Using Bit Recycling to Reduce Knuth's Balanced Codes Redundancy

Ahmad Al-rababa'a
Laval University, Canada
Ahmad.Al-rababaa.1@ulaval.ca

Danny Dubé
Laval University, Canada
Danny.Dube@ift.ulaval.ca

Jean-Yves Chouinard
Laval University, Canada
Jean-Yves.Chouinard@gel.ulaval.ca

***Abstract*—Donald Knuth published an efficient algorithm for constructing a code with balanced codewords. A balanced codeword is a codeword that contains an equal number of zero's and one's. The redundancy of the codes built using Knuth's algorithm is about twice the lower bound on redundancy. In this paper we propose a new scheme based on the bit recycling compression technique to reduce Knuth's algorithm redundancy. The proposed scheme does not affect the simplicity of Knuth's algorithm and achieves less redundancy. Theoretical results and an analysis of our scheme are presented as well.**

I. Introduction

A binary balanced codeword is a codeword that contains an equal number of ones and zeroes. The balanced codes have found many applications, such as the digital recording on magnetic and optical storage devices like magnetic tapes, Compact Disks (CDs), Blue-ray disks (BD) and DVDs, and the applications in both the optical networks and error correcting and detecting codes.

In 1986, Donald Knuth published an efficient algorithm for constructing balanced codes [1]. Knuth's algorithm is efficient due to its simplicity and the fact that it does not need look-up tables. The main idea of Knuth's algorithm is that any unbalanced user data word w of length m, m even, can be encoded to construct a new balanced codeword, say $\hat{w}$, by complementing the first k bits of w. A balanced prefix codeword v of even length p, which represents the index k, is appended in front of $\hat{w}$. Thus the whole new constructed balanced codeword is composed of v followed by $\hat{w}$ ($v\hat{w}$). The decoder decodes $v\hat{w}$ as follows. It reads the first p bits which are the prefix codeword for k; then it knows the number of bits of the start of $\hat{w}$ that need to be complemented in order to retrieve the original word w. The prefix codeword itself will be discarded once the original word is retrieved.

Restricting codewords to balanced ones necessarily introduces redundancy. However, the redundancy of Knuth's balanced codes is about twice the lower bound (theoretical bound) on redundancy, as it has been shown by Weber and Immink [2]. Later in the same work [2], the authors presented their first attempt to close the gap between Knuth's redundancy and the lower bound redundancy. The first improvement attempt achieved only little profit, but later on they achieved a significant improvement [3].

In this paper, we propose a new scheme to address the same problem. Our scheme to reduce Knuth's algorithm redundancy is based on the *bit recycling* compression technique, presented by Dubé and Beaudoin [4], [5], [6], [7], [8]. The bit recycling has been introduced to minimize the redundancy caused by the *multiplicity of encodings*. The multiplicity of encodings means that the source data may be encoded in more than one way. In its simplest form, it occurs when a technique has the opportunity, at certain steps, to encode the same symbol differently, i.e. different codewords for the same symbol can be sent to the decoder. Any one of theses codewords can be decoded correctly. Knuth's algorithm does have the multiplicity of encoding property as we show later on in this paper. Hence, the objective of this work is to apply bit recycling on Knuth's algorithm to reduce the redundancy gap mentioned above.

The outline of the next sections is as follows. In Section II we present the notion of generating balanced codes and the minimum redundancy needed to produce the full sets of balanced codewords. We also review Knuth's algorithm and its redundancy. Section III contains the description of the proposed scheme. A theoretical analysis and results are given in Section IV. Section V concludes this paper.

II. Background

A. Balanced codes redundancy

A binary codeword that has an equal number of zeros and ones is called a balanced codeword, i.e. if m is the length of the balanced codeword then it should contain exactly $m/2$ ones and $m/2$ zeros. Generating balanced codeword necessarily incurs redundancy, since among the m-bit words, there are fewer than 2^m balanced codewords. To show how many bits we need to do that, we have first to know how many balanced codewords exist among all the possible m-bit words.

