ERROR-CORRECTING CODES WEEK 4

Non-binary BCH codes, in the form of **Reed-Solomon (RS) codes**, are amongst the most frequently used block codes. They have a particularly simple structure. The vectors are over a finite field GF(q) and in practice this is usually a field with a binary characteristic, so that q = 2^r. A typical value for r is r=8. **In RS codes the polynomial functions of g(x) are of first degree**: (x-1), (x-α), (x- α^2) etc. where α is in the ground field and has order (q-1) if primitive. These polynomials must divide (x^n – 1) where n is the length of the code, but n = (q^1 –1), **so n= q-1.** If there are m first degree polynomials in g(x) then g(x) is of degree m and (n-k) = m. If the polynomials have consecutive roots then d = m+1, which means **d = n-k +1**. There are only (n-k) columns in H, so they must all be linearly independent and no larger distance than that is possible: so in that sense RS codes are optimal.

Example: With q = 2^3 suppose the RS code has consecutive roots α, α^2, α^3, α^4 where α is a root of x^3 + x + 1 which defines the field. The successive powers i of α^i = xxx can be represented as follows i:xxx. 0:001. 1:010, 2:100, 3:011, 4:110, 5:111, 6:101, 7:001. We have g(x) = (x-1).(x-α).(x-α^2).(x-α^3) which works out as x^4 + α^2.x^3 + α^5.x^2 + α^5.x + α^6.

Error-correction of RS, or other non-binary BCH codes is similar to the binary procedure but the values of the errors Y[k] are no longer 1, and must be taken into account. We get the syndromes **S[i] = Sum(k=1 to t) Y[k].X[k]^i for i = 1 to m.** As before we assume an f(x) which has roots X[k], find the coefficients of f(x) as before because the S[i] are observed, and solve f(x) to find X[k] the error locators. But then an extra step comes in which **we solve to find the Y[k] from the above simultaneous equations**. Finally we subtract the Y[k] to correct the received vector at the positions indicated by the error locators.

Example: Using the above RS code over GF(8) we receive (11110100000000001000000) or (α^5 α^6 0 0 1 0 0) and evaluate S[1] = α^5 + α^6 + 1 = α^3, S[2] = α^2 +α^4 + α^4 = α^2, S[3] = α^4 + α^2 + α^3 = 1, S[4] = α^6 + 1 + α^2 = 0.
From S[3] + f1.S[2] + f2.S[1] = 0 we get 1+ f1.α^2 + f2. α^3 = 0, and from S[4] + f1.S[3] + f2.S[2] = 0 we get f1 + f2.α^2 =0. These solve to f2= α and f1 = α^3. Then the roots of f(x) are the error locators x = 1 and x = α. The first two syndrome equations give Y[1] + Y[2] = α ^3 and Y[1].α + Y[2] = α^2, giving Y[1] = α^2 and Y[2] = α^5. *Students to check that the third and fourth syndrome equations are consistent.* So the corrected codeword is (α^5 α^6 0 0 1 α^2 α^5) or (11110100000000001100111).

Exercise. With the same code correct (α ^3 0 0 α^6 0 1 α^6). Beware!

A particularly useful property of RS codes is that their weight distribution can be calculated (see Annexe 4A). This enables the probability of incorrect error-correction to be calculated. See Annexe 4B.

Convolutional/Trellis Codes

The codewords of convolutional codes are of indeterminate length. If the code is systematic a codeword is a sequence of alternating small (1 to four, say) groups of b data symbols and v check symbols. For example one might have a continuous sequence b=1 data bit accompanied by v=2 check bits, giving a 1/2-rate code. Typically code-rates are low. The codes are mostly used on high capacity multi-megabit-per-

second channels where retransmission of erroneous data is not an option (e.g. satellite channels), and so-called forward eror-correction (FEC) must apply – that is correction on reception.

The discussion assumes the codes are binary, but of course the use of non-binary symbols is possible.

The coding of user data may be viewed using trellis diagrams (see Annexe 4B). The encoder is in one of a limited number of states (e.g. 4). The trellis represents a set of valid connections between states, and each set of b input data bits for encoding causes a move to another state and the emission of v output bits, which go to make up the codeword. As input data bits are processed a path through the trellis from state to state is followed, and the connectivity of the trellis determines which paths are possible. It is the job of the receiving decoder to determine from the possibly corrupted codewords which path was followed at the transmitting encoder. There is no reason why the code should be systematic, but if it is perhaps decoding is easier.

Annexe 4C illustrates how an encoder may function. It is linear, so the distances between codewords may be ascertained from their weight. *Why?* It has **constraint length k = 3, in the form of a 3-bit shift register, whose two left-hand bits determine the current state**: 00=a, 10=b, 01=c, 11=d. The next data bit enters from the left. The two output bits are formed from linear XOR circuits as shown, and are read by a switch immediately after a new input bit is fed in. The trellis diagram illustrates the possible connection (a-a,b) (b-c,d) (c-a,b) (d-d,c).

The state diagram and the trellis indicate the relation between input (shown at the start of each transition arrow) and output shown at the arrow head. *What is the distance of the code?*

The operation of the encoder may be summarised in the following four equations:

1) b = LND^2.x + LND.c
2) c = L.d + LD.b
3) d = LND^2.d + LND.b
4) y = LD.c

We consider a valid deviation from the all-zero vector (staying at state (a)). D is a distance operator, the distance between the emitted symbols and 00. L is the number of inter-state hops involved on the transition. N is the number of input 1s involved. The starting point is x(=a) and the ending point is y(=a). Solving these equations for y in terms of x gives the (powers of) distances D, lengths L, input 1s involved for all the paths round the state diagram or through the trellis. (In solving we use 1/(1-X) = 1+X+X^2 + X^3 . . . . where X is an operator).

