Low-Complexity Compression with Random Access

Srikanth Kamparaju, Shaik Mastan, Shashank Vatedka
Department of Electrical Engineering, IIT Hyderabad

***Abstract*—We investigate the problem of variable-length compression with random access for stationary and ergodic sources, wherein short substrings of the raw file can be extracted from the compressed file without decompressing the entire file. It is possible to design compressors for sequences of length n that achieve compression rates close to the entropy rate of the source, and still be able to extract individual source symbols in time $\Theta(1)$ under the word-RAM model. In this article, we analyze a simple well-known approach used for compression with random access. We theoretically show that this is suboptimal, and design two simple compressors that simultaneously achieve entropy rate and constant-time random access. We then propose dictionary compression as a means to further improve performance, and experimentally validate this on various datasets.**

I. Introduction

In this paper, we investigate the problem of variable-length compression of stationary and ergodic random sequences. A classical result in information theory states that the minimum rate of compression for any stationary and ergodic source is equal to the entropy rate [1]. This is a well-studied problem, and there exist very efficient universal [2], [3] compressors that achieve the entropy rate.

However, these (traditional) compressors have been fundamentally designed for purposes of archival. In particular, they do not allow one to process the source directly in the compressed domain. In recent years, we have witnessed a massive increase in the amount of generated data [4], [5], [6], particularly genomic and astronomical data and current solutions are insufficient to handle these [7], [8]. In particular, there is a growing requirement for efficient compression algorithms that allow one to read, write, and search through short fragments of the source by directly operating in the compressed domain.

For instance, let us consider the scenario where we have a large database of genomes that are stored on the cloud. In order to save space, we would store the database in compressed form. However, when running experiments/analyzing this data, we are typically interested in only extracting short substrings (for e.g., specific genes) from various sequences. If we use a general-purpose compressor such as zip/gzip/7zip (which are based on the Lempel-Ziv algorithms), then we would have to decompress the entire database every time we want to access a substring. This is computationally very expensive. It is therefore necessary to design compression algorithms that allow one to extract short substrings of the source sequence without decompressing the entire compressed file. This problem is referred to as compression with random access, or locally decodable compression in the literature.

The problem of compression with random access can be stated formally as follows: Given a random source sequence X^n which is stationary and ergodic, design a universal compressor that achieves a compression rate that approaches the entropy rate, and a local decompression algorithm that is able to recover any X_i from the compressed sequence in time $O(1)$, a constant that is independent of n. We operate under the word-RAM model [9] of computation wherein it is assumed that a single read/write or arithmetic/logical operations on a block of w symbols (called the word size) takes unit time. As is common in the literature, we assume that $w = \Theta(\log n)$, since $\log n$ bits are necessary to even address the symbols of an n-length sequence.

The requirement of random access mentioned above ensures that substrings of length $m \ll n$ can be extracted in time $O(m/w)$, instead of the $O(n/w)$ time which can be achieved using classical algorithms such as LZ77 and LZ78.

The widely used approach used to solve the problem of compression with random access is the following: Partition the source/raw file into smaller blocks, and compress each block independently using a general purpose compressor such as zip. In order to recover a substring, it is sufficient to only decompress the corresponding blocks. Despite there being more sophisticated, theoretically near-optimal algorithms known in the literature [10], [11], the above solution is widely used [12], [13] due to its simplicity and low complexity.

1) Contributions: In this article, we perform a careful theoretical analysis of the above scheme in terms of the achievable compression length and random access time. We show that this approach is in fact suboptimal and it is impossible to simultaneously guarantee a rate approaching entropy rate as $n \to \infty$ and $O(1)$ random access time (the time required to extract any X_i). We then propose two practical solutions, which simultaneously achieve entropy rate and constant random access time. Both follow a similar approach of partitioning X^n into blocks and compressing each block independently, but vary in the way these compressed blocks are stored and addressed. These are very simple to implement, and we also compare their performance with experimental results. Our theoretical analysis shows that the proposed approaches achieve entropy, but the redundancy (difference between the compression rate and entropy rate) is much lower than that obtained by compressing the entire sequence using Lempel-Ziv. We then propose dictionary compression as a solution to improve the rate. While we do not have a theoretical analysis

The work of S. Vatedka was supported by a seed grant from IIT Hyderabad and and a Start-up Research Grant (SRG/2020/000910) from SERB, India.

