AAC: An anomaly aware time series compression algorithm towards green computing

Jingwen Meng
Nanjing University of Aeronautics and Astronautics Nanjing
Liang Liu (liangliu@nuaa.edu.cn)
Nanjing University of Aeronautics and Astronautics Nanjing
Yulei Liu
Nanjing University of Aeronautics and Astronautics Nanjing
Ning Wang
Nanjing University of Aeronautics and Astronautics Nanjing

Research Article

Keywords: Time Series, Data Compression, Anomaly Detection, Green Computing

Posted Date: November 9th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3505032/v1

Additional Declarations: No competing interests reported.

AAC: An anomaly aware time series compression algorithm towards green computing

Jingwen Meng[1], Liang Liu[1*], Yulei Liu[1], Ning Wang[1]

[1*]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics Nanjing, 29 Yudao St, Nanjing, 210016, Jiangsu Province, China.

*Corresponding author(s). E-mail(s): liangliu@nuaa.edu.cn;
Contributing authors: jwmeng@nuaa.edu.cn; liu_yulei@nuaa.edu.cn;
nuaawangning@nuaa.edu.cn;

Abstract

The storage of time series big data in Internet of Things (IoT) requires a lot of environmentally unfriendly servers that generate a large amount of energy consumption. Reducing the number of storage servers with efficient time series compression techniques is one of the available solutions to decrease energy consumption to realize green computing. Therefore, in this paper, we propose an Anomaly Aware Compression algorithm (AAC) aiming to achieve green storage by obtaining higher compression ratios. Anomalies appear frequently in time series data in scenarios such as IoT, but no compression algorithm has been designed for them. AAC realizes better compression results by dividing and compressing abnormal and normal data separately. For the two types of data, we present Window Delta encoding and Dispersion Tolerant encoding to efficiently compress them. Our experimental results demonstrate that the approach achieves about 5% - 120% improvement in compression ratio compared to the state of art.

Keywords: Time Series, Data Compression, Anomaly Detection, Green Computing

1 Introduction

Due to the promotion of artificial intelligence, Internet of Things (IoT), 5G and other industries, time series data is quickly accumulated and widely used[1]. For example, monitoring systems[2] constantly collect time series data and perform anomaly detection so that they can respond promptly to anomalies. In addition, in industries such as energy[3], financial[4] and medical[5], companies have to learn historical time series data for trend prediction to enhance industry competitiveness.

In order to adapt to the growth rate and special structure of time series data, a tailored time series databases (TSDBs) has been emerged and developed rapidly, such as InfluxDB[6] and Prometheus[7]. However, the explosion growth of time series data means that more and more storage devices are needed. Also, the manufacturing and operation of hardware facilities, TSDBs, generate large amounts of energy consumption and greenhouse gas, which is not conducive to the sustainable development of the IoT industry. For the purposes of reducing the environmental pollution, it is an essential way to find more effective time series data compression algorithms to reduce the

space required for storage. Therefor, time series compression is discussed in this paper.

Time series compression consists of timestamp compression and metric value compression. Since timestamp compression technology have been sufficiently mature, exploring efficient compression algorithms of metrics is the major concern of this paper. For metric values, the data types are broadly divided into integer and floating-point. In practical terms, floating-point data are often stored in TSDBs in integer form to save storage space and improve transmission speed. In this paper, we are concerned with the study of integer-oriented compression.

Plenty of time series compression technologies have been proposed and employed in TSDBs, such as DoD and XOR encoding in Gorilla[8]. Existing algorithms generally take a whole data block as a compression object, and they always use a single method to compress a data block. However, a data block usually contains some data points that cannot be compressed well by the current compression method. These points may lead to undesirable encoding results, and we refer to them as abnormal points. Take the measurements in Figure 1 as an example, $v_4 = 2400$ is an abnormal points for $V = \{120, 240, 180, 2400, 270, 480, 350\}$. For Delta of Delta (DoD) Encoding [8], the second order difference sequence obtained is: $D = \{120, 120, -180, 2280, -4350, 2340, -340\}$. Notice that v_4 with a large value leads d_4 to be much bigger than other deltas. Not only that, since v_4 is involved in the second order difference calculation of adjacent data points, it also affects the results of d_5 and d_6. If v_4 is reset to 300, the new sequence is:$D^{'} = \{120, 120, -180, 180, -150, 240, -340\}$. The values of d_4, d_5 and d_6 are reduced by at least one order of magnitude, and the final encoding length is diminished from 144 bits to 69 bits. Therefore, it can be said that the anomalies severely depresses compression ratios.

In the research field of time series compression, even though the issue of abnormal points is rarely mentioned so far, its impact on compression ratios cannot be ignored. Because in IoT, the monitored objects might be in abnormal conditions, such as hardware failure, which can lead to abnormal metric values. In order to mitigate the negative impact of abnormal values on compression ratios, thus decreasing the cost and energy consumption of data storage and realizing green

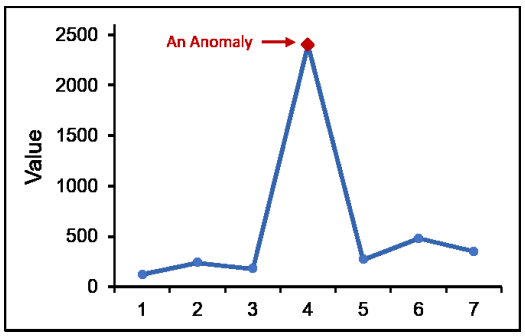


Fig. 1 An example of an anomaly in a dataset

computing, we propose an anomaly-aware compression algorithm (AAC). AAC is structured in two phases: At the bottom level, data points that differ from the majority in data patterns are identified and separated from the original sequence, and a data block is divided into two parts, normal and abnormal. Then, they are encoded separately by different encoding methods at the top level.

During the design and implementation process we are mainly confronted with the following challenges:

1. How to realize a accurate anomaly detection?
2. How to efficiently encode abnormal points?
3. How to encode normal points better?

We aim for higher compression ratios, thus well-performing encoding methods for normal and abnormal values are crucial. Moreover, the anomaly-aware compression algorithm is only meaningful if the anomalies can be found accurately. Therefore, we design detailed solutions for the above challenges respectively. Generally speaking, our key contributions are as follows:

1. We introduce an Anomaly-Aware Compression algorithm (AAC) to minimizes the influence of anomalies on compression ratio. A effective separation strategy is proposed to extract abnormal points from the original sequence .
2. A new integer time series compression algorithm named Window Delta (WD) is proposed. As the encoding method for normal values, it has better compression ratios than Delta of Delta (DoD).
3. We design a dispersion-tolerant compression method to encode anomalous data points, which can efficiently process numerically dispersed values.

4. We assess AAC using an enormous number of real datasets. The experimental results show that AAC gains up to 120% improvement in compression ratio over existing algorithms.

Additionally, in Section 2 we review progress in research on data compression; Then, the framework of AAC are described in section 3, and section 4-6 show the design and implementation details; Next, for section 7, we evaluate the method through substantial experiments; Finally, we make a conclusion and give expectations for future work.

