INFORMATICA, 2026, Vol. 37, No. 1, 229–249

DOI: https://doi.org/10.15388/26-INFOR621

Context-Aware Word-Based Forward Coding

Igor ZAVADSKYI[1], Shmuel T. KLEIN[2], Dana SHAPIRA[3,*]

[1] *Department of Computer Science and Cybernetics, Taras Shevchenko National University of Kyiv, Kyiv, Ukraine*
[2] *Department of Computer Science, Bar Ilan University, Ramat Gan 52900, Israel*
[3] *Department of Computer Science, Data Science and Artificial Intelligence Research Center, Ariel University, Ariel 40700, Israel*
e-mail: ihorzavadskyi@knu.ua, tomi@cs.biu.ac.il, shapird@g.ariel.ac.il

Received: June 2025; accepted: February 2026

Abstract. Forward-looking coding has recently been introduced as a source modelling paradigm that exploits predictions of forthcoming symbols. In this paper, we extend this methodology to word-level alphabets, enabling improved compression performance for large and variable-length symbol sets. We present a space-efficient scheme for encoding header information, with particular emphasis on the accurate representation of symbol frequency distributions. In addition, we propose an alternative ordering strategy for word-based dictionaries that leverages the adaptive nature of forward-looking compression. We further show how these techniques can be integrated with a word-based *Prediction by Partial Matching* model of order one, while avoiding the zero-frequency problem. Experimental results confirm the effectiveness of the proposed approach across multiple datasets.

Key words: data compression, adaptive coding, arithmetic coding, entropy, forward coding.

1. Introduction

Lossless data compression constitutes a fundamental paradigm in information theory, enabling the reduction of the required size for data representation, while guaranteeing the exact reconstruction of the original input file. Unlike lossy methodologies, which approximate data to enhance compression ratios, lossless algorithms operate by identifying and eliminating redundancy without introducing distortion. Consequently, lossless data encoding remains a foundational component of modern storage and transmission.

The utility of lossless data compression has evolved beyond sequential archival storage to encompass random access functionality through the use of compact data structures, see Navarro (2016). By organizing compressed data into navigable forms, such as wavelet trees (see Grossi *et al.*, 2003 and Baruch *et al.*, 2020) or compressed suffix arrays (see Grossi and Vitter, 2005), these data structures enable direct access and querying of the underlying information without the computational overhead of full decompression (e.g. Baruch *et al.*, 2017, 2020). This capability allows massive datasets to reside entirely within

*Corresponding author.

faster levels of the memory hierarchy, permitting algorithms to operate directly over the compressed representation.

The entropy of a memory-less source X is defined by the well-known Shannon formula $H_0(X) = -\sum_{s\in\Sigma} p(s)\log p(s)$, where Σ denotes the alphabet and $p(s)$ is the probability of the symbol s appearance. This quantity is widely used as a theoretical lower bound on the efficiency of *semi-static* zero-order compression methods, that is, compression schemes that rely solely on symbol frequencies, which are assumed to be constant throughout the entire file. If applicable, one can discard parts of the data to achieve compression ratios beyond the H_0 value, which results in a lossy compression process. Another way to overcome the Shannon limit without compromising on data loss is to design a more accurate model that reflects the actual data source and yields a lower entropy, e.g. to design an enhanced probabilistic modelling of the next encoded element. This implies the potential for further storage savings.

Common *adaptive* compression methods, like the dynamic Huffman algorithm due to Vitter (1987), update the model based on the data encountered in the processed portion of the input file. To predict the probability of the next element being encoded at the current position in the file, the model analyses the statistics of the elements that have appeared up to that point. However, in situations where the precise occurrence count of each element throughout the entire file is known, an alternative approach called *Forward-Looking* Adaptive Compression, proposed by Klein *et al.* (2020), can be utilized. This method adjusts the dynamic model based on predictions of what is yet to come, effectively peering into the future. This contrasts with traditional *Backward-Looking* dynamic methods, which develop their current model from past observations of what has already happened. Hence, forward-looking compression bridges semi-static and adaptive approaches, combining the advantages of both. The net encoding produced by forward-looking has been proven to be more efficient than a static (Huffman, 1952) code, well known to be optimal, by at least $|\Sigma| - 1$ bits, where $|\Sigma|$ refers to the size of the alphabet (Klein *et al.*, 2020, Theorem 2). This upper bound of the net encoding is smaller than all known dynamic variants to date.

We use the term "alphabet" in a broader interpretation than just individual letters. It includes various types of substrings of different lengths. The crucial factor is that a precise procedure should be followed to parse the input text into a well-defined sequence of elements within the alphabet. When dealing with natural language texts, using *words* as the elements of the alphabet often leads to significantly better compression than using single characters alone, see Moffat (1989). The first contribution of this paper is to extend the forward-looking encoding method to alphabets of variable-length strings, instead of just characters. This involves increasing the size of the alphabet, which results in further savings to the net encoding due to the upper bound performance of the forward method being a factor of the alphabet size.

In *Information Retrieval*, it is often assumed that the list of distinct words is necessary for the retrieval process and thus already stored. Therefore, implementing the word-based compression technique incurs no additional overhead except for the list of word frequencies. However, storing exact symbol frequencies for the entire alphabet in the header may incur significant overhead, making the method impractical, especially for large alphabets

considered here. This drawback is even more significant when the list of distinct words does not exist in advance, e.g. in file archiving.

An enhanced forward-looking approach that further improves the net encoding has been proposed in Fruchtman *et al.* (2022). It assigns increased weights to positions in the file that are closer to the currently processed element, rather than treating all positions equally. Yet, the overhead arising from the encoded file's prelude is even more expensive than that of forward-looking and led to the adoption of the Backward-Looking heuristic approach in Fruchtman *et al.* (2021), which compromises on provable optimality guarantees. The current paper's second contribution is an alternative solution for encoding headers of statistical methods that include the exact frequencies of the alphabet elements. It allows us to compress such a header into a tiny portion of the archived text.

A popular method of compressing a sequence of distinct words is by ordering them lexicographically and applying the *Prefix Omission Method* (POM) by Bratley and Choueka (1982). Our third contribution improves the compression ratio by replacing some low-frequency words in the text with meta-characters of higher frequency. This is made possible by using an alternative order for word alphabets. Experiments show that our method provides a competitive alternative to dictionary compression using specialized techniques such as POM.

The compression technique known as *Prediction by Partial Matching* (PPM) by Cleary and Witten (1984) enhances the compression efficiency by exploiting dependencies in the data via contextual models. PPM considers a variable-length history of symbols to capture dependencies in the data and makes more precise predictions about the following character based on the context. Our fourth contribution is a PPM of order-1 algorithm for a word-based alphabet, which 'looks into the future', unlike the work of Avrunin *et al.* (2022), which combines future-based forward-looking with the past-based PPM. The most interesting theoretical contribution of the proposed algorithm that combines PPM with forward-looking compression is that, in contrast with traditional backward PPM, there is no zero-frequency problem in the forward compression, as the probability of the special 'escape' symbol, which is used to indicate a shorter context, can be calculated precisely.

