Efficient Inverted Index Compression Algorithm Characterized by Faster Decompression Compared with the Golomb-Rice Algorithm

Andrzej Chmielowiec 1,* and Pawel Litwin 2

1 The Faculty of Mechanics and Technology, Rzeszow University of Technology, Kwiatkowskiego 4, 37-450 Stalowa Wola, Poland
2 The Faculty of Mechanical Engineering and Aeronautics, Rzeszow University of Technology, Powstańców Warszawy 8, 35-959 Rzeszow, Poland; pilitwin@prz.edu.pl
* Correspondence: achmie@prz.edu.pl

Abstract: This article deals with compression of binary sequences with a given number of ones, which can also be considered as a list of indexes of a given length. The first part of the article shows that the entropy $H$ of random $n$-element binary sequences with exactly $k$ elements equal one satisfies the inequalities $k \log_2(0.48 \cdot n/k) < H < k \log_2(2.72 \cdot n/k)$. Based on this result, we propose a simple coding using fixed length words. Its main application is the compression of random binary sequences with a large disproportion between the number of zeros and the number of ones. Importantly, the proposed solution allows for a much faster decompression compared with the Golomb-Rice coding with a relatively small decrease in the efficiency of compression. The proposed algorithm can be particularly useful for database applications for which the speed of decompression is much more important than the degree of index list compression.

Keywords: inverted index compression; Golomb-Rice coding; runs coding; sparse binary sequence compression

1. Introduction

Our research is aimed at solving the problem of efficient storage and processing of data from production processes. Since the introduction of control charts by Shewart [1,2], quality control systems have undergone a continuous improvement. As a result of the ubiquitous computerization of production processes, enterprises today collect enormous amounts of data from various types of sensors. With annual production reaching tens or even hundreds of millions of items, this implies the need to store and process huge amounts of data. Modern factories with such high production capacity are highly automated, thus being equipped with a substantial number of sensors. Today, the number of sensors installed in a production plant usually ranges between $10^3$ (a medium sized factory), and $10^5$ (a very large factory). We can, thus, describe the state of a factory at time $t$ as a sequence of bits $S_t = (s_{1,t}, s_{2,t}, \ldots, s_{m,t})$, where the bit $s_{i,t}$ indicates if the production process in a given place proceeds correctly. By noting down states $S_t$ at times $t_1 < t_2 < \cdots < t_r$, we obtain a matrix of binary states:

$$(s_{i,t_j}) = \begin{pmatrix}
  s_{1,t_1} & s_{2,t_1} & \cdots & s_{m,t_1} \\
  s_{1,t_2} & s_{2,t_2} & \cdots & s_{m,t_2} \\
  \vdots & \vdots & & \vdots \\
  s_{1,t_r} & s_{2,t_r} & \cdots & s_{m,t_r}
\end{pmatrix},$$

which illustrates the state of production processes in a given time period. The data from the matrix of measurements can be analyzed, e.g., in order to detect the relationship between
the settings of production processes and the number of defective products [3]. It should be noted, however, that, by saving sensor readings every 0.1 s, a factory collects from $2.5 \cdot 10^{10}$ to $2.5 \cdot 10^{12}$ bits of new data monthly ($3 \div 300$ (GB)). Such large monthly data increases require the application of appropriate compression methods enabling quick access to information stored in a database. Note that this type of data is characterized by a significant disproportion between the number of zeros and ones. For example, if 0 means no process defects/no failure, and 1 denotes a defect/a failure, then a typical production process database usually stores binary sequences with a large number of zeros and a small number of ones. In practice, a need arises to store a list of indexes containing the numbers of detectors and times for which specific deviations have been observed. Such databases naturally contain lists (or matrices) of indexes comprising information about various types of abnormalities. This is why inverted index methods, which today are widely used in all kinds of text document search engines [4–6], provide a workable solution to the problem of efficient storage of such databases. In its simplest form, the database comprises only zeros (OK) and ones (FAIL), which means that there is basically only one word to index (FAIL). However, we can easily imagine that the database also stores error types, in which case it is necessary to index many words—one word for every error code. Such a database can effectively store information about mechanical failures, overheating, thermal damage, product defects, downtime, etc. This is why the inverted index method proves useful for storing data on various failures and irregularities. In addition, it must be stressed that the data stored in the database will not be modified. This means that the focus must be on maximizing the degree of compression, while ensuring high speed data search (decompression). These two postulates underlie the construction of the algorithm presented in this article.

Currently there are three main approaches to inverted indexing compression.

1. Methods for compressing single indicators, including: Shannon-Fano coding [7,8], Huffman coding [9], Golomb-Rice coding [10,11], Gamma and Delta coding [12], Fibonacci coding [13], (S,C)-dense coding [14], and Zeta coding [15].

2. Methods for compressing entire lists, including Elias-Fano method [16,17], interpolation methods [18], Simple-9, Relative-10, and Carryover-12 [19].

3. Methods for compressing many lists simultaneously, which mainly adapt and increase the potential of the methods listed above.

The article by Pibiri and Venturini [20] gives an extensive review of inverted index compression algorithms. Some publications deal with efficient decompression and compare the speed of individual algorithms in various environments [19,21,22]. As mentioned earlier, our research looks into the problem of compressing databases storing binary information about the status of manufactured items or even the entire production plant. The vast majority of this kind of data can be represented by binary sequences containing a small number of ones and a large number of zeros. When solving such problems, it is often assumed that the resulting sequences are random sequences. The problem of coding sparse binary sequences is often raised in the literature, for example in the book by Solomon [23]. One of the first algorithms was proposed by Golomb [10], who introduced quotient and remainder coding for the distance between successive occurrences of ones. Then, Gallager and van Voorhis [24] derived a relationship that allows for optimal selection of compression parameters for Golomb codes. Somasundaram and Domnic [25] proposed an extended Golomb code for integer representation. The method proposed by Reference [26,27] and developed, among others, by Fenwick [28] is a special case of Golomb coding. Since these two types of coding are interrelated, they are often referred to as Golomb-Rice codes. The work on the appropriate selection of parameters for these codes was later continued, among others, by Robinson [29] and Kiely [30]. On the other hand, Fraenkel and Klein [31] proposed completely different methods for coding sparse binary sequences by combining Huffman coding with the new numeral systems. Another algorithm that can be considered as useful for coding these types of sequences is the prefix coding proposed by Salomon [32]. In turn, Tanaka and Leon-Garcia [33] developed efficient coding of the distance between
ones when the probability of their occurrence varies. It is also worth mentioning the work by Ferragina and Venturini [34], who proposed a new method of creating word indexes.

