1 Coding and Decoding

We have spent quite a while discussing modulation and demodulation. Modulation can be viewed as the entire process of mapping a bit stream at the input to the transmitter into a waveform for transmission, or it can be viewed as following a digital encoder.

We also looked at orthogonal sets of codewords, which allowed us to map each of the $m = 2^b$ tuples of $b$ binary digits into a set of $m$ orthogonal codewords. In terms of modulation, this can be viewed as a direct mapping of $b$-tuples into waveforms, and we found the corresponding error probability on a WGN channel and showed that in the limit $m \to \infty$, this scheme could approach the infinite bandwidth capacity with arbitrarily small error.

There is a question, however, about what orthogonal waveforms to choose. We can always view each orthogonal waveform as an amplified basis vector in some orthonormal basis for $L_2$, but the question then becomes the choice of basis (and the basis waveforms to use for the orthogonal set of interest). Part of the point of the signal space approach we have developed in this course is that to a certain extent, any basis is equivalent to any other basis, but it turns out, in Orwell’s terms, that some bases are more equivalent than others. In particular, choosing the orthogonal functions to be sinc functions, or orthogonal shifted pulses, would be a very poor choice, since it would lead to an extraordinarily large signal amplitude at one time and almost zero amplitude at all other times.

1.1 Orthogonal binary codes

A preferable approach to generating a large set of orthogonal waveforms comes from first generating a set of $m$ binary codewords, each of length $m$ and each differing from each other in $m/2$ places. Each binary digit can then be mapped into an antipodal signal, $0 \rightarrow +a$ and $1 \rightarrow -a$. This yields an $m$-tuple of real valued signals, $s_1, \ldots, s_m$, which is then mapped into the waveform $\sum_j s_j \phi_j(t)$ where $\{\phi_j(t)\}$ is an orthonormal set (such as sinc functions or square root of Nyquist pulses). It is easy to verify that, since each binary codeword differs from each other codeword in $m/2$ places, each such waveform is orthogonal to each other such waveform, and each has equal energy.

We say that a binary code with $m$ codewords, each differing from each other in $m/2$ places, is an orthogonal binary code. The notation arises from the fact that the mapping above yields orthogonal waveforms.

There are many ways to generate orthogonal codeword sets. Probably the simplest is from a Hadamard matrix. A Hadamard matrix $H_1$ of order 2 has the rows 00 and 01. For any
$b$, the Hadamard matrix $H_{b+1}$ of order $2^{b+1}$ can be expressed as four $2^b$ by $2^b$ submatrices. The upper two submatrices are $H_b$ and $H_b$ and the lower two are $H_b$ and $\overline{H}_b$, where $\overline{H}_b$ is the complement of $H_b$. This is illustrated in Figure 1 below.

Note that each row of the matrices in Figure 1, other than the all zero row, contain half zeroes and half ones. It can be verified by induction that this remains true for all larger values of $b$. To see this, we assume it is true for one value of $b$ and prove it to be true for $b+1$. For the first $2^b$ rows of $H_{b+1}$, each row has twice the length and twice the number of ones as the corresponding row in $H_b$. Note also that $\overline{H}_b$ has all ones in the first row and $2^b-1$ ones in each other row. Thus the first row in the second set of $2^b$ rows of $H_{b+1}$ has no ones in the first $2^b$ positions and $2^b$ ones in the final $2^b$ positions, yielding $2^b$ ones in $2^{b+1}$ positions. Finally, each remaining row has $2^b-1$ ones in the first $2^b$ positions and $2^b-1$ ones in the final $2^b$ positions, totalling $2^b$ ones as required.

It can also be seen, by a similar inductive argument, that the modulo two sum of any two rows of $H_b$ is another row. Since the mod 2 sum of two rows gives the positions in which the rows differ, this means that the set of rows is an orthogonal set.

We have accomplished two things with the above example. First we see how to generate orthogonal codes (and, by extension, bi-orthogonal codes and simplex codes) while using relatively constant power over the individual symbols, and second, we see that modulation over a large alphabet can (sometimes at least) be viewed as binary coding followed by modulation over a binary or very small alphabet.