978-1-7281-8895-9/20/$31.00 © 2022 IEEE

for the latter, we demonstrate using experiments that dictionary compression can indeed provide significant gains for low-complexity compression with random access.

Related works

Compression with random access under the word-RAM model has been studied extensively by the computer science community; see, *e.g.*, [14], [15], [16], [17], [18], [19] and the references therein. Here, the problem is to design compressors for the worst-case input sequence x^n. These works show that it is theoretically possible to achieve compression rates that asymptotically approach the m-th order empirical entropy rate of the input sequence, and still be able to recover individual symbols in $O(1)$ time. Despite these sophisticated algorithms, the naive approach outlined in the introduction still remains the most popular scheme used in practice because of its simplicity and very low complexity. The goal of this work is to understand why this is the case and what improvements can be made to improve performance while retaining the simplicity and low complexity.

In the information theory community, locally decodable source coding of *random sequences* was first explored in [20], [21]. Here, the random access performance of a scheme is measured in terms of the *local decodability,* or the number of compressed bits needed to recover a single source symbol. This is also referred to as the bitprobe complexity of a compressor [22]. Subsequent works *et al.* [23], [24] designed compressors for memoryless sources that achieve a rates of $H(X) + \varepsilon$ and local decodability of $d_{\rm wc} = \Theta(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. Mazumdar [25] also showed that for non-dyadic sources, $d_{\rm wc} = \Omega(\log(1/\varepsilon))$ for any compression scheme that achieves rate $H(X)+\varepsilon$. Tatwawadi *et al.* [26] extended the achievability result to Markov sources and provided a universal scheme that achieves $d_{\rm wc}(1) = \Theta(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$. Both these works used the bitvector compressor of [22] which is a nonexplicit construction based on expander graphs. Pananjady and Courtade explored variable-length compression of sparse sequences [27] and gave upper and lower bounds on the achievable rate as a function of local decodability. The works [28], [29] considered simultaneous local decodability and update efficiency.

II. Modifying general purpose compressors to achieve random access

We assume that the source/raw file is a random sequence X^n and is stationary and ergodic. Since every ergodic source can be approximated by a stationary m-th order time-homogeneous Markov chain for some finite m, we will henceforth assume that X^n is indeed a stationary m-th order time-homogeneous Markov chain for some[1] $m = O(1)$. We do not however assume that m or the source distribution are known.

We now describe a simple (and the most common) approach to obtaining random access. The raw sequence X^n is split into multiple blocks of b symbols each, and each block is compressed independently using a general purpose compressor such as[2] LZ78 [3]. The compressed blocks are concatenated and stored as a single bit string. Since the compressed chunks could have varying sizes, we also require an addressing mechanism to efficiently identify the beginning and end of each compressed block. We will explore this in detail in the forthcoming subsections.

Let us introduce some notation. A sequence of length n, say $X_1, X_2, \ldots, X_n$ is denoted X^n. A subsequence, say, $(X_i, X_{i+1}, \ldots, X_j)$ is denoted X_i^j. The m-th order entropy rate of a stationary source X^n, denoted $H_m(X^n)$, is defined to be equal to $H(X_{m+1}|X_m, \ldots, X_1)$. This is equal to the entropy rate of X^n if X^n is an m-th order stationary Markov chain. Given a compressor for an m-th order stationary Markov chain X^n that achieves expected length equal to k, the *redundancy* of the compressor is defined as $k/n - H_m(X^n)$.

Let $X^b(i) = X_{(i-1)b+1}^{ib}$ denote the i'th block of b consecutive symbols of X^n. Suppose that $C^{k_i}(i)$ is the binary string obtained by compressing $X^b(i)$ using LZ78. Furthermore, let $k = \sum_{i=1}^{n/b} k_i$, and $C^k = (C^{k_1}(1), \ldots, C^{k_{n/b}}(n/b))$ denote the concatenation of all the compressed blocks. Let us further denote $s_i = \sum_{j=1}^{i-1} k_j + 1$ and $e_i = \sum_{j=1}^{i} k_j$ to be the locations of the start and end of the i'th compressed block in the concatenated string. Given s_i and e_i, we can recover $X^b(i)$ by running the LZ78 decompressor on $C_{s_i}^{e_i} = C^{k_i}(i)$. It therefore suffices to store a data structure from which we can store and retrieve (s_i, e_i) efficiently.

