Class-Based Delta-Encoding for High-Speed Train Data Stream

Yangxin Lin [1], Ping Wang [2,1], Jinlong Lin [1], Meng Ma [3], Ling Liu [4,1], Lin Ma [4]

[1] School of Software & Microelectronic, Peking University, Beijing, China
[2] National Engineering Research Center for Software Engineering, Peking University, Beijing, China
[3] School of Electronics Engineering and Computer Science, Peking University, Beijing, China
[4] Beijing National Railway Research & Design Institute of Signal & Communication Ltd, Beijing, China

martinlin@pku.edu.cn; pwang@pku.edu.cn; linjl@ss.pku.edu.cn;
mameng@pku.edu.cn; 627ll@crscd.com.cn; malin@crscd.com.cn;

***Abstract*—Railway transportation plays an important role in both economic and social development. The requirements of the railway traffic increase in recent decades. In order to meet the growing demand, a new generation control system of railway transportation emerges. It consists of collection, transmission, analysis and scheduling module. In such a context, an information transmission system is built to connect trains and scheduling center. However, the infrastructure of the railway system cannot provide enough bandwidth for such amount of data. As a result, the efficiency of data transmission cannot be ensured. In this paper, we focus on the compression algorithm that reduce the amount of transmitted data and improve the system performance. Based on the analysis of the common algorithms, an efficient compression algorithm, named delta-encoding, is proposed. It consists of two steps: preprocessing and compression. Delta-encoding utilizes a class-based difference model, which reduces the data redundancy, to realize a preprocessing algorithm. With the combination of preprocessing algorithm and a regular compression algorithm, delta-encoding has better performance on compression ratio, and becomes a universal hybrid algorithm for structured data in IoT system rather than a specific algorithm in high-speed train system. Finally, several experiments are provided to prove that delta-encoding have advantages in both compression ratio and compression time.**

Keywords —Delta-encoding; Juridical Recorder Unit; High-speed Train

I. INTRODUCTION

Railway transportation plays an important role in both economic and social development. In recent years, the requirements of the railway traffic increase annually, and high-speed railway system is becoming one of the most important part of the transportation industry. In order to meet the growing demand, a new generation control system of railway transportation emerges, which consists of information collection, network transmission, data analysis, and scheduling module [1]. In high-speed railway system, by analyzing the real-time status of trains, the train scheduling center could evaluate the state of the system and make appropriate decision to guarantee the efficiency of the system. It is inevitable that large amount of data need to be uploaded in real time. However, the infrastructure of the railway system cannot provide enough bandwidth for such amount of data. So far, there have been two solutions to solving this problem. The first one is improving the transmission ability of the infrastructure, while the other one is reducing the amount of the data that need to be transmitted. The first approach is impractical because of the restriction of power and computation resource. Instead, the compression algorithms have been widely discussed to improve the performance of such systems.

The high-speed train data stream consists of structural operational records, and control/scheduling commands. They represents the train status and the message between train and railway-side devices, such as balise and track circuit. In order to provide real-time information for the scheduling center, trains have to upload their status in a short period. However, trains stay in a stable status in most of their running time. Short upload period will lead to uploading more redundant information. Although the commonly used data compression algorithms can help reducing the bandwidth, they are not designed for specific data structure so that they cannot minimize the compressed size. In this paper, we present a compression algorithm, named delta-encoding, for high-speed train system. With the combination of preprocessing algorithm and a regular compression algorithm, delta-encoding has better performance, and becomes a universal hybrid algorithm for structured data in IoT system rather than a specific algorithm in high-speed train system. The contribution of this paper are threefold:

- First, based on difference algorithm and the characteristic of high-speed train data stream, a class-based difference model for data compression is proposed.
- Next, a compression algorithm is designed with the difference model, and has a better efficiency than common algorithms for high-speed train data stream.
- Finally, based on the class-based difference model, a universal hybrid algorithm is proposed for structured data in IoT system.

The rest of this paper is organized as follows: Some background on lossless compression algorithms are introduced in Section II. Section III introduces the structure of the data stream in high-speed train, and analyzes the problem in data compression with commonly used algorithms. In Section IV, we present a hybrid compression algorithm for high-speed train data stream. Experiences and evaluation of the proposed algorithm are provided in Section V. Section VI concludes the paper and discusses the future works.

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

II. Related Works

Lossless data compression is a class of data compression algorithms that can reconstruct exactly the original data from the compressed data. Lossless compression is commonly used in applications, such as text compression, where the loss of even a single bit is unacceptable [2-3]. In order to ensure the integrity of original data in high-speed train system, we will limit ourselves to the consideration of only lossless compression in this paper.

