ADAPTATION OF BIT RECYCLING TO ARITHMETIC CODING

Ahmad Al-rababa’a and Danny Dubé

Computer Science and Software Engineering Department
Laval University, Canada
Ahmad.al-rababaa.1@ulaval.ca and Danny.Dube@ift.ulaval.ca

ABSTRACT

The bit recycling compression technique has been introduced to minimize the redundancy caused by the multiplicity of encoding. The multiplicity of encoding means that the source data may be encoded in more than one way. In its simplest form, it occurs when a compression technique with multiplicity of encoding feature has the opportunity at certain steps during the encoding process to encode the same symbol differently, i.e. different codewords for the same symbol can be sent to the decoder, any one of these codewords can be decoded correctly. When this opportunity occurs, the default behaviour of most techniques is to encode the symbol using the shortest codeword and less computation if possible. Many applications include the multiplicity of encoding feature, such as LZ77 (Lempel and Ziv, 1977) [1] and its variants, some variants of Prediction by Partial Matching (PPM) technique [2], Volf and Willems switching-compression technique [3], and Knuth’s algorithm [4] for the generation of balanced codes.

The bit recycling technique [5] has been introduced to minimize the redundancy caused by the multiplicity of encoding problem. It has exploited the multiplicity of encoding in a certain way, in which it is not always necessary to select the shortest codeword, but instead to take all the appropriate codewords into account with some agreement between the encoder and the decoder. It turned out that bit recycling was able to achieve better compression by exploiting the multiplicity of encoding feature rather than systematically selecting the shortest codeword. Variants of bit recycling have been applied on LZ77 algorithm in [6] and [8], the experimental results of [7] showed that bit recycling has achieved a reduction of about 9% in the size of files that has been compressed by Gzip [9].

1. INTRODUCTION

Data compression aims to reduce the size of data so that it requires less storage space and less bandwidth of the communication channels. Many compression techniques suffer a problem that we call the redundancy caused by the multiplicity of encoding. The multiplicity of encoding means that the source data may be encoded in more than one way. In its simplest form, it occurs when a compression technique with multiplicity of encoding feature has the opportunity at certain steps during the encoding process to encode the same symbol differently, i.e. different codewords for the same symbol can be sent to the decoder, any one of these codewords can be decoded correctly. When this opportunity occurs, the default behaviour of most techniques is to encode the symbol using the shortest codeword and less computation if possible. Many applications include the multiplicity of encoding feature, such as LZ77 (Lempel and Ziv, 1977) [1] and its variants, some variants of Prediction by Partial Matching (PPM) technique [2], Volf and Willems switching-compression technique [3], and Knuth’s algorithm [4] for the generation of balanced codes.

The bit recycling technique [5] has been introduced to minimize the redundancy caused by the multiplicity of encoding problem. It has exploited the multiplicity of encoding in a certain way, in which it is not always necessary to select the shortest codeword, but instead to take all the appropriate codewords into account with some agreement between the encoder and the decoder. It turned out that bit recycling was able to achieve better compression by exploiting the multiplicity of encoding feature rather than systematically selecting the shortest codeword. Variants of bit recycling have been applied on LZ77 algorithm in [6] and [8], the experimental results of [7] showed that bit recycling has achieved a reduction of about 9% in the size of files that has been compressed by Gzip [9].

2. BACKGROUND

2.1. LZ77 and the multiplicity of encoding

LZ77 is a compression technique that compresses a string of characters $S$ by transmitting a sequence of messages. A message is either a literal message, denoted by $c$, which means that the next character is $c$, or a match message, denoted by $(l,d)$, which means that the next $l$ characters are a copy of the $l$ characters that appear $d$ characters before the position of the symbol being encoded in $S$. For example, let $S$ be “abbaaababbabbb”, the underlined substring is the prefix that has been encoded so far. The next character to be encoded is “a”. The position of character “a” is called the current position of encoding. Notice that the substrings “a”, “ab” (“a” followed by the first “b”) and “abb” have many copies at different $d$’s in the already encoded prefix before the current position. For instance, “abb” has a copy at the distances 3, 6, and 11. LZ77 typically selects the longest match (“abb”) and, among the available longest matches, it selects the choice at the closest distance ($d = 3$), therefore the match message $(3,3)$
will be transmitted and the current position will be moved to the last "b".

