UDC 519.72: 519.688

doi.org/10.15407/fmmit2023.36.007

Space-Time Optimization in Natural Language Text Compression

Anatoly Anisimov[1], Igor Zavadskyi[2]

[1] Doctor of Physical and Mathematical Sciences, professor, Taras Shevchenko National University of Kyiv, 4d Glushkov Ave., 03022, Kyiv, Ukraine, e-mail: avatatan@gmail.com

[2] Doctor of Physical and Mathematical Sciences, associate professor, Taras Shevchenko National University of Kyiv, 4d Glushkov Ave., 03022, Kyiv, Ukraine, e-mail: ihorzavadskyi@knu.ua

In this paper, we discuss various problems arising in space and time optimization of natural language text compression methods. We define a new class of variable-length universal data compression codes with multiple delimiters — the Reverse Multi-Delimiter (RMD) codes. They are synchronizable, allow us to perform fast Boyer-Moore-style search in a compressed file, and at the same time provide the best compression ratio among all codes of a discussed class. In combination with a special technique of preprocessing a natural language text and its dictionary, they improve the performance of modern powerful achievers. Also, we construct a very fast decoding algorithm for RMD-codes operating almost at the same speed as (s,c)-dense codes and times faster than Fibonacci codes decoding. The provided experiments show that RMD-codes occupy a very attractive position by the means of space/decoding time tradeoffs in natural language text compression.

Keywords: word-based; compression; archiver; code; multi-delimiter

Introduction. Compression of large textual databases is one of the key elements of modern information retrieval systems. We focus on methods operating words as atomic symbols since they rely on partitioning texts in most natural semantic units and thus provide the best compression ratios. Well-known classical solutions based on entropy encoding, such as Huffman codes, can be applied to word-level text compression and operate close to the theoretical limit defined by Shannon's entropy. However, not only the compression ratio matters but also such features as fast search in compressed data, high decoding speed, and code robustness in the sense of limiting possible error propagation. As is known, Huffman codes are not well suited for such requirements.

An alternative approach stems from the use of variable-length codes with delimiters, such as Fibonacci [1] or (*s*,*c*)-dense codes (SCDC) [2] developed in 2003. Delimiters are special bit sequences denoting the beginning or the end of a codeword. This implies the synchronizability of a code and allows the fast Boyer-Moore-style pattern search in a compressed file. Of course, these properties are achieved at the cost of compression ratio. However, on a word-based alphabet, the price is not too high as it contains rather more elements than a character-based and distribution of their frequencies is flatter. This equates the compression performance of different codes.

Apart from compression ratio, another important quantitative characteristic of a compression code is the speed of decoding. It values higher than the encoding speed since the encoding is often performed offline, while the decoding, should be done 'on the fly' in most applications, e.g. in search engines. SCDC are specifically intended for

fast decompression as their codewords comprise the whole number of bytes and can be processed in a byte-by-byte manner. An SCDC-encoded text can be decoded twice faster as Fibonacci-encoded text even if we apply the fastest known today decoding method for Fibonacci codes developed in 2009 [3]. Thus, from the beginning of 2000s, Fibonacci and (*s*,*c*)-dense codes can be considered as the two most attractive options among codes allowing the synchronizability and fast compressed search.

In recent years, we developed a new type of variable-length data compression codes with delimiters, which provide a better compression ratio than Fibonacci codes and can be decoded almost at the same speed as (*s*,*c*)-dense codes. They are Multi-Delimiter (MD) codes [4] and Reverse Multi-Delimiter (RMD) codes [5]. In this paper, we overview the main properties of these codes and their application to natural language text compression.

1. Codes Definition and Construction

The family of Fibonacci codes is parameterized with the length of a delimiter which is the series of ones. In natural language text compression, the code Fib3 has the best compression ratio, while Fib2 and Fib4 are much worse. Likely, the performance of the Fibonacci codes can be improved further if we found the compromise between Fib3 and Fib2/ Fib4. E.g., it can be done by using more than one delimiter in the code. However, it is impossible for delimiters 1...1 since one delimiter is a prefix of another.

By contrast, a code can contain several suffix delimiters of the form 01 . . . 10. A variant of such code with multiple delimiters (MD-code) has been introduced and thoroughly studied in [4]. It was proved that any multi-delimiter code is uniquely decodable, complete, and universal. Compared with byte and Fibonacci codes, MD-codes demonstrate a much better text compression ratio.

Nonetheless, for MD-codes, the important problem of optimal dictionary indexing was not clarified. During encoding, a mapping *word of a text* → *codeword* is used, where words of a text are sorted in descending order of their frequencies, while codewords are sorted in ascending order of their lengths. The decoding process is reversed. For fast decoding, a data structure with low access time should be used to store the words of a text, e.g. an array with integer indices. However, it requires constructing an invertible monotonous mapping of the set of integer indices to the set of codewords. Although for MD-codes it seems problematic, this issue can be resolved by a simple trick: writing bit representations of MD-codewords from right to left. This way we obtain a non-prefix but uniquely decodable code with a monotonous bijection to the set of natural numbers — the *Reverse Multi-Delimiter* code.

