Lossless Compression of Time Series Data with Generalized Deduplication

Rasmus Vestergaard, Qi Zhang, and Daniel E. Lucani
DIGIT and Department of Engineering
Aarhus University, Denmark
{rv, qz, daniel.lucani}@eng.au.dk

***Abstract*—To provide compressed storage for large amounts of time series data, we present a new strategy for data deduplication. Rather than attempting to deduplicate entire data chunks, we employ a generalized approach, where each chunk is split into a part worth deduplicating and a part that must be stored directly. This simple principle enables a greater compression of the often similar, non-identical, chunks of time series data than is the case for classic deduplication, while keeping benefits such as scalability, robustness, and on-the-fly storage, retrieval, and search for chunks. We analyze the method's theoretical performance, and argue that our method can asymptotically approach the entropy limit for some data configurations. To validate the method's practical merits, we finally show that it is competitive when compared to popular universal compression algorithms on the MIT-BIH ECG Compression Test Database.**

***Index Terms*—Data compression, Deduplication, Storage, Time Series Data**

I. Introduction

Many common universal compression strategies, such as the ones based on the LZ algorithms [1][2], are poorly suited for large-scale storage, since they only exploit structure in a relatively small window (KB to MB), and not globally across the large amounts of data in a storage system. Data deduplication has, for this reason, become an important technique for compressed storage of large amounts of data [3], since it achieves a good compression in many practical scenarios, and scales well to large data volumes. The basic idea is to discard any duplicate file chunks (or files, depending on deduplication granularity), and maintain a record of how to reconstruct files from these unique chunks. One shortcoming of this classic form of deduplication is inherent: Although two chunks are highly similar, no compression can occur, since chunks need to be identical to deduplicate. Some proposals on how to resolve this issue exist, e.g., Xia et al. [4] presented a strategy for backup systems. Their approach involves a multi-step approach, where first duplicate chunks in a file are detected and deduplicated, then two attempts to find similar chunks are made, which are finally delta compressed.

Generalized deduplication is a novel alternative approach [5]. By separating each chunk in two parts, one to be deduplicated, and one to be stored directly, it allows deduplication of similar chunks. For example, if two bytes per chunk are stored directly, then 65536 similar chunks can be deduplicated, which can greatly increase the compression achieved. The process is the same for all chunks, and only one pass over each chunk is needed. If the data tends to have similar but non-identical chunks, then the achieved gain can easily be higher than in classic deduplication, owing to the fact that significantly less data from a particular source is required to achieve a compression [6].

One type of data that is increasing in amount is time series data, for example, in wireless sensor networks as the result of monitoring some physical process such as temperature or humidity. This type of data deduplicates poorly with classic deduplication, due to small signal fluctuations causing chunks to be non-identical. Time series data is often streamed from sensors, so small data packets need to be stored at regular intervals. In this paper, we instantiate generalized deduplication to present a method suited for large amounts of time series data. Adjacent chunks are handled independently, using only the same dictionary as reference. This enables on-the-fly storage, retrieval, or search for any chunk independently of the rest. It provides robustness, since damage to some chunks will not impact any remaining chunks. Storing the data in this manner also allows us to achieve a compression factor comparable to popular compression algorithms, even for small data sets, and the method scales to large data sets like classic deduplication.

This paper is centered around the compression aspect of the method, and is organized as follows. The generalized deduplication method is presented in Section II. A theoretical analysis of the scheme follows in Section III. In Section IV a theoretical example illustrates the results of the preceding section. The results of applying the method to real-world data are in Section V, where it is compared to other compression algorithms. Finally, the paper is concluded in Section VI.

II. Generalized Deduplication

A specific way to instantiate generalized deduplication is presented. The goal is to compress c data chunks, $z_1, z_2, ..., z_c$. The set $\mathcal{Z}$ contains all chunks that may occur. Each chunk is considered a vector of n bits, i.e., $z_i = \left(z_1^{(i)}, z_2^{(i)}, \ldots, z_n^{(i)}\right)^T$. With no compression, these chunks need

$$S_O(c) \triangleq cn \quad \text{bits} \tag{1}$$

to be stored[1]. Our approach involves splitting the data chunks in two parts, a base $\boldsymbol{x}_i = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_k^{(i)}\right)^T$ and a deviation

[1]The overhead of storing files is ignored, since it occurs whether a file is compressed or not. We also assume that both the encoder and decoder knows the length of chunks, n.