Theoretically, the number of m-bit balanced codewords is

$$\binom{m}{m/2} \approx \frac{2}{\sqrt{2\pi m}} \cdot 2^m. \tag{1}$$

The approximation is due to Stirling. Since

$$\frac{2}{\sqrt{2\pi m}} < 1; \text{ for } m \geq 1; \tag{2}$$

then extra bits are needed as a redundancy to produce a full sets of balanced codewords (i.e. 2^m codewords). The number of extra bits (parity bits), p, that are needed to form the full sets of 2^m balanced codewords can be calculated as follows

$$p = m - \log_2 \binom{m}{m/2}, \tag{3}$$

which has been approximated by Knuth [1] to

$$p \approx \frac{1}{2}\log_2 m + 0.326, \text{ for } m \gg 1. \tag{4}$$

Therefore, the redundancy R equals

$$R(m) = \frac{\frac{1}{2}\log_2 m + 0.326}{m} \approx \frac{1}{2m}\log_2 m. \qquad (5)$$

R represents the minimum redundancy (the lower bound) to generate the full sets of balanced codewords by any technique or algorithm.

B. Knuth's algorithm

Before we delve into the details, let's explain the main principle of Knuth's algorithm using the following concrete example. Let w of length 8 bits be '10101101', it is clear that this word is unbalanced, since it contains 5 one's and 3 zero's. By complementing the first bit ($k = 1$), the new constructed balanced codeword, $\hat{w}$, will be '00101101'. Let v, the balanced prefix codeword for k, be '000111', '001011', '001101', '001110', '010011', '010101','010110', and '011001' for k = 1, 2,..., and 8 respectively. Then the whole new constructed balanced codeword is '**000111**00101101' of length $p+m = 14$ bits. The decoder reads the first 6 bits '000111'. knowing that '000111' is the prefix codeword for $k = 1$, then the decoder complements the first bit of '**0**0101101' to reconstruct the original word '10101101'. The prefix codeword '000111' is discarded after reconstructing the original word.

Notice that the encoder does not have to choose $k = 1$. It could also choose $k = 3$, 5, or 7. So it has the freedom to construct any of the following four balanced codewords '00101101', '01001101', '01010101', and '01010011' by complementing the first $k = 1$, 3, 5, and 7 bits respectively. This *selection freedom* means that Knuth's algorithms does have the multiplicity of encoding property, since the encoder can encode the original word '10101101' in multiple ways. It has been proved by Knuth [1] that there is at least one *balance point* (k) within the word w of length m, m even. Weber and Immink also showed that the number of k's (balance points) is at most $m/2$. Thus the number, c, of the available k's to balance any arbitrary m-bit word is

$$1 \leq c \leq m/2. \qquad (6)$$

According to Knuth's basic algorithm, we need p extra bits to encode k. Knuth showed that, in his best construction, p equals

$$p \approx \log_2 m + \frac{1}{2}\log_2 \log_2 m. \qquad (7)$$

Therefore, Knuth's algorithm redundancy, *KR*, equals

$$KR(m) = \frac{\log_2 m + \frac{1}{2}\log_2 \log_2 m}{m} \approx \frac{1}{m}\log_2 m, \ m \gg 1. \quad (8)$$

As it has been concluded in [2], *KR* is about twice R which is given in (5). I.e. Knuth's redundancy *KR* is about twice the minimum redundancy R. Weber and Immink [2] have computed the average information, AV_c, that can be conveyed by the selection freedom described above. AV_c has been approximated for large m to

$$AV_c(m) = \frac{1}{2}\log_2 m - 0.196. \qquad (9)$$

They commented on the value of AV_c, explaining that it compensates for the loss in code rate between codes based on Knuth's algorithm and codes based on full balanced codeword

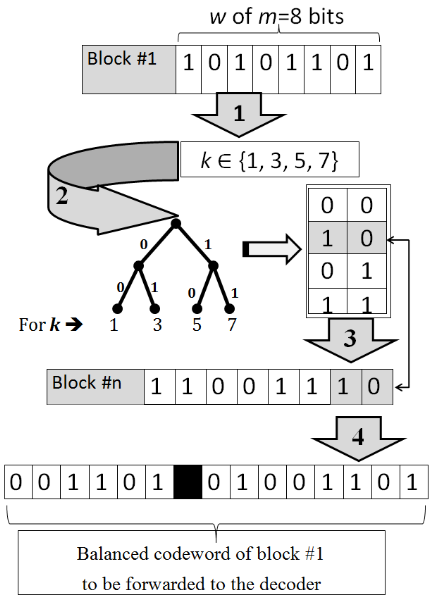

Fig. 1. The main steps of bit recycling for Knuth's algorithm.

