Théorie des codes - TP 4
ZZ3 F5 - Réseaux et Sécurité Informatique
Codes convolutionnels : Correction d'erreurs en GSM

Les codes convolutionnels $\mathcal{C}(n,k,r)$ code une trame de k bits par un bloc de n bits, mais le codage fait intervenir également la donnée des r trames précédentes (différence avec le codage par bloc). Une fonction logique permet de calculer les n bits de sortie à partir des $(k \times (r+1))$ bits dans les registres.
Le décodage peut se réaliser grâce à l'algorithme de Viterbi, d'Andrew Viterbi, qui permet de corriger des erreurs survenues lors d'une transmission à travers un canal bruité. Cet algorithme a pour but de trouver la séquence d'états la plus probable ayant produit la séquence mesurée.

A. Viterbi

Code convolutionnel $\mathcal{C}(2,1,4)$

GSM (*Global System for Mobile Communication*) est une norme élaborée au cours des années 80 et 90 qui est utilisée pour les réseaux de communication sans fil à travers le monde. Le codage du canal en GSM consiste à ajouter des bits de redondance à l'information de manière à détecter et corriger si possible les erreurs. Ce codage varie en fonction du canal auquel l'information est destinée. Nous allons considérer un signal de voix destiné à un canal dit de voix. Avant d'être traité par le codage du canal, les bits de la trame de voix (260 ici) sont divisés en 3 classes selon leur fonction et leur importance. Par des tests subjectifs, on a établi que certains bits sont plus important pour la perception de la qualité de la voix que d'autres :

— Classe Ia 50 bits : les plus sensibles aux erreurs de bit ;
— Classe Ib 132 bits : sensibilité modérée aux erreurs de bit ;
— Classe II 78 bits : sensibilité la plus faible aux erreurs de bit.

Trois bits de parité sont ajoutés aux bits de la classe Ia. Ces 53 bits sont ajoutés à la classe Ib. À ces 185 bits, on ajoute 4 bits mis à 0. On applique un code de convolution $\mathcal{C}(2,1,4)$ (figure 1) pour obtenir un bloc de sortie de 378 bits. On ajoute les bits de la Classe II pour obtenir un bloc de 456 bits. Donc, pour chaque 20 ms de voix, on obtient 456 bits, soit un débit de 22,8 Kbps. Dans ce TP, nous allons nous intéresser uniquement au codage convolutionnel.

1. Réalisez une fonction permettant le décodage du code convolutionnel par l'algorithme de Viterbi (cf. Annexe).

2. Réalisez une fonction permettant de simuler le canal de transmission en rajoutant aléatoirement des erreurs.

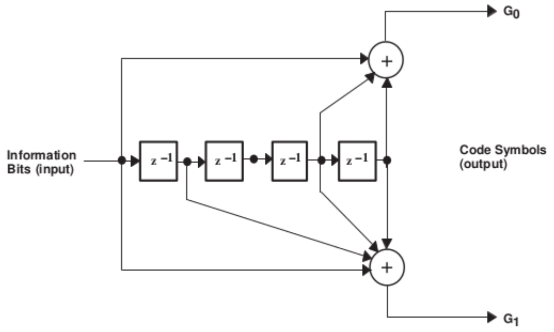

FIGURE 1 – Code convolutionnel utilisé dans la norme GSM

Appendix Convolutional Encoding and Viterbi Decoding

Convolutional Versus Block-Level Coding[1] :

Convolutional coding is a bit-level encoding technique rather than block-level techniques such as Reed-Solomon coding. Advantages of convolutional codes over block-level codes for telecom/datacom applications are :

— With soft-decision data, convolutionally encoded system gain degrades gracefully as the error rate increases. Block-level codes correct errors up to a point, after which the gain drops off rapidly ;

— Convolutional codes are decoded after an arbitrary length of data, while block-level codes introduce latency by requiring reception of an entire data block before decoding begins ;

— Convolutional codes do not require block synchronization.

