Received September 19, 2020, accepted October 2, 2020, date of publication October 8, 2020, date of current version October 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3029705

Autonomous Parameter Adjustment Method for Lossless Data Compression on Adaptive Stream-Based Entropy Coding

SHINICHI YAMAGIWA[1], (Senior Member, IEEE), AND SUZUKAZE KUWABARA[2]
[1]Faculty of Engineering, Information, and Systems, University of Tsukuba, Tsukuba 305-8573, Japan
[2]Department of Computer Science, University of Tsukuba, Tsukuba 305-8573, Japan

Corresponding author: Shinichi Yamagiwa (yamagiwa@cs.tsukuba.ac.jp)

This work was supported in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 20H04152, and in part by the Japan Science and Technology Agency (JST) CREST under Grant JPMJCR1402.

ABSTRACT As speed of communication data path is drastically improved in this decade due to the high data rate, evolutional technology is demanded to address the fast communication implementation. In this paper, we focus on data compression technology to speed up the communication data path. We have proposed a stream-based data compression called ASE coding. It compresses data stream based on the instantaneous data entropy without buffering and stalling for the compression processes. It is also suitable for hardware implementation. However, the stream-based data compression works heuristically with sensitive parameters that affect to the data compression ratio. If the parameters are statically configured, it does not follow the dynamic data entropy, and thus, the data compression performance becomes unstable. In this paper, we will disseminate the parameters, discuss the behaviors of those parameters and propose its autonomous adjustment methods. We will also propose adjustment algorithms for those parameters that follow the data entropy of the input data stream autonomously. Through experimental evaluations applying the algorithms, we will confirm the parameters are adjusted with depending on the data entropy in the data stream. And then, the compression ratio becomes stable as the compressor exploits the minimal entropy adaptively.

INDEX TERMS Lossless data compression, parameter adjustment, entropy coding.

I. INTRODUCTION

Due to the rapid growth of information systems, generation of data is getting drastically promoted. Efficient processing such Big Data, for example, generated from sensors, cameras and human movements needs novel approach to overcome performance fence due to the conventional approach. The advanced researches regarding Big Data processing are focusing mainly on two aspects: One is the speed of data transfer between a system that produce data at Giga or Tera Hz frequency and the one consumes the data. Another is the amount of data to be communicated among the systems. The former needs to address the transfer speed by fast interconnections such as PCI Express and Infiniband network. However, those are now facing physical limit of the frequency. By increasing the number of physical connections, those are still increasing the speed. It is not scalable and makes a system larger and larger. The latter case needs to reduce the amount of data generation. However, it is impossible to reduce it because the resolution of producing data is getting higher and higher, for example, like 4K, 8K and 16K image sizes and also 32bit sensor data from motion devices. Especially, IoT systems now equip sensing functions on the edge side and machine learning algorithms such as DNN (Deep Neural Network) on the cloud side. It inevitably needs to communicate the large data for the accurate inference to process the learning process of the DNN on the cloud side. To resolve such communication overhead, [1], [2] are trying to employ lossy compression on the communication. However, the lossy compression has a tradeoff among the communication speed and the accuracy of inference. Medical applications keep the use of lossy compression at a distance. This problem does not have a good solution except reducing the original data after the generation from the source devices. Therefore, our research focuses on lossless data compression that can reduce the amount of data without racks of the original data. Lossless data compression is able to keep the original data quality. Additionally, it also has potential to accelerate the communication speed by

The associate editor coordinating the review of this manuscript and approving it for publication was Theofanis P. Raptis.

reducing the data amount. However, the recent type of generating data used in a communication is stream-based. For instance, camera application of MIPI standard needs to process a video frame data stream in GHz order. Therefore, it is necessary to compress/decompress such fast data stream in a very small delay.

When we consider the conventional lossless data compression algorithms, Shannon's entropy is the first focus. It derives the probabilities p_i of each unit of data (called *symbol*), calculates the entropy ($-\sum p_i \log_2 p_i$) and assigns bits in the length from the entropy. The total number of bits of the encoded data results less than the original. This mechanism needs to buffer all data and analyses whole data. Therefore, the Shannon's entropy inevitably does not fit to process data stream. The arithmetic coding [3] is the next generation of compression algorithm. It represents the target data by numeric values. It assigns each data pattern to a value in a domain where includes all symbols are presented. This mechanism improves the compression ratio than Shannon's entropy. However, it needs to process all input data and must decide the domain to express all symbols appeared in the input data. This needs to derive the frequent information of all symbols in the input data. Thus, the arithmetic coding does not fit to compress stream data due to the domain calculation. Huffman coding [4] is another lossless data compression. It also has a disadvantage of the buffering problem against processing data stream because it needs to create a binary tree of whole data. To overcome it, the dynamic Huffman coding [5] was proposed. It arranges a binary tree dynamically created during the compression process. However, it is too heavy calculation to process very fast data stream. The second generation of data compression algorithms works based on the look-up table. LZW (Lempel-Ziv-Welch) [6], [7] is the typical and well-known algorithm. The deflate (commonly implemented as ZIP) employs it combining with Huffman coding. However, LZ-based methods, so called, have problem on the table size that is not deterministic during the compression. Therefore, it is not possible to implement it on hardware with fixed resources. Our final goal of stream-based data compression is to develop an effective algorithm that processes a symbol at every clock and also that is completely implementable on a compact and a fixed size hardware.

To speedup the conventional compression methods mentioned above, fast data compression methods are proposed in the aspects of parallelization such as on GPU [8] and hardware-based approach [9]. However, the algorithms and implementations just improve the processing speed without targeting to process data stream in real-time. Therefore, it is not avoidable for such methods to spend multiple cycles to compress/decompress original symbols in stream with a large hardware and also to buffer the original symbols during the operations.

To address the problems above, we have developed a stream-based data compression called Adaptive Stream-based Entropy (ASE) Coding [10], [11]. It truly compresses data stream by calculating instantaneous entropy from the number of occupied entries in the look-up table at the timing when a symbol is received by the compressor. It works faster and provides 10 times smaller implementation on FPGA than the previous version of our stream-based data compression method [12]. However, those stream-based method needs to adjust parameters to achieve the best compression ratio depending on the data types. In both algorithms, to reach the best compression ratio, we need to adjust annoying parameters; the size of the look-up table and the removing frequency of table entries. Especially, ASE coding provides a solution for implementing low latency and high bandwidth data path by compressing data stream. However, it does have unstable data compression performance due to the manual adjustment for the annoying parameters. Potentially, the stream-based compression needs to analyze data redundancy in a short time window. It is not able to analyze the frequency of available data patterns from whole input data. It can compress data in a short time window in real-time. Therefore, the stream-based data compression must exploit the redundancy from the instantaneous data entropy with shifting the window through a data stream. Thus, the compressor/decompressor are not able to decide the best static settings of parameters manually and statically to accept dynamic entropy changes of data stream. In the other words, the parameters should be configured dynamically by following the entropy of data stream.

The problem regarding parameter adjustment discussed above can be pointed out as a drawback also in the conventional data compression. For example, the recent stream-*like* data compression such as LZ4 [13] makes chunks of data and compresses one chunk after another. The size of a data chunk depends on the size of the look-up table. The advanced researches such as [14] report coarse solutions based on processes of OS-level. However, these are not perfect solutions because they are not proposing the adjustment method for the chunk size autonomously.

This paper proposes a novel technique to implement a stream-based data compression method with autonomous configuration for the parameters related to compression ratio. This technique eliminates manual adjustment for the parameters. This paper contributes to the autonomous adjustment algorithms and its implementations of;

- dynamic management of look-up table to maintain effective compression ratio, focusing on an algorithm for removing occupied entries,
- dynamic management of look-up table to avoid excessive removal of the entries,
- and adaptive modification of the look-up table size to obtain high probability by matching input symbols to the registered entries

for stream-based data compression. These contributions are applicable to also the conventional methods that use look-up table.

This paper is organized as follows. The next section describes the backgrounds and the definitions. The section III will propose the techniques and the methods of autonomous configuration for the parameters. In the section IV, we will show evaluations using the proposed techniques and prove the validity. Finally, we will conclude the paper.

