# Channel coding

# Channel coding scheme is used to offer a combination of error detection and error correction for the transmitted bits.

# The Usage of channel coding scheme and coding rate in LTE is shown in the figure below:

# 

Figure 1: Usage of channel coding scheme and coding rate as defined in the 3GPP 36.212 standard

# Tail biting Convolutional Encoder:

A tail biting convolutional code with constraint length 7 and coding rate 1/3.

The configuration of the convolutional encoder is presented in the figure below:

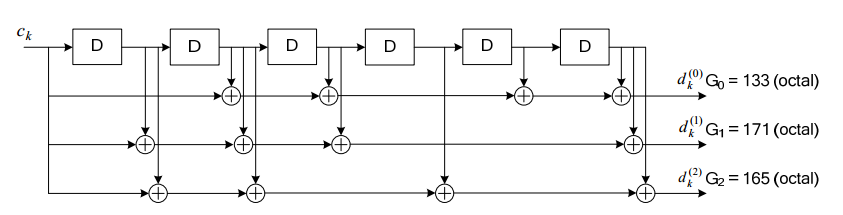


Figure 2: Configuration of convolutional encoder as defined in the 3GPP 36.212 standard

The convolutional encoder is called “**tail biting”** because the initial value of the shift register of the encoder shall be set to the values corresponding to the last 6 information bits in the input stream so that the initial and final states of the shift register are the same.

For every entering bit the encoder emits 3 bits depending on a polynomial defined in the standard as shown in the previous figure.

By denoting the shift registers of the encoder with , then the initial value of the shift register shall be set to

The encoder output streams correspond to the first, second and third parity streams, respectively.

# Decoding of convolutional encoding:

Several algorithms exist for decoding convolutional codes. For relatively small values of k, the **Viterbi algorithm** is universally used as it provides maximum likelihood performance and is highly **parallelizable**. Viterbi decoders are thus easy to implement in software on CPUs with SIMD instruction sets.

Longer constraint length codes are more practically decoded with any of several sequential decoding algorithms, of which the “Fano algorithm” is the best known.

The Viterbi algorithm is very suitable for the requirements of the project for its parallelization capability.

# Viterbi overview:

The Viterbi algorithm is best illustrated using the trellis diagram. The trellis diagram is just a way to show the transition from one state to another state when the input to the convolutional encoder during time evolution. First, we will show the trellis diagram when encoding the bits.

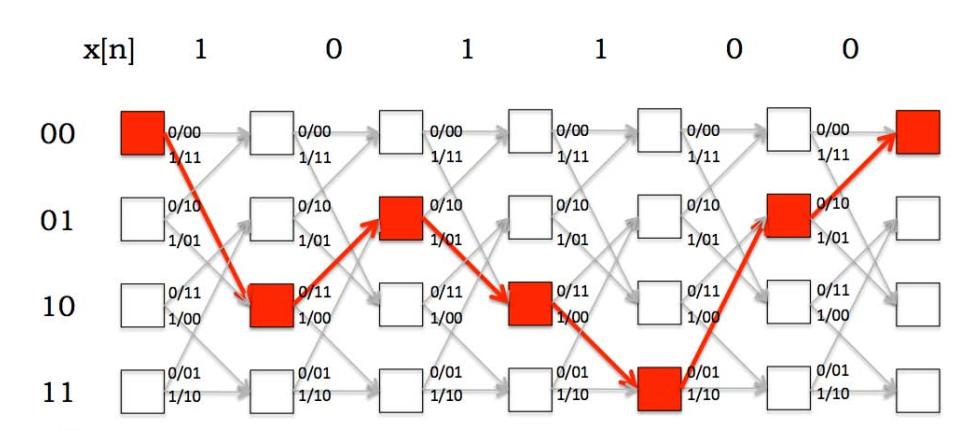
The trellis diagram is shown in the figure below:

Figure 3: A trellis diagram showing the time evolution of the state machine.

In the figure above the x[n] represents the input to the convolutional encoder at each time clock.