Let us define an RMD-code. Assume $m_1, \dots, m_t$ is the ascending sequence of natural numbers. The codeword set of the RMD-code $R_{m_1,\dots,m_t}$ consists of codewords of the form $01^{m_i}, i = \overline{1,t}$, and also codewords that:

- start from the sequence $01^{m_i}0, i = \overline{1,t}$ and do not contain any of these sequences anywhere else in the codeword;
- do not end with a sequence $01^{m_i}, i = \overline{1,t}$.

The bit sequences $01^{m_i}0$ can be considered as delimiters. Although such

delimiters do not belong to codewords of the form 01^{m_i}, these codewords constitute delimiters together with the leading 0 of the next codeword. In this paper, we use only RMD-codes with an infinite number of delimiters as they demonstrate a better compression ratio. By $R_{m_1,\dots,m_t-\infty}$ we denote the RMD-code having the delimiters with $m_1, \dots, m_t$ or more of ones. E.g. $R_{2,4-\infty}$ is the code with delimiters 0110 and 01^t0, where $t \geq 4$, while $R_{2-\infty}$ is the code with all delimiters consisting of 2 or more ones.

2. Fast Decoding Algorithm

Any Reverse Multi-Delimiter code can be considered a regular language and thus recognized by the finite automaton. However, it processes a text bit-by-bit, which is quite slow. The main idea of a fast decoding algorithm is a "quantification" of a decoding automaton so that it reads bytes of a code and produces the corresponding output numbers. Below we present the byte-aligned decoding algorithm for RMD-codes that operates 25–35% faster than the decoding algorithm given in [5].

The notations are the following. Assume we have processed some byte of a code. The pointer *ptr* is a combination of the decoding automaton state *a* and the number *l* of already decoded bits of the last codeword. It can be calculated as follows: $ptr = a \cdot l_{max} + l$, where l_{max} is the maximal possible bitlength of a codeword. If we multiply *ptr* by 256 and add a current byte of the text, we get the index *x* of lookup tables (line 3). The lookup tables are: *Pointers*[*x*] – the pointer for decoding the next byte; *Numbers*[*x*] – a 64-bit number, which consists of four 16-bit numbers we get after decoding a current byte; *c*[*x*] – the number of codewords fully decoded during processing the current byte.

Also, assume the dictionary contains no more than 2^{16} words (this case can be easily generalized). Then, 4 sequential decoded integers can be output with one assignment of a 64-bit value (line 5), and a byte of a code can be fully processed at one iteration of the decoding loop without time-consuming conditional statements.

Algorithm 1. Byte-aligned decoding of the RMD-code for short texts

input: RMD-bitstream composed of bytes, *Code*[1…*n*].
output: Array of numbers, *Out*.

1. *ptr* ← 0; *k* ← 0; // initialize the pointer and output index
2. **for** *i* ← 1 **to** *n* **do**
3. *x* ← 256*ptr* + *Code*[*i*]
4. *ptr* ← *Pointers*[*x*]
5. *Out*[*k*,…,*k*+3] ← *Numbers*[*x*] + *tr*
6. *k* ← *k* + *c*[*x*]
7. *tr* ← *Out*[*k*] // value obtained from the partially decoded codeword

3. Experiments

To estimate the compression ratio and the decoding time we encoded three English texts of different sizes: *small* (The Bible, 3.83 MB), *middle-sized*: articles randomly taken from Wikipedia (116 MB), and *large*: the first half of the largest file from the Pizza&Chili Corpus (512MB). We chose RMD-codes that have the best compression ratio: $R_{2-\infty}$ for small text and $R_{2,4-\infty}$ for middle-size and large. For comparison, we chose the Fibonacci code Fib3 and the byte-aligned (*s*, *c*)-dense codes. For SCDC we

chose parameters providing the best compression ratio for each text.

In Fig. 1 small, middle-sized, and large texts are shown as small, middle-sized, and large markers respectively. As seen, RMD-codes outperform Fibonacci code Fib3 both in decoding speed and compression ratio. Also, they outperform essentially SCDC in compression ratio and even they are a bit faster than SCDC in decoding small text.

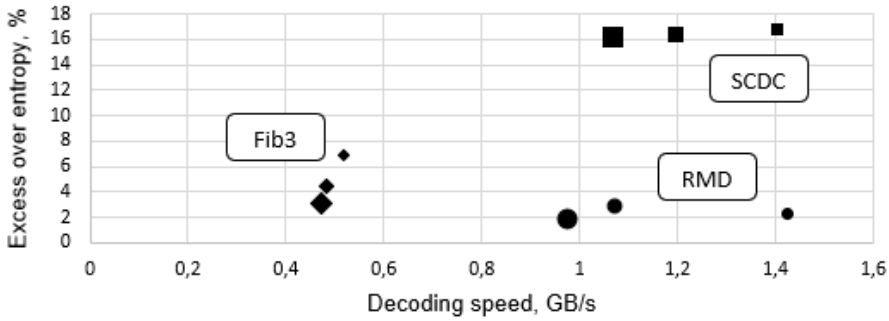

Fig. 1. Experiments on compression and decoding

Let us note that the mentioned compression ratios characterize codes themselves, without auxiliary structures, such as dictionaries required for encoding and decoding. In contrast to character-level, in word-level text compression, a dictionary is a more significant part that should be stored together with the compressed file. The uncompressed word-level dictionary for 1 GB English text occupies about 2.5–3% of the text itself and about 5% of 100 MB text. If we compare the size of a compressed dictionary with the compressed text, the percentage becomes even higher. However, a text already consists of dictionary elements, and thus we can save space by special marking dictionary elements in the text. Also, some other regularities of natural language can be exploited. This leads us to construct a word-level text preprocessor, that is placed in front of a standard postcompressor to improve compression ratio.

