Enumerative Algebraic Coding for ACELP

Tom Bäckström

International Audio Labs Erlangen, Fraunhofer IIS, Erlangen, Germany
tom.baekstroem@iis.fraunhofer.de

Abstract

Speech coding algorithms based on Algebraic Code Excited Linear Prediction (ACELP) quantize and code the residual signal with a fixed algebraic codebook. This paper generalizes the conventional encoding to provide optimal bit consumption for all codebook designs by enumerating all possible states. The codebook designed is aimed for constant bit-rate applications applied on mobile devices.

1. Introduction

Recent mainstream speech coding standards employ at their core a technology known as Algebraic Code Excited Linear Prediction (ACELP). It is based on modelling the speech signal by a linear predictor, cascaded with a long time predictor (known as the LTP or adaptive codebook) and excited by a residual. The name for the method comes from the encoding of the residual, which employs an algebraic codebook (also known as the fixed or pulse codebook) [1]. This approach is included in major standards such as AMR-WB, G.718 and MPEG USAC [2–4].

The advantage of ACELP over its predecessor Code Excited Linear Prediction (CELP), is that it describes the residual signal algebraically, instead of a vector codebook [1, 5]. The benefit therein is that the codebook does not need to be stored, since it is algorithmically generated and even more importantly, searching for the best matching codebook entry is much more efficient than in vector quantization. Due to these benefits, ACELP is the cornerstone of recent standards [2–4], although it was invented already a quarter of a century ago.

To be precise, the algebraic codebook consists actually of two steps, quantization and coding. Firstly, the residual signal is quantized very roughly such that only a small number of non-zero samples remain. Secondly, the pulse positions are encoded to form the algebraic code [1]. To scale the quantized signal to match the energy of the original signal an additional gain factor is applied.

To allow faster codebook searches and for a more efficient bit-allocation, usually some additional constraints are introduced. Namely, the residual signal of a sub-frame is generally divided into interlaced tracks, such that each track has a fixed number of pulses. For example, a common configuration, for a sub-frame of length 64, is that every fourth sample belongs to the same track and that each track has exactly two pulses. It can be readily shown that if the residual is white noise, then this division to tracks does not significantly reduce the descriptive power of the codebook, while saving bits.

The current contribution addresses the encoding of the pulse positions. While existing methods encode pulse positions efficiently when the number of pulses is small, at higher bit-rates, when the number of pulses increases, available encoding strategies are either inoptimal in terms of bit-consumption or unable to provide a constant bit-rate.

The proposed approach is based on enumerating all possible, unique combinations of pulse constellations. This approach is optimal in the sense that with a given codebook design, it gives the smallest possible bit-consumption. The proposed approach is also a constant bit-rate approach in the sense that a given codebook design is always coded with the same, predetermined amount of bits. In addition, the computational complexity of the proposed method is negligible in comparison to the residual quantization (i.e. the pulse search) and is thus well suited for speech coding technology applied on e.g. mobile devices.

2. Codebook Structure

The residual signal in ACELP codecs is generally represented by a vector with a fixed number of signed pulses. The pulses have unit-length, constrained such that overlapping pulses have equal sign and the energy of the whole vector is adjusted by a scalar gain. Generally, the number of pulses is uniquely determined by the bit-rate and there is thus no need to signal it separately. Coding of the gain factor is a separate topic and will not be further discussed in this work.

The conventional approach for coding such pulses can be described as follows [2, 6]: A vector of length $N$, with a single signed pulse can clearly be coded by $\log_2 N$ bits for the position and one bit for the sign. A vector with two signed pulses, however, provides a more interesting case. Encoding both pulses separately, with $\log_2 N$ bits for the position and one bit for the sign, does not take advantage of the fact that the order of pulses is information
which is not necessary, nor from the fact that overlapping pulses must have the same sign.

The ordering of pulses can therefore be used as a way to encode the sign of the second pulse. Specifically, if the first pulse is encoded by its position \( n_1 \) and sign \( s_1 \), then the second pulse can be encoded by its position \( n_2 \) only. The sign of the second pulse can then be determined such that if \( n_2 < n_1 \) then \( s_2 = -s_1 \) and if \( n_2 \geq n_1 \) then \( s_2 = s_1 \). The bit-consumption for two pulses is then \( 2(\log_2 N) + 1 \), or, one bit less than when encoding the pulses separately.

When the number of pulses is 3, we immediately realize that if we split the range of positions to two partitions, then one of the partitions must contain at least two pulses. This partition can be encoded with the two-pulse strategy described above, while the remaining pulse has to be encoded on the full range. The bit-consumption of this approach is \( 3(\log_2 N) + 1 \) bits assuming \( N \) is even.