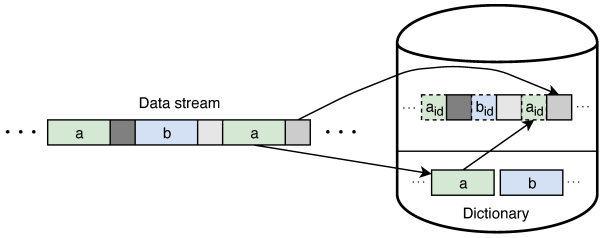

Fig. 1. Visualization of generalized deduplication. The compressed representation has clear boundaries, as in the original stream, so it is easy to access individual chunks, manipulate the compressed representation, or search.

$\boldsymbol{y}_i = \left(y_1^{(i)}, y_2^{(i)}, \ldots, y_m^{(i)}\right)^T$, such that $\boldsymbol{z}_i = \boldsymbol{x}_i || \boldsymbol{y}_i$, where $||$ denotes concatenation. All possible bases and deviations are contained in the sets $\mathcal{X}$ and $\mathcal{Y}$, respectively. An intermediate rearrangement step is added, where bits may be shifted around, so that the split aids the compression as much as possible. The compression procedure is initialized with an empty deduplication dictionary, $\mathcal{D}_0 = ()$. This dictionary will contain all bases that have been identified at a given point, in the order that they have been encountered. The chunks are compressed one at a time. In order to compress chunk $\boldsymbol{z}_i$, it is first split into its base $\boldsymbol{x}_i$ and deviation $\boldsymbol{y}_i$. The dictionary is then updated,

$$\mathcal{D}_i = \begin{cases} \mathcal{D}_{i-1} & \text{if } \boldsymbol{x}_i \in \mathcal{D}_{i-1} \\ \mathcal{D}_{i-1} || \boldsymbol{x}_i & \text{otherwise,} \end{cases} \tag{2}$$

i.e., the base is appended to the dictionary if it is not already in it. The base is thus deduplicated against the dictionary. The compressed chunk will consist of two elements, the base's location in the dictionary and the deviation. A visualization of the process is shown as Fig. 1. The compression is not more complex than classic deduplication, and the dictionary is always ready for compression of newly arriving chunks. Since chunks can be independently compressed, this allows immediate and lossless retrieval of any data down to one chunk granularity, and search for specific patterns without decompression. The base's location in the dictionary can be stored using $\lceil \log |\mathcal{D}_i| \rceil$ bits[2], and the deviation is stored in full, requiring m bits. The compression of the c chunks requires

$$S_C(c) \triangleq |\mathcal{D}_c| k + c(\lceil \log |\mathcal{D}_c| \rceil + m) \quad \text{bits} \tag{3}$$

when both the dictionary and the individual chunks are included. This allows definition of the compression factor,

$$G_c \triangleq \frac{S_O(c)}{S_C(c)} = \frac{cn}{|\mathcal{D}_c| k + c(\lceil \log |\mathcal{D}_c| \rceil + m)}, \tag{4}$$

where $G_c > 1$ implies that a compression occurs.

Remark. The special case of classic deduplication arises by choosing $k = n$, as thus $m = 0$, so deduplication of entire chunks is attempted.

[2]All logarithms in this paper are to base 2.

A. Rearranging chunks

To ensure that the split into bases and deviations is effective, the chunks must have their bits rearranged. Time series will often have some bits that have low correlation to the rest of the chunk. An example is that the least significant bits typically are hard to predict given the most significant bits. To formalize this, a metric to rank the chunk's bits based on their relationship with the rest of chunk is desired. One such metric is the mutual information between two random variables X and Y, defined as

$$I(X,Y) = H(X) + H(Y) - H(X,Y), \tag{5}$$

where H denotes the entropy [7]. If it is known that exactly m bits should be included in the deviation, then the problem

$$\begin{aligned} \underset{J}{\text{minimize}} \quad & I(Z_J, Z_{J^C}) \\ \text{subject to} \quad & |J| = m \end{aligned} \tag{6}$$