sets. Obviously, this is an opportunity for bit recycling to minimize this gap in redundancy, since the aim of bit recycling is to exploit such a selection freedom (the multiplicity of encodings) to minimize the redundancy as we show in Section III.

III. Applying bit recycling on Knuth's algorithm

We have shown the presence of multiplicity of encoding in Knuth's algorithm and the significant average amount of information that can be saved by this property to compensate for the loss in code rate due to Knuth's redundancy. For the sake of space, we explain the principle of bit recycling [4] implicitly by applying it directly on Knuth's algorithm.

We start explaining our scheme, Bit Recycling for Knuth's Algorithm (*BRKA*), using Fig. 1, which contains a practical example to illustrate the main steps of the *BRKA* encoder. To keep consistency, the first block to be encoded in Fig. 1 is the same word, '10101101', that we considered in Section II. For simplicity, we consider the whole unbalanced stream, σ, which consists of n blocks to be the input stream of *BRKA*. The *BRKA* encoder constructs for each block *BL* of length $m = 8$, the corresponding balanced block *BL*$'$. *BL*$'$ is composed of the prefix balanced codeword v, of fixed-length $p = 6$, followed by $\hat{w}$ of length $m = 8$. We assume that all 2^m original blocks are equiprobable and independent. We first explain the *BRKA* encoder.

A. The BRKA encoder

The encoder encodes σ step by step as follows:

1) Read the next block, *BL*, and inspect it for the full set of balance point, i.e. the k's that can make it balanced. In the example depicted in Fig. 1, we can observe that *BL* can be balanced by using any k in {1,3,5,7}.
2) Generate the prefix code for the choices (the k's) identified in step (1).
The prefix codes for the available k's are constructed

as shown in Fig. 1. The Huffman tree *TR* for k's is constructed assuming that the k's are equiprobable. We call the constructed prefix codewords, '00', '01', '10', and '11' for k = 1, 3, 5, and 7 respectively, the *recycled codewords*. Notice that the table besides *TR* lists in reversed order the recycled codeword bits.

3) Compare each recycled codeword constructed in step (2) with the last bits of σ. This matching should be done backward starting from the end of σ. There should exist one and only one match. Delete from the end of the stream the bits that match one of the recycled codewords.
In our example, the constructed codeword '01', the recycled codeword for k=3, matches the last two bits of σ from right to left. So the last two bits '10' are deleted from σ. Next, we explain the reasoning behind this step.
4) The processed block is balanced by complementing the first k bits, where k is the choice whose recycled codeword matches the last bits of σ in step (3), and the balanced prefix codeword, v, of the *selected* k is appended in front of it. Thus the whole new constructed balanced codeword *BL*$'$ ($v\hat{w}$) will be the block to be sent to the decoder.
According to this step, $\hat{w}$ = '01001101' is constructed by complementing the first 3 bits ($k = 3$) of *BL*, and *BL*$'$ ($v\hat{w}$) will be '**001101**01001101'.
5) If there remains unprocessed blocks in the stream then go to step (1), otherwise, make the necessary termination and stop.

In step (3) the encoder provides a complete solution by constructing a new prefix recycled codewords, '00', '01', '10', and '11' according to the available k's that can balance the original word w, since the last two bits of any arbitrary binary stream should be either '00', '01', '10', or '11'. In case of three choices, the generated recycled codewords would be '0', '10', '11', (or '1', '00', '01') and so on. Notice that it is step (3) that performs the operations that reap the benefits of bit recycling.