One of the common techniques of lossless compression is the family of textural-substitution algorithm, of which Lempel-Ziv is the most popular [2]. LZ77 algorithm forms the basis of most modern compression algorithm. It maintains a dictionary using triples representing offset, run length, and a deviating character. The offset is how far from the start of the file a given phrase starts at, and the run length is how many characters past the offset are part of the phrase. The deviating character is an indication that a new phrase was found, and that phrase is equal to the phrase from offset to offset+length plus the deviating character [4]. LZ77 also introduces the concept of a "sliding window", which is used to restrict the time of creating triples. At the end of the twentieth century, there are many variants of LZ77 algorithm spring up, include LZSS, LZO, Deflate, and LZMA. Deflate is invented by Phil Katz [5], which is the combination of LZ77 and Huffman coding. Deflate impressed people with a better performance compared to those previous algorithms. The Lempel-Ziv-Markov chain algorithm (LZMA) is another variant of the basic LZ77 algorithm, and was first used in the 7z format of the 7-Zip archiver [6]. LZMA is considerably improved in compression ratio over most other LZ variants mainly due to the bitwise method of compression rather than bytewise. Another study [7] reports that, in most cases, LZMA has better performance than Deflate [5] and Bzip2 [10] in compression ratio.

Another technique of lossless compression is Burrows-Wheeler transform (BWT). Different from Lempel-Ziv family, BWT is non-dictionary algorithms [8]. BWT rearranges a character string into runs of similar characters [9]. Bzip2 is an open source implementation of BWT [10]. It combines several lightweight algorithms such as run-length encoding, move-to-front transform, and huffman code [11]. The operating principles of Bzip2 is so simple that they achieve a very good compromise between speed and compression ratio.

Common lossless algorithms are widely used in practical, but they still need to be improved in particular cases. When the original data has a large amount of redundant information, common algorithm will use more resource to deal with them. If an improved algorithm can reduce the redundant information properly, the compressed size may decrease in some conditions. Psounis K [12] proposes a class-based delta-encoding and a scalable scheme to perform delta-encoding on dynamic web-traffic. The proposed approach increases the web-caching performance. The idea is to group documents into classes, and store one document per class as base-data on the server-side. Experimental results report that class-based delta-encoding combined with compression reduces the bandwidth consumption by a factor of 30, and the latency perceived by most users by a factor of 10 on average.

Hu Y [13] addresses the problem of efficiently modeling identifier collections occurring in RFID-based item-tracking applications and databases. This paper proposes a data compression algorithm for RFID data based on bitmap. The main idea of this paper is that, the Electronic Product Codes list can be represented using a bitmap data type, which can lead to significant storage savings.

III. High-Speed Train Data Stream

Juridical Recorder Unit (JRU) is a recorder that records the movement authorities received and the actions of the driver or operator [14]. Besides, JRU records the critical data of the train, such as position, speed, etc. Basing on the JRU data, we can detect and diagnose the fault in the high-speed train system. For the purpose of supervising the high-speed train system, the train scheduling center need to get and analyze real-time JRU data. As a result, this will bring hundreds of gigabytes of JRU data transfer volume per day. That is why JRU data becomes one of the main source of the data stream in high-speed train data system[15].

A. Data Format

The format of JRU data is demonstrated in Figure 1 [16]. Each piece of JRU data consists of different fields. Table I lists the meaning of these fields. As illustrated in Figure 1, the content of *NID_MESSAGE* and *L_MESSAGE* determine the format and length of *Variables*. For simplicity in the remainder of this paper, we call the field *Variables* as the suffix of JRU data, and the rest part as the prefix.

The content of the suffix of JRU data is defined in [16-18], the suffix records two different kinds of information individually. The first one is the changes of the train status, which is recorded in a fixed-length bit stream. The other is the communication data between train and railway-side devices or train scheduling center. Different from the status change data, the communication data, which is called *message* in [18], has more complex structure. Each message has a prefix, which records the unique message identifier and length of message. Moreover, the rest part of the message, which is called the suffix, consists of few well-defined packets. Similarly, the packet also consists of prefix and suffix, and has an identifier in the prefix. Figure 2 illustrates the format of two different information in JRU data. As mentioned above, there are three layers in JRU data, include JRU data layer, message layer and packet layer. The relationship between them can be seen in Figure 2(b). As note that, all of the identifier in prefix represents the content in suffix. In other words, the content of each piece of JRU data can be expressed by a triple of identifiers (*JID*, *MID*, (PID_1,…, PID_n)), which is called Format Identifier (FID). JID indicates the field *NID_MESSAGE* in the prefix of JRU data; MID and PID refer to the identifier of message and packet respectively.

