Nombre y Apellidos: SALVADOR ESTRÁN GEA

EXPERTO EN TEORÍA DE LA INFORMACIÓN Y LA CODIFICACIÓN
TRABAJO FINAL

CÓDIGOS CONVOLUCIONALES

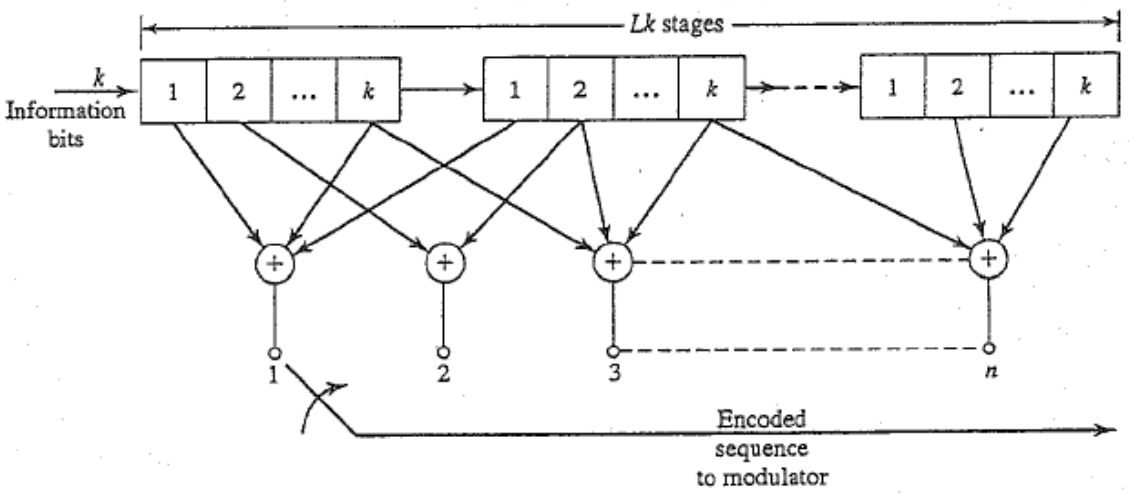
1.- Estructura de los códigos lineales convolucionales

Los códigos convolucionales son códigos lineales al igual que los códigos bloque (como los códigos de Hamming, por ejemplo) y, por tanto, se utilizan para proteger la información añadiendo redundancia a la misma, de manera que las palabras del código tengan la distancia mínima necesaria.

Sin embargo, a diferencia de los códigos bloque, las palabras de un código convolucional se generan no sólo a partir de los dígitos de información actuales sino también con la información anterior en el tiempo. Es decir, un codificador convolucional es un sistema con memoria y, en consecuencia, lleva asociada una cadena de Markov aunque ésta no es visible en la salida pero sí la condiciona, como se mostrará más adelante.

Dado que los sistemas que codifican y decodifican la información son digitales en la práctica, en adelante nos centraremos en los códigos convolucionales **binarios** y hablaremos de ***bits*** para referirnos a los dígitos binarios.

En la figura siguiente se muestra el diagrama de bloques genérico de un codificador convolucional, en el que cada secuencia de entrada de k bits es mapeada en una secuencia de salida de n bits.



Según se puede apreciar, un codificador convolucional está formado por un registro de desplazamiento de $L \cdot k$ bits, donde L es la **longitud limitada** u obligada del codificador, que se conecta a tantos sumadores módulo 2 como bits de salida se deseen. La respuesta del codificador queda determinada por la manera en que cada uno de los n sumadores calcula los respectivos bits de salida.

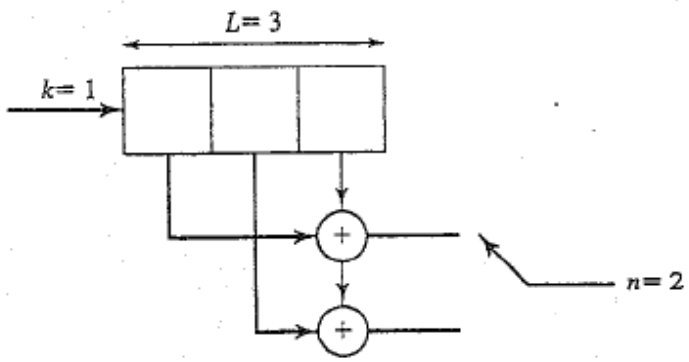
También se observa que un codificador convolucional memoriza las L-1 secuencias de k bits anteriores a la secuencia actual. Por tanto, L-1 representa el **orden de memoria** del codificador. Asimismo, al igual que en los códigos bloque, la relación k/n es la **tasa** o cadencia del codificador.