Step (3) is necessary for the decoder since the received block, '**001101**01001101', contains the value of k=3 as a signal, on which the decoder depends to restore the corresponding recycled codeword that has been deleted by the encoder at the end of σ. The decoder decodes the block '**001101**01001101', realizing that the k value of the received block is 3 and the original word w is '10101101'. Also, it can infer that w can be balanced by using any k in {1, 3, 5, 7}. Therefore, the set of k values that can balance the original word can be established by both the encoder and the decoder *identically* and *implicitly*. Accordingly, the decoder can rebuild *TR*, which is identical to the one that has been built by the encoder, and it can infer the recycled codeword '01' that corresponds to the received k. This means that one of the recycled codewords of the available k's has been transmitted implicitly from the encoder to the decoder. The selection of the received k among {1,3,5,7} constitutes an *eye wink* from the encoder to the decoder. The bits that have been matched with the end of σ (deleted) 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 are one of the recycled codewords created at that step. If there is only one k to make the block *BL*, balanced then the encoder does not delete anything

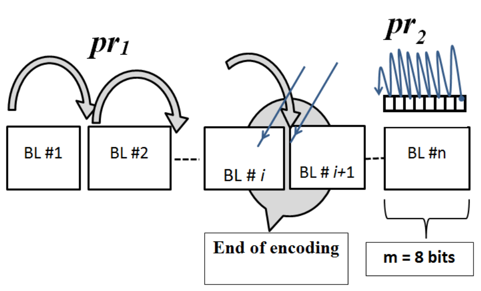

Fig. 2. The termination process.

from σ. The idea of *recycling* comes from the fact that bits that would have been wasted due to the multiplicity are turned into useful information. Based on this arrangement, the decoder can restore the bits that have been deleted by the encoder at the same location as we show below in the *BRKA* decoder section.

Before moving on to the decoder, let's briefly explain the termination process mentioned in step (5). The termination process keeps the whole stream balanced as follows. The encoding process ends when the *encoding* pointer, pr_1, which goes forward and the *recycling* pointer, pr_2, which goes backward, meet or have crossed each other as depicted in Fig. 2. The encoding pointer, pr_1, moves forward block by block, and pr_2 moves backward a few bits a time. Thus, at any time during the encoding process, σ can be viewed as follows. The part of σ to the left of pr_1 represents the already encoded data, the part to the left of pr_2 represents the already recycled data, and the part between pr_1 and pr_2 represents the unprocessed data. When pr_1 and pr_2 have met then the whole stream is balanced, but if they have crossed over, then the encoding process ends at a point where a *part* of the last block is transmitted; thus we can not confirm that this part is balanced. Accordingly, the termination process sends an integer number of blocks to guarantee that the whole transmitted stream is balanced. This is the main purpose of the termination process (the details are not described here to save space for more important issues). The termination process causes the encoder to describe at most T bits twice; once by encoding and a second time by recycling. The maximum value of T is

$$\lceil \log_2(m/2) \rceil - 1 + p + m. \tag{10}$$

Therefore, given that T does not depend on n, the termination process does not affect the performance of our scheme.

B. The BRKA decoder

To this point, we saw how the encoder works, and how the new constructed balanced codewords contain an implicit message (the eye wink) within each block that indicates the selected choice. The decoder at the other side undoes what the encoder did according to the implicit message in each block to recover the recycled bits. The decoder decodes the whole stream received from the encoder, σ', block by block, as follows:

1) Read the next block, $v\hat{w}$, and retrieve the original word w according to the first p bits (v).
In Fig. 1, the first 6 bits of the first encoded block,

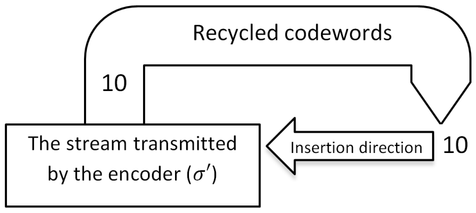

Fig. 3. The decoding process of *BRKA*

BL$'$, were '001101'. Realizing that '001101' is the prefix codeword for $k = 3$, then the original word $w =$'10101101' can be retrieved by complementing the first 3 bits of $\hat{w} =$'01001101'.