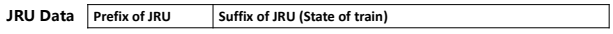
Fields	NID_MESSAGE	L_MESSAGE	DATE	TIME	TRAIN_POSITON
Length(Bits)	8	11	16	22	75

V_TRAIN	NID_DRIVER	NID_ENGINE	LEVEL	MODE	Variables
7	384	24	3	4	From 0 to relate to NID_MESSAGE

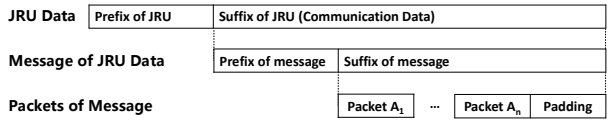
Figure 1. Format of Each Piece of JRU Data

Table I. JRU Data Format

Fields	Remarks
NID_MESSAGE	Message identification number
L_MESSAGE	Message length including fields 1 to N
DATE	Current date
TIME	Current time
TRAIN_POSITON	Current train position
V_TRAIN	Current train speed
NID_DRIVER	Driver identifier
NID_ENGINE	On-board ETCS identity
LEVEL	Current level
MODE	Current mode
Variables	Data associated to the message. Its length depends on the message content, but it's always rounded up to a bytes unit.



(a) JRU Data with state of train



(b) JRU Data with communication data

Figure 2. Structure of JRU data

B. Compression Process of JRU Data

To ensure the validity of uploaded data, the compression process have to be executed as soon as possible when a new piece of JRU data is generated. However, each piece of JRU data contains around 108 bytes in average, and is so short that its content approximates to a random string. It is unwise to compress JRU data piece by piece, because random string is too difficult for common algorithms to compress. In order to avoid inefficient compression, we would like to compress JRU data by group. As a result, a buffer is need for this, and will delay the transmission.

Considering the capability of transmission system, we divide JRU data into two categories: real-time data and delayed data. Real-time data denotes the new generated JRU data. A real-time data buffer is created to hold those data. In general, each train generates new JRU data every 125 millisecond [16]. In order to balance the compression ratio against the delay, the size of the real-time data buffer should less than 20 piece of data. Delayed data denotes the real-time data that cannot be uploaded in time. The delay may be caused by small bandwidth or device fault. In other words, real-time data will be transformed to delayed data if in need. Identically, delayed data is held in a delayed data buffer, which holds more data than real-time buffer. As note that, the delayed data will be uploaded again when the network conditions become better.

C. Characteristic of JRU Data

As mentioned above, when two pieces of JRU data have the same FID, they will have the same structure. As result, there is a great possibility that two piece of data will share more content. Because of the time interval between two pieces of JRU data is small, part of the information in JRU data may not change, and this will bring unnecessary redundancy, which will take up more bandwidth. To evaluate the proportion of unnecessary redundant information, we divide the content of JRU data into static and dynamic part. Static part denotes the content that seldom changed in the running processes of the train, such as *DATE*, *NID_ENGINE*, *NID_DRIVER* field etc. Dynamic part is the rest part of JRU, which could change at any time, such as *TIME*, *V_TRAIN*, *TRAIN_POSITION* field etc. The content of static part is the one of the source of unnecessary redundancy. For instance, there are no less than 438 bits content of the JRU prefix (around $438/(108\times 8)\approx 50\%$ content) will not change when the train is running in stable status. Moreover, if two pieces of JRU data have the same FID, this will cause more unnecessary redundancy. In extreme case, when the length of Variables field is zero, 90% content of two pieces of JRU data are the same.

Basing on the characteristic of JRU data, we define few concept below:

- *Public Byte*: bytes appear in the same position of both pieces of JRU data
- *Private Byte*: bytes other than public bytes of two pieces of JRU data
- *Public String*: a string consists of continuous public byte
- *Private String*: a string consists of continuous private byte

In other words, each piece of JRU data is composed of public strings and private strings. Figure 3 illustrates the relationship between public strings and private strings in JRU data. In the Figure 3, $S_{ij,n}$ indicates the n^{th} public string between two pieces of JRU data i and j, and $D_{ij,n}$ indicates the n^{th} private string. As illustrate, Figure 3 shows a special situation that each piece of JRU data shares the same public strings with each other. Because of the existence of a great amount of public string, the common compression algorithms

have to use more space to record those repeated strings. Such as deflate algorithm, public strings will be replaced by triples. However, even if these strings have been compressed, triples will still take up a lot of space. Therefore, a preprocessing algorithm, which remove the public string as much as possible, is needed before compression.

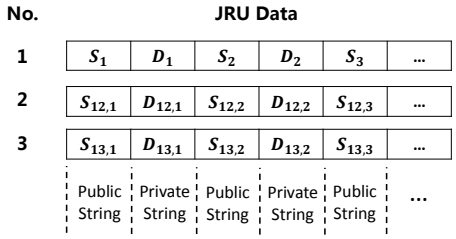


Figure 3. JRU Data Divided by Public and Private Strings

IV. DELTA-ENCODING ALGORITHM

A. Basic Idea

The basic idea of delta-encoding is reducing the unnecessary redundant information in original data by a preprocessing algorithm to improve the compression ratio of common compression algorithm. The reference [12] and [13] puts forward two similar solution, which is based on the difference algorithm. Difference algorithm utilizes the data structure to reduce the redundancy and avoid the repeated transmission. In this paper, preprocessing algorithm will refer to difference algorithm. We will discuss the difference algorithm and the improvement of it below.

