Decoding Algorithms and Error Probability Bounds for Convolutional Codes

EE 229B - Error Control Coding Project

Hari Palaiyanur

May 13, 2005

1 Introduction

Error control codes are a way of combatting uncertainty in the transmission of information from one point to another. Information theory shows the existence of codes that allow for arbitrarily reliable communication when the rate of transmission is less than the channel capacity. However, it doesn't provide schemes that are practical to achieve this. Error control coding strives to find codes that provide for good performance in terms of error probability, but also low complexity in encoding and decoding.

The first major class of codes designed for this purpose was linear block codes, having been invented in the late 1940's and early 1950's. Linear block codes take a fixed number, k, of input symbols and encode them into a larger, fixed number, n, of output symbols in some linear fashion. Some of the major classes of linear block codes we have discussed in class are Hamming codes, Reed-Muller codes, BCH codes and Reed-Solomon codes. Each of these codes introduces redundancy into a sequence of input symbols in a controlled manner to improve the resistance of the transmitted codewords to noise. However, each block has no dependence on other blocks.

Convolutional codes were first introduced in the mid-1950's as an alternative to block codes. At each time step, a convolutional code takes k input symbols and encodes them into n output symbols. However, we allow for each block to depend on other blocks of k inputs in a manner that can be implemented as a finite state machine. We will assume that each block is allowed to depend only on previous blocks of inputs, and not future blocks of inputs. The use of memory in convolutional codes makes the code structure more complicated than a block code and increases complexity in the encoder by the use of shift registers storing previous inputs, but the additional structure has advantages in decoding.

A convolutional code can be represented by a trellis. The trellis describes the codewords of the code as paths through a directed graph. If we weight the edges of this trellis with log-likelihoods, then maximum-likelihood decoding can

be viewed as the problem of finding the shortest path from the root or start of the trellis to the end. Maximum likelihood (ML) decoding of convolutional codes is accomplished in this manner through the well known Viterbi Algorithm [1].

While the Viterbi Algorithm provides the optimal solution, it may not be practical to implement for certain code parameters. Sequential decoding algorithms, such as the Stack Algorithm and Fano Algorithm can be useful in some cases where the Viterbi algorithm is not practical.

The performance criteria we use when evaluating these various algorithms will be probability of error. In an unterminated convolutional code used over a noisy channel, the probability of error will tend to 1, so instead we will focus on the probability of error per unit time in most cases.

The purpose of this project was to become familiar with several decoding procedures and then evaluate their performance in terms of this probability of error per unit time for randomly generated convolutional codes. In this process, roughly speaking, we come across results concerning convolutional codes using ML decoding, block codes derived from terminated convolutional codes using ML decoding, and convolutional codes using sequential decoding. (The code itself doesn't use the decoding process, but the probability of error depends both on the code and the decoding process chosen.) We will mostly cover results, but since my personal goal for the project was to become familiar with proof techniques used for ML decoding analysis, we give a few detailed proofs of theorems. The main sources from which these results are drawn are Forney's papers on ML decoding [2] and sequential decoding [3] and Zigangirov's book [4] on convolutional codes. For a quick, general introduction to convolutional codes, Lin and Costello's textbook[5] is useful.

2 Description of Convolutional Codes

We begin by describing the convolutional code as Forney [2] does, ultimately leading to the trellis description of convolutional codes. At any time step, the convolutional encoder is assumed to have k inputs which can each take on q values, and n outputs. Hence, there are $M = q^k$ possible inputs at any given time. Perhaps the simplest description of a convolutional code is with a tree. We note that the tree is really a description of the code for a specific encoder.