The works of Trotman [21] and Zang [35] were the motivation for choosing Golomb-Rice as the reference algorithm. Trotman [21] compares set sizes and decompression times using the Golomb-Rice, Elias gamma, Elias delta, Variable Byte, and Binary Interpolative Coding algorithms. These compression techniques were studied in the context of accessing the Wall Street Journal collection. The Golomb-Rice algorithm is reported as a second in the compression ratio category (Binary Interpolative Coding is slightly better), and the lowest compression ratio is achieved by Variable Byte. The shortest decompression time was obtained using Variable Byte Coding. The Golomb-Rice algorithm ranks second in the decompression time category. It is worth it to note that Binary Interpolative Coding in this category is clearly weaker than the other methods. In the work of Zang [35], the size of sets and the decompression speed of the Golomb-Rice, Variable Byte, Simple9, Simple16, and PForDelta algorithms were analyzed. The task concerned decompression of inverted index files in search engines. In the analysis, the Golomb-Rice algorithm obtains the smallest sizes of the compressed sets, but, in the case of decompression time, it is weaker than the competition (only Variable Byte achieves longer decompression times).

When we consider the problem of storing and processing large data sets, e.g., data collected in enterprises operating according to the Industry 4.0 framework, both the size of the saved files and the access time are important. The Golomb-Rice algorithm is characterized by a high compression ratio and a high decompression speed, lower only than the algorithms that use vector instructions available in modern processors. However, such instructions are not implemented in all devices of the industrial Internet of Things solutions.

In the next parts of the article we present: a theorem on the entropy of random binary sequences with a given number of zeros and ones, a new algorithm for coding sparse binary sequences, and an analysis of the results of the implemented algorithm. We decided to compare the algorithm proposed in the article with an algorithm using Golomb-Rice coding and the DEFLATE algorithm implemented in the ZLIB library, which was developed based on the foundations laid by Lempel and Ziv [36,37]. At this point, it should be stressed that the main point of reference of the presented results is the Golomb-Rice algorithm, which proved to be most efficient as far as compression is concerned but leaves much to be desired in terms of speed.

2. Materials and Methods

At the beginning of this section, we analyze the entropy of $n$-element binary sequences consisting of exactly $k$ ones and $n - k$ zeros. Let $X$ be a random variable defined in space:

$$\Omega = \{ x : x = (x_1, \ldots, x_n), x_1 + \cdots + x_n = k, x_i \in \{0, 1\} \},$$

for which all events are equally probable. Consequently, for every $x \in \Omega$, the following holds:

$$P(X = x) = P_{n,k} = \frac{k!(n-k)!}{n!} = \frac{k!}{(n-k+1) \cdots (n+1)n},$$

which, directly from the definition of entropy, leads to:

$$H(X) = -\log_2(P_{n,k}).$$

Later in the article, we will show that the asymptotic entropy of a random variable $X$ can be expressed by $H(X) = \Theta(k \log_2\left(\frac{n}{k}\right))$. This is of great practical importance because it means that such sequences cannot be encoded more efficiently than with $c \cdot k \log_2\left(\frac{n}{k}\right)$ bits, where $c$ is a constant. Looking at the entropy $H(X)$ from the perspective of the encoding of natural numbers, we can conclude that it is on the same level as the entropy of $k$ numbers not greater than $n/k$. This observation refers to a real situation where $k$ ones are randomly distributed in an $n$-element sequence, while the average distances between them are $n/k$. 
This very observation forms the basis for the compression algorithm presented later in the article. The algorithm is based on the encoding of the distance between adjacent ones on words of a fixed width equal approximately to \( \log_2 \left( \frac{n}{k} \right) \). At this point, we must strongly emphasize that the case analyzed in the article is fundamentally different from a situation where a random variable \( Y \) equals 1 with probability \( p = \frac{k}{n} \). Entropy is given by the well-known expression \( H(Y) = -p \log_2(p) - (1 - p) \log_2(1 - p) \), while the \( n \)-element sequences generated using this random variable do not have a fixed number of ones equal to \( k \). A very accurate estimate of the value of \( H(Y) \) was proposed by Mascioni [38]. His inequalities could also be used to estimate the entropy \( H(X) \), but, unfortunately, they lead to much less transparent and non-intuitive relationships. This is why, in this section, we present our own original estimate of the value of \( H(X) \).

The value \( H(X) = -\log_2 \left( \frac{P_{n,k}}{n} \right) \) is the number of bits of information carried by the survey. Unfortunately, this formula is not an intuitive reflection of the actual size of the coded information. That is why it is our goal to express \( H(X) \) using more intuitive formulas.

An element that we will use is the Stirling’s formula given by Robbins [39]:

**Theorem 1** (Stirling’s formula). If \( n \) is a natural number, then:

\[
 n! = \sqrt{2\pi n} \left( \frac{n}{e} \right)^n e^{\lambda_n},
\]

where \( \frac{1}{12n+1} < \lambda_n < \frac{1}{12n} \).

Having the above in mind, we are ready to prove the main theorem about the entropy of \( n \)-element binary sequences with exactly \( k \) elements other than zero.