II. BACKGROUNDS AND DEFINITIONS

The recent industrial applications demand to process enormous amount of data. The type of data used in the systems is generated continuously such as MIPI camera interface and sensor devices. In this paper, we call this kind of data *streaming data*. It becomes higher resolution and is processed in a short time. We focus on reducing streaming data by employing data compression technique. There are two types of data compression mechanisms: Lossy and Lossless. The lossy compression reduces amount of data by removing redundant part according to signal processing approach such as high frequency part of discrete cosine transform and wavelet filtering. For example, MPEG (Moving Picture Experts Group) format is a typical example to compress video frames by removing high frequency colors that can not be sensed by human eyes. In these days, lossy compression is evolved to apply DNN-based methods. It is used in IoT system to reduce communication between the edge devices and the cloud [15], [16]. However, the lossy compression is not suitable for applications that does not accept losses in the original data such as food industrial image inspection and medical applications. Therefore, these applications inevitably need to employ lossless data compression. We focus on the lossless one in this paper.

A. LOSSLESS DATA COMPRESSION

Technological development of lossless data compression origins on 1950s. The Shannon's information entropy is the original idea of the data compression and is used for a baseline of the compression ratio. When we consider an ASCII text string "AABABCABCD" that is 80bits, the entropy S is calculated by the following equation:

$$S = -\sum p_i \log_2 p_i \tag{1}$$

where p_i is a probability of the i-th symbol in a data pattern. In the case above, the probabilities of 'A','B','C' and 'D' are 0.4, 0.3, 0.2 and 0.1 respectively. Here, S becomes 1.85. The *ceil* of S is the number of bits to present a symbol, which is 2 in the case here. The example data is compressed to '00000100011000011011', that is 20bits. Next, the arithmetic coding [3] extends a pattern to a value in a domain where all data patterns appeared in the target data are mapped. It compresses the patterns to values in the domain. This mechanism results better compression performance than the Shannon's entropy because it can translate a long pattern to a value. The domain is derived from frequency statistics of all data patterns appeared in the input data. For example, A, B, C and D in the patterns here are mapped to 0, 0.25, 0.5, 0.75. Then, the patterns AA, AB, BA, BC and CD are extended as 0.083, 0.166, 0.333, 0.416 and 0.625 by dividing parts of domains defined in the previous level. Thus, finally a value is derived and resulted as the compressed data.

Huffman coding [4] is also an well-known method that encodes the input data to a shorter bit patterns. In the case of example data above, Huffman coding assigns '1', '01', '001', '000' to the symbols from 'A' to 'D' respectively by creating a binary tree and assigning '0' and '1' to the nodes. Thus, the example data pattern is compressed to '1101101001101001000', which is 19bits. On 1970s, *universal* compression algorithms were proposed such as well-known LZ-based methods [6], [7]. It creates a look-up table and the index becomes the compressed symbol. In the example case above, a table is created with entries of "A", "AB", "ABC", "ABCD". Here, the compressed symbol becomes "A0B1C2D" in which the numbers are the indices of the table. This is now becoming a major compression mechanism combining with Huffman coding such as LZSS (Lempel-Ziv-Storer-Szymanski), deflate [17], snappy [18], LZO (Lempel-Ziv-Oberhumer). LZ-based compression algorithms are eagerly investigated and many solutions were proposed to accelerate the speed such as parallelization on CPUs [19], [20] and GPUs [8], [21].

On the other hand, when we consider to compress stream data, we need an algorithm that never buffers any part of data and also never blocks the compression process. In the decompression side, we need the same characteristics in the algorithm. However, the conventional algorithms such as Shannon's entropy, the arithmetic coding and Huffman coding mentioned above need to check whole data to calculate frequency of symbols. This inevitably must block the compression processes by buffering whole data in memory to derive frequency available data patterns. This avoids the compressor to process stream data. On the other hand, LZ-based algorithms need a table memory which size is not deterministic because the compressor is not able to know frequency of data patterns before it receives the patterns. This eliminates to implement in hardware with a concrete architecture because we are not able to decide the memory size maximally used during compression processes. As mentioned above, the conventional algorithms assume that compressor/decompressor can use infinite processing time and also infinite hardware resources. This does not need to consider parameters in the algorithms. However, when the resource size is limited, we need to setup parameters such as the size limitation of the look-up table and the number of contiguous symbols in LZ-based methods. Moreover, there do not exist any solutions to optimize those parameters.

B. STREAM-BASED LOSSLESS DATA COMPRESSION

To address drawbacks in the conventional lossless data compression algorithms, we have developed a new data compression mechanism for streaming data. The main concepts of the compression mechanism are; 1) to compress streaming data without buffering and stalling, 2) to have architecture for easy hardware implementation and 3) to decompress without buffering as soon as receiving the compressed data. Our first trial is LCA-SLT (Lowest Common Ancestor-Static Lookup Table) [22]. It is based on a look-up table that statically

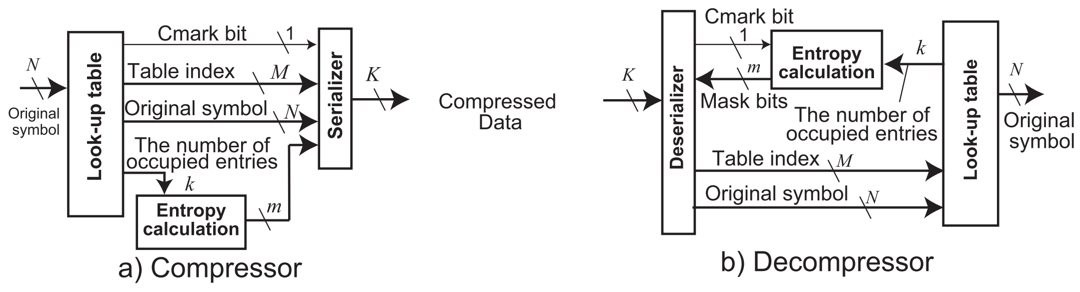

FIGURE 1. System organization of ASE coding.

maintains frequent symbol pairs. The symbol pair in the table is converted to an index of the table. The pairs are selected from a sample data exploited from the target data to be compressed. This mechanism works on a hardware and the resource size becomes very small. However, it is not flexible against change of data entropy. Thus, we confirmed that an adaptive table management is indispensable.

We have added a dynamic table management to the stream-based data compression and developed LCA-DLT (Lowest Common Ancestor-Dynamic Lookup Table) [12], [23]. It receives symbol pairs in data stream and compresses to an index of the look-up table as well as LCA-SLT. The table management employs an adaptive mechanism to replace table entries based on frequency of the input data stream. The table management has a usage counter per table entry. The counter is incremented when a symbol pair is matched to its entry. In addition, a remove pointer is rounding around the entries in the table and if the counter becomes zero, the entry is purged. This allows to dynamically replace table entries depending on data entropy of data stream. We have proposed an effective hardware implementation of this mechanism [12], [24]. However, we still had a problem that the symbol is not compressed to smaller than the index width of the table. Moreover, the number of entries (i.e. the number of bits of the table index) must be fixed during compression/decompression process. Therefore, when the symbol size is typically 8bit, the index is 8bit wide. Thus, the pair can be reduced to only 50%. It is the bottleneck to contribute to the compression ratio. The compressed data should consist of a variable number of bits.

C. ADAPTIVE STREAM-BASED ENTROPY CODING

To overcome the potential drawback of LCA-DLT that the compressed symbol is fixed size, we have developed a new stream-based data compression mechanism called ASE Coding [10], [11]. It is elegant data compression dedicated for data stream using a look-up table by compressing it according to an entropy calculation. The entropy calculation returns the shortest code bits from the number of occupied entries in the table. The table maintains frequent symbols. The table management includes effective removing method of occupied entries called *entropy culling*. The entropy culling contributes to reduce the number of bits from the entropy calculation as the number of matching to the look-up table increases.