Figure 1 shows an example of a convolutional code represented as a tree. In this example, at each time step, one input bit comes into the encoder, and two output bits are given out. A node's depth in the tree denotes the time at which the node occurs. A node branches upwards if the input bit is a '0', or downwards if the input bit is a '1'. The outputs of the encoder at time t are the labels of edges connecting nodes at time $t-1$ to nodes at time t. The output bits depend not only on the present input bit, but also the previous input bit. For the code depicted, in the notation of Lin and Costello [5], $G(D) = [1 \quad 1+D]$. This signifies that the first output bit is equal to the input bit at the present time and the second output bit is equal to the sum of the input bits at the

previous time and the present time. So for this code, $q = 2$, $k = 1$, $M = 2$ and $n = 2$.

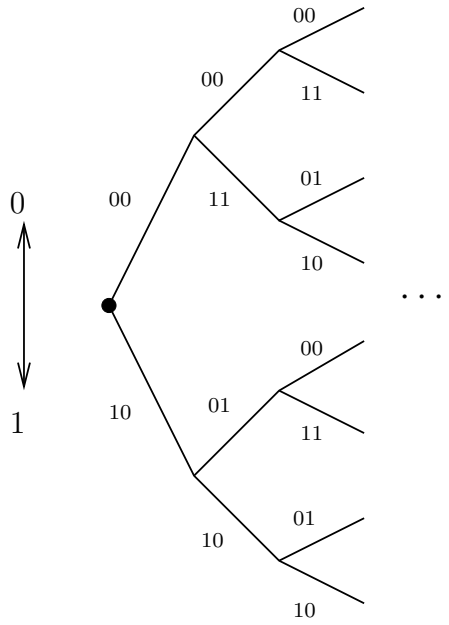

Figure 1: An example of a convolutional code represented by a tree.

In general, at time zero, there is only the root node, at time 1, there are M nodes, at time 2, there are M^2 nodes and so on. Now we note that we have required the convolutional code encoder to be implemented with a finite state machine. Eventually there is a depth in the tree when the number of nodes at that depth exceeds the number of possible states in the encoder. At this depth and afterwards, the tree contains redundant information about the encoder. This is because we again require that future outputs depend only on the state and future inputs. Hence, all we really need to keep track of for purposes of knowing the output of the encoder is the state of the encoder at each time. The code *trellis* is formed from the code tree by merging nodes corresponding to the same encoder state.

Figure 2 shows the trellis for the example in Figure 1. As can be seen, there are two possible states, corresponding to the fact that the present output depends on the previous input as well as the present input. Define ν_i to be the memory of the i^{th} input in the convolutional encoder. Also, define the *overall constraint length* as $\nu_0 = \sum_{i=1}^{k} \nu_i$. We only consider $\nu_i = \nu$ so that $\nu_0 = k\nu$, because in general, it is harder to prove results about convolutional codes in which the memories of the inputs are different. We define the state at time t to be $S_t = (\mathbf{X_{t-1}}, \mathbf{X_{t-2}}, \ldots, \mathbf{X_{t-\nu}})$ where $\mathbf{X_t}$ is the k-tuple of inputs at time t. After time ν, each state in the trellis has M predecessors and M successors.

The M successors of any given state all have the same M predecessors.

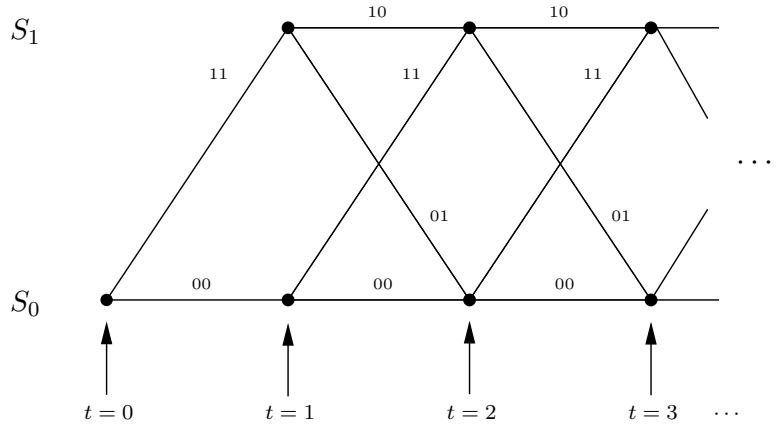

Figure 2: Example of a trellis for a convolutional code.