The following summarizes the contributions of this paper.

1. Extending the forward-looking encoding method to big alphabets, including variable-length strings, rather than just characters, leveraging the larger alphabet to amplify forward-looking gains.
2. A compact header encoding method for storing word frequencies.
3. Improving compression by selectively mapping low-frequency words to higher-frequency meta-characters.
4. A forward-looking order-1 word-based PPM adaptation.

This work is an extended version of a paper that has been presented in 2024 at the Data Compression Conference, DCC, and appeared in its proceedings in Zavadskyi *et al.* (2024). The additional contributions over those in the conference paper are:

1. The algorithms in Section 4 have been simplified.

2. In the newly added Section 5, the context-aware word-based forward-looking paradigm has been adapted to the PPM compression algorithm.
3. The experimental results presented in Section 6 have been extended.

Our paper is organized as follows. Section 2 recalls the details of forward-looking compression. Section 3 presents an approach for encoding frequency data, which constitutes a significant part of the compressed file's header overhead. Section 4 provides the word-based forward-looking algorithm featured with the low-frequency words replacement technique, Section 5 suggests a word-based PPM forward-looking algorithm, Section 6 presents experimental results, and finally, Section 7 concludes the paper.

2. Forward-Looking Encoding

As mentioned above, traditional adaptive compression algorithms concentrate solely on the item currently processed, increasing its frequency after each occurrence, and thereby potentially leading to shorter codewords for future use. These savings may, however, result in the elongation of certain other codewords. An alternative adaptive *forward-looking* approach has been proposed, assuming the exact frequencies of each element throughout the entire file are available, possibly by a pre-scan of the source file. It enhances the conventional 'self-centered' behaviour by adopting a more 'collaborative' strategy. In this approach, the frequency of the currently processed item is *decremented*, even if this results in the associated codeword becoming longer. This operation, however, has the potential to reduce the average codeword length for the remaining elements, resulting in more efficient overall space utilization.

The *forward-looking* paradigm can be applied to any statistical encoding algorithm. In Klein *et al.* (2020), the forward-looking method is presented in the context of Huffman coding, whereas in Fruchtman *et al.* (2023), the forward-looking algorithm also considers arithmetic coding. The latter work provides *bidirectional* adaptive compression, taking both past and future into account, and provably improves on the net encoding of the forward-looking variant.

Algorithm 1 presents the generic *forward-looking* modelling method suited to all statistical encodings. As any semi-static compression method, this algorithm requires two passes over the text. In line 1, the text T is scanned in a preprocessing stage in order to gather the underlying statistics, storing the frequency $\mathsf{freq}(\sigma)$ for each element σ of the alphabet Σ. In line 2, these statistics are transmitted to the decoder as part of the compressed file's prelude, and are used as the model by both the encoder and decoder (line 3). In the processing stage, the text $T = x_1, \ldots, x_n$ is rescanned. The element x_i is encoded, its frequency is decremented by 1 and the model is updated. In particular, in case the weight of σ becomes 0, the corresponding element is removed from the model. For simplicity, from now onward, we assume that the '$\mathsf{update\ the\ model}$' statement also removes zero-frequency elements from the model. The generic Forward-Looking-Decode algorithm is straightforward and therefore omitted. The time complexity of both preprocessing and encoding passes of Algorithm 1 is $O(n)$, where n is the text length. Similarly, the decoding

Algorithm 1: GENERIC FORWARD-LOOKING ENCODING

```
Forward-Looking-Encode(T = x_1 ··· x_n)
preprocess T to get freq(σ)   ∀σ ∈ Σ
output encoding of freq(σ)   ∀σ ∈ Σ
initialize the model according to the distribution {freq(σ), σ ∈ Σ }
for i ← 1 to n do
    encode x_i according to current model
    freq(x_i) ← freq(x_i) − 1
    if freq(x_i) = 0 then
        remove x_i from the model
    update the model
```

process executes in $O(n)$ time, as it requires a single pass to parse the frequency header and reconstruct the original text T from the compressed stream.

3. Word Frequency List Compression

As is known, the word frequencies in a natural language text may be modelled by Zipf's law, see Zipf (1949). It implies that in a large enough text, the frequency of the most frequent word is much larger than the frequency of the second frequent word, which, in turn, is much larger than the frequency of the following word and so on. The decay being logarithmic; at the bottom of a table of frequencies sorted by non-ascending order, there are many words of frequency 1, many of frequency 2, etc. Considering this specificity, we propose the following compact representation for frequency lists.

Split the list into two parts according to an integer parameter t. For the frequencies greater than t, apply delta encoding; for all natural numbers less than or equal to t, store the number of words having a frequency equal to this number. This way, we get a sequence consisting of many small numbers, which can be efficiently encoded using some universal code, e.g. one of those defined by Elias (1975) or the *Reverse Multi-Delimiter* (RMD) codes defined in Zavadskyi and Anisimov (2020). Given a list of unique words in the text in non-increasing frequency order, we can easily reconstruct the frequencies of all words from that compressed number sequence.

Let $\mathcal{E}$ denote an encoding function employing any universal code. Algorithm 2 describes the compression of a non-increasing input frequency list $f_1, \ldots, f_m$, given a threshold t. The dot sign $\cdot$ is used for concatenation. The threshold t can be adjusted to minimize the space occupied by the encoded sequence $\mathcal{L}$. Note that changing t a little requires recalculating only a few values of $\mathcal{L}$, which simplifies the search. Our experiments show that the optimal value t is close to the fixed point of the frequencies f_i, that is, the index i such that $i \simeq f_i$, which can be a good starting position to search for this optimal value. The symmetrical decoding algorithm, Freq-List-Decode, is straightforward.

Algorithm 2: FREQUENCY LIST ENCODING

Freq-List-Encode($f_1, \ldots, f_m, t$)
$\mathcal{L} \leftarrow \mathcal{E}(f_1)$
$i \leftarrow 2$
while $f_i > t$ **do**
  $\delta \leftarrow f_i - f_{i-1}$
  $i \leftarrow i + 1$
  $\mathcal{L} \leftarrow \mathcal{L} \cdot \mathcal{E}(\delta)$
for j **from** t **down to** 1 **do**
  $n_j \leftarrow$ number of words having frequency j
  $\mathcal{L} \leftarrow \mathcal{L} \cdot \mathcal{E}(n_j)$
return $\mathcal{L}$

Table 1
Alice in Wonderland, word frequencies.

w	the	and	to	...	though	no	is	could	*	Mock
freq(w)	1505	714	703	...	64	63	63	62	60	56
δ	1505	791	11		...	1	0	1	2	4

w	my	its	Alice.	your	quite	by	an		then	their
freq(w)	55	55	54	53	53	53	52	51	50	50
mult(f)	2		1		3		1	0	2	

As an example, let us consider *Alice in Wonderland* by Lewis Carroll. The words w with frequencies freq(w) greater than 49 are given in Table 1. The optimal threshold value is $t = 55$. The upper part of the table refers to words with frequencies larger than 55, while the lower part of the table refers to those with frequencies 55 or smaller. The values being encoded are shown in rows δ and mult(f). In the upper part of the table the differences, δ, between consecutive original values are calculated, starting with the maximum value, 1505, in this example. In the lower part of the table, we store mult(f) – the multiplicity of the frequency f, which is the number of words having the frequency f.