Figure 1 shows the system organization of ASE coding. The compressor consists of a look-up table, an entropy calculation and a serializer. When a symbol in a data stream is received by the compressor, it is compared to the entries in the look-up table. When it matches to an entry in the table, its index is selected as the compressed symbol. The index is shrunk to m bits by the result from the entropy calculation:

$$m = \log_2 e \tag{2}$$

where e is the number of occupied entries of the look-up table. Finally, the lowest m bits of the table index is selected as the compressed symbol. Because this is variable length, the serializer reforms a stream of the compressed symbols to a width of an interface to communicate with its decompressor. Here, again, let us explain this in formal. A symbol s which width N is converted to the index I of the look-up table which width is $\log_2 E$ bits where E is the total number of entries in the table. I is shrunk to m bits according to the equation (2). A *Cmark* bit $(=1)$ is added to the MSB of the m bits. On the other hand, when the symbol s misses any entry in the table, the symbol is registered to the table. A Cmark bit $(=0)$ is added to the MSB of the symbol and finally the $N+1$ bits are passed to the serializer. The serializer aligns the stream that includes both $m+1$ bit compressed and $N+1$ bit original symbols to K bit wide and outputs the stream to the decompressor side from the MSB of the symbols.

The decompressor is organized with a deserializer, a look-up table and an entropy calculation. The parameters K, E and N are the same as the compressor's. When the K bit wide compressed stream is received by the deserializer, the first bit is exploited. It is the Cmark bit. When the Cmark bit is 1, m bits calculated by the equation (2) are exploited from the subsequent output of the deserializer. The bits are extended to $\log_2 E$ bits as '0's are added to the MSB. It is used as an index of the look-up table and a symbol is associated from the table. The symbol is the decompressed symbol s. When the Cmark bit is 0, N bits are exploited from the deserializer and treated as the original symbol s. It is registered to the table and outputted from the decompressor.

Here, the look-up table management is a key feature in ASE coding. The registration of a symbol to the table consists of typical operations. When the compressor misses the search in the table or when the decompressor receives a compressed symbol with the Cmark $= 0$, it pushes the

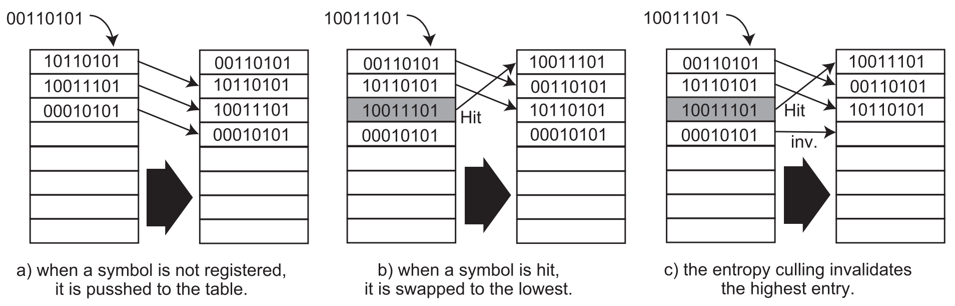

FIGURE 2. Look-up table operaitions on ASE coding.

symbol to the table from the top like LRU (Least Recently Used) manner as depicted in Figure. 2a) and b). When the table becomes full, the registration operation pushes out the entry in the bottom of the table. This allows that the table is implemented by a fixed size E. Additionally, this mechanism contributes that the table index can be shrunk from $\log_2 E$ bits to m bits because all registered entries are always placed in the upper part of the table. However, if we employ only this table operation, the table becomes always full as the compression/decompression operations are repeated. To avoid this situation, we introduce a removing mechanism from occupied entries in the table, called *entropy culling* as illustrated in Figure. 2c). The entropy culling invalidates an occupied entry placed in the bottom after the replacement of the matched entry is invoked. This mechanism is performed independently in both compression/decompression sides. We have a concrete reason why the entropy culling activates the removal of occupied entry at the matching timing because the shorter code bits should be assigned as a compressed symbol according to m derived from the entropy calculation as long as the table hits are repeated. Here, we need to decide a parameter d that is the number of matchings to the look-up table until the entropy culling is activated. The parameter d is defined as a velocity to remove occupied entries. The smaller d accelerates to assign shorter code bits. According to the table operations above, the number of entries in the table is reduced by the entropy culling. And then, the number of bits of a compressed symbol is reduced because the m resulted by the entropy calculation can become less than $\log_2 E$. Thus, ASE coding can maintain adaptive compression ratio depending on entropy of data stream.

The algorithm of ASE coding is suitable for hardware implementation. We have implemented it on an FPGA at 200MHz clock frequency reported in [10]. The resource size is almost 1 of 10th against the one of LCA-DLT. Therefore, ASE coding provides a truly stream-based variable compression method that can perfectly be implemented on hardware. And also it can achieve very high bandwidth.

However, to acquire the best data compression ratio (i.e. compressed data size divided by the original one), we need to decide manually the parameters that dominate to the compression mechanism. ASE coding has two unstable parameters: One is the velocity d of the entropy culling and another is the number E of look-up table entries. The first parameter affects not only to the number of bits of compressed symbol decided by the result from the entropy calculation but also to the hit ratio of a symbol searched through the table. If the entropy culling removes entries too much (i.e. d is too small), it degrades the hit ratio and then brings worse compression ratio. Besides, if it removes few (i.e. d is too large), the table becomes always full. This provides the longer code bits to a compressed symbol. Another parameter E affects also to the number of bits in the compressed data because the compressed symbol is decided by the result of entropy calculation. The smaller the number of the entries is, the smaller the number of bits of compressed symbol is generated. Thus, the two parameters above must be adjusted to fit to the input data stream before compressor receives the stream. But it is almost impossible and there is no concrete strategy to decide those. Therefore, an autonomous mechanism to configure such parameters adaptively should be added to the ASE coding.

There exist advanced researches based on hardware lossless data compression mechanism such as *nx842* [9]. It is implemented on POWER processor as a data compression accelerator of the processor functions. Applications utilize the function for various accessing data on the memory bus controlled by the instructions of the processor. The compressor also uses the algorithm based on the look-up table and supports data stream from the data path of the processor. It outputs a template for compressed data that matches to a combination of compressed and original symbols. The larger size of look-up table is allocated, the better compression performance the nx842 can achieve. Therefore, the adjustable parameter is just the number of entries of the look-up table. It is easy to resolve the parameter setting because it depends on the available hardware resources targeted for the implementation. However, the larger number of table entries increases required hardware resources and eliminates compact implementation.

D. DISCUSSION

As we have discussed above, data stream is not acceptable by the conventional lossless data compression algorithms. Those compress/decompress the data by multiple cycles with buffering the chunk or whole of the data. To overcome such drawbacks, we have proposed a stream-based data compression with the adaptive entropy coding mechanism called ASE coding. However, we need to decide indispensable two parameters before the compression processes to achieve the best data compression ratio. But there is no concrete method to decide a promised parameter set before a data stream, which can be continuously generated from equipment, is inputted to the compressor. To address this problem, we need to develop an autonomous method to configure the parameters. Thus, in this paper, we will propose a novel entropy coding algorithm for data stream that automatically and adaptively adjusts the key parameters related to the compression ratio.

III. AUTONOMOUS PARAMETER ADJUSTMENT METHOD FOR STREAM-BASED DATA COMPRESSION

We propose methods to dynamically and adaptively optimize the parameters discussed above in stream-based data compression. We will focus on ASE coding. The optimization techniques must work autonomously while the input data symbols in a data stream are compressed one after another. Here, let us begin to focus on the behaviors of the parameters on ASE coding.

A. METHODS FOR AUTONOMOUS PARAMETER ADJUSTMENT

The first parameter is related to the removing mechanism of occupied entries in the look-up table. It is the entropy culling mechanism. The entropy culling can reduce the number of bits for a compressed symbol due to the entropy calculation associated from the number of occupied entries. However, if the entropy culling works at highly frequent, the number of occupied entries in the look-up table decreases too much. This degrades the matching probability in the opposite. To avoid this situation, we introduce a new mechanism called *entropy culling limit*.

The entropy culling limit defines two areas in the look-up table by a limit line as depicted in Figure 3. The protected area inhibits the entropy culling. Besides, the entropy culling can remove occupied entries in the free area. The entropy culling removes the entry from the higher index of the table to the lower one and stops removing occupied entry when it reaches to the line. This mechanism reserves protected entries in the top of the look-up table from the culling operation. This affects to the matching probability and finally controls compression ratio. However, because it is impossible to decide the best position of the entropy culling limit we need to add another parameter L that is the border index in the look-up

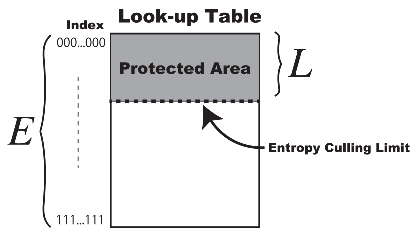

FIGURE 3. Entropy culling limit.

table between the protected area and the free one. In this paper, we also try to automate adaptive decision of L.