can be formulated, where Z is a random variable of the chunks. The solution J is the set of indices for the symbols of the chunk that should be part of the deviation. $J^C = \{1, 2, \ldots, n\} \backslash J$ is the set of indices of symbols in the base. $\binom{n}{m}$ different sets satisfy the constraint, so the problem can be difficult if the chunks are large. A greedy approximate solution can be found by letting $J^{(0)} = \emptyset$ and performing m iterations of

$$J^{(i+1)} = J^{(i)} \cup \left\{ \underset{j \in (J^{(i)})^C}{\operatorname{argmin}} \; I(Z_{J^{(i)} \cup \{j\}}, Z_{(J^{(i)} \cup \{j\})^C}) \right\}, \tag{7}$$

and letting $J = J^{(m)}$. This formulation also allows greedy discovery of m, by choosing a stopping condition ε, and terminating when $I(Z_{J^{(i)}}, Z_{(J^{(i)})^C}) > \varepsilon$, letting $J = J^{(i-1)}$.

Since the mutual information between independent random variables is 0, this ensures that indices of independent bits will be prioritized in J. If no bits are truly independent, then bits that are hard to predict from the remaining bits will be added. In practice, the mutual information needs to be estimated based on the available data. There are several approaches for this [8]. Once the set J has been determined, all chunks have the m bits indicated by this set moved to the end of chunk. This operation can easily be reversed given knowledge of J.

The estimation process needs to be done only once, using a training data set. When the system is initialized it can store the incoming data for a while, until enough is available to satisfactorily evaluate what the set J should be. This amount of data required will depend on the estimation algorithm chosen. Generalized deduplication can immediately be applied to any data after J has been estimated.

Remark. The set J needs to be stored. As it consists of m integers no greater than n, this can be done with an Elias gamma code [9], requiring less than $m(2\lfloor \log n \rfloor + 1)$ bits. This will have only a minor impact on the overall system, so it will be ignored in the subsequent analysis.

B. Impact of independent bits

A clarification of why it is valuable to identify independent bits is in order. Assume that $\mathcal{X}$ contains all the bases of length

k. The process of sampling a chunk could then be to pick a random base $\boldsymbol{x}_i$, and concatenate a Bernoulli distributed bit $b \in \{0,1\}$ with $\mathbb{P}[b = 1] = \alpha$ and $\alpha \notin \{0,1\}$. The set of chunks with nonzero probability now satisfies $|\mathcal{Z}| = 2|\mathcal{X}|$. If m independent bits are added, then this is repeated m times, so

$$|\mathcal{Z}| = 2^m|\mathcal{X}|, \tag{8}$$

which will decrease the probability that any chunk is already in the deduplication dictionary massively. Thus, to achieve a similar compression, a larger number of chunks will be needed. By moving independent bits to the deviation, matches will happen sooner, allowing the compression potential to be realized with significantly less data.

C. Concatenation

Time series data often exhibit correlation between adjacent values. In order to exploit this, we allow our scheme to form chunks by concatenating multiple samples. If a chunk is a concatenation of p samples of length n, then $z_i^{(p)} = z_i||z_{i+1}||\cdots||z_{i+p}$. This implies that s original samples of length n will generate s/p concatenated chunks of length pn in practice. The number of chunks available for matching is thus reduced, but this can be offset by the potential of exploiting correlation between adjacent values. It will later be made clear that concatenation of multiple samples to form each chunk indeed can provide significant advantages. We let $G_c^{(p)}$ denote the compression factor when p samples forms each chunk.

III. Theoretical Analysis

For the analysis in this section, it is assumed that the chunks have been properly ordered, such that the last m bits are independent, and that the value m is known.

Theorem 1. *Generalized deduplication will never cause an expansion of more than* $\min\{\lceil\log c\rceil, \lceil\log|\mathcal{X}|\rceil\}$ *bits per chunk.*

Proof. Clearly $c \geq |D_c|$, where equality is the worst-case where no chunks have a common base, and thus no deduplication occurs. If this happens, the total cost becomes

$$\begin{aligned} S_C(c) &= |\mathcal{D}_c|k + c(\lceil\log|\mathcal{D}_c|\rceil + m) \\ &\leq ck + c(\lceil\log c\rceil + m) \\ &= c(m+k) + c(\lceil\log c\rceil) \\ &= cn + c(\lceil\log c\rceil) \\ &= S_O(c) + c(\lceil\log c\rceil), \end{aligned} \tag{9}$$

which gives the first upper bound. A similar argument can be made using that $\mathcal{D}_c \subseteq \mathcal{X}$, so $|\mathcal{D}_c| \leq |\mathcal{X}|$, which gives the second bound. □