In total, for 5 311 unique words in this text, there are only 112 encoded elements in the sequence $\mathcal{L}$ compressed into 81 bytes using one of the RMD codes, $R_{2\text{-}\infty}$ (see Zavadskyi and Anisimov, 2020). In comparison, the sequence of frequencies of words in the lexicographically sorted dictionary of this text can not be compressed to less than 1 900 bytes with the bzip2 archiver, and to less than 370 bytes, if the dictionary elements are sorted in order of descending frequencies.

Since the frequency list is given in a sorted order, Algorithm 2 requires $O(m)$ time, and $O(n + m) = O(n)$ time together with the forward encoding Algorithm 1, where m is the number of distinct frequencies of symbols, and n is the length of the text. Our method of frequency processing is faster than POM, as it does not require lexicographical dictionary sorting and time-consuming string processing.

4. A Word-Based Forward-Looking Algorithm

The order of the words in the dictionary contains information that can be used during the compression. We showed above that sorting the dictionary elements by non-increasing frequency helped us reduce the size of the list of compressed frequency values. In addition to sorting the dictionary by frequencies, let us sort the words of the same frequency in the order of the leftmost (first) occurrence in the text. Then, if the encoder replaces all leftmost occurrences of words of the same frequency f with some special *meta-character* $\mathsf{MC}(f)$ (different for every frequency f), the decoder can replace such symbols with words of the respective frequencies, taking them from the dictionary one by one. If the frequency of the meta-character is larger than f, this technique may enhance the compression performance, as it replaces some terms of frequency f with more frequent ones.

Let us consider the following example text to clarify this definition:

```
The dog is dog, and the cat is cat, and mice, oh,
all the mice, know that,
```

not taking into account capitalization and punctuation marks. The dictionary of words, sorted as described above, is the following (frequencies are given in brackets):

the(3), dog(2), is(2), and(2), cat(2), mice(2), oh(1), all(1), know(1), that(1).

We obtain four words of frequency 1, five words of frequency 2, and a single word of frequency 3. We suggest extending the alphabet of words to include the meta characters {MC(1), MC(2)} with frequencies 4 and 5, respectively. MC(1) replaces all words of frequency 1, while MC(2) replaces all leftmost occurrences of words of frequency 2. As a result, the encoder receives the following preprocessed text:

the MC(2) MC(2) dog MC(2) the MC(2) is cat and MC(2) MC(1) MC(1) the mice MC(1) MC(1).

The decoder, reading the first meta-character MC(2), starts processing the list of words of frequency 2 in the dictionary and outputs dog. The second meta-character MC(2) corresponds to the next word in the list, i.e. is, and so on. Note that using MC(1) eliminates all words with frequency 1, while MC(2) reduces all frequencies 2 by 1. Thus, the alphabet and corresponding probabilities used by the encoder are

{the (3/17), dog (1/17), is (1/17), and (1/17), cat (1/17), mice (1/17), MC(1) (4/17), MC(2) (5/17)}.

To evaluate the savings of this replacement method, we calculate the estimated size of the text as the sum of the *information-contents*, $-\log p_i$, where p_i is the probability of

occurrence of the i-th word, of all the words in the text: for the obtained text this equals 45.12 bits, while for the original text, the sum is 54.73 bits. This estimated size may be achieved if arithmetic coding is used as a compression method.

In fact, the proposed replacement method is a generalization of the idea presented by Adiego *et al.* (2009) of using a special character to encode singleton words. It can be applied to any encoding where frequencies are known in advance, either static (as shown in the example) or adaptive forward encoding, implemented in Algorithm 3. But before formalizing the encoding algorithm, let us derive the criterion of expedience of the replacement of words with meta-characters in the general case of static compression.

Theorem 1. *Let f be some frequency with multiplicity k, that is, the text T contains k symbols of frequency f. Let T' be the text obtained from T by replacing each leftmost occurrence of a symbol having frequency f with a meta-character* $\mathsf{MC}(f)$. *Then,*

$$H_0(T) > H_0\big(T'\big) \quad \text{if and only if} \quad k(f-1)^{f-1} > f^f.$$

Proof. The meta-character $\mathsf{MC}(f)$ has frequency k and probability k/n, where n is the number of symbols in the text T. As the compression performance of arithmetic coding can be approximated by the entropy, we express the space usage of an individual element s with probability $p(s)$ by its *information content*, $-\log p(s)$. Thus, replacing each leftmost occurrence of a symbol with its meta-character *saves*

$$\log_2 \frac{k}{n} - \log_2 \frac{f}{n} = \log_2 \frac{k}{f}$$

bits of space, or $k \log_2(k/f)$ bits in total. However, if $f > 1$, all $k(f-1)$ non-leftmost occurrences of symbols of frequency f become less frequent by 1. Thus, the replacement causes increasing the length of the code of these symbols by

$$k(f-1)\left(\log_2 \frac{f}{n} - \log_2 \frac{f-1}{n}\right) = k(f-1)\log_2 \frac{f}{f-1}$$

bits. In total, the described replacement technique allows us to save

$$k\left(\log_2 \frac{k}{f} - (f-1)\log_2 \frac{f}{f-1}\right) = k \log_2 \frac{k(f-1)^{f-1}}{f^f}$$

bits of space, given k words of frequency f. This value is positive, if and only if the argument of the log is larger than 1, that is, $k(f-1)^{f-1} > f^f$. □

If, for a given frequency f, the condition of Theorem 1 is satisfied, f is inserted into a replacement frequency set $\mathcal{R}$, as shown in line 3 of Algorithm 3. In adaptive encoding, savings may be a bit different than $k \log_2 \frac{k(f-1)^{f-1}}{f^f}$. However, experiments we performed on real-world texts show that a list of replacement frequencies obtained by the criterion $k(f-1)^{f-1} > f^f$, appears to be optimal for the adaptive forward encoding as well.

