Tema 2
Codificación de canal

1 Introducción

El objetivo principal de un sistema de comunicaciones digital es transmitir un mensaje desde una fuente a un receptor a través de un determinado canal. Un codificador de canal tiene como objetivo encontrar la forma de representar el mensaje de forma que se consiga una transmisión libre de errores aunque el canal sea ruidoso.

En principio parece claro que la única forma de conseguir este objetivo es restringir los mensajes que se envían por el canal, de forma que éstos sean tan solo un subconjunto de todos los mensajes posibles. Esta idea se muestra en la Figura 1. Como se puede observar, en el primer caso, se utiliza un alfabeto de 4 mensajes con 2 bits. Cualquier mensaje que recibamos es, en principio, válido, por lo que no nos es posible saber si se ha producido algún error.

En el segundo caso, por otra parte, se introduce un codificador de canal que lo que hace es añadir un bit al mensaje transmitido, haciendo que, aunque haya 8 mensajes posibles con 3 bits, sólo 4 son válidos. Así, si recibimos un mensaje no válido sabremos con seguridad que se ha producido algún error en la transmisión, y podremos tomar las medidas oportunas.

Debemos distinguir aquí dos tipos de sistemas distintos: **sistemas detectores de errores** y **sistemas correctores de errores**. Los primeros son, como se puede suponer, más sencillos de implementar que los segundos, y pueden aprovecharse de la posible existencia de un canal de retorno para que el receptor solicite la retransmisión de aquella información en la que se han detectado errores. Este tipo de sistemas son típicos en sistemas como por ejemplo redes de ordenadores.

En otros casos la retransmisión de la información no es posible o no resulta práctica. Trataremos entonces de técnicas de corrección de errores.

Empezaremos viendo algunos ejemplos de códigos muy simples que nos permitirán empezar a comprender las ideas básicas del problema para centrarnos después en los dos grandes tipos de codificadores de canal: los códigos bloque y los códigos convolucionales.

2 Códigos de repetición

Quizás la forma más sencilla de construir un código de protección frente a errores es repetir directamente cada bit n veces, de forma que cada palabra código constará de n bits idénticos. Esto no deja de ser lo mismo que hacemos nosotros cuando intentamos mantener una conversación en un ambiente muy ruidoso: repetir las cosas más de una vez.

Imaginemos por ejemplo que construimos un código de triple repetición con las palabras código 000 y 111. Cualquier otra palabra recibida (p.ej. 101 o 010) nos indica que se ha producido algún error. Este código tan simple nos permite tanto detectar como corregir errores, en función de cómo sea la implementación del decodificador:

- Si sólo queremos detectar errores, basta con marcar cualquier palabra recibida distinta de las dos permitidas 000 y 111 y solicitar que se reenvíe.
- Para corregir errores se puede utilizar una regla de mayoría, suponiendo que habrá menos bits erróneos que correctos en la palabra. Así, 101 se decodificará como 111 y 010 como 000. Es interesante darse cuenta de que este sistema es capaz de corregir correctamente palabras con un error, pero si hay dos o tres errores se va a equivocar.

En este caso vemos que hemos construido un codificador que, para cada bit de entrada ($k = 1$) produce n bits a la salida (en nuestro ejemplo $n = 3$). Esto nos da una medida de la eficiencia del codificador, y así se define la **tasa de codificación** como:

$$R_c \triangleq \frac{k}{n}$$

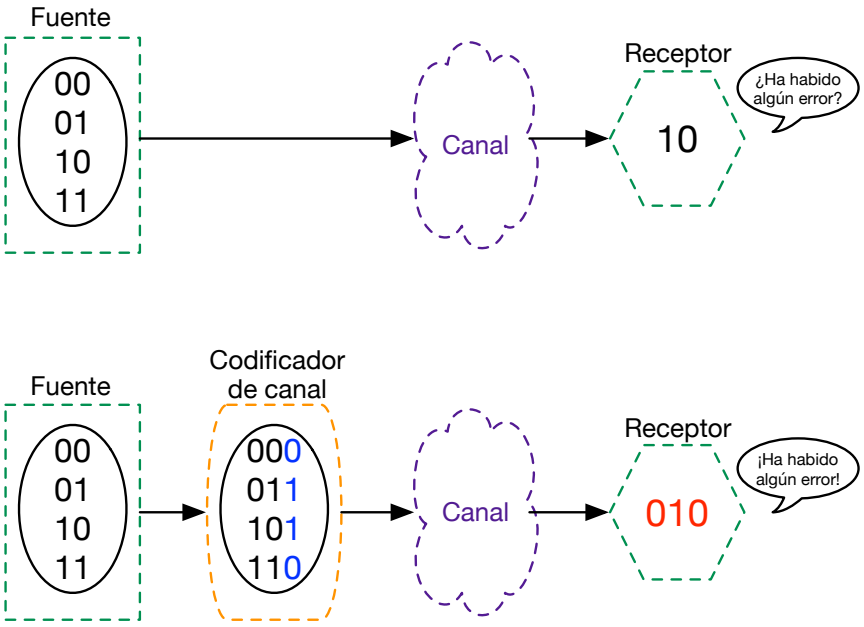


Figura 1: Esquema de un sistema de comunicaciones digitales (a) sin codificación de canal y (b) con un codificador de canal que añade un bit extra.

Otro parámetro que podemos definir a partir de lo visto para este codificador es la **distancia mínima** de un código, que se define como la distancia Hamming más pequeña existente entre dos palabras código válidas.

La **distancia Hamming** entre dos vectores, $d(\mathbf{x}, \mathbf{y})$, se calcula como el número de elementos distintos entre los dos vectores. Así, por ejemplo, si $\mathbf{x} = (101)$ e $\mathbf{y} = (001)$, entonces $d(\mathbf{x}, \mathbf{y}) = 1$ ya que sólo el primer elemento es diferente entre ambos.

Para nuestro ejemplo de codificador de repetición podemos ver que se cumple que $d_{min} = 3$. La utilidad de la distancia mínima es que nos permite deducir cuál va a ser la capacidad de un determinado código en cuanto a la detección y/o corrección de errores. En concreto, si un código posee una distancia minima d_{min}, entonces

- Podremos detectar no más de $d_{min} - 1$ errores por palabra.
- Podremos corregir no más de $\frac{d_{min}-1}{2}$ errores por palabra.

En nuestro caso, con $d_{min} = 3$ podemos ver que es posible detectar hasta 2 errores por palabra y corregir 1 error, lo cual coincide con lo que ya habíamos deducido anteriormente.

Supongamos ahora que el canal de transmisión utilizado hace que la probabilidad de que ocurra un error es $p_e \ll 1$. Es posible demostrar que la probabilidad de que ocurran i errores en una palabra código de n bits viene dada por:

$$P(i,n) = \left(\begin{array}{c} n \\ i \end{array} \right) \cdot p_e^i \cdot (1-p_e)^{n-i} \approx \left(\begin{array}{c} n \\ i \end{array} \right) \cdot p_e^i$$

siendo:

$$\left(\begin{array}{c} n \\ i \end{array} \right) = \frac{n!}{i!\,(n-i)!}$$

Utilizando este resultado podemos ver que la probabilidad de que ocurra un error y éste no sea detectado (porque ha habido 3 errores en la palabra código) es:

$$P(3,3) = p_e^3$$

Y, para el caso de corrección de errores, seremos capaces de corregir sólo errores en un bit, pero no en dos o tres. Así, la probabilidad de cometer un error en la decodificación será:

$$P = P(2,3) + P(3,3) = 3p_e^2 - 2p_e^3$$

3 Códigos de paridad

Otra técnica de codificación de canal muy utilizada por su sencillez es la codificación de paridad. Se dice que la paridad de una determinada palabra binaria es *par* si el número de unos que contiene es par (p.ej. 110101). De forma análoga, una palabra tendrá paridad impar si contiene un número impar de unos (p.ej. 101010).