We will frequently use the following landmark result:

Theorem 1 ([31])**.** *The expected length of the LZ78 compressed string for a source X^n distributed according to an m-th order stationary Markov chain is equal to*

$$\mathbb{E}[k_{LZ78}] = nH_m(X^n) + O\left(\frac{n}{\log n}\right).$$

The redundancy of the LZ78 compressor therefore scales as $O(1/\log n)$.

A. TAB−LZ: Tabulating $\{e_i : 1 \leqslant i \leqslant n/b\}$

The naive approach is to store a table $E = (e_1, e_2, \ldots, e_{n/b})$. Since $s_i = e_{i-1} + 1$, we can obtain (s_i, e_i) by making two accesses to the table E. Each e_i require $\log n$ bits of space, and therefore (s_i, e_i) can be obtained by reading $2\log n$ bits of E. The table requires an additional $(n\log n)/b$ bits to store.

Lemma 1. *The tabulation scheme (TAB−LZ) achieves random access time of $O(\frac{b+\log n}{\log n})$ for extracting any X_i. The redundancy is*

$$\delta_{tab} = \frac{\log n}{b} + O\left(\frac{1}{\log b}\right)$$

From Lemma 1, we must have $b = \omega(\log n)$ if we want the redundancy to be vanishing as $n \to \infty$. It is impossible

[1]In this paper, we use the Bachmann-Landau notation for asymptotics, and all asymptotics are with respect to the source length $n \to \infty$.

[2]Our choice of LZ78 is based on the extensive literature on the compression rates and redundancy achievable using LZ78. Our approach is nevertheless independent of the choice of the base compressor and similar results could be derived for LZ77 or other algorithms. All our bounds also hold for a variant of LZ77 [30] with $\Theta(n)$ window size.

to simultaneously achieve vanishing redundancy and $O(1)$ random access time for extracting individual symbols for $w = \Theta(\log n)$.

B. DS−LZ: Dense-Sparse compression

We now describe a novel scheme inspired by, but strictly improves on [26], the best known result in the information theory literature. As before, we split the source into blocks of $b = O(\log n)$ symbols each, and compress each block using LZ78. However, we do not simply concatenate the compressed blocks. Instead, we split the compressed sequence into two sequences: a *dense stream* and a *compressed sparse stream.* We choose

$$\beta = bH_m(X^n) + O\left(\frac{b}{\log n}\right)$$

Let k_i denote the length of the LZ78 encoding of $X^b(i)$, for $1 \leqslant i \leqslant n/b$.

- *Dense stream* U^{k_d}: This is a binary sequence of length $k_d = n\beta/b$, consisting of n/b blocks $U^\beta(1), U^\beta(2), \ldots, U^\beta(n/b)$ of length β each. Here, $U^\beta(i)$ consists of the first β bits of the LZ78 compression of $X^b(i)$ if $k_i > \beta$. If $k_i < \beta$, then we pad $\beta - k_i$ zero bits to this to obtain $U^\beta(i)$.
- *Sparse stream* $V^n(i)$: This is a concatenation of n/b blocks $V^b(1), \ldots V^b(n/b)$. Each V^{b_i} is obtained by appending the last $k_i - \beta$ bits of the LZ78 encoding of $X^b(i)$ with a string of $b - \max\{k_i - \beta, 0\}$ zeros. On average, this would contain very few 1's and is therefore sparse. The *compressed sparse stream* is obtained by compressing $V^n(i)$ using the bitvector compressor of [32]. Call this W^{k_s}. Individual blocks of the compressed sparse stream can be recovered in constant time using a constant number of rank and select queries [32].