Although bit-level codes do not allow reconstruction of burst errors like block-level codes, interleaving techniques spread out burst errors to make them correctable. Convolutional codes are decoded by using the trellis to find the most likely sequence of codes. The Viterbi algorithm simplifies the decoding task by limiting the number of sequences examined. The most likely path to each state is retained for each new symbol.

Encoding Process :

Convolutional encoder error-correction capabilities result from outputs that depend on past data values. Each coded bit is generated by convolving the input bit with previous uncoded bits. An example of this process is shown in figure 1. The information bits are input to a shift register with taps at various points. The tap values are combined through a boolean XOR function (the output is high if one and only one input is high) to produce output bits.

Error correction is dependent on the number of past samples that form the code symbols. The number of input bits used in the encoding process is the constraint length and is calculated as the number of unit delays plus one. In Figure 1, there are four delays. The constraint length is five. The constraint length represents the total span of values used and is determined regardless of the number of taps used to form the code words. The symbol r represents the constraint length. The constraint length implies many system properties ; most importantly, it indicates the number of possible delay states.

Coding Rate :

Another major factor influencing error correction is the coding rate, the ratio of input data bits to bits transmitted. In figure 1, two bits are transmitted for each input bit for a coding rate of 1/2. Although any coding rate is possible, rate $1/n$ systems are most widely used due to the efficiency of the decoding process. The output-bit combination is described by a polynomial. The system, as shown in figure 1, uses the polynomials :

$$G_0(x) = 1 + x^3 + x^4 \quad G_1(x) = 1 + x + x^3 + x^4$$

Polynomial selection is important because each polynomial has different error-correcting properties. Selecting polynomials that provide the highest degree of orthogonality maximizes the probability of finding the correct sequence.

Decoding Process :

Convolutionally encoded data is decoded through knowledge of the possible state transitions, created from the dependence of the current symbol on past data. The allowable state transitions are represented by a trellis diagram.

1. extrait de "Viterbi Decoding Techniques for the TMS320C55x DSP Generation", Texas Instruments, Henry Hendrix, Application Report SPRA776A, April 2009

The delay states represent the state of the encoder (the actual bits in the encoder shift register), while the path states represent the symbols that are output from the encoder. Each column of delay states indicates one symbol interval. The number of delay states is determined by the constraint length. The number of possible states is $2r - 1$. Knowledge of the delay states is very useful in data decoding, but the path states are the actual encoded and transmitted values.

The number of bits representing the path states is a function of the coding rate. Since path states represent the actual transmitted values, they correspond to constellation points, the specific magnitude and phase values used by the modulator. The decoding process estimates the delay state sequence, based on received data symbols, to reconstruct a path through the trellis. The delay states directly represent encoded data, since the states correspond to bits in the encoder shift register.

Viterbi Algorithm and Trellis Paths :
The Viterbi algorithm provides a method for minimizing the number of data-symbol sequences (trellis paths). As a maximum-likelihood decoder, the Viterbi algorithm identifies the code sequence with the highest probability of matching the transmitted sequence based on the received sequence. The Viterbi algorithm is composed of a metric update and a traceback routine. In the metric update, probabilities are accumulated for all states based on the current input symbol. The traceback routine reconstructs the data once a path through the trellis is identified. A brief pseudo-code sequence of the major steps for the Viterbi algorithm is :

Algorithm 1 Pseudo Code for the Viterbi Algorithm

```
 1: for each frame (block message transmitted) do
 2:    Initialize metrics
       Metric Update or Add-Compare-Select
 3:    for each delay state do
 4:       Calculate local distance of input to each possible path
 5:       Accumulate total distance for each path
 6:       Select and save minimum distance
 7:       Save indication of path taken
 8:    end for
       Traceback
 9:    for each bit in a frame (or for minimum # bits) do
10:       Calculate position in transition data of the current state
11:       Read selected bit corresponding to state
12:       Update state value with new bit
13:    end for
14:    Reverse output bit ordering
15: end for
```