Theorem 2. *If infinitely many chunks are available, then the average coded length per chunk,* $L \triangleq \lim_{c\to\infty} \frac{S_C(c)}{c}$*, becomes*

$$L = \lceil\log|\mathcal{X}|\rceil + m.$$

Proof. The average length is found as the number of chunks is allowed to go to infinity:

$$\begin{aligned} \lim_{c\to\infty}\frac{1}{c}S_C(c) &= \lim_{c\to\infty}\frac{1}{c}\Big[|\mathcal{D}_c|k + c(\lceil\log|\mathcal{D}_c|\rceil + m)\Big] \\ &= \lim_{c\to\infty}\frac{1}{c}|\mathcal{D}_c|k + \lim_{c\to\infty}\lceil\log|\mathcal{D}_c|\rceil + m \\ &= \lceil\log|\mathcal{X}|\rceil + m, \end{aligned} \tag{10}$$

where the last equality follows from the fact that all bases with non-zero probability will eventually be encountered, so $\lim_{c\to\infty}\mathcal{D}_c = \mathcal{X}$. □

Corollary 1. *The compression factor can be expected to be*

$$G_\infty = \frac{n}{\lceil\log|\mathcal{X}|\rceil + m}$$

if infinitely many chunks are available.

Proof. The result follows immediately from dividing the original size of a chunk, n bits, with the average encoded length in the limit, L bits. □

This corollary allows quantification of the compression gain in the limit. While specific sets of chunks with many matching bases may achieve a greater compression, this is the best gain that can be expected on average. A greater compression can be achieved if subsequent samples are not independent, but are correlated. We assume that, given sample i, there are q distinct values with non-zero probability for sample $i+1$, $i \in \{1,2,\ldots,p-1\}$.[3]

Theorem 3. *With concatenation of p samples to form each chunk, the compression factor of infinitely many chunks is*

$$G_\infty^{(\infty)} = \frac{n}{\log q + m},$$

as $p \to \infty$. This is at least as much as without concatentation.

Proof. A chunk, formed by concatenating p samples of n bits each, will have a length of pn bits. Likewise, the deviation representation is scaled by p, and will require pm bits. The set of concatenated bases with non-zero probability is denoted $\mathcal{X}_p$. Corollary 1 can be used to find the compression factor with infinite amounts of data, i.e.,

$$G_\infty^{(p)} = \frac{pn}{\lceil\log|\mathcal{X}_p|\rceil + pm}. \tag{11}$$

When adjacent samples are limited to one of q possibilities, then the set of concatenated chunks grows as

$$|\mathcal{X}_p| = |\mathcal{X}|q^{p-1}, \tag{12}$$

where $\mathcal{X}_p$ denotes the set of chunks with p concatenated samples. This means that

$$\begin{aligned} \lceil\log|\mathcal{X}_p|\rceil &\leq \log|\mathcal{X}_p| + 1 \\ &= \log\left(|\mathcal{X}|q^{p-1}\right) + 1 \\ &= \log|\mathcal{X}| + (p-1)\log q + 1. \end{aligned} \tag{13}$$

[3] We assume that q is the same for all sample values. Each sample value could have its own number of following values with non-zero probability. Then $q = \max q_i$ is an upper bound, giving a lower bound on the gain.

This is a tight upper bound, and is less than one bit greater. Equivalently, for some value $\delta \in (0, 1]$,

$$\lceil \log |\mathcal{X}_p| \rceil = \log |\mathcal{X}| + (p-1)\log q + 1 - \delta. \tag{14}$$

This can be combined with (11), obtaining an expected gain of

$$\begin{aligned} G_\infty^{(p)} &= \frac{pn}{\log |\mathcal{X}| + (p-1)\log q + 1 - \delta + pm} \\ &= \frac{n}{\frac{1}{p}(\log |\mathcal{X}| - \log q + 1 - \delta) + \log q + m}, \end{aligned} \tag{15}$$

when p samples are concatenated for each chunk. Finally, the limit in the number of samples concatenated is

$$\lim_{p\to\infty} G_\infty^{(p)} = \frac{n}{\log q + m}. \tag{16}$$

The final observation is that necessarily $q \leq |\mathcal{X}|$, so concatenation can never reduce the compression potential. □