**Theorem 2.** If \( X \) is a random variable with domain:

\[
 \Omega = \{ x : x = (x_1, \ldots, x_n), x_1 + \cdots + x_n = k, x_i \in \{0, 1\} \},
\]

\( 1 \leq k < \frac{n}{2} \) and probability \( P(X = x) \) is the same for every sequence, and then entropy \( H(X) \) satisfies the following inequalities:

\[
 k \log_2 \left( \frac{0.48 \cdot n}{k} \right) < H(X) < k \log_2 \left( \frac{2.72 \cdot n}{k} \right).
\]

**Proof.** The proof consists of two parts. First, it is shown that the entropy \( H(X) \) is bounded from above:

\[
 H(X) = \log_2 \left( \frac{n - k + 1}{k!} \cdots \frac{n - 1}{k!} \right)
 < \log_2 \frac{n^k}{k!}
 < \log_2 \left( n^k \left( \frac{e}{k} \right)^k \cdot e^{-\frac{1}{12n+1}} \cdot (2\pi k)^{-\frac{1}{2}} \right)
 = k \log_2 \left( \frac{en}{k} \cdot e^{-\frac{1}{12n+1}} \cdot (2\pi k)^{-\frac{1}{2}} \right).
\]

The property of function \( e^x \) shows that \( e^{-\frac{1}{12n+1}} < 1 \), while, based on elementary facts about the limits of numerical sequences, we know that \( (2\pi k)^{-\frac{1}{2}} < 1 \). Consequently, it follows that:

\[
 H(X) < k \log_2 \left( \frac{en}{k} \right) < k \log_2 \left( \frac{2.72 \cdot n}{k} \right).
\]
In the second part of the proof, we show that the entropy $H(X)$ is bounded from below:

$$H(X) = \log_2 \frac{(n-k+1) \cdots (n-1)n}{k!}$$

$$> \log_2 \frac{(n-k+1)^k}{k!}$$

$$> \log_2 \left( \frac{e(n-k+1)^k}{k} \cdot e^{-\frac{k}{2\pi}} \cdot (2\pi k)^{-\frac{1}{2}} \right)$$

$$= k \log_2 \left( \frac{e(n-k+1)^k}{k} \cdot e^{-\frac{1}{12k^2}} \cdot (2\pi k)^{-\frac{1}{2}} \right).$$

For $k < \frac{n}{2}$ and $k \geq 1$, we have $\frac{e(n-k+1)}{k} > \frac{en}{2k}$. The property of function $e^x$ shows that $e^{-\frac{1}{12k^2}} > e^{-\frac{1}{12}} > 0.92$, while, based on elementary facts about the limits of numerical sequences, it follows that $(2\pi k)^{-\frac{1}{2}} > 0.39$. By combining all the above inequalities, we finally obtain:

$$H(X) > k \log_2 \left( \frac{0.48 \cdot n}{k} \right).$$

This completes the proof. □

It is worth noting how accurate the obtained estimates are. Figure 1 shows a complete graph of the entropy $H(X)$ and its bounds arising from Theorem 2. A graphical interpretation of these relationships clearly shows that the lower bound strongly deviates from the value of entropy, whereas the graph of the upper bound almost converges with the graph of entropy for a fairly large range of $k$ values. Figure 2 shows graphs for $k$ bound by $\frac{1}{10}n$. Therefore, we can expect that a compression algorithm developed based on estimate (2) will have greater compression efficiency for small $k$ values. This matches our expectations, as the degree in which a database of a production process is filled with various errors and failures should be relatively low (low $k/n$ value).

Figure 1. Graph of entropy and its upper and lower bounds in the function of the number of ones for a sequence consisting of $10^4$ elements for $k \in [0, 10^4]$. 
Figure 2. Graph of entropy and its upper and lower bounds in the function of the number of ones for a sequence consisting of $10^4$ elements for $k \in [0, 10^3]$.

The estimate obtained in Theorem 2 also allows for a good approximation of the value of Newton’s binomial coefficients. Our observations show that the entropy can be well expressed by formula $H(X) \approx k \log_2 \left( \frac{2.72 \cdot n}{k} \right)$. Based on the fact that $H(X) = -\log_2(P_{n,k})$, we obtain the following approximation of binomial coefficients for small $k$ values:

$$\binom{n}{k} = P_{n,k} = e^{-H(X)} \approx e^{k \log_2 \left( \frac{2.72 \cdot n}{k} \right)}.$$  

Although the approximation is not as good as that proposed by Mascioni [38], the formula is so compact and elegant that it seems worthy of attention.

Underlying the new algorithm is the Golomb-Rice coding [10, 26, 27], which is still used in many applications that require efficient data compression. It is vital for modern image coding methods [40, 41], video compression [42, 43], transmission coding in wireless sensor networks [44], compression of data from exponentially distributed sources [45], Gaussian distributed sources [46], and high variability sources [47]. Moreover, it is used for compressing data structures, such as R-trees [48] and sparse matrices [49]. The universality and versatility of Golomb-Rice codes results from the fact that with appropriate selection of parameters the level of compression is very close to the level of entropy. Figure 5 shows how close the entropy curve and the Golomb-Rice coding compression curve are to each other. In fact, the only defect of Golomb-Rice coding is the variable length of the codeword, which, in the case of software applications, negatively affects the time of compression and decompression. Bearing in mind the applications mentioned in the introduction, we decided to propose a new algorithm with a compression level, similar to that offered by Golomb-Rice coding, but with faster decompression.

Let us consider a binary sequence compressed with the use of Golomb-Rice coding with module $m = 2^2 = 4$: $0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0$. (4)

For this particular sequence, it is necessary to encode a sequence of integers $(5, 0, 5, 2, 4)$ and transform it under the Golomb-Rice method into the following sequence of bits:

$$\begin{array}{c|c|c} 
1 & 0 & 01 \\
5 & 0 & 00 \\
1 & 0 & 01 \\
5 & 0 & 10 \\
1 & 0 & 00 \\
4 & & 
\end{array}.$$  

Vertical bars | in the code above mark places in which it will be necessary to execute a conditional instruction. We will call them codeword divisions, while their number will be called the number of codewords. The general Golomb-Rice compression process is presented as Algorithm 1. Whenever used in the algorithm, $(m)_{2,n}$ denotes that the number $m$ is coded in the binary system using exactly $n$ bits. It should be noted that the optimal length
of the suffix is strictly dependent on the fraction \( k/n \). The best estimate of this value was provided by Gallager and Voorhis [24]. In steps 4–5, the algorithm determines the length of the string of zeros to be encoded. Then, in steps 6–7, parameters for the codewords are determined. In step 8, the algorithm creates \( q_i + 1 \) single-bit codewords and one \( w \)-bit codeword for the suffix.

Algorithm 1: Golomb-Rice compression of a binary sequence.

**Data:** Number of bits of the suffix codeword denoted by \( w \), sequence \( x = (x_1, \ldots, x_n) \) such that \( x_i \in \{0, 1\} \) and \( \sum x_i = k \).