The last parameter is related to the number of entries in the look-up table. The entropy calculation in ASE coding uses the number of occupied entries when it matches to the look-up table. Therefore, we need to find a method to keep smaller number of occupied entries E in the table. However, if the number of table entries is too small, the compressor matches few original symbols to any entry in the table. Here, we introduce a new method, called *dynamic look-up table*, to adjust the number of look-up table entries E adaptively.

The entropy culling limit protects occupied entries obtained in the lower indices of the look-up table. However, the maximal number of protected entries is limited to the table size E. Therefore, the E should be increased when the data entropy is high because there exists a possibility that the subsequent input symbols should be maintained in the table for the near future matchings. This is the same concept as the entropy culling limit. Therefore, the dynamic look-up table contributes to support the effect of the entropy culling limit. Thus, when the number of mishits in the look-up table increases, it increases the number of table entries E. On the other hand, when the matchings in the table repeat, the number of table entries E should be decreased because the number of bits of a compressed symbol should be shrunk to a shorter code bits while frequent patterns are continuously coming to the compressor. This is the same characteristic as the entropy culling. Here, when the velocity of its removing operation is too slow to follow the frequency, the decreasing E accelerates the removing operation. Thus, the dynamic look-up table mechanism helps the effects from the entropy culling and the entropy culling limit.

According to the discussion above, we will propose autonomous adjustment techniques for the three parameters d, L and E respectively in the following sections. In this paper, we also consider hardware implementation of the adjustment algorithms. Therefore, the algorithms have to be implemented as simple as possible.

B. AUTONOMOUS ADJUSTMENT FOR ENTROPY CULLING

The entropy culling is performed after d hits in the look-up table. Here, our final goal is to design an algorithm that autonomously adjust d by following entropy of a data stream. Before explaining the adjustment algorithm, let us discuss the strategy to optimize d dynamically during the compression.

During the compression operations, d works as the velocity to remove the occupied entries in the look-up table. When the number is large, the removing speed is low, vice versa. As we have discussed in the previous section, the occupied entries should be removed fast because the number of bits in the compressed symbol can be shrunk as short as possible. This means that decreasing d during repeated hits to the table works effectively to acquire the low entropy of the input data stream. On the other hand, when the number of hits to the table is few, the data entropy is high. In the other words, the data stream consists of low frequent data. In this situation, the table should save much symbols as possible for the future hits. This cause ASE coding to generate compressed symbols with almost M bits where $M = \log_2 E$. However, it will become an advantage when the repeated hits begin again. Thus, we can control the entropy culling by decreasing d at the hit operation in the table or by increasing d at the mishit one.

Algorithm 1 Autonomous adjustment of entropy culling

```
a) Initialization:
$d\_shift\_bits \leftarrow \texttt{D\_MIN\_BITS}$
$d \leftarrow \texttt{INITIAL\_CULLING\_VAL}$

b) When a symbol is missed in the look-up table:
$d \leftarrow d + 1$
if $d > (1 \ll d\_shift\_bits)$ then
    if $d\_shift\_bits < \texttt{D\_MAX\_BITS}$ then
        $d\_shift\_bits \leftarrow d\_shift\_bits + 1$
    else
        $d \leftarrow 1 \ll d\_shift\_bits$
    end if
end if

c) When a symbol is matched in the look-up table:
$d \leftarrow d - 1$
if $d == 0$ then
    Remove an occupied entry in the highest index (when the index is lower than the entropy culling limit).
    if $d\_shift\_bits > \texttt{D\_MIN\_BITS}$ then
        $d\_shift\_bits \leftarrow d\_shift\_bits - 1$
    end if
    $d \leftarrow 1 \ll d\_shift\_bits$
end if
```

According to the discussion above, Algorithm 1 demonstrates an implementation of the adjustment control for the velocity d of the entropy culling autonomously while an input symbol is searched in the table. In the algorithm, we use a shift function depicted by $\ll$ to increase d based on $2^{d\_shift\_bits}$. At the reset timing of ASE coding, the a) initialization is executed. The `D_MIN_BITS` is the initial number of shifts to be applied to d. The `INITIAL_CULLING_VAL` is any number for the initial value of d. This can be d_shift_bits (i.e. `D_MIN_BITS`). After ASE coding starts, as the compressor registers every symbol to the look-up table, the code of b) or the one of c) is executed. When a symbol is missed in the look-up table as demonstrated in b), by aligned to $2^{d\_shift\_bits}$, the d is incremented up to $2^{\texttt{D\_MAX\_BITS}}$. Here, the d_shift_bits is incremented when the d becomes more than the current $(1 \ll d\_shift\_bits)$. This makes the operation of removing an occupied entry slow. On the other hand, when a symbol hits to the table, the code of c) is executed. As following the rule of the entropy culling operation, the d is decremented at every table hit. When d equals zero, an entry in the look-up table at the highest index is removed. Here, the d_shift_bits is also decremented. This means that repeating hits in the table accelerates the removing speed. Finally, the d is set by the decreased d_shift_bits. Due to this algorithm, the velocities for removing and accumulating entries in the look-up table are balanced.

We have another method to adjust the velocities of the removing and the accumulating the look-up table entries using a hit ratio in the table. This mechanism accumulates the number of hits in the table in a T symbols. If the number of hits is more than $T/2$, for example, it increments d_shift_bits vice versa. However, because we need to define a constant number T, this needs to increase the number of parameters again. Therefore, Algorithm 1 uses every timing when a symbol is registered to the table. According to the algorithm, we do not need to define any interval of a data stream. Additionally, the velocity control will follow entropy of the data stream instantaneously.

The autonomous adjustment of d proposed in this section removes the occupied entry as fast as the entropy becomes low. However, when all table entries are removed, the compression does not work efficiently because it iterates the registrations and the removals of symbols to the table. To avoid this situation, we apply the *entropy culling limit*. Let us discuss the autonomous adjustment for the mechanism in the next section.

C. AUTONOMOUS ADJUSTMENT FOR ENTROPY CULLING LIMIT

The entropy culling limit works as a protection from excessive removal of occupied entries in the look-up table. It avoids the removal operations against the entropy culling. The protected area defined by L, which is the highest index of the area, should be extended or shrunk by depending on entropy of the data stream to the compressor. The fundamental strategy of modifying L is the same as d. When high frequent symbols in a data stream are coming to the compressor (i.e. the data entropy is low), the L should be decreased. On the other hand, when low ones are coming (i.e. the data entropy is high), it should be increased. The merit of the entropy culling limit is to protect a minimal number of occupied entries saved in the lower indices of the look-up table. This eliminates an effect that a compressed symbol and an original one are alternately outputted from the compressor by repeating a hit and a mishit in the look-up table due to the frequent removal of the entropy culling. However, it is very hard to define the minimal number of L statically because the number can be defined due to the data entropy. Therefore, we propose an algorithm that autonomously adjusts L following the data entropy from a data stream.