Figure 4 demonstrates the result of compressing the data, which is illustrated in Figure 3, with difference algorithm. Difference algorithm compresses data by encoding one piece of data in terms of base-data which is set as the first piece of JRU data in this case. After that, other pieces of JRU data will take bitwise difference operation with base-data, and the result is called delta, which records the difference between two pieces of data. Note that, in this case, delta is composed of private strings. Difference algorithm uses delta and base-data to represent the content in original data with less space. As illustrated in Figure 4, when the volume of original data is huge enough, the compressed size with difference algorithm will approximate to the proportion of private strings. Nevertheless, high reduction is under two assumptions of huge data volume and low proportion of private strings. In practical, however, both of two assumptions may be invalid. In order to improve the performance of difference algorithm, we need to consider the relationship between those two factors and compression efficiency.

On the one hand, without the huge data volume, we cannot ignore the fact that recording delta need extra space, because the exact position of delta in the original data is need to be saved, and this will increase the compressed size. The reference [12] and [13] use bitmap and difference-coding to record delta respectively. Bitmap denotes if there are public bytes between two pieces of JRU data. The original data could be expressed by private strings and the bitmap. Different from the bitmap, difference-coding uses NOR operation to get the difference. Note that, the NOR operation will bring sequential '*0*' when there are public strings between two pieces of JRU data. To reduce redundancy, a encode algorithm, such as run-length, is needed. Because the difference-coding could lead to overmuch '*0*' fragments, which will take up more space than bitmap, we propose bitmap way in this paper.

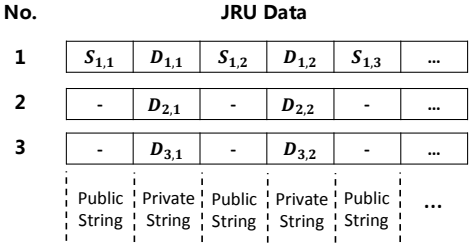


Figure 4. JRU Data with Difference Algorithm

On the other hand, it is difficult to decrease the proportion of private strings with only one base-data. Different structures of JRU data will lead to widely different in both content and length between two pieces of JRU data. In addition, the compressed size will increase with the increase of delta (private strings). Therefore, instead of single base-data, we need to involve multi-base-data to overcome this disadvantage. By finding a proper base-data for each piece of JRU data, we could decrease the proportion of private strings. To locate base-data, the structure of data should be under consideration, because same structure between two pieces of JRU data will lead to higher similarity and less difference (private strings). As result, each piece of JRU data should have the same structure with its base-data. On this premise, the problem is turned into finding the most similar data. Different from the base-file choosing algorithm in [12], we solve this problem by a more precise approach instead of random sampling. As mentioned above, FID can represent the structure of each piece of JRU data. So we could classify the dataset by FID. Moreover, an equivalence class of FID, in which elements have the same FID, is defined. Basing on this classification, we could quickly locate the most similar data, which is the proper base-data, in equivalence class.

B. Notation

We define several notation to help the descrption of the delta-encoding algorithm, they are shown in Table II.

Table II. Notation of Delta-Encoding

Notation	Description
F_i	A piece of JRU data with a serial number *i*. In the delta-encoding, each piece of JRU data has different serial number.
/	Separator between two pieces of JRU data.
x:*y*	The connection between string *x* and *y*.
M	Original JRU data, which is composed of several pieces of data. It can be expressed as $M=F_{a1}:/:F_{a2}:/:F_{a3}:\ \ldots\ldots$
E(*x*)	The feature of JRU data *x*. It is used for the classification of JRU data. In this paper, it is equal to the FID.

R	An equivalence relation, xRy means that x and y have the same feature.
$[x]_R$	Equivalence class of x under the equivalence relation R, it can be expressed as $[x]_R = \{y \mid y \in M \wedge E(x)=E(y)\}$
$\|x\|$	The length of string x (in byte)
$F_i[n]$	The content of n^{th} byte in F_i
$Diff(F_i,F_j)$	Difference between F_i and F_j (in byte). It is recorded in bitmap. If $F_i[n]=F_j[n]$, the n^{th} bit of $Diff(F_i,F_j)$ is 0, otherwise, the n^{th} bit of $Diff(F_i,F_j)$ is 1.
$S_{ij,n}$	The n^{th} public string between F_i and F_j
$D_{ij,n}$	The n^{th} private string between F_i and F_j
N_i	The serial number of the base-data of F_i
F_i^*	The result of F_i after being preprocessed.
$Prefix(F_i^*)$	Prefix of F_i^* which record the serial number of the base-data and the delta bitmap.
$Suffix(F_i^*)$	Suffix of F_i^* which record the content of delta.
M^*	Preprocessed JRU data, which can be expressed as $M^*=F_1^*:/:F_2^*:/:F_3^*$: ... .
Z	Represent a kind of compression algorithm.
Z^{-1}	The uncompression algorithm of Z.