It is clear that LZ77 has the selection freedom to encode "abb" by any one of the equivalent messages \( (3,3) \), \( (3,6) \), and \( (3,11) \), denoted by \( M_1 \), \( M_2 \), and \( M_3 \) respectively. These messages are called equivalent messages since any one of them can be used to encode the same substring "abb". Accordingly, many different sequences of messages can be transmitted to represent the compressed stream of \( S \), any possible sequence will be decoded correctly. This feature is an instance of the multiplicity of encoding. Most implementations of LZ77 select the longest match, since selecting the longest match enables the encoder to use almost the same number of bits to encode the longest possible string. For instance, encoding three characters ("abb") using \( M_1 \), \( M_2 \), or \( M_3 \) is better than encoding one character ("a") using the message [a] or two characters ("ab") using the message (3, 2). When many longest matches exist, the closest one is preferred based on the notion that the statistical distribution of the transmitted distances tends to be skewed, with higher frequencies for the short distances. This allows the statistical encoders to take advantage of this skewness and send shorter codewords on average.

2.2. Bit recycling compression technique

The main objective of the bit recycling technique is to minimize the redundancy caused by the multiplicity of encoding. To show the principle of bit recycling by applying it on the same example given above, consider the bit recycling compressor depicted in Fig. 1. The current position of encoding is represented by the point at time \( t \). The same three equivalent messages \( M_1 \), \( M_2 \), and \( M_3 \) are available to encode the string "abb" from \( t \) to \( t + 1 \). The arc length represents the cost of each message, i.e. the message codeword length in bit. We already mentioned that \( M_1 \) is the default choice in most implementations of LZ77.

What could we hope to obtain by selecting a message other than \( M_1 \)? The answer of this question is that the act of choosing among the equivalent messages constitutes a form of implicit communication from the compressor to the decompressor; bit recycling utilizes this implicit communications to achieve better compression. Let’s explain the answer based on Fig. 1. At time \( t \), the compressor creates a prefix code for the messages \( M_1 \), \( M_2 \), and \( M_3 \) and according to their probability. The created codewords, say 0, 10, and 11 for \( M_1 \), \( M_2 \), and \( M_3 \) respectively are called the recycled codewords. Before moving on, it will be much easier to explain how the decoder behaves at time \( t \), then to resume the discussion of the encoding process.

On the other side, the decoder will receive \( M_t, i \in \{1, 2, 3\} \), at time \( t \). So it first decodes the received message to "abb", then it can infer that the decoded substring "abb" has two other copies in the decoded prefix, i.e. the decoder can list the three equivalent messages that could have been sent by the encoder. Therefore, the list of equivalent messages at time \( t \) can be established by both the encoder and the decoder identically and implicitly. Knowing the list of equivalent messages, the decoder can rebuild the corresponding Huffman tree that has been built by the encoder, which means that one of the recycled codewords of the available messages has been transmitted implicitly from the encoder to the decoder. If the compressor has decided to send \( M_1 \), \( M_2 \), or \( M_3 \) then the codeword 0, 10, or 11 is transmitted implicitly respectively. The selection of message \( M_i \) among \( M_1 \), \( M_2 \), and \( M_3 \) constitutes an eye wink from the encoder to the decoder. The key question now is: where is the compression? According to an agreement between the encoder and decoder and based on the available information, the decoder inserts the recycled codeword of the received message into the compressed stream just before \( t + 1 \); accordingly, these bits have been omitted from that location by the encoder. Note that the compressor can offer to omit these bits because they are transmitted implicitly by the selection process among the equivalent messages. Next, we show how and why the recycled codeword of the selected message (eye wink) can be omitted by the encoder.