Con esta idea en mente, un código de paridad construye palabras código de n bits formadas por $n-1$ bits de mensaje a los que se añade un bit de control elegido de forma que todas las palabras código tengan la misma paridad (par o impar, según se decida). La tasa de codificación vendrá dada por tanto por $R_c = \frac{n-1}{n}$.

Así, por ejemplo, si decidimos utilizar paridad par y $n = 3$, la secuencia de bits 01 se complementará con un tercer bit 1 para hacer que su paridad sea par, mientras que si los bits de entrada son 11 el bit de control será 0. En este caso las únicas palabra código permitidas con este esquema serán: 000, 011, 110 y 101, es decir aquellas palabras de 3 bits con un número par de unos. Cualquier otra palabra (p.ej 010) no cumple con la condición de paridad par y nos estaría indicando que se ha producido algún error (nótese que este es el ejemplo ilustrado en la Figura 1(b)).

Si lo pensamos, veremos que en este caso sólo es posible la detección de errores, y no la corrección, ya que no sabemos qué bit de la palabra código es el que contiene el error. Podemos llegar a la misma conclusión si nos fijamos en que para este código la distancia mínima es $d_{min} = 2$.

Además, si el número de errores es par, la palabra código resultante sigue siendo válida y por tanto no seremos capaces tampoco de detectar el error. Podemos por tanto calcular la probabilidad de que un error no sea detectado haciendo un desarrollo similar al del apartado anterior. Podemos asumir que la probabilidad de que se produzcan dos errores en una palabra es mucho mayor que la de que se produzcan cuatro, seis o más errores, por lo que podemos aproximar la probabilidad por:

$$P \approx P(2,n) \approx \frac{n(n-1)}{2} p_e^2$$

4 Códigos bloque

Un código bloque (n,k) está formado por vectores de n bits de los cuales $k < n$ bits se corresponden con la información, y el resto $(n-k)$ son bits de control utilizados para la detección o corrección de errores. La tasa de codificación de un código bloque viene dada por tanto por $R_c = \frac{k}{n}$.

Podemos representar uno de estos vectores de n bits como $\mathbf{x} = (x_1, x_2, \cdots x_n)$, donde cada uno de los x_i representa un bit.

Decimos que un código es **lineal** si se cumple que la suma de dos vectores código produce otro vector código[1].

[1] Definimos la suma de dos vectores como:

$$\mathbf{x} + \mathbf{y} \triangleq (x_1 \oplus y_1, x_2 \oplus y_2, \cdots x_n \oplus y_n)$$

donde $\oplus$ representa al operador de suma módulo 2, definido como:

$$x_i \oplus y_i = \begin{cases} 0 & \text{si } x_i = y_i \\ 1 & \text{si } x_i \neq y_i \end{cases}$$

Asimismo, la multiplicación módulo 2 se define como:

$$x_i \cdot y_i = \begin{cases} 1 & \text{si } x_i \text{ o } y_i = 1 \\ 0 & \text{en otro caso} \end{cases}$$

Estas operaciones son las conocidas operaciones lógicas O-Exclusivo (XOR) e Y (AND) del álgebra de Boole. Es importante tener en cuenta además que la suma y la resta en este caso son la misma operación, ya que 1 es su propio elemento opuesto.

Como consecuencia de lo anterior, para un código lineal también se cumple *el vector* $\mathbf{0}$ *debe formar parte del código*.

La propiedad de la linealidad nos permite estudiar los códigos bloque aplicando los conceptos y técnicas conocidos del álgebra lineal.

Podemos definir también el **peso** de un vector ($w(\mathbf{x})$) como el número de elementos distintos de cero. A partir de este valor se puede calcular la distancia entre dos vectores $\mathbf{x}$ e $\mathbf{y}$ de forma muy sencilla como:

$$d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} + \mathbf{y})$$

De aquí también se puede deducir que *la distancia mínima de un código se puede obtener buscando el vector código, no nulo, con menor peso*.

Supongamos ahora que se tiene un mensaje consistente en un vector fila de k elementos:

$$\mathbf{m} = (m_1, m_2, \cdots m_k)$$

Podemos obtener el vector código correspondiente a este mensaje haciendo uso de la llamada **matriz generadora** del código, $\mathbf{G}$, de la siguiente forma:

$$\mathbf{x} = \mathbf{mG}$$

siendo $\mathbf{G}$ una matriz de dimensiones $k \times n$ cuyas filas son k vectores linealmente independientes.

Sabemos, del álgebra lineal, que es posible realizar una serie de operaciones (método de Gauss) sobre la matriz $\mathbf{G}$

- Intercambiar filas o columnas
- Sumar una fila a otra
- Multiplicar una fila por un escalar no nulo

de forma que obtengamos una nueva matriz $\mathbf{G}'$ con la siguiente estructura:

$$\mathbf{G} = [\mathbf{I_k}|\mathbf{P}]$$

donde $\mathbf{I_k}$ representa a la matriz identidad $k \times k$.

Una matriz con esta estructura se denomina **sistemática**, y un código caracterizado por una matriz generadora sistemática se denomina **código sistemático**.

Es posible demostrar que $\mathbf{G}$ y $\mathbf{G}'$ no definen en general la misma función de codificación, aunque sí el mismo conjunto de vectores, salvo algún intercambio en las posiciones de los símbolos. No obstante, las propiedades del código relativas a la detección y corrección de errores dependen solamente de cuáles sean los símbolos de los vectores del código y no del orden particular de sus elementos. Por ello se dice que las matrices $\mathbf{G}$ y $\mathbf{G}'$ son equivalentes, y también son equivalentes los códigos que representan.

Es inmediato observar que en un código sistemático los k bits de información son un subconjunto de los n bits de la palabra código, pudiendo representar un vector del mismo como:

$$\mathbf{x} = (m_1, m_2, \cdots m_k, c_1, \cdots c_q)$$

donde $q = n - k$. Podemos escribir la expresión anterior de forma más compacta como:

$$\mathbf{x} = (\mathbf{m} \,|\, \mathbf{c})$$

siendo $\mathbf{m}$ un vector de mensaje con k bits y $\mathbf{c}$ un vector de control con $q = n - k$ bits.

Por tanto, si tenemos un vector $\mathbf{m}$, el correspondiente vector código se puede obtener matricialmente como:

$$\mathbf{x} = \mathbf{m}\,\mathbf{G}$$

4.1 Decodificación por síndrome

Supongamos ahora que tenemos un vector de datos recibidos $\mathbf{y}$. El decodificador deberá detectar y corregir los posibles errores de transmisión en $\mathbf{y}$ utilizando la información de la que dispone en el código.

La forma más sencilla de conseguir esto es utilizar la denominada matriz de comprobación de paridad, $\mathbf{H}$, que es una matriz $q \times n$ definida como:

$$\mathbf{H} \triangleq \left[\mathbf{P}^T \,|\, \mathbf{I_q}\right]$$

donde $\mathbf{P}^T$ es la matriz traspuesta de $\mathbf{P}$ e $\mathbf{I_q}$ una matriz identidad $q \times q$. Esta matriz de comprobación de la paridad tiene la propiedad de que, si $\mathbf{x}$ pertenece al conjunto de palabras código,

$$\mathbf{x}\,\mathbf{H}^T = (0, 0, \cdots 0)$$

En cualquier otro caso, el producto $\mathbf{y}\,\mathbf{H}^T$ contendrá al menos un elemento distinto de cero. Al resultado de esta operación, $\mathbf{s} = \mathbf{y}\,\mathbf{H}^T$ se le denomina **síndrome**.

Si todos los elementos del síndrome son cero, entonces o bien no se ha producido ningún error o sí que lo ha habido pero es indetectable, ya que se ha generado una palabra código válida pero distinta.