2 Related Work

In terms of completeness of data restoration, compression can be classified as lossless and lossy[9]. From the perspective of data types, integers and floats[10] are major concerns. Present relevant effort is basically focused on the above four aspects.

2.1 Lossy Compression

In contrast to lossless compression, lossy compression has characteristics such as fast compression speed, high compression ratio and incomplete data recovery, so it is often utilized in situations where data volumes are heavy but storage resources are limited, such as IoT[11]. Data dimension reduction is a mainstream solution for lossy compression. The most classical methods include SVD (Singular Value Decomposition)[12, 13], DFT (Discrete Fourier Transform)[14–16] and DWT (Discrete Wavelet Transform)[17, 18]. Their primary idea is to store the crucial data that can represent data patterns and tendency instead of raw data, thus reducing the amount of data stored. Keogh[19] et al. proposed another data approximation representation method, PAA (Piecewise Aggregation Approximation), which segments data into equal lengths and replaces original values with average of each segment. Though PAA may lead to the omission of some important data points, the thought of approximate representation of time series inspired a great number of associated scholars afterwards. PLA (Piecewise Linear Approximation)[20] and curve fitting model[21] take advantage of linear and polynomial functions to represent subsegments. In comparison to PAA, they have more accurate data representation. Paper[22] and [23] respectively consider the segmentation techniques of time series under the above two methods. The former takes a global viewpoint by treating the relatively important maximums of time series as breakpoints, and the data between adjacent breakpoints are denoted by linear functions. And the latter decides to segments data dynamically. Data points are sequentially incorporated into a compression window, and each time the window is updated, the algorithm performs a process to determine whether the subsequence in the window can still be fitted with a polynomial. If not, a new segment is started from the current point. To ensure availability and tolerability of decompressed data, Squeeze(SZ)[24] and LFZip[25] not only pursue high compression ratios but also take into account error bounds.

2.2 Lossless Compression

Despite the fact that lossless compression hardly achieves the same efficiency as lossy compression, it still plays an undeniably important role in industry due to data recoverability. Lossless compression is sensitive to data type, so integer and floating-point compression are always discussed separately in this place.

2.2.1 Floating Point Data Compression

For floating-point time series compression, XOR[8] from Facebook is currently one of the most recognized compression algorithms. The general idea of XOR encoding is to compute XOR of the current value with the previous. FPC[26] performs a XOR by raw and predicted value of a point, instead of the values of two neighboring nodes. By the means, the possible threat of numerical differences between two adjacent data points is avoided. Not only that, Bruno[27] et al. realized that data points that are similar in decimal may not be similar in binary, and therefore proposed TSxor. Both FPC and TSxor made optimization for simple XOR in Gorilla, and get better compression ratio and rate. There are also plenty of useful algorithms for integer compression.

2.2.2 Integer Compression

The modern prevalent integer compression ideology is to shorten the stored bit length to the greatest extent through a new calculation method or data structure. Delta of Delta (DoD)[8] calculates all second order difference of sequences, and codes them by Variable Length Encoding. Simple8b[28] is a simple and classical coding method for integer. It attempts to package integers into 64-bit words. ZigZag[29] increases leading zeros of complements by some operations, thus reducing the length of valid bits that need to be saved in the end. ZigZag is negative friendly. PFD (PFor-Delta)[30] and newPFDs (variants of PFD)[31, 32] are fixed-length lossless compression algorithms, and they take a two-stage compression framework. At the bottom level, the majority of points in a time series are assigned fixed-length bits for storage. And then, points with extremely high values will be compressed at the top level. RBM (Roaring BitMap Encoding)[33] is a new structure designed on the foundation of bitmap with the thought of bucketing. RBM divides each 32-bit integer into upper 16 bits and lower 16 bits, and places the points sharing the same high 16 bits into the same big bucket. In this way, the storage space required for each value is reduced by 50%. In particular, RBM has significant benefits when the distribution of values is scattered. As a gradually matured technique, machine learning is increasingly applied in the research of data compression algorithms. In the area of multivariate integer time series compression, Sprintz[34] Leverages time series prediction to obtain the most advanced compression ratio.

It is not hard to see that the prevailing idea to improve compression ratio is still to explore a common solution to compress a whole time series. But the truth of it is that a single compression method cannot be suitable for all data points. For most existing algorithms, compression ratios are ideal when data distribution is stable. On the contrary, compression ratios decrease when values of data fluctuate a lot or even leading to breakpoints. Therefore, we pay attention to the issue, and designed an anomaly-aware compression algorithm, AAC. AAC try to separate anomalies from the original sequence and encode them independently. While PFD is carried out with the similar question in mind, it merely distinguishes between

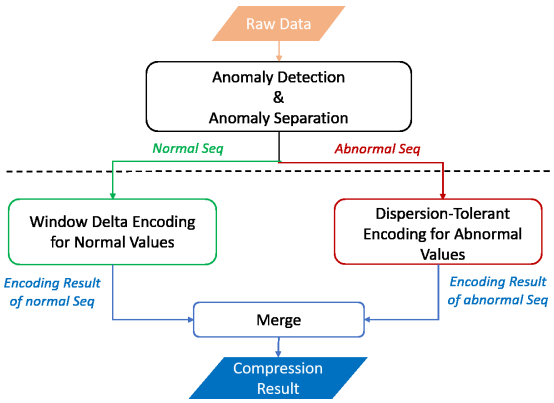


Fig. 2 Structure of Anomaly-Aware compression

normal and abnormal points by setting a simple threshold. Obviously, that is not a real anomaly detection. In contrast, AAC introduces for the first time a well-structured anomaly-aware compression framework for integer, including initial coding, anomaly detection and anomaly management.

3 Anomaly-Aware Compression

3.1 Structure

To address the interference of anomalies on compression ratios, we introduce an anomaly-aware compression algorithm (AAC). As shown in the figure 2, AAC consists of three modules, which are: anomaly detection and separation, window delta encoding for normal values, and dispersion tolerant encoding for abnormal values.

First, we find out abnormal values and depart them from the original sequence, so that the dataset is divided into two sets: normal and abnormal. Then, they are respectively encoded by different compression methods. At the end, we merge the two encoding results and obtain the final compression result.

We choose Isolation Forest as the foundation for anomaly detection, because it is characterized by quick calculating speed and small footprint and satisfies the time requirement of data compression. After anomaly detection and extraction, the remaining normal points in which the adjacent values tend to differ very little, and the overall data

pattern is always frequently fluctuating. Theoretically, the smoother the data pattern the better the compression. In order to realize higher compression ratios, we want to minimize fluctuations as many as possible. Therefore, we propose Window Delta Encoding, which is capable of processing fluctuating values more flexibly and efficiently. For abnormal points, they tend to be extremely scattered in value and therefore cannot be effectively compressed by traditional methods. For the property, we design a dispersion tolerant encoding, which is a grouping compression algorithm based on common prefixes.