2) Inspect w for the full set of k's (balance points).
3) Generate the prefix code for the available choices (k's) by building the corresponding *TR*.
4) Insert the recycled codeword for the received k from right to left as shown in Fig. 3. If there is only one k then the decoder inserts nothing.
   Notice that by receiving a particular choice of k, the decoder deduces what are the bits that should be recycled.
5) If there remains unprocessed blocks in the stream then go to step (1), otherwise, make the necessary termination and stop.

The *BRKA* scheme described above uses the bit recycling based on Huffman coding (*HuBR*). Another variant of bit recycling based on arithmetic coding (*ACBR*) [9] can also be applied. *ACBR* can achieve better recycling than *HuBR*. We found that the improvement in the code rate efficiency by applying *ACBR* instead of *HuBR* on Knuth's redundancy is very small in the equiprobable case, which is the case of Knuth's algorithm. Therefore and for sake of space, the principle of *ACBR* [9] is not further considered here. Next, in the theoretical analysis, we briefly show the difference in the performance of *ACBR* and *HuBR*.

The main factors of the per-block computational complexity for Knuth's algorithm are reading m-bit block and finding a balance point in it. Our scheme introduces additional overhead due to the following operations: Determining all the available balance points in the block instead of determining only one balance point, building the Huffman tree for the set of balance points, and the recycling procedure. The computation complexity of reading m-bit block is the same for Knuth's algorithm and the *BRKA* scheme, which is $\Theta(m)$. The complexity of determining the balance point(s) is $\Theta(1)$ for Knuth's algorithm and $\Theta(m)$ for *BRKA*. Let c denote the number of available balance points in m-bit block, then the complexity of constructing the Huffman tree of c leafs is $\Theta(c)$, and that of the recycling procedure is $\Theta(\log_2 c)$. Given that $1 \le c \le m/2$, then the overall computation complexity of Knuth's algorithm and the *BRKA* scheme is $\Theta(m)$. Only the hidden constant changes.

IV. Theoretical analysis and results

In order to evaluate the performance of *BRKA* scheme, we first have to calculate the average number of bits that can be recycled per m-bit block. Let the original word, w, has i as a balance point. Let c denotes the number of the available balance points, then the *BRKA* encoder has the opportunity to recycle if $c \ge 2$. Accordingly, the *BRKA* encoder builds a Huffman tree of c leafs. A Huffman tree of c equiprobable leafs contains two codeword lengths, $k_1 = \lfloor \log_2 c \rfloor$ and $k_2 = \lceil \log_2 c \rceil$. Then there will be $c - 2d$ codewords of length k_1 and $2d$ codewords of length k_2, where d is given by:

$$d = c - 2^{\lfloor \log_2 c \rfloor} \tag{11}$$

The probability $p_i = (1/2^{k_j})$ is the probability of selecting the balance point i, i.e. it is the probability that σ ends with these exact k_j bits. Since there are two different lengths (two j's), then the average number of recycled bits, AV, for c number of choices equals

$$AV(c) = (c - 2d) \cdot k_1 \cdot \frac{1}{2^{k_1}} + 2d \cdot k_2 \cdot \frac{1}{2^{k_2}}. \tag{12}$$

Therefore,

$$AV(c) = (c-2d) \cdot \lfloor \log_2 c \rfloor \cdot \frac{1}{2^{\lfloor \log_2 c \rfloor}} + 2d \cdot \frac{1}{2^{\lceil \log_2 c \rceil}} \cdot \lceil \log_2 c \rceil . \tag{13}$$

We have already mentioned that the value of c for any random unbalanced m-bit word, m even, is $1 \le c \le m/2$. Therefore, the average amount of information, H, that can be recycled per block by applying *BRKA* is

$$H(m) = \sum_{c=1}^{m/2} P(c) \cdot AV(c), \tag{14}$$

where $P(c)$ denote the probability that w offers exactly c balance points. $P(c)$ has been computed by Weber and Immink in [2] as follows

$$P(c) = 2^{c+1-m} \binom{m-1-c}{m/2-c}, \quad 1 \le c \le m/2. \tag{15}$$

Now we can compute $H(m)$ for different values of m to evaluate the performance of *BRKA*. Accordingly, the value of $H(m)$ has been calculated for different values of m as shown in Table I in order to be able to evaluate the improvement added by *BRKA* with respect to Knuth's original algorithm and the minimum redundancy. One issue we have to take into account is the loss, $L(m)$, in $H(m)$ due to the deleted blocks, since the average amount of information that can be conveyed