Hasta aquí ya tenemos las herramientas para poder detectar errores. El siguiente paso es intentar corregirlos. Si consideramos que se ha transmitido una palabra código $\mathbf{x}$ y que se recibe un vector $\mathbf{y}$, podemos el error como la diferencia entre estos dos vectores:

$$\mathbf{y} = \mathbf{x} + \mathbf{e}$$

Aplicando lo visto antes, podemos calcular el síndrome para el vector $\mathbf{y}$, y sustituyendo:

$$\mathbf{s} = \mathbf{y}\mathbf{H}^T = (\mathbf{x} + \mathbf{e})\mathbf{H}^T = \mathbf{e}\mathbf{H}^T$$

Es decir, vemos que *el síndrome depende exclusivamente del error cometido*, y no del vector transmitido. Por tanto en teoría es posible deducir cuál ha sido el error a partir del síndrome, y a partir de ahí calcular el vector transmitido.

El problema es que dado que el síndrome sólo tiene q bits, no hay más que 2^q síndromes distintos, mientras que puede haber hasta 2^n posibles vectores de error. Por tanto la correspondencia entre síndrome y error no es unívoca, y tendremos que realizar alguna suposición adicional para conseguir nuestro objetivo.

La estrategia más habitual es la conocida como **decodificación de máxima verosimilitud (ML)**. Consiste en que vamos a considerar sólo aquellos patrones de error más probable, olvidándonos del resto. Para ello, debemos crear una tabla con los síndromes generados por los $2^q - 1$ vectores de error más probables (aquellos con un peso menor o igual a $\lfloor \frac{d_{min}-1}{2} \rfloor$). Después, el decodificador calcula el síndrome a partir del vector recibido $\mathbf{y}$, y busca el vector de error correspondiente a dicho síndrome, $\hat{\mathbf{e}}$, en la tabla. Con eso, lo único que queda es calcular la suma $\mathbf{y} + \hat{\mathbf{e}}$, que será la palabra decodificada.

Todo lo visto hasta ahora nos proporciona las herramientas necesarias para poder analizar un código bloque, pero no nos dice nada sobre cómo definir la matriz generadora o la de comprobación de paridad. Vamos a ver a continuación un ejemplo de código bloque: los códigos Hamming.

4.2 Códigos Hamming

Los códigos Hamming fueron propuestos entre 1947 y 1948 por Richard Hamming y son códigos lineales capaces de corregir errores que afecten a un sólo bit. Constan de $q \geq 3$ bits de control y:

$$\begin{aligned} n &= 2^q - 1 \\ k &= n - q = 2^q - q - 1 \end{aligned}$$

La relación de codificación viene dada por tanto por:

$$R_c = \frac{k}{n} = 1 - \frac{1}{2^q - 1}$$

Independientemente del valor de q la distancia mínima de un código Hamming es 3, por lo que este tipo de códigos se pueden utilizar para corregir hasta un error y/o para detectar hasta dos errores.

Construir la matriz de comprobación de paridad $\mathbf{H}$ es muy sencillo. Basta con colocar en sus columnas todos los números desde 1 hasta $2^q - 1$. El orden no es importante pero resulta conveniente colocar en último lugar las columnas que generan la matriz identidad, para conservar la notación que hemos venido utilizando.

Así, por ejemplo, si consideramos un código Hamming con $q = 3$, una posible matriz de comprobación de paridad será:

$$\mathbf{H} = \left[\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

A partir de aquí podemos obtener la matriz generadora $\mathbf{G}$ de forma inmediata recordando que $\mathbf{H} = [\mathbf{P}^T|\mathbf{I_q}]$:

$$\mathbf{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right]$$

4.3 Códigos perfectos

Los códigos de Hamming son un ejemplo de un tipo de códigos más generales que se denominan **códigos perfectos**. Para describirlos es necesario que antes definamos lo que se conoce como cota de Hamming.

La **cota de Hamming** establece que un código con capacidad para corregir t errores debe tener una redundancia $q = n - k$ tal que:

$$q \geq log_2 V(n,t)$$

donde a $V(n,t)$ se le denomina esfera de Hamming de radio t, e indica el número de vectores que hay a una distancia menor o igual a t de cada palabra. Viene dado por la siguiente expresión:

$$V(n,t) = \sum_{j=0}^{t} \left(\begin{array}{c} n \\ j \end{array} \right)$$

En el caso particular de que un código cumpla que $q = log_2 V(n,t)$ decimos que el código es perfecto.

Existen en concreto cuatro tipos de códigos perfectos:

Nombre	n	k	t
Trivial	$\mathbb{N}$	$\mathbb{N}$	0
Repetición	Impar	1	$(n-1)/2$
Hamming	$2^q - 1$	$2^q - q - 1$	1
Golay	23	11	3

4.4 Decodificación blanda y decodificación dura

Hasta ahora hemos estando dando por hecho que la decodificación se hacía bit a bit, sin hacer uso de ningún tipo de información adicional. A este tipo de decodificación se le denomina **decodificación dura**. En contraste con ella, existen otros esquemas que utilizan algún tipo de medida de la verosimilutd de cada bit transmitido dentro del receptor de comunicaciones. Hablaremos en este caso de una **decodificación blanda**. Para entender mejor cómo funcionan estos dos esquemas y sus diferencias vamos a utilizar un ejemplo muy sencillo.

Imaginemos que utilizamos un código de paridad con $n = 3$, y que cada bit transmitido se modula utilizando una modulación PAM polar con amplitud $\sqrt{E_s}$, de modo que un 1 se transmite como $+\sqrt{E_s}\ V$ y un 0 como $-\sqrt{E_s}\ V$.

4.4.1 Decodificación dura

Para este caso, tal y como hemos venido haciendo hasta ahora, sabemos que la distancia mínima del código planteado es 2 y por tanto se trata de un código que no tiene capacidad para corregir errores. La probabilidad de que el decisor se equivoque, por tanto, será igual a la probabilidad de que haya más de un error en la palabra código recibida. Esto es:

$$P_e = P(1,3) + P(2,3) + P(3,3) = 3P_b(1-P_b)^2 + 3P_b^2(1-P_b) + P_b^3 \approx 3 \cdot P_b$$

donde estamos suponiendo que la probabilidad de error de bit (P_b) es muy pequeña, y por tanto que la relación señal a ruido del canal de comunicaciones es suficientemente elevada.

A partir de lo sabido por teoría de la detección podemos escribir la probabilidad de error de bit en este caso como:

$$P_b = Q\left(\frac{d_{min}}{\sqrt{2N_0}}\right)$$

donde d_{min} representa la distancia mínima entre dos símbolos (no confundir con la distancia Hamming mínima entre dos palabras códigos que hemos estado utilizando hasta ahora). En nuestro caso, según lo que hemos dicho: $d_{min} = 2\sqrt{E_s}$.

Por tanto, finalmente, la probabilidad de que el decisor cometa un error utilizando decodificación dura será:

$$P_e \approx 3Q\left(\sqrt{\frac{2E_s}{N_0}}\right)$$

Esto que hemos obtenido es la probabilidad de que el decisor tome una decisión errónea a partir de los datos recibidos. En general resulta interesante conocer también la tasa de error de bit (BER, *Bit Error Rate*), que nos indica el número de bits recibidos erróneamente con respecto al total de bits transmitidos.

Imaginemos que se ha producido un error en la transmisión y que, habiendo transmitido la palabra 000, recibimos la palabra binaria 001. El decisor, a la hora de decidir qué palabra código se ha enviado tiene tres opciones a una distancia Hamming de 1: 000, 011 y 101 (la cuarta opción sería la palabra código 110, pero la descartaríamos ya que está a mayor distancia Hamming).