## 1.2 Convolutional Codes

Coding is a very large and rich subject that will be treated in considerable depth in 6.451. Here we only give two examples. The first, above, generates orthogonal codes, which are useful in applications where bandwidth is not constrained and a small numbers of bits, $b$, can be mapped into a large number $2^b$ of dimensions. Another simple example is that of convolutional codes. These can be used when bandwidth is more constrained, and allow for a more modest expansion of bit rate from input to output. We give an example below where there are two output bits for each input bit. Such a code is said to have rate $1/2$. More generally, such codes can produce $m$-tuple of output bits for each $k$-tuple of input bits. but arbitrary integers $k < m$. These are said to have rate $k/m$. 

\[ \begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
\end{array} \]

\[ \begin{array}{cccc}
0000 & 0000 & 0101 & 0101 & 0011 & 0011 & 0110 & 0110 & 0000 & 1111 & 0101 & 1010 & 0011 & 1100 & 0110 & 1001 \\
\end{array} \]

Figure 1: Hadamard Matrices
A convolutional code looks very much like a discrete time filter. Instead of having a single input and output stream, however, we have $k$ input streams and $m$ output streams. For the example given here, there is one input stream and two output streams, thus producing two output bits per input bit. There is another difference between a convolutional code and a discrete time filter; the inputs and outputs for a convolutional code are binary and the addition is modulo 2. Consider the example below in Figure 2.

For the example above, the equations for the outputs are

$$U_{j,1} = D_j \oplus D_{j-1} \oplus D_{j-2}$$
$$U_{j,2} = D_j \oplus D_{j-2}$$

Thus each of the two output streams are linear modulo two convolutions of the input stream. This encoded pair of binary streams can now be mapped into a pair of symbol streams, such as antipodal signals $\pm a$. This pair of symbol streams can then be interleaved and multiplied by a single stream of orthonormal pulses at twice the rate. This baseband waveform can then be modulated and transmitted.

The structure of this code can be most easily visualized by a “trellis” diagram as illustrated in Figure 3.

![Figure 2: Example of a Convolutional Code](image)

![Figure 3: Trellis Diagram; each transition is labeled with the input and corresponding output](image)
To understand this trellis diagram, note from Figure 2 that the encoder is characterized at any discrete time $jT$ by the previous binary digits, $D_{j-1}$ and $D_{j-2}$. Thus the encoder has four possible states, corresponding to the four possible values of the pair $D_{j-1}, D_{j-2}$. Given any of these four states, the encoder output depends only on the current binary input, and the next state depends also only on that binary input. The diagram below shows these four states arranged vertically and shows time horizontally. We assume the encoder starts at time 0 with $D-1 = D_{-2} = 0$.

In the convolutional code of the above example, the output at time $j$ depends on the current input and the previous two inputs. In this case, the constraint length of the code is 2. In general the output could depend on the input and the previous $n$ inputs, and the constraint length is then defined to be $n$. If the constraint length is $n$ (and a single binary digit enters the encoder at each time $j$), then there are $2^n$ possible states, and the trellis diagram contains $2^n$ nodes at each time instant rather than 4.

As we have described convolutional codes above, the encoding starts at time 1 and then continues forever. In practice, because of packetization of data and various other reasons, the encoding usually comes to an end after some large number, say $j_0$, of binary digits have been encoded. After $D_{j_0}$ enters the encoder, two final 0’s enter the encoder, at times $(j_0+1)T$ and $(j_0+2)T$, and 4 final encoded digits come out of the encoder. This restores the state of the encoder to state 0, which, as we see later, is very useful for decoding. For the more general case with a constraint length of $n$, we need $n$ final zeros to restore the encoder to state 0. Thus, altogether, $j_0$ inputs lead to $2(j_0+n)$ outputs, for a code rate of $j_0/[2(j_0+n)]$. Since $j_0$ is usually large, we still refer to this as a rate 1/2 code. Figure 4 below shows the part of the trellis diagram corresponding to this termination.

![Figure 4: Termination of Trellis](image)

### 1.3 Decoding of Convolutional Codes

Decoding a convolutional code is essentially the same as using detection theory to choose between each pair of codewords, and then choosing the best overall (the same as we did
for the orthogonal code). There is one slight conceptual difference in that, in principle, the encoding continues forever. When we terminate the code, however, this problem does not exist, and we simply take the maximum likelihood choice of all the (finite length) possible code words.

As in our earlier discussions of detection, we assume that the incoming binary digits are equally likely. This means that the code words are equally likely, which then justifies maximum likelihood (ML) decoding. As before, we assume that the binary digits coming into a channel encoder have already been digitized from a source, and we know that good source compression leads to almost independent equi-probable binary digits.

We also use ML detection so that codes for error correction can be designed independently of the source data that will be transmitted through the given channel. For all the codes under discussion, the error probability using ML decoding is independent of the transmitted codeword, and thus ML decoding is robust in the sense that the error probability is independent of the probability distribution of the incoming bits.

We could, in some cases, design a code with smaller probability of error for a given input probability distribution, but we feel, given the above considerations, that it is clearly preferable to spend such extra effort on better source coding.

Another issue, given iid inputs, is determining what is meant by probability of error. In all of the examples above, given a received sequence of symbols, we have attempted to choose the code word that minimizes the probability of error. An alternative would have been to minimize the probability of error individually for each binary information digit. It turns out to be easier to minimize the error probability over the entire block of data than to minimize error probability per bit. This in fact is what happens when we use ML detection between codewords, as suggested above.