Next, we note that a block code can be thought of as a trivial block code with no memory. Also, we can form a block code from a convolutional code by terminating it at some time τ. In such a terminated convolutional code, there are a total of $K = k\tau$ input symbols and $N = n\tau$ output symbols. Sometimes, in terminated convolutional codes, it is necessary to end all input sequences with a particular length ν sequence to 'clear' the states of the encoder. In this case, the terminated convolutional code has $M_0 = M^{\tau-\nu}$ total codewords. Below we define the rates of a convolutional code and a terminated convolutional code as r and R respectively. Finally, we note that we can still use a trellis to describe a terminated convolutional code.

$$\begin{aligned} r &= \frac{\ln M}{n} \\ \theta &\triangleq \frac{\nu}{\tau} \\ R &= \frac{1}{N} \ln M_0 \\ &= r(1-\theta) \end{aligned}$$

3 ML Decoding and the Viterbi Algorithm

We will only discuss decoding for the discrete memoryless channel (DMC) model. Figure 3 shows the channel, with input y_i and output z_i. We assume the channel input symbol, y_i, is from the set $\{1, 2, \ldots, L\}$ and the channel output symbol, z_i, is from the set $\{1, 2, \ldots, J\}$. The channel is described by a probability transition

matrix $p_{jl} = Pr(z_i = j | y_i = l)$. Inherent in this notation is the assumption that the channel is time-invariant.

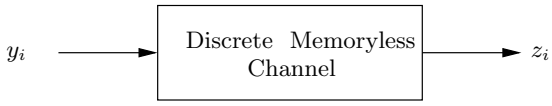

Figure 3: The discrete memoryless channel model.

In a terminated convolutional code, a codeword $\mathbf{y}$ consists of $N = n\tau$ channel inputs, and the received word $\mathbf{z}$ also consists of $N = n\tau$ channel outputs. A maximum-likelihood decoder, by definition, chooses the $\hat{\mathbf{y}} \in \mathfrak{C}$ that makes the received word $\mathbf{z}$ most likely to occur, where $\mathfrak{C}$ is the set of codewords. That is, if $L(\mathbf{y}) = Pr(\mathbf{z}|\mathbf{y})$, an ML decoder chooses the decoded word $\hat{\mathbf{y}}$ as follows.

$$\begin{aligned} \Gamma(\mathbf{y}) &\triangleq -\ln Pr(\mathbf{z}|\mathbf{y}) \\ \hat{\mathbf{y}} &= \arg\max_{\mathbf{y}\in\mathfrak{C}} L(\mathbf{y}) \\ &= \arg\min_{\mathbf{y}\in\mathfrak{C}} \Gamma(\mathbf{y}) \end{aligned}$$

Sometimes, a tie will occur in choosing $\hat{\mathbf{y}}$, in which case we can pick one of the best choices randomly or deterministically. Also, we note that Γ is actually the negative log-likelihood, but we will refer to it freely as the log-likelihood.

The Viterbi Algorithm allows us to find this optimal codeword efficiently. To do this we first note that,

$$\begin{aligned} \Gamma(\mathbf{y}) &= -\ln Pr(\mathbf{z}|\mathbf{y}) \\ &= -\sum_{i=1}^{N} \ln Pr(z_i|y_i) \\ &= -\sum_{t=1}^{\tau} \sum_{i=n(t-1)+1}^{nt} \ln Pr(z_i|y_i) \end{aligned}$$

The total log-likelihood for a potential codeword is the sum of the log-likelihoods of each branch of the codeword. So in the code trellis, we can label each branch with the log-likelihood of channel outputs on that branch. Now, as we go through the trellis for each codeword, we add up these 'lengths' along the way until we go from the start to the end. The problem of finding the optimal codeword for ML decoding now becomes the problem of finding the shortest path through the trellis. The Viterbi Algorithm provides a way to do this and it is summarized below.