Ante esta disyuntiva, el decisor va a actuar al azar, y por tanto el valor de la BER vendrá dado por:

$$BER \approx \frac{2}{3}3Q\left(\sqrt{\frac{2E_s}{N_0}}\right)$$

Si además consideramos que estamos empleando tres símbolos para enviar 2 bits, tenemos que $E_s = 2E_b/3$, y por tanto:

$$BER \approx 2Q\left(\sqrt{\frac{4E_b}{3N_0}}\right)$$

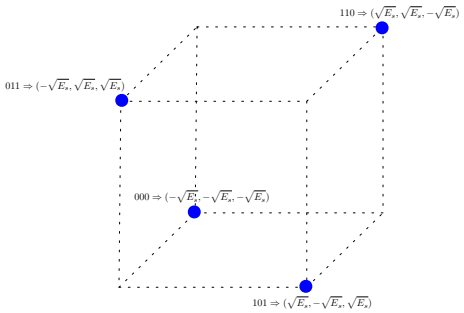
4.4.2 Decodificación blanda

En este caso el decisor observa a su entrada un vector de dimensión 3, que será la suma de uno de los cuatro símbolos de la constelación que se muestra aquí más el ruido introducido por el canal de transmisión:

La distancia mínima de esta constelación es[2] $d_{min} = 2\sqrt{2}\sqrt{E_s}$ y, utilizando la expresión de la cota de la unión simplificada para obtener la probabilidad de error:

$$P_e \leq 3Q\left(2\sqrt{\frac{E_s}{N_0}}\right)$$

[2]Aunque en este caso es muy sencillo verlo gráficamente, se puede comprobar que en general se cumple que $d_{min} = \sqrt{d_H} \cdot d(a_0, a_1)$, donde $d(a_0, a_1)$ es la distancia entre los símbolos de la constelación binaria ($2\sqrt{E_s}$ en nuestro caso) y d_H la distancia Hamming mínima del código.



Igual que hicimos antes podemos calcular el valor de la BER:

$$BER \approx 2Q\left(2\sqrt{\frac{E_s}{N_0}}\right) = 2Q\left(\sqrt{\frac{8E_b}{3N_0}}\right)$$

Se puede observar que el valor obtenido será siempre menor al obtenido para el caso de decodificación dura.
A partir de lo visto podemos extraer las siguientes conclusiones:

- El uso de un **decodificador duro** nos va a permitir corregir o detectar errores.
- El uso de un **decodificador blando** nos va a permitir "evitar" los errores, aumentando la distancia mínima del código con respecto a la de la constelación empleada.

5 Códigos cíclicos

El principal problema de los códigos convolucionales tal y como los hemos estudiado es que los requisitos hardware para la codificación y decodificación se vuelven prohibitivos en cuanto aumentamos los valores de n y k. Los códigos cíclicos son una subclase de códigos bloque que permiten una implementación práctica más sencilla.

Supongamos que tenemos una palabra código de n bits con la siguiente estructura:

$$\mathbf{c} = (c_0, c_1, \cdots, c_{n-2}, c_{n-1})$$

Imaginemos ahora que desplazamos circularmente todos los bits de la palabra anterior una posición hacia la derecha. El nuevo vector que obtendríamos sería:

$$\mathbf{c}' = (c_{n-1}, c_0, c_1 \cdots, c_{n-3}, c_{n-2})$$

Pues bien, decimos que un código lineal es cíclico si cualquier desplazamiento de una palabra código es otra palabra código. Cada una de estas palabras código se puede representar mediante un polinomio binario, que permite realizar la codificación y el cálculo del síndrome de una forma muy eficiente. En nuestro caso, podemos escribir $\mathbf{c}$ como:

$$\mathbf{c} = c_0 + c_1 x + \cdots c_{n-2} x^{n-2} + c_{n-1} x^{n-1}$$

Donde x es una variable real arbitraria cuyo exponente indica las posiciones de cada uno de los bits en la palabra binaria.

Si multiplicamos el polinomio $\mathbf{c}$ por x,

$$\mathbf{c}x = c_0 x + c_1 x^2 + \cdots c_{n-2} x^{n-1} + c_{n-1} x^n$$

esto se puede reescribir de la siguiente forma, sumando y restando c_{n-1}:

$$\mathbf{c}x = c_{n-1}(x^n - 1) + c_{n-1} + c_0 x + c_1 x^2 + \cdots c_{n-2} x^{n-1}$$

Es decir, tenemos que:

$$\mathbf{c}x = c_{n-1}(x^n - 1) + \mathbf{c}'$$

Y aprovechando las propiedades de la suma en módulo 2:

$$\mathbf{c}' = \mathbf{c}x + c_{n-1}(x^n - 1)$$

Finalmente, de esta expresión deducimos que el vector $\mathbf{c}'$ se obtiene como el resto de la división de $\mathbf{c}x$ entre $(x^n - 1)$ (operación módulo):

$$\mathbf{c}' = (\mathbf{c}x)_{(x^n-1)}$$

El polinomio $x^n - 1$ juega un papel muy importante en los códigos cíclicos. De forma más general, se puede comprobar que el desplazamiento cíclico de i posiciones de $\mathbf{c}$ es:

$$\mathbf{c}^{(i)} = (x^i\mathbf{c})_{(x^n-1)}$$

Aunque en principio pueda parecer que este planteamiento es excesivamente farragoso, la realidad es que de esta forma se pueden sustituir las operaciones matriciales por operaciones polinómicas, mucho más simples de realizar.

Los códigos cíclicos se caracterizan por tener un **polinomio generador** $g(x)$ de grado $r = n - k$ a partir del cual se pueden generar todas las palabras código multiplicándolo por todos los polinomios $b(x)$ de grado máximo $k - 1$ que forman las palabras de entrada al codificador.

Vamos a intentar comprender mejor este procedimiento con un ejemplo. Supongamos que tenemos un código cíclico $n = 7$ definido por el siguiente polinomio generador: $g(x) = x^3 + x + 1$. Vamos a obtener todas las posibles palabras código.

En primer lugar nos fijamos en el orden del polinomio, $r = 3$. De aquí se deduce que $k = n - r = 4$. Para codificar una palabra de entrada cualquiera, por ejemplo 0101, lo que haremos es, en primer lugar, convertirla a notación polinómica: $b(x) = x^2 + 1$, y a continuación multiplicar este polinomio por el polinomio generador:

$$b(x) \cdot g(x) = (x^2 + 1) \cdot (x^3 + x + 1) = x^5 + x^2 + x + 1$$

Esta expresión, convertida a formato de palabra binaria, sería: 1110010.

Si repetimos este mismo proceso para todas las demás posibles palabras de entrada de 4 bits, obtendremos el resultado que se muestra en la siguiente tabla:

Mensaje	Código	Polinomio del código
0000	0000000	$0 \cdot g(x) = 0$
0001	1101000	$1 \cdot g(x) = x^3 + x + 1$
0010	0110100	$x \cdot g(x) = x^4 + x^2 + x$
0011	1011100	$(x + 1) \cdot g(x) = x^4 + x^3 + x^2 + 1$
0100	0011010	$x^2 \cdot g(x) = x^5 + x^3 + x^2$
0101	1110010	$(x^2 + 1) \cdot g(x) = x^5 + x^2 + x + 1$
0110	0101110	$(x^2 + x) \cdot g(x) = x^5 + x^4 + x^3 + x$
0111	1000110	$(x^2 + x + 1) \cdot g(x) = x^5 + x^4 + 1$
1000	0001101	$(x^3) \cdot g(x) = x^6 + x^4 + x^3$
1001	1100101	$(x^3 + 1) \cdot g(x) = x^6 + x^4 + x + 1$
1010	0111001	$(x^3 + x) \cdot g(x) = x^6 + x^3 + x^2 + x$
1011	1010001	$(x^3 + x + 1) \cdot g(x) = x^6 + x^2 + 1$
1100	0010111	$(x^3 + x^2) \cdot g(x) = x^6 + x^5 + x^4 + x^2$
1101	1111111	$(x^3 + x^2 + 1) \cdot g(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$
1110	0100011	$(x^3 + x^2 + x) \cdot g(x) = x^6 + x^5 + x$
1111	1001011	$(x^3 + x^2 + x + 1) \cdot g(x) = x^6 + x^5 + x^3 + 1$

Como se podrá observar, el código que hemos construido no es sistemático, esto es, los bits de la entrada no forman parte de la salida. Existe un procedimiento para obtener una versión sistemática de un código cíclico que, aunque algo más complicado que el anterior, resulta muy sencillo de entender con un ejemplo.