The compressed sequence is the concatenation of U^{k_d} and W^{k_s}. The overall compressed length is

$$n\beta/b + nh_2(\beta_s/n) + o(1)$$

where $\beta_s = \sum_{i=1}^{n/b} \max\{k_i - \beta, 0\}$. It is easy to see that $\mathbb{E}[\beta_s] = O(n/\log b)$. Using Jensen's inequality, we can obtain an upper bound on the overall compressed length as

$$\mathbb{E}[k] \leqslant n\beta/b + nh_2(\mathbb{E}[\beta_s]/n) + o(1)$$

In this case, the entropy term dominates the redundancy term, and

$$h_2(\mathbb{E}[\beta_s]/n) = O\left(\frac{\mathbb{E}\beta_s}{n} \log \frac{n}{\mathbb{E}\beta_s}\right) = O\left(\frac{\log\log b}{\log b}\right)$$

and hence,

Lemma 2. *The dense-sparse scheme* DS−LZ*achieves random access time of* $O(1)$ *with word sizes* $w = O(\log n)$*, and the average redundancy is*

$$\delta_{ds} = O\left(\frac{\log\log b}{\log b}\right) = O\left(\frac{\log\log\log n}{\log\log n}\right)$$

This clearly improves upon TAB−LZ.

C. BV−LZ: Succinct data structure for bitvectors

We can in fact do better than DS−LZby a simple modification of TAB−LZ. Instead of tabulating the e_i's we use the bitvector compressor of [32] that can store a binary vector of length n and having n/b ones, using $nh_2(1/b) + o(1)$ bits. This permits rank and select queries on the compressed bitvector in time $\Theta(1)$. We use this compressor on the bitvector Y^k of length n, where $Y_i = 1$ for $i \in \{e_1, e_2, \ldots, e_{n/b}\}$, and $Y_i = 0$ otherwise. Two select queries are sufficient to recover (s_i, e_i).

Lemma 3. *The bitvector compressor-based scheme (*BV−LZ*) achieves random access time of* $O(\frac{b}{w})$ *for extracting any* X_i*. The redundancy is*

$$\delta_{bv} = O\left(\frac{\log b}{b}\right) + O\left(\frac{1}{\log b}\right) = O\left(\frac{1}{\log\log n}\right)$$

1) Comparison with optimal schemes: Let us assume that $w = \Theta(\log n)$. If we require random access time of $\Theta(1)$, then we must choose $b = \Theta(\log n)$. The redundancy of BV−LZwould scale as $O\left(\frac{\log\log\log n}{\log\log n}\right)$ whereas that of TAB−LZwould be $O(1)$. In comparison, the best known schemes [10] achieve random access time of $\Theta(1)$ and redundancy of $O\left(\frac{\log\log n}{\log n}\right)$. However, BV−LZhas a very simple implementation, and can be adapted in a straighforward manner for use with any compressor. The redundancy is higher than [10], but may still be acceptable if b is not too small.

D. Improving the redundancy using dictionary compression

Based on the discussion above, it appears to be difficult to achieve a smaller redundancy if we require $O(1)$ random access time. However, we believe that dictionary compression can allow us to reduce the redundancy.

Recall that LZ78 operates by constructing a dictionary of previously seen phrases, and the next phrase is the shortest unseen phrase [1]. A careful analysis of the derivation of the expected length suggests that the dominant redundancy term depends on the number of phrases in the LZ78 parsing of the sequence. By default, the LZ78 dictionary is initialized to the empty set, and this is expanded based on the source sequence. The algorithm can potentially be improved by initializing a pre-constructed dictionary of frequently occurring phrases. The pre-constructed dictionary constitutes an additional overhead in terms of space (since it must be included as part of the compressed sequence), but this can potentially be chosen to be even as large as $O(n/\log n) \gg O(b)$ without any significant loss in redundancy. Our experimental results suggest that the gains of using a pre-constructed dictionary can be substantial, and we leave the theoretical analysis of this approach for future work.