C. Proposed Algorithm

1) Compression process

The compression process is separated into following steps:

Step 1: Divide the original data M into several pieces of JRU data (F_i), and set serial number for each piece of data based on the original order. M can be expressed as $M=F_1:/:F_2:/:F_3$: ... ...

Step 2: Get the feature (*FID*) for each piece of JRU data.

Step 3: Follow the operations below for each piece of JRU data in ascending order of serial number:

a) If i is the smallest serial number in M, set $N_i=i$, $F_i^*=N_i:F_i$, and repeat the Step 3.

b) If i is the smallest serial number in $[F_i]_R$, set $N_i=i-1$, or else $N_i = arg \max_{F_j \in [F_i]_R, j<i} \sum |S_{ij,m}|$.

c) Set $F_i^*=N_i:|F_i|:Diff(F_i,F_{N_i}):D_{iN_i,1}$:... ..., and repeat the Step 3.

Step 4: Set $M^*=F_1^*:/:F_2^*:/:F_3^*$: ... ..., and compress M^* with a compression algorithm Z.

Note that, as mentioned in **Step 3**, the base-data selected in delta-encoding is not always the optimal one. This is based on the following considerations. First, finding the global optimal base-data will increase the time complexity of the algorithm. Second, because of the characteristic of JRU data, it is easier to find an approximately optimal base-data, which still has a high similarity with the optimal one, near the target data. Finally, without the restriction of the serial number of base-data, two pieces of JRU data might choose each other as the base-data, and it is impossible to extract the origin data in uncompression process.

Figure 5 demonstrates the result of JRU data after being preprocessed. Each line denotes structure of F_i^*. The array in the Figure 5 points from F_i^* to its base-data. For instance, data F_1 and F_3 have the same FID and belong to class A; F_2 and F_4 share the other FID and belong to class B. Based on the compression process, F_3 selects F_1 as the base-data, and F_4 selects F_2 as the base-data. As note that F_1 does not need to do any difference operation, because F_1 has the smallest serial number in all of the JRU data. As result, delta-encoding algorithm have to storage the whole content of F_1. In addition, there is no equivalence element before F_2, so F_2 has to do the difference operation with F_1, even if they are not belong to the same equivalence class.

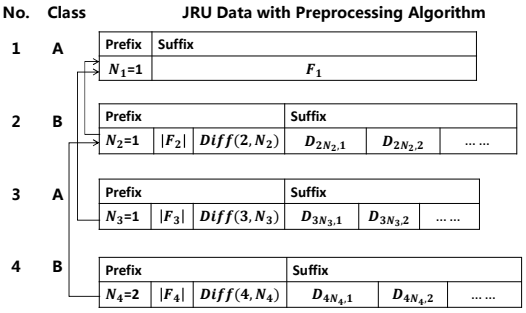


Figure 5. JRU Data with Pre-Processing Algorithm

2) Uncompression process

The uncompression process is divided into following several steps:

Step 1: Extract M^* from the compressed data with the algorithm Z^{-1}.

Step 2: Divide M^* into substring (F_i^*) by the separator /, each substring correspond to a piece of JRU data (F_i). Set serial number for each substring with the same strategy in compression process, we get $\{F_1^*, F_2^*, F_3^*, \ldots\}$.

Step 3: Follow the operations below for each substring in ascending order of serial number:

a) Scan the prefix of F_i^*, and read the message of N_i, $|F_i|$, and $Diff(F_i,F_{N_i})$.

b) If $N_i=i$, set $F_i=Suffix(F_i^*)$, or else, get F_i from F_{N_i} and $Diff(F_i,F_{N_i})$, repeat Step 3.

Step 4: Set $M=F_1:/:F_2:/:F_3$: ... ..., recover the original data from compressed data.

D. Precondition of Algorithm

In this part, we will discuss the precondition of delta-encoding, because it will guarantee the improvement in compression ratio.