El procedimiento se basa en crear la palabra código resultante, $c(x)$, a partir del polinomio generador $g(x)$ de la forma:

$$c(x) = b(x)x^{n-k} + d(x)$$

donde $d(x)$ se obtiene mediante como el resto de dividir $b(x)x^{n-k}$ por $g(x)$.

En nuestro ejemplo anterior, donde $g(x) = x^3 + x + 1$, $n = 7$ y se deseaba codificar la palabra 1001, haremos lo siguiente:

1. Dividimos $b(x)x^3$ entre $g(x)$, obteniéndose que el resto es $d(x) = x^2 + x$.
2. Construimos la palabra código como $c(x) = b(x)x^{n-k} + d(x)$. Así:

$$c(x) = x^6 + x^3 + x^2 + x \Rightarrow \mathbf{c} = [0111001]$$

En cuanto a la decodificación, debemos obtener el polinomio de comprobación de paridad, $h(x)$, que cumple que:

$$(g(x)h(x))_{(x^n-1)} = 0$$

El síndrome será ahora un polinomio de grado $n-k-1$ que se obtiene a partir del polinomio recibido $r(x)$:

$$s(x) = (r(x)h(x))_{(x^n-1)}$$

Hemos visto cómo realizar la codificación y decodificación de los códigos binarios cíclicos, pero no hemos visto cómo diseñarlos para cumplir algún determinado requisito de detección y corrección de errores. Para ello es necesario recurrir a la estructura algebraica de los cuerpos de Galois basados en polinomios, y en este sentido se han propuesto distintos algoritmos entre los que los más conocidos son el BCH (propuesto por Bose, Chaudhuri y Hocquenghem) o los Reed-Solomon.

Vamos a ver, de forma muy simplificada, la forma de diseñar el polinomio generador de un código cíclico BCH de longitud n y con capacidad para corregir t errores.

En concreto vamos a verlo para el caso particular de $n = 15$ y $t = 2$.

El proceso consta de los siguientes pasos:

1. **Calcular el valor de m**

   En este paso se trata de encontrar un m tal que $((2^m - 1))_n = 0$. En nuestro caso es sencillo ver que si $m = 4$, $2^m - 1 = 15$ y $((15))_{15} = 0$.

2. **Encontrar un polinomio primitivo de grado m**

   Un polinomio primitivo es aquel polinomio que no es posible factorizar en un producto de polinomios de menor grado y que cumple que el menor entero n para el cual divide a $x^n - 1$ es $n = 2^m - 1$.

   En este cuadro se muestran los polinomios primitivos de grado 2 a 0:

$p_2(x) = x^2 + x + 1$
$p_3(x) = x^3 + x + 1$
$p_4(x) = x^4 + x + 1$
$p_5(x) = x^5 + x^2 + 1$
$p_6(x) = x^6 + x + 1$
$p_7(x) = x^7 + x + 1$
$p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$
$p_9(x) = x^9 + x^4 + 1$
$p_{10}(x) = x^{10} + x^3 + 1$

Siguiendo con nuestro ejemplo, y dado que habíamos visto que $m = 4$, tomaremos el polinomio $p_4(x) = x^4 + x + 1$.

3. **Generar los elementos del cuerpo de Galois**

   Como decíamos antes, la estructura algebraica de los códigos sistemáticos se sustenta sobre la base de los cuerpos de Galois. En este caso tenemos que generar los elementos de un cuerpo de Galois $GF(2^m)$. La forma más sencilla de hacer esto es calcular todas las posibles raíces de $x^n - 1$, de forma que:

   $$x^n - 1 = (x - \alpha^0) \cdot (x - \alpha^1) \cdot (x - \alpha^2) \cdots (x - \alpha^{2^m - 2})$$

   La forma de hacerlo es partiendo de una raíz $\alpha^0 = 1$ y generando las siguientes, hasta α^{2^m-2} de la siguiente forma:

   $$\alpha^i = ((\alpha^{i-1}))_{p_m(x)}$$

   En nuestro caso:

$\alpha^0 = 1$
$\alpha^1 = x$
$\alpha^2 = x^2$
$\alpha^3 = x^3$
$\alpha^4 = ((x^4))_{x^4+x+1} = x + 1$
$\alpha^5 = x^2 + x$
$\alpha^6 = x^3 + x^2$
$\alpha^7 = ((x^4 + x^3))_{x^4+x+1} = x^3 + x + 1$
$\alpha^8 = x^2 + 1$
$\alpha^9 = x^3 + x$
$\alpha^{10} = x^2 + x + 1$
$\alpha^{11} = x^3 + x^2 + x$
$\alpha^{12} = x^3 + x^2 + x + 1$
$\alpha^{13} = x^3 + x^2 + 1$
$\alpha^{14} = x^3 + 1$

   A partir de aquí es fácil comprobar que $\alpha^{15} = \alpha^0$, y en general que $\alpha^i = \alpha^{((i))_{2^m-1}}$.

4. **Construir las clases conjugadas y los polinomios minimales**

   Para crear la primera clase conjugada tomaremos la raíz α y contruimos la secuencia de cuadrados consecutivos $\alpha, \alpha^2, \alpha^4, \alpha^8, etc.$ y crearemos un polinomio que contenga todas estas raíces: $(x - \alpha)(x - \alpha^2)(x - \alpha^4) \cdots$

   Haremos lo mismo para el resto de las raíces, y obtendremos:

Clases conjugadas	Polinomios minimales
$\{\alpha^0\}$	$m_0(x) = x - 1$
$\{\alpha, \alpha^2, \alpha^4, \alpha^8\}$	$m_1(x) = x_4 + x + 1$
$\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^{((24))_{15}} = \alpha^9\}$	$m_3(x) = x^4 + x^3 + x^2 + x + 1$
$\{\alpha^5, \alpha^{10}\}$	$m_5(x) = x^2 + x + 1$
$\{\alpha^7, \alpha^{14}, \alpha^{((28))_{15}} = \alpha^{13}, \alpha^{((56))_{15}} = \alpha^{11}\}$	$m_7(x) = x^4 + x^3 + 1$

5. **Construir el polinomio generador**

   El último paso consiste en construir el polinomio generador de menor grado posible que contenga $2t$ raíces consecutivas de $x^n - 1$ partir de sus polinomios minimales.

En nuestro ejemplo podemos ver que ninguna de las clases conjugadas tiene 4 raíces consecutivas, pero si combinamos, por ejemplo, la segunda con la tercera, obtenemos el conjunto $\{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^6, \alpha^8, \alpha^9, \alpha^{12}\}$ donde sí que hay cuatro raíces consecutivas: $\alpha, \alpha^2, \alpha^3, \alpha^4$.

Así, finalmente el polinomio generador será:

$$g(x) = m_1(x) \cdot m_3(x) = x^8 + x^7 + x^6 + x4 + 1$$

6 Códigos convolucionales

Los códigos convolucionales se diferencian de los códigos bloque lineales que hemos estado viendo hasta ahora en que tienen memoria, y su salida se calcula mediante a una operación que recuerda a una convolución.

Para entender mejor su funcionamiento vamos a utilizar el ejemplo mostrado en la Figura 2.

Como se puede observar, el codificador consta de un registro de desplazamiento con dos retardos. Cada nuevo bit introducido a la entrada del codificador, m_j, se combina con los dos bits anteriores almacenados en el registro, m_{j-1} y m_{j-2} para formar dos bits codificados que constituirán la salida del codificador, x'_j y x''_j.