Let us get back to the encoder at time \( t \). The encoder at this point has three recycled codewords 0, 10, and 11 available, but it does not know which one to select yet. Hence, these codewords need to be compared with the bits starting from \( t + 1 \), which is the beginning of the next message codeword. There will be one and only one match between the first bits of the next message and a recycled codewords. The corresponding message of the match will be the eye wink. As a consequent, the matched bits starting from \( t + 1 \) need not be sent. The matched bits can be omitted from the stream, since they can be deduced implicitly by the decoder as described above. The bits that have been omitted at the encoder side and restored at the decoder side are called the recycled bits and the technique itself, bit recycling. The recycled bits at each step is one of the recycled codewords created at that step, that is why we called it the recycled codeword. Therefore, the inserted codeword is sent for free by transmitting it through the message selection process rather than through the explicit communication which is the compressed stream. The sequence of implicitly transmitted recycled codewords forms a kind of implicit channel between the encoder and the decoder. We call this implicit channel the side-channel.

For example, let the standard codewords of \( M_1 \), \( M_2 \), and \( M_3 \) be \((1001)_6\), \((11110)_6\), and \((1111100000010101)_6\),

![Fig. 1. Principle of bit recycling compressor](image-url)
and then determines that codeword by the encoder. It can build the recycling code of Fig. 2 and then determines that codeword (10)_b should be recycled. The recycled codeword (10)_b need to inserted into the bit stream as illustrated at instant III. At time t + 1 and Instant I, the bit stream is exactly left as it was at time t and instant III. If the bit stream at instant I were (1001001101010101...)_b, the underlined codeword would be the codeword of M_2, then the decoder would have to recycle bit 0. So the bit stream at instant III be (001101101010101...)_b. Since recycling proceeds by extracting prefixes of the bit stream, which is a sequence of entropy-encoded events, the bit stream starts by 0 half the time, and by 1 the other half of the time; also the bit stream starts by 11 one fourth the time and so on. Accordingly, M_1 has probability \( \frac{1}{2} \) of being selected because its recycled codeword is 1-bit long (0), M_2 has probability of \( \frac{1}{4} \) of being selected because its recycled codeword is 2-bits long (10) and so on.

Let \( c_i \) is the cost of message \( i \). The cost of the message is the codeword length in bits for that message. Let the costs of the messages M_1, M_2, and M_3 be 4, 5, and 15 bits, respectively. Let \( p_i \) be the probability of message \( i \), and assume roughly \( p_1 \), \( p_2 \), and \( p_3 \) be 0.0625, 0.03125, and 0.0000305 respectively. By default 4 bits is the cost of the message selected by LZ77, i.e. the closest longest match. By applying bit recycling, the net cost of M_1 is 3 bits, since the cost of M_1 is 4 bits minus 1 recycled bit, which is less than the default cost. If M_2 was the selected message, then the net cost would be 3 bits (5 minus 2). But what if M_3 was the selected message? The net cost would be 13 bit (15 minus 2) which is much greater than the default cost. The average net cost \( NC \) of the available messages is:

\[
NC = \sum_{i=1}^{n} (c_i - |r_i|) \cdot \frac{1}{2^{|r|}}
\]

where \( n \) is the number of the available messages, \( r \) is the recycled codeword, and \( |r| \) is the length of the recycled codeword. So \( NC \) for M_1, M_2, and M_3 is 5.5 bits, which is also greater than the default cost, the reason of this high average net cost is the costly message M_3. Therefore the authors solved the problem of the costly messages by developing an efficient algorithm for constructing an optimal whole-bit recycling code in their work [8]. The main target of the algorithm is to drop the costly messages that affect \( NC \) negatively. To examine \( NC \) after dropping M_3; the recycled codewords for the two messages M_1 and M_2 become 0 and 1, therefore each time 1 bit get recycled. Then \( NC \) will be: \( (4-1)2^{-1} + (5-1)2^{-1} = 3.5 \) bits. This is cheaper than systematically choosing M_1.