III. Experimental results

The three compression schemes were tested on the Wikipedia corpus (enwik8 [33], enwik9 [34]), IMDB principal data [35] and a protein FASTA file [36]. The ZSTD library [37] was used for base compression (which has an optional dictionary compression feature) and SDSL library [38] was

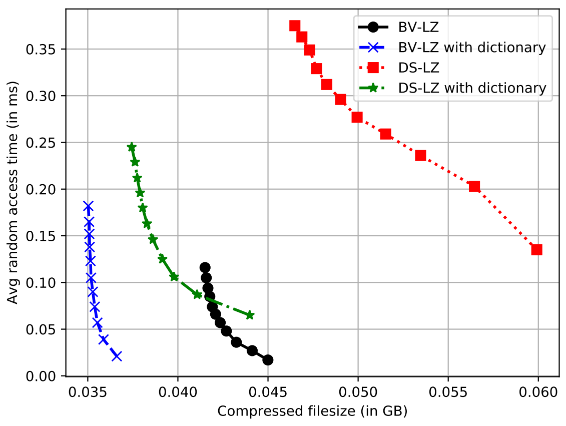

Fig. 1: Compressed size v/s random access time for enwik8 file

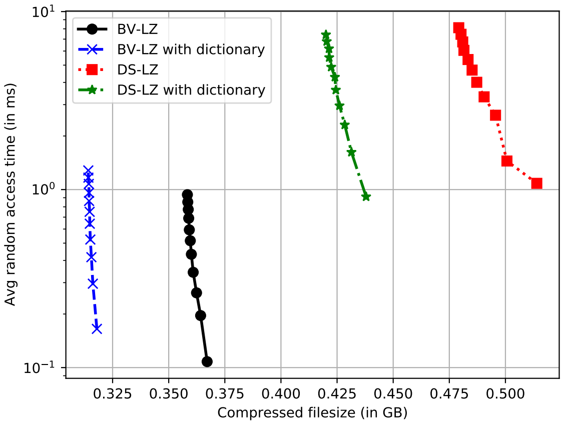

Fig. 2: Compressed size v/s random access time for enwik9 file

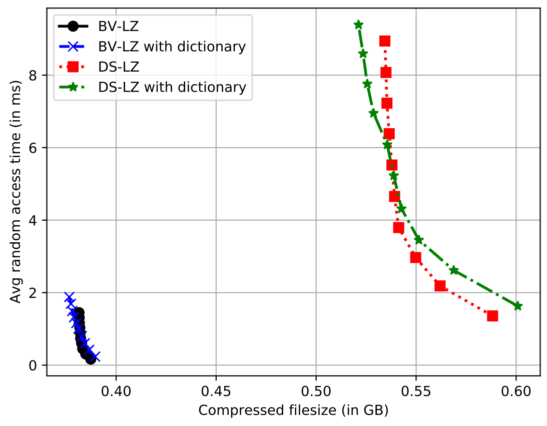

Fig. 3: Compressed size v/s random access time for imdb principal data file

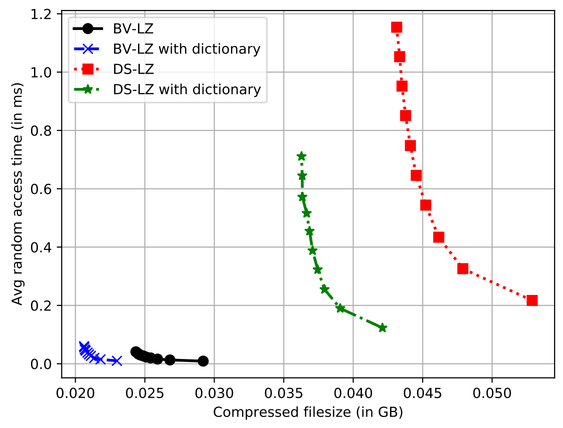

Fig. 4: Compressed size v/s random access time for a protein FASTA file

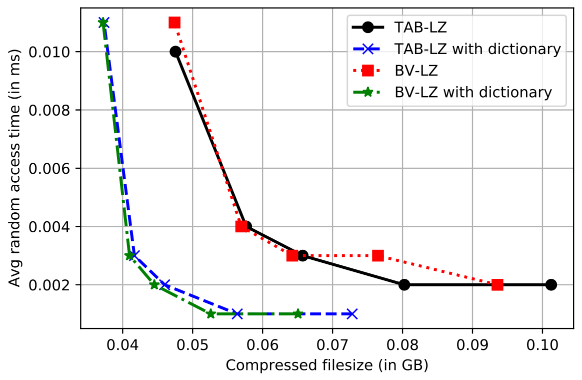

Fig. 5: Compressed size v/s random access time for the enwik8 file for TAB−LZand BV−LZschemes

used for bit-vector compression. The experiments were run on Linux machine having Ubuntu 20.04.3 LTS and AMD EPYC 7452 32-Core Processor with 128GiB RAM. All the methods were implemented using a single thread in C++14 and compiled using g++ 9.3.0. Block sizes b used were in the range $[fileSize/10000, fileSize/1000]$, where $fileSize$ is the size of the original file. For DS−LZ, we chose β to be $1.01/n$ times the compressed filesize of X^n.