Algorithm 2 shows an implementation of our strategy for L in the entropy culling limit. Initially before the compressor starts, a) initialization is executed to reset L to the initial constant value `INITIAL_CULLING_LIMIT`. The value is any number less than E. In the algorithm, we use a shift left operation for increasing/decreasing L. This means that L is always between `MIN_CULLING_LIMIT` and E, where the `MIN_CULLING_LIMIT` is a constant

Algorithm 2 Autonomous adjustment of entropy culling limit L

```
a) Initialization:
$L \leftarrow$ INITIAL_CULLING_LIMIT

b) When a symbol is missed in the look-up table:
if $(L \ll 1) \leq E$ then
    $L \leftarrow L \ll 1$
else
    $L \leftarrow E$
end if

c) When a symbol is matched in the look-up table:
if $(L \gg 1) \geq$ MIN_CULLING_LIMIT then
    $L \leftarrow L \gg 1$
else
    $L \leftarrow$ MIN_CULLING_LIMIT
end if
```

fence value for limiting the entropy culling limit not to decrease the number of occupied entries in the look-up table than the value. The MIN_CULLING_LIMIT must be any value from 1 to E. Note that MIN_CULLING_LIMIT $\leq$ INITIAL_CULLING_LIMIT. However, we can not set zero to MIN_CULLING_LIMIT because repeating hits in the look-up table induces that L becomes zero. After that, the entropy culling limit does not work at all according to the Algorithm 2c). The typical MIN_CULLING_LIMIT is 1, which is the minimal valid L. The L is modified right after the look-up table operation. The code b) is executed when a symbol is missed in the table. It extends the protected area for the entropy culling limit up to E. On the other hand, the code c) is executed when a symbol is hit in the look-up table. It decreases L down to MIN_CULLING_LIMIT by shifting one bit to right. We avoid to use multiplication/division or summation/subtraction due to decreasing calculation overhead. When we perform the algorithm on hardware, it can be implemented by smaller number of resources than other arithmetic calculations such as adding some offset or multiplying some factor to L. Additionally, the offset or the factor will become another parameter for the compression performance. The algorithm is executed in both the compressor and the decompressor at the same timings right after the table operations has been finished. This makes the contents of the look-up tables in both sides equivalent. Thus, the compressed data from the compressor will be decoded to the original data in the decompressor.

As we have seen in this section, while the entropy culling limit works with the interaction of d, the proposed algorithm protects the excessive removal of the occupied entries by the entropy culling operation. This mechanism will follow autonomously data entropy of a data stream inputted to the compressor.

D. AUTONOMOUS ADJUSTMENT BY DYNAMIC LOOK-UP TABLE

The last parameter is the number of entries E in the look-up table. The E is related to m from the entropy calculation. Therefore, if the E is modified appropriately and dynamically, the number of bits in the compressed symbol can be reduced, and thus, the compression ratio can become better with following data entropy of the input data stream. The number of the table entries E should be shrunk when the data entropy is low. On the other hand, the E should be extended when the data entropy is high. We called this mechanism the dynamic look-up table. The table size should not be changed often because the entropy culling does not work well when the table is shrunk fast. Therefore, we introduce a counter to moderate changing E. The counter is incremented at every hit in the look-up table. If the hit counter is more than a threshold value after a number of symbols, the E is decreased. Otherwise, E is incremented.

Here, we need to resolve two more parameters: the number of symbols for the statistical calculation and the threshold value to decide if E is increased or not. First, let us discuss the number of symbols to evaluate the hit counter. It is not easy to decide the value intuitively. Therefore, we use the E for the number of symbols. Accumulating the results if symbols are hit or mishit in the look-up table with counting the number of symbols, after the number becomes E, the compressor/decompressor modifies E. In this case, E is decided with depending on the data entropy of data stream. This means that E is an adaptive number that follows the data entropy. Second, let us consider the threshold value for increasing/decreasing the table entries. This can be decided based on the table size. We use the half number of symbols for evaluating the table size (i.e. $E/2$) as discussed above. This can be easily implemented by shifting one bit of E to right. The threshold value can be sensitive to the compression ratio. However, this method is suitable to decide the best value for it dynamically because it follows data entropy of a data stream adaptively. Therefore, we use the half number of symbols for evaluating the table size.

Algorithm 3 demonstrates an implementation of the dynamic look-up table. The codes are executed on both the compressor and the decompressor and the same table sizes are kept and the equivalent table operations are performed between those with respect to the corresponding symbol. In the algorithm, we employ two valuables *hit_counter* and *try_counter*. The former is used for counting the number of hits in the table. The latter is used as a counter for the one of symbols until evaluating the table size E. At the initialization shown in a), the E is reset to a constant value INITIAL_TABLE_SIZE. The value is the maximal table size physically allocated. After a symbol registration is performed, the code b) is executed. If the symbol is hit in the table, the *hit_counter* is incremented. The *try_counter* is always decremented. When the *try_counter* equals zero, the code changes the table size with making E half or double by shifting 1 bit to right or left. And then, the *hit_counter* and

Algorithm 3 Autonomous adjustment of dynamic look-up table E

```
a) Initialization:
$E \leftarrow$ INITIAL_TABLE_SIZE
$hit\_counter \leftarrow 0$
$try\_counter \leftarrow E$

b) when a symbol registration is performed:
if a symbol is missed in the look-up table then
    $try\_counter \leftarrow try\_counter - 1$
else
    $hit\_counter \leftarrow hit\_counter + 1$
    $try\_counter \leftarrow try\_counter - 1$
end if
if $try\_counter == 0$ then
    if $hit\_counter < (E \gg 1)$ then
        if $(E \ll 1) \leq$ INITIAL_TABLE_SIZE then
            $E \leftarrow (E \ll 1)$
        else
            $E \leftarrow$ INITIAL_TABLE_SIZE
        end if
    else if $hit\_counter > (E \gg 1)$ then
        if $(E \gg 1) \geq$ MIN_TABLE_SIZE then
            $E \leftarrow (E \gg 1)$
            if $E < L$ then
                $L \leftarrow E$
            end if
        else
            $E \leftarrow$ MIN_TABLE_SIZE
        end if
    end if
    $hit\_counter \leftarrow 0$
    $try\_counter \leftarrow E$
end if
```

the *try_counter* are reset to zero and the newly modified E. Here, we need to care also the entropy culling limit because the E can be less than L. When the table size E is less than the entropy culling limit L, the code modifies the L to the E because the L must be included in the index domain of the table. The MIN_TABLE_SIZE controls the minimal table size. This can be any constant value more than zero.

As explained above, the dynamic look-up table will work dynamically to optimize the code bits of a compressed symbol by changing E that related to the m from the entropy calculation. The proposed algorithm will also work with the entropy culling limit. Thus, the dynamic look-up table will contribute to the compression ratio with following data entropy of the input data stream.

E. TIMING AND ORDER FOR APPLYING AUTONOMOUS ADJUSTMENTS

Now, let us discuss the timing and the order to evaluate the parameters. The timing when we can change the parameters exists at the right after the compression/decompression operations are finished. The modifications of d, L and E will be performed after a compressed/original symbol is outputted from the compressor/decompressor. These three parameters can be changed in any order. However, note that the entropy culling limit L is dominant to the table size E. It must follow the relationships $L \leq E$. The compression ratios will become different when L is modified after changing E vice versa because L is shrunk to E if $E < L$. For example, assume that a case when E is changed from 128 to 64 and when L was 128. Applying the autonomous parameter adjustments mentioned above, if L is modified first, L will be modified to 64. However, if E is modified first, L will be changed to 32 after E is modified to 64. In the latter case (i.e. changing L after E), L is changed twice. This invalidates the effect of the entropy culling limit. Therefore, we have to follow the order of modifications for L and E in this order. Regarding the order of d and L, we do not need to care it because those are diagonal parameters with respect to the look-up table size. Therefore, we will apply the evaluation of d first, then will do the one of E. According to the discussion above, in our implementation, we invoke the evaluation of the parameters d, L and E in the order.

IV. EXPERIMENTAL EVALUATION

A. EXPERIMENTAL SETUP

This section will show evaluations for the algorithms proposed above focusing on the compression ratios. We will perform three evaluations: evaluation for the entropy culling regarding d, the one of the entropy culling limit regarding L and the one of the dynamic look-up table for E. During the evaluations, we use benchmark data available from [25]–[27] and [28]. From the benchmarks, we use two ASCII text data and two image data. The text data potentially includes some rules based on ASCII code. This provides frequency that makes the entropy low. On the other hand, the image data is randomly generated depending on the image sensor or the graphics rendering algorithm. This provides high entropy than the text data. Here, we will use single frame data from the video files available in the sites because it is easy to compare data which entropies are statically known.

The gene DNA sequences and the English text are picked up from [25]. The contents of the files are organized with ASCII text data sequences. We use the first 10Mbyte of the downloadable file from the site. The *Beauty* is picked up from the second website. It is provided by a file with a video frame sequence in YUV420 format in 8bit depth. Note that in the YUV420 format, Y:U:V is 4:1:1. It stores 8bit Y element for every pixel. It also stores U and V elements that are also 8bits respectively derived from 2×2 pixel block. We use the 100th frame of the sequence in the video file. The size of the frame data is 12Mbyte. The *Sintel* is picked up from [28]. It is a video file of a computer graphics amination which frame size is 10MByte. We use the 1000th frame of the Sintel formatted in YUV420 of 8bit depth by converting from the TIFF file. Here, let us explain the reason why we have chosen these data sequences as the benchmark.

