FRACTIONAL-BIT AND VALUE-LOCATION LOSSLESS ENCODING IN G.711.0 CODER

Jacek Stachurski and Lorin Netsch

Texas Instruments, DSP Solutions R&D Center, Dallas TX

ABSTRACT

The paper describes two lossless coding tools employed in the new ITU-T G.711.0 Recommendation: fractional-bit and value-location encoding. Instead of encoding each sample individually as done in G.711, the fractional-bit coding tool identifies the total number of signal levels that exist within an input frame and then combines several samples for joint encoding with fractional-bits per sample. The value-location tool encodes positions of all values within an input frame that differ from a reference value. The method efficiently represents an input frame as a sum of value-location code vectors that are sequentially encoded using Rice, binary, or explicit location encoding. Presented results illustrate how the described coding techniques were adopted for usage within the new ITU-T G.711.0 standard.

Index Terms — Speech Coding, Lossless Compression, ITU-T Standardization, G.711, G.711.0

1. INTRODUCTION

In many signal processing applications, a digitized signal must be transmitted or stored such that the exact original may be recovered. As part of the transmission and/or storage process, it is desirable to reduce the number of bits needed to represent the signal in order to maximize the amount of data that can be handled. Lossless compression provides means for achieving this goal.

Many techniques exist for lossless compression [1] including Huffman coding [2], run-length coding [3], and predictive coding [4]. Each encoding method may provide better compression for certain classes of signals. In a system without complexity limitations, verifying several combinations of available compression methods and selecting the best performing set (“closed loop” approach) may be possible providing that bits are available to transmit side information about the methods selected. When complexity limits are imposed, it may be necessary to limit the number of encoding tools, and select the compression methods based only on input signal characteristics without guarantee that the selection indeed provides best compression (“open loop” approach).

We describe two coding techniques that were found useful in the context of lossless compression of the ITU-T G.711 [5] bitstream. The presented methods were validated to provide extra data compression in addition to that achieved by other coding tools and were incorporated into the ITU-T G.711.0 Recommendation [6]. They work alongside a series of other lossless encoding tools: constant-value coding, plus-minus zero Rice coding, binary coding, pulse mode, min-max level coding, mapped-domain and direct linear predictive coding [6].

A comprehensive analysis of the G.711-encoded data was performed to identify how the proposed methods may contribute best in the G.711.0 context. While providing good results by themselves, the challenge was to provide an additional compression when combined with other encoding tools with little or no complexity increase. As a result, the fractional-bit (FB) coding is selected in G.711.0 based on direct comparison of its performance to that achieved by the other coding tools (“closed-loop” approach) while the value-location (VL) coding is selected based on the input signal characteristics (“open-loop”) which results in an overall computational complexity decrease [6]. Both methods are applied to a set of identified G.711 bit-stream cases in its 8-bit-sample domain.

The following sections describe the fractional-bit and value-location encoding tools and their application in G.711.0.

2. FRACTIONAL-BIT PER SAMPLE ENCODING

The number of bits required to encode a data sample taking on one of \( L \) possible data values equals \( b_L = \log_2(L) \). In most cases, the number \( b_L \) is not an integer. In a practical system, an integer number of bits must be transmitted so, for each data sample, \( b_L \) is typically rounded up to the nearest integer \( B_L = \lceil b_L \rceil \). For uniformly distributed input with \( N \) samples encoded per frame vector \( s \), the minimum number of bits required to represent the full data frame is equal to \( b_{NL} = \lceil N b_L \rceil \). The number of bits used, however, may be larger than required since \( N B_L > b_{NL} \).

In fractional-bit (FB) per sample encoding, the overall number of bits used per frame is reduced by combining several samples into blocks for joint encoding. The proposed approach is capable of achieving bit-rate reduction with respect to the \( N B_L \) bits-per-frame reference with low computational complexity. Low complexity is maintained at the expense of a possible small bit-rate increase with respect to the \( b_{NL} \) minimum bits required.