1. Assign length 0 to the initial node and set the 'time' $t = 0$.

2. For each node at time $t+1$ find, for each of its predecessors, the sum of the predecessor's length with the length of the branch connecting the predecessor at time t with the node at time $t+1$. Assign the minimum of these sums as the 'length' of the node, and label the node with the shortest path to it from the root. If there is a tie, choose one of the paths randomly.

3. If $t = \tau$, stop. The decoded codeword is the shortest path from the root to a node at time τ. If t is not τ, go to step 2.

A nice feature of the Viterbi Algorithm is that its complexity is proportional to M^ν, which is the number of states in the encoder. Hence, we can increase the length of the terminated code and expect just a linear increase in complexity. This is while increasing the number of codewords exponentially.

Now that we have described the Viterbi Algorithm, we can set up the analysis of probability of error for ML decoding of convolutional codes.

4 Error Events and Random Trellis Codes

An error occurs in the Viterbi algorithm when an incorrect path has a shorter length than the correct path. This happens either by poor design of the code, or because the channel was very noisy and transitioned symbols in an unlikely manner. An *error event* is defined to be any finite period during which the ML path and the correct path are unmerged. The error event starts at time t if the last common node of the correct path and the ML path was at time t in the trellis.

As stated in the introduction, we are not interested particularly in the probability that the an error event occurs, because as the code length increases, this quantity will go to 1. Rather, we are interested in the frequency with which errors occur. That is, we are actually interested in the probability that an error event starts at any particular time. This is different from the probability of bit error, for example, because even though a branch on the ML path may not lie on the correct path, some of the symbols along the two branches may be the same.

A *possible error event* is any path through the code trellis that begins and ends on the correct path and touches it nowhere in between. A possible error event, therefore, is just a portion of the code trellis. An *actual* error event is always a possible error event, but depending upon the realization of the channel outputs, most possible error events will not be actual error events. The minimum length of any possible error event is $\nu + 1$ because the memory of the encoder is ν time steps. Define the *time-t incorrect subset* S_t as consisting of all possible error events that start at time t. Define E_t to be the event that an actual error event starts at time t.

Next, we define the kind of convolutional codes about which we will state and prove a few results. An (M, ν, n) *random trellis code* is code described

by a trellis corresponding to a shift register of length ν, with M-state storage elements, when any M-ary sequence is the input and every channel input (codeword symbol) on every branch is chosen independently and identically at random according to some probability distribution $\mathbf{p} = \{p_k, 1 \leq k \leq L\}$. The rate of such a code is $r = \frac{1}{n} \ln M$ nats/symbol.

A *random* (M, ν, n, τ) *terminated trellis code* is a random (M, ν, n) trellis code in which the last $n\nu$ inputs to the encoder are fixed. The rate for such a code, as discussed before, is $R = r(1 - \frac{\nu}{\tau})$.

These codes are in general, non-linear and time-varying, but just like the channel coding theorem using random block codes, the symmetry given by the random choice of channel inputs will lead to a tractable and usable bound on the probability of error per unit time. We are now in a position to describe some main results and prove a few of them.

5 Error Probability Bounds for ML Decoding

The first theorem is a bound on $Pr(E_t)$ derived by Yudkin and Viterbi.

Theorem 1 *Over the ensemble of random* (M, ν, n) *trellis codes using ML decoding,*

$$Pr(E_t) \leq K_1 \exp[-n\nu E_0(\rho)]$$

where K_1 *is a constant independent of* ν, $E_0(\rho)$ *is Gallager's function [6],*

$$E_0(\rho) = -\ln \sum_j \left[\sum_l p_l p_{jl}^{\frac{1}{1+\rho}} \right]^{1+\rho}$$

and $0 \leq \rho \leq 1$ *and* $\rho < \rho_r$, *where* ρ_r *is related to* r *by* $r = \frac{E_0(\rho_r)}{\rho_r}$.