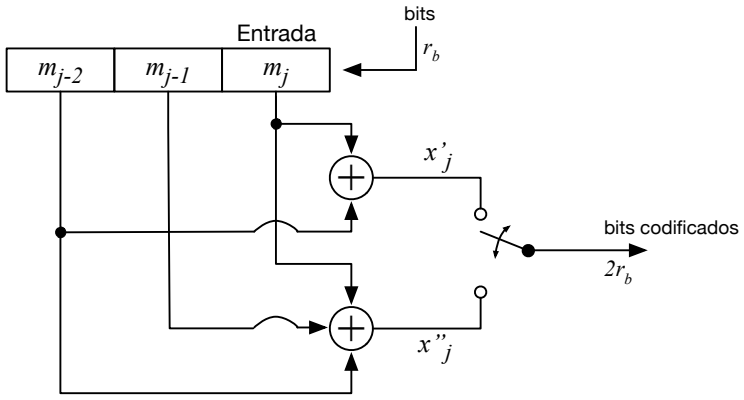


Figura 2: Esquema de un codificador convolucional con $n = 2$, $k = 1$ y $L = 2$.

A partir de la figura podemos ver que los bits que se generan a la salida responden a la siguiente expresión:

$$x'_j = m_j \oplus m_{j-2}$$
$$x''_j = m_j \oplus m_{j-1} \oplus m_{j-2}$$

Estos bits finalmente se intercalan entre sí para obtener la trama binaria de salida:

$$\mathbf{x} = (x'_1\ x''_1\ x'_2\ x''_2\ x'_3\ x''_3\ \cdots)$$

Podemos ver que se trata de un codificador con una tasa de codificación $R_c = 1/2$, ya que por cada bit a la entrada ($k = 1$) se obtienen dos bits a la salida ($n = 2$). Además, al número de retardos que introduce el sistema, en nuestro caso $L = 2$ se le denomina **memoria** del codificador.

El ejemplo mostrado, por tanto, se corresponde con un código convolucional (n, k, L) con $n = 2$, $k = 1$ y $L = 2$.

Si suponemos un estado inicial en el que todos los registros estén a cero ($m_{j-1} = m_{j-2} = 0$) podemos a partir de ahí intentar representar todos los posibles estados del codificador en función de las posibles entradas y obtener así una representación en forma de árbol como la mostrada en la Figura 3.

El codificador podrá estar en cuatro estados distintos, marcados por los valores de los bits almacenados en el registro: (m_{j-2}, m_{j-1}). Por claridad, denominamos a los estados por letras: $a = 00$, $b = 01$, $c = 10$ y $d = 11$. Si, por ejemplo, el codificador se encuentra inicialmente en el estado a (00) y el bit a la entrada es $m_j = 1$, el nuevo estado del codificador será el b (01). En la Figura 4 se muestra este caso y la evolución si la secuencia a la entrada es 1101.

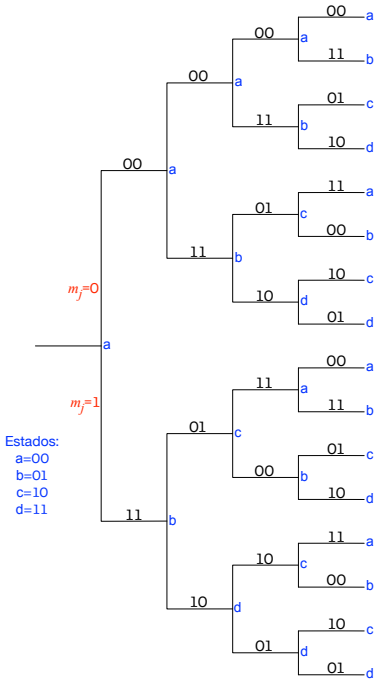


Figura 3: Diagrama de árbol para el código convolucional $(2, 1, 2)$ del ejemplo.

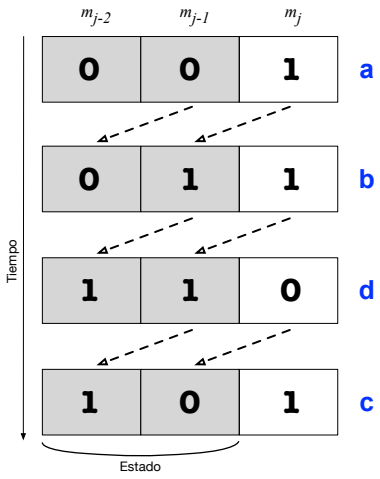


Figura 4: Evolución de los estados en un codificador convolucional.

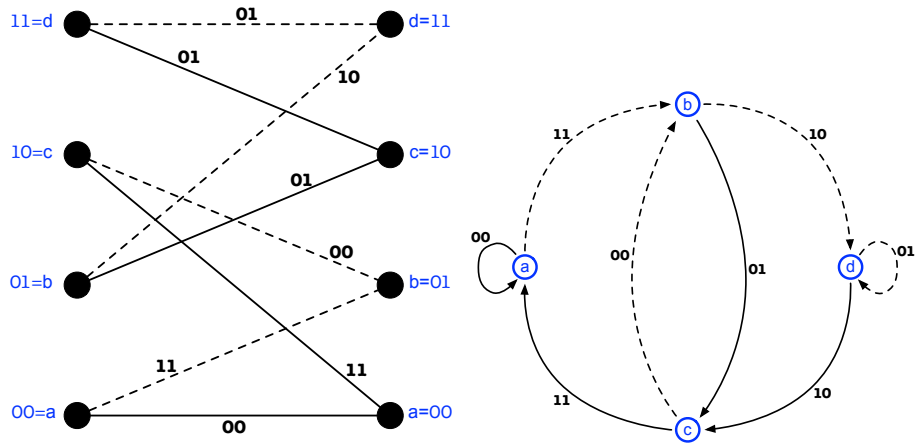


Figura 5: Diagrama de rejilla (a) y diagrama de estados (b) para el código convolucional $(2, 1, 2)$ del ejemplo.

Además de mostrar las transiciones entre estados del codificador, el diagrama de árbol también nos muestra la salida del codificador cuando se pasa de un estado a otro.

Así, por ejemplo, si partimos como antes del estado 00 y el bit de entrada al codificador es $m_j = 0$, entonces la salida será:

$$x'_j = m_j \oplus m_{j-2} = 0 \oplus 0 = 0$$
$$x''_j = m_j \oplus m_{j-1} \oplus m_{j-2} = 0 \oplus 0 \oplus 0 = 0$$

La salida será por tanto $(x'_j x''_j) = (0\,0)$. Por otra parte, si la entrada fuese $m_j = 1$, entonces:

$$x'_j = m_j \oplus m_{j-2} = 1 \oplus 0 = 1$$
$$x''_j = m_j \oplus m_{j-1} \oplus m_{j-2} = 1 \oplus 0 \oplus 0 = 1$$

En este caso la salida es $(x'_j x''_j) = (1\,1)$. Es decir, del estado a se puede pasar al estado a de nuevo si la entrada es 0 (produciendo una salida 0 0) o bien al estado b, si la entrada es 1 (siendo en este caso la salida 1 1).

Esto es lo que podemos ver representado en el árbol de codificador. Si ahora suponemos que estamos en el estado $d = 11$ y que la entrada es 0, entonces ahora tendremos:

$$x'_j = m_j \oplus m_{j-2} = 0 \oplus 1 = 1$$
$$x''_j = m_j \oplus m_{j-1} \oplus m_{j-2} = 0 \oplus 1 \oplus 1 = 0$$

Habremos pasado del estado $d = 1\,1$ al $c = 1\,0$ produciendo una salida igual a 1 0. Siguiendo este mismo proceso podemos ir construyendo todas las ramas del árbol, y si nos damos cuenta a partir de un determinado momento se va a ir repitiendo, dado que el codificador sólo cuenta con dos retardos, y por tanto la salida nunca va a depender de más de 2 bits anteriores al actual.

Una forma más compacta de representar todas las posibles transiciones entre estados en un codificador convolucional es mediante un diagrama de rejilla o un diagrama de estados, tal y como se muestra en la figura 5.