A series of \( N \) samples, \( l_0 \) to \( l_{N-1} \), each taking on a value from 0 to \( L-1 \), may be encoded by a bit-stream corresponding to the polynomial

\[
V = l_0 + l_1 L + l_2 L^2 + \ldots + l_{N-1} L^{N-1}.
\]  

(1)

For a large \( N \), however, it may be unacceptable to calculate the polynomial \( V \) due to its large size. Therefore, we propose a block-adaptive approach with a constraint \( B_L \) specifying the maximum number of encoding bits per block (number of bits required to represent \( V \)). For a convenient fixed-point implementation, the block-size constraint \( B_L \) may be set, for example, to 15 (number of bits that represent a positive value in a
16-bit signed integer). We calculate the polynomial $V$ for $M$ samples at a time, with the frame-by-frame adaptive data-block size $M$ set such that the number of bits required to represent the data block is $M$ bits. Note that even with the block-size constraint $B_k$ being fixed, the data block size $M$ may be adaptive based on the frame-by-frame data value range $L$. For each data block $k$ from 0 to $K - 1$, with $K = \left\lfloor N/M \right\rfloor$ being the number of $M$-size blocks within a data frame, we calculate the block-stream polynomial

$$V_k = l_{KM} + l_{KM-1}L + \ldots + l_{K-M+1}L^{M-1}. \quad (2)$$

The calculation of $V_k$ is repeated (and its value encoded in the output bit-stream) until the end of an input frame is reached. Each $V_k$ represents $M$ data samples and is encoded with $B_{ML} = \left\lceil MB_{k1} \right\rceil$ bits. The remaining $m = N - KM$ data samples in a frame (if any left) would be encoded using

$$V_k = l_{KM} + l_{KM-1}L + \ldots + l_{K-m-1}L^{m-1}. \quad (3)$$

The total number of bits required to encode $N$ frame samples with the proposed approach is

$$B_H = K \left\lceil MB_{k1} \right\rceil + \left\lceil mb_{k2} \right\rceil. \quad (4)$$

Assuming that $N$, $L$, and $M$ are known to the decoder, the proposed method does not require transmission of extra information in addition to the bits representing the encoded data. The computational complexity increase with the proposed method is on the order of $N$ additional multiplications and additions (to calculate $V_k$ for each block) or about 0.01 WMOPS (Weighted Millions of Operations per Second) at 8 kHz sampling rate.

Note that when $B_k = b_{k1}$ or $M = 1$, the proposed method does not provide any bit-rate reduction. The number of bits per frame to be used is bound by $NB_{k1} \geq B_k \geq b_{k1} = \left\lceil Nb_{k1} \right\rceil$; our fractional-bit method is potentially most effective when $B_k > b_{k1}$ for $M = 1$. In general, $B_k$ should be minimized for $M = N$ but this requires calculation of a possibly very large polynomial $V$ that would exceed the specified bit-length constraint $B_k$. The rounding of $B_k$ to an integer number of bits introduces a coding "inefficiency" $\varepsilon$ which is largest when the rounding occurs for each data sample, $\varepsilon_{\text{max}} = \varepsilon_1 = N \left\lceil \frac{b_{k1}}{2} \right\rceil - b_{k1}$, and smallest when performed only once per frame $\varepsilon_{\text{max}} = \varepsilon_N = \left\lceil Nb_{k1} \right\rceil - Nb_{k1}$ (as done for the $M = N$ case). We have $\varepsilon_1 \geq \varepsilon_M \geq \varepsilon_N$ with

$$\varepsilon_M = K \left\lceil MB_{k1} \right\rceil - Mb_{k1} + \left\lceil mb_{k2} \right\rceil - mb_{k2}. \quad (5)$$