Similar approaches can be applied to tracks with 4, 5 or 6 pulses, whereby the number of required bits is, respectively, \( 4(\log_2 N) \) bits, \( 5(\log_2 N) \) bits or \( 6(\log_2 N) - 2 \) bits.

3. State enumeration

Let us describe the proposed approach through examples. The approach is then formalized to an algorithm in the following section.

The process of enumerating states necessarily consists of two steps. Firstly, we must determine the number of possible states, or equivalently, the number of required bits for coding. Secondly, we can choose the enumeration strategy to use. The number of possible states is unambiguously determined by the codebook design, while the enumeration of states can to some extent be freely chosen as long as the strategy is deterministic and provides a one-to-one mapping between codebook entries and the signed pulse constellations.

For a single sample with \( p \) pulses, it is easy to specify the possible states. Namely, all pulses at the same position must have the same sign, whereby if \( p > 0 \), then there are two possible states corresponding to the two possible signs. With \( p = 0 \) obviously there are no degrees of freedom and only a single state is possible.

A vector of length \( N = 2 \) is already more interesting as it reveals the basic idea of the presented approach. Given that we have, for example, \( p = 2 \) pulses, the 8 possible states are

| Position 1 | +2 | -2 | +1 | -1 | -1 | +1 | 0 | 0 |
| Position 2 | 0 | 0 | +1 | +1 | -1 | -1 | +2 | -2 |

or equivalently

| Position 1 | 2 pulses | 1 pulse | 0 pulses |
| 2 states   | 2 states | 1 state  |
| Position 2 | 0 pulses  | 1 pulse  | 2 pulses |
| 1 state    | 2 states  | 2 states |
| Total      | 2 states  | 4 states | 2 states |

We find that the total number of possible states is 18. In this manner we can extend the vector iteratively to obtain the number of states for any length \( N \) and number of pulses \( p \).

The proposed enumeration strategy follows a similar approach as that described above. Suppose we have a vector of length \( N = 1 \) to encode with \( p \) pulses. We can choose to encode this vector by setting the state to 1 if the pulses are negative and otherwise to zero. For \( p = 2 \) pulses on a length \( N = 2 \) vector we can choose

| Position 1 | +2 | -2 | +1 | -1 | -1 | +1 | 0 | 0 |
| State no   | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 |

For \( N = 3 \) we obtain

| Positions   | 2 pulses | 1 pulse | 0 pulses |
| 1 and 2     | 8 states | 4 states | 1 state  |
| Position 3  | 0 pulses  | 1 pulse  | 2 pulses |
| 1 state     | 2 states  | 2 states |
| State no    | 0...7    | 8...11   | 12...15  |

Again, this approach can be readily extended to vectors of any length and any number of pulses. In the following section, this general idea is formalized to an enumeration algorithm.

4. Derivation of the Algorithm

Consider a vector of length \( N \), partitioned to two parts of length \( N_1 \) and \( N_2 \) such that \( N = N_1 + N_2 \). If the complete vector has \( p \) pulses and the first partition has \( p_1 \) pulses, then the second partition has \( p_2 = p - p_1 \) pulses.

Let \( f(p, N) \) denote the number of states that a vector of length \( N \) with \( p \) pulses can attain. It follows that using the two partitions, we can write \( f(p, N) \) as a function of the number of states in each partition (for any \( p \geq 0, N \geq 0, N_1 \in [0, N] \)):

\[
f(p, N) = \sum_{k=0}^{p} f(k, N_1) f(p - k, N - N_1). \tag{1}
\]

This formula corresponds to the tables in Section 3, where the number of states in a column were multiplied, and the number of states per column were summed.
The trivial case \( N = 1 \) clearly has
\[
f(p, 1) = \begin{cases} 
1 & \text{for } p = 0 \\
2 & \text{for } p > 0
\end{cases}
\]  
(2)
since a vector without pulses \( (p = 0) \) has only a single state and in all other cases we encode the sign only (as there is only a single position).

The formulas 1 and 2 can be recursively used to form the number of states for vectors of any length and with any number of pulses. Note that in a practical application it can be computationally too complex to evaluate \( f(p, N) \) in (1), whereby a look-up table is much more efficient.

Let us then consider the state of a particular vector \( x = [x_1 \ x_2] \), which has \( \pi_1 \) pulses in partition \( x_1 \) and \( \pi_2 = p - \pi_1 \) in partition \( x_2 \).

Assume that the states \( s(x_1) \) and \( s(x_2) \) of the two partitions are known. Since they are independent, we can jointly encode these states by \( s(x_1) + f(p_1, N_1) s(x_2) \), which is the state of a vector where the number of pulses for each partition is fixed to \( \pi_1 \) and \( \pi_2 \).

