An LZ Approach to ECG Compression

R. Nigel Horspool[1] and Warren J. Windels[2]
[1] Dept. of Computer Science, University of Victoria,
P.O. Box 3055, Victoria, BC, Canada V8W 3P6
[2] Harley Street Software Ltd., 310 - 1900 Richmond St.,
Victoria, BC, Canada V8R 4R2

Abstract

Two highly effective families of lossless compression methods were created by Ziv and Lempel. However, these methods are normally ineffective when applied to digitized signals. We have successfully adapted the LZ77 method to take advantage of the repetitive nature of ECG waveforms. In our experiments, the resulting algorithm produces much better compression ratios than competing techniques while retaining the essential characteristics of the waveform.

1: Introduction

A typical electrocardiagram monitoring device generates massive volumes of digital data. Depending on the intended application for the data, the sampling rate ranges from 125 to 500 Hz. Each data sample may be digitized to a 8 to 12 bit binary number. Furthermore, up to 12 different streams of data may be obtained from various sensors placed on the patient's body. Even at the lowest sampling rate in the range and assuming just one sensor that generates 8-bit data, we would accumulate ECG data at a rate of 7.5 KB per minute or 450 KB per hour. At the other extreme (12 sensors generating 12-bit values at 500 Hz), data is generated at a rate of 540 KB per minute or more than 30 MB per hour.

If we wish to store an ECG recording that spans more than a few minutes, some form of data compression is highly desirable. Recordings over quite long periods, as much as 24 hours, may be needed when a patient has complained of irregular heart rhythms. Doctors may wish to build a database of ECG recordings for their patients so that two ECG traces taken on different dates may be compared. Compression is also desirable if a doctor wishes to transmit ECG over a telephone line to a cardiac specialist foran expert opinion.

An ECG trace is a digitized version of a continuous signal. Therefore compression techniques applicable to such signals are also applicable to ECG traces. For example, DPCM (Differential Pulse Code Modulation) and various versions of that technique are eminently suitable. Another possibility based on signal processing techniques is to apply a transform coding and to encode the subsequent coefficients in a lossy manner. For example, a Cosine Transform (CT) or Fourier series transformation may be applied. The Karhunen-Loeve transform (KLT) appears to be particularly suitable and is often used for ECG compression because (to quote [5]) "... the least number of orthonormal functions is needed to represent the input signal for a given rms error."

A particularly simple technique, frequently used for ECG data, is the TP (turning point) method. The essential idea is to eliminate one point from every pair of points in the trace. Usually, every second point is deleted. However, valuable diagnostic information may be lost if a maximum or a minimum point is lost. Thus, if one of the two points in the pair is a local maximum or a local minimum, then it is the other point in the pair that is deleted. The turning point method gives a compression factor of exactly 50%, that is the volume of the data is halved.

Another standard technique is known as *Fan* which is an algorithm for replacing the original waveform with straight line segments such that none of the original points lies further from the line segment than some predetermined maximum deviation. A related family of techniques, still approximating the waveform with straight line segments, is known as SAPA (Scan-Along Polygonal Approximation).

Yet another commonly used technique is known as AZTEC (Amplitude Zone Time Epoch Coding) [4]. This converts the ECG waveform into plateaus (flat line segments) and sloping lines. As there may be two consecutive plateaus at different heights, the reconstructed waveform shows discontinuities. To quote [5]: “Even though the AZTEC provides a high data reduction ratio, the fidelity of the reconstructed signal is not acceptable to the cardiologist because of the discontinuity (step-like quantization) that occurs in the reconstructed ECG waveform.” An enhanced method known as CORTES (Coordinate Reduction Time Encoding System) [1] applies TP to some portions of the waveform and AZTEC to other portions and does not suffer from discontinuities.

A comparison between the ECG compression methods mentioned above and several others was published by Jalaleddine et al. [5]. It includes Table 1, below, that collects reported results from many different sources. (Unfortunately some details are omitted from the table.)

Table 1. Summary of ECG Data Compression Schemes

Method	Compression Ratio	Sampling Rate (No. of Bits)	Percent RMS Difference
AZTEC	10.0	500 (12)	28.0%
TP	2.0	200 (12)	5.3%
CORTES	4.8	200 (12)	7.0%
Fan/SAPA	3.0	250	4.0%
Entropy Coding of 2nd Differences	2.8	250 (10)	–
Peak Picking (Spline) with Entropy Coding	10.0	500 (8)	14.0%
DPCM – Delta Coding with Threshold	4.0	300 (8)	–
DPCM – Linear Prediction	2.5	250 (12)	–
DPCM – Linear Pred, Interpl., Entropy Coding	7.8	500 (8)	3.5%
Orthogonal Transforms – CT, KLT, HT	3.0	250	–
Dual Application of KLT	12.0	250 (12)	–
Fourier Descriptors	7.4	250 (12)	7.0%

2: ALZ77 – A New Approach to Compressing ECG Traces

To the untrained human eye, an electrocardiogram appears to be extremely repetitive. Each heart beat yields a wave form that looks much like the preceding heart beat. Unfortunately, the data values are not *exactly* the same. Even if the heart beat is perfectly regular, the wave form may not have been sampled at exactly the same positions within the heartbeat and there may be digitizing errors introduced by the analogue to digital conversion process. The heart beat is not, of course, perfectly regular. The heart beat rate may be changing, the amplitudes of the signals may be changing, and there may indeed be arhythmias present (perhaps the reason why the ECG recording was being made in the first place). Finally, the patient is probably not keeping perfectly still – movements of the body and even the action of breathing cause the sensors to move slightly and cause the ECG readings to be affected.