Since this bound is an average over all codes in the ensemble, it is true for at least one code in the ensemble. Assuming for the moment that the theorem is true, we can quickly prove a corollary about block codes derived from convolutional codes.

Corollary 1 *Let* C *be the capacity of the channel. Then, for* $0 < \theta \leq 1$, $0 \leq r \leq C$ *and* $\epsilon > 0$, *there exists a block code of (large enough) length* N *and rate* $R = r(1 - \theta)$ *such that the probability of block error* $Pr(E)$ *satisfies*

$$Pr(E) \leq K_1 N \exp[-N\theta(e(r) - \epsilon)]$$

Proof: Select parameters n and τ so that $\tau = \frac{N}{n}$. Then, for $\nu = \theta\tau$, select a (M, ν, n) trellis code from the random ensemble which satisfies the bound in Theorem 1. We can select these parameters and avoid any complications with integer effects so that, after taking a union bound of the probability of error per unit time over the $(\tau - \nu)$ information times,

$$\begin{aligned} Pr(E) &\leq (\tau - \nu) K_1 \exp[-n\nu(e(r) - \epsilon)] \\ &\leq N K_1 \exp[-n\nu(e(r) - \epsilon)] \\ &= N K_1 \exp[-N\theta(e(r) - \epsilon)] \end{aligned}$$

□

Now, we give an outline of a proof of Theorem 1. The proof involves several steps starting with a Chernoff bound on the probability of error, followed by 'configuration-counting' and then the use of a bounding procedure found in Gallager [6].

Proof of Theorem 1: We assume a random ensemble of trellis codes as stated in the theorem. Recall the definition of E_t as the event that an actual error event starts at time t. Let $\mathbf{y}$ denote the correct path through the code trellis. Then, E_t happens only if there is a path $\mathbf{y}'$ in S_t that re-merges with $\mathbf{y}$ at some time $t + \kappa$, $\kappa > \nu$ AND

$$-\ln[Pr(\mathbf{z}|\mathbf{y}')]_t^{t+\kappa} \leq -\ln[Pr(\mathbf{z}|\mathbf{y})]_t^{t+\kappa}$$

where the indices on the brackets indicate restriction of the paths to within those times. Now define

$$\Gamma(\mathbf{y}') \triangleq \ln \left[\frac{Pr(\mathbf{z}|\mathbf{y}')}{Pr(\mathbf{z}|\mathbf{y})} \right]_t^{t+\kappa}$$

Restating in terms of Γ, E_t occurs only if there is some $\mathbf{y}'$ in S_t for which $\Gamma(\mathbf{y}') \geq 0$. Now consider the following quantity,

$$T_t(\alpha, \rho) \triangleq \left[\sum_{\mathbf{y}' \in S_t} \exp[\alpha \Gamma(\mathbf{y}')] \right]^{\rho} \qquad 0 \leq \alpha,\ 0 \leq \rho \leq 1$$

It can be seen that if $\Gamma(\mathbf{y}') \geq 0$ for at least one $\mathbf{y}'$ in S_t, then $T_t(\alpha, \rho) \geq 1$. Hence, we can use T_t to bound $Pr(E_t)$. That is,

$$Pr(E_t) \leq E\left[T_t(\alpha, \rho) \right]$$

where the expectation is taken over all randomness in the codes and the channel. Next, we need to bound the number of paths that can diverge at time t and re-merge at time $t + \kappa$. Hence we are interested in the number of possible error events in the *configuration*, as Forney calls it, of paths that diverge at time t and re-merge at time $t + \kappa$. Define $C_{t\kappa}$ to be the set of all possible error events starting at time t and ending at time $t + \kappa$. Figure 4 shows a typical element of $C_{t\kappa}$ along with the correct path. Since the last ν time steps' inputs must be the same as those of the correct path to re-merge with it, a bound on the size

of the configuration is $|C_{t\kappa}| \leq M^{\kappa-\nu} = e^{nr(\kappa-\nu)}$. Now we can partition the $\mathbf{y}'$ in S_t into the $\mathbf{y}'$ in each $C_{t\kappa}$, for $\kappa > \nu$.