TABLE I. Calculated $H(m)$, L, and N for different values of m.

m	H(m)	L	N
8	0.95	0.113	0.837
16	1.37	0.117	1.253
32	1.80	0.101	1.699
64	2.25	0.079	2.171
128	2.70	0.057	2.643
256	3.16	0.039	3.121
512	3.68	0.026	3.654
1024	4.10	0.016	4.084
2048	4.60	0.010	4.590
4096	5.10	0.006	5.094

by the available choices in the recycled blocks could not be exploited by the *BRKA* scheme. Therefore, we have to calculate $L(m)$ as follows. On average, *BRKA* recycles $H(m)$ bits per m-bit unbalanced block. The full stream σ contains n blocks, then the integer average number of recycled blocks, AV_B, in σ equals $\lfloor A \cdot H(m)/m \rfloor$, where A is the actual number of the blocks that can be exploited (encoded) including the termination block. The relation between n and A is:

$$n = A + \left\lfloor \frac{A \cdot H(m)}{m} \right\rfloor . \tag{16}$$

The total number of bits that can be recycled in σ is $A \cdot H$ bits. Accordingly, the loss, $L(m)$, in $H(m)$ per block due to the recycled (deleted) blocks equals

$$L(m) = \left(\frac{A \cdot (H(m))^2}{n \cdot m} \right) < \left(\frac{(H(m))^2}{m} \right) . \tag{17}$$

The corresponding value of $L(m)$ for each m are also indicated in Table I. It is clear that the average lost bits per block can be neglected, especially for large m, since is does not affect the code efficiency of $BRKA$. However, we will consider the average net number of recycled bits per block, N, in the remaining part of this section.

The ideal average amount of information, $I(m)$, that can be conveyed per block via the selection freedom in Knuth's algorithm has been computed and approximated for large m by Weber and Immink in [2] to

$$I(m) = \sum_{c=1}^{m/2} P(c) \cdot \log_2 m \approx \frac{1}{2} \log_2 m - 0.916. \tag{18}$$

The difference between (14) and (18) represents the difference between the performance of *HuBR* and *ACBR*, which can be observed by comparing the value of $H(m)$ achieved by *BRKA* and the value of $I(m)$ for various values of m. Now to put it all together, we calculated the absolute minimum number of redundant bits per m-bit block (lower bound), the number of redundant bits per m-bit block in Knuth's algorithm and in the *BRKA* scheme, and $I(m)$ for different values of m as shown in Table II. Finally, the numerical values of Table II are plotted in one graph for clear observations, in Fig. 4.

By looking at Fig. 4 we can come up with the following conclusions. First, the *BRKA* scheme achieves a significant reduction in Knuth's algorithm redundancy, i.e. we minimized the redundancy gap between Knuth's algorithm and the lower bound. Second, the performance of *BRKA* is very close to $I(m)$ value, which means that *BRKA* achieves a utilization, for the information that can be conveyed via the selection freedom per m-bit word, that is very close to the ideal case, as it has been computed in [2]. It is clear from Fig. 4 that the performance of *BRKA* is almost the same as the $I(m)$ value. The $I(m)$ value is depicted in Fig. 4 by the thin dashed line and the reader may observe how small the difference is. Notice that, *BRKA* achieved better performance than the $I(m)$ value for $m < 32$, because the value of $I(m)$ is approximated in (18) for large m, thus the $I(m)$ value for $m < 32$ does not represent the exact value. Third, it is clear that there is still some gap that needs to be closed. Using variable-length prefix codes might further reduce the remaining gap as it has been computed in [2]. Fourth, the results of the straightforward solution proposed by Weber and Immink in [3] are also depicted in Fig. 4. It is clear that our scheme is less redundant than that of Immink and Weber [3]. In the scheme by Immink and Weber, for $m \leq 64$ the redundancy is almost like Knuth's redundancy, therefore they recommended to switch to another solution for $m \leq 64$. Hence, our scheme achieved a significant reduction in the redundancy gap, and is less redundant than the scheme proposed by Immink and Weber [3] for all values of m.