The data-block size $M$ may be adapted based on the frame size $N$ and the data range $L$, or solely based on $L$. In general, $M$ should be chosen so that the number of bits per frame $B_k$ is minimized while the polynomial $V_k$ contains a sufficiently small number of elements to comply with the constraint $B_k$. In addition, since each data frame may be encoded with an integer number of bytes, minimizing $\left\lceil B_k / 8 \right\rceil$ instead of minimizing $B_k$ may be the key criterion. If the final number of bits per frame $B_k$ does not correspond to an integer number of bytes, the remaining bits may be used, for example, to encode the data range $L$. Generally, good candidates for $M$ result in $B_{ML} = \left\lceil MB_{k1} \right\rceil$ being only slightly less than an integer so that the coding inefficiency, $\varepsilon = B_{ML} - Mb_{k1}$, is minimized.

3. VALUE-LOCATION ENCODING

In value-location (VL) coding, we decompose a data frame $s$ into

$$s = v_0 + \sum_{k=1}^{K-1} (v_k - v_0) c_k \quad (6)$$

where $v_0$ is a chosen reference value and the code vectors $c_k$ represent the locations of all other values $v_k$ within the signal $s$. A code vector $c_k$ contains 1 at the locations at which the value $v_k$ occurs, and 0 elsewhere. The number of occurrences of each $v_k$ value, $N_k$, equals the number of non-zero elements in the corresponding $c_k$ vector:

$$N_k = \sum_{i=0}^{K-1} c_k(i), \quad \text{with} \quad N_0 = N - \sum_{k=1}^{K-1} N_k. \quad (7)$$

To represent a data frame $s$, it is sufficient to specify the number of signal values $L$, the reference value $v_0$, values $v_k$, and the corresponding locations in $c_k$.

3.1 Encoding values $v_0$ and $v_k$

The mapping of $v_0$ and $v_k$ to the values occurring in a frame can be done in several ways. Since this mapping defines the order of encoding the code vectors $c_k$ as explained in Section 3.2, it should be chosen to achieve an overall efficient compression.

One method to encode $v_0$ and $v_k$ is to explicitly encode each of the $L$ values in the signal based on the desired order of $c_k$ encoding. If $R_k$ bits are required to encode one value, then this method would require $LB_k$ bits to encode all $v_0$ and $v_k$. Alternatively, one may choose to encode $v_0$ and the $v_k - v_0$ offsets. The number of bits used to code $v_k$ could be further reduced by specifying a pre-determined coding order, for example from the minimum to maximum $v_k$, with only $v_0$ encoded.

3.2 Encoding code vectors $c_k$

To encode the code vectors $c_k$, we take advantage of the fact that they contain non-zero elements in unique locations as follows:

$$c_k(n) = 1 \quad \text{if} \quad c_k(n) = 0 \quad \text{for} \quad l \neq k, \quad n = 0, \ldots, N-1.$$

Using this property, once a given vector $c_k$ is encoded, the elements corresponding to the $N_k$ non-zero $c_k$ locations need not be considered when encoding all subsequent vectors $c_k$, $l > k$.

Let $z_k$ denote a vector obtained from the code vector $c_k$ by removing elements corresponding to all non-zero-value locations in vectors $c_1, \ldots, c_{l-1}$ (or, equivalently, by preserving only elements corresponding to zero-value locations). The $D_k$ dimension (length) of a vector $z_k$ equals

$$D_k = N - \sum_{i=1}^{k-1} N_i. \quad (8)$$

Let vectors $\delta_k$ specify the value locations in vectors $c_1, \ldots, c_k$,
\[
\delta_k(n) = \sum_{i=1}^{k} c_i(n).
\]

With \( z_i = c_i \), the code vectors \( z_k \) can be computed as

set \( m = 1 \)

for each \( n = 1, \ldots, N \)

if \( \delta_k(n) = 0 \)

then \( z_k(m) = c_i(n) \) and \( m = m + 1 \)

Our proposed method sequentially encodes the vectors \( z_k \).