From the figures, the BV−LZscheme performs better than DS−LZscheme both in terms of compressed size and random access time, which is in agreement with the theoretical results. For the BV−LZdictionary-based scheme, random access time for a given compressed file size is higher due to additional overhead of dictionary search, when compared to BV−LZwithout dictionary. For the DS−LZdictionary-based scheme, random access time for a given compressed file size is lower because of the smaller compressed size of each chunk and thereby less random access time, when compared to DS−LZwithout dictionary. Here the compressed size of a chunk is significant because it affects the size of the subsequence that requires access in the compressed bit-vector.

Fig. 3 (imdb) is interesting because for particular chunk sizes, the dictionary-based schemes have a higher random access time for a given compressed file size. And for other chunk sizes, the dictionary-based scheme perform better. The performance can be attributed to the magnitude of dictionary search time and decompression time of each chunk.

In Fig. 5, the `BV-LZ`scheme gives a smaller compressed size when compared to `TAB-LZ`scheme for small chunk sizes. This is because the header size is significant to affect the compressed file size in the two schemes, when the number of chunks is large (or) the size of chunk is small.

The following table contains the decompression time for each file using the gzip tool.

File	Original file size (in GB)	Compressed file size (in GB)	Decompression time (in ms)
enwik8	0.093	0.034	0.683
enwik9	0.931	0.301	6.278
imdb	1.998	0.36	10.46
Protein	0.079	0.024	0.564

The source codes and instructions to run are in this https://drive.google.com/drive/folders/11Z4JbVI5r2U-uWsplKKdUHhoGxI4KXGx?usp=sharing

References

[1] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, 2012.

[2] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[3] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[4] M. P. Ball, J. V. Thakuria, A. W. Zaranek, T. Clegg, A. M. Rosenbaum, X. Wu, M. Angrist, J. Bhak, J. Bobe, M. J. Callow, *et al.*, "A public resource facilitating clinical use of genomes," *Proceedings of the National Academy of Sciences*, vol. 109, no. 30, pp. 11920–11927, 2012.

[5] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis," *Nature reviews genetics*, vol. 11, no. 9, p. 647, 2010.

[6] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big data: astronomical or genomical?," *PLoS biology*, vol. 13, no. 7, p. e1002195, 2015.

[7] D. Pavlichin, T. Weissman, and G. Mably, "The quest to save genomics: Unless researchers solve the looming data compression problem, biomedical science could stagnate," *IEEE Spectrum*, vol. 55, no. 9, pp. 27–31, 2018.

[8] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of "big data" on cloud computing: Review and open research issues," *Information systems*, vol. 47, pp. 98–115, 2015.

[9] P. Nicholson, "Space-efficient data structures in the word-ram and bitprobe models," 2013.

[10] K. Sadakane and R. Grossi, "Squeezing succinct data structures into entropy bounds," in *Proceedings of the seventeenth ann. ACM-SIAM Symp. Disc. Alg. (SODA)*, pp. 1230–1239, Society for Industrial and Applied Mathematics, 2006.

[11] G. Navarro and Y. Nekrich, "Optimal dynamic sequence representations," *SIAM Journal on Computing*, vol. 43, no. 5, pp. 1781–1806, 2014.

[12] "Random access compression in BlobFS." https://fuchsia.dev/fuchsia-src/concepts/filesystems/random-access-compression. Accessed: 2022-03-31.

[13] "IBM real-time compression with IBM XIV storage system model 314," *Technical Report*, 2016.

[14] M. Patrascu, "Succincter," in *2008 49th Annual IEEE FOCS*, pp. 305–313, IEEE, 2008.

[15] Y. Dodis, M. Patrascu, and M. Thorup, "Changing base without losing space," in *Proceedings of the forty-second ACM STOC*, pp. 593–602, ACM, 2010.