**Result:** Sequence \( t \) containing the coding of sequence \( x \).

1. \( x_{n+1} \leftarrow 1; \)
2. \( s_0 \leftarrow 0; \)
3. for \( i = 1, 2, \ldots, k + 1 \) do
   4. /* Determine index of the next one */
      \( s_i \leftarrow \min \{ s : \sum_{j=1}^{s} x_j = i \}; \)
   5. /* Determine zeros gap size */
      \( d_i \leftarrow s_i - s_{i-1} - 1; \)
   6. /* Determine number of leading ones */
      \( q_i \leftarrow \lfloor d_i / 2^w \rfloor; \)
   7. /* Determine suffix value */
      \( r_i \leftarrow d_i \mod 2^w; \)
   8. /* Append codewords to the sequence \( t \) */
      \( t \leftarrow t \cup (2^{q_i+1} - 2)_2 q_i + 1 \mid (r_i)_{2^w}; \)
4. end

For the sake of completeness, Algorithm 2, in which the task is to decompress the Golomb-Rice coding, is also presented. Both Algorithm 1 and Algorithm 2 were used in the implementation designed for the comparative analysis of both methods.

Algorithm 2: Golomb-Rice decompression (zero series length encoding).

**Data:** Number of bits of the suffix codeword denoted by \( w \), binary sequence \( t = (t_1, \ldots, t_m) \) such that \( t_i \in \{0, 1\} \) containing the coding of some sequence.

**Result:** Sequence of zero series length \((d_1, d_2, \ldots, d_{k+1})\).

1. \( j \leftarrow 1; \)
2. \( d_1 \leftarrow 0; \)
3. for \( i = 1, 2, \ldots, m \) do
   4. if \( t_i = 1 \) then
      5. /* Decode single bit codeword */
         \( d_j \leftarrow d_j + 2^w; \)
   7. else
      8. /* Decode suffix codeword with \( w \) bits */
         \( d_j \leftarrow d_j + (t_{i+1}2^{w-1} + t_{i+2}2^{w-2} + \cdots + t_{i+w}); \)
   10. \( i \leftarrow i + w + 1; \)
   11. \( j \leftarrow j + 1; \)
   12. \( d_j \leftarrow 0; \)
5. end
4. end

A large number of codewords negatively impacts on software performance. Unfortunately, the use of variable codeword length and unary coding implies that many more conditional instructions will have to be executed in Golomb-Rice coding compared to fixed
Entropy length codes. For example, if we use 3-bit fixed length words to encode sequence (4), we will obtain the following sequence:

\[
\begin{array}{c|c|c|c|c}
101 & 000 & 010 & 100 \\
5 & 3 & 2 & 4
\end{array}
\]

There are far fewer places in the sequence in which it is necessary to execute a conditional instruction. Now, we will use the theorem proved in the previous section to develop a new binary sequence compression algorithm.

Theorem 2 clearly shows that the entropy of an \( n \)-element binary sequence containing \( k \) ones equals \( \Theta(k \log_2(\frac{n}{k})) \). This formula gives some intuition about potential coding with its binary representation being of similar size to the entropy. Expression \( k \log_2(\frac{n}{k}) \) suggests that \( k \) numbers, each of them of order \( \frac{n}{k} \), should be enough to code the sequence. But \( k \) is the number of ones in the sequence, and \( \frac{n}{k} \) is the average distance between them. Therefore, coding the distances between ones seems to be the most natural way of coding this type of sequences. This observation forms the basis for a new compression algorithm presented further in this section. The proving of Theorem 2 was crucial to making this observation. In the next section, we will analyze the properties of the proposed algorithm and set out the conditions under which the proposed method performs better than the standard algorithms of lossless compression.

We define the algorithm for compression sparse binary sequences (AC-SBS) as a parameterized algorithm with the parameter being the bit size of the codeword. The analysis of the algorithm will show that, for a given number of ones in a sequence, there exists exactly one codeword length minimizing the size of the compressed sequence. However, this property will not be analyzed in this section and we will present a version of the algorithm allowing for the use of any codeword length for every bit sequence. We will use the following example to illustrate the compression algorithm. Let us assume that we want to compress sequence (4) using a 2-bit codeword. To start with, let us note that a 2-bit word allows us to code four symbols. Since we need a way to code any distance between adjacent ones, one symbol will have a special meaning. Therefore, we propose the following method of representing distance:

- distance of 0 is coded as \( c(0) = (0)_2 = 00 \),
- distance of 1 is coded as \( c(1) = (1)_2 = 01 \),
- distance of 2 is coded as \( c(2) = (2)_2 = 10 \),
- distance of \( m > 2 \) is coded as \( l = \lfloor \frac{m}{3} \rfloor \) symbols \( (3)_2 = 11 \) ended with a symbol coding the number \( m - 3l \).

Moreover, in order to precisely mark the end of a sequence, we will always add one at the end, which will then be removed once the decoding is complete. Consequently, the following sequence:

\[
00001100001001000 \quad \tilde{1}
\]

is coded as:

\[
\begin{array}{c|c|c|c|c}
11 & 10 & 00 & 11 & 10 \\
5 & 3 & 0 & 5 & 2
\end{array}
\]

This simple example illustrates the mechanism of the algorithm developed based on the analysis of entropy of random binary sequences. Now, we are ready to provide a formal description of the compression algorithm in a general case.

Let us compress an \( n \)-element binary sequence consisting of exactly \( k \) ones. In addition, we will assume that \( w \)-bit words are used for coding. Whenever used in the algorithm, \( (m)_{2,w} \) denotes that the number \( m \) is coded in the binary system using exactly \( w \) bits (similarly to the example presented above, we allow the occurrence of leading zeros). Under these assumptions, the compression algorithm is presented in Algorithm 3.
Algorithm 3: AC-SBS compression of a binary sequence.

**Data:** Number of bits of the codeword denoted by \( w \), sequence \( x = (x_1, \ldots, x_n) \) such that \( x_j \in \{0,1\} \) and \( \sum x_j = k \).