The solution is y = x.[D^4.L^3.N(1+LN) + D^6.L^5.N^2(1+N+2LN+L^2.N^2) + D^8 . . . cutting off at distances D greater than 6. The corresponding paths are xbcy, xbdcy, xbcbcy, xbddcy, xbdcbcy, xbcbdcy,xbdcbdcy. *Students to check.*

More generally we can suppose we have y = Sum(k to infinity) f[k](L,N).D^k. Evaluate f[k](1,1) and we have the number of codewords of weight = k. The expression can also be used to calculate probabilistic bounds on erroneous decoding (because errors cause a transmitted codeword to be nearer another

codeword) and also to calculate the post-decoding bit error-rate using N, which gives the implied number of input bits associated with a wrongly decoded vector..

Exercise. A three-stage register has a switched two-bit output, as before. But the top output is the XOR of all three stages, and the bottom output is the XOR of the first and third stages. Draw the state diagram and the trellis. What is the distance of the code it generates?

We can also represent the encoder analytically, with X(T) as input where T is a delay operator of one time unit, X(T) = x[0] + x[1]T + x[2]T^2 + x[3]T^3 . . . of indefinite length; and with outputs – two in our examples - Y[s](T) similarly. They are related by the coding polynomials of degree (k-1) G[s] = g[0] + g[1]T + g[2]T^2 . . +g[k-1]T^(k-1) in the form Y[s](T) = G[s](T).X(T). Our first example was Y[1](T) = X[T] and Y[2](T) = (1 + T + T^2).X(T), because it was a systematic code. In principle one can invert such equations to decode, getting two or more estimates for X(T) viz X*[s](T) = G[s]^(-1),Y[s](T), which hopefully agree.

Some infinite inputs (e.g.X[T} = 1+T+T^2+T^3+ . . .) can give a finite output (e.g with G[T] = 1+T)). Inverting could lead to a single bit-error in Y[T] producing an infinite error X*[T]. This would not matter unless *all* the estimates X*[s](T) make the same mistake, which could occur if the G[s](T) have a common factor (e.g G[1](T) = 1+T and G[2](T) = 1+T^2), but cannot occur with systematic codes. *Students to explain.*

Annexes 4D, 4E present some standard convolutional codes.

Decoding Convolutional Codes

The optimum decoding method, finding a valid input x nearest to the possibly corrupted outputs y[s], is Viterbi's method – but it is slow and complex. For each instant of time and for each state on the trellis two values are held:

1) The identity of the preceding state from which a path of least distance to the received vector can be traced backwards (the 'nearest' path).
2) The accumulated distance between the output bits of this nearest path and the received vector bits.

At the next instant we consider one by one each state. It can be reached from a selection of the last states (as limited by the trellis) and from each such last state there is an increase in distance, namely the difference between the v bits emitted according to the trellis and the v bits actually received in the last instant. Add this increase to the accumulated distance for each of those last states, and them select as new last state that with least accumulated distance. Do the same for all other current states. Record the pairs of values for this instant for each state.

At any instant we can look at all the states, pick that of least accumulated distance and follow back the nearest path using the pointers to preceding states. Annexe 4F illustrates the procedure for correcting a received vector with two errors, using the trellis as defined by the previous student exercise. It shows how at each instant and state there are two possible previous states, how the nearest one is retained and the other crossed off. It also shows the decoded path through the trellis to be followed backwards from the right-

hand side – and then once the correct sequence of states has been determined the original outputs can be re-created.

Although, as decoding proceeds, there may be major changes in the most recent parts of the nearest path, the less recent parts become stable. The earlier history of the transitions becomes clear and fixed. This is often used to simplify the algorithm: Don't hold an excessive list of previous states and back-pointers but make a decision as to the correct sequence from some not-too-remote instant backwards; Make the most recent history simpler by a convention that all transmissions will end with a standard repetition of, say, ten returns to state zero (a = 00).

Other De-coding Methods

There are many different approaches to faster, albeit sub-optimum decoding than Viterbi's method. One such is to make a decoding decision immediately new symbols are received, and to record progress by means of a **metric**. The metric is arranged to increase if our latest decision as to the input symbols to date implies (according to the trellis) an output equal to that observed. The metric decreases if observed and implied do not agree. At any instant, with its associated accumulated metric, decoding moves us to the next state allowed by the trellis – the symbols emitted on this move being nearest to those observed. We update the metric, incrementing it if these symbols are identical, decreasing if not. The scale used in these operations is 'soft' not one-or-zero, and depends on the nearness of implied and observed symbols, on the reliability (as indicated by a soft-decision decoder in the demodulator), etc.. A presumed error-rate in the transmission medium allows an average rate of increase of the metric to be calculated. If the calculated metric drops below some moving threshold we assume decoding has gone wrong, and we back-track an appropriate number of states, make a new decision there, and advance from it hopefully more successfully. Clearly there is plenty of scope for creative design, and what works well in one situation may not in another.

Another approach to decoding is to refrain from making a decision as to the input until the length of new received output is well beyond that implied by the constraint length, k. The most recently received symbol is a function of the current input state and certainly k previous states, so the earliest such state cannot be properly estimated until all the subsequent outputs are read. If the code is systematic this opens the possibility of forming a syndrome - the XOR of the check symbols recreated from the possibly corrupted k+ input symbols and the received possibly corrupted check symbols - to help decoding. A suitable circuit based on the syndrome would then confirm or alter the earliest of the k+ input symbols, and now possibly make the syndrome zero. Depending on the distance of the code errors can thus be corrected with no need for further back-tracking.

Where sub-optimal decoding techniques are not good enough we must use Viterbi decoding, and for that various standard ASICs (Application Specific Integrated Circuits) are available.