Algorithm 3: WORD-BASED FORWARD ENCODING WITH WORD REPLACEMENT

```
   Word-Based-Forward(T)
   // Create a list of frequencies of unique words
1  F ← {freq(w) : w ∈ T}
   // Generate a set of frequencies for word replacement
2  R ← {f ∈ F : mult(f)(f − 1)^(f−1) > f^f}
   // Generate a set of words for replacement
3  S ← {w ∈ T : freq(w) ∈ R}
   // Decrease the frequencies of words to be replaced
4  foreach w ∈ S do freq(w) ← freq(w) − 1
   // Assign multiplicities to meta-character frequencies
5  foreach f ∈ R do freq(MC(f)) ← mult(f)
   // Generate a set of terms
6  M ← {σ ∈ Σ : freq(σ) > 0} ∪ {MC(f) : f ∈ R}
7  initialize the model with the distribution {freq(σ), σ ∈ M}
8  while not EOF(T) do
9      w ← next input word
10     if w is seen for the first time in T and w ∈ S then
11         w ← MC(freq(w))
12     encode w
13     freq(w) ← freq(w) − 1
14     update the model
15 return
```

In Algorithms 3 and 4, we denote the frequency of a word w by $\mathsf{freq}(w)$. By $\mathsf{MC}(x)$, we denote the meta-character corresponding to frequency x. In Algorithm 3, we get the set of different frequencies (line 1) and form the replacement sets of frequencies $\mathcal{R}$ and words having these frequencies $\mathcal{S}$ in lines 2 and 3. Since the leftmost occurrences of words from $\mathcal{S}$ are replaced with meta-characters, we decrement the frequencies of these words (line 4) and compensate for this reduction by assigning appropriate multiplicities to the meta-character frequencies (line 5). In lines 6–7, we initialize the model with the frequencies of regular words and meta-characters. The encoding algorithm itself is quite straightforward: for each next word w, if w is seen for the first time and belongs to the replacement set, we substitute it with the corresponding meta-character (lines 10–11). In lines 12–14, we encode w, decrement its frequency and update the model, in accordance with the generic forward-looking encoding Algorithm 1.

To ensure the correctness of the encoding algorithm, in line 12, we have to encode w with respect to its forward probability $p(w) = \mathsf{freq}(w)/(n - i + 1)$, where i is the current text position. As at each text position, we decrease some frequency by 1 (line 13), $\sum_{w \in \mathcal{M}} p(w)$ is always 1, which proves that the model built by Algorithm 3 is correct.

The reversed decoding algorithm presented in Algorithm 4 receives the encoded text $\mathcal{E}(T)$ and the dictionary D as arguments. We assume D is sorted in order of word frequency, where every group of words of frequency belonging to the replacement set $\mathcal{R}$ is second-level sorted by the position of the leftmost occurrence in the text T. In line 1, the

Algorithm 4: Word-based Forward Decoding with Word Replacement

```
   Word-Based-Forward-Decode(E(T), D)
   // Decompress a list of frequencies
1  F = {f_1, ..., f_m} ← Freq-List-Decode(D)
   // Generate a set of frequencies for word replacement
2  R ← {f ∈ F : mult(f)(f − 1)^{f−1} > f^f}
   // Store initial frequencies of meta-characters
3  foreach f ∈ R do G[MC(f)] ← mult(f)
   // Decrease the frequencies of words to be replaced
4  foreach w ∈ T such that freq(w) ∈ R do freq(w) ← freq(w) − 1
   // Assign multiplicities to meta-character frequencies
5  foreach f ∈ R do freq(MC(f)) ← mult(f)
   // Generate a set of terms
6  M ← {σ ∈ Σ : freq(σ) > 0} ∪ {MC(f) : f ∈ R}
7  initialize the model with the distribution {freq(σ), σ ∈ M}
   // Initialize the list of indices of replaced words in the
    dictionary
8  foreach f ∈ R do J[f] ← index of the first word of frequency f in D
9  while not EOF(E(T)) do
10     w ← next decoded term from E(T)
11     if w is a meta-character then
12        f ← G[w]      //get the initial frequency of meta-character w
13        w ← D[J(f)]   //recover the word
14        J[f] ← J[f] + 1 //advance to the following word with frequency
            f
15     freq(w) ← freq(w) − 1
16     output w
17     update the model
18 return
```

list of word frequencies, encoded as part of the dictionary, is decompressed by applying the decoding algorithm, Freq-List-Decode(D), for reversing the encoding of Algorithm 2. We maintain the array J of indices of replaced words that maps each frequency f to the list of words with frequency f in the dictionary. In line 8, $\mathsf{J}(f)$ is initialized by pointing to the first word with frequency f, which is the leftmost occurrence in T of those words. An auxiliary array $\mathsf{G}[w]$ is initialized in line 3 and used in line 12 to return the initial frequency corresponding to a meta-character w. The word itself is recovered in line 13 using the dictionary D, and the pointer is advanced to the following word associated with the same meta-character in line 14.

The time complexity of both encoding and decoding Algorithms 3 and 4 is, obviously, $O(n)$, where n is the length of the text. It is better than the worst-case encoding time $O(n \log^2 n)$ of the bzip2 compressor, used as the comparison base in Section 6, and the same as the bzip2 decoding time (according to the bzip2 white paper, Seward, 2019).

5. Context-Based Forward-Looking Algorithm

As mentioned in Aljehane and Teahan (2017), the most efficient PPM model for word-based alphabets is an order-one model, i.e. to estimate the probability of a word in its context, it is advisable to consider only one previous word. Combining PPM with forward coding, we can store the forward frequency of each word after every other word together with the compressed file and update the model accordingly at each step of encoding and decoding. This may significantly reduce the compressed text size, but at the expense of a high overhead from storing numerous context-frequency tables. Thus, we suggest to store these tables partially. Namely, it is reasonable to save the frequency of the word w after the word c only if the probability of w appearing after c differs significantly from the unconditional probability of the word w.

Let T' be the sequence of terms obtained after replacing low-frequency words in the text by meta-characters. Let us discuss two consecutive terms $c = T'[i]$ and $w = T'[i+1]$. We call c the *context* and w the *term in the context* c. Let $\mathsf{freq}(w|c)$ be the context forward frequency of w, i.e. the number of occurrences of w right after c in the suffix $T'[i+1 \ldots n]$. The idea is to store context frequencies of some terms in some contexts in a *context table* CT: $CT[c][w] = \mathsf{freq}(w|c)$. This allows us to use context forward frequencies of terms in the model instead of their unconditional forward frequencies, which makes the model more accurate. Note that we represent the two-dimensional matrix CT as an array of arrays using $[c][w]$ rather than $[c, w]$ as notation for its indices, to permit references $CT[c]$ to one-dimensional sub-matrices in what follows.