<table>
<thead>
<tr>
<th>Encoded vector</th>
<th>Number of elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z_1 )</td>
<td>( N )</td>
<td>equivalent to ( c_1 )</td>
</tr>
<tr>
<td>( z_2 )</td>
<td>( N - N_k )</td>
<td>( c_2 ) with elements corresponding to non-zero locations in ( c_1 ) removed</td>
</tr>
<tr>
<td>( z_{L-1} )</td>
<td>( N - \sum_{i=1}^{L-2} N_i )</td>
<td>( c_{L-1} ) with elements corresponding to non-zero locations in ( c_1, \ldots, c_{L-2} ) removed</td>
</tr>
</tbody>
</table>

Note that, as elements are removed from \( c_1 \), vectors \( z_k \) become shorter with each processing step \( k \) which helps to improve compression. While the total number of elements in the initial sequence \( c_1, \ldots, c_{L-1} \) equals \( N(L-1) \), the number of elements in the sequence \( z_1, \ldots, z_{L-1} \) is reduced to

\[
\sum_{i=1}^{L-1} (N - \sum_{j=i}^{L-1} N_j) = N(L-1) - \sum_{i=1}^{L-2} \sum_{j=i}^{L-1} N_j,
\]

i.e., reduced by

\[
\sum_{i=1}^{L-2} \sum_{j=i}^{L-1} N_j = \sum_{i=1}^{L-2} N_i(L-1-k)
\]

which is maximized when \( N_1 \geq N_2 \geq \ldots \geq N_{L-1} \).

As stated in Section 3.1, the order of encoding vectors \( c_k \) is determined by the mapping of the \( L-1 \) signal values to \( v_k \). Encoding first a \( z_k \) vector that has a larger number of non-zero elements \( N_k \) will reduce the number of elements in subsequent \( z_l, l > k \), but may require a larger number of bits to encode these non-zero \( z_k \) locations. On the other hand, encoding a \( z_k \) vector with smaller number of non-zero elements first may require fewer bits, but it will not remove as many elements in subsequent \( z_l \). One obvious method to maximize compression is to calculate the compression obtained for each possible ordering of the \( L-1 \) values \( v_k \) and the corresponding vectors \( z_k \), and choose the ordering that provides the best result. However, this requires computational resources that may not be available and would also require explicit encoding of \( v_k \). Another method is to choose to encode the \( L-1 \) code vectors in a specific pre-determined order known to the decoder. While possibly not providing best compression, this lower-complexity method may be preferred as it would not require explicit \( v_k \) encoding.

The proposed signal decomposition results in code vectors \( z_k \) that contain a series of zeros and ones. A number of methods exist for efficient encoding of such vectors, for example run-length coding [3]. The method or methods used to encode \( z_k \) may depend on a particular application, resources available, signal characteristics, and performance required.

4. APPLICATION TO G.711.0 LOSSLESS CODER

The two presented lossless encoding methods, FB and VL, were adapted for use in the ITU-T G.711.0 standard [6]. The FB coding tool identifies the total number of signal levels that exist within an input frame and combines \( M = 5 \) samples at a time to compute the polynomial \( V \) as described in Section 2, resulting in an integer number of blocks per frame. The VL coding tool sequentially encodes positions of values within an input frame that differ from the reference \( v_0 \) value which is set to zero. The \( v_k \) value assignment alternates between above and below \( v_0 \) starting with the larger number of value occurrences; a single bit indicating whether \( v_1 \) corresponds to above or below \( v_0 \) is encoded in the bit-stream. The reduced-dimensionality vectors \( z_k \) are coded using run-length Rice coding [1], binary encoding, or explicit location encoding. The information about the encoding method for each \( z_k \) is included in the bit-stream.