First, we will approximately evaluate the compression ratio of common compression algorithm. We assume that the similarity between two pieces of JRU data, which are belong

to the same equivalence class, is $\delta\%$ in average. In addition, we suppose that common algorithms have compression ratio r_S for public strings in original data, and r_D for private strings. Based on the assumption above, the compression ratio r_o for JRU data with common compression algorithms can be approximated as:

$$r_o \approx \delta \cdot r_S + (1-\delta) \cdot r_D$$

In delta-encoding, a common algorithm will compress the preprocessed data which consist of bitmap and the content of private strings. To distinguish between public bytes and private bytes, delta-encoding have to use bitmap which will take up extra space around 12.5% of the length of the original data. Besides, bitmap not only records the position of public and private strings, it also keeps the information of the data structure which is related to original data. Therefore, we could assume that the compression ratio for bitmaps with common algorithms can be seen as same as the compression ratio r_S for public strings. The compression ratio r_{dc} of delta-encoding can be approximated as:

$$r_{dc} \approx 12.5\% \cdot r_S + (1-\delta) \cdot r_D$$

Therefore, as long as the average similarity in the same equivalence class is higher than 12.5%, delta-encoding will have advantage in compression ratio.

E. Optimization and Time Complexity

To optimize the delta-encoding, we will limit ourselves to the consideration of preprocessing algorithm, because the optimization of common compression algorithms is not the focus of this paper. The preprocessing algorithm needs to search a proper base-data for each F_i. Without considering the content of JRU data, the search times will increase and the search process will take more time when serial number of F_i is large, because the algorithm needs to traverse all of the possible equivalence element of F_i. As note that, JRU data records a large amount of static information which will not change much in a short time. Moreover, the serial number is relate to the temporal order of each piece of JRU data. We could assume that, there is higher possibility for two pieces of JRU data to be similar when their serial number are closer. Therefore, we will reach the proper base-data in several search without traversing all of the equivalence elements. Details of choosing proper search times will be discussed in Section V.

The time complexity of delta-encoding is the superposition of preprocessing algorithm and common compression algorithm, because those two algorithms are individual to each other. We assume that there are m pieces of JRU data that need to be compressed, and the size of the whole JRU data is n bytes. The preprocessing part is divided into two processes, classification and encoding. The time complexity of classification process is $O(m)$, because it is easy to get FID from specify position in m pieces of JRU data. In encoding process, because the algorithm will traverse all content for bytewise NOR operation, the time complexity of encoding is $O(n)$. Note that $n \gg m$, the time complexity of delta-encoding is $O(n)$. As result, delta-encoding will cost $O(n)$ more time to improve the common compression algorithm.

V. Experiment and Evaluation

This section discusses the performance of delta-encoding algorithm. The test dataset is provided by Beijing National Railway Research & Design Institute of Signal & Communication Ltd. Based on this dataset, the compression time and compression ratio of delta-encoding will be tested. We choose several state-of-the-art compression algorithms as contrastive algorithm of delta-encoding, include LZMA [6], Deflate [5], and Bzip2 [10]. At the same time, delta-encoding will hybrid these algorithms to figure out if delta-encoding takes advantage in compression.

A. Experiment Design

There are two experiments designed to evaluate delta-encoding in different ways:

Efficiency of Preprocessing: This experiment is designed to evaluate and maximize the efficiency of preprocessing algorithm of delta-encoding. As mentioned in section IV, we do not have to traverse all the elements in the equivalence class. It is important to fix a threshold of search times for each base-data. By comparing the compressed size, we will fix the best search times of preprocessing. That will help us find the proper base-data in the shortest time.

Efficiency of Delta-Encoding: This experiment is designed to evaluate the efficiency of delta-encoding. It will compare the compression ratio and time between some common compression algorithms and their hybrid algorithm with preprocessing algorithm (delta-encoding).

B. Evaluation Metrics

There are two metrics to evaluate the efficiency of compression algorithm in this paper, compression ratio and compression time. Compression ratio is defined as the ratio between the uncompressed size and compressed size [19]. Smaller compression ratio means that the algorithm could reach smaller compressed size with the same original data. Compression time evaluates the speed of a compression algorithm. Smaller compression time denotes that the algorithm is faster in compression. In practical, there is a contradiction between compression ratio and compression time. Usually, higher compression ratio will sacrifice the compression time, and vice versa.

C. Result and Discussion

All experiments were performed on a laptop with an Intel Core i5 CPU 2.4GHz and 8GB RAM running in 64 bit Windows 8. The range of the size of data buffer is from 5 to 2500, which cover the range of both real-time data and delayed data. The result of experiments is shown below.

1) Effciency of Preprocessing

Figure 6 illustrates the relationship between search times and the size of preprocessed data. The horizontal axis indicates the search times, and the vertical axis indicates the compression ratio and time of preprocessing algorithm. As illustrated, the increase of search times will decrease the compression ratio before the 5th search. After that, an inflection point occurs, and the compression ratio remains steady at around

24%. Different from compression ratio, compression time performs reserved trend. Time consumption tends to increase with the search times growing, and remains steady when the search times is more than 30. We believe that this phenomenon is caused by small equivalence class, because the search will be ended in advance when all possible elements have been traversed. Based on the analysis above, the search times of preprocessing algorithm is set as 5. This will have a better result and an acceptable time consumption.