The trellis diagram above corresponds to a convolutional encoder with 2 shift registers and coding rate =

Each box represents one of the states for the shift registers. Therefore, we have 4 states which mean 4 vertical boxes. When the input to the convolutional encoder is “1’, the state in the 1st stage will change from “00” to “10”. The input then completes forming a path and emitting a codeword of 2 bits at each transition.

At the receiver, suppose we now receive a sequence of bits representing the output from the encoder. Then, there will be some path through the trellis that would exactly match the received sequence. But, when there are bit errors, we need to find the *most likely* transmitted message sequence. If we can come up with a way to capture the errors introduced by going from one state to the next, then we can accumulate the errors along a path and come up with an estimate of the total number of errors along the path. Then, the path with the smallest such accumulation of errors is the path we want and it represents the most likely transmitted message.

The Viterbi decoder solves these problems by computing the cost (errors) when going from one state to another. Then, the costs are accumulated and the path with the least cost represents the most likely transmitted message.

# Viterbi algorithm Parameters:

The decoding algorithm has two metrics: the **branch metric (BM)** and the **path metric (PM)**. The branch metric is a measure of the "distance" between what was transmitted and what was received, and is defined for each arc in the trellis. In hard decision decoding, where we are given a sequence of digitized parity bits, the branch metric is the *Hamming distance* between the expected parity bits and the received ones.

Path metric is defined for hard decision and it corresponds to the Hamming distance with respect to the received parity bit sequence over the most likely path from the initial state to the current state in the trellis. By “most likely”, we mean the path with smallest Hamming distance between the initial state and the current state, measured over all possible paths between the two states.

# Steps of Viterbi algorithm:

1. Calculating the branch metric for all arcs in the trellis.

As stated previously we will calculate the hamming distance between both the received sequence and all possible values coming out from the Viterbi decoder at all states.

1. ACS calculation (**A**dd, **C**ompare, and **S**hift).

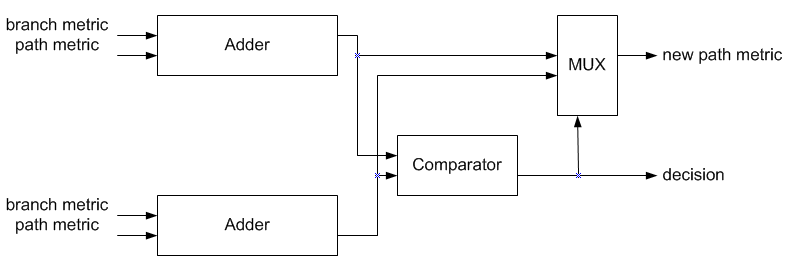


Figure 2: Block diagram of ACS

As shown in the figure 2 we can sum up the ACS algorithm in each stage in the trellis as follows:

**Add** the predecessor state path metric to the current state branch metric for both predecessor states. **Compare** between the cost of the two addition results. By "cost", we mean the Hamming distance. **Select** the minimum cost to be the new path metric for the current state which will be predecessor in the next iteration.

Mathematically, at any state 'S' at time 'N':

PM[S, N]: path metric at state S at time N.

α and β are the two predecessor states.

1. Finding the most likely path.

After all data have been processed, we must find the most likely path through the trellis done by searching through the final “column” in PM and selecting the most likely (lowest cost) operation done infrequently compared to other calculations.

1. Traceback and Estimating the output.

In the 4th step we will find the total path from the final minimum cost state that we have already found it in the 3rd step to the first state.

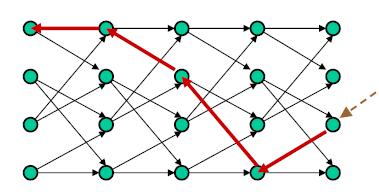


Figure 3: Traceback on trellis

We can perform traceback by:

* Read the Path metric of the both predecessor states
* Select the minimum and add the new backward state into our traceback path.
* By comparing both the previous state and the current state we can guess the input bit.

After that we can reorder both the output bits and the traceback states.