Corollary 2. *Concatenation may increase the maximum gain slightly, even if samples are independent.*

Proof. If samples are independent, then $q = |\mathcal{X}|$. Since $\lceil \log |\mathcal{X}| \rceil \geq \log |\mathcal{X}|$, then clearly $G_\infty \leq G_\infty^{(\infty)}$. □

It should be noted that, although concatenation can only increase the potential, it may also increase the amount of chunks required to reach the potential so much that a smaller gain is achieved in practice, especially if concatenated samples are independent, where the potential advantage is small.

Theorem 4. *If bases are uniformly distributed and the independent bits are entropy coded, then generalized deduplication can converge to the entropy of the chunk source.*

Proof. Let $\mathcal{Z}_p$ be a concatenated chunk source obtained by concatenating p samples for each chunk. We first determine the entropy of this source. Let $\mathcal{X}_p$ be the concatenated base source, and $\mathcal{Y}_p$ the concatenated deviation source. The entropy of $\mathcal{X}_p$ is $\log |\mathcal{X}_p|$, since bases are assumed uniformly distributed over $\mathcal{X}_p$. Using (12), the entropy for the base source is

$$\begin{aligned} H(\mathcal{X}_p) &= \log |\mathcal{X}_p| \\ &= p \log q + (\log |\mathcal{X}| - \log q + 1) \\ &= p \log q + O(1), \end{aligned} \tag{17}$$

where $O(1)$ is a constant overhead, independent of p. As p grows, the overhead per sample goes to 0. Due to the independence of the deviations, $H(\mathcal{Y}_p) = pH(\mathcal{Y})$. The entropy for concatenated chunks is then

$$\begin{aligned} H(\mathcal{Z}_p) &= H(\mathcal{X}_p, \mathcal{Y}_p) \\ &= H(\mathcal{X}_p) + H(\mathcal{Y}_p|\mathcal{X}_p) \\ &= H(\mathcal{X}_p) + H(\mathcal{Y}_p) && (18) \\ &= p \log q + pH(\mathcal{Y}) + O(1), && (19) \end{aligned}$$

where (18) follows due to the independence of $\mathcal{X}_p$ and $\mathcal{Y}_p$. Finally, the entropy per sample is

$$H(\mathcal{Z}) = \lim_{p\to\infty} \frac{1}{p} H(\mathcal{Z}_p) = \log q + H(\mathcal{Y}). \tag{20}$$

Now we need to show that generalized deduplication can converge to this cost. Indeed, the expected base cost, $\log q$, is immediate from Theorem 3. Now it remains to conclude that a code exists such that a sequence of p iid symbols from $\mathcal{Y}$ will require an expected $H(\mathcal{Y})$ bits per symbol on average. Classic source coding results confirm that such codes exist [7], allowing an expected length arbitrarily close to the entropy as the number of symbols encoded jointly grows, which is exactly what occurs when multiple deviations are concatenated. An example of a suitable scheme is arithmetic coding [10]. □

IV. A motivating example

We assume that a data source satisfies:

- $n = 12$, i.e., each sample is 12 bits in total,
- $k = 8$, i.e., the samples have bases that are 8 bits long,
- $m = n - k = 4$, i.e., deviations are 4 bits, and each bit is iid Bernoulli with $\alpha = 1/2$.
- $|\mathcal{X}| = 30$, i.e., 30 different bases exist,
- $\mathbb{P}[X = x] = \frac{1}{|\mathcal{X}|}$ for $x \in \mathcal{X}$, i.e., the bases are uniformly distributed when chosen independently,
- $q = \max_{x_j \in \mathcal{X}} |\{i : \mathbb{P}\left[X_t = x_i | X_{t-1} = x_j\right] > 0\}| = 6$, i.e., once a certain base is observed at time t, then the subsequent must take one of 6 distinct values.
- $\mathbb{P}\left[X_t = x_i | X_{t-1} = x_j\right] \in \{0, \frac{1}{q}\}$, i.e., the conditional distribution is uniform over bases with non-zero probability.