En resumen, todo código convolucional queda definido por la terna [n, k, L].

2.- Secuencias generadoras y representación polinómica

A diferencia de los códigos bloque, en los códigos convolucionales no existe una matriz de generadores para construir las palabras código sino n secuencias generadoras que caracterizan la salida. Veámoslo con un ejemplo.

Sea el código convolucional [2,1,3] generado con el codificador que se muestra en la siguiente figura:



En este caso, cada bit de entrada es mapeado en 2 bits n_1 y n_2 que se entregan a la salida de forma secuencial (en serie) de manera que, si cada bit entra en el codificador en un instante de tiempo t resulta que:

- n_1 es la suma de los bits de entrada en los instantes t y t-2
- n_2 es la suma de los bits de entrada en los instantes t, t-1 y t-2

Dicha relación se expresa mediante las secuencias g_1 = (**1 0 1**) y g_2 = (**1 1 1**) las cuales reciben el nombre de **secuencias generadoras** del código.

Las secuencias generadoras también se pueden expresar en **forma polinómica** asociando un polinomio generador a cada una de las n salidas de los sumadores.

Así, en nuestro ejemplo resulta:

$$g_1(X) = 1 + X^2$$
$$g_2(X) = 1 + X + X^2$$

La representación polinómica permite obtener fácilmente las palabras código multiplicando la entrada (en forma polinómica) por cada uno de los polinomios generadores de la salida.

Supongamos que queremos conocer, en nuestro codificador ejemplo, la palabra código asociada a la cadena de entrada k = (**1 1 0 0**). Para ello, calculamos los productos[1]:

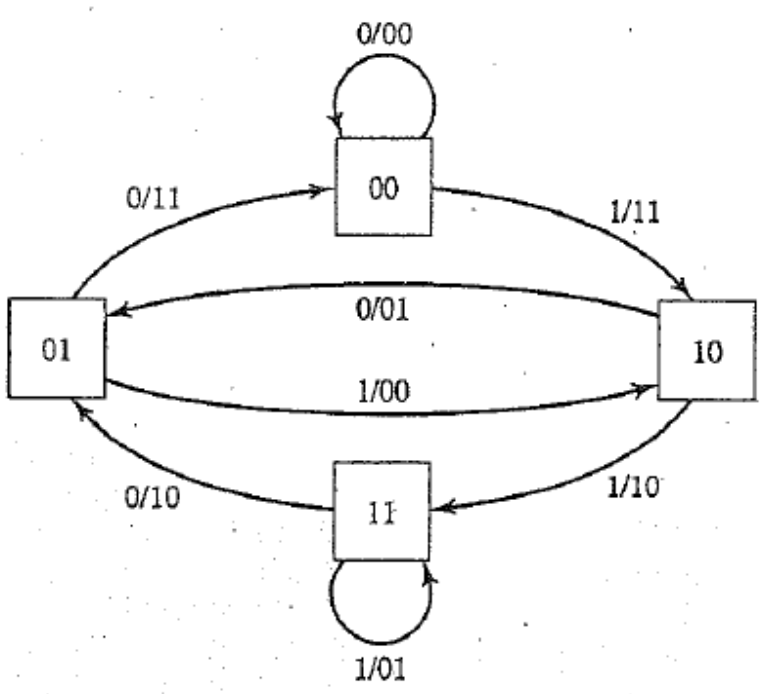
$$n_1(X) = k(X)\cdot g_1(X) = (1+X)\cdot(1+X^2) = 1+X+X^2+X^3$$
$$n_2(X) = k(X)\cdot g_2(X) = (1+X)\cdot(1+X+X^2) = 1+X^3$$

Por tanto, la salida es n_1 = (**1 1 1 1**) y n_2 = (**1 0 0 1**) que al ser entregada en serie genera la palabra código n = (**11 10 10 11**).

3.- Diagrama de estados

Tal y como se ha mencionado en el punto 1, todo codificador convolucional memoriza los $(L\text{-}1)\cdot k$ bits anteriores a la secuencia actual (de k bits). Por ello, el codificador se comporta como una máquina de estados finitos cuya entrada condiciona (junto con el estado presente) tanto la salida como el estado futuro. El número de estados posibles de dicha máquina es precisamente $2^{(L\text{-}1)\cdot k}$.