**Result:** Sequence \( t \) consisting of \( w \)-bit words containing the coding of sequence \( x \).

\[
\begin{align*}
1 & \quad x_{n+1} \leftarrow 1; \\
2 & \quad s_0 \leftarrow 0; \\
3 & \quad \text{for } i = 1,2,\ldots,k+1 \text{ do} \\
4 & \quad \quad \quad /* \text{Determine index of the next one} */ \\
5 & \quad \quad \quad s_i \leftarrow \min\{s : \sum_{j=1}^{s} x_j = i\}; \\
6 & \quad \quad \quad /* \text{Determine zeros gap size} */ \\
7 & \quad \quad \quad d_i \leftarrow s_i - s_{i-1} - 1; \\
8 & \quad \quad \quad \text{while } 2^w - 2 < d_i \text{ do} \\
9 & \quad \quad \quad \quad /* \text{Append codeword to the sequence } t */ \\
10 & \quad \quad \quad \quad t \leftarrow t \mid (2^w - 1)_{2,w}; \\
11 & \quad \quad \quad \quad /* \text{Decrease zeros gap size} */ \\
12 & \quad \quad \quad \quad d_i \leftarrow d_i - (2^w - 1); \\
13 & \quad \quad \quad \text{end} \\
14 & \quad \quad \text{end} \\
15 & \quad \text{end}
\end{align*}
\]

At the first step of the algorithm, we add one at the end of the sequence to be compressed. Next, in step 4, we set the number \( s_i \), being in fact an index of the \( i \)-th one. In step 5, we set the number of zeros between adjacent ones. The number \( d_i \) denotes the number of zeros between ones with index \( i - 1 \) and \( i \). Then, in a 6–9 loop, we perform the main part of the algorithm, consisting of coding the number of zeros \( \{d_i\} \).

Algorithm 4, on the other hand, is an AC-SBS decompression algorithm. This version of the algorithm was used in the comparative implementation. Like Algorithm 2, this one also returns the recovered lengths of a series of zeros. This is due to the fact that such a representation is much more useful for further work on the recovered data.

Algorithm 4: AC-SBS decompression (zero series length encoding).

**Data:** Number of bits of the codeword length denoted by \( w \), binary sequence \( l = (l_1, \ldots, l_m) \) such that \( l_i \in \{0,1\} \) containing the coding of some sequence.

**Result:** Sequence of zero series length \( (d_1, d_2, \ldots, d_{k+1}) \).

\[
\begin{align*}
1 & \quad j \leftarrow 1; \\
2 & \quad d_1 \leftarrow 0; \\
3 & \quad \text{for } i = 1,2,\ldots,m \text{ do} \\
4 & \quad \quad d \leftarrow (t_{i+1}2^{w-1} + t_{i+2}2^{w-2} + \ldots + t_{i+w}); \\
5 & \quad \quad d_j \leftarrow d_j + d; \\
6 & \quad \quad \text{if } d \neq 2^w - 1 \text{ then} \\
7 & \quad \quad \quad i \leftarrow i + w; \\
8 & \quad \quad \quad j \leftarrow j + 1; \\
9 & \quad \quad \quad d_j \leftarrow 0; \\
10 & \quad \text{end}
\end{align*}
\]

Algorithm 3 requires the length of the codeword to be used. Figure 3 shows the results of the statistical dependence of the optimal codeword length on the \( \log_{62}(k/n) \) value. The regres-
sion line determined on the basis of the simulation has the form \( l(k/n) = A \cdot \log_2(k/n) + B \), where \( A = -1.07 \), \( B = 0.98 \), and leads to the following formula:

\[
\frac{k}{n} = \left[ -1.07 \cdot \log_2 \left( \frac{k}{n} \right) + 0.98 + \frac{1}{2} \right].
\]

(6)

The presented formula is practical, but perhaps it can be simplified and derived based on theoretical assumptions. As can be seen, the determination of \( w(k/n) \) values requires the knowledge of \( k \) and \( n \), and, more precisely, their \( k/n \) ratio. For a given string of bits, these values can be simply computed by looking at the string. We can also predict the \( k/n \) value based on a sample of the analyzed data.

Figure 3. Graph of dependence of the optimal codeword length in the algorithm for compression of sparse binary sequences (AC-SBS) algorithm on the value of \( \log_2(k/n) \).

As mentioned earlier, the number of codewords to be read is a measure of the complexity of a decompression algorithm. In this case, the complexity \( T_{GR} \) of the Golomb-Rice and \( T_{AC} \) of the AC-SBS decompression algorithm can be approximated by the following formulas:

\[
T_{GR}(k, n) = k \sum_{l=0}^{n-k} \left( 1 - \frac{k}{n} \right)^l \left( \frac{k}{n} \right)^l \left( w_{GR} + 1 + \left\lfloor \frac{l}{2^{w_{GR}}} \right\rfloor \right),
\]

(7)

\[
T_{AC}(k, n) = k \sum_{l=0}^{n-k} \left( 1 - \frac{k}{n} \right)^l \left( \frac{k}{n} \right)^l \left( w_{AC} + \left\lceil \frac{l}{2^{w_{AC} - 1}} \right\rceil \right),
\]

(8)

where \( w_{GR} = w_{GR}(k/n) \) is optimal suffix length in Golomb-Rice algorithm, and \( w_{AC} = w_{AC}(k/n) \) is optimal codeword length in AC-SBS algorithm. Due to the complexity of both formulas, it was decided that the exact number of codewords needed to encode a given binary sequence would be determined by the numerical experiment. The results of the research are presented below.

In order to make an initial comparison of Golomb-Rice coding with AC-SBS coding, we analyzed the ratio of the number of codewords of both algorithms for various values of the \( k \) parameter. Figure 4 presents a graph of the ratio of the number of Golomb-Rice codewords to the number of AC-SBS codewords. The analysis was made for every \( k \in \{ 100, 200, 300, \ldots, 4 \cdot 10^3 \} \) for 100 10^5-bit random sequences with the highest possible compression for every analyzed sequence (the optimal value was adopted for \( w_{GR} \) and \( w_{AC} \)). The result clearly shows that the number of codewords in the case of AC-SBS is from three to two times lower compared with Golomb-Rice. The characteristic jumps in the graph are closely related to the change in the codeword length caused by the changing \( k/n \) ratio (the higher the ratio, the shorter the codewords). When the length \( w_{AC} \) of the codeword in the AC-SBS algorithm increases, then we see a sharp increase in the graph. The sharp drops are caused by the change
in suffix length $w_{GR}$ in the Golomb-Rice coding. In the next section, we will show correlation between the number of codewords and the rate of decompression of both algorithms.