The gene DNA sequences consist of only four characters (A, T, G, C) of eight bit wide. This case is a good example that we can image an effective configuration of static parameters. For instance, the d can be four due to the number of patterns appeared in the data sequence. The L can be also four to protect the four patterns. Finally, the E can be also four because it is the maximal number of patterns appeared in the sequence. Using the sequence, we can see how effective the algorithms for the autonomous adjustment works with comparing the static settings. The English text is an example of a more complex data patterns of alphabets and some other symbols with eight bit wide. The evaluation using this will show a compression performance of a data sequence with higher entropy than the gene DNA sequence. The Beauty is an example of a natural image data which resolution is 3840×2160 based on YUV420 format. The evaluation with this data sequence will show effects of the algorithms in the case of high entropy data stream. On the other hand, the Sintel is an example of a data sequence based on an artificial image which resolution is 4096×1744 formatted in YUV420. The data size is 10MB. This includes frequent color patterns due to the creation algorithm of the computer graphics. Therefore, the evaluation using this image data will show effects when we apply the algorithms to such data sequence with lower data entropy than the natural image.

As we have discussed in the section III-E, we will apply the algorithms 1, 2 and 3 in the order. Additionally, we will use the default settings of $E = 16$ for the ASE coding in the case without autonomous adjustment for the dynamic look-up table. To compare performance impacts among the different input symbol widths, we apply 8bit and 16bit symbol widths to the settings for the evaluations. The other parameters are varied as depending on the evaluation. During the evaluations from the next section, we will use the compression ratio calculated by (compressed data size)/(original data size)$\times$100. When we compare the compression ratios, if a ratio is smaller than the others, the compression ratio is the best among the ratios. This means the method of the best ratio compresses data to the minimal size among those comparisons.

Table 1 summarizes the entropies of the benchmark data used in the following evaluations calculated from the equation 1. The table shows two entropies when the symbol sizes are 8bit and 16bit respectively. As we have discussed above, the complexities of data are proved numerically according to the comparisons in the table. In addition, as reference performances derived from ZIP (deflate) compression, Table 1 also shows the compression ratios. The performances are better than the ones of ASE coding because the stream-based compression can only refer frequency in a limited time window performed by the look-up table. On the other hand, ZIP tries to compress with scanning whole data as much as possible applying Huffman coding also. Therefore, the performances of ZIP will become better than ASE coding. However, some will be near to the performances of ASE coding. Now, let us evaluate the effects of the algorithms from the next section.

TABLE 1. Entropy and reference compression ratio of benchmark data sets.

Name of data	8bit entropy	16bit entropy	ZIP comp. ratio (%)
Gene DNA Sequences	1.99	3.92	28.26
English text	4.55	8.15	37.92
Beauty	6.95	11.10	70.6
Sintel	6.70	7.53	14.66

B. EFFECT OF AUTONOMOUS ADJUSTMENT OF ENTROPY CULLING

The first evaluation focuses on the autonomous adjustment of the entropy culling by comparing the cases with static and the autonomous settings of the parameter d. Figure 4 shows the evaluation when the d is varied from 1 to 16 and the case when the autonomous algorithm is applied. The parameter for the entropy culling limit L is also varied from 1 to 16 statically. Note that the E is fixed to 16 during this evaluation. The graph shows both cases of 8bit and 16bit symbol widths.

According to the graphs, each benchmark seems to have an optimal setting of the d and the L statically. When the d is increased from a small number to a large one, we can find the best setting. For example, there are the best setting when we change d with fixing L. A typical example is the result when $d = 16$ during $L = 1$ and 8bit symbol of gene DNA sequences. In the same manner, we can find the best L. For instance, all the cases of $L = 4$ in gene DNA sequences result the best performances when the symbol size is 8bit. However, the problem here is that we can not decide the best setting for d and L statically for the compressor because there is no guarantee that data entropy of the data stream does not constantly continues with respect to the static configuration.

Let us discuss the detailed compression performances of each benchmark. We can confirm that almost all cases when the symbol width is 16bit show bad compression ratios because $E = 16$ is too small for the setting. The large symbol size causes high probability of data patterns due to the large number of bits. This affects to high entropy data such as Beauty and English text. Therefore, we can consider the symbol size also as a parameter to be adjusted autonomously. However, if we change it dynamically, we need to change N in the compressor and the decompressor at every modification of the symbol width. This means that the organization of the look-up table (i.e. the width of an entry) and the data paths must be reconfigured dynamically. Unfortunately, it is inevitably impossible for the current ASE coding to transform the organization dynamically. In the case of 8 bit symbol, the performance is less than 100%. This means that ASE coding compresses the data stream effectively when the symbol size is small.

Let us focus on the effect of d. In the case of a) gene DNA sequences, we can expect the best performance when the d and the L are configured to both 16. The compression performance when the symbol width is 16 bits shows better than the one of 8 bits when $L = 16$. In this case, almost all ASCII data patterns appeared in the data sequence are

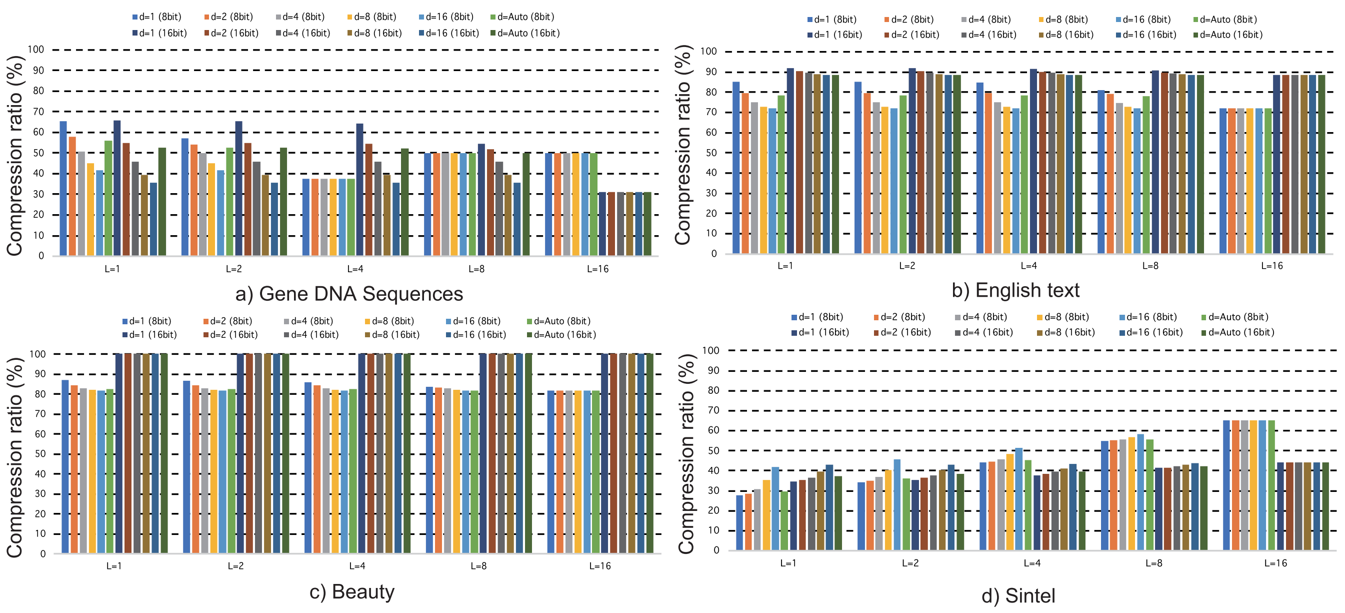

FIGURE 4. **Evaluation of autonomous adjustment of entropy culling.**

saved in the look-up table. The compression ratio becomes the best by increasing the number of hits in the table due to the protected occupied entry by the entropy culling limit. Here, we focus on the effect when the d is too large or too small. Those cases make the compression ratios worse as shown in the cases of b) English text with 8bit symbol width and d) Sintel. The case of English text shows the effect that the small d degrades the compression performance. Because such small d accelerates to invalidate occupied entries, mishits increase and thus, the compression ratio becomes worse. In the case of Sintel, on the other hand, the number of bits in compressed symbols becomes larger because the large d makes the speed to invalidate occupied entry slow. These facts mean that the static settings do not follow the dynamic data entropy of the input data stream. The performances of c) Beauty show the same tendency as the one of b) English text. On the other hand, the case of d) Sintel shows that the compression ratio becomes worse when we increase d at a L. Comparing to the performances of the static settings of d as observed above, the autonomous adjustment of d results the middle performance between the best and the worst cases even if the L is changed. Therefore, we confirmed that the algorithm 1 works to follow the dynamic data entropy of the data sequence. And thus, it results the adaptive performance effectively.

