attacks—see [22,33]), it could be used effectively in low-security applications. The price to pay is a complete lack of control over the compression ratio of ANS. This weakness can be mitigated by applying our Algorithm 7.

Note that the symbol spread $\{L_s|p_s \in \mathbb{S}\}$ is public but $\{L_s^{best}|p_s \in \mathbb{S}\}$ is secret as to reconstruct it, an adversary needs to recover K and execute Algorithm 7. The cryptographic ANS is illustrated in Table 4. Unlike the Duda–Niemiec ANS, it achieves a close to the best compression ratio. However, the effective security level (the length of the cryptographic key) is determined by the number of ANS instances produced by Algorithm 7. For instance, the ANS from Figure 1 guarantees 15-bit security (or $\log_2 30240 \approx 15$). For ANS with a large number of states, it is difficult to determine a precise security level. However, this may be acceptable for low-security applications.

Table 4. Cryptographic ANS.

Sender		**Receiver**
Secret K Public $\mathcal{P} = \{p_s|s \in \mathbb{S}\}$ • Run Algorithm 6 $\rightarrow \{L_s|s \in \mathbb{S}\}$ • Run Algorithm 7 with PRNG(K) $\rightarrow \{L_s^{best}|s \in \mathbb{S}\}$ • Design encoding table for $\{L_s^{best}|s \in \mathbb{S}\}$	$\longrightarrow$	Secret K Public $\mathcal{P} = \{p_s|s \in \mathbb{S}\}$ • Run Algorithm 6 $\rightarrow \{L_s|s \in \mathbb{S}\}$ • Run Algorithm 7 with PRNG(K) $\rightarrow \{L_s^{best}|s \in \mathbb{S}\}$ • Build decoding table for $\{L_s^{best}|s \in \mathbb{S}\}$

9. Experiments

Algorithm 7 has been implemented using the Go language and has been executed on a MacBook Pro with an M1 chip. The algorithm has been slightly modified so it finds both the lower and upper bounds for ΔH. The lower bound points to ANS, which is close to the best. In contrast, the upper bound shows ANS, whose coding redundancy ΔH is big (close to the worst case). Note that a random selection of spreads produces ANS instances whose ΔH fall somewhere between the bounds. The following results have been obtained for 10^5 iterations of the `FOR` loop.

# ANS States	128	256	512	1024
ΔH_{min}	1.5770×10^{-5}	4.1869×10^{-6}	1.0470×10^{-6}	2.7868×10^{-7}
ΔH_{max}	0.0346	0.0298	0.0348	0.0306
# Spreads with ΔH_{min}	22	68	138	156
# Spreads with ΔH_{max}	149	357	701	1044
Search Time for ΔH_{min}	1 m 12 s	8 m 30 s	1 h 11 m 37 s	9 h 2 m 59 s
Search Time for ΔH_{max}	1 m 23 s	8 m 13 s	1 h 9 m 58 s	8 h 47 m 5 s

We have increased the number of iterations of the `FOR` loop to n^2, where n is the number of states for $n = \{512, 1024\}$. The results are presented below.

n	512	1024
# Iterations	262,144	1,048,576
# Good Swaps	508	792
ΔH_{min}	$1.0222607274 \times 10^{-6}$	2.5842×10^{-7}
ΔH_{min} after additional rounds	$1.0222607271 \times 10^{-6}$	2.5842×10^{-7}
# Additional Rounds	10^6	10^5
Better Spreads Found in Additional Rounds	8	0
Execution Time	872 m	6115 m

We see that due to the probabilistic nature of the Algorithm 7, even after a large number of iterations, there is a non-zero chance of finding a spread with lower ΔH. In practise, there are time restrictions imposed on the time needed for the execution of Algorithm 7. The following results illustrate how much time is needed between two consecutive good swaps that improve ΔH. The number of iterations of the `FOR` loop is 10^5.

# ANS States	128	256	512	1024
Tunning Time	54 ms	235 ms	300 ms	374 ms
Optimisation Time	1 m 12 s	8 m 30 s	1h 11 m 37	9 h 2 m 59 s
# Good Swaps	22	68	138	156
Time between Two Good Swaps	3 s	7 s	31 s	209 s
Average ΔH Gain per Good Swap	3×10^{-7}	2.2×10^{-8}	2.4×10^{-9}	3.7×10^{-10}
ΔH_{min}	1.5×10^{-5}	4.1×10^{-6}	1.04×10^{-6}	2.78×10^{-7}

Note that matrix inversion consititues the main computational overhead of Algorithm 7. The experiments presented above have applied a standard Gaussian elimination (GE) for matrix inversion, whose complexity is $\mathcal{O}(L^3)$. Algorithm 7 can be sped up by (1) using a more efficient algorithm for sparse matrix inversion (SMI) and (2) recycling computations from previous matrix inversions. The table below gives the complexity of Algorithm 7 for different matrix inversion algorithms and for the classical and quantum computers.