En la siguiente figura se muestra el diagrama de estados del codificador convolucional [2,1,3] que se ha usado en el punto 2 como ejemplo:



En dicho diagrama, las cajas representan los estados del codificador y las flechas indican las transiciones posibles entre estados, de manera que cada flecha parte del estado

[1] La cadena de entrada debe finalizar con dos ceros para dejar la memoria del codificador en el estado inicial. Para el cálculo se debe tener en cuenta que la suma es módulo 2.

presente y apunta hacia el estado futuro según el bit de entrada. Sobre cada flecha se indica el bit de entrada (a la izquierda de la barra) y los dos bits de salida correspondientes (a la derecha de la barra).

Obsérvese cómo el número de flechas que parten de un estado determinado es 2^k e igual al número de flechas que llegan a cada estado (dos en este caso).

4.- Respuesta impulsiva y función de transferencia

Desde el punto de vista de la teoría de señales y sistemas, un codificador convolucional es un sistema lineal en invariante en el tiempo (LTI), discreto y con memoria. Por este motivo, el código de salida $y[t]$ del codificador cumple la expresión:

$$y[t] = x[t] * h[t]$$

donde $x[t]$ son los k bits de entrada, $h[t]$ es la **respuesta impulsiva** del sistema y la operación "*****" representa la ***convolución*** temporal de ambos. De ahí, que el código generado por el sistema reciba el nombre de convolucional.

La teoría de sistemas también nos enseña que la igualdad anterior puede expresarse en la forma transformada, en este caso aplicando la transformada z (por ser un sistema discreto) obteniendo:

$$Y(z) = X(z) \cdot H(z) \text{ o, lo que es lo mismo: } H(z) = \frac{Y(z)}{X(z)}$$

donde $H(z)$ recibe el nombre de **función de transferencia** del sistema. Veamos estos conceptos para el caso particular de nuestro codificador [2,1,3] ejemplo.

La repuesta impulsiva se puede hallar fácilmente sin más que observar cuál es la secuencia de salida cuando introducimos un impulso unidad en la entrada.

El impulso unidad está formado por una cadena de k bits en la que todos son cero menos el primero que ha de ser un uno. Dado que el codificador parte del estado **0 0 0 … 0**, es preciso completar la cadena anterior con los $(L\text{-}1)\cdot k$ ceros necesarios para dejar la memoria del codificador en el estado inicial.

Por tanto, en nuestro ejemplo, el impulso unidad es la cadena (**1 0 0**), la cual, al ser introducida en el diagrama de estados del punto 3, produce la transición de estados $00 \rightarrow 10 \rightarrow 01 \rightarrow 00$ que genera la secuencia de salida (**11 01 11**).

La respuesta impulsiva también se puede obtener a partir de las secuencias generadoras g_1 = (**1 0 1**) y g_2 = (**1 1 1**) vistas en el punto 2, alternando un bit de g_1 con un bit de g_2.

Para hallar la función de transferencia, aplicaremos la transformada z a cada una de las salidas del sistema.

$$y_1[t] = x[t] + x[t-2] \rightarrow Y_1(z) = X(z) + z^{-2} \cdot X(z)$$
$$y_2[t] = x[t] + x[t-1] + x[t-2] \rightarrow Y_2(z) = X(z) + z^{-1} \cdot X(z) + z^{-2} \cdot X(z)$$

Por tanto, por cada salida obtenemos la función de transferencia:

$$H_1(z) = 1 + z^{-2} = \frac{z^2 + 1}{z^2}$$
$$H_2(z) = 1 + z^{-1} + z^{-2} = \frac{z^2 + z + 1}{z^2}$$

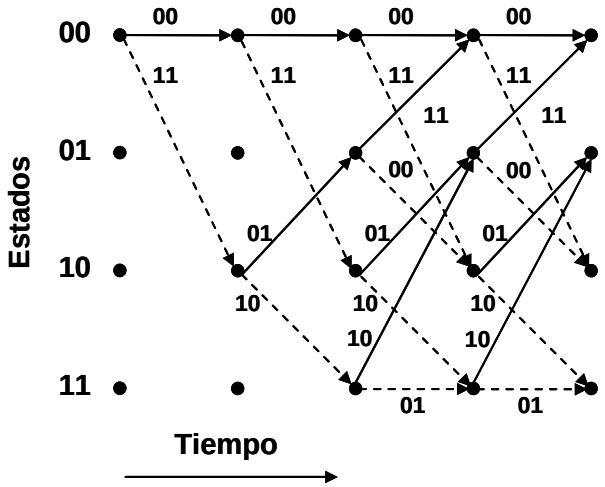
Obsérvese la analogía entre las funciones anteriores y las secuencias generadoras en forma polinómica vistas en el punto 2.