3.2 Definition of Anomaly

It is worth noting that the definition of anomalies (abnormal points) in AAC differs from the traditional. The conventional statement is: "An anomaly is an observation which deviates so much from the other observations." In fact, the definition of anomalies in AAC is not directly related to whether they are numerically significantly different from most data points. As long as a data point has a far greater negative impact on the compression method than the most others, we regard it as an anomaly (abnormal point).

For example, suppose there is a set of values $V = \{120, 240, 180, 2400, 270, 480, 350\}$ in figure 1. Under Simple8b Encoding[28], bit lengths allocated to each value that packed in the same 64-bit word are equal. Owing to the binary form of v_4 has much more significance bits(12 bits) than other points(9 bits), only the first five values can be packed into the first word, and the final compressed length is 128. If v_4 is numerically similar to the others, it only takes one 64-bit word to pack V. Accordingly, for the current compression algorithm, Simple8b, it is believed that v_4 drops the compression ratios, while most other values are suitable for it. Identically, for DoD Encoding, the value 2400 is also an element that regrade the compression ratio. We can conclude that the data point $v_4 = 2400$ is an anomaly in the data sequence V for Simple8b and DoD Encoding. Nonetheless, v_4 is not always treated as anomalous. Take Variable-Byte Encoding as an example, it allocates different lengths of bytes for each value, thus the value 2400 dose not impact on the encoding results of others. In summary, in terms of compression, we define the anomaly as: After a data point is encoded, if not only does it require a much longer encoding length than the others, but it also affects the encoding results of the other data points, then we consider it abnormal based on the current encoding method. Conversely, data points that can be ideally compressed by the current method are normal data points.

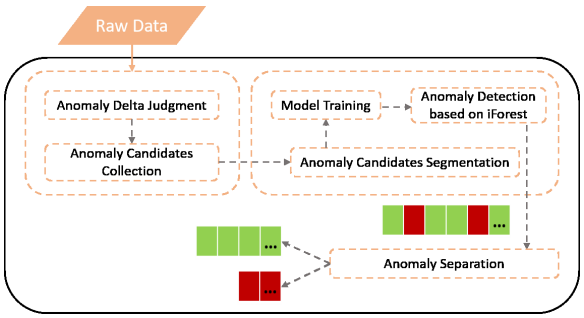


Fig. 3 Anomaly Detection Procession

4 Anomaly Detection

For data blocks that contain both abnormal and normal points, a compression method that can effectively compress normal points always do not work well on abnormal. Hence, it is necessary to separate abnormal points from normal points and process them individually. To achieve that, we introduce an anomaly detection strategy. As shown in figure 3, the strategy are divided into two stages: We first decide a anomaly candidate set $A_c = \{c_1, c_2, \ldots, c_n\}$ through abnormal w-deltas, thus reducing the scope of anomaly detection and increasing the detection accuracy. And then, the abnormal points are located base on Isolation Forest algorithm.

4.1 Determination of Anomaly Candidates

Before anomaly detection, we first collect points that possible abnormal, and they are treated as anomaly candidates, A_c. Only points in A_c are allowed to be involved in the anomaly detection instead of the entire time series. The reason for this is to narrow down the range and improve the accuracy of anomaly detection.

Section 3.2 mentions that points that cannot be ideally compressed by compression method for normal points (Window Delta) are abnormal

data points. Therefore, we first try to compress all points by Window Delta, and then back out the abnormal points from the abnormal encoding results.

The idea behind Window Delta is to store intermediate values, w-delta, rather than raw values. The w-delta stands for the difference between a value and the average within each sliding window with size 3, and the detailed approach is described in Section 5. The more extreme the w-delta values, the more storage space is required. As a consequence, extreme w-deltas are abnormal, and abnormal points could be detected by abnormal w-deltas.

Since an abnormal w-delta caused by three data points from its corresponding window, a abnormal point must exist among them. Accordingly, we denote the three data points as anomaly candidates of the abnormal w-delta, and the collection of anomaly candidates of all abnormal w-deltas is A_c. A_c surely contains all abnormal values.

In general, we collect anomaly candidates A_c through abnormal w-deltas. Suppose there is a dataset V, the specific approach is as follows:

1. Calculate all w-deltas and trace over them. The extreme w-deltas that beyond the range [-255,256] are regarded as abnormal w-deltas. The range is set to [-255,256] because it is sufficient to cover most w-deltas, and the required bit length for w-deltas within the range is reasonable.
2. Find all the data points that result in abnormal w-deltas, collect and de-duplicate them to derive the candidate set, A_c. A element in A_c is a tuple that include $c_k.v$ and $c_k.i$. The $c_k.v$ represent values and $c_k.i$ refer to index of c_k in V.

It is known that $c_k.i$ records the index of c_k, and the number of data points that result in a abnormal w-delta is three. Therefore, the set of $c_k.i$, $A_c.i$, is segmented continuous, which means that $A_c.i$ is composed of several numerically consecutive subsequence. For instance, there is an anomaly candidate set $A_c = \{(c_1.v, 3), (c_2.v, 4), (c_3.v, 5), (c_4.v, 8), (c_5.v, 9), (c_6.v, 10), (c_7.v, 11)\}$. From the perspective of $c_k.i$, the A_c is partitioned into $A_{c1} = \{(c_1.v, 3), (c_2.v, 4), (c_3.v, 5)\}$ and $A_{c2} = \{(c_4.v, 8), (c_5.v, 9), (c_6.v, 10), (c_7.v, 11)\}$, they are handled separately as anomaly detection units.

4.2 Anomaly detection based on Isolation Forest

Following determination of the anomaly candidates, A_c, the next stage is to recognize abnormal points among them.

Considering the speed and accuracy, Isolated Forest (iForest) is used to locate abnormal values. It is an efficient anomaly detection technique that builds multiple binary trees (iTrees) based on values by random sampling, thus indicating the degree of isolation of corresponding values from most using the depth of the nodes.

There are three main benefits of adopting iForest to identify abnormal points. Above all, iForest work well in scenarios where the values of abnormal points are different from normal. What's more, compared to distance-based methods and density-based methods, iForest dose not need to calculate metrics about distance or density, which can significantly increase calculation speed and reduce system overhead. At last, iForest is an integrated algorithm, which means its decisions are usually more reliable.

Suppose there is an anomaly candidate set $A_c = \{(c_1.v, c_1.i), (c_2.v, c_2.i), \ldots, (c_m.v, c_m.i)\}$. The iForest-based anomaly detection consists of three steps as shown in Algorithm 1:

1. Divide A_c into multiple detection units based on the index $c.i$, and then we get $A_c = \{A_{c1}, A_{c2}, \ldots\}$.
2. The longest detection unit is taken to train the iForest model. Training the iForest model by the longest detection unit rather than the whole A_c is aimed at saving the model training time.
3. Traverse the detection units in turn, and the iForest model is reused by them to identify abnormal points. Gather all values and indexes of detected abnormal points, and we get V_a. The record of indexes is for decompression.