In decoding for error correction, the objective is almost invariably to minimize the probability of error over the code word. Along with the convenience suggested above, the reason for this is that the binary input is usually a source coded version of some other source sequence or waveform, and thus a single output error is often as serious as multiple errors within a code word.

Our assumption from now on is that we will do ML detection over code words.

### 1.4 The Viterbi Algorithm

The Viterbi algorithm is a ML detection algorithm over code words for a convolutional code. We assume for the time being that the code is terminated as in figure 3, but we shall soon see that whether or not it is terminated is irrelevant. We explain the algorithm for the example above; the extension to arbitrary convolutional codes is obvious except for the notational complexity of the general case. For any given input \(d_1, \ldots, d_{j_0}\), let the encoded input symbol sequence be \(u_1, \ldots, u_{2(j_0+2)}\) and let the output, after modulation, addition of WGN, and demodulation, be \(v_1, \ldots, v_{2(j_0+2)}\).

There are \(2^{j_0}\) possible code words, corresponding to the \(2^{j_0}\) possible binary \(j_0\)-tuples \(d_1, \ldots, d_{j_0}\), so an unimaginative approach to decoding would be to compare the likelihood
for each of these code words. For large \( j_0 \), even with today’s technology, such an approach would be prohibitive. It turns out, however, that by using the trellis structure of figures 2 and 3, this decoding effort can be greatly simplified.

Each input \( d_1, \ldots, d_{j_0} \) (i.e., each code word) corresponds to a particular path through the trellis from time \( T \) to time \( (j_0 + 2)T \), and each path, at each time \( jT \), corresponds to a particular trellis state.

Consider two paths \( d_1, \ldots, d_{j_0} \) and \( d'_1, \ldots, d'_{j_0} \) through the trellis that pass through the same state at time \( jT + 1 \) (i.e., at the time immediately after the input and state change at time \( jT \)) and remain together thereafter. Thus \( d_{j + 1}, \ldots, d_{j_0} = d'_{j + 1}, \ldots, d'_{j_0} \). For example, from figure 2, we see that \((0, \ldots, 0)\) and \(1, 0, \ldots, 0\) are both in state \( 00 \) at \( 3T + 1 \) and both remain in the same state thereafter. Since the two paths are in the same state at \( jT + 1 \) and have the same inputs after this time, they both have the same outputs after this time, so \( u_{2j + 1}, \ldots, u_{2(j_0 + 2)} \) is the same as \( u'_{2j + 1}, \ldots, u'_{2(j_0 + 2)} \).

This means that for any output \( v_1, \ldots, v_{2(j_0 + 2)} \),

\[
\frac{f_{V_1, \ldots, V_{2(j_0 + 2)}|D_1, \ldots, D_{j_0}}(v_1, \ldots, v_{2(j_0 + 2)}|d_1, \ldots, d_{j_0})}{f_{V_1, \ldots, V_{2(j_0 + 2)}|D_1, \ldots, D_n}(v_1, \ldots, v_{2(j_0 + 2)}|d'_1, \ldots, d'_{j_0})} = \frac{f_{V_1, \ldots, V_{2j}|D_1, \ldots, D_j}(v_1, \ldots, v_{2j}|d_1, \ldots, d_j)}{f_{V_1, \ldots, V_{2j}|D_1, \ldots, D_j}(v_1, \ldots, v_{2j}|d'_1, \ldots, d'_j)}
\]

In plain English, this says that if two paths merge at time \( jT + 1 \) and then stay together, the most likely of them can be detected from the first \( 2j \) outputs. Thus if the right hand side exceeds \( 1 \), then \( d_1, \ldots, d_{j_0} \) is more likely than \( d'_1, \ldots, d'_{j_0} \). This conclusion holds no matter how the final inputs \( d_{j + 1}, \ldots, d_{j_0} \) are chosen.

We then see that when two paths merge at a node, no matter what the remainder of the path is, the most likely of the paths is the one that is most likely at the point of the merger. Thus, whenever two paths merge, we can eliminate the least likely of the paths at that point. If we do this elimination successively starting at the smallest \( j \) for which paths merge (3 for the example), then there is only one survivor for each state at each time.

To be specific, let \( h(d_1, \ldots, d_j) \) be the state at time \( jT + 1 \) with input \( d_1, \ldots, d_j \). For the example, \( h(d_1, \ldots, d_j) = (d_{j - 1}, d_j) \). Let

\[
\max_{h(d_1, \ldots, d_j) = s} f_{V_1, \ldots, V_{2j}|D_1, \ldots, D_j}(v_1, \ldots, v_{2j}|d_1, \ldots, d_j).
\]

These quantities can then be calculated iteratively for each state and each time \( j \) by the iteration

\[
\max_{r, r \rightarrow s} f_{V|U}(v_{2j + 1}|u_1(r \rightarrow s)) f_{V|U}(v_{2j + 2}|u_2(r \rightarrow s)).
\]

where the maximization is over the set of states \( r \) that have a transition to state \( s \) in the trellis and \( u_1(r \rightarrow s) \) and \( u_2(r \rightarrow s) \) are the two outputs from the encoder corresponding to a transition from \( r \) to \( s \).