Applying Jensen's inequality ($(\sum_i a_i)^\rho \leq \sum_i (a_i)^\rho$, for $\rho \leq 1$) to T_t, we get

$$\begin{aligned} T_t(\alpha,\rho) &= \left[\sum_{\kappa=\nu+1}^{\infty} \sum_{\mathbf{y}'\in C_{t\kappa}} \exp[\alpha\Gamma(\mathbf{y}')]\right]^\rho \\ &\leq \sum_{\kappa=\nu+1}^{\infty} \left[\sum_{\mathbf{y}'\in C_{t\kappa}} \exp[\alpha\Gamma(\mathbf{y}')]\right]^\rho \\ T_{t\kappa}(\alpha,\rho) &\triangleq \left[\sum_{\mathbf{y}'\in C_{t\kappa}} \exp[\alpha\Gamma(\mathbf{y}')]\right]^\rho \\ P(E_t) &\leq \sum_{\kappa=\nu+1}^{\infty} E\left[T_{t\kappa}(\alpha,\rho)\right] \end{aligned}$$

Then, by some algebraic manipulations similar to the channel coding theorem proof of [6] and setting $\alpha = \frac{1}{1+\rho}$, we get the following lemma.

Lemma 1

$$E\left[T_{t\kappa}(\alpha,\rho)\right] \leq |C_{t\kappa}|^\rho \exp[-n\kappa E_0(\rho)] \qquad 0 \leq \rho \leq 1$$

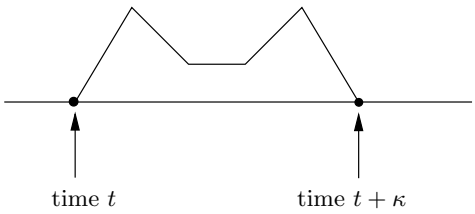

Figure 4: The configuration of the correct path and the configuration of a possible error event of length κ starting at time t.

To finish the proof we have,

$$\begin{aligned} Pr(E_t) &\leq \sum_{\kappa=\nu+1}^{\infty} E\left[T_{t\kappa}(\alpha,\rho)\right] \\ &\leq \sum_{\kappa=\nu+1}^{\infty} |C_{t\kappa}|^{\rho} \exp[-n\kappa E_0(\rho)] \\ &\leq \sum_{\kappa=\nu+1}^{\infty} M^{(\kappa-\nu)\rho} \exp[-n\kappa E_0(\rho)] \\ &\leq \sum_{\kappa=\nu+1}^{\infty} e^{nr(\kappa-\nu)\rho} \exp[-n\kappa E_0(\rho)] \\ &\leq K_1 \exp[-n\nu E_0(\rho)] \\ &\quad \text{for } r < \frac{E_0(\rho)}{\rho} \\ K_1 &= \frac{\exp[-n(E_0(\rho)-\rho r)]}{1-\exp[-n(E_0(\rho)-\rho r)]} \end{aligned}$$

Hence, Theorem 1 says that the probability or error per unit time is exponentially decreasing in constraint length. This shows explicitly that the increased complexity in the structure of the code helps in decoding of the code.

A few more bounds on error probability for ML decoding are stated before we move on to sequential decoding algorithms.