In the process of the G.711.0 development, a Figure of Merit (FoM) was used to determine which coding technologies provide the desirable tradeoff between signal compression ratio and computational complexity [7]. While compression ratio was given a priority, the FoM was decreased as computational complexity increased. Given a set of encoding tools, a new tool would only be added if it would result in an FoM increase. On one hand, a new proposed method could be included “in parallel” with existing tools (i.e., a set of tools tested with the best one selected – “closed loop” decision) but then the additional achieved compression would need to outweigh the resultant complexity increase. Another approach would be to replace existing tools with the new proposed method to increase compression and/or reduce complexity (“open loop” decision). As explained below, the lower-complexity (about 0.01 WMOPS) FB method was included in G.711.0 for selected signal cases “closed loop”; it is selected only when performing better than the other available encoding tools. The higher-complexity (about 0.30 WMOPS) VL method, on the other hand, was included “open loop”; it is always the selected tool for the identified signal cases (other tools are then bypassed). This design helped maintain the low-complexity feature of the G.711.0 codec (1.0 WMOPS average, <1.7 WMOPS worst case).

For both FB and VL, analysis of G.711-encoded data served as basis for deciding how they are to be applied. As specified in the ITU-T selection phase processing plan [8], the P.301 speech corpus [9] was used for the compression ratio tests and complexity analysis. The set of processed conditions included ten languages and dialects at -16, -26 and -26 dBov levels for clear speech, at -26 dBov level for five different background-noise types (at 15, 20 and 25 dB SNR), and tandem with four other speech codecs, adding up to the total of 137 hours of G.711-encoded data. In addition, a 1.4 GB G.711 µ-law corpus recorded from an in-service network operated in Japan (including FAX and DTMF signals) was provided by NTT and included in the tests. The cumulative results provided in here were obtained from this extensive data set.
Several signal cases were identified where the proposed methods provide most benefit in addition to the other G.711.0 encoding tools. The set of these cases was limited to fit the information about the selected method and signal case into the G.711.0 prefix bytes which define the input frame length (40, 80, 160, 240, or 320 samples) and the coding tool used [6]. Table 1 summarizes the 40-sample frame cases for which the FB encoding is considered. The first two columns list all values that may occur within a considered frame and the percentage of how often a particular case occurred in the full test set; the listed cases add up to 13.9% of all input frames. The next two columns give percentages of how often the FB encoding performed better or worse than all other G.711.0 tools for the individual cases. The final two columns, provide the FB compression ratio improvement when FB is selected always (“open loop” selection) or selected only when it is better than all other tools (“closed loop” selection) in each individual signal case. With the “open loop” selection, some of the listed cases would result in a compression ratio drop; based on these results, “closed loop” selection was therefore chosen. (Note that the “open loop” selection could also be beneficial for a reduced number of cases.) Overall, in the frames where it is selected, FB encoding provides on average about 77% compression ratio with respect to the original G.711 bit-stream. As they are used in about 14% and 8% of the frames, this translates to about 11% and 7% compression-ratio contributions to the over 50% compression ratio achieved by G.711.0. The proposed methods may also be applied to provide performance improvement in the context of other lossless compression algorithms.