Classical Computer			Quantum Computer
GE	SMI [38]	SMI [40]	GE [41]
$\mathcal{O}(\theta L^3)$	$\mathcal{O}(\theta L^{2.21})$	$\mathcal{O}(\theta L \log_2 L)$	$\mathcal{O}(\theta(\log_2 L)^3)$

The experiments have confirmed that Algorithm 7 works well and is practical for $L < 128$. However, for a larger L, it grows slower and quickly becomes impractical. Optimisation of Algorithm 7 is beyond the scope of this work, and it is left for our possible future investigations. Note that the algorithm becomes very fast when it uses quantum matrix inversion.

We have taken the Calgary data corpus (see http://links.uwaterloo.ca/calgary.corpus.html, accessed on 13 April 2023) and have compressed its files using ANS instances obtained from Algorithm 7. Table 5 illustrates the results obtained.

Table 5. Compression results for Calgary corpus.

File Name	Original Size in bytes	Compressed Size (Random Spread)	Compressed Size (Optimised Spread)
bib	111,261	77,932	76,790
book1	768,771	458,920	440,678
book2	610,856	384,861	370,693
geo	102,400	69721	68,648
news	377,109	255,586	248,842
obj1	21,504	14,828	14,579
obj2	246,814	172,622	169,043
paper1	53,161	41,120	40,283
paper2	82,199	55,736	53,842
paper3	46,526	33,800	33,104
paper4	13,286	9948	9766
paper5	11,954	8954	8785
paper6	38,105	25,849	25,053
pic	513,216	120,041	115,319
progc	39,611	28,456	28,028
progl	71,646	46,669	44,905
progp	49,379	37,381	36,806
trans	93,695	74,192	73,107

The third column shows the sizes of files compressed using an instance of ANS with a random spread. The fourth column gives sizes of the corpus files when compressed with an instance ANS with optimised symbol spread. On average, we are getting a roughly 2.4% improvement in compression rates.

Discussion

The main goal of compression is to reduce the redundancy of a file by encoding more frequent symbols into shorter binary strings and less frequent symbols into longer ones. Typically, for compression to work, it is necessary to describe the symbol statistics by its symbol probabilities. In this case, a file can be treated as a sequence of independent and identically distributed (i.i.d.) random variables, which correspond to the occurrence of single symbols. Clearly, such single-symbol statistics do not reflect all existing probabilistic characteristics of the file. Consequently, even the best (lossless) compression algorithm is not able to squeeze the average length of a single symbol beyond the symbol alphabet entropy. To achieve a better compression ratio, it is necessary to model file statistics by considering N-symbol alphabets, where $N = 2, 3, \ldots$ (see [42]). Clearly, the higher N, the better approximation of real statistics of the file and, consequently, better the compression rate. But the price to pay is a significant (exponential) increase in the compression complexity. We would like to emphasise that given a fixed symbol alphabet, both AC and ANS allow us to achieve a compression ratio close to the alphabet entropy. The main difference is processing speed which allows ANS to compress files for better file statistics (for N-symbol alphabets with higher N). Let us compare CPU implementations of AC and ANS. AC can reach speed of $\approx$200 MB/s/core while ANS achieves a speed of $\approx$2000 MB/s/core. For GPU implementations, ANS can be run 100 times faster than AC. This implies that, assuming the same computing resources, ANS provides better compression ratios compared to AC. This is due to the fact that ANS is faster and can apply more complex statistics for N-symbol alphabets (a higher N).

10. Conclusions

The work addresses an important practical problem of compression quality of the ANS algorithm. In the description of ANS, its symbol spread can be chosen at random. Each symbol spread has its own characteristic probability distribution of ANS states. Knowing the distribution, it is possible to compute the ANS compression ratio or, alternatively, its coding redundancy ΔH.

We present two algorithms that allow a user to choose symbol spreads that minimise ΔH. Algorithm 5 determines an ANS instance (its symbol spread) whose state probabilities follow the natural ANS state bias. It is fast even for $L > 2^{12}$, but unfortunately, it does not provide the minimal ΔH. Algorithm 7 provides a solution. It is able to find minimal ΔH with a probability that depends on the number of random coin tosses θ.

We have conducted an experiment for $L = 16$ that shows the behaviour of the average length of ANS encodings. Further experiments have confirmed that matrix inversion creates a bottleneck in Algorithm 7 and makes it impractical for a large L. An immediate remedy is the application of specialised algorithms for sparse matrix inversion, together with recycling computations from previous matrix inversions. Development of a fast version of Algorithm 7 is left as a part of our future research.