Theorem 2 *For any probability vector on channel inputs* $\mathbf{p}$ *and any* $\epsilon > 0$, *there exists a block code of length* N *and rate* R *with block error probability,* $Pr(E)$, *so that*

$$Pr(E) \leq K_1 N \exp[-N(E(R)-\epsilon)]$$

where $E(R) = \max_{0\leq\rho\leq 1} E_0(\rho) - \rho R$.

The random block coding error exponent, $E(R)$, is known to be tight for rates close to channel capacity. The proof of Theorem 2 also uses terminated convolutional codes. Interestingly, Theorem 2 says that one can get an optimal block code from an appropriately terminated convolutional code.

The last sample result concerns bounded delay decoding of convolutional codes.

Theorem 3 *If code symbols at time* t *must be decided upon by time* $t+\kappa$, *then the additional probability of error caused by deciding early is*

$$Pr(E_a) \leq exp[-n\kappa E(r)]$$

where $E(r)$ *is the block code exponent evaluated at the rate of the convolutional code.*

This result comes essentially from the proof of the channel coding theorem with block codes. An incorrect symbol decision at time t would lead the Viterbi decoder on a path which is statistically independent of the correct path, and so the analysis of Gallager[6] can be used, with the block length equal to $n\kappa$.

6 Sequential Decoding and the Stack Algorithm

In the Viterbi Algorithm, all paths through the code trellis are examined to make an ML decision on the sent codeword. For larger constraint lengths, which we saw led to lower error probability, this can be inefficient because the complexity of the Viterbi decoder is exponential in ν. Sequential decoding algorithms were developed heuristically before the Viterbi Algorithm was recognized to give the ML decision. The idea is to search paths on a branch by branch basis, following the best path first, in order to avoid searching all paths. The 'metric' or 'length' on the branches can be the log-likelihoods of the branch along with some bias. The bias essentially adjusts how aggressive the sequential decoder is in searching deep into the code tree. The choice of metric affects the computation time of the algorithm as well as its probability of error. The algorithm is done when the next path to be extended is at the terminating level. First, let us state the definition of sequential decoder given by Jacobs and Berlekamp [7].

A decoder is *sequential* only if it computes a subset of path metrics or lengths in a sequential fashion, with each new path examined being an extension of a previously examined path and the decision of which path to extend based only on previously examined paths. In other words, a decoder is sequential if it searches through the code tree going to nodes that are direct descendants of previously examined nodes. The decision of which node to visit must be made on the information collected about already visited nodes. It is possible, however, for a sequential decoder to visit a node more than once.

There are many sequential algorithms based on various heuristics; two main ones being the Fano Algorithm and the Stack Algorithm. The Stack Algorithm is simpler to describe, but because it requires sorting of a list of numbers on each iteration, the Fano Algorithm is preferred in many situations. The Stack Algorithm was discovered independently by Zigangirov in 1966 and Jelinek in 1969. At a given time, assume there are N partial paths in the stack and $\mathbf{y}'$ is the one with the best metric. The Stack Algorithm is summarized below.

1. $N = 1$, $\mathbf{y}'$ is the origin node.

2. Compute the metrics of the M successors of $\mathbf{y}'$ as the sum of the branch metric and the metric for $\mathbf{y}'$. Place each of the M partial paths onto the stack, removing $\mathbf{y}'$. Update N.

3. Sort the stack based on the partial path metrics. The best partial path is once again $\mathbf{y}'$. If $\mathbf{y}'$ is at the terminating depth, then the algorithm is done and $\mathbf{y}'$ is the decoded codeword. Otherwise, return to 2 and iterate again.

One variation on the Stack Algorithm is called the Stack-Bucket Algorithm (SBA). In the SBA, rather than one stack, there are buckets that hold all the partial paths with metrics confined to a certain range. Instead of sorting the paths in a bucket, the bucket is used as a true 'push-pop' stack without sorting at each time. If the granularity of the buckets is small enough, then there is very little difference in the performance of the Stack Algorithm and the SBA.