5.- Codificación: diagrama de "Trellis"

El diagrama de estados proporciona una buena información sobre las distintas transiciones de la máquina de estados del codificador, pero no permite ver la evolución de dichas transiciones ni sus salidas asociadas con el paso del tiempo, es decir, a medida que van llegando cadenas de k bits al codificador.

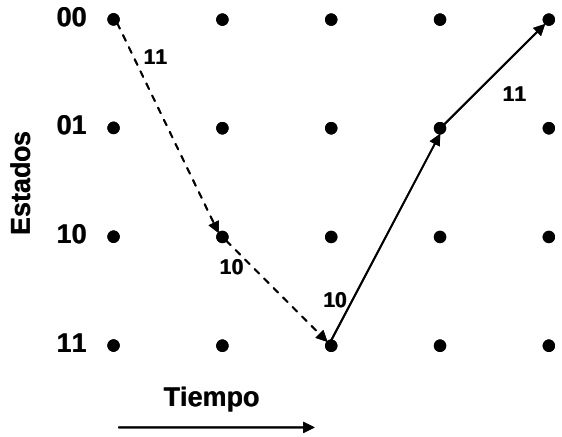
Por este motivo, en los códigos convolucionales se suele utilizar el denominado diagrama de Trellis que sí permite ver la evolución temporal.

En la siguiente figura se muestra el diagrama de Trellis de nuestro codificador convolucional [2,1,3] ejemplo:



En dicho diagrama, las flechas con trazo continuo representan un cero en la entrada del codificador mientras que las flechas con trazo discontinuo representan la entrada de un uno. Junto a las flechas se indica la salida del codificador correspondiente a cada entrada, y la dirección de la flechas indica los estados inicial y final de cada transición.

La codificación de una cadena de entrada conlleva un camino concreto en el diagrama de Trellis que determina la secuencia de salida y las transiciones entre estados. Así, la codificación de la cadena k = (**1 1 0 0**) da como resultado el camino siguiente en el diagrama Trellis, cuya secuencia de salida es la palabra código n = (**11 10 10 11**):



Aquel camino que partiendo del estado inicial **0 0 0 … 0** regresa a dicho estado con el menor número de unos en la secuencia de salida, determina lo que se denomina la **distancia libre** del código (d_{free}), la cual es igual al número de unos de dicha secuencia, esto es, el peso Hamming de la misma.

La distancia libre de un código convolucional es similar a la distancia Hamming de un código bloque y, por tanto, se puede demostrar que d_{free} representa la mínima de las distancias de dos palabras código cualesquiera.

Así, a la vista del diagrama de Trellis de nuestro ejemplo, se constata que la distancia libre queda determinada por el camino $00 \rightarrow 10 \rightarrow 01 \rightarrow 00$ el cual genera la secuencia de salida (**11 01 11**). Por tanto, en nuestro ejemplo d_{free} = **5**.

Obsérvese que, en este caso, la secuencia de salida asociada a d_{free} coincide con la respuesta impulsiva del codificador, pero nada impide que exista una secuencia con menor peso Hamming que la respuesta impulsiva.

La distancia libre aumenta con la longitud limitada L del codificador, si bien, depende de las secuencias generadoras del código. En las tablas siguientes se indican las secuencias generadoras (en formato octal) para los códigos de tasa ½ y 1/3 que tienen mayor distancia libre, en función de L:

TABLE 9.2 RATE $\frac{1}{2}$ MAXIMUM FREE DISTANCE CODES

Constraint length L	Generators in octal		d_{free}
3	5	7	5
4	15	17	6
5	23	35	7
6	53	75	8
7	133	171	10
8	247	371	10
9	561	753	12
10	1167	1545	12
11	2335	3661	14
12	4335	5723	15
13	10533	17661	16
14	21675	27123	16

Odenwalder (1970) and Larsen (1973).

TABLE 9.3 RATE $\frac{1}{3}$ MAXIMUM FREE DISTANCE CODES

Constraint length L	Generators in octal			d_{free}
3	5	7	7	8
4	13	15	17	10
5	25	33	37	12
6	47	53	75	13
7	133	145	175	15
8	225	331	367	16
9	557	663	711	18
10	1117	1365	1633	20
11	2353	2671	3175	22
12	4767	5723	6265	24
13	10533	10675	17661	24
14	21645	35661	37133	26

Odenwalder (1970) and Larsen (1973).