Figure 4. Average ratio of the number of Golomb-Rice codewords to the number of AC-SBS codewords.

3. Results

A comparative analysis of the AC-SBS algorithm was performed using two different hardware architectures. The first one was x86-64 with the AMD FX-6100 processor clocked at 1.573 GHz used for testing. The second was the ARM architecture with the Broadcom BCM2837 Cortex-A53 processor clocked at 1.200 GHz used for testing. In testing, we used our original implementation of the AC-SBS algorithm and the Golomb-Rice algorithm. The implementation has been made available in the GitHub repository at https://github.com/achmie/acsbs. All programs were compiled with the gcc compiler (version 9.3.0) with the -O2 optimization option turned on. In addition, we compared the obtained results with one of the fastest and most commonly used general compression libraries—ZLIB. The algorithms were selected to show the profound difference in compression efficiency and performance rate in the case when the compressed binary sequences have a large disproportion between the number of ones and zeros. Another advantage of this approach is that it shows that the efficiency of our original implementation does not differ a lot from the commonly used software.

The proposed compression algorithm was tested by means of compressing sparse binary sequences. The sequence consisted of $10^6$ elements. We generated random sequences with number of ones ($k$) ranging from 1 to $2.5 \cdot 10^5$. For every $k$, $10^3$ random binary sequences were generated which were then compressed. The length of the compression word ranged from 2 to 12 bits. The purpose of our analysis was to estimate the size of the compressed sequence, as well as its decompression time. The results were then compared with the sequence size and decompression time obtained using the DEFLATE algorithm implemented in the ZLIB library (further referred to as the zlib-deflate algorithm) and Golomb-Rice coding. Moreover, the size of the compressed sequences was also compared with the theoretical level of entropy for a given type of sequence. Below, we will discuss the results and present graphs of sequence sizes and decompression times.

The first stage of our analysis is to compare the size of sequences compressed using the AC-SBS algorithm with sequences compressed using zlib-deflate, Golomb-Rice coding, and sequence entropy. The results shown in Figure 5 present an estimate of the sequence size for $k/n$ from 0-0.25.

The graph presented in Figure 5 clearly indicates that the largest size are sequences compressed using the popular zlib-deflate algorithm. On the other hand, the size of the sequences obtained by AC-SBS compression is slightly larger than the sequences encoded by the Golomb-Rice method, which in turn gives a size slightly larger than the entropy of
the sequences. Table 1 shows a comparison of the relative size of sequences obtained by the mentioned compression methods in relation to the entropy for selected $k/n$ values.

It is worth noting that, for $k/n$ not exceeding 0.02, the difference between the AC-SBS and Golomb-Rice methods does not exceed 7%, while the sequence compressed with the zlib-deflate method can be up to five times larger. The $k/n$ values in this range are particularly important for storing information about the number of defective products in mass production, e.g., in the glass industry [3].

![Figure 5. Sizes of compressed test sequences.](image)

**Table 1.** Relative sequence sizes for Golomb-Rice and AC-SBS compression methods compared to entropy.

| $k/n$ | 0.0005 | 0.001 | 0.002 | 0.005 | 0.01 | 0.02 | 0.05 |
|-------|--------|-------|-------|-------|------|------|------|
| ZLIB/ENT | 5.051  | 3.676 | 3.000 | 2.415 | 1.969| 1.691| 1.458 |
| ACSBS/ENT | 1.254  | 1.180 | 1.132 | 1.095 | 1.085| 1.077| 1.092 |
| RICE/ENT  | 1.237  | 1.117 | 1.068 | 1.029 | 1.017| 1.011| 1.014 |

In the case of collecting production data for the purpose of future analyzes, aimed at, for example, improving the quality of products, access time to the stored data sequences is important. The factor that largely determines fast access to historical data is the time of sequence decompression. The access time affects not only the convenience of working with data, but also the cost of data processing (e.g., the cost of cloud computing). Due to the comparable sizes of compressed sequences, the difference in decompression times will be significant for the Golomb-Rice and AC-SBS algorithms. Sequence decompression times were examined independently for the ARM (RISC) architecture (Figure 6) and x86 (CISC) architecture (Figure 7). For completeness, graphs of compression times for the compared algorithms are also presented. Compression times as a function of $k/n$ were also examined independently for the ARM (RISC) architecture (Figure 8) and for the x86 (CISC) architecture (Figure 9). However, the compression process itself will not be analyzed in detail. This is due to the fact that, for the interesting $k/n$ intervals, the compression time for both Golomb-Rice and AC-SBS is similar.
Figure 6. Decompression time of test sequences (ARM architecture).

Figure 7. Decompression time of test sequences (x86 architecture).

Figure 8. Compression time of test sequences (ARM architecture).
The results presented in Figures 6 and 7 clearly show that, in both hardware configurations, the zlib-deflate algorithm achieves by far the longest decompression time. For both types of processors, sequences created using the AC-SBS method are decompressed the fastest. For $k/n$ changing from 0 to 0.02, Golomb-Rice decompression time exceeds AC-SBS decompression time by an average of 56.1% in ARM architecture and by 50.9% in x86 architecture. Figure 10 shows the ratio of sequence decompression times by the Golomb-Rice and AC-SBS methods for $k/n$ changing from 0 to 0.02 for x86 and ARM architecture.