The main research challenge is, however, how to construct ANS instances in such a way that their minimum coding redundancy is guaranteed by design. It means that we have to understand the interplay between symbol spreads and their equilibrium probabilities. As ANS is also FSM, it can be visualised as a random graph whose structure is determined by symbol spread. This brings us to an interesting link between ANS and random graphs [43].

Author Contributions: Conceptualisation and methodology J.P. and J.D.; software implementation and validation M.P.; analysis and investigation S.C., A.M. and P.M.; original draft preparation and review and editing all coauthors. All authors have read and agreed to the published version of the manuscript.

Funding: Josef Pieprzyk, Marcin Pawłowski and Paweł Morawiecki have been supported by Polish National Science Center (NCN) grant 2018/31/B/ST6/03003.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Draft of this paper is available at https://arxiv.org/pdf/2209.02228.pdf, accessed on 13 April 2023.

Acknowledgments: We would like to thank the Reviewers for their critical comments, which have significantly improved the contents and presentation of the work. We also thank Monika Pieprzyk for a thorough proofreading of our paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shannon, C.E. A mathematical theory of communication. *Bell Sys. Tech. J.* **1948**, *27*, 623–656. [CrossRef]
2. Huffman, D. A Method for the Construction of Minimum-Redundancy Codes. *Proc. IRE* **1952**, *40*, 1098–1101. [CrossRef]
3. Ahmed, N.; Natarajan, T.; Rao, K. Discrete Cosine Transform. *IEEE Trans. Comput.* **1974**, *C-23*, 90–93. [CrossRef]
4. Hudson, G.; Léger, A.; Niss, B.; Sebestyén, I.; Vaaben, J. JPEG-1 standard 25 years: Past, present, and future reasons for a success. *J. Electron. Imaging* **2018**, *27*, 1. [CrossRef]
5. Britanak, V. On Properties, Relations, and Simplified Implementation of Filter Banks in the Dolby Digital (Plus) AC-3 Audio Coding Standards. *IEEE Trans. Audio Speech Lang. Process.* **2011**, *19*, 1231–1241. [CrossRef]
6. Ehmer, R.H. Masking by Tones vs Noise Bands. *J. Acoust. Soc. Am.* **1959**, *31*, 1253–1256. [CrossRef]
7. Kochanek, J.; Lansky, J.; Uzel, P.; Zemlicka, M. The new statistical compression method: Multistream compression. In Proceedings of the 2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT), Ostrava, Czech Republic, 4–6 August 2008; pp. 320–325. [CrossRef]
8. Langdon, G.; Rissanen, J. A simple general binary source code (Corresp.). *IEEE Trans. Inf. Theory* **1982**, *28*, 800–803. [CrossRef]
9. Langdon, G.G. An Introduction to Arithmetic Coding. *IBM J. Res. Dev.* **1984**, *28*, 135–149. [CrossRef]
10. Rissanen, J. Generalized Kraft Inequality and Arithmetic Coding. *IBM J. Res. Dev.* **1976**, *20*, 198–203. [CrossRef]
11. Storer, J.A.; Szymanski, T.G. Data compression via textual substitution. *J. ACM* **1982**, *29*, 928–951. [CrossRef]
12. Welch. A Technique for High-Performance Data Compression. *Computer* **1984**, *17*, 8–19. [CrossRef]
13. Ziv, J.; Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory* **1978**, *24*, 530–536. [CrossRef]
14. Cleary, J.; Witten, I. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Trans. Commun.* **1984**, *32*, 396–402. [CrossRef]
15. Robinson, A.; Cherry, C. Results of a prototype television bandwidth compression scheme. *Proc. IEEE* **1967**, *55*, 356–364. [CrossRef]
16. Duda, J. Asymmetric Numeral Systems. *arXiv* **2009**, arXiv:0902.0271.