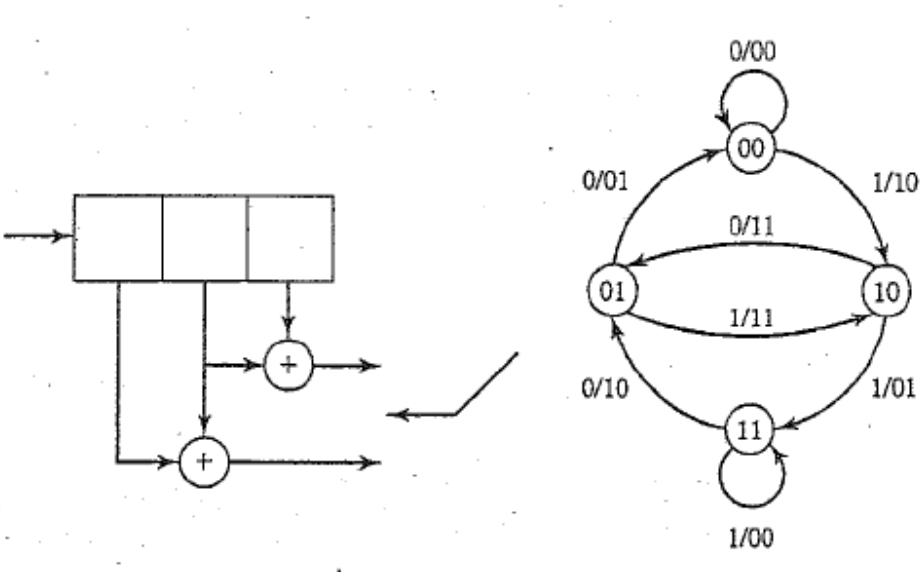
6.- Códigos catastróficos

Todo código convolucional debe mapear una cadena de bits de entrada, normalmente larga, en palabras de código con el fin de proporcionar un mayor nivel de protección frente al ruido, que puede introducir el canal de transmisión o el soporte de almacenamiento que se utilice. Obviamente, no es deseable que el codificador traduzca cadenas de información distintas en secuencias de código parecidas ya que éstas podrían confundirse en la decodificación, provocando un aumento en los bits de información erróneos.

El caso peor lo tendríamos si dos cadenas de entrada que difieren en infinitos bits son mapeadas en dos secuencias de código que difieren en unos pocos bits, ya que con un código así, existe una alta probabilidad de decodificar erróneamente las palabras código, provocando un número infinito de errores en los bits de información.

Un código que se comporta de dicha manera se dice que es un **código catastrófico** y, si bien puede tener interés matemático, en la práctica se debe evitar su uso. Veámoslo con un ejemplo.

La siguiente figura muestra el esquema y el diagrama de estados de un codificador convolucional cuyas secuencias generadoras son g_1 = (**1 1 0**), g_2 = (**0 1 1**):



Se observa que los bucles de los estados **00** y **11** generan la misma secuencia en la salida del codificador. Así, la cadena de entrada k = (**11111111 … 1100**) genera el código de salida n = (**10 01 00 00 00 00 00 … 00 00 10 01**) que tiene una distancia Hamming igual a 4 con respecto al código asociado a otra cadena de entrada con todo ceros.

Por tanto, tenemos dos entradas cuya distancia tiende a infinito mientras que los respectivos códigos de salida están a una distancia fija de tan sólo 4 bits. Es evidente que dichos códigos podrían confundirse en la decodificación si aparecen errores en unos pocos bits.

7.- Decodificación de máxima probabilidad: algoritmo de Viterbi

Tal y como se ha visto hasta ahora, la memoria de un codificador convolucional constituye una máquina de estados que va pasando de un estado a otro, a medida que van llegando los k bits a codificar.

Se puede demostrar que la secuencia de estados forma una cadena de Markov de orden $(L\text{-}1)\cdot k$ oculta, esto es, en la salida del codificador no se entrega el valor de los estados sino los n bits que dependen de los mismos.

Esto permite adivinar (o, más bien deducir) cuál es la secuencia de estados más probable si previamente se conoce el diagrama de estados, con sólo observar la secuencia de salida del codificador con el paso del tiempo. Para ello, existen varias técnicas pero la más sencilla de implementar es el **algoritmo de Viterbi**.

Un decodificador basado en el algoritmo de Viterbi va almacenando la secuencia de los n bits que va recibiendo y va calculando la secuencia de estados más probable mediante los caminos en el diagrama de Trellis que tienen menor distancia acumulada. Es decir, por cada n bits recibidos, calcula la distancia entre la secuencia recibida y la secuencia correspondiente a cada transición en el diagrama de Trellis, y descarta las transiciones que tienen mayor distancia acumulada.