After anomaly detection, V is divided into two sections: V_n (normal values) and V_a (abnormal values). Moreover, the storage structure changes accordingly: a whole data block is partitioned into two areas to store V_n and V_a separately. Compression methods for normal and abnormal values are described later in Section 5 and 6.

Algorithm 1 Anomaly Detection

Require: $A_c = c_1, c_2, \ldots, c_m$
Ensure: V_a
1: $V_a = \Phi$
2: $A_c = \{A_{c1}, A_{c2}, \ldots, A_{ct}\} = \text{Segmentation}(A_c)$
3: $A_{c.max} = \text{MaxLength}(A_c)$
4: iForest $= \text{ModelTrain}(A_{c.max})$
5: **for** $k = 1, 2, ..., t$ **do**
6: $\quad af_k = \text{iForest}(A_{ck})$
7: $\quad V_a = V_a \cup af_k$
8: **end for**
9: **return** V_a

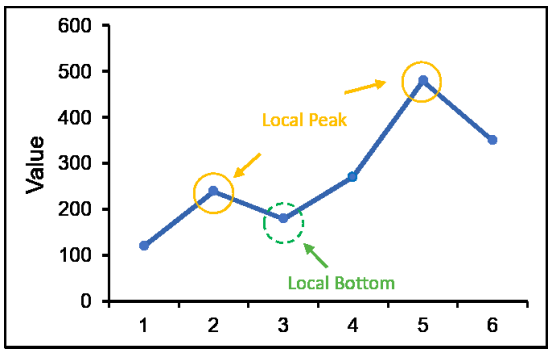


Fig. 4 Visualization of V_n

5 Compression of normal data

In the section, we discuss the compression method for V_n, Window Delta (WD). In many situations, we find that metric values tends to fluctuate within a range and not always in a linear trend. In addition, existing data compression methods have much room for optimization in compressing the data pattern. Therefore, We present WD, which is an optimized integer compression method based on Delta of Delta (DoD) of Gorilla. WD always achieves better compression ratios than DoD.

5.1 Delta of Delta

The primary approach of DoD is to calculate the second order difference and then store them by Variable Length Encoding. The detailed steps are as follows:

1. Recaord the raw value of the first data point by 4 bytes, v_1.
2. Calculate the second order difference: $d = (v_n - v_{n-1}) - (v_{n-1} - v_{n-2})$.
3. Allocate storage space and identifiers for each d based on the value. The storage space refers to the bit length required to store d, and identifiers are used to identify different bit lengths to enable decompression accurately. The specific allocation rules are indicated in Table 1, which is a Variable Length Encoding.

Taking the first 7 data points of the dataset in Figure 5a as an example, $V = \{120, 240, 180, 2400, 270, 480, 350\}$. After anomaly detection and elimination, $V_n = \{120, 240, 180, 270, 480, 350\}$ is generated, and it is represented by a line graph as shown Figure 4. With the second order difference calculation, the values to be stored turns out to be $D = \{120, 120, -180, 150, 120, -340\}$, and the encoded size is $12 \times 5 + 16 = 76$ bits. In fact, the storage space required for V_n after the same Variable Length Encoding is $12 \times 3 + 16 \times 3 = 84$ bits. It means that calculating the second order difference does not significantly improve the compression ratios. This is because DoD is skilled in encoding datasets with monotonic datait, and its compression effect is not ideal for frequently fluctuating data patterns.

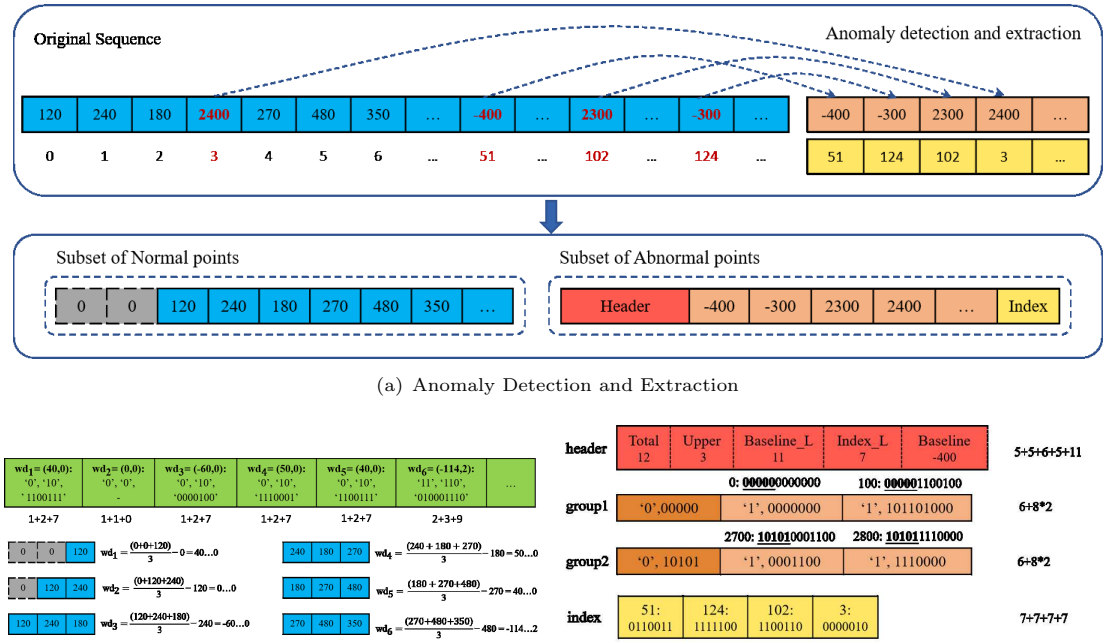
The frequently fluctuating could be characterized by **local peaks** and **local bottoms**. A local peak consists of three data points, where the middle value is larger than both sides. Taking V_n as an example, three consecutive data like $\{120, 240.180\}$ could be considered as a local peak. Similarly, we define data pattern such as $\{240, 180, 270\}$ forming a local bottom. It is not difficult to imagine that when there are frequent

d	Identifier	Bit Length	Total Bit Length
0	0	-	1
$[-63, 64]$	10	7	$2 + 7 = 9$
$[-255, 256]$	110	9	$3 + 9 = 12$
$[-2047, 2048]$	1110	12	$4 + 12 = 16$
> 2048	1111	32	$4 + 32 = 36$

Table 1 Allocation rule of bit length in DoD and WD

$wd_n.r$	Identifier
0	0
1	10
2	11

Table 2 Allocation rule of bit length for $wd_n.r$



(a) Anomaly Detection and Extraction

(b) Encoding for normal points (Window Delta)

(c) Encoding for abnormal points

Fig. 5 Example of AAC's workflow

local peaks or local bottoms in a dataset, whose overall pattern is jagged.

Whether local peak or local bottom, $(v_{i+1}-v_i)$ and $(v_i - v_{i-1})$ are often calculated with opposite signs, which may cause deltas $((v_{i+1} - v_i)$ - $(v_i - v_{i-1}))$ larger than expected. In V_n, random three continuous values, $\{v_{i-1}, v_i, v_{i+1}\}$, form a local peak or a local bottom, and thus the compression of DoD is not ideal. Generally, faced with the frequently fluctuating data patterns, DoD has difficulty in performing the desired compression effect.