There is a trade-off between the size of context tables and the compression gain they provide. Let $\mathsf{P}(w|c) = \mathsf{freq}(w|c)/\mathsf{freq}(c)$ be the context forward probability of w, while $\mathsf{P}(w) = \mathsf{freq}(w)/(n - \mathsf{left}(w))$ is the unconditional forward probability of w, where $\mathsf{left}(w)$ is the position of the leftmost occurrence of w. We set two threshold values th_f and th_g, representing, respectively, a bound on the frequency and a bound on the expected compression gain, and define an element in the table CT at index $[c][w]$ if and only if

$$\mathsf{freq}(c) > th_f \quad \text{and} \quad \left|\log \frac{P(w|c)}{P(w)}\right| \cdot \mathsf{freq}(w|c) > th_g \tag{1}$$

(the absolute value is needed, because $P(w|c)$ may be smaller or larger than $P(w)$). The first inequality enables us to avoid storing symbol probabilities in low-frequency contexts, since the cost of storing the context itself may exceed the compression gain it provides. The left part of the latter inequality can be considered as a measure of the compression gain achieved by defining the element of the context table at index $[c][w]$. We need it as this gain has to cover at least the cost of storing the term w and its context frequency $\mathsf{freq}(w|c)$. Examples of chosen threshold values are given in the experimental section in Table 5.

We shall henceforth use the notations $c \in CT$ and $w \in CT[c]$ to refer to the fact that $CT[c]$ and $CT[c][w]$ have been defined, respectively. Accordingly, we also say "c belongs to CT".

Algorithm 5 shows the forward-looking context-based compression method. Its general logic is the same as in Algorithm 3, and its initialization is the same as in lines 1–7 of

Algorithm 5: CONTEXT-AWARE FORWARD ENCODING

Context-Aware-Forward($T = x_1, \cdots, x_n; th_f; th_g$)
1 Initialize by lines 1–7 of Alg. 3
2 Calculate the context table CT for the entire text and given thresholds th_f, th_g
// Calculate the frequencies of context terms in CT
3 **foreach** $c \in CT$ **do** cfreq(c) $\leftarrow \sum_{w \in CT[c]}$ freq($w|c$)
4 $c \leftarrow$ null
5 **for** $i \leftarrow 1$ **to** n **do**
6 **if** x_i *is seen for the first time in* T *and* $x_i \in \mathcal{S}$ **then**
7 $w \leftarrow$ MC(freq(x_i))
else
8 $w \leftarrow x_i$
9 Encode-and-Update-the-Model($c, w, n - i+1$)
10 $c \leftarrow w$
11 **return**

Algorithm 3, except for the initialization of the context table CT (line 2) and frequencies of using terms as contexts cfreq (line 3). However, we can use different models depending on whether the term c belongs to the context table. This choice is reflected in the function Encode-and-Update-the-Model given in Algorithm 6, using values and arrays precomputed in lines 1–8 of Algorithm 5.

- If $c \notin CT$, then we construct the model based on unconditional forward frequencies of the terms (line 8).
- If $c \in CT$, the probability of a term is calculated differently, depending on whether it belongs to $CT[c]$. If it does, we use its context probability (line 3) and update the context frequencies (lines 4, 5). Otherwise, we relate its forward frequency to the total frequency of terms that do not belong to $CT[c]$ in the remaining text suffix (the first fraction in line 7). Of course, we should multiply this fraction by the 'escape' (out-of-context) probability. Unlike traditional backward PPM, the forward compression has no zero-frequency problem, and the 'escape' probability can be calculated precisely. Namely, it equals to the ratio of the number of the term c occurrences, where the next term does not belong to $CT[c]$, to the total forward frequency of c (second fraction in line 7).

Note that we use freq(c) + 1 instead of freq(c) in lines 3 and 7, since when we encode the term w, its context c has already been encoded and its frequency has been decreased by 1. However, we must not count the current term w as already encoded, and in relation to it, its context c should also be addressed as 'not yet processed'.

Let us demonstrate the correctness of the model built in Algorithm 6. If $c \notin CT$, then the regular forward model is applied,[1] the correctness of which is proved in Section 4. Let

[1] In the case $c \notin CT$, we can use more accurate probabilities of terms based on their occurrences not preceded by any context. However, the cost of storing the 'out-of-context' frequencies in a separate table outweighs the savings.

Algorithm 6: ENCODE AND UPDATE THE MODEL FUNCTION

```
   Encode-and-Update-the-Model(c, w,suffixLen)
1  if c ∈ CT then
2      if w ∈ CT[c] then
3          p(w) ← CT[c][w]/(freq(c) + 1)
4          CT[c][w] ← CT[c][w] − 1
5          cfreq(c) ← cfreq(c) − 1
       else
6          s ← Σ_{y∈CT[c]} freq(y)
7          p(w) ← freq(w)/(suffixLen − s) · (freq(c) + 1 − cfreq(c))/(freq(c) + 1)
   else
8      p(w) ← freq(w)/suffixLen
9  encode w with respect to p(w)
10 freq(w) ← freq(w) − 1
11 return
```

us discuss the case $c \in CT$. Summing up both sides of the assignment in line 3, we get

$$\sum_{w \in CT[c]} p(w) = \sum_{w \in CT[c]} \frac{CT[c][w]}{\mathsf{freq}(c)+1} = \frac{\sum_{w \in CT[c]} \mathsf{freq}(w|c)}{\mathsf{freq}(c)+1} = \frac{\mathsf{cfreq}(c)}{\mathsf{freq}(c)+1}.$$

Summing up both sides of the assignment in line 7, we get

$$\begin{aligned}\sum_{w \notin CT[c]} p(w) &= \frac{\sum_{w \notin CT[c]} \mathsf{freq}(w)}{\mathsf{suffixLen} - s} \cdot \frac{\mathsf{freq}(c)+1-\mathsf{cfreq}(c)}{\mathsf{freq}(c)+1} \\ &= \frac{\mathsf{suffixLen} - s}{\mathsf{suffixLen} - s} \cdot \frac{\mathsf{freq}(c)+1-\mathsf{cfreq}(c)}{\mathsf{freq}(c)+1} = \frac{\mathsf{freq}(c)+1-\mathsf{cfreq}(c)}{\mathsf{freq}(c)+1},\end{aligned}$$

where s is calculated in line 6. Summing up the probabilities of all terms w in the case $c \in CT$, we get:

$$\sum_{w \in CT[c]} p(w) + \sum_{w \notin CT[c]} p(w) = \frac{\mathsf{cfreq}(c)}{\mathsf{freq}(c)+1} + \frac{\mathsf{freq}(c)+1-\mathsf{cfreq}(c)}{\mathsf{freq}(c)+1} = 1.$$

Now, we estimate the time complexity of Algorithm 5. The context forward frequencies $\mathsf{freq}(w|c)$ can be calculated together with the generic forward frequencies of symbols in $O(n)$ time. One extra pass over the text is needed to check the inequalities (1) and to construct the context table. This pass can be combined with calculating the frequencies of context terms in line 3 and takes $O(n)$ time. The main loop in lines 5–10 has n iterations, however, summing up $\mathsf{freq}(y)$ in the function Encode-and-Update-the-Model (line 6

of Algorithm 6) may require $O(CT[c])$ time. Therefore, the worst-case time complexity of Algorithm 5 is $O(nc)$, where c is the maximal number of terms in a particular context. This may be greater than $O(nk)$ time of the conventional PPM approach, where k is the maximal context level. However, for all texts we experimented with, the average value c per word does not exceed 8, which implies an acceptable time overhead, comparable to that of the preliminary word-splitting phase.