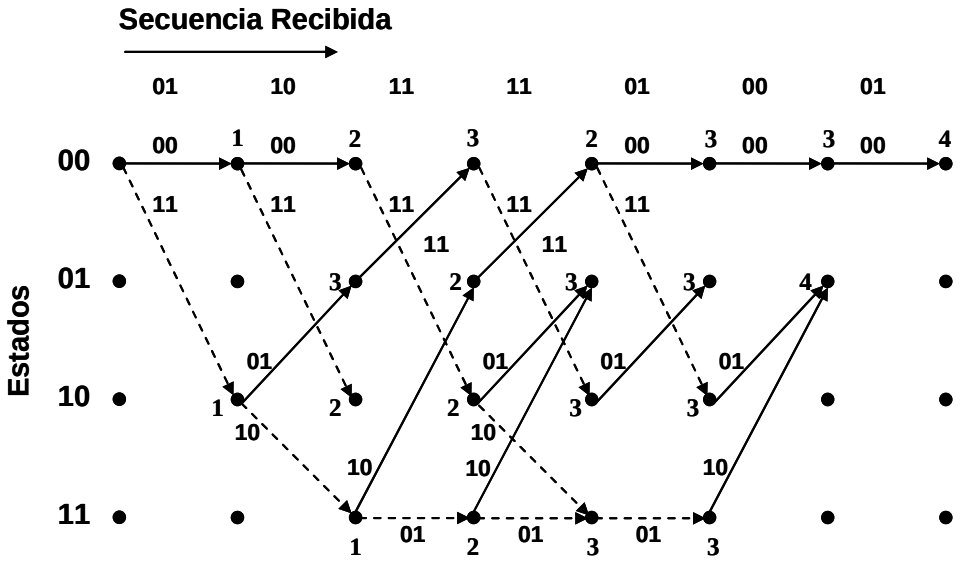
El algoritmo finaliza con el cálculo del camino que lleva al estado inicial **0 0 0 … 0** con la menor distancia acumulada. Una vez deducido el camino más probable en el diagrama de Trellis, resulta inmediato obtener la cadena de k bits de información asociada a las sucesivas transiciones.

Cabe señalar que el algoritmo de Viterbi puede manejar dos tipos de distancia, a saber: o bien la distancia euclídea de los distintos símbolos que se van recibiendo (decodificación con decisión ***blanda*** empleada en la etapa demoduladora de los canales digitales paso banda), o bien la distancia Hamming de la secuencia de bits recibida (decodificación con decisión ***dura*** empleada en los canales digitales banda base). A continuación se verá un ejemplo de decodificación dura mediante el algoritmo de Viterbi.

Sea el codificador convolucional [2,1,3] cuyo diagrama de Trellis se mostró en el punto 5 y sea r = (**01 10 11 11 01 00 01**) la secuencia código recibida. Dado que dicha secuencia consta de 14 bits, la cadena de información decodificada estará formada por 7 bits y por tanto, el algoritmo necesitará 7 transiciones de estado para finalizar la decodificación.

Además, sabemos que la cadena de información decodificada termina con $(L\text{-}1)\cdot k$ ceros de relleno lo cual significa que, en este caso, las dos últimas transiciones tienen que corresponder con dos ceros de información, para finalizar en el estado inicial **00**.

La figura siguiente muestra el diagrama de Trellis que resume el proceso de decodificación.



Según se puede observar, el algoritmo comienza en el estado inicial **00**. En la primera transición, la salida asociada a cada camino de los **dos posibles** tiene una distancia Hamming igual a 1 en ambos casos, con respecto a la secuencia recibida.

En la segunda transición, la salida asociada a cada camino de los **cuatro posibles** tiene la distancia acumulada que se indica sobre el diagrama de Trellis. Así, el camino con menor distancia acumulada viene determinado por los estados $00 \rightarrow 10 \rightarrow 11$, camino cuya salida asociada es (**11 10**).

En la tercera transición, el algoritmo comienza a descartar el camino con mayor distancia acumulada de los dos que pueden conducir a cada estado. Por ejemplo, al estado **01** se puede acceder desde el estado **10** (ver diagrama de Trellis del codificador en el punto 5) con una distancia acumulada igual a 3 o se puede acceder desde el estado **11** con una distancia acumulada igual a 2. Por ello, el primer camino queda descartado. Si dos caminos confluyen en un estado con la misma distancia acumulada, prevalecen los dos.