5.2 Window Delta

To address the issue of frequently fluctuating data patterns affecting compression ratios, we propose Window Delta Encoding. The general idea of WD is to smooth local peaks and local bottoms in fluctuation data by a sliding window. It first calculates w-deltas and then encodes w-deltas through the Variable Length Encoding rules like DoD.

The point to WD is the calculation of w-deltas. Suppose the data to be compressed is $V = \{v_1, v_2, \ldots, v_n\}$, the specific practices are described briefly below:

1. Add two zero at the head of V, resulting $V^{'} = \{v_{-1}, v_0, v_1, v_2, \ldots, v_n\}$, $v_{-1} = v_0 = 0$.
2. Start from v_1 and iterating through each value in $V^{'}$, w-deltas are calculated by the formula shown below:

$$wd_k = \frac{(v_{k-2} + v_{k-1} + v_k)}{3} - v_{i(i=k-2,k-1,k)}$$

The v_i is variable and it is selected from a data in sliding window. As shown in Figure 6, different point in sliding window represent different formulas, and the final formula is derived by two steps. Firstly, we calculate the sum of $|d_i|$ with the three method. Secondly, since we prefer to keep values of w-delta as low as possible, the method with the smallest result is chosen. For a sequence, once the formula has been determined, it does not change. Due to the division operation, the calculated result consists of a quotient and a remainder: $wd =< q, r >$, and r could be 0, 1 or 2.
3. For each w-delta wd_n, only the value of $wd_k.q$ are stored, and $wd_k.r$ is recorded by identifier to satisfy the needs for storage space saving and data recovery:

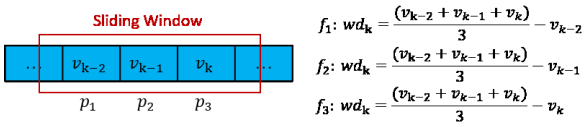


Fig. 6 Three calculation methods of wd_k

(1) Depending on the value of $wd_k.r$, different identifiers are stored. The identifiers are determined according to Table 2.
(2) Following $wd_k.r$, $wd_k.q$ is encoded by the same encoding rule as DoD (see Table 1).

The reason for setting the sliding window size to 3 is that: On the one hand, if the window size is too large, which means that the length of signal will not be limited within 2 bits, which cause the compression ratios to be less than ideal. On the other hand, tiny sliding windows might fail to achieve the purpose of smoothing local peaks and local bottoms, since the two patterns consist of at least three numbers.

We illustrate exactly how WD works on V_n in Figure 5a. Initially, two 0 values are placed in the head of the dataset, and then we get $V_n^{'} = \{0, 0, 120, 240, 180, 270, 480, 350\}$. Moreover, by analysis, formula f_1 will be applied to calculate w-deltas. Afterwards, starting from $v_1 = 240$, there are three data values $< 0, 0, 120 >$ in the sliding window at the moment. We compute the w-delta $(0 + 0 + 120)/3 - 0$, and obtain $wd_1 =< 40, 0 >$. Since $wd_1.q$ is between -63 and 64, it would be stored '10' followed by the value that occupies 7-bit length. Additionally, in view of $wd_1.r$ is 0, the identifier is set to '0'. As a result, 10 bits are needed to store the window delta of v_1. And so on, the intermediate results of WD are displayed in figure 5b. Under Window Delta Encoding, the eventual code length is 56 bits, it hold a 1.36× higher compression ratio than DoD Encoding.

6 Compression of abnormal data

Generally, abnormal points, V_a, tend to emerge at extreme local peaks or local bottoms, so they are always very scattered in value. For better compress results, we propose a dispersion tolerant encoding, which including a common-prefix based grouping encoding and a Variable Length Encoding. As a V_a consists of $V_a.v$ and $V_a.i$, common-prefix based grouping encoding and Variable Length Encoding are used to compress them separately.

The compression method of V_a involves three steps. Initially, the V_a is preprocessed so that a new $V_a^{'}$ is created. Then, $V_a^{'}.v$ are grouped according to fixed length common prefix, and those in the same group shares the common prefix. Finally, the indexes in original dataset of abnormal data, $V_a^{'}.i$, are encoded by a Variable Length Encoding.

6.1 Preprocession

The detailed procedures for preprocessing V_a can be described as follows:

1. Find out the maximum v_{max} and minimum v_{min} of $V_a.v$. Afterwards, the least bit length required to store v_{max} and v_{min} are registered as Total Length and Baseline Length in block header. The intention of Total Length is to dynamically standardize the bit length of $V_a.v$, which is capable of reducing the storage space needed. In the traditional approach, each integer occupies 32 bits by default, but the metric values are generally comparatively small. As a result, that method leads to wasted space. In addition, v_{min} is recorded as Baseline, which is used to preprocess V_a.
2. Traverse V_a to execute $v_i.v^{'} = v_i.v - v_{min}$ and reorder it by $v_i.v^{'}$. Then, we get a new set of $v_i^{'} = (v_i.v^{'}, v_i.i)$, $V_a^{'}$.

Taking the abnormal values in Figure 5a as an example, V_a = {(2400,3), (-400,51), (2300,102), (-300,124)}. The Baseline, Total Length and Baseline Length are -400, 12 and 11 respectively, and they are recorded in the header of the data block. After preprocession, V_a turns into $V_a^{'}$ = {(0,51),(100, 124),(2700,102),(2800,3)}. Afterwards, $V_a^{'}.v$ = {0, 100, 2700, 2800} and $V_a^{'}.i$ = {51, 124, 102, 3} are encoded by different method.

6.2 Common-prefix Based Grouping Encoding

$V_a^{'}.v$ is compressed by common-prefix based grouping encoding. The main approach is to split the binary of values in $V_a^{'}.v$ at a fixed position

to get upper and lower bits. Values with the same upper bits are collected together, and only the lower bits are stored. The upper bits are hold at the group header, which is shared by all values in the group. In doing so, compression for abnormal values is realized by reducing the number of redundant bits.

Finding the reasonable separation position is the key to this method. Note that too many or few groups may cause the grouping method to fail. Let the upper bit length is n, then there are 2^n groups in total. If there are large amounts of groups, in the worst case, different elements belong to different groups. What is more, in case the n has an extreme small value, the bit length of each value needs for storage must be long. Both of these states cause inefficient use of the shared bits. To avoid the above situations, we start with setting $n = 1$ to get a compression length l. Then we repeat the $n = n + 1$ operation and keep receiving new $l^{'}$. Until $l^{'} >= l$, the loop is terminated and $n = n - 1$ is executed.