### Table 1: Fractional-bit (FB) encoding for 40-sample frame

<table>
<thead>
<tr>
<th>Levels</th>
<th>Total occurrence (%)</th>
<th>FB Better (%)</th>
<th>FB Worse (%)</th>
<th>Apply FB %</th>
<th>Always</th>
<th>Better</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0,1}</td>
<td>0.96</td>
<td>90.12</td>
<td>9.88</td>
<td>1.76</td>
<td>2.25</td>
<td></td>
</tr>
<tr>
<td>{-2,0}</td>
<td>1.12</td>
<td>96.81</td>
<td>3.19</td>
<td>8.99</td>
<td>9.15</td>
<td></td>
</tr>
<tr>
<td>{-2,-1,0}</td>
<td>0.56</td>
<td>68.61</td>
<td>11.98</td>
<td>2.18</td>
<td>2.65</td>
<td></td>
</tr>
<tr>
<td>{-1,0,1}</td>
<td>2.99</td>
<td>23.21</td>
<td>44.76</td>
<td>-1.05</td>
<td>0.74</td>
<td></td>
</tr>
<tr>
<td>{-2,0,1}</td>
<td>1.44</td>
<td>85.53</td>
<td>3.28</td>
<td>4.78</td>
<td>4.87</td>
<td></td>
</tr>
<tr>
<td>{-2,-1,0,1}</td>
<td>3.81</td>
<td>29.43</td>
<td>39.67</td>
<td>-0.60</td>
<td>0.74</td>
<td></td>
</tr>
<tr>
<td>{-3,2,0,1}</td>
<td>0.43</td>
<td>73.17</td>
<td>12.45</td>
<td>4.06</td>
<td>4.53</td>
<td></td>
</tr>
<tr>
<td>{-2,0,1,2}</td>
<td>0.22</td>
<td>76.70</td>
<td>13.16</td>
<td>5.37</td>
<td>5.97</td>
<td></td>
</tr>
<tr>
<td>{-2,-1,0,1,2}</td>
<td>0.98</td>
<td>13.02</td>
<td>72.92</td>
<td>-3.44</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td>{-3,2,0,1,2}</td>
<td>0.21</td>
<td>59.07</td>
<td>26.91</td>
<td>1.63</td>
<td>3.18</td>
<td></td>
</tr>
<tr>
<td>{-3,-2,0,1,2}</td>
<td>1.01</td>
<td>19.34</td>
<td>64.64</td>
<td>-2.98</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>{-4,-3,2,0,1,2}</td>
<td>0.16</td>
<td>69.81</td>
<td>21.01</td>
<td>2.05</td>
<td>3.33</td>
<td></td>
</tr>
</tbody>
</table>

In addition to the 40 and 320-sample frames, both FB and VL tools are used in G.711.0 for all other supported frame lengths although the selected signal cases differ between frame sizes [6]. In general, the FB method is relatively more effective for shorter frames while VL is more effective for longer frames.

### Table 5. CONCLUSIONS

We described two lossless coding techniques that proved useful in the G.711.0 standard. Based on analysis of G.711-encoded data, the presented FB and VL encoding methods were adapted to complement a set of other G.711.0 encoding tools for selected signal cases. For the presented 40-samples and 320-sample-frame use cases, the FB and VL encoding methods provide, respectively, on average about 77% and 86% compression ratios with respect to the original G.711 bit-stream. As they are used in about 14% and 8% of the frames, this translates to about 11% and 7% compression-ratio contributions to the over 50% compression ratio achieved by G.711.0. The proposed methods may also be applied to provide performance improvement in the context of other lossless compression algorithms.

### Table 2: Value-level (VL) encoding for 320-sample frame

<table>
<thead>
<tr>
<th>Levels</th>
<th>Total occurrence (%)</th>
<th>VL Better (%)</th>
<th>VL Worse (%)</th>
<th>Apply VL %</th>
<th>Always</th>
<th>Better</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0,1}</td>
<td>0.35</td>
<td>99.86</td>
<td>0.07</td>
<td>5.48</td>
<td>5.48</td>
<td></td>
</tr>
<tr>
<td>{-1,0,1}</td>
<td>2.83</td>
<td>85.85</td>
<td>13.55</td>
<td>4.58</td>
<td>4.64</td>
<td></td>
</tr>
<tr>
<td>{-2,0}</td>
<td>0.74</td>
<td>93.42</td>
<td>6.52</td>
<td>4.06</td>
<td>4.11</td>
<td></td>
</tr>
<tr>
<td>{-2,-1,0,1}</td>
<td>3.95</td>
<td>60.95</td>
<td>34.24</td>
<td>1.41</td>
<td>1.83</td>
<td></td>
</tr>
</tbody>
</table>

### 6. ACKNOWLEDGEMENTS

The authors would like to thank Rongqiang Hu as well as the G.711.0 consortium members for their kind support and help.

### 7. REFERENCES