Another variation, which Forney[3] calls Algorithm A, takes merging into account, by combining paths corresponding to the same state in the code trellis.

Before stating results about sequential decoding, we note that the amount of computational effort expended by a sequential decoder is a random variable depending on the codeword sent and the channel transitions. Much of the research into sequential decoders has been into bounding the computational effort. It turns out that the biasing term used in the metrics for each branch will affect this computational effort and the probability of error. There is a tradeoff between achieving a low probability of error and low computational effort. The effect of the biasing term on these two performance parameters was investigated by Jelinek[8].

7 Bounds for Sequential Decoding

First, a bound on probability of error is stated and then a lower bound on the computational distribution.

Theorem 4 *Let ρ_r related to r as $r = \frac{E_0(\rho_r)}{\rho_r}$. Then, on any DMC, for $0 \leq \rho < \min(\rho_r, 1)$ and using Algorithm A, with the appropriately chosen bias, the probability of error per unit time satisfies*

$$Pr(E_t) \leq K \exp[-n\nu\rho r]$$

where K is a constant independent of ν.

The proof of this result is similar to the proof of Theorem 1, with an adjustment of the quantity T_t being bounded. By the properties of Gallager's function E_0, if $0 \leq \rho_r \leq 1$, then $E_0(1) = R_{comp} \leq r \leq C$ and as ρ goes to ρ_r, we have asymptotically the same result in Theorem 1 as in this theorem about sequential decoding. That is, for the high rate region, sequential decoding is asymptotically as good as ML decoding.

The last result stated is one about the random variable of computation.

Theorem 5 *Let C be the number of computations performed by any sequential decoder. Then $P[C > L] \simeq L^{-\rho}$ where ρ is a parameter that depends on both the channel and the rate of the code. That is, the distribution of the random variable of computation is lower bounded by a Pareto distribution.*

The impact of this theorem, proved by Jacobs and Berlekamp[7], is that there always exists a moment of computation that is infinite. Hence, there is some unavoidable probability that the sequential decoder will perform extremely long searches and run into a buffer overflow in a real system.

8 Conclusion

In this survey paper, we first described a convolutional code as a directed graph called a trellis, or a tree code. Then, we looked at the problem of ML decoding over DMC channels and the Viterbi Algorithm as a solution to this problem. We stated some results about probability of error per unit time for random trellis codes and gave one detailed proof. We stated the result that one can get an optimum block code from a properly terminated convolutional code. Then we explained sequential decoding, the Stack Algorithm, and a few variants. Finally, we stated a few results about sequential decoders, namely that for certain rates they can be as 'good' as ML decoders, and that they are always 'bad' in the sense that there exist moments of the random variable of computation that are infinite.

I would like to thank Professor Anantharam for giving me and the rest of the class the opportunity in the project to look into an area of coding theory in depth when we didn't have time to cover it in class.

References

[1] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Information Theory*, IT-13:260-69, April 1967

[2] G.D. Forney Jr., "Convolutional Codes II: Maximum Likelihood Decoding," *Information and Control,* 25:222-66, July 1974.

[3] G.D. Forney Jr., "Convolutional Codes III: Sequential Decoding," *Information and Control,* 25:267-97, July 1974.

[4] R. Johannesson and K. Zigangirov, *Fundamentals of Convolutional Coding,* Wiley-IEEE, New York, 1999.

[5] S. Lin and D.J. Costello Jr., *Error Control Coding*, Second Edition, Prentice Hall, 2004.

[6] R.G. Gallager, *Information Theory and Reliable Communication,* John Wiley, New York, 1968.

[7] I.M. Jacobs and E.R. Berlekamp, "A Lower Bound to the Distribution of Compuation for Sequential Decoding," *IEEE Trans. Information Theory*, IT-13, 167-74.

[8] F. Jelinek, "Upper Bounds on Sequential Decoding Performance Parameters," IEEE Trans. Information Theory, IT-20, 227-239.