C. EFFECT OF AUTONOMOUS ADJUSTMENT OF ENTROPY CULLING LIMIT

Next, we evaluate the autonomous adjustment of the entropy culling limit as proposed in the algorithm 2 regarding L. In this evaluation, we preform experiments by using the ASE coding with the algorithm 1. The remaining two parameters L and E are statically varied from 1 to 16. We compare those performances with the one with the autonomous adjustment of the L. Figure 5 shows the compression performances of the benchmarks. We apply 4 and 1 to the `INITIAL_CULLING_LIMIT` and the `MIN_CULLING_LIMIT` respectively used in the algorithm 2. The former constant value can be set to any small number because the compressor/decompressor accumulates data symbols in the beginning of the data stream into entries of the look-up table due to increasing L caused by the mishits. We have tried to change the constant value from 1 to 16. However, the performances did not change significantly. Therefore, we apply 4 to the `INITIAL_CULLING_LIMIT` in this evaluation.

The graphs in Figure 5 show the compression performances when the L and the E are varied. The cases of b) English text and c) Beauty show stable compression ratios when the L is varied during an E. In the cases when $L \geqq E$ and L is statically configured, all entries in the look-up table are protected by the entropy culling limit. This reason is because the look-up table is always full due to many mishits in the table. Therefore, we confirm that the L did not affect to a data sequence with high data entropy. On the other hand, the cases of a) gene DNA sequences and d) Sintel show different compression ratios when the L is varied during an E. The case of a) indicates the worse compression ratios when the L is too large or too small. However, the case of d) illustrates that the compression ratio becomes worse when the L increases. Comparing with those, we can confirm that the compression ratios with the autonomous adjustment of L show the middle performances between the best and the worse ones. This means that the algorithm 2 follows dynamic data entropy and the compressor assigns adaptively the small

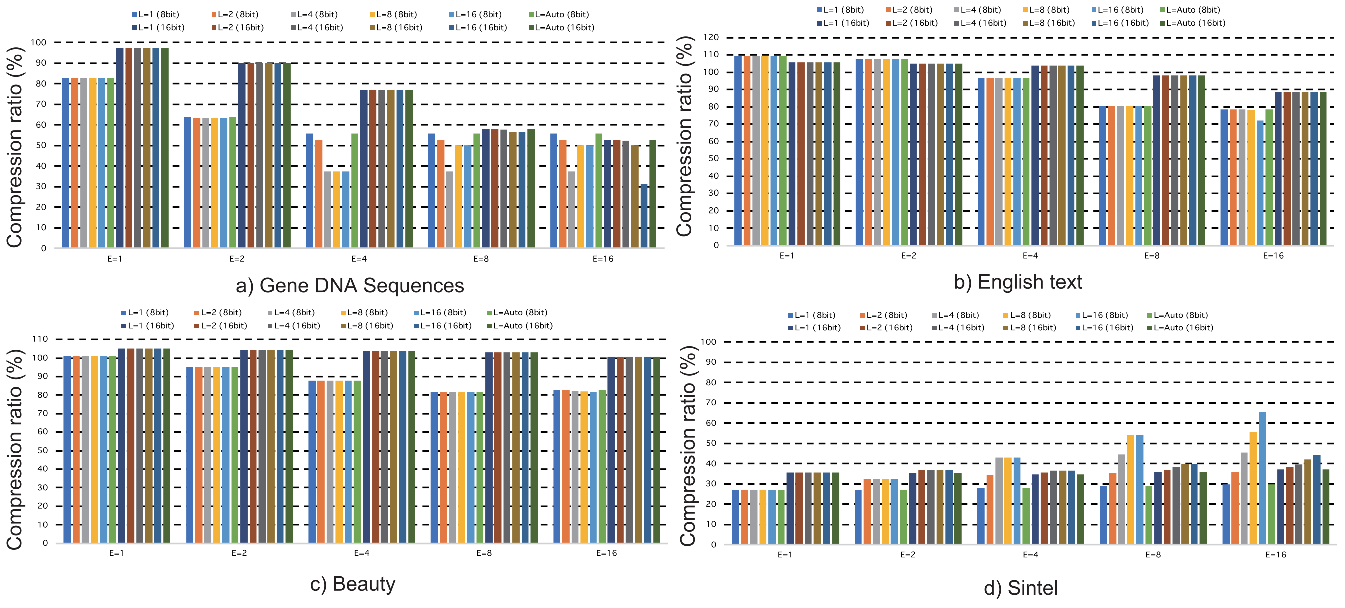

FIGURE 5. Evaluation of autonomous adjustment of entropy culling limit.

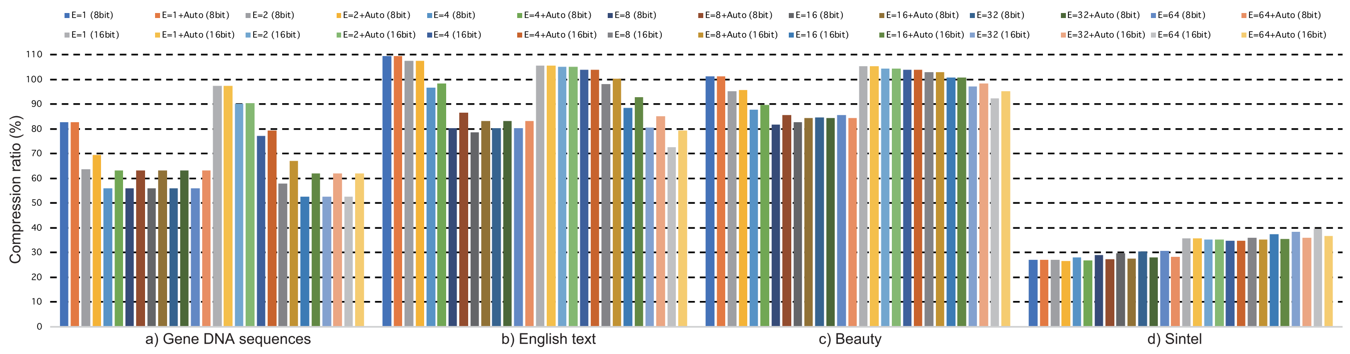

FIGURE 6. Evaluation of dynamic look-up table.

numbers of bits to the compressed symbols. Thus, according to this evaluation, we have confirmed that the proposed method regarding L works effectively.

D. EFFECT OF AUTONOMOUS ADJUSTMENT OF DYNAMIC LOOK-UP TABLE

Finally, we perform evaluation regarding the autonomous adjustment for the dynamic look-up table. This evaluation compares the compression ratios among the cases when the static table size is set to an E and the one when the autonomous method is applied. We use the ASE coding in which the algorithms 1 and 2 are embedded. We vary the table size E from 1 to 64 for each benchmark. In this evaluation, two types of evaluations for each E: one is the evaluation with a static E. This does not change the look-up table size dynamically. Another is the one with the autonomous adjustment regarding the E. This case changes the table size with following the algorithm 3. Regarding the autonomous adjustment of E, the `INITIAL_TABLE_SIZE` and `MIN_TABLE_SIZE` are set to the corresponding E for the experiment and 1 respectively.