### 2.3. The weakness of bit recycling

We have shown that the average net cost (3.5 bits) of the choices after dropping the costly message is less than the default cost (4 bits) that has been achieved by LZ77. But did bit recycling achieve the perfect (maximum) recycling by this average net cost? To answer this question, we first need to know what the minimum average net cost according to the messages probabilities is. Let \( T \) be the self-information of the set of the equivalent messages \( \{M_i\}_{i=1}^n \), where \( n \) is the number of the available messages. \( T \) represents the minimum average net cost of \( n \) of the available messages. \( T \) equals:

\[
T = -\log \sum_{i=1}^{n} p_i
\]

\( T \) in our example without dropping the costly message is 3.414 bits, and with dropping the costly message is 3.415 bits. So Huffman-based bit recycling (HuBR) did not achieve perfect recycling neither with dropping the costly message nor without. Notice that, after dropping M_3, HuBR has assigned the two remaining messages M_1 and M_2 the same opportunity (50% for each one) to recycle one bit, while \( p_1 \gg p_2 \). To achieve perfect recycling, a higher probability (\( p_1/(p_1 + p_2) = 0.667 \)) should be assigned to M_1 to recycle less bits (4 − 3.415 = 0.58 bit), and less probability (\( p_2/(p_1 + p_2) = 0.333 \)) for M_2 to recycle more bits (5 − 3.415 = 1.58 bits). This fractional assignment will enable bit recycling to achieve perfect recycling by utilizing the choices probabilities fractionally. But Huffman code is constrained to generate codewords of integer lengths, so it can utilize only powers of \( \frac{1}{2} \) probabilities. Therefore, HuBR could not achieve perfect recycling due to the nature of Huffman coding.

Arithmetic coding is unlike Huffman coding, since it does have the ability to utilize the ratio between the messages probabilities fractionally and to recycle fractions of bits. It is able to use the fractional assignment explained above. Moreover, it is much closer to the theoretical lower bound described above, since it has been proven by Howard and Vitter in [12] that the degradation in the code efficiency caused by practical arithmetic coding is negligible. Consequently, it is worthwhile to adapt bit recycling to arithmetic code.

### 2.4. Arithmetic code

In this section we briefly review the main notations of the first-in first-out (FIFO) recursive version of arithmetic coding [13]. Let the statistical model consist of the alphabet symbols indexed by \( i = 0,1 \ldots m-1 \), with the corresponding symbol probability \( p_i \) and the cumulative probability \( Q_i \), where:

\[
Q_i = \sum_{i=0}^{i-1} p_i
\]
and \( Q_0 = 0 \). The unit interval \([0,1)\) is divided into subintervals proportionally to the alphabet symbols probabilities. The following two recursive equations will be updated at each coding step until the last symbol is encoded.

\[
C_{n+1} = C_n + (A_n \times Q_n) \quad (4)
\]
\[
A_{n+1} = A_n \times p_n \quad (5)
\]

where \( n \) is the number of the coding step, \( C \) is the code point of the encoded string, which is the code that is ready to be emitted to the decoder and will never be changed again and \( A \) is the interval width from \( C \) as the base of the interval. The value of the product \((A_n \times Q_n)\) is called the augend, that is the new added value to the current \( C \) by encoding the next symbol. Initially (at time \( t = 0 \)), \( C_0 = 0 \) and \( A_0 = 1 \), so that the initial interval is \([0,1)\).

At the last step of coding, the encoder emits any value \( v \) within the interval \([C, C + A)\).

The decoder at the other side reverses what the encoder did. It starts with the initial values \( A = 1 \) and \( C = v \). The first symbol is decoded directly by mapping \( C \) value with the cumulative probabilities of the model.

To decode the next symbol, the value of \( C \) will be updated according to the following equation:

\[
C_{n+1} = C_n - (A_n \times Q_n) \quad (6)
\]

3. ARITHMETIC BIT RECYCLING SCHEME

We explain our scheme based on the same example given in Section 2. We start by describing the interval composition and calculation for arithmetic-based bit recycling scheme (ACBR), in which we set up the necessary arrangement between the encoder and decoder to control the transmission of equivalent messages self-information. To describe how ACBR works, let \( S \) be the same string at the same current position of encoding. The encoder at time \( t \) has the interval \( I_t \) shown in Fig. 3. \( I_t \) is divided into \( k \) subintervals proportionally to the probabilities of the set of messages \( \{m_i\}_{i=0}^{k-1} \) that could be sent at time \( t \). This set of messages can be seen as the alphabet at time \( t \). The encoder has the same equivalent messages \( M_1, M_2, \) and \( M_3 \).