Ambas representaciones muestran esencialmente lo mismo: las distintas transiciones posibles entre los estados del codificador y los bits que se obtienen a la salida para cada una de estas transiciones.

Una última forma de analizar este tipo de códigos es haciendo uso de polinomios en D (D representa un retardo unidad, equivalente a la z^{-1} de la Transformada Z). Así, utilizando esta notación, nuestro codificador de ejemplo se podría representar mediante los siguientes polinomios generadores:

$$G_1(D) = 1 + D^2$$
$$G_2(D) = 1 + D^1 + D^2$$

Esta representación nos permite obtener de forma muy sencilla la salida del codificador a cualquier secuencia binaria de entrada. Si, por ejemplo, tuviésemos a la entrada la secuencia 110111001000, que puede escribirse de forma polinomial como $1 + D + D^3 + D^4 + D^5 + D^8$, la salida vendrá dada por:

$$\begin{aligned} X'_j(D) &= M(D)G_1(D) = (1 + D + D^3 + D^4 + D^5 + D^8)(1 + D^2) = \\ &= 1 + D + D^2 + D^4 + D^6 + D^7 + D^8 + D^{10} = 11101011101 \end{aligned}$$

$$\begin{aligned} X''_j(D) &= M(D)G_2(D) = (1 + D + D^3 + D^4 + D^5 + D^8)(1 + D^1 + D^2) = \\ &= 1 + D^5 + D^7 + D^8 + D^9 + D^{10} = 10000101111 \end{aligned}$$

Finalmente, la salida se obtiene intercalando los bits anteriores:

$$x_j = 11\ 10\ 10\ 00\ 10\ 01\ 10\ 11\ 11\ 01\ 11$$

6.1 Decodificación. Algoritmo de Viterbi

Existen varias formas de decodificar una secuencia binaria obtenida con un codificador convolucional. Nosotros no obstante nos vamos a centrar en estudiar el algoritmo de Viterbi, que permite implementar un receptor de máxima verosimilitud aunque a costa de un gran consumo de memoria por parte del decodificador.

Vamos a seguir con nuestro codificador de ejemplo, que recordamos venía dado por las siguientes expresiones:

$$\begin{aligned} x'_j &= m_j \oplus m_{j-2} \\ x''_j &= m_j \oplus m_{j-1} \oplus m_{j-2} \end{aligned}$$

Vamos a suponer que partimos del estado inicial $a = 00$ y que también terminamos en ese mismo estado (para ello basta con añadir dos ceros al final de la secuencia recibida). Supongamos también que la secuencia transmitida es 11010100 (ya hemos incluido los dos ceros al final para asegurar que el último estado sea el 00. La salida del codificador para esta secuencia de entrada es: 11 10 10 00 01 00 01 11.

Supondremos por último que el canal es ideal, y que por tanto la secuencia recibida es exactamente igual a la transmitida, sin errores.

En la Figura 6 se muestran las dos primeras iteraciones del algoritmo. La secuencia de entrada se muestra en la parte inferior, y en el interior de cada nodo se muestra la distancia Hamming acumulada hasta ese punto. Partiendo del estado 00 existen dos opciones:

- Ir al estado $b = 01$, con lo que la salida del codificador sería 11, tal y como indicamos junto a la flecha de la transición. En este caso la distancia Hamming entre esta salida (11) y la entrada al codificador (11) será cero.
- Continuar en el estado $a = 00$, siendo en este caso la salida del codificador 00. La distancia Hamming entre la salida (00) y la entrada (11) es de dos.

En la segunda iteración repetimos el proceso, partiendo de los dos nodos que tenemos ahora "activos". Tenemos cuatro opciones:

- Pasar del estado a al a, sumando uno a la distancia Hamming acumulada, con lo que ahora sería de tres.
- Pasar del estado a al b, obteniendo una distancia acumulada de tres.
- Pasar del estado b al c, que supone añadir dos a la distancia acumulada hasta entonces.
- Pasar del estado b al d, que suma cero a la distancia acumulada, manteniéndose ésta en este valor, por tanto.

Llegamos ahora a la tercera iteración, mostrada en la Figura 7. El proceso es idéntico al que hemos seguido hasta ahora, aunque ahora nos encontramos con que a cada estado podemos llegar desde dos estados distintos. Por ejemplo, al estado d se puede llegar desde el propio estado d, añadiendo dos a la distancia acumulada, que totalizaría dos, o bien desde el estado b, siendo en este caso la distancia acumulada igual a tres (tres del estado anterior más cero debido a esta transición). Lo que haremos en este caso es descartar aquel camino con una mayor distancia

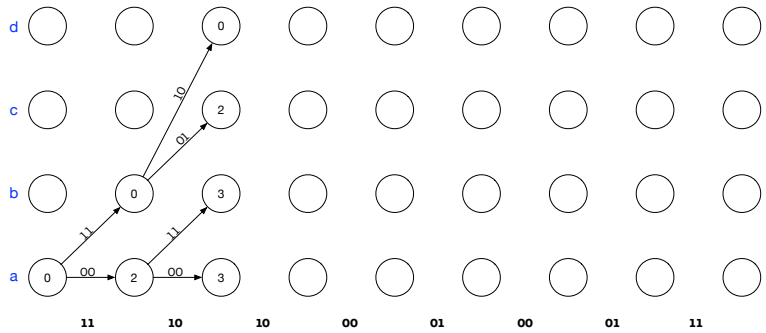


Figura 6: Primeras dos iteraciones del algoritmo de Viterbi.

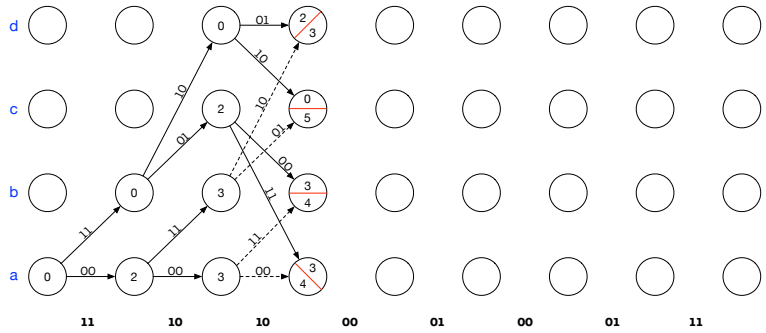


Figura 7: Tercera iteración del algoritmo de Viterbi.

acumulada (el que viene del nodo b, y que está marcado con línea de puntos en la figura), dejando únicamente "vivo" el camino con menor distancia Hamming acumulada.

En la figura 8 se muestra cómo quedaría el diagrama una vez que hayamos eliminado los caminos con mayor distancia acumulada en cada uno de los nodos.

En la Figura 9 se muestra la siguiente iteración del algoritmo, con los caminos con mayor distancia acumulada marcados con línea discontinua.

Por último, en la figura 10 se muestra el estado final del algoritmo, una vez que hemos llegado al último estado. El camino con menor distancia Hamming acumulada se ha destacado en línea más gruesa. Podemos comprobar que la salida del sistema para este camino coincide de forma exacta con la entrada, algo que no nos debería sorprender, pues hemos supuesto que no ha habido ningún error en la transmisión.

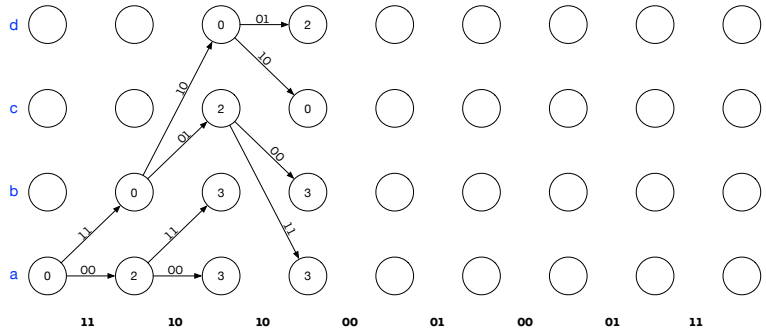


Figura 8: tercera iteración del algoritmo de Viterbi, tras eliminar los caminos con mayor distancia Hamming acumulada.