Figure 5c graphically shows the encoding results of $V_a{}^{'}.v = \{0, 100, 2700, 2800\}$. At the outset $n = 1$, and the encoding length $l = 52$. Then $n = n+1$ is repeatedly executed and the encoding length gradually decreases until $n = 5$, at which point $l = 44$, which is greater than the case under $n = 4$,$l = 46$. This is where the process ends, and it can be derived that $n = 4$, which is also recorded as Upper just like Total Length, Baseline Length and Baseline. Consequently, we arrive at the grouping strategy: the upper bit length is 8 and the lower bit length is 2. The four values in $V_a{}^{'}.v$ are grouped by the upper bits, resulting in two groups whose headers are '00000' and '10101', where $v_1{}^{'}.v = 0$ and $v_2{}^{'}.v = 100$ are located in the group 1 and the remaining two data are attributed to the second one. That way, otherwise completely dispersed data points are efficiently grouped and aggregated

6.3 VL Encoding

In order to recover the timestamp-value pairs without disrupting the sequence of timestamp, we need to record the indexes of the abnormal values in the original sequence as well, $V_a{}^{'}.i$. They are encoded by a Variable Length encoding, that is to say we dynamically allocate storage space for them depending on their values. The bit length occupied by indexes from the same $V_a{}^{'}.i$ must be the uniform, so we look for the largest index and then recognize the length of non-zero bits in its binary form (Index Length), which is recorded in the block header as well.

In $V_a.i = \{51, 124, 102, 3\}$, the maximum value, 124, takes 7 bits to store. As a result, Index Length is set to 7, and all the values of $V_a{}^{'}.i$ are stored with 7-bit length.

7 Experiment

To evaluate the performance of the AAC (Anomaly Aware Compression) algorithm, we compare it with typical integer lossless compression methods (Gorilla and Simple8b) in terms of compression ratio over a significant number of publicly available real datasets. In addition, the compression effect of AAC is affected by anomaly detection strategy, so the accuracy of anomaly detection is validated at Subsection 7.3.

Our experiment is conducted on a Windows 10 64-bit Operating System with a 1.8GHz Intel Core i7-8550U processor and 16GB of RAM.

7.1 Experiment Setup

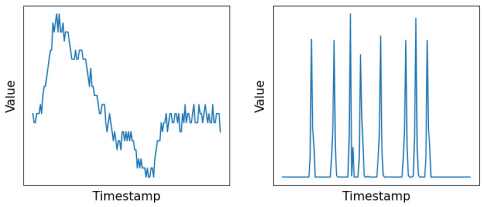
7.1.1 Datasets

In order to guarantee the universality of AAC, we guided by the principle of covering many scenarios and a wide distribution of values when selecting the dataset. Eventually, we select the following datasets: **UCI**, **UCR**, and **NAB**.

Both **UCI** (University of California Irvine) and **UCR** are open source and reliable data warehouses diffusely appear in machine learning, and they capture lots of true time series data from various fields. Meanwhile, **NAB** is an accessable dataset offered by *Numenta*, which serves the evaluation of the performance of anomaly detection programs about streaming time series and has become a new benchmark.

To assess the adaptability of AAC to data arriving from different scenarios, 11 datasets are selected from the above-mentioned data sources for our experiments, covering transportation, industry, human life, etc. Figure 7 is a visual indication of the **Temperature** and **DiskWrite**

datasets, which represent two universal distribution patterns on values: a pattern of tiny fluctuations with local peak and local bottoms in the majority, and another pattern of sharp fluctuations accompanied scattered outliers. It can be seen that the metric values of time series gained from the real world is not always stable, but constantly change within a range. As demonstrated in Figure 7a, the overall trend of the data is steady but accompanied by a lots of little glitches. Besides, some external factors might lead to numerical mutation, see Figure 7b.



(a) Temperature (b) DiskWrite

Fig. 7 Example visualization of two datasets

We categorize the 11 datasets by the presence or absence of abnormal values as the judgment criterion, and apply them to validate WD (6 datasets) and AAC (5 datasets), respectively.

7.1.2 Comparison Objects

A total of four compression algorithms participated in the experimental comparison, including two broadly recognized and adopted algorithms and the two proposed in the paper.

- **Simple8b:** A non-negative compression algorithm based on the concept of word packaging. Data points to be encoded are packed sequentially into 64-bit words.
- **Delta of Delta (DoD):** An advanced integer compression approach that appears for the first time in Gorilla and is now widely adopted in practice. It stores the second order difference by Variable Length Encoding.
- **Window Delta (WD):** It overcomes the impact of the frequently fluctuating data pattern and reduces the storage space of w-deltas through a new w-delta calculation method.
- **Anomaly Aware Compression (AAC):** It takes advantage of the anomaly detection strategy to separate abnormal values from the normal and then compresses them separately.

7.2 Compression Ratio

In this section, we evaluate the compression ratios of Window Delta Encoding (WD) and Anomaly-Aware Compression algorithms (AAC). The compression ratio is calculated as follows:

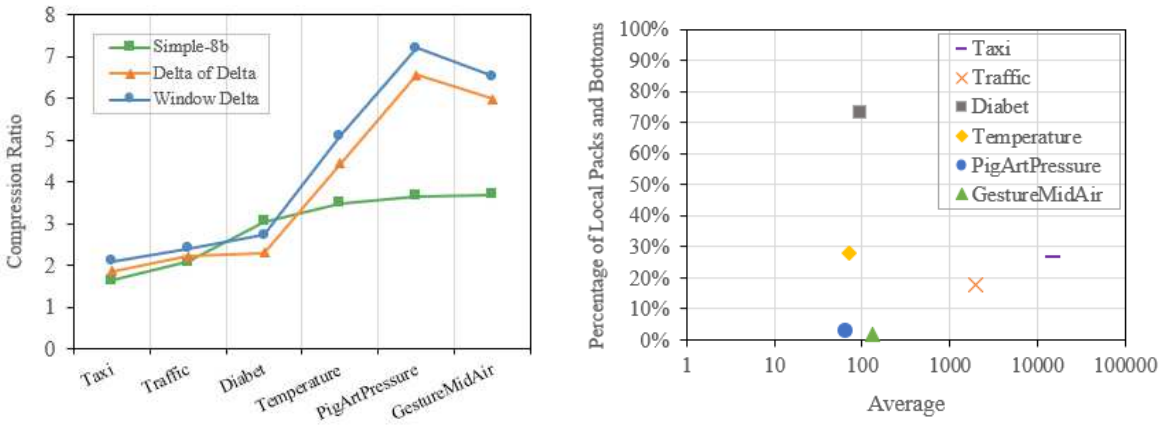
$$CR = \frac{Bit\ length\ of\ row\ data}{Bit\ length\ of\ compressed\ data}$$

Normally, the larger the compression ratio, the better the compression algorithm.

In reality, time series data generated in a variety of industrial scenarios are continuously collected by sensors, and then they are regularly delivered to databases waiting to be written to disk. In the case of Influxdb, for example, data are accumulated in memtables (in-memory tables) upon arrival at a database. When a memory table is filled up, they will be swiped to disk as a table unit. When a memtable is filled up, it is queued up and waits to be flushed to disk. As a result, in order to simulate the above data writing strategy, we chose to segment datasets into equal parts (default 512 points), and we regard the data segments as compression units in our experiments. In besides, abnormal points occur intermittently in the real world, and thus there are always some compression units that do not contain abnormal points. Taking this situation into account, we record the experimental results only for the cases where abnormal points can be detected.