The context-aware forward decoding algorithm mirrors the encoding Algorithm 5 featured with the meta-character processing technique given in Algorithm 4.

6. Experimental Results

6.1. *Experiments on Word-Based Forward-Looking Compression*

We tested the word-based compression techniques on three texts in English:

- Small – Alice's Adventures in Wonderland;
- Middle-sized – The Bible, King James Version;
- Large – English text from the Pizza&Chili corpus.[2]

A word is defined as a character sequence between two general white spaces; capitalization and punctuation matters, e.g. we differentiate between 'word', 'Word', and 'word'. The results of the net encoding and some statistics are presented in Table 2. The first column indicates the input files. Then follow the size of the file in MB, the total number of words, and the number of unique words. The fifth column gives the value of the static entropy H_0 in bytes calculated via the Shannon formula, which is used here as a baseline. The following three columns present the differences, in bytes, from the values in the fifth column.

- *Backward*: the entropy calculated over frequencies produced by a standard adaptive encoding. Every word occurrence adds 1 to its frequency, and the initial frequency of a word is also 1.
- Algorithm 1: the entropy calculated over frequencies produced by a standard word-based forward-looking method described in Section 2.
- Algorithm 3: forward-looking compression combined with the replacement of the left-most occurrences of infrequent words described in Section 4.

The last column, $|\mathcal{L}|$, presents the size of word frequency information compressed by Algorithm 2.

As can be seen, the size of a compressed frequency sequence is tiny, and the total performance of the forward-looking compression (Alg. 1+$|\mathcal{L}|$) is essentially better than the traditional adaptive backward one. Combined with the word replacement technique, the outperformance of the forward adaptive encoding over the backward becomes even

[2] http://pizzachili.dcc.uchile.cl/texts/nlang/

Table 2
Comparison of text compression methods (in bytes).

	Size MB	*Words* $\times 10^6$	*Words unique*	H_0	*Backward*	Algorithm 1	Algorithm 2	$\lvert\mathcal{L}\rvert$
Small	0.15	0.03	5 311	32 015	+1 380	−1 205	−6 557	81
Middle	3.95	0.77	28 659	907 866	+14 480	−7 764	−38 322	407
Large	200	37.0	836 002	52 805 108	+509 430	−214 429	−1 660 355	2 517

stronger. However, as the text size grows, the relative advantage decreases since the size of a dictionary becomes less significant in relation to a text.

The values in Table 2 are obtained assuming that the list of different words is needed anyway for Information Retrieval and is thus already stored. However, when we consider the application of a compression technique to file archiving, the size of a dictionary should be taken into account. In word-based compression, a dictionary is large and has to be compressed itself using a non-word-based technique. The simplest way to compress a dictionary is to use a universal archiver. It performs much better if words are lexicographically sorted. Moreover, special methods like POM, or more sophisticated methods for large dictionaries, e.g. Ferragina *et al.* (2025), may further improve the compression ratio of the lexicographically ordered dictionary. By contrast, our word replacement method in Section 4 requires ordering the dictionary according to the leftmost appearance of words having the same frequencies in the text, which usually does not coincide with the lexicographical order. These two approaches appear to be alternative and competitive. We tested them both on texts of different sizes in two languages: two English texts from the Canterbury corpus,[3] the Bible in English, Shakespeare's complete works, *The Magic Mountain* by Thomas Mann in German, and the Ukrainian fiction corpus.[4] Also, we archived these texts with one of the most powerful BWT-based, Burrows and Wheeler (1994), universal archivers, bzip2 (which outperforms LZMA2 and Zstd, ultra level 22, on all tested texts). The results are shown in Table 3, where the first two columns present the input files and their uncompressed size, while the next columns contain the following:

– bzip2(T): size of a file compressed by bzip2 archiver.
– POM(D) + Alg. 1(T): word-based forward-looking text compression plus the sequence of word frequencies compressed using RMD codes of Zavadskyi and Anisimov (2020) and the lexicographically ordered dictionary compressed using the POM technique.[5]
– bzip2(D) + Alg. 3(T): The proposed forward-looking compression with word replacement, given in Algorithm 3 plus the dictionary D compressed by bzip2 and the fre-

[3] https://corpus.canterbury.ac.nz/
[4] https://github.com/zavadsky/corpus/
[5] POM outputs concatenated word suffixes and two sequences of integers: suffix lengths and lengths of common prefixes with the preceding words in the dictionary. We compress concatenated suffixes with bzip2 while the two sequences of numbers are compressed separately using static Huffman codes. Also, we should compress the word frequencies given in the order of the lexicographically sorted dictionary. Huffman codes are inefficient for this task, as the set of values to be compressed is too large. Thus, we choose the universal encoder that optimizes its compressed size among Elias, Fibonacci, and RMD codes. For all texts, the RMD code $R_{1,2,4–\infty}$ or $R_{1–3,5–\infty}$ performed the best.

Table 3
Results of full archiving the natural language texts (in bytes).

	Size MB	bzip2	POM(D) + Alg. 1(T)	bzip2(D) + Alg. 3(T)
alice29.txt	0.148	43 144	46 011	**42 576** (−1.3%)
plrabn12.txt	0.460	145 310	146 165	**141 591** (−2.6%)
The Bible	3.95	**844 818**	958 420	959 525 (+12%)
Shakespeare	5.19	1 473 411	1 401 683	**1 396 240** (−5.5%)
German	1.01	293 411	280 671	**277 944** (−5.5%)
Ukrainian	15.2	3 061 124	2 976 830	**2 899 135** (−5.6%)

quency values compressed via Algorithm 2. Also, the outperformance over the bzip2 is shown in percents.

In all schemes, the bzip2 has been used on the 9th *ultra* compression level. The forward-looking compression results correspond to arithmetic encoding. The source code can be found at https://github.com/zavadsky/forward.

The best compression performance is highlighted in bold. The results show that for all texts, except for the Bible, the best choice is combining Algorithms 2 and 3 for compression of the frequency list and the text itself, respectively, with bzip2-compression of the non-lexicographically ordered word list. The Bible contains many highly repetitive words, which is not beneficial for our word replacement method, intended for the efficient representation of non-frequent alphabet elements. By contrast, for the Ukrainian text, the combination of Algorithm 2 and 3 proves to be the most efficient, as Ukrainian is an inflectional language and thus exhibits a much larger number of distinct, low-frequency words.

The English language has indeed the ability to express a term or concept in precise words, with only a small number of variants, e.g. eat, eats. The conjugation of verbs in French provides many more terms, e.g. mange, manges, mangeons, mangez, mangent. Similarly, the practically unlimited possibility of concatenating terms in German to form Komposita generates many different words, like Haus, Haustor, Hausnummer, Holzhaus, Bauhaus, Hausbau, Bauhausepoche, Hausbaugesellschaft, all of which are translated into phrases of several terms. While for English, these phrases would increase the count of the terms house or home, they are split over many more terms in German. For the German text, the combination of Algorithms 2 and 3 also proves to be highly efficient.