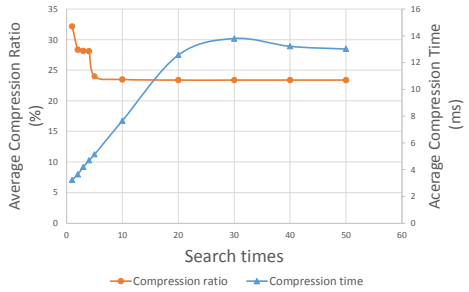


Figure 6. Algorithm Efficiency with Different Search Times

Basing on the search times set above, we also make a comparison between several common algorithms and the preprocessing algorithm, the result is shown in Figure 7 and 8. The horizontal axis indicates the buffer size, and vertical axis indicates the compression ratio and time of preprocessing, only the vertical axis of Figure 8 is non logarithmic.

As illustrated in Figure 7, the preprocessing algorithm is light-weight. Its time consumption is only a half of other algorithms. Delta-encoding has a shorter compression time, however, in Figure 8, its compression ratio is more than two times as much as other algorithms. Fortunately, preprocessing algorithm keeps the structure of original data, it is easy for the algorithm behind to make further compression.

2) Effciency of Delta-Encoding

The result of this experiment is listed in Table III. We analyze the data from two aspects.

First, we compare the efficiency among all of test algorithms. As illustrated, both delta-encoding with Deflate and Bzip2 perform well on compression ratio no matter what the size of data buffer, and the compression ratio is close to or even exceed the LZMA, which is known as one of the most efficiency compression algorithm in the world. Moreover, delta-encoding use less time to approach the similar compression ratio of LZMA. Delta-encoding algorithm save around 99% compression time compared with LZMA.

Next, each common compression algorithm is compared with delta-encoding. We also define the increase rate for both evaluation metrics. The increase rate denotes the improvement of delta-encoding compared with original compression algorithm. Delta-encoding will have a better performance if the increase rate is positive, or else, the original algorithm performs better. In Deflate, the compression ratio have a 20% growth in real-time data, and 36.5% in delayed data. This indicates that delta-encoding decreases the redundancy efficiently. However, delta-encoding benefits the compression ratio at the expense of compression time. No matter what the amount of data, delta-encoding with Deflate takes 60% more time compared to original Deflate. Besides, Bzip2 has some different result from Deflate. In this case, delta-encoding improves both evaluation metrics. Delta-encoding with Bzip2 has similar improvement in compression ratio compared to Deflate. Moreover, there are more growth in compression time, and the maximum increase rate can reach 43.2%. Basing on experiments above, we can confirm that delta-encoding has advantage in the compression of high-speed train data stream.

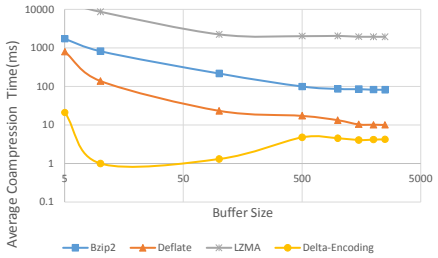


Figure 7. Compression Time of Different Algorithms

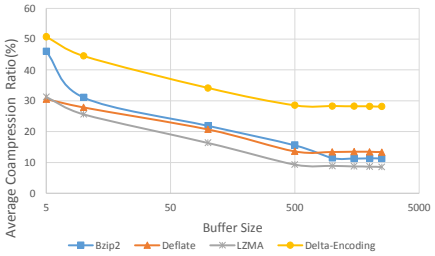


Figure 8. Compression Ratio of Different Algorithms

Table III. Efficiency of different algorithms

	Real-time data				Delayed data			
	Compression Ratio(%)	*Increase Rate(%)*	*Compression Time(ms)*	*Increase Rate(%)*	*Compression Ratio(%)*	*Increase Rate(%)*	*Compression Time(ms)*	*Increase Rate(%)*
Deflate	20.73	+18.6	22.2	-59.9	13.44%	36.5	9.94	-64.5
Delta-Encoding with Deflate	16.87		35.5		8.54%		16.35	
Bzip2	20.57	+7.2	234.6	+15.3	11.6%	33.0	88.81	+43.2
Delta-Encoding with Bzip2	19.1		198.8		7.77%		50.44	
LZMA	16.38	/	4115	/	8.76%	/	2564	/

VI. Conclusion and Future Works

JRU data is one of the main content in high-speed train data stream. By collecting and analyzing the information in JRU data, train scheduling center could optimize the operation of the whole system. In order to reduce the bandwidth of the high-speed train system, JRU data need to be compressed before transmission. Different from regular data format, JRU data has much unnecessary redundant information between two pieces of data. Delta-encoding is an algorithm designed to compress this kind of data. It is a hybrid algorithm with preprocessing algorithm and common compression algorithm. Preprocessing algorithm compresses the public content of JRU data, and retains the main content of the private part. This ensures that the common algorithm behind will still have a good performance on compression. Delta-encoding can reduce the compressed size of high-speed train data by 35%, which is significant for data transmission.