17. Grumbling, E.; Horowitz, M. (Eds.) *Quantum Computing: Progress and Prospects*; The National Academies Press: Washington, DC, USA, 2019. [CrossRef]
18. Petz, D. Quantum Compression. In *Theoretical and Mathematical Physics*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 83–90. [CrossRef]
19. Zhang, Q.; Lai, H.; Pieprzyk, J.; Pan, L. An improved quantum network communication model based on compressed tensor network states. *Quantum Inf. Process.* **2022**, *21*, 253. [CrossRef]
20. Rozema, L.A. Quantum Data Compression of a Qubit Ensemble. *Phys. Rev. Lett.* **2014**, *113*, 160504. [CrossRef]
21. Pivoluska, M.; Plesch, M. Implementation of quantum compression on IBM quantum computers. *Sci. Rep.* **2022**, *12*, 5841. [CrossRef]
22. Camtepe, S.; Duda, J.; Mahboubi, A.; Morawiecki, P.; Nepal, S.; Pawłowski, M.; Pieprzyk, J. ANS-based Compression and Encryption with 128-bit Security. *Int. J. Inf. Secur.* **2022**, *21*, 1051–1067. [CrossRef]
23. Dube, D.; Yokoo, H. Fast Construction of Almost Optimal Symbol Distributions for Asymmetric Numeral Systems. In Proceedings of the 2019 IEEE International Symposium on Information Theory (ISIT), Paris, France, 7–12 July 2019; IEEE: New York, NY, USA, 2019. [CrossRef]
24. Townsend, J.; Bird, T.; Barber, D. Practical Lossless Compression with Latent Variables Using Bits Back Coding 2019. Available online: http://xxx.lanl.gov/abs/1901.04866 (accessed on 13 April 2023).
25. Lettrich, M. Fast and Efficient Entropy Compression of ALICE Data using ANS Coding. In *EPJ Web of Conferences*; EDP Sciences: Les Ulis, France, 2020; Volume 245, p. 01001.
26. Ko, H.H. Enhanced Binary MQ Arithmetic Coder with Look-Up Table. *Information* **2021**, *12*, 143. [CrossRef]
27. 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. [CrossRef]
28. Giesen, F. Interleaved Entropy Coders. 2014. Available online: http://xxx.lanl.gov/abs/1402.3392 (accessed on 13 April 2023).
29. Najmabadi, S.M.; Tran, T.H.; Eissa, S.; Tungal, H.S.; Simon, S. An architecture for asymmetric numeral systems entropy decoder-a comparison with a canonical Huffman decoder. *J. Signal Process. Syst.* **2019**, *91*, 805–817. [CrossRef]
30. Collet, Y.; Kucherawy, M. Zstandard Compression and the 'application/zstd' Media Type. RFC 8878. 2021. Available online: https://www.rfc-editor.org/info/rfc8878 (accessed on 13 April 2023).
31. Alakuijala, J.; Van Asseldonk, R.; Boukortt, S.; Bruse, M.; Comșa, I.M.; Firsching, M.; Fischbacher, T.; Kliuchnikov, E.; Gomez, S.; Obryk, R.; et al. JPEG XL next-generation image compression architecture and coding tools. In Proceedings of the Applications of Digital Image Processing XLII, San Diego, CA, USA, 12–15 August 2019; Volume 11137, pp. 112–124.
32. Duda, J. Asymmetric numeral systems as close to capacity low state entropy coders. *CoRR* **2013**. Available online: http://xxx.lanl.gov/abs/1311.2540 (accessed on 13 April 2023).
33. Camtepe, S.; Duda, J.; Mahboubi, A.; Morawiecki, P.; Nepal, S.; Pawłowski, M.; Pieprzyk, J. Compcrypt—Lightweight ANS-Based Compression and Encryption. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 3859–3873. [CrossRef]
34. Brémaud, P. *Markov Chains*; Springer International Publishing: Berlin/Heidelberg, Germany, 2020. [CrossRef]
35. Seabrook, E.; Wiskott, L. A Tutorial on the Spectral Theory of Markov Chains 2022. Available online: http://xxx.lanl.gov/abs/2207.02296 (accessed on 13 April 2023).
36. Haggstrom, O. *Finite Markov Chains and Algorithmic Applications*; London Mathematical Society: London, UK, 2002.
37. Duda, J. Encoding of Probability Distributions for Asymmetric Numeral Systems. *CoRR* **2021**. Available online: http://xxx.lanl.gov/abs/2106.06438 (accessed on 13 April 2023).
38. Casacuberta, S.; Kyng, R. Faster Sparse Matrix Inversion and Rank Computation in Finite Fields. *arXiv* **2021**, arXiv:2106.09830v1. Available online: http://xxx.lanl.gov/abs/2106.09830 (accessed on 13 April 2023).
39. Duda, J.; Niemiec, M. Lightweight compression with encryption based on Asymmetric Numeral Systems. *arXiv* **2016**, arXiv:1612.04662.
40. Hsieh, C.J.; Sustik, M.A.; Dhillon, I.S.; Ravikumar, P. Sparse Inverse Covariance Matrix Estimation Using Quadratic Approximation. 2013. Available online: https://arxiv.org/pdf/1306.3212.pdf (accessed on 13 April 2023).
41. Harrow, A.W.; Hassidim, A.; Lloyd, S. Quantum algorithm for solving linear systems of equations. *Phys. Rev. Lett.* **2009**, *15*, 150502. [CrossRef]
42. Pieprzyk, J.; Hardjono, T.; Seberry, J. *Fundamentals of Computer Security*; Springer: Berlin/Heidelberg, Germany, 2003. [CrossRef]
43. Bollobás, B. *Random Graphs*; Cambridge University Press: Cambridge, UK, 2001. [CrossRef]