This phenomenon will be even more accentuated in highly inflected languages like Hebrew, whose morphology provides the possibility of prefixing a root by articles and prepositions like the, and, to, ..., and suffixing it by possessive pronouns, like mine, his, ... A Hebrew noun can thus have hundreds of variants, and a verb even thousands, which will evidently lead to the appearance of many terms with very low occurrence rates.

This principle of getting an advantage from the appearance of rare terms also applies when assessing the efficiency of our approach for other types of data: the more low-frequency symbols the data contain, the greater the benefits of our method, unless the dictionary grows so large that compressing it becomes more critical than compressing the text itself.

In summary, the word-level forward-looking adaptive compression, together with dictionary-aware text preprocessing techniques, demonstrate impressive compression ratios for different types of natural language texts. Combining this word-level approach with character-level BWT-based dictionary compression has been shown to be usually more beneficial than using only the latter for the whole text.

6.2. *Experiments on Context-Aware Forward Compression*

To evaluate the compression performance of the proposed context-aware forward compression, we considered the following prose and poetry masterpieces in English:

– *Alice* – Alice's Adventures in Wonderland;
– *Sonnets* – Shakespeare's sonnets;
– *plrabn12.txt* – Paradise Lost, a poem by John Milton;
– *Harry Potter* – Harry Potter and the Philosopher's Stone;
– *lcet10.txt* – workshop on electronic text proceedings, the file from Canterbury corpus.

The file sizes and optimal values for th_f and th_g for both the mainstream and the punctuation stream appear in Table 4. Note that, unlike the previous experiments, we do not compress text corpora since different files in a corpus may have different optimal threshold values.

Table 4
Test data for context-aware forward compression.

	Size KB	Mainstream		Punctuation	
		th_f	th_g	th_f	th_g
Alice	152	2	14	3	14
Sonnets	100	4	20	3	15
plrabn12	467	4	18	3	14
Harry Potter	429	2	15	2	12
lcet10	416	3	14	3	14

We, therefore, split the characters of each data file into two kinds of words, regular words and punctuation words as follows:

– a word in the *main text stream* is a longest contiguous sequence of alphanumeric characters;
– for a prose text, each sequence of characters between the words of the mainstream is considered as a word of the *punctuation stream*;
– for poetry, a sequence of spaces and punctuation characters in each line of a text is considered a single word of the punctuation stream; groups of two or more consecutive spaces are represented with unique predefined symbols.

Table 5
Results of full archiving the natural language texts (in bytes).

	Text	*Alice*	*Sonnets*	*plrabn12*	*Harry Potter*	*lcet10*
mainstream	comp-txt	22591	16646	86138	76695	59703
	comp-dic	10065	10735	33923	26798	24661
	con-freq	1899	289	2248	5416	4966
	uncon-freq	84	66	137	140	117
punctuation	comp-txt	5735	1543	9492	4817	10780
	comp-dic	688	921	4812	355	664
	con-freq	73	57	40	213	197
	uncon-freq	38	28	55	40	47
Total		41173	**30285**	**136845**	**114474**	**101135**
PPMd		**38969**	30361	137386	118529	102519
BZip2		43144	31805	144011	119600	107683

Consider the following prose fragment

␣␣‘Well!’␣thought␣Alice␣to␣herself,␣‘after␣such␣a␣fall␣as␣this,␣I EOL
shall␣think␣nothing␣of␣tumbling␣downstairs!’

Its punctuation stream contains the following words:
␣␣‘; !’␣; ␣; ␣; ␣; ,␣‘; ␣; ␣; ␣; ␣; ␣; ,␣; EOL; ␣; ␣; ␣; ␣; ␣; !’.

In a poetry example from Shakespeare's sonnet

Music␣to␣hear,␣why␣hear’st␣thou␣music␣sadly?
Sweets␣with␣sweets␣war␣not,␣joy␣delights␣in␣joy:

there are two punctuation words: ␣␣,␣␣␣␣␣?; ␣␣␣␣␣,␣␣␣␣␣:.

Both streams are encoded independently, applying the context-aware forward-looking technique described in Section 5. During the decoding, words of a mainstream are interleaved with symbols of a punctuation stream.

We have also tested the ‘backspace character’ approach discussed in Klein and Shapira (2009). It relies on the fact that a single space follows most words. Therefore, these single spaces could be eliminated from the punctuation stream, and words followed by non-space characters could be marked with a special ‘backspace’ symbol that never occurs in a text. The decoder then takes a word from the punctuation stream and puts it after a word from the mainstream, only if the latter is followed by the backspace symbol. Otherwise, the decoder appends a mainstream word with a single space. This approach appeared to be efficient only for the *Harry Potter* file, as it is not split into lines by EOL characters artificially and thus contains many more single spaces than other files. Thus, Table 5 demonstrates the compression of the Harry Potter file with the ‘backspace character’ approach and other files without it.

To compress context frequency tables, we use different multi-delimiter codes. Each table $CT[c]$ consists of the sequence of terms and the sequence of respective frequencies.

Terms can be indexed by their positions in the dictionary appended with meta-characters. We sort $CT[c]$ in the order of ascending term indices. Then, we encode two sequences of numbers: (1) the differences between consecutive term indices and (2) the differences between term frequencies and the minimal frequency in $CT[c]$. Also, we use differential encoding to encode the sequence of context terms, $\{c\}$.

The results of full archiving appear in Table 5 in which the upper part corresponds to the mainstream, and the lower part to the punctuation signs. The numbers report the sizes, in bytes, of the compressed text, the compressed dictionary, the context frequencies, and the unconditional forward frequencies, denoted by comp-txt, comp-dic, con-freq and uncon-freq, respectively. As can be seen, an improvement in the performance ratio is evident for the *Alice* and *plrabn12* texts when comparing the results in Table 5 with those in Table 3. Moreover, our method performs a bit better than PPMd for all files, except for the shortest one, and essentially better than bzip2. The largest improvement (about 3.5%) is achieved for *Harry Potter*, which is a 'pure' text file without artificial EOL characters and with a relatively low number of punctuation signs.

7. Conclusion

This paper presents an innovative adaptive coding method that aligns the forward-looking approach with word-based alphabets, improving natural language text compression efficiency. The paper's key contributions include:

1. Extending the forward-looking approach to accommodate word-based alphabets;
2. Introducing an efficient encoding for the header information consisting of word frequencies;
3. Replacing low-frequency words in the text with higher-frequency meta-characters and using an alternative ordering for word alphabets;
4. Adapting the order-one PPM algorithm for forward adaptive encoding on word-based alphabets.