7.2.1 Window Delta

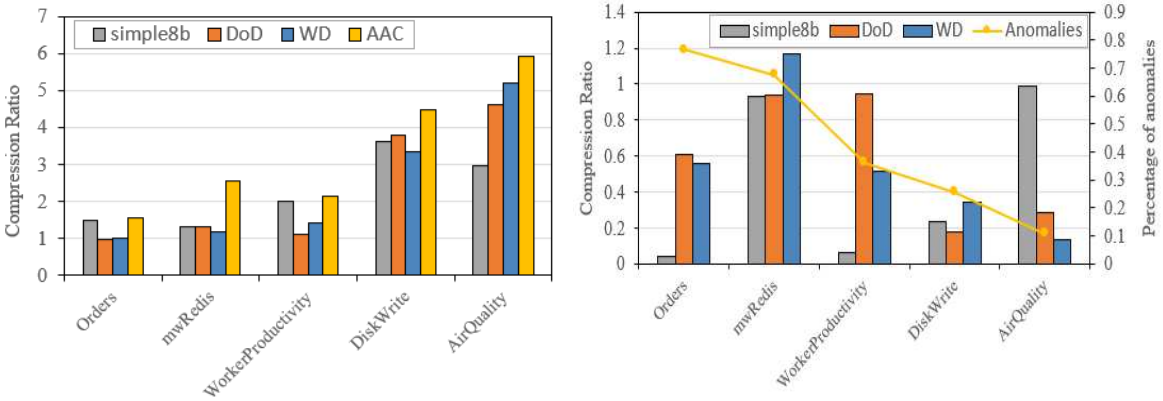
Figure 8a presents the compression ratios on datasets without abnormal points. Because of the absence of anomalies, we only compare the performance of simple8b, DoD and WD. According to the result, it is convenient to find that Simple8b is the worst in most cases, because its ability is highly affected by data fluctuations. Furthermore, the effects of DoD and Wd are quite synchronized, except that WD always has a better compression ratio than DoD, which proves the progress of WD



(a) Comparison of compression ratios

(b) The overall value distributions and oscillations of different datasets

Fig. 8 Compression effect of Window Delta



(a) Comparison of compression ratios

(b) Improvement over other methods

Fig. 9 Compression effect of AAC

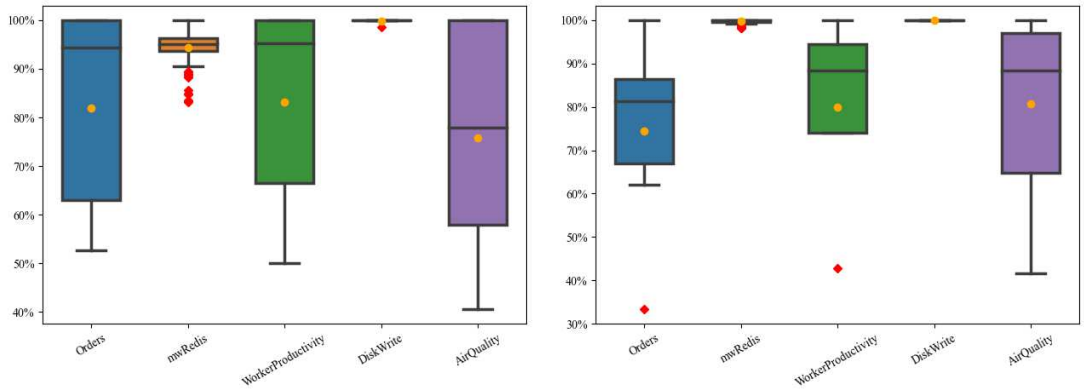
based on DoD. Combining the average and fluctuation of the 6 datasets involved (Figure 8b), it can be concluded that all algorithms are affected by the two properties in certain degree. In particular, regardless of the degree of fluctuation in compressed datasets, WD essentially has the best compression ratios.

7.2.2 Anomaly Aware Compression

Figure 9a compares the compression effect of the four compression algorithms, and Figure 9b shows the improvement rate of AAC over the other methods. In this paper, the improvement rate is defined as shown below:

$$IR = \frac{CR_{AAC} - CR_{Others}}{CR_{Others}}$$

On the one hand, we can clearly understand from Figure 9a that AAC always has the best compression ratio, and each of the other algorithms demonstrates their advantages on different datasets. On the other hand, as can be seen in Figure 9b, in contrast to others, AAC has more or less improved the compression rate. The graph also reflects the correlation between the improvement rate and the percentage of outliers in a dataset: the improvement rate on WD is almost proportional to the percentage of outliers. This observation is anticipated because the more



(a) Completeness (b) Accuracy

Fig. 10 Boxplot of consequent of the anomaly detection in AAC

anomalies in a dataset, the worse the WD compression ratio is, while the separate treatment of AAC for outliers can greatly mitigate this negative impact.

7.3 Anomaly Detection of AAC

AAC is useful only if abnormal points are accurately identified and separated. Therefore, it is necessary to verify the effectiveness of anomaly detection strategy. In this paper, two metrics are used for validating the capability of AAC for anomaly detection: completeness and accuracy.

Completeness. Completeness estimates the degree of improvement of anomalies in time series. The purpose of removing the anomalies from time series is to decrease the number of abnormal deltas, and the fewer the abnormal deltas, the lower the anomalies of the time series. Therefore, completeness can be quantified as $\frac{N_d - M_d}{N_d}$, where N_d tands for the number of abnormal deltas in the original series, and M_d denotes the count of anomalous deltas of the sequence after outlier elimination. Figure 10a displays the distribution of the completeness on different datasets. Anomaly detection in AAC could discover half of the outliers at least, and in most cases, it is capable of finding all outliers.

Accuracy. Accuracy represents the percentage of real anomalies among all data points recognized as anomalies by the anomaly detection algorithm. We argue that for a data point, when we drop it from the original sequence, the number of abnormal deltas arising from the new sequence, then it is a real anomaly. Accordingly, accuracy is calculated as $\frac{N_{true}}{N_{detected}}$. $N_{detected}$ is the total number of anomalies detected by AAC, and N_{true} means the number of points that are true anomalies among those discovered by AAC. Distribution of accuracy is exhibited in Figure 10b. In some datasets, the outliers identified by AAC are basically valid, while the accuracy in some cases needs to be improved.

8 Conclusion and Future

In the paper, an anomaly aware lossless time series compression algorithm named AAC is proposed. The design goal of AAC is to improve compression ratios and thus decrease storage costs for green computing. AAC achieves higher compression ratios by identifying and separately compressing abnormal points in time series. We propose an anomaly detection strategy to separate abnormal and normal points, and two encoding method to encode them respectively. Finally, it is experimentally demonstrated that AAC compresses better than DoD and Simple8b.

In this work, the scenario of compressing multidimensional time series data is not considered. In the future, the question of how to make our method applicable in multidimensional data compression will be explored.