This expression is simplified (for WGN) by taking the log, which is proportional to the negative squared distance between \( \mathbf{v} \) and \( \mathbf{u} \). For the antipodal signal case in the example,
this is further simplified by simply taking the dot product between \( v \) and \( u \). Letting \( L(j, s) \) be this dot product,

\[
L(j + 1, s) = \max_{r : r \rightarrow s} \left( L(j, r) + v_{2j+1}u_1(r \rightarrow s) + v_{2j+2}u_2(r \rightarrow s) \right). \tag{2}
\]

What this means is that at each time instant \((j+1)T\), it is necessary to calculate the inner product in (2) for each link in the trellis going from \( j \) to \( j + 1 \). These must be maximized over \( r \) for each state \( s \) at time \((j+1)T\). The maximum must then be saved as \( L(j + 1, s) \) for each \( s \). One must, of course, also save the paths taken in arriving at each merging point.

Those familiar with dynamic programming will recognize this as an example of the dynamic programming principle.

The entire computation for decoding a block of \( j_0 \) information bits is proportional to \( 4(j_0+2) \). In the more general case where the constraint length of the convolutional coder is \( n \) rather than 2, there are \( 2^n \) states and the computation is proportional to \( 2^n(j_0+n) \). The Viterbi algorithm is usually used in cases where the constraint length is moderate, say 6 - 10, and in these situations, the computation is quite moderate, especially compared with \( 2^{j_0} \).

Usually one does not wait until the end of the block to start decoding. Usually when the above computation is done at time \( jT \), all the paths up to \( j'T \) have merged for \( j' \) a few constraint lengths less than \( j \). In this case, one can decode without any bound on \( j_0 \), and the error probability is viewed in terms of “error events” rather than block error.

### 1.5 Decoding for fading channels

For simplicity, consider a Rayleigh fading model in which the channel is described by a single tap, \( G_m \), which is slowly varying with \( m \). The complex output at discrete time \( m \) is then described by

\[
V_m = U_mG_m + Z_m
\]

where \( U_m \) is the complex channel input, and \( G_m \) and \( Z_m \) are complex circularly symmetric Gaussian random variables with variance \( 1/2 \) and \( N_0/2 \) per real and imaginary part respectively. The noise sequence \( Z_m \) is iid.

Assume that the fading is slow enough and the system clever enough so that the sample value \( g_m \) of the tap weight is continuously tracked at the receiver and fed back to the transmitter with small enough delay that both the transmitter and the receiver know \( g_m \) at all times.

This allows the transmitter to adjust the power level and to adjust the code rate dynamically. Code rate adjustment does not seem to be much used in practice, but it will be seen that it is probably more important than power adjustment.

The receiver knows the channel level \( g_m \) at each time, and we assume that it also knows the transmitter’s power and rate.
If some given code is used at the transmitter, the receiver knows the set of code words, the power level used at each time, and the channel state at each time. The receiver can then use ML decoding, based on all this information. The likelihoods still depend on the Gaussian noise, but this Gaussian noise is the only unknown quantity in the problem.

If we look at the Viterbi algorithm, we see that nothing has changed because of the time varying nature of the channel and the input power choice. The likelihoods calculated at each state and time will now depend on the channel and power at that time, but the structure of the decoder, making one ML decision at each time for each possible state of the decoder, remains the same. To be more precise, we can see from (2) that it is the amplitude of $u_1(r \rightarrow s)$ and $u_2(r \rightarrow s)$ that changes due to the channel fading. In the ML decision at any given state and time, this fading level determines how much weight is placed on the current outputs, and how much on the previous likelihoods.

Although the implementation of the Viterbi algorithm is not changed in any major way by known fading levels, the performance of the decoder (and similarly of an ML decoder for orthogonal signals) will depend heavily on the channel fading level and on the transmitted power level.

We can see, without any computation, that if the channel is slowly varying relative to the constraint length of the encoding (the block length for orthogonal codes and the convolutional constraint length $n$ for convolutional codes), then performance will be quite poor.

There are only three remedies for this situation. One is to send more power when the channel is faded. As shown in the exercises, however, if the input power compensates completely for the fading (i.e., the input power at time $m$ is $1/|g_m|^2$, then the expected input power is infinite. This means that, with finite average power, deep fades for prolonged periods cause outages.