The experiments show that, in terms of compression efficiency, the new approach can outperform existing ones, even when the entire list of words is considered to be a part of the header. Furthermore, the proposed PPM style method outperforms bzip2 by 4–6% and is, in most cases, even better than the powerful PPMd-based compressor. Notably, the integration of the PPM framework with forward adaptive encoding effectively resolves the zero-frequency problem, which constitutes a major limitation of the classical PPM method. Furthermore, the proposed compression schemes are shown to be efficient in terms of both compression performance and algorithmic time complexity, while satisfying acceptable asymptotic bounds.

Experimental evaluation and optimization of encoding and decoding times are left for future work, along with the integration of our approach with other dictionary-based compression and context selection techniques. This will enable the proposed methods to be incorporated into high-performance data compression and information retrieval systems.

References

Adiego, J., Martínez-Prieto, M.A., de la Fuente, P. (2009). High performance word-codeword mapping algorithm on PPM. In: *Data Compression Conference, DCC 2009*, Snowbird, UT, USA, pp. 23–32.

Aljehane, N.O., Teahan, W.J. (2017). Word-based grammars for PPM. *International Journal of Advanced Computer Science and Applications*, 8(10), 1–11.

Avrunin, R.M., Klein, S.T., Shapira, D. (2022). Combining forward compression with PPM. *SN Computer Science*, 3(239), 239.

Baruch, G., Klein, S.T., Shapira, D. (2017). A space efficient direct access data structure. *Journal of Discrete Algorithms*, 43, 26–37.

Baruch, G., Klein, S.T., Shapira, D. (2020). Accelerated partial decoding in wavelet trees. *Discrete Applied Mathematics*, 274, 2–10. https://doi.org/10.1016/j.dam.2018.07.016.

Bratley, P., Choueka, Y. (1982). Processing truncated terms in document retrieval systems. *Information Processing and Management*, 18(5), 257–266.

Burrows, M., Wheeler, D.J. (1994). *A Block-Sorting Lossless Data Compression Algorithm*. Technical Report 124. Digital Equipment Corporation.

Cleary, J., Witten, I. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4), 396–402.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2), 194–203.

Ferragina, P., Rotundo, M., Vinciguerra, G. (2025). Two-level massive string dictionaries. *Information Systems*, 128, 102490

Fruchtman, A., Gross, Y., Klein, S.T., Shapira, D. (2021). Backward weighted coding. In: Bilgin, A., Marcellin, M.W., Serra-Sagristà, J., Storer, J.A. (Eds.), *31st Data Compression Conference, DCC 2021*, Snowbird, UT, USA, March 23–26, 2021. IEEE, pp. 93–102.

Fruchtman, A., Gross, Y., Klein, S.T., Shapira, D. (2022). Weighted forward looking adaptive coding. *Theoretical Computer Science*, 930, 86–99.

Fruchtman, A., Gross, Y., Klein, S.T., Shapira, D. (2023). Bidirectional adaptive compression. *Discrete Applied Mathematics*, 330, 40–50.

Grossi, R., Vitter, J.S. (2005). Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2), 378–407. https://doi.org/10.1137/S0097539702402354.

Grossi, R., Gupta, A., Vitter, J.S. (2003). High-order entropy-compressed text indexes. In: *SODA '03: Proceedings of the 14th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA)*, pp. 841–850.

Huffman, D. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9), 1098–1101.

Klein, S.T., Saadia, S., Shapira, D. (2020). Forward looking Huffman coding. *Theory of Computing Systems*, 65, 593–612.

Klein, S.T., Shapira, D. (2009). On the usefulness of backspace. In: *Proceedings of the Prague Stringology Conference*, 2009, Prague, Czech Republic, August 31–September 2, pp. 80–89.

Moffat, A. (1989). Word-based text compression. *Software: Practice and Experience*, 19(2), 185–198.

Navarro, G. (2016). *Compact Data Structures: A Practical Approach*. Cambridge University Press, Cambridge, UK.

Seward, J. (2019). bzip2 and libbzip2, version 1.0.8. A program and library for data compression.

Vitter, J.S. (1987). Design and analysis of dynamic Huffman codes. *Journal of the ACM*, 34(4), 825–845.

Zavadskyi, I.O., Anisimov, A.V. (2020). Reverse multi-delimiter compression codes. In: Bilgin, A., Marcellin, M.W., Serra-Sagristà, J., Storer, J.A. (Eds.), *Data Compression Conference, DCC 2020*, Snowbird, UT, USA, March 24–27, 2020. IEEE, pp. 173–182.

Zavadskyi, I.O., Klein, S.T., Shapira, D. (2024). Word-based forward coding. In: *Data Compression Conference, DCC 2024*, Snowbird, UT, USA, pp. 352–361.

Zipf, G.K. (1949). *Human Behavior and the Principle of Least Effort: An Introduction to Human Eoclogy*. Addison-Wesley Press, Cambridge, UK.

I. Zavadskyi, PhD, DSc, professor at the Department of Mathematical Informatics, Faculty of Computer Science and Cybernetics, Taras Shevchenko National University of Kyiv, Ukraine. He received his PhD in computer science from the National Academy of Sciences of Ukraine in 2001 and his DSc degree in 2020. He is the author of more than 100 research papers and 25 textbooks and instructional manuals in computer science. His current research interests include information and coding theory, data compression, database management systems, and e-learning technologies. Prof. Zavadskyi is a member of the programme committee of the *Data Compression Conference* (DCC).

S.T. Klein received a BSc in mathematics and statistics from the Hebrew University of Jerusalem in 1978, an MSc degree in applied mathematics, and a PhD in computer science, both from the Weizmann Institute of Science in Rehovot, Israel, in 1981 and 1988, respectively. He then spent three years as a visiting assistant professor at the University of Chicago, and returned in 1990 to Israel, where he has been with the Department of Computer Science of Bar Ilan University in Ramat Gan, Israel.

Prof. Klein's research focuses on text processing and, in particular, data compression. He has worked on two of the largest full-text information retrieval systems: the Responsa Project at Bar Ilan and the Trésor de la Langue Française at the University of Chicago. He has authored two books, more than a hundred refereed journal and conference papers, and thirteen patents. He is currently a professor emeritus at Bar Ilan University, where he has been the chair of the Computer Science Department for several years.

D. Shapira received the BSc (cum laude), MSc (magna cum laude), and PhD degrees in mathematics and computer science from Bar Ilan University, Israel, in 1993, 1996 and 2001, respectively. She was then a postdoctoral fellow with Brandeis University, Waltham, MA, USA. She is currently a professor at Ariel University, Israel. She has coauthored over 50 refereed articles in scientific journals and over 70 in conference proceedings, and has been granted 4 patents. Her research interests include data compression, algorithm development and analysis and scheduling theory. She has served as the head of the Department of Computer Sciences, and is currently the head of the MSc Degree Program. Prof. Shapira has been a member of the programme committee of the *Data Compression Conference* (DCC) since 2011.