Declarations

- Funding: This work is supported by the National Key R&D Program of China under

No. 2021YFB2700500, 2021YFB2700502 and 82004499, the Open Fund of Key Laboratory of Civil Aviation Smart Airport Theory and System, Civil Aviation University of China under No. SATS202206, the Open Fund of Key Laboratory of Complex Electronic System Simulation under No. 614201002022205, the National Natural Science Foundation of China under No. U20B2050

- Conflict of interest: No potential conflict of interest was reported by the authors.
- Ethics approval: Not applicable.
- Data availability: The datasets generated during and/or analysed involved in the study are available from the corresponding author, Liang Liu, if the request is reasonable.
- Author contributions: Jingwen Meng: Conceptualization, Methodology, Software, Data curation, Formal analysis, Investigation, Writing - Original Draft, Writing - Review & Editing; Liang Liu: Methodology, Validation, Funding acquisition, Writing - Review & Editing ; Yulei Liu: Project administration, Supervision, Writing - Review & Editing; Ning Wang: Writing - Review & Editing;
- Consent to publish: All of the authors have approved the contents of this paper and have agreed to the submission policies of Peer-to-Peer Networking and Applications.

References

[1] Fu Tc (2011) A review on time series data mining. Engineering Applications of Artificial Intelligence 24(1):164–181

[2] Brutlag JD (2000) Aberrant behavior detection in time series for network monitoring. In: LISA, pp 139–146

[3] Koprinska I, Wu D, Wang Z (2018) Convolutional neural networks for energy time series forecasting. In: 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, pp 1–8

[4] Sezer OB, Gudelek MU, Ozbayoglu AM (2020) Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Applied soft computing 90:106181

[5] Juang WC, Huang SJ, Huang FD, et al (2017) Application of time series analysis in modelling and forecasting emergency department visits in a medical centre in southern taiwan. BMJ open 7(11):e018628

[6] Naqvi SNZ, Yfantidou S, Zimányi E (2017) Time series databases and influxdb. Studienarbeit, Université Libre de Bruxelles 12

[7] Bader A, Kopp O, Falkenthal M (2017) Survey and comparison of open source time series databases. Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband

[8] Pelkonen T, Franklin S, Teller J, et al (2015) Gorilla: A fast, scalable, in-memory time series database. Proceedings of the VLDB Endowment 8(12):1816–1827

[9] Moon A, Park J, Song YJ (2022) Prediction of compression ratio for transform-based lossy compression in time-series datasets. In: 2022 24th International Conference on Advanced Communication Technology (ICACT), IEEE, pp 142–146

[10] Hollmig G, Horne M, Leimkühler S, et al (2017) An evaluation of combinations of lossy compression and change-detection approaches for time-series data. Information Systems 65:65–77

[11] Moon A, Kim J, Zhang J, et al (2017) Lossy compression on iot big data by exploiting spatiotemporal correlation. In: 2017 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, pp 1–7

[12] Wu D, Singh AK, Agrawal D, et al (1996) Efficient retrieval for browsing large image databases. Conference on Information and Knowledge Management

[13] Guo J, Xie R, Guowen J (2019) An efficient method for nmr data compression based on fast singular value decomposition. IEEE Geoscience and Remote Sensing Letters

[14] Agrawal R, Faloutsos C, Swami AN (1993) Efficient similarity search in sequence

databases. Lecture Notes in Computer Science

[15] Ratanamahatana C, Keogh E, Bagnall AJ, et al (2005) A novel bit level time series representation with implication of similarity search and clustering. In: Advances in Knowledge Discovery and Data Mining: 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005. Proceedings 9, Springer, pp 771–777

[16] Kithulgoda CI, Pears R, Naeem MA (2018) The incremental fourier classifier: Leveraging the discrete fourier transform for classifying high speed data streams. Expert Systems With Applications

[17] Keogh E, Chakrabarti K, Pazzani M, et al (2001) Locally adaptive dimensionality reduction for indexing large time series databases. In: Proceedings of the 2001 ACM SIGMOD international conference on Management of data, pp 151–162

[18] Azar J, Makhoul A, Couturier R, et al (2020) Robust iot time series classification with data compression and deep learning. Neurocomputing 398:222–234

[19] Keogh E, Chakrabarti K, Pazzani M, et al (2001) Dimensionality reduction for fast similarity search in large time series databases. Knowledge and information Systems 3:263–286

[20] Luo G, Yi K, Cheng SW, et al (2015) Piecewise linear approximation of streaming time series data with max-error guarantees. International Conference on Data Engineering

[21] Hosseini SA, Ghassemian H (2016) Hyperspectral data feature extraction using rational function curve fitting. International Journal of Pattern Recognition and Artificial Intelligence

[22] Fink E, Gandhi HS (2011) Compression of time series by extracting major extrema. Journal of Experimental & Theoretical Artificial Intelligence 23(2):255–270

[23] Gao J, Ji W, Zhang L, et al (2020) Fast piecewise polynomial fitting of time-series data for streaming computing. IEEE Access 8:43764–43775

[24] Di S, Cappello F (2016) Fast error-bounded lossy hpc data compression with sz. In: 2016 ieee international parallel and distributed processing symposium (ipdps), IEEE, pp 730–739

[25] Chandak S, Tatwawadi K, Wen C, et al (2020) Lfzip: Lossy compression of multivariate floating-point time series data via improved prediction. In: 2020 Data Compression Conference (DCC), IEEE, pp 342–351

[26] Burtscher M, Ratanaworabhan P (2008) Fpc: A high-speed compressor for double-precision floating-point data. IEEE transactions on computers 58(1):18–31

[27] Bruno A, Nardini FM, Pibiri GE, et al (2021) Tsxor: A simple time series compression algorithm. In: String Processing and Information Retrieval: 28th International Symposium, SPIRE 2021, Lille, France, October 4–6, 2021, Proceedings 28, Springer, pp 217–223

[28] Anh VN, Moffat A (2010) Index compression using 64-bit words. Software: Practice and Experience 40(2):131–147

[29] Candra R, Madenda S, Sudiro SA, et al (2017) The implementation of an efficient zigzag scan. Journal of Telecommunication, Electronic and Computer Engineering (JTEC) 9(2):95–98

[30] Zukowski M, Heman S, Nes N, et al (2006) Super-scalar ram-cpu cache compression. In: 22nd International Conference on Data Engineering (ICDE'06), IEEE, pp 59–59

[31] Al Hasib A, Cebrian JM, Natvig L (2015) V-pfordelta: Data compression for energy efficient computation of time series. In: 2015 IEEE 22nd International Conference on High Performance Computing (HiPC), IEEE, pp 416–425

[32] Yan H, Ding S, Suel T (2009) Inverted index compression and query processing with optimized document ordering. The Web Conference

[33] Chambi S, Lemire D, Kaser O, et al (2016) Better bitmap performance with roaring bitmaps. Software: practice and experience 46(5):709–719

[34] Blalock D, Madden S, Guttag J (2018) Sprintz: Time series compression for the internet of things. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2(3):1–23