In addition, a comparison of data decompression speed was made for data sequences containing $n = 10^8$ elements. Such research is justified by the possibility of using such extensive data sets, e.g., to analyze problems of product quality in mass production [3]. Figure 11 shows the ratio of decompression times using the Golomb-Rice method and the AC-SBS algorithm. It illustrates the results of an analysis of decompression of sparse data sets ($k/n \leq 0.001$) in x86 architecture. The presented results confirm that decompression efficiency using the AC-SBS algorithm is even higher. Golomb-Rice decompression lasts approximately 1.5 to 2 times longer compared with AC-SBS. The abrupt changes in the time ratio visible in the Figure 11 result from the automatic codeword length selection mechanism used in both compression methods.
To end this section, we would like to discuss one more relationship mentioned at the end of the section dedicated to the AC-SBS algorithm, namely the correlation between the ratio of the number of codewords and the ratio of decompression rate. Figure 12 shows two graphs: the first one (solid line) shows the ratio of the number of Golomb-Rice codewords to the number of AC-SBS codewords, while the second one shows the ratio of Golomb-Rice coding decompression rate to AC-SBS decompression rate. At this point, we would like to point out that we assume a Golomb-Rice codeword to be any sequence of bits in which processing involves the execution of a conditional instruction. We explain this fact to avoid ambiguous interpretations. For example, we assume that a sequence of 10 zeros encoded using a binary sequence 11010 consists of four codewords $1 \mid 1 \mid 0 \mid 10$. It clearly shows from the graphs that there is a certain relationship between the analyzed values. It breaks down when the $k/n$ ratio is higher than 0.38, but this results from the fact that, in such a case, both methods fall into their degenerate form in which effective compression is basically impossible. The determined correlation coefficient for $k/n < 0.38$ is 0.946 for the x86 and 0.957 for ARM architecture. Based on the obtained results, we can conclude that the number of conditional instructions in the decompression method directly affects its implementation time.

4. Discussion

In the article, we estimate the entropy of $n$-bit binary sequences with exactly $k$ terms equal to one. We used the proven inequalities as the starting point for a new binary sequence compression algorithm to be applied mainly for coding long series and inverted indexes.
The AC-SBS algorithm presented in the third part of the article is up to 8% less efficient than the Golomb-Rice algorithm in terms of compression. On the other hand, the speed of decompression attained with the use of the AC-SBS algorithm can be even twice higher compared with the Golomb-Rice method. The graphs presented in the article clearly show that the relationship between the two methods strongly depends on the value of the $k/n$ ratio (the probability of ones appearing in the sequence). We can definitely say, however, that there are such $k/n$ values for which AC-SBS is an attractive method of storing inverted indexes. The presented results clearly show that the speed of compression and decompression for both AC-SBS and Golomb-Rice depends stepwise on the optimal length of the codeword for a given sequence. The characteristic steps on the speed graphs appear each time when there is a change in the codeword length being the major determinant of the computational complexity of the entire process. It is also noteworthy that universal compression algorithms (such as the Lempel-Ziv algorithm) differ markedly in terms of both compression and speed from methods dedicated to coding long series and inverted indexes. The difference is so large that it practically rules out the use of universal methods in such applications. The algorithm presented in the article has a practical value. It has already been used by the authors in one of the commercial projects to reduce energy consumption. The project was carried out on a microcontroller with low computing power and assumed the replacement of Golomb-Rice decompression with AC-SBS in order to further reduce energy demand.

**Author Contributions:** A.C. and P.L. researched the literature and justified the research. A.C. formulated the problem, proposed the concept and implemented the AC-SBS algorithm. P.L. tested the operation of the AC-SBS and developed a research plan. A.C. and P.L. performed AC-SBS compression and decompression tests compared to the DEFLATE and Golomb-Rice algorithms. P.L. developed and analyzed the research results. A.C. and P.L. discussed the results and formulated final conclusions. Both authors have read and agreed to the published version of the manuscript.

**Funding:** This research and the APC was funded by Rzeszow University of Technology.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