Furthermore, delta-encoding algorithm is also suitable in some scenario of IoT. In the world of IoT, there are number of kinds of format similar to JRU data, which has much unnecessary redundant information. For instance, in a scenario of wireless sensor network, a sensor node need to upload data to gateway node. One piece of transmitted data, which is based on XML, is demonstrated in Figure 9. Identically, this kind of data contains less valuable information (private string), and large amount of public string. In order to reduce the bandwidth, there is no need to transmit the redundant information. Delta-encoding meets this demand, and will have a better performance than common algorithm as long as the data format has the characteristics mentioned above.

```
<?xml version="1.0" encoding="UTF-8"?>
<CommonData>
   <Data Type="sensor1">
      <Value>100</Value>
   </Data>
   <Data Type="sensor2">
      <Value>200</Value>
   </Data>
   <Data Type="sensor3">
      <Value>300</Value>
   </Data>
</CommonData>
```

Figure 9. XML based data format in wireless sensor network

Though delta-encoding has various application scenarios, there are still some problems need to be solve. In the classification process, it is easy for some well-defined data format to locate its feature (such as FID) from specific position. However, in other situations, due to the restriction of the protocol, some classification message could be filtered out. Without detailed description, manual classification in advance will not work well. Therefore, a machine learning algorithm is needed to replace the manual approach. It will capture the feature from the whole content of each piece of data but not from specific position. This simplify the work of manual definition, and it becomes the next step of research.

Acknowledgment

This work was supported by the Beijing National Railway Research & Design Institute of Signal & Communication (CRSC). The authors would like to thank Ling Liu of the CRSC China for providing data and specifications of high-speed train data.

References

[1] Givoni M. Development and Impact of the Modern High‐speed Train: A Review[J]. Transport reviews, 2006, 26(5): 593-611.

[2] Fowler J E, Yagel R. Lossless compression of volume data[C]//Proceedings of the 1994 symposium on Volume visualization. ACM, 1994: 43-50.

[3] Howard P G. The design and analysis of efficient lossless data compression systems[J]. 1993.

[4] Ziv J, Lempel A. A universal algorithm for sequential data compression[J]. IEEE Transactions on information theory, 1977, 23(3): 337-343.

[5] Deutsch L P. DEFLATE compressed data format specification version 1.3[J]. 1996.

[6] Pavlov I. 7-Zip and LZMA SDK,"[J]. 1999.

[7] Benchmark A Q. Gzip vs. Bzip2 vs. LZMA[J].

[8] Burrows M, Wheeler D J. A block-sorting lossless data compression algorithm[J]. 1994.

[9] Manzini G. An analysis of the Burrows—Wheeler transform[J]. Journal of the ACM (JACM), 2001, 48(3): 407-430.

[10] Seward J. bzip2[J]. 1998.

[11] Seward J. bzip2 and libbzip2, version 1.0. 3[J]. A program and library for data compression, 2007.

[12] Psounis K. Class-based delta-encoding: A scalable scheme for caching dynamic web content[C]//Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on. IEEE, 2002: 799-805.

[13] Hu Y, Sundara S, Chorma T, et al. Supporting RFID-based item tracking applications in Oracle DBMS using a bitmap datatype[C]//Proceedings of the 31st international conference on Very large data bases. VLDB Endowment, 2005: 1140-1151.

[14] Barger P, Schön W, Bouali M. A study of railway ERTMS safety with colored Petri nets[C]//The European Safety and Reliability Conference (ESREL'09). Taylor & Francis Group, 2009, 2: 1303--1309.

[15] Wang J, Lin Z. Research on intelligent control strategy used in CTCS-3 train control system[C]//Service Operations, Logistics, and Informatics (SOLI), 2011 IEEE International Conference on. IEEE, 2011: 447-450.

[16] SUBSET U. FFFIS Juridical Recorder-Downloading tool, Version 2.3.0[J]. 27.

[17] Subset U. System Requirement Specification, Version 2.3.0[J]. 26.

[18] Subset U. FFFIS STM Application Layer[J]. 58.

[19] Sullivan G J, Topiwala P N, Luthra A. The H. 264/AVC advanced video coding standard: Overview and introduction to the fidelity range extensions[C]//Optical Science and Technology, the SPIE 49th Annual Meeting. International Society for Optics and Photonics, 2004: 454-474.