El algoritmo continúa hasta llegar a la penúltima transición, a partir de la cual, se calculan los caminos correspondientes a los dos ceros de información de relleno para terminar en el estado inicial **00**. De esta forma, el camino más probable en el diagrama de Trellis será aquel que nos permita retroceder desde la última transición a la primera con la menor distancia acumulada. Se constata que, en este caso, el camino más probable viene determinado por los estados $00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00 \rightarrow 00 \rightarrow 00 \rightarrow 00$, camino cuya salida asociada es la secuencia n = (**11 10 10 11 00 00 00**).

Obsérvese que la diferencia entre la secuencia recibida r = (**01 10 11 11 01 00 01**) y la secuencia más probable n = (**11 10 10 11 00 00 00**) da como resultado la secuencia de error e = (**10 00 01 00 01 00 01**) que tiene peso 4, esto es, la menor distancia acumulada.

Una vez hallado el camino más probable, resulta inmediato obtener la cadena de información decodificada, la cual está formada por los 7 bits k = (**1 1 0 0 0 0 0**) incluyendo los dos últimos ceros de relleno.

En la práctica, la secuencia recibida tendrá una longitud variable y mucho mayor que la del ejemplo. Por este motivo, el decodificador no sabrá a priori cuándo comienzan las transiciones asociadas a los bits de relleno. Esta dificultad se podría superar empleando un decodificador con capacidad de almacenar secuencias muy largas, de 32, 64 o 128 transiciones, por ejemplo. Sin embargo, al aumentar el número de transiciones almacenadas, aumenta el retardo de decodificación, lo cual repercute negativamente en cualquier aplicación práctica del código.

Mediante numerosas simulaciones con ordenador, se ha demostrado que es suficiente con que el decodificador almacene $5 \cdot L$ transiciones, de manera que la búsqueda del camino más probable se realiza truncando sucesivamente la primera transición que entró en el decodificador, sin que la probabilidad de error del algoritmo se vea afectada.

8.- Capacidad de corrección de errores

Del mismo modo que en un código bloque la capacidad de corrección de errores viene determinada por la distancia Hamming del código, en un código convolucional es la distancia libre d_{free} la que determina la capacidad de corrección de errores.

Todo código convolucional es capaz de corregir $(d_{free} -1)/2$ errores por cada $n \cdot L$ bits de código. Así, nuestro código convolucional [2,1,3] ejemplo es capaz de corregir 2 errores por cada 6 bits de código, dado que d_{free} = 5. Eso significa que si transmitimos nuestro código por un canal BSC cuya probabilidad de error es $p = 10^{-3}$, la probabilidad de error de bit (P_{eb}) después de decodificar será siempre inferior a la probabilidad de que ocurran 2 errores por cada 6 bits en dicho canal, es decir:

$$P_{eb} < \binom{6}{2} \cdot p^2 \cdot (1-p)^4 = 15 \cdot 10^{-6} \cdot (0{,}999)^4 = 1{,}494 \cdot 10^{-5}$$

Sin embargo, el propio algoritmo de Viterbi reduce dicha tasa de error ya que, según se puede demostrar, con decodificación dura y para valores pequeños de p (10^{-2} o inferior) P_{eb} es aproximadamente:

$$P_{eb} \leq \frac{1}{k} \left(2\sqrt{p}\right)^{d_{free}}$$

Por tanto, particularizando dicha expresión para nuestro código ejemplo y utilizando el canal BSC anterior, resulta:

$$P_{eb} \leq \left(2\sqrt{10^{-3}}\right)^5 = 32 \cdot \left(10^{-15/2}\right) = 1{,}012 \cdot 10^{-6}$$

Obsérvese con este ejemplo, cómo un código convolucional sencillo permite utilizar un canal BSC de mala calidad, obteniendo una tasa de error aceptable.

9.- Aplicaciones de los códigos convolucionales. Modulación codificada Trellis (TCM)

Los códigos convolucionales se utilizan generalmente como códigos de canal para corregir los errores de los canales ruidosos, ya sean debidos a limitaciones de potencia (típica en los canales digitales banda base) como a limitaciones en banda (propia de los canales digitales paso banda).

En algunas aplicaciones, los códigos convolucionales se utilizan ***concatenados*** con otro código (como Reed-Solomon, por ejemplo) de manera que éste es el código exterior y el código convolucional es el código interior al canal. Dicha combinación de códigos se prefiere, para obtener mayor capacidad de corrección de errores, frente a la utilización de un código convolucional con mayor distancia libre, en aras a la simplicidad del decodificador.

Así, encontramos códigos convolucionales en los sistemas audiovisuales que requieren corrección de errores en tiempo real como, por ejemplo, en los estándares de televisión digital por satélite (DVB-S), por cable (DVB-C) y terrestre (DVB-T), así como en algunos estándares de telefonía móvil como GSM.