The second remedy is diversity, in which each codeword is spread over enough coherence bandwidths or coherent time periods to achieve averaging over the channel fades. This, done over time, causes delays proportional to the time coherence period, which is usually unacceptable for voice. We will see in the next section how this strategy can be partially employed using diversity over frequency.

The third remedy is using variable rate transmission. This is not usually done for voice, since the voice encoding produces a constant rate stream of input bits into the channel, and the delay constraint is too stringent to queue this input and transmit it when the channel is good. It would be possible to break the source channel separation principle and have the source produce “important bits” at one rate and “unimportant bits” at another rate. Then when the channel is poor, only the important bits would be transmitted. One of the standards has a feature somewhat resembling this.

For data, however, variable rate transmission is very much a possibility since there is usually not a stringent delay requirement. Thus, data can be transmitted at high rate when the channel is good and at low rate or zero rate when the channel is poor. Newer systems take advantage of this possibility.
2 CDMA; The IS95 Standard

In this section, we briefly describe one of the major cellular standards. We have chosen IS95 since it is more interesting, from a conceptual viewpoint, than the others. This standard uses a spread spectrum approach, which is often known by the name CDMA (Code Division Multiple Access). Newer systems are also focusing primarily on spread spectrum approaches. There is no convincing proof that spread spectrum is inherently better than other approaches, but there are a number of reasons why it might have inherent engineering advantages. Our purpose is to get some insight into how a major commercial cellular network system deals with some of the issues we have been discussing. We discuss only the use of voice.

IS95 uses a frequency band from 800 to 900 megahertz (MH). The lower half of this band is used for transmission from cell phones to base station (the reverse channel), and the upper half is used for base station to cell phones (the forward channel). There are multiple subbands\(^1\) within this band, each 1.25 MH wide. Each base station uses each of these subbands, and multiple cell phones within a cell can share the same subband. Each forward subband is 45 MH above the corresponding reverse subband. The transmitted waveforms are sufficiently well filtered at both the cell phones and the base stations so that they don’t interfere appreciably with reception on the opposite channel.

The other two major established cellular standards use TDMA (time division multiple access). The subbands are more narrow in TDMA, but only one cell phone uses a subband at a time to communicate with a given base station. In TDMA, there is little interference between different cell phones in the same cell, but considerable interference between cells. CDMA has more interference between cell phones in the same cell, but less between cells.

A high level block diagram for the parts of a transmitter is given in figure 5.

Voice Waveform → Voice Compressor → Channel Coder → Modulator → Channel

Figure 5: High Level Block Diagram of Transmitters

The receiver, at a block level viewpoint (see figure 6), performs the corresponding receiver functions in reverse order. This can be viewed as a layered system, although the choice of function in each block is somewhat related to that in the other blocks.

\(^1\)It is common in the cellular literature to use the word channel for a particular frequency subband; we will continue to use the word channel for the transmission medium connecting a particular transmitter and receiver. Later we use the words multiaccess channel to refer to the reverse channel for multiple cell phones in the same cell.
We next describe these three blocks. The voice compression and channel coding are quite similar in each of the standards, but the modulation is very different.

2.1 Voice Compression

The voice waveform, in all of these standards, is first segmented into 20 ms. increments. These segments are long enough to allow considerable compression, but short enough to cause relatively little delay. In IS95, each 20 ms segment is encoded into 172 bits. The digitized voice rate is then \( 8600 = \frac{172}{0.02} \) bits per second (bps). Voice compression has been an active research area for many years. In the early days, voice waveforms, which lie in a band from about 400 to 3200 Hz, were simply sampled at 8000 times a second, corresponding to a 4 KH band. Each sample was then quantized to 8 bits for a total of 64,000 bps. Achieving high quality voice at 8600 bps is still a moderate challenge today and requires considerable computation.

The 172 bits per 20 ms segment from the compressor is then extended by 12 bits per segment for error detection. This error detection is unrelated to the error correction algorithms to be discussed later, and is simply used to detect when those systems fail to correct the channel errors. Each of these 12 bits is a parity check (i.e., a modulo 2 sum) of a prespecified set of the data bits. Thus, it is very likely, when the channel decoder fails to decode correctly, that one of these parity checks will fail to be satisfied. When such a failure occurs, the corresponding frame is mapped into 20 ms of silence, thus avoiding loud squawking noises under bad channel conditions.

Each segment of 172 + 12 bits is then extended by 8 bits, all set to 0. These bits are used as a terminator sequence for the convolutional code to be described shortly. With the addition of these bits, each 20 msec segment generates 192 bits, so this overhead converts the rate from 8600 to 9600 bps. The timing everywhere else in the transmitter and receiver is in multiples of this bit rate. In all the standards, many overhead items creep in, performing small but necessary functions, but increasing the data rate that must be transmitted by the encoder and modulator.