TABLE II. THE COMPUTED VALUES OF THE MINIMUM REDUNDANCY, KNUTH'S REDUNDANCY WITH AND WITHOUT BIT RECYCLING, AND I VALUE.

m	Log *m*	Minimum redundancy	Knuth's redundancy	(BRKA) redundancy	*I (m)* Value
8	3	1.826	3.48	2.60	2.77
16	4	2.326	4.60	3.32	3.39
32	5	2.826	5.70	3.98	3.99
64	6	3.326	6.78	4.60	4.57
128	7	3.826	7.85	5.20	5.14
256	8	4.326	8.90	5.78	5.69
512	9	4.826	9.95	6.30	6.24
1024	10	5.326	11.00	6.92	6.79
2048	11	5.826	12.04	7.45	7.33
4096	12	6.326	13.08	7.99	7.87

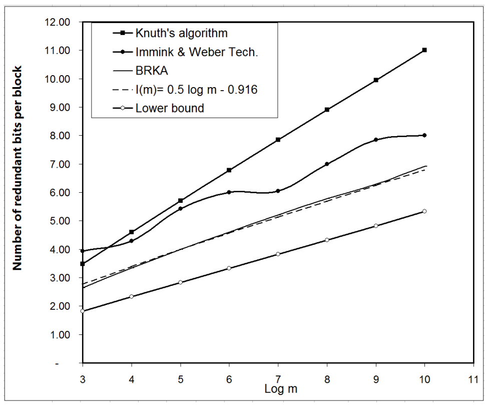

Fig. 4. Comparison between the performance of Knuth's algorithm with and without bit recycling, the Immink & Weber technique, and the I value.

V. Conclusion

A new simple scheme named Bit Recycling for Knuth's Algorithm (*BRKA*) has been proposed to minimize the redundancy caused by the multiplicity of encoding in Knuth's algorithm. *BRKA* scheme has achieved a significant reduction in the gap between Knuth's algorithm redundancy and the lower bound redundancy for all values of m (word length).

More work is needed to find a solution to the remaining small gap between the lower bound redundancy and *BRKA*'s redundancy. We believe that the remaining gap could be closed, since *BRKA* currently exploits the multiplicity of encodings only in m-bit word and not on the whole length of the generated $(m+p)$-bits balanced words.

VI. 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.

References

[1] D. E. Knuth, 'Efficient balanced codes', IEEE Trans. Inf. Theory, vol. IT-32, pp. 51-53, 1986.

[2] Weber, J.H and Immink, K.A.S., 'Knuth's balancing of codewords revisited', IEEE International Symposium on Information Theory (ISIT2008).

[3] Immink, K.A.S. and Weber, J.H, 'Very Efficient Balanced Codes', IEEE Journal on Selected Areas in Communications, vol. 28, no. 2, February 2010.

[4] D. Dubé and V. Beaudoin, 'Recycling bits in LZ77-based compression', In Proceedings of the (SETIT 2005), Sousse, Tunisia, March 2005.

[5] D. Dubé and V. Beaudoin, 'Improving LZ77 data compression using bit recycling', In Proc. of ISITA, Seoul, South Korea, October 2006.

[6] D. Dubé and V. Beaudoin, 'Bit recycling with prefix codes', In Proc. Of DCC, page 379, Snowbird, Utah, USA, March 2007.

[7] D. Dubé and V. Beaudoin, 'Improving LZ77 bit recycling using all matches', In Proc. of ISIT, Toronto, Canada, July 2008.

[8] D. Dubé and V. Beaudoin, 'Constructing Optimal Whole-Bit Recycling Codes', In Proc. of IEEE Information Theory Workshop, Greece, 2009.

[9] A. Al-Rababa'a and D. Dubé, 'Adaptation of Bit Recycling to Arithmetic Coding', has been accepted to be published in 'The 9th International Workshop on Systems, Signal Processing, and their Application's', Algeria, May 2013.