Asimismo, incluyen códigos convolucionales algunos sistemas de transmisión para el bucle de abonado como, por ejemplo, los antiguos modem telefónicos V.32bis y V.34 o los modernos estándares xDSL como ADSL2+ y SHDSL. También es frecuente incluir codificación convolucional en los sistemas de comunicación para el espacio profundo (sondas espaciales, misiones no tripuladas, etc…).

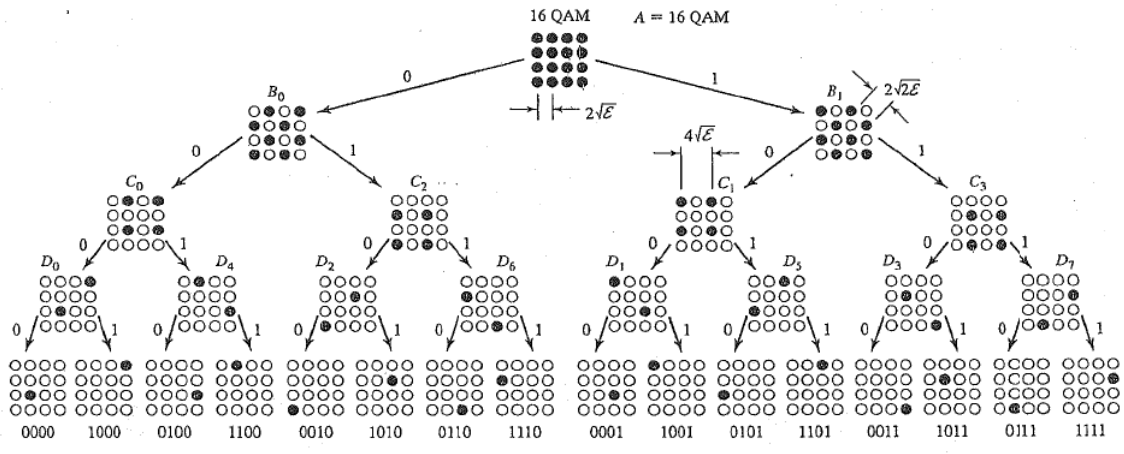
Los códigos convolucionales se pueden combinar con las técnicas de modulación digital para generar la denominada modulación codificada Trellis o **TCM** (del acrónimo inglés) cuyo funcionamiento se resume a continuación.

En el punto anterior se vio cómo un código convolucional [2,1,3] puede mejorar significativamente la probabilidad de error de un canal BSC de mala calidad, eso sí, a costa de duplicar la velocidad binaria en el canal. En los canales limitados en banda, la única manera de aumentar la velocidad binaria sin aumentar la velocidad de símbolo es usar una técnica de modulación más eficiente, esto es, que contemple un mayor número de bits por símbolo.

Sin embargo, una modulación más eficiente conlleva por un lado una constelación de símbolos más poblada, es decir, una menor distancia euclídea entre símbolos adyacentes y, por otro lado, una menor energía de bit, factores que contribuyen ambos a aumentar la tasa de error de bit.

Para contrarrestar este efecto, la modulación TCM ***particiona*** la constelación de símbolos de manera que los códigos asociados a dos símbolos consecutivos en el tiempo siguen un diagrama de Trellis definido, aumentando la distancia euclídea entre símbolos y disminuyendo la probabilidad de error. Veámoslo con un ejemplo.

Supongamos que se pretende mejorar la tasa de error de un canal digital paso banda que utiliza modulación 8-PSK (3 bits por símbolo). Para ello, podemos intercalar un codificador convolucional [4,3,3] antes de la modulación y posteriormente usar un esquema de modulación 16-QAM (4 bits por símbolo) cuya constelación se particiona según la figura siguiente:



En recepción, el demodulador utilizará el algoritmo de Viterbi con decodificación blanda para hallar el camino más probable en el diagrama de Trellis, y así, obtener los bits de información iniciales.

Se ha demostrado que la modulación TCM consigue una ganancia neta de código (definida como el cuadrado del cociente entre las distancias mínimas con y sin codificación convolucional) que varía de 3 a 6 dB según la longitud limitada L del codificador. Dicha ganancia se traduce en una reducción de la probabilidad de error de bit.

Bibliografía

- J.G. Proakis and M. Salehi, Communication Systems Engineering, 2nd ed. 2002
- Alan V. Oppenheim and Alan S. Willsky, Signals and Systems, 1983