Corollary 1 tells us that with an infinite amount of data $G_\infty = 4/3$, or that this is the maximum expected compression gain if no concatenation is used. On the other hand, Theorem 3 and 4 reveals that this can be increased to the entropy limit of $G_\infty^{(\infty)} \approx 1.82$ if concatenations are used. The concatenation causes a trade-off between the potential gain and the amount of data required to reach it. It is important to select this parameter responsibly, and the results previously stated can help. Indeed, when a certain number of samples, s, is available, our goal is to maximize the expectation of (4), the compression factor, over the number of concatenated chunks,

$$\begin{aligned} \underset{p}{\text{maximize}} \quad & \mathbb{E}\left[G_c^{(p)}\right] \\ \text{subject to} \quad & p > 0,\ p \in \mathbb{Z}. \end{aligned} \tag{21}$$

By noting that s samples will result in $c = s/p$ chunks of length pn, the objective function can be formulated as

$$\begin{aligned} \mathbb{E}\left[G_c^{(p)}\right] &= \mathbb{E}\left[\frac{cnp}{|\mathcal{D}_c|kp + c(\lceil \log |\mathcal{D}_c| \rceil + mp)}\right] \\ &= \mathbb{E}\left[\frac{sn}{|\mathcal{D}_{\frac{s}{p}}|kp + \frac{s}{p}\lceil \log |\mathcal{D}_{\frac{s}{p}}| \rceil + sm}\right] \\ &\approx \frac{sn}{\mathbb{E}\left[|\mathcal{D}_{\frac{s}{p}}|\right]kp + \frac{s}{p}\lceil \log \mathbb{E}\left[|\mathcal{D}_{\frac{s}{p}}|\right] \rceil + sm}, \end{aligned} \tag{22}$$

where the approximation arises from moving the expectation into the ceiling and log function. This approximation is good enough for our purposes. Finally, the fact that both the first base of a chunk and subsequent ones are uniformly distributed

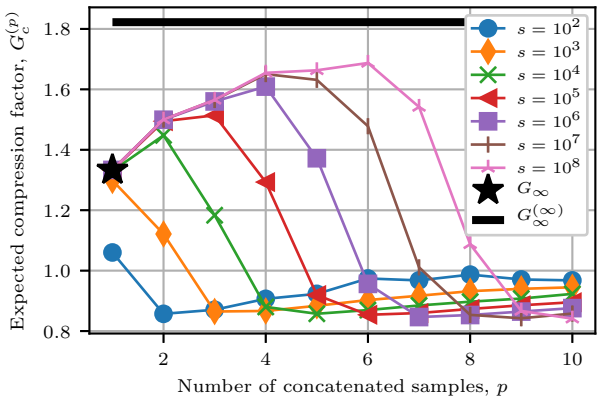

Fig. 2. Results for example configuration. The colored lines are for different numbers of unconcatenated samples. $c = s/p$.

implies that the concatenated bases are uniformly distributed over $\mathcal{X}_p$, so

$$\mathbb{E}\big[|\mathcal{D}_c|\big] = |\mathcal{X}_p|\left(1 - \left(\frac{|\mathcal{X}_p| - 1}{|\mathcal{X}_p|}\right)^c\right), \tag{23}$$

where $|\mathcal{X}_p|$ is the number of bases from chunks with p samples concatenated, as in (12). A full equation for the expected gain is now available. It is complicated to solve analytically, but easy to evaluate, so an optimal solution is easily found by evaluating (22) at realistic p values. Fig. 2 shows the result for the parameters of this section, and different numbers of available samples. The theoretical limits are also indicated. The p maximizing the compression factor for a given number of samples is the solution to the optimization problem. It is clearly seen how the bound of Corollary 1 is achieved, and it is also seen that a very large amount of samples is needed to approach the bound of Theorem 3. Although this bound is not achieved, concatenation still proves advantageous, and increases the gain.

V. A real example

To provide a real example, the MIT-BIH ECG Compression Test Database [11][12], a small data set containing a variety of features intended to be difficult to compress, is used. Two-lead ECG signals are available for 168 subjects, but only the first lead is considered for brevity. 20.48 seconds are available for each subject, sampled at 250 Hz at a 12 bit resolution. This results in 7.68 KB per subject, or a total of 1.31 MB when storage overhead is included.