Notably, the 3-level schema consisting of mentioned preprocessor, RMD-codes, and standard archiver as postcompressor allows us to improve the compression ratio of archivers itself. The experiments were conducted for the 1GB text from the Pizza&Chilie corpus. The original text consists of 1,073,741,824 bytes, 189,528,100 words, 2,523,827 unique words. The file word-level entropy is 273,284,721 bytes, 13.535 bits per word. The results are shown in Table 1.

Table 1

Efficiency of using RMD-preprocessing with further archiving

Text	Original text	$R_{2-\infty}$	$R_{2,4-\infty}$	$R_{3-\infty}$
7z	258,428,183	258,705,282	257,252,378	256,751,472
Rar	290,584,311	264,645,314	261,948,637	262,563,336
Gzip	405,714,638	277,592,303	273,990,536	274,736,358

We applied 7z, version 16.02, 64-bit, RAR, and gzip archivers in the maximum compression mode to the original and RMD-encoded texts with all mentioned above preprocessing transformations. As observed, preliminary RMD- encoding significantly

improves RAR- or gzip-compression ratio, by more than 10% or 50% respectively. LZMA-based 7z compresses texts much better than RAR or gzip and it is recognized as one of the most powerful modern archivers. However, even in this case RMD-codes open room for improvements. For example, the 7z-archiving of the $R_{3-\infty}$-encoded text produces 0.7% smaller file than archiving this text without the RMD-preprocessing. It is interesting to note that not archived RMD-encoded files are 32−34% smaller than files archived with gzip. Considering the possibility of compressed search and ultra-fast decoding, RMD-codes can be considered as a preferred format to store large textual databases compared with gzip, which decodes texts 3−4 times slower.

Conclusion. The reverse multi-delimiter (RMD) compression codes can be used as a key element for word-based natural language text compression as well as for the compact representation of unbounded integer sequences. We constructed a very fast byte-aligned decoding algorithm based on lookup tables, which is comparable with the (*s*,*c*)-dense byte-aligned decoding method. Given a good compression ratio, the RMD-codes provide an attractive point in the trade-off between the compression ratio and the decoding speed in natural language text compression. Together with the special word-level text preprocessing technique, the RMD-codes can serve as a preprocessing tool improving the compression ratio of known archivers.

References

[1] *A. Apostolico and A. S. Fraenkel*. Robust transmission of unbounded strings using Fibonacci representations, IEEE Trans. Inf. Theory, vol. 33, 1987, pp. 238–245.

[2] *N. Brisaboa, A. Farina, G. Navarro, and M. Esteller*. (s,c)-dense coding: an optimized compression code for natural language text databases, in: Proc. Symposium on String Processing and Information Retrieval, ser. LNCS, no. 2857. SVB, 2003, pp. 122–136.

[3] *S. T. Klein and M. Ben-Nissan*. On the usefulness of fibonacci compression codes, Computer Journal, vol. 53, no. 6, pp. 701–716, 2010.

[4] *A. Anisimov and I. Zavadskyi*. Variable-length prefix codes with multiple delimiters, IEEE Transactions Information Theory, vol. 63, no. 5, 2017, pp. 2885–2895.

[5] *I. Zavadskyi and A. Anisimov*. Reverse multi-delimiter compression codes, in: 2020 Data Compression Conference, 2020, pp. 173–182.

Ємнісно-часова оптимізація у стисканні природномовних текстів

Анатолій Анісімов, Ігор Завадський

У роботі розглянуто різноманітні аспекти оптимізації методів стискання природномовних текстів за ємністю та часом. Визначено новий клас стискальних кодів змінної довжини з кількома роздільниками — реверсні мультироздільникові коди (РМР). Вони є синхронізовними, дають можливість виконувати швидкий пошук типу Бойера-Мура у стиснутому файлі й водночас забезпечують найкращий коефіцієнт стискання серед кодів описаного типу. Як засіб передобробки тексту ці коди покращують характеристики найпотужніших сучасних архіваторів. Також було запропоновано надшвидкий алгоритм декодування РМР-кодів, що працює майже з тією самою швидкістю, що й декодування (s,c)-щільних кодів і в рази швидше, ніж декодування кодів Фібоначчі. Експерименти свідчать про високу часово-ємнісну ефективність РМР-кодів у стисканні природномовних текстів.

Received 14.03.23