2.2 Channel Coding and Decoding

The channel encoding and decoding use a convolutional code and a Viterbi decoder. The convolutional code has rate 1/3, thus producing three output bits per input bit, and mapping the 9600 bps input into a 28.8 Kbps output. As we see later, the choice of rate is not very critical, and simply makes the timing come out in a convenient way. The encoder
has a constraint length of 8, so each of the three outputs corresponding to a given input depends on the current input plus the eight previous inputs. There are then $2^8 = 256$ possible states for the encoder, corresponding to the possible values for the previous 8 inputs.

The complexity of the Viterbi algorithm is directly proportional to the number of states, so there is a relatively sharp tradeoff between complexity and error probability. The fact that decoding errors are caused primarily by more fading than expected (either a very deep fade that cannot be compensated by power control or by an inaccurate channel measurement), suggests that increasing the constraint length from 8 to 9 would, on the one hand be largely ineffective, and, on the other hand, double the decoder complexity.

The decision was made to terminate the convolutional code at the end of each voice segment, turning the convolutional encoder into a block code of block length 576 and rate 1/3, with 192 inputs bits per segment. As mentioned in the previous subsection, this 192 bits includes 8 bits to terminate the code and return it to state 0. Part of the reason for this termination is the requirement for small delay, and part is the desire to prevent a fade in one segment from causing errors in multiple voice segments (the failure to decode correctly in one segment makes decoding in the next segment less reliable in the absence of this termination).

When a Viterbi decoder makes an error, it is usually detectable from the likelihood ratios in the decoder, so the 12 bit overhead for error detection could probably have been avoided. There are many such tradeoffs between complexity, performance, and overhead that must be made in both standards and products.

The decoding uses soft decisions from the output of the demodulator. The ability to use likelihood information (i.e., soft decisions) from the demodulator is one of the principle reasons for the use of convolutional codes and Viterbi decoding. Viterbi decoding uses this information in a natural way, whereas, for some other coding and decoding techniques, this can be unnatural and difficult. All of the major standards use convolutional codes, terminated at the end of each voice segment, and decode with the Viterbi algorithm. It is worth noting that channel measurements are useful here in generating good likelihood inputs to the Viterbi decoder.

The final step in the encoding process is to interleave the 576 output bits from the encoder corresponding to each voice segment. Correspondingly, the first step in the decoding process is to de-interleave the bits (actually the soft decisions) coming out of the demodulator. We can see without any analysis that if the noise coming into a Viterbi decoder is highly correlated, then the Viterbi decoder, with its short constraint length, is more likely to make a decoding error than if the noise is independent. We will see in the next subsection that the noise from the demodulator is in fact highly correlated, and thus the interleaving breaks up this correlation.

We can now summarize the source and channel coding with the following block diagram:
2.3 Modulation and Demodulation

The final part of the high level block diagram of the IS95 transmitter is to modulate the output of the interleaver before channel transmission. Here is where the notion of spread spectrum comes in, since this data stream at 28.8 Kbps is now to be spread onto a 1.25 MH bandwidth. Before explaining why this spreading might be desirable, we first describe how it is done.

The first step of the modulation is to segment the interleaver output into strings of length 6, and then map each successive 6 bit string into a 64 bit binary string. The mapping maps each of the 64 strings of length 6 into the corresponding row of the $H_6$ Hadamard matrix described in the last section. Each row of this Hadamard matrix differs from each other row in 32 places and each row, except the all zero row, contains exactly 32 ones and 32 zeros. It is thus an orthogonal code.

The selected word from this code is then mapped into a PAM sequence by the 2-PAM map $\{0, 1\} \rightarrow \{+a, -a\}$. These 64 sequences of binary antipodal values are called Walsh functions. The data rate coming out of this 6 bit to 64 bit mapping is $(64/6) \cdot 28.8 = 307.2$ kbps.

To get some idea of why these Walsh functions are used, let $x_k^1, \ldots, x_{64}^k$ be the $k^{th}$ Walsh function, amplified by a factor $a$, and consider this as a discrete time baseband input. For simplicity, assume flat fading with a single channel tap of amplitude $g$. Suppose that baseband WGN of variance $N_0/2$ (per real and imaginary part) is added to this sequence, and consider detecting which of the 64 Walsh functions was transmitted. Let $E_s$ be the expected received energy for each of the Walsh functions. Using the incoherent detection result from Lecture 22, we see that the probability that hypothesis $j$ is more likely than $k$, given that $x_k(t)$ is transmitted, is $1/2 \exp\left(-\frac{E_s^2}{2N_0}\right)$. Using the union bound over the 63 possible incorrect hypotheses, the probability of error, using incoherent detection and assuming a single tap channel filter, is

$$Pr(e) \leq \frac{63}{2} \exp\left(-\frac{E_s^2}{2N_0}\right)$$

(3)

It is not really the probability of error that we are interested in here, since the detector output is soft decisions that are then used by the Viterbi decoder. However, the error probability lets us understand the rationale for using such a large signal set with orthogonal signals.
If we could do coherent detection, the analogous union bound on error probability would be $63Q(\sqrt{E_s/N_0})$. As discussed in Lecture 22, this goes down exponentially with $E_s$ in the same way as (3), but the coefficient is considerably smaller. If we look at how many dB are required in increased power to get the same result with incoherent detection as with coherent detection, we find that this number decreases almost inversely with the exponent in (3). What this means is that by using a large number of orthogonal functions (64 in this case), we make the exponent in (3) large in magnitude, and thus come closer (in dB terms) to what could be achieved by coherent detection.