A. Implementation of our scheme

Encoding: An initial empty file is created for storage of the bases. The data files are then processed one-by-one. Each data file is organized into chunks of the specified length by concatenating subsequent samples. The intended reordering is found by using the first supplied file as a training data set. The Pearson correlation, a well-known simple measure of correlation capturing only linear relationships, is used. In many cases such linear coefficients carries the same information as the mutual information [8], and they are much simpler to estimate. The bits of the chunk with least absolute correlation are moved to the deviation, and this reordering is stored in a separate file. Chunks of all the files are then rearranged to fit this desired reordering. All of the bases in the file are then added to the dictionary file, where identical bases are deduplicated. The compressed file is created, and starts with an Elias gamma coding [9] of $d = \lceil \log |\mathcal{D}| \rceil$, the number of bits required to represent any base in the dictionary at the time of encoding. The location of each chunk's base in the dictionary is identified, and its d bit representation is stored alongside the deviation. The next file is then processed.

Decoding: Decoding a file is straightforward. First, the Elias gamma coded integer d is decoded, determining how many bits are used for the base references in this specific file. Each chunk can now independently be decoded by finding the indicated base in the dictionary, concatenating the deviation, and finally reversing the rearrangement. We note that decompression of single chunks is always possible, since there is no dependency between the chunks of a file. Similarly, it is easy to search for any specific chunk, as the base representation can be identified from the dictionary, and searched for in the compressed files. If the base is not in the dictionary, then the chunk will not be in any file.

B. Comparison schemes

The compression achieved by our scheme is compared to two popular general purpose compression methods:

- GZIP [13], which uses the DEFLATE algorithm [14] based on LZ77 [1] and Huffman coding [15].
- 7z [14], which uses LZMA, an extension of LZ77.

We perform the compression in two ways: Either each file is compressed individually, or the entire data set is compressed jointly. Independently compressing files allows compression and decompression of individual files, whereas joint compression requires all of the data to be available before compression, and all to be decompressed jointly. Our method allows a much finer granularity, and allows storage and retrieval of individual chunks independently of other chunks in the system.

C. Numerical results

Rather than estimating the number of independent bits, $m \in \{1, 2, \ldots, n\}$, where n is the length of the concatenated chunks, the compression is run for all values to see what the effect of this parameter is. The cost of storing the data is shown in Fig. 3 for a variety of generalized deduplication configurations. Five different chunk sizes are shown, resulting from concatenation of $p \in \{1, 2, 4, 8, 12\}$ samples. The x-axis corresponds to the fraction of the chunk used as deviation, i.e., m/n. Depending on the chunk length, there is a different deviation length that results in minimum compressed size. The length of the deviation has a clear impact on the number of matching bases, so if fewer chunks are available (which is the case when more samples are concatenated), it might make sense to use a longer deviation. It should also be mentioned that, if the chunk length is long, and the deviation is very

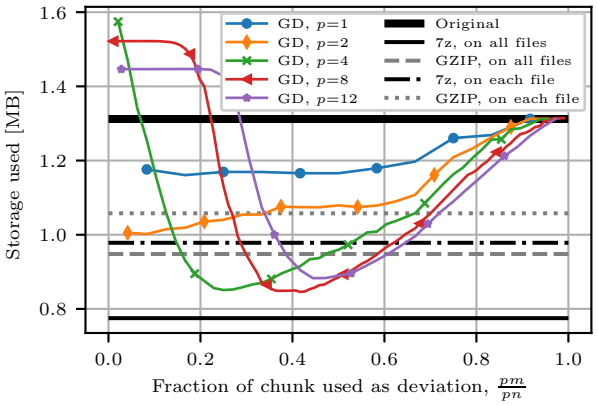

Fig. 3. Comparison of compressed size after generalized deduplication with different deviation sizes, m, for MIT-BIH ECG Compression Test Database.

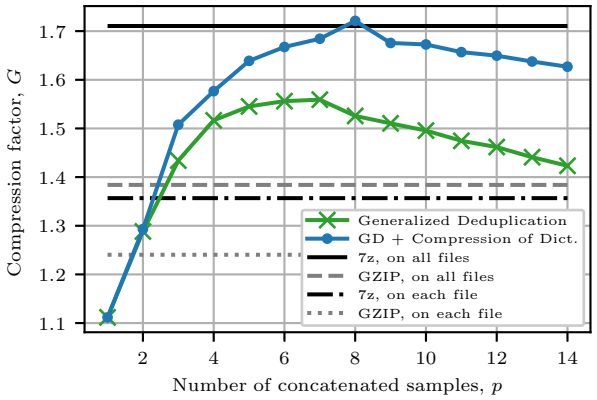

Fig. 4. Maximum compression factor as a function of number of concatenated samples for MIT-BIH ECG Compression Test Database.

small, then no or very few matches are observed. The worst-case scenario is seen for $p = 8$ and $p = 12$ with small m, where Theorem 1 applies. Our method not only has better on-the-fly capabilities, the best configuration also achieves an 11% smaller size than GZIP with all files bundled does, or a 14% smaller size than 7z with files individually compressed.