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

**References**

1. Deming, W. *Out of the Crisis*; MIT Press: Cambridge, MA, USA, 1986.
2. Shewhart, W. *Economic Control of Quality Manufactured Product*; D. Van Nostrand: New York, NY, USA, 1931.
3. Paśko, Ł.; Litwin, P. Methods of Data Mining for Quality Assurance in Glassworks; Collaborative Networks and Digital Transformation; Springer International Publishing: Berlin, Germany, 2019; pp. 185–192.
4. Butcher, S.; Clarke, C.; Cormack, G. *Information Retrieval: Implementing and Evaluating Search Engines*; MIT Press: Cambridge, MA, USA, 2010.
5. Manning, C.; Raghavan, P.; Schütze, H. *Introduction to Information Retrieval*; Cambridge University Press: Cambridge, UK, 2008.
6. Zobel, J.; Moffat, A. Inverted files for text search engines. *ACM Comput. Surv.* 2006, 38, 1–56. [CrossRef]
7. Fano, R. *Transmission of Information: A Statistical Theory of Communications*; The MIT Press: Cambridge, MA, USA, 1961.
8. Shannon, C. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* 1948, 27, 379–423. [CrossRef]
9. Huffman, D. A Method for the Construction of Minimum-Redundancy Codes. *Proc. IRE* 1952, 40, 1098–1101. [CrossRef]
10. Golomb, S. Run-Length Encodings. *IEEE Trans. Inf. Theory* 1966, IT-12, 399–401. [CrossRef]
11. Rice, R.; Plaunt, J. Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data. *IEEE Trans. Commun.* 1971, 16, 889–897. [CrossRef]
12. Elias, P. Universal codeword sets and representations of the integers. *IEEE Trans. Inf. Theory* 1975, 21, 194–203. [CrossRef]
13. Apostolico, A.; Fraenkel, A. Robust transmission of unbounded strings using Fibonacci representations. *IEEE Trans. Inf. Theory* 1987, 33, 238–245. [CrossRef]
14. Brisaboa, N.; Fariña, A.; Navarro, G.; Esteller, M. (S,C)-Dense Coding: An Optimized Compression Code for Natural Language Text Databases. In *String Processing and Information Retrieval*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 122–136.
15. Boldi, P.; Vigna, S. Codes for the World Wide Web. *Internet Math.* 2005, 2, 407–429. [CrossRef]
16. Elias, P. Efficient Storage and Retrieval by Content and Address of Static Files. *J. ACM* 1974, 21, 246–260. [CrossRef]
17. Fano, R. *On the Number of Bits Required to Implement an Associative Memory*; MIT Project MAC Computer Structures Group: Cambridge, MA, USA, 1971.
18. Moffat, A.; Stuiver, L. Binary Interpolative Coding for Effective Index Compression. *Inf. Retr. J.* 2000, 3, 25–47. [CrossRef]
19. Anh, N.; Moffat, A. Inverted Index Compression Using Word-Aligned Binary Codes. *Inf. Retr. J.* 2005, 8, 151–166. [CrossRef]
20. Pibiri, G.; Venturini, R. Techniques for Inverted Index Compression. *ACM Comput. Surv.* 2021, 53, 1–36. [CrossRef]
21. Trotman, A. Compressing inverted files. *Inf. Retr. J.* 2003, 6, 5–19. [CrossRef]
22. Catena, M.; Macdonald, C.; Ounis, I. On Inverted Index Compression for Search Engine Efficiency. In *Advances in Information Retrieval*; Springer International Publishing: Berlin, Germany, 2014; pp. 359–371.
23. Salomon, D.; Motta, G. *Handbook of Data Compression*; Springer: Berlin, Germany, 2010.
24. Gallager, R.; Van Voorhis, D. Optimal Source Codes for Geometrically Distributed Integer Alphabets. *IEEE Trans. Inf. Theory* 1975, IT-21, 228–230. [CrossRef]
25. Somasundaram, K.; Domnic, S. Extended Golomb Code for Integer Representation. *IEEE Trans. Multimed.* 2007, 9, 239–246. [CrossRef]
26. Rice, R.; Robert, F. *Some Practical Universal Noiseless Coding Techniques*; Technical Report 79-22; Jet Propulsion Laboratory—JPL Publication: Pasadena, CA, USA, 1979.
27. Rice, R. *Some Practical Universal Noiseless Coding Techniques—Part III. Module PSI14.K*; Technical Report 91-3; Jet Propulsion Laboratory—JPL Publication: Pasadena, CA, USA, 1991.
28. Fenwick, P. *Punctured Elias Codes for Variable-Length Coding of the Integers*; Technical Report Technical Report 137; Department of Computer Science, The University of Auckland: Auckland, New Zealand, 1996.
29. Robinson, T. *Simple Lossless and Near-Lossless Waveform Compression*; Technical Report Technical Report CUED/F-INFENG/TR.156; Cambridge University: Cambridge, UK, 1994.
30. Kiely, A. Selecting the Golomb Parameter in Rice Coding; Technical Report 42-159; Jet Propulsion Laboratory, California Institute of Technology: Pasadena, CA, USA, 2004.
31. Fraenkel, A.; Klein, S. Novel Compression of Sparse Bit-Strings—Preliminary Report. *Comb. Algorithms Words* 1985, 12, 169–183.
32. Salomon, D. Prefix Compression of Sparse Binary Strings. *ACM Crossroads Mag.* 2000, 6, 22–25. [CrossRef]
33. Tanaka, H.; Leon-Garcia, A. Efficient Run-Length Encodings. *IEEE Trans. Inf. Theory* 1982, IT-28, 880–890. [CrossRef]
34. Ferragina, P.; Venturini, R. A simple storage scheme for strings achieving entropy bounds. *Theor. Comput. Sci.* 2007, 372, 115–121. [CrossRef]
35. Zhang, J.; Long, X.; Suel, T. Performance of Compressed Inverted List Caching in Search Engines. In Proceedings of the 17th International Conference on World Wide Web, New York, NY, USA, 21–25 April 2008; pp. 387–396. [CrossRef]
36. Ziv, J.; Lempel, A. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inf. Theory* 1977, IT-23, 337–343. [CrossRef]
37. Ziv, J. The Universal LZ77 Compression Algorithm is Essentially Optimal for Individual Finite-Length N-Blocks. *IEEE Trans. Inf. Theory* 2009, 55, 1941–1944. [CrossRef]
38. Mascioni, V. An Inequality for the Binary Entropy Function and an Application to Binomial Coefficients. *J. Math. Inequal.* 2012, 6, 501–507. [CrossRef]
39. Robbins, H. A remark on Stirling’s formula. *Am. Math. Mon.* 1995, 62, 26–29.
40. Zhang, N.; Wu, X. Lossless compression of color mosaic images. *IEEE Trans. Image Process.* 2006, 15, 1379–1388. [CrossRef]
41. Hashimoto, M.; Koike, A.; Matsumoto, S. Hierarchical image transmission system for telemedicine using segmented wavelet transform and Golomb-Rice codes. Seamless Interconnection for Universal Services. In Proceedings of the Global Telecommunications Conference, GLOBECOM’99 (Cat. No.99CH37042), Rio de Janeiro, Brazil, 5–9 December 1999; pp. 2208–2212.
42. Brunello, D.; Calvagno, G.; Mian, G.; Rinaldo, R. Lossless Compression of Video Using Temporal Information. *IEEE Trans. Image Process.* 2003, 12, 132–139. [CrossRef] [PubMed]
43. Nguyen, T.; Marpe, D.; Schwarz, H.; Wiegand, T. Reduced-Complexity Entropy Coding of Transform Coefficient Levels Using Truncated Golomb-Rice Codes in Video Compression. In Proceedings of the 2011 18th IEEE International Conference on Image Processing, Brussels, Belgium, 11–14 September 2011; pp. 753–756.
44. Kalaivani, S.; Tharini, C. Analysis and implementation of novel Rice Golomb coding algorithm for wireless sensor networks. *Comput. Commun.* 2020, 150, 463–471. [CrossRef]
45. Sugiura, R.; Kamamoto, Y.; Harada, N.; Moriya, T. Optimal Golomb-Rice Code Extension for Lossless Coding of Low-Entropy Exponentially Distributed Sources. *IEEE Trans. Inf. Theory* 2018, 64, 3153–3161. [CrossRef]
46. Sugiura, R.; Kamamoto, Y.; Moriya, T. Integer Nesting/Splitting for Golomb-Rice Coding of Generalized Gaussian Sources. In Proceedings of the 2018 Data Compression Conference, Snowbird, UT, USA, 27–30 March 2018.
47. Vasilache, A. Order Adaptive Golomb Rice Coding for High Variability Sources. In Proceedings of the 2017 25th European Signal Processing Conference (EUSIPCO), Kos Island, Greece, 28 August–2 September 2017.
48. Domnic, S.; Glory, V. Extended Rice Code and Its application to R-Tree Compression. *IETE J. Res.* 2015, 61, 634–641. [CrossRef]
49. McKenzie, B.; Bell, T. Compression of sparse matrices by blocked rice coding. *IEEE Trans. Inf. Theory* 2001, 47, 1223–1230. [CrossRef]