[16] J. I. Munro and Y. Nekrich, "Compressed data structures for dynamic sequences," in *Algorithms-ESA 2015*, pp. 891–902, Springer, 2015.

[17] R. Raman and S. S. Rao, "Succinct dynamic dictionaries and trees," in *ICALP*, pp. 357–368, Springer, 2003.

[18] V. Chandar, D. Shah, and G. W. Wornell, "A locally encodable and decodable compressed data structure," in *Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing*, pp. 613–619, IEEE, 2009.

[19] V. B. Chandar, *Sparse graph codes for compression, sensing and secrecy*. PhD thesis, MIT, 2010.

[20] A. Makhdoumi, S.-L. Huang, M. Medard, and Y. Polyanskiy, "On locally decodable source coding," *arXiv preprint arXiv:1308.5239*, 2013.

[21] A. Makhdoumi, S.-L. Huang, M. Médard, and Y. Polyanskiy, "On locally decodable source coding," in *Proceedings of the 2015 IEEE International Conference on Communications (ICC)*, pp. 4394–4399, IEEE, 2015.

[22] H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh, "Are bitvectors optimal?," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1723–1744, 2002.

[23] A. Mazumdar, V. Chandar, and G. W. Wornell, "Local recovery in data compression for general sources," in *Proceedings of the 2015 IEEE International Symposium on Information Theory (ISIT)*, pp. 2984–2988, IEEE, 2015.

[24] K. Tatwawadi, S. S. Bidokhti, and T. Weissman, "On Universal Compression with Constant Random Access," in *2018 IEEE International Symposium on Information Theory (ISIT)*, (Vail, CO, USA), pp. 891–895, IEEE, June 2018.

[25] A. Mazumdar, V. Chandar, and G. W. Wornell, "Local recovery in data compression for general sources," in *2015 IEEE International Symposium on Information Theory (ISIT)*, (Hong Kong, Hong Kong), pp. 2984–2988, IEEE, June 2015.

[26] K. Tatwawadi, S. Bidokhti, and T. Weissman, "On universal compression with constant random access," in *Proceedings of the 2018 IEEE International Symposium on Information Theory*, pp. 891–895, 2018.

[27] A. Pananjady and T. A. Courtade, "The effect of local decodability constraints on variable-length compression," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 2593–2608, 2018.

[28] S. Vatedka and A. Tchamkerten, "Local Decode and Update for Big Data Compression," *IEEE Trans. Inf. Theory*, vol. 66, pp. 5790–5805, Sept. 2020.

[29] S. Vatedka, V. Chandar, and A. Tchamkerten, "O(log log n) Worst-Case Local Decoding and Update Efficiency for Data Compression," p. 7.

[30] A. Jain and R. K. Bansal, "On point-wise redundancy rate of bender-wolf's variant of swlz algorithm," in *2016 IEEE Information Theory Workshop (ITW)*, pp. 116–120, IEEE, 2016.

[31] S. A. Savari, "Redundancy of the lempel-ziv incremental parsing rule," *IEEE Trans. Inf. Theory*, vol. 43, no. 1, pp. 9–21, 1997.

[32] D. Okanohara and K. Sadakane, "Practical Entropy-Compressed Rank/Select Dictionary," in *2007 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pp. 60–70, Society for Industrial and Applied Mathematics, Jan. 2007.

[33] "Enwik8." https://deepai.org/dataset/enwik8. Accessed: 2022-03-31.

[34] "Enwik9." https://archive.org/details/enwik9. Accessed: 2022-03-31.

[35] "IMDB principal data." https://datasets.imdbws.com/title.principals.tsv.gz. Accessed: 2022-03-31.

[36] "Protein FASTA." https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/001/405/GCF_000001405.39_GRCh38.p13/GCF_000001405.39_GRCh38.p13_protein.faa.gz. Accessed: 2022-03-31.

[37] "Zstd compression." https://facebook.github.io/zstd/. Accessed: 2022-03-31.

[38] S. Gog, T. Beller, A. Moffat, and M. Petri, "From theory to practice: Plug and play with succinct data structures," in *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pp. 326–337, 2014.