Consider the arrangement shown in Fig. 3. Each equivalent message offered a copy of the new composed interval \( I_{t+1} \), that is composed by stacking all the equivalent messages according to their occurrence in the model from message 0 to \( k-1 \). Let the subinterval corresponding to the equivalent message \( i \) at time \( t+1 \), denoted by \( I_{t+1}^i \). Interval \( I_{t+1} \) represents the self-information of the available equivalent messages. The width \( L \) of interval \( I_{t+1} \) equals: \((P(M_1) + P(M_2) + P(M_3)) \times L(I_t)\), where \( P \) is the probability. \( I_{t+1} \) will be the available interval to encode the next message at time \( t + 1 \). According to the position of the bit stream that continues the encoding of the remaining messages at time \( t + 1 \), one and only one of the equivalent messages will be encoded. The position of the of the bit stream at time \( t+1 \) depicted by the zero thickness arrow in Fig. 3. Notice that the arrow points to \( I_{t+1} \) and to \( I_{t+1}^2 \) at the same time. Since \( I_{t+1}^2 \) belongs to \( M_2 \) (in its scope), then the encoder selects message \( M_3 \). The encoder provides the new interval \( I_{t+1} \) for the next message to be encoded instead of \( I_{t+1}^2 \). Accordingly, the \( C \) and \( A \) values corresponding to \( I_{t+1} \) need to be provided for the next message instead of the \( C \) and \( A \) values of \( I_{t+1}^2 \) and according to (4) and (5). The same principle for \( M_1 \) and \( M_3 \) as follows. If the arrow points above \( I_{t+1}^2 \), so it points to \( I_{t+1}^2 \), which is the part of \( I_{t+1} \) related to \( M_1 \), accordingly, the encoder selects \( M_3 \). In the same way, the encoder selects \( M_1 \) if the arrow points below \( I_{t+1}^2 \). Clearly, this procedure reduces the amount of information required to encode any one of \( M_1, M_2, \) and \( M_3 \) messages individually and according to (2). Hence, one and only one of the three available copies will be used according to the cumulative probability of next message to be encoded.

The reasoning behind this is that, the \( C \) value represented by the arrow with respect to \( I_t \) is the value that will be used by the decoder to decode one of the messages \( M_1, M_2, \) and \( M_3 \), so the decoder will be able to decode \( M_2 \) successfully.

Encoding the next symbol using a wider interval results in fewer bits to be sent to the decoder, i.e. more recycled bits. Widening the interval from \( I_{t+1}^2 \) to \( I_{t+1} \) represents the arithmetic recycling, and according to (2), the recycled bits will be: \( \log L(I_{t+1}) - \log L(I_{t+1}^2) \). In our example, the number of recycled bits by ACBR is 1.585 bits, and the average net cost is 3.414. Notice that, 3.414 bits is less than 3.5 bit that has been achieved by HuBR. On top of that, the costly message \( (M_3) \) does not affect the average net cost negatively like in HuBR, but conversely, it contributes positively in widening the composed interval, therefore, we need not to be worried about the costly messages anymore.

The knowledge of this arrangement will be known for the decoder; therefore this knowledge represents the implicit information that can be sent for free to the decoder. Intuitively, the extra compression is a result of sending the knowledge about redundancy implicitly rather than explicitly through the compressed stream. Based on this ar-
rangement, the decoder at time \( t \) and instant \( I \) will decode as usual according to (6), notice that the value of \( C_t \) (the value of \( C \) at time \( t \)) is the position of the arrow with respect to \( I_t \) and \( A_t \) is the width of \( I_t \). It is obvious that the arrow points to \( M_2 \), therefore \( M_2 \) can be decoded successfully without any modification in the decoding process. At instant II, the decoder realizes that \( M_2 \) has two other equivalents, \( M_1 \) and \( M_3 \), thereby, it has to do the necessary modification according to the eye wink, of course it can rebuild the necessary knowledge to do the required modifications for both \( C \) and \( A \). In other words, the decoder has to decode the next message according to \( I_{t+1} \) instead of \( I_{t+1}^2 \). Therefore, the values of \( C_{t+1} \) and \( A_{t+1} \) will be:

\[
C_{t+1} = C_t - Q_{M_2} \times L(I_t) + P(M_1) \times L(I_t) \quad (7)
\]

\[
A_{t+1} = L(I_t) \times (P(M_1) + P(M_2) + P(M_3)) \quad (8)
\]