The number of states that have less pulses than \( \pi_1 \) in partition \( x_1 \) is
\[
s(x_1, x_2) = \sum_{p=0}^{\pi_1 - 1} f(p_1, N_1) f(p - p_1, N - N_1). \quad (3)
\]
In other words, we can make the choice that the state of vector \( x \) has \( s(x) \geq s(x_1, x_2) \).

The overall state \( s(x) \) of the vector \( x \) is then
\[
s(x) = s(x_1, x_2) + s(x_1) + f(p_1, N_1) s(x_2). \quad (4)
\]
Similarly to the number of states, also here we need a seed for the recursion. We can choose to set the state of a unit length vector as
\[
s(x) = \begin{cases} 
0, & \text{for } x \geq 0 \\
1, & \text{for } x < 0
\end{cases}.
\]  
(5)
The state of any vector \( x \) can then be determined recursively using (3)–(5). The state can be decode by increasing \( \pi_1 \) as long as \( s(x) \geq s(x_1, x_2) \). When this limit is reached, partition states can readily be solved from (4).

Note that this enumeration is indeed arbitrary and it is not unique. For example, the positive and negative signs were chosen to be associated to 0 and 1, respectively, where we could have chosen the reverse instead. Still, observe that the final state is independent of how we choose to partition the vector, as long as the order of samples is retained. The choice of the partitions is important only in the sense that it can influence the computational complexity.

The two most obvious partition rules are

1. the incremental approach: \( N_1 = 1 \) and \( N_2 = N - 1 \), whereby \( f(\pi, N_1) \) can always be calculated from (2) and \( s(x_1) \) from (5).

2. the divide and conquer approach: \( N_1 = N_2 = N/2 \) (assuming \( N \) is a power of two), which means that each vector is split into two.

Note that the interlaced track concept can be interpreted as a combination of a permutation and partitioning rule, with the addition that the number of pulses per track is fixed. In particular, when the number of pulses per track is fixed, then there are no states which have a different number of pulses, \( s(x_1, x_2) = 0 \) if \( x_1 \) and \( x_2 \) are two of the tracks. Joint encoding of tracks is discussed in more detail in Section 5.

5. Encoding of Track Structure

Encoding tracks with the algorithm presented above does not always give a state whose range is a power of two. For example, a track with \( N = 16 \) samples and \( p = 3 \) pulses has 5472 possible states. To encode this state would require \( \log_2(5472) \approx 12.42 \) bits, which is not realizable in digital systems. By using 13 bits all states can be represented but with an 0.58 bit overhead per track.

To reduce the overhead, it is possible to encode tracks jointly. Assuming there are four tracks of length \( N_k \) and with \( p_k \) pulses on track \( k \), the number of states jointly represented by the four tracks is the product
\[
f = \prod_{k=1}^{4} f(p_k, N_k). \quad (6)
\]
We can also readily see that the joint state can be defined as
\[
s = s(x_1) + f(p_1, N_1) \left[ s(x_2) + f(p_2, N_2) \left[ s(x_3) + f(p_3, N_3) s(x_4) \right] \right]. \quad (7)
\]
where \( s(x_k) \) is the state of track \( k \).

The individual track states can be recovered by successively dividing by each \( f(p_k, N_k) \) whereby the remainder is \( s(x_k) \) for respective division.

Instead of (7), the algorithm for encoding individual tracks can be generalized to encode jointly all tracks. Namely, observe that the interlaced tracks structure can be modified by a permutation such that the samples of each track form partitions of a vector, for example, as \( x = [x_1 \ x_2 \ x_3 \ x_4] \). Since the number of pulses in a track is usually constrained to a fixed number, the function \( f(p_k, N_k) \) for the number of available states has to modified accordingly. With these changes, the algorithm for encoding individual tracks can be used to jointly encode all tracks in a subframe. Moreover, with appropriate choice of permutation and partitioning, both encodings give the same result and are thus equivalent. The formula in (7) is, however, easier to handle in a practical application and thus preferred, and the above generalization was presented mostly for its pedagogic value.
Table 1: Bit-consumption per track. The gain is the improvement for the proposed approach relative to AMR-WB and the gain in parenthesis is the improvement for integer bit-consumption.