Fig. 4 shows a different angle, the maximum achieved compression factor for different concatenations. Our method can achieve a better compression than 7z on individual files, and GZIP on the entire data set. The dictionary may contain redundancy for some configurations. Fig. 4 also shows the compression factor achieved after compression of the dictionary, which reaches the same level as the best-case scenario for 7z. This is a more fair comparison to 7z using a single archive, since the on-the-fly capabilities are similar, as both methods need to extract a large amount of data to access even a single chunk.

VI. Conclusions and Future Work

A new principle for on-the-fly compression of large amounts of time series data has been presented. Our method builds on the ideas behind data deduplication, and maintains advantages such as scalability, retrieval of individual file chunks, and robustness to errors, which makes it promising for storage systems. Through our theoretical analysis of the scheme, we have argued that our method may converge to the entropy limit, given enough data. Further, we have used a real ECG data set to show that our method can achieve a better compression than GZIP, and better than 7z under some conditions, while providing desirable qualities that these methods lack.

For our future work we will solve some of the remaining practical issues of the scheme. In particular, an efficient implementation will be made, allowing a more thorough evaluation with other data sets, and a comparison to other methods on parameters such as memory usage and throughput. In this paper, we also operated under the assumption that bases were uniformly distributed, and encoded the base identifiers according to this. However, this assumption might not be good in practice, and we expect that a greater compression will be achieved if the distribution is leveraged. Finally, we will investigate strategies for automatically selecting important parameters, such as the number of samples concatenated and the deviation length, in a data-aware manner.

Acknowledgments

This work was partially financed by the SCALE-IoT project (Grant No. DFF-7026-00042B) granted by the Danish Council for Independent Research, the AUFF Starting Grant AUFF-2017-FLS-7-1, and Aarhus University's DIGIT Centre.

References

[1] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, may 1977.

[2] ——, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, sep 1978.

[3] W. Xia, H. Jiang, D. Feng *et al.*, "A Comprehensive Study of the Past, Present, and Future of Data Deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[4] W. Xia, H. Jiang, D. Feng, and L. Tian, "Combining Deduplication and Delta Compression to Achieve Low-Overhead Data Reduction on Backup Datasets," in *Data Compression Conf.*, Mar 2014, pp. 203–212.

[5] R. Vestergaard, Q. Zhang, and D. E. Lucani, "Generalized Deduplication: Bounds, Convergence, and Asymptotic Properties," in *IEEE GLOBECOM*, Waikoloa, USA, 2019.

[6] R. Vestergaard, D. E. Lucani, and Q. Zhang, "Generalized Deduplication: Lossless Compression for Large Amounts of Small IoT Data," in *European Wireless Conf.*, Aarhus, Denmark, May 2019.

[7] T. M. Cover and J. A. Thomas, *Elements of information theory*. Wiley-Interscience, 2006.

[8] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical Review E*, vol. 69, no. 6, p. 66138, jun 2004.

[9] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inf. Theory*, vol. 21, no. 2, pp. 194–203, mar 1975.

[10] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, jun 1987.

[11] G. B. Moody, R. G. Mark, and A. L. Goldberger, "Evaluation of the 'TRIM' ECG data compressor," in *Proc. Computers in Cardiology*, 1988.

[12] A. Goldberger, L. Amaral, L. Glass *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet : Components of a New Research Resource for Complex Physiologic Signals," *Circulation*, vol. 101, pp. E215–20, 2000.

[13] P. Deutsch, "GZIP File Format Specification Version 4.3," Internet Requests for Comments, RFC 1952, 1996.

[14] D. Salomon and G. Motta, *Handbook of Data Compression*. Springer, London, 2010.

[15] D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, sep 1952.