The graphs in Figure 6 show the compression ratios of the benchmarks when we apply 8bit and 16bit to the symbol size. When a large number of E is applied to the experiment, in the case of 16bit symbol, the compression ratio becomes better than the ones shown in Figure 5 because the table size is larger than those. In the cases except d) Sintel, the compression ratios become better as the E is increased, and also the cases when the autonomous adjustment of E is applied shows a slightly worse than the one without the autonomous adjustment. However, the case of d), as the E is increased, the compression ratios become worse. When we focus on the compression ratios with the adjustment method of E, it becomes better than the ones without the adjustment. This shows clearly the good effect of the algorithm 3. In the cases of a), b) and c), the larger number of the look-up table entries brings the better compression ratios. However, the effect of increasing the number of table entries saturates when the E is large because the frequencies of registering symbols to the table and the one of removing the occupied entries from it are balanced. Therefore, the compressor and the decompressor should allocate the number of E as large as possible before the compression starts. While the compression process proceeds, the number of occupied entries in the table is optimized by the adjustment algorithm. Here, the most important point on which we should focus is to decide the best number of the table entries. But, it is impossible to decide it before the compressor starts its processes for a data stream. Therefore, if the `INITIAL_TABLE_SIZE` in the algorithm 3 is set to a number as large as possible, the algorithm optimizes the actual number of entries used during the compression autonomously. Thus, we have confirmed that the algorithm 3 works effectively to shrink the number of bits in a compressed symbol and provides the good compression ratios adaptively. According to the graphs, we can see the effect of the algorithm that the compression ratio can be adjusted to the middle performance between the best and the worst cases.

Through the experiments in this section, we have tried to see the effects of the proposed algorithms for the autonomous adjustments of the unstable parameters in ASE coding. The parameters d and L are adaptively adjusted by the algorithm 1 and 2 even if we set any small numbers to the initial values. Additionally, the parameter E can be initialized to the available number of allocated entries for the look-up table. The actual number of occupied entries in the table is autonomously adjusted to the data entropy of the input data stream. According to the discussion in this section, we conclude that the proposed strategies and the algorithms in this paper works effectively with contributing to the optimal data compression ratio.

V. CONCLUSION

This paper focused on autonomous adjustment method for stream-based data compression. We focused on ASE coding to optimize the compression performance that was heuristically determined by the parameters. There were three major parameters to be adjusted in ASE coding. The first parameter is the entropy culling that removes occupied entries in the look-up table. This mechanism works to reduce the number of bits in a compressed symbol if the setting follows the data entropy. The second parameter is the entropy culling limit that protects from excessive removal of the occupied entry by the entropy culling. The entropy culling limit contributes to raise probability of the table hit. The last parameter is related to the number of entries in the table. We proposed a new mechanism called the dynamic look-up table. It helps to shrink the compressed symbol by limiting the number of available entries in the table. We have proposed new algorithms regarding these three parameters. According to the evaluations using the algorithms, we have confirmed that the ASE coding with the autonomous parameter adjustment follows the data entropy dynamically and compresses data stream into between the best and the worst compression ratios. Thus, we conclude that the proposed autonomous adjustment algorithms worked effectively.

For the future work, according to the evaluations regarding the compression ratios, we have confirmed that the symbol width is sensitive to the compression ratio. However, it is related to the organization of the compressor and the decompressor of ASE coding. Therefore, we need to consider a new mechanism to decide the symbol width dynamically. We will investigate this problem for the next step of our research regarding the stream-based entropy coding.

REFERENCES

[1] A. Ukil, S. Bandyopadhyay, and A. Pal, "IoT data compression: Sensor-agnostic approach," in *Proc. Data Compress. Conf.*, Apr. 2015, pp. 303–312.

[2] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, "An energy efficient IoT data compression approach for edge machine learning," *Future Gener. Comput. Syst.*, vol. 96, pp. 168–175, Jul. 2019.

[3] P. G. Howard and J. S. Vitter "A universal algorithm for sequential data compression," *Inf. Process. Manage.*, vol. 28, no. 6, pp. 749–763, Nov. 1992.

[4] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[5] J. S. Vitter, "Design and analysis of dynamic Huffman codes," *J. ACM*, vol. 34, no. 4, pp. 825–845, Oct. 1987.

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

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

[8] S. Funasaka, K. Nakano, and Y. Ito, "Fast LZW compression using a GPU," in *Proc. 3rd Int. Symp. Comput. Netw. (CANDAR)*, Dec. 2015, pp. 303–308.

[9] P. A. Franaszek, L. A. Lastras-Montano, S. Peng, J. T. Robinson, "Data compression with restricted parsings," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2006, pp. 203–212.

[10] S. Yamagiwa, E. Hayakawa, and K. Marumo, "Stream-based lossless data compression applying adaptive entropy coding for hardware-based implementation," *Algorithms*, vol. 13, no. 7, p. 159, Jun. 2020.

[11] S. Yamagiwa, E. Hayakawa, K. Marumo "Adaptive entropy coding method for stream-based lossless data compression," in *Proc. 17th ACM Int. Conf. Comput. Frontiers (CF)*, 2020, pp. 264–268.

[12] K. Marumo, S. Yamagiwa, R. Morita, and H. Sakamoto, "Lazy management for frequency table on hardware-based stream lossless data compression," *Information*, vol. 7, no. 4, p. 63, 2016

[13] *LZ4*. Accessed: Sep. 3, 2020. [Online]. Available: https://lz4.github.io/lz4/

[14] E. Zohar and Y. Cassuto, "Automatic and dynamic configuration of data compression for Web servers," in *Proc. 28th USENIX Conf. Large Installation Syst. Admin.*, 2014, pp. 97–108.

[15] J. Liu, F. Chen, J. Yan, and D. Wang, "CBN-VAE: A data compression model with efficient convolutional structure for wireless sensor networks," *Sensors*, vol. 19, no. 16, p. 3445, Aug. 2019.

[16] Y. Z. Watkins and M. R. Sayeh, "Image data compression and noisy channel error correction using deep neural network," *Procedia Comput. Sci.*, vol. 95, pp. 145–152, Nov. 2016.

[17] P. Deutsch, *DEFLATE Compressed Data Format Specification Version 1.3*, document RFC 1951, Aladdin Enterprises, 1996

[18] *Google*. Accessed: Sep. 3, 2020. [Online]. Available: https://github.com/google/snappy

[19] W. T. Penzhorn, "A parallel algorithm for high-speed data compression," in *Proc. South Afr. Symp. Commun. Signal Process.*, Sep. 1992, pp. 173–174.

[20] B. Zhou, H. Jin, and R. Zheng, "A parallel high speed lossless data compression algorithm in large-scale wireless sensor network," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 6, Jun. 2015, Art. no. 795353.

[21] S. D. Agostino, "A parallel decoding algorithm for LZ2 data compression," *Parallel Comput.*, vol. 21, no. 12, pp. 1957–1961, Dec. 1995.

[22] S. Yamagiwa and H. Sakamoto, "A reconfigurable stream compression hardware based on static symbol-lookup table," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2013, pp. 86–93.

[23] S. Yamagiwa, K. Marumo, and H. Sakamoto, "Stream-based lossless data compression hardware using adaptive frequency table management," in *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, vol. 9495. Cham, Switzerland: Springer, 2015.

[24] K. Marumo and S. Yamagiwa, "Time-sharing multithreading on stream-based lossless data compression," in *Proc. 5th Int. Symp. Comput. Netw. (CANDAR)*, Nov. 2017, pp. 305–310.

[25] *Compressed Indexes and Their Testbeds*. Accessed: Sep. 3, 2020. [Online]. Available: http://pizzachili.dcc.uchile.cl/

[26] *Ultra Video Group*. Accessed: Sep. 3, 2020. [Online]. Available: http://ultravideo.cs.tut.fi/

[27] A. Mercat, M. Viitanen, and J. Vanne, "UVG dataset: 50/120fps 4K sequences for video codec analysis and development," in *Proc. 11th ACM Multimedia Syst. Conf.*, New York, NY, USA, May 2020, pp. 297–302.

[28] *Test Media*. Accessed: Sep. 3, 2020. [Online]. Available: https://media.xiph.org/

SHINICHI YAMAGIWA (Senior Member, IEEE) received the B.S. and M.S. degrees in information engineering and the Ph.D. degree in engineering from the University of Tsukuba, Japan, in 2002. He is currently an Associate Professor with the Faculty of Engineering, Information and Systems, University of Tsukuba. His current research interests in the computer science include the computer architecture, the embedded systems, the parallel and distributed computing, the stream-based data compression, and the data sciences for sports.

SUZUKAZE KUWABARA received the B.S. degree in information engineering from the University of Tsukuba, Japan, in 2019, where he is currently pursuing the M.S. degree in computer science. His main research interest in the computer science includes the data compression in embedded systems.