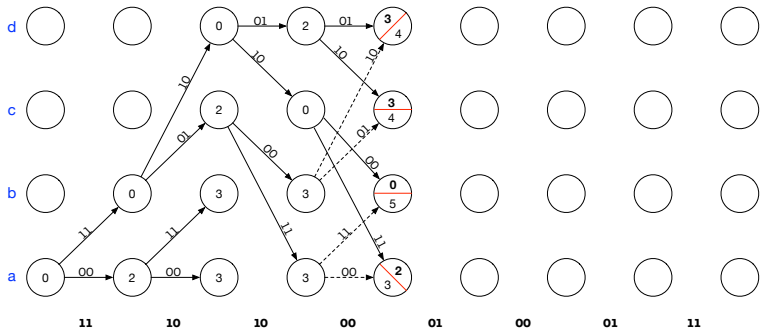


Figura 9: Cuarta iteración del algoritmo de Viterbi.

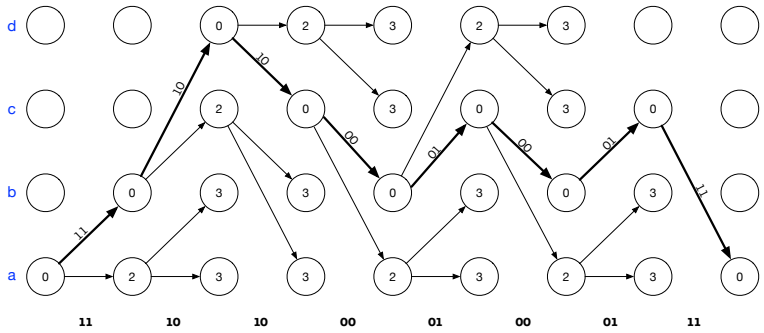


Figura 10: Resultado final del algoritmo de Viterbi.

7 Problemas propuestos

Problema 2.1

Para un código de repetición con $n = 5$, y una probabilidad de error de bit $p_e = 10^{-3}$, calcule:

a. La eficiencia del codificador.

b. La distancia mínima

c. El número de errores que pueden ser detectados

d. El número de errores que pueden ser corregidos.

e. La probabilidad de que se produzca un error y no sea detectado.

f. La probabilidad de que se produzca un error en la decodificación.

NOTA: Considere que $P(i,n) \approx \binom{n}{i} \cdot p_e^i$

RESULTADO:

1. $R_c = 1/5$
2. $d_{min} = 5$
3. 4 errores
4. 2 errores

5. $p = 10^{-15}$
6. $p \approx 10^{-8}$

Problema 2.2

Calcule la distancia Hamming entre la palabra 0100101 y las siguientes palabras: 0111111, 1010111 y 1101000.

RESULTADO:

3, 4 y 4 respectivamente.

Problema 2.3

Calcule la probabilidad de que una palabra tenga a) errores detectados b) errores no detectados y c) no tenga errores cuando usamos un código de paridad con $n = 4$ y $p_e = 0.1$.

NOTA: Considere que $P(i,n) = \begin{pmatrix} n \\ i \end{pmatrix} \cdot p_e^i \cdot (1-p_e)^{n-i}$

RESULTADO:

1. $p \approx 0.2916$
2. $p \approx 0.0486$
3. $p = 0.6561$

Problema 2.4

Un código tiene las siguientes siete palabras código: 0010, 0100, 1110, 1000, 1010, 1100, 0110. ¿Es lineal?

RESULTADO:

No.

Problema 2.5

Un código de repetición puede verse como un caso particular de un código bloque lineal. Para el caso $(4,1)$, ¿Cuáles serían sus matrices generadora y de comprobación de paridad?

RESULTADO:

$G = (1111)$

Problema 2.6

La matriz de codificación de un código lineal Reed-Muller (3,1) es:

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Determine la tasa del código, su distancia mínima y la tabla de síndromes.
PISTA: El cálculo de la tabla de síndromes es más sencillo si antes se transforma el código en uno sistemático.

RESULTADO:

$R_c = 1/2$, $d_{min} = 4$

Error	Síndrome
00000001	0001
00000010	0010
00000100	0100
00001000	1000
00010000	0111
00100000	1011
01000000	1101
10000000	1110

Problema 2.7

Dada la matriz generadora para un código lineal $(7, 3)$:

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

a. Construya la matriz generadora de un código sistemático equivalente.

b. Encuentre la matriz de control de paridad.

c. Construya la tabla de síndromes

d. Calcule la distancia mínima del código. ¿Qué capacidad correctora de errores tiene este código?

e. Compare las prestaciones de este código con las de un Hamming $(7, 4)$. ¿Existe alguna relación entre ambos?

f. Calcule la palabra codificada cuando a la entrada tenemos al palabra 101. Verifique que su síndrome es $\mathbf{0}$.

RESULTADO:

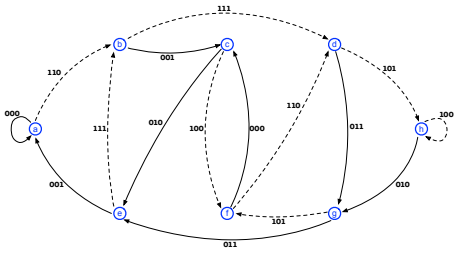
1. $\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$
2. $\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$
3. 0000001 → 0001
   0000010 → 0010
   0000100 → 0100
   0001000 → 1000
   0010000 → 1110
   0100000 → 0111
   1000000 → 1101
4. $d_{min} = 4 \Rightarrow$ Se pueden detectar hasta 3 errores y corregir 1 error.
5. El Hamming tiene mayor eficiencia, puede corregir los mismos errores, pero sólo puede detectar 2.
6. 101 → 1010011

Problema 2.8

Construya el árbol y el diagrama de estados para un código convolucional $(3, 1, 3)$ definido por las siguientes ecuaciones:

$$x'_j = m_j$$
$$x''_j = m_{j-2} \oplus m_j$$
$$x'''_j = m_{j-3} \oplus m_{j-1}$$

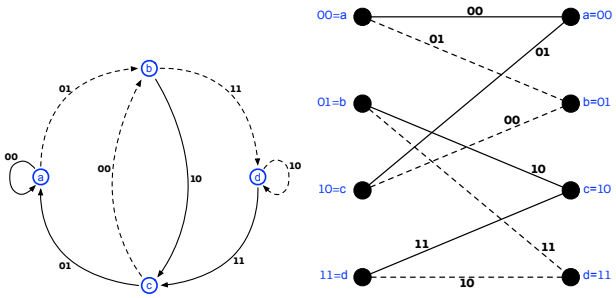
RESULTADO:



Problema 2.9

Considere un código convolucional de tasa $R_c = 1/2$ con matriz generadora $\mathbf{G} = \begin{bmatrix} D & D^2 + 1 \end{bmatrix}$. Dibuje su diagrama de estados y su diagrama de rejilla etiquetando cada transición con el bit de entrada y los dos bits de salida.

RESULTADO:



(a) (b)

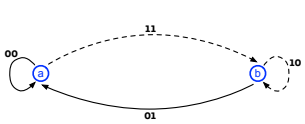
Problema 2.10

Considere un código convolucional de tasa $R_c = 1/2$ con matriz generadora $\mathbf{G} = [1 \quad D + 1]$. Dibuje su diagrama de estados y su diagrama de rejilla etiquetando cada transición con el bit de entrada y los dos bits de salida.

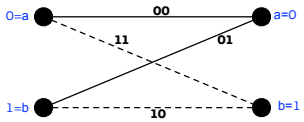
RESULTADO:

Problema 2.11