The bold parts in (7) and (8) are the parts that are calculated by the decoder without recycling, which also represents the \( C \) and \( A \) values of \( I_{t+1}^2 \). The value of \((P(M_1) \times L(I_t))\) is added in (7) because it has been subtracted from the \( C \) value of \( M_2 \) message by the encoder, i.e. the decoder exactly undoes what the encoder did. At time \( t + 1 \) and Instant III, the same position of the arrow with respect to \( I_{t+1} \) will be used to decode the next message as described above and so on. The performance of the ACBR scheme is a bit above an order of magnitude slower than arithmetic code.

### 4. THEORETICAL ANALYSIS

To compare the performance of HuBR and ACBR, we assume that we have different number of choices to choose among the equivalent messages with uniform probabilities. This situation does not need to drop the costly choices and will not affect the performance of HuBR negatively. Note that this assumption will not affect the performance of the ACBR. The estimated average number of recycled bits for different number of choices will be computed for both HuBR and ACBR based on this assumption.

To compute the average number of recycled bits for HuBR with equiprobable choices, let \( m \) be the number of the available choices and \( w_i \) be the codeword length of choice \( i \) for \( i = 2, \ldots, m \). Huffman tree of \( m \) equiprobable choices contains two codeword lengths, \( k_1 = \lceil \log_2 m \rceil \) and \( k_2 = \lfloor \log_2 m \rfloor \). Accordingly, there will be \( m - 2d \) codewords of length \( k_1 \) and \( 2d \) codewords of length \( k_2 \), where \( d \) is given by:

\[
d = m - 2 \lfloor \log_2 m \rfloor \quad (9)
\]

The probability \( p_i = (1/2^{k_i}) \) is the probability of choice \( i \) to be selected. Since there are two different lengths (two \( j \)’s), then the average number of recycled bits \( AV_H \) for HuBR will be:

\[
AV_H = (m - 2d) \cdot k_1 \cdot \frac{1}{2^{k_1}} + 2d \cdot k_2 \cdot \frac{1}{2^{k_2}} \quad (10)
\]

Therefore,

\[
AV_H = (m - 2d) \cdot k_1 \cdot \frac{1}{2^{k_1}} + 2d \cdot k_2 \cdot \frac{1}{2^{k_2}} \quad (10)
\]

Now, the average net cost \( C_H \) for HuBR without dropping the costly choice is:

\[
C_H = \frac{1}{2} (c_1 - 1) + \frac{1}{2} (c_2 - 1) \quad (14)
\]

By substituting (13) into (14), we end up with the following equation:

\[
C_H = (-\frac{1}{2} \log_2 (p_1 \cdot p_2))^{-1} \quad (15)
\]

In order to take into consideration the ability of HuBR to drop the costly choice, let \( NC_H \) be the average net cost for HuBR with dropping the costly choice, which should be selected such that, the minimum value of \( \{C_H, c_1, c_2\} \). Therefore \( NC_H \) is:

\[
\min \left\{ \left(-\frac{1}{2} \log_2 (p_1 \cdot p_2) \right) - 1, -\log_2 p_1, -\log_2 p_2 \right\} \quad (16)
\]
ical analysis showed that the weakness of Huffman-based bit recycling (HuBR), by adapting it to arithmetic code. The framework and the main concepts of ACBR have been explained. The theoretical analysis showed that ACBR achieves perfect recycling in all cases whereas HuBR achieves perfect recycling only in particular cases. Consequently, a significant amount of better compression can be achieved by ACBR. A lot of work is needed to be done. A more general and precise algorithm needs to be designed. We intend to adjust the proposed scheme so that it can be implemented using fixed-length registers, since it currently uses arbitrary-precision calculations. Adapting ACBR with the multiplication-free arithmetic coding [14] would be an issue to lessen the computation complexity. Afterwards, ACBR can be implemented and applied on the Calgary corpus files [15] to evaluate its performance in practice.

6. ACKNOWLEDGEMENT

The authors would like to thank the anonymous referees whose comments helped to improve this paper. This work has been funded by NSERC of Canada.

7. REFERENCES