When we look more carefully, the argument above is incomplete, because $E_s$ is the transmitted energy per Walsh function. However, we are sending 6 binary digits out of the encoder for each transmitted Walsh function. Thus, $E_b$ in this case is $E_s/6$ and (3) becomes

$$Pr(e) \leq 63\exp(-3E_b/N_0).$$

(4)

This large signal set also avoids the 3 dB penalty for orthogonal signaling rather than antipodal signaling that we have seen for binary signal sets. Here the cost of orthogonality essentially lies in using an orthogonal code rather than the corresponding biorthogonal code with 7 bits of input and 128 code words, i.e., a factor of 6/7 in rate.

Another questionable issue here is that we are really using two codes (the convolutional code as an outer code, followed by the Walsh function code as an inner code) in place of a single code. There seems to be no clean analytical way of showing that this choice makes good sense over all choices of single or combined codes. On the other hand, each code is performing a rather different function. The Viterbi decoder is eliminating the errors caused by occasional fades or anomalies, and the Walsh functions are permitting incoherent detection with very little loss.

We now continue with our explanation of the modulation scheme used in IS95. The stream of binary digits out of the Walsh encoder is combined with a pseudo-noise (PN) sequence at a rate of 1228.8 kbps, i.e., four PN bits for each signal bit. In essence, each bit of the 307.2 kbps stream out of the Walsh encoder is repeated three extra times (to achieve the 1228.8 kbps rate) and is then added mod 2 to the PN sequence.

PN sequences are generated in a way very similar to the way the output streams are generated in a convolutional encoder. In a convolutional encoder of constraint length $n$, each bit in a given output stream is the mod 2 sum of the current input and some particular pattern of the previous $n$ inputs. Here, there are no inputs, but instead, the output of the generator is fed back to the input. By choosing the pattern used for the mod 2 sum in an appropriate way (a so-called maximal length shift register), any non-zero initial state will cycle through all possible $2^n - 1$ non-zero states before returning to the initial state.

One of the nice properties of a maximal length shift register is that it is linear (over modulo 2). That is, let $y$ be the sequence of length $2^n - 1$ bits generated by the initial state $x$, and let $y'$ be that generated by the initial state $x'$. Then it can be seen with a

---

2 This biorthogonal code is called a (64, 7, 32) Reed Muller code in the coding literature
little thought that $y \oplus y'$ is generated by $x \oplus x'$. Thus the difference between any two such cycles started in different initial states contains $2^n - 1$ ones and $2^n - 1 - 1$ zeros. In other words, the set of cycles forms a simplex code.

The constraint length here is $n = 42$ binary digits, so the period of the cycle is $2^{42} - 1$. We can ignore the difference between simplex and orthogonal, and simply regard each cycle as orthogonal to each other cycle. Since the cycle is so long, however, it is better to simply approximate each cycle as a sequence of iid binary digits. There are several other PN sequences used in the IS-95 standard, and this one, because of its constraint length, is called the “long PN sequence.” PN sequences have many interesting properties, but for us it is enough to view them as iid but also known to the receiver.

The initial state of the long PN sequence is used to distinguish between different cell phones, and in fact this initial state is the only part of the transmitter system that is specific to a particular cell phone.

The resulting binary stream, after adding the long PN sequence, is at a rate of 1.2288 Mbps, and this stream is then duplicated into two streams prior to being quadrature modulated onto a cosine and sine carrier. The cosine stream is then mod 2 added to another PN-sequence (called the in-phase or I-PN) sequence at rate 1.2288 Mbps, and the sine stream is mod 2 added to another PN sequence called the quadrature or Q-PN sequence. The two streams then go through the 2 PAM map, are amplified, modulated on the carrier and transmitted. Each of these streams (over blocks of 256 bits) maintains the orthogonality of the 64 Walsh functions. The I-PN and Q-PN sequences are the same for all cell phones and are used to aid in demodulation.