<table>
<thead>
<tr>
<th>Length</th>
<th>Pulses</th>
<th>States</th>
<th>AMR-WB</th>
<th>Proposed</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ 1</td>
<td>15</td>
<td>32</td>
<td>5</td>
<td>5</td>
<td>0 (0)</td>
</tr>
<tr>
<td>16 2</td>
<td>512</td>
<td>9</td>
<td>9</td>
<td>0 (0)</td>
<td></td>
</tr>
<tr>
<td>16 3</td>
<td>5472</td>
<td>13</td>
<td>12.42</td>
<td>4.5 (0)</td>
<td></td>
</tr>
<tr>
<td>16 4</td>
<td>44032</td>
<td>16</td>
<td>15.43</td>
<td>3.5 (0)</td>
<td></td>
</tr>
<tr>
<td>16 5</td>
<td>285088</td>
<td>20</td>
<td>18.12</td>
<td>9.4 (5.0)</td>
<td></td>
</tr>
<tr>
<td>16 6</td>
<td>1549824</td>
<td>22</td>
<td>20.56</td>
<td>6.5 (4.5)</td>
<td></td>
</tr>
</tbody>
</table>

Note that other codebooks designs, such as flexible pulse allocation per track, can readily be incorporated similarly as the joint encoding of tracks. Moreover, observe that the jointly encoded state of a set of smaller 8- and 16-bit representations used in common CPUs. Some additional effort must therefore be taken to handle arithmetic operations with long integers.

6. Results

The presented algorithm can readily be applied to encode algebraic codebooks of any size and it can be adopted to match almost any desired level of bit consumption. To compare the performance with the most commonly used codebooks, in Table 1, a comparison between AMR-WB and the proposed algorithm is presented. We can clearly see that for fairly low bit-rates with 1 or 2 pulses per track, AMR-WB already has optimal performance and nothing is gained. However, when a higher number of pulses are encoded the improvement in bit-consumption is up to 9.4 percent. For example, for 6 pulses in a track of length 16, the structure used in AMR-WB would require 22 bits, whereas the proposed algorithm requires 20.56 bits, giving an improvement of 6.5 percent. Clearly, an integer number of bits must be used for the encoding. Therefore the 20.56 bits have to be encoded with 21 bits which would give a slightly smaller gain of 4.5 percent.

Since joint encoding of tracks provides further advantage, a comparison between AMR-WB and the proposed algorithm with joint encoding of tracks is also presented in Table 2. Like for the individual tracks, also here the low bitrates have no improvement, but for the higher bit-rates improvements up to 8.8 percent are observed. Note that presented codebooks are simpler in form than those used in AMR-WB, but allow a more straightforward comparison with the current approach without loss of generality.

In AMR-WB, at 23.85 kbit/s, the algebraic codebook requires 352 bits (73.8 percent) of the total payload of 477 bits per frame. This includes 4 subframes with 4 tracks each. With the proposed algorithm, the same algebraic codebooks can be achieved with 332 bits per frame, giving a total bit-rate per of 457 bits per frame, or 22.85 kbit/s. The proposed algorithm thus gives an improvement of 1 kbit/s.

Table 2: Bit-consumption per sub-frame with four tracks. The gain is the improvement for the proposed approach relative to AMR-WB and the number in parenthesis is the integer bit consumption.

<table>
<thead>
<tr>
<th>Tracks</th>
<th>Length</th>
<th>Pulses</th>
<th>AMR-WB</th>
<th>Proposed</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ 4</td>
<td>16</td>
<td>1</td>
<td>20</td>
<td>20 (20)</td>
<td>0 (0)</td>
</tr>
<tr>
<td>4 16</td>
<td>2</td>
<td>36</td>
<td>36 (20)</td>
<td>0 (0)</td>
<td></td>
</tr>
<tr>
<td>4 16</td>
<td>3</td>
<td>49.7</td>
<td>49.7 (50)</td>
<td>3.8%</td>
<td></td>
</tr>
<tr>
<td>4 16</td>
<td>4</td>
<td>64</td>
<td>64 (62)</td>
<td>3.1%</td>
<td></td>
</tr>
<tr>
<td>4 16</td>
<td>5</td>
<td>80</td>
<td>72.5 (73)</td>
<td>8.8%</td>
<td></td>
</tr>
<tr>
<td>4 16</td>
<td>6</td>
<td>88</td>
<td>82.3 (83)</td>
<td>5.7%</td>
<td></td>
</tr>
</tbody>
</table>

7. Conclusion and Discussion

In this contribution, an algorithm is presented for efficient encoding of the algebraic codebook used in speech and audio codecs of the ACELP-type. A descriptive characteristic of the proposed encoding is that each possible constellation of pulses is represented by an unique state. It exploits the fact that the ordering of pulses does not carry useful information, and that overlapping pulses must have the same sign.

Due to the constant bit-rate design, it is always possible to choose a suitable codebook based on the number of available bits. The algorithm is optimal in the sense that it gives the smallest possible bit-consumption for a given design of codebooks, or conversely, given a fixed bit-allocation, it gives the largest available codebook. Informal experiments show that the computational complexity of the proposed algorithm is comparable to the encoding used in AMR-WB. In terms of bit-rate, in comparison to AMR-WB, the proposed algorithm gives an advantage of up to 8.8 percent of the bit-rate. In other words, for AMR-WB at 23.85 kbit/s, the proposed algorithm gives an improvement of 1 kbit/s.

8. References