In spite of the foregoing explanation, the ECG trace is approximately regular. Even in the presence of arhythmias and artifacts caused by the patient moving, most heart beats should be very similar to the preceding heart beat. If we maintain a sliding window that contains the readings from at least one entire heart beat cycle, we should be able to match fragments of the next heart beat – at least in an approximate manner. What we have just described is very suggestive of the LZ77 [6] family of compression methods. (An excellent survey of the LZ77 family appears in [2].) It seems that we need only use approximate matching instead of exact matching and we will have a compression algorithm.

To encapsulate the argument given so far, our first version of a compression algorithm has the structure shown below. The code is given at a relatively high level (in pseudo-C code), omitting numerous details. To list two of these details: First, it assumes that there is cyclic buffer B with a capacity to contain at least as many data samples as are obtained in one entire period of the heart beat. Second, the two different kinds of output, a raw data value or a position/length pair, need to be distinguishable (e.g by prefixing the codes with a 0 bit or 1 bit); some encoding scheme must be used for these two kinds of outputs.

```
while( input remains ) {
    find the largest value of K such that the next K input
        values approximately match a sequence of K consecutive
        values in the buffer B (and let the position of the
        first value in B be denoted by P);
    if (K < minimumMatchLength) {
        read one data value into V;
        output V as a raw data value;
    } else {
        output the pair <P,K>;
        read and copy K data values into the buffer B;
    }
}
```

We say that two sequences of values *approximately match* if the absolute difference between the values at the same position in the two sequences do not exceed a predefined tolerance value for every position. Some C code that implements our notion of approximate matching is as follows:

```
int approxMatch( int *array1, int *array2, int maxLength) {
    int len;
    for( len = 0; len < maxLength; len++ ) {
        if (ABS(array1[len]-array2[len]) > tolerance) break;
    }
    return length;
}
```

Unfortunately, our algorithm does not work. The de-compression algorithm would (as with any LZ77-based technique) take the form shown below.

```
while( compressed input remains ) {
    if (next input item is a raw data value) {
        read next input item into V;  output V;
        append V to the buffer B;
    } else {  /* the data is a position/length pair */
        read a position/length pair into P and L;
        for( i = 0; i < L; i++ ) {
            output B[i];  append B[i] to the buffer B;
        }
    }
}
```

If we de-compress the results of the earlier compression algorithm, the re-constructed ECG trace slowly but steadily diverges from the correct value. The problem is that the many small allowed deviations in the data values progressively accumulate. We can even identify where the errors in the reconstructed waveform originate. They are caused by these two lines in the compression algorithm:

```
output the pair <P,K>;
read and copy K data values into the buffer B;
```

When the de-compression algorithm inputs the pair <P, K>, it re-creates the *K* data values that are found at position *P* in its buffer. It appends these *K* values to the end of its cyclic buffer. However, the compression algorithm did not append those same value to the end of *its* cyclic buffer. Thus the encoding and decoding buffer disagree slightly as to what the last *K* values should be. As more ECG values are processed, the disagreements tend to grow upon themselves.

Fortunately, there is a solution to the problem. If we wish to remove the disagreement between the compression algorithm's buffer and the de-compression algorithm's buffer, we can make the compression algorithm update its buffer differently. It should execute the following actions instead of the two lines above:

```
output the pair <P,K>;
discard K data values from the input;
for( i = 0; i < K; i++ )
    copy one value from the buffer B at position P+i
        and append it to the buffer B;
```

Our change implies that ALZ77 is not strictly a sliding window compression technique. However, the buffer does contain data values that are very similar to the values read from the input stream. The maximum difference is guaranteed to be no more than the tolerance parameter used by the sequence matching code.

3: Experimental Results

The results shown below were obtained from an ECG trace that has a duration of a little over 15 minutes. We used a sampling rate of 250 samples/second, generating 8 bits per sample. The actual range of values was from -59 to +91. We used an implementation of the compression algorithm with a buffer size of 4096. We encoded each raw data value in the compressed output as a one bit tag followed by 8 bits; we encoded each position/length pair as a one bit tag followed by a 12 bit position plus a 8 bit length. (We imposed a maximum match length of 255 on the implementation to simplify the coding of the length field.)

The experimental results are shown in Table 2. We ran the ALZ77 compression algorithm using four different values for the matching tolerance, 0 through 3. The columns, except the rightmost, are intended to simplify comparison against Table 1. The extra column at the right provides some information about how ALZ77 obtains the compression. The entry in the first row, 11036v+23580p, indicates that the compressed file contains 11036 raw data values plus 23580 position/length pairs.

Note that the first row of the table has been obtained with a tolerance of 0. In other words, this is lossless compression using the normal LZ77 compression method (albeit with a rather inefficient mechanism for locating longest matches). The compression ratio so obtained is already better than several of the methods reported in Table 1. When we make the matching tolerance non-zero, the lengths of the matches between the input and the buffer contents increase dramatically and the compression ratio becomes excellent.

Table 2. Results Obtained Using the ALZ77 Method

Tolerance	Compression Ratio	Sampling Rate (No. of Bits)	Percent RMS Difference	Characteristics of Compressed File
0	3.2	250 (8)	0%	11036v + 23580p
1	12.9	250 (8)	0.3%	2915v + 5657p
2	22.5	250 (8)	0.8%	1106v + 3494p
3	31.4	250 (8)	1.6%	495v + 2633p

The Percent RMS Difference (PRD) is used to compare how well the re-constructed waveform matches the original data. It is computed as follows [1]:

$$\text{RMS Error} = \sqrt{\left(\sum_i (R_i - V_i)^2\right) / N}$$

$$\text{RMS Value} = \sqrt{\left(\sum_i V_i^2\right) / N}$$

$$\text{PRD} = (\text{RMS Error}) / (\text{RMS Value}) \times 100\ \%$$

where V_i is the *i*-th ECG value in the input source, R_i is the *i*-th value in the reconstructed waveform, and there are N values in total.

The PRD is an imperfect measure of distortion in the reconstructed waveform because some regions of the ECG trace have much greater importance for diagnostic purposes than other regions. However, it is the only objective measure that is consistently used in the literature and we should therefore report it too.

Visual before-and-after comparisons are provided by Figures 1 through 3. The first ECG trace shows five consecutive heart beats in the original recording. The second and third traces show the results of compressing with tolerances of 2 and 3 (the largest tolerance value used in our experiments) and then de-compressing.

4: Summary and Conclusions

We have shown that a well-known lossless data compression technique, LZ77, can be extended to work with digitized ECG traces. The new algorithm, ALZ77, appears to be much better than competing techniques. We have, in fact, not even attempted to squeeze every last drop of compression performance from the algorithm. If we were to apply Huffman coding to the raw data values and to the position/length pairs in the encoded data, we could do better (but at the cost of a little more CPU time). Another possibility for improving compression that could be pursued is the simultaneous compression of signals obtained from more than one sensor – since the signals are strongly correlated. We also foresee the possibility that the tolerance permitted by the approximate matching code might be dynamically adjusted according to the current position in the heart beat cycle. Some regions of the waveform are more important than others for diagnostic purposes; it should be possible to record precise waveform information in an important region and be less precise elsewhere.

References

1. J. P. Abenstein and W. J. Tompkins. A New Data-Reduction Algorithm for Real-Time ECG Analysis. IEEE Trans. on Biomedical Eng., vol. 29, 1 (Jan. 1982), pp. 43-48.
2. T. C. Bell, J. G. Cleary and I. Witten. *Text Compression*. Prentice-Hall, 1990.
3. T. C. Bell and D. Kulp. Longest-Match String Searching for Ziv-Lempel Compression. Software – Practice & Experience, vol. 23, 7 (July 1993), pp. 757-771.
4. J. R. Cox, F. M. Nolle, H. A. Fozzard and G. C. Oliver Jr. AZTEC – A Preprocessing Program for Real-Time ECG Rhythm Analysis. IEEE Trans. on Biomedical Eng., vol. 15 (1968), pp. 128-129.
5. S. M. S. Jalaleddine, C. G. Hutchens, R. D. Strattan and W. A. Coberly. ECG Data Compression Techniques – A Unified Approach. IEEE Trans. on Biomedical Eng., vol. 37, 4 (April 1990), pp. 329-341.
6. J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. IEEE Trans. on Information Theory, vol. 23, 3 (1977), pp. 337-343.

Figure 1. ECG Trace Before Compression with ALZ77

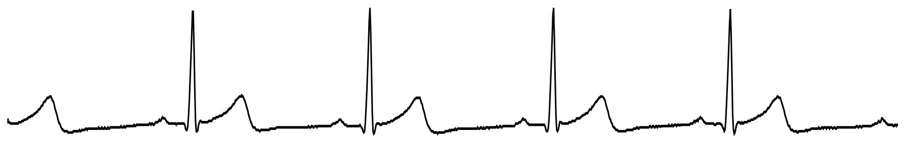

Figure 2. Reconstructed ECG Trace (Tolerance = 2)

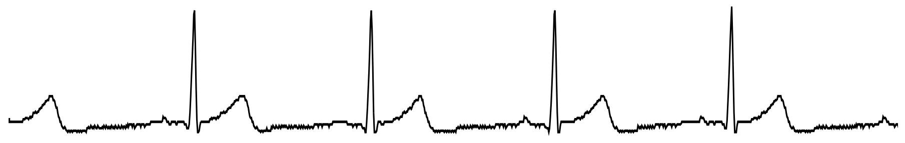

Figure 3. Reconstructed ECG Trace (Tolerance = 3)