The final part of modulation is to pass the digital streams of $\pm a$ through a filter (turning them into narrow band waveforms of 614.4 KH). This is then quadrature modulated onto the carrier with a bandwidth of 614.4 KH above and below the carrier, for an overall bandwidth of 1.2288 MH. Note that almost all the modulation operation here is digital, with only the final filter and modulation being analog. Note that the question of what should be done digitally and what in analog form (other than the original binary interface) is primarily a question of ease of implementation.

A block diagram of the modulator is shown in figure 8.

![Figure 8: Block diagram of Source and Channel Encoding](image_url)
functions, each of length 256 and each (viewed at baseband) containing both a real and an imaginary part. The received waveform, after demodulation to baseband and filtering, is passed through a Rake receiver similar to the one discussed earlier. The Rake receiver here has a signal set of 64 signals rather than 2. Also, the channel here is viewed not as taps at the sampling rate, but rather as 3 taps at locations dynamically moved to catch the major received paths.

As mentioned before, the detection is incoherent rather than coherent.

The output of the rake receiver is a likelihood value for each of the 64 hypotheses. This is then converted into a likelihood value for each of the 6 bits in the inverse of the 6 bit to 64 bit Hadamard code map.

We now can see one of the reasons for using an interleaver between the convolutional code and the Walsh function encoder. After the Walsh function detection, the string of 6 bits coming out of the detection circuit have highly correlated errors. The Viterbi decoder, however, does not work well with bursts of errors, so the interleaver spreads these errors out, allowing the Viterbi decoder to operate with noise that is relatively independent from bit to bit.

2.4 Multiaccess Interference in IS95

A number of cell phones will use the same 1.2288 MH frequency band in communicating with the same base station, and other nearby cell phones will also use the same band in communicating with their base stations. We now want to understand what kind of interference these cell phones cause for each other. We consider the detection process for any given cell phone, and look at the effect of the interference from the other cell phones.

Since each cell phone uses a different long PN sequence, we can model the PN sequences from the interfering cell phones as random iid binary streams. Since each of these streams is modeled as iid, the addition of the PN stream and the data is still an iid stream of binary digits. If the filter used before transmission is very sharp (which it is, since the 1.2288 MH bands are quite close together), the orthonormal pulses can be approximated by sinc pulses. It also makes sense to model the sample clock of each interfering cell phone as being uniformly distributed. This means that we can model the interfering cell phones as being wide sense stationary with a flat spectrum over the 1.2288 MH band.

The more interfering cell phones there are in the same frequency band, the more interference there is, but also, since these interfering signals are independent of each other, we can invoke the central limit theorem to see that this aggregate interference will be approximately Gaussian.

To get some idea of the effect of the interference, assume that each interfering cell phone is received at the same baseband energy per information bit given by $E_b$. Since there are 9600 information bits per second entering the encoder, the power in the interfering waveform is then $9600E_b$. This noise is evenly spread over 2,457,600 dimensions per second, so is $(4800/2.4576\times10^6)E_b = E_b/512$ per dimension. Thus the noise per dimension is increased from $N_0/2$ to $(N_0/2 + kE_b/512$ where $k$ is the number of interferers. With this change,
(4) becomes

\[
\text{Pr}(e) \leq \frac{63}{2} \exp \left[ -\frac{3E_b}{N_0 + kE_b/256} \right]
\]  

(5)

In reality, the interfering cell phones are received with different power levels, and because of this, the system uses a fairly elaborate system of power control to attempt to equalize the received powers of the cell phones being received at a given base station. Those cell phones being received at other base stations presumably are received at somewhat lower powers, and thus cause less interference. It can be seen that with a large set of interferers, the assumption that they form a Gaussian process is even better than with a single interferer.

The factor of 256 in (5) is due to the spreading of the waveforms (sending them in a bandwidth of 1.2288 MH rather than in a narrow band. This spreading, of course, is also the reason why appreciable numbers of other cell phones must use the same band. Since voice users are typically silent half the time while in a conversation, and the cell phone need send no energy during these silent periods, there is an additional factor of 2 gain in the number of tolerable interferers.

The other types of cellular systems (GSM and TDMA) attempt to keep the interfering cell phones in different frequency bands and time slots. If successful, this is, of course, preferable to CDMA, since there is then no interference rather than the limited interference in (5). The difficulty with these other schemes is that frequency slots and time slots must be reused by cell phones going to other cell stations (although preferably not by cell phones connected with neighboring cell stations). The need to avoid slot re-use between neighboring cells leads to very complex algorithms for allocating re-use patterns between cells, and these algorithms can not make use of the factor of 2 due to users being quiet half the time.

Because these transmissions are narrow band, when interference occurs, it is not attenuated by a factor of 256 as in (5). Thus the question boils down to whether it is preferable to have a large number of small interferers or a small number of larger interferers. This, of course, is only one of the issues that differ between CDMA systems and narrow band systems. For example, narrow band systems can not make use of rake receivers, although they can make use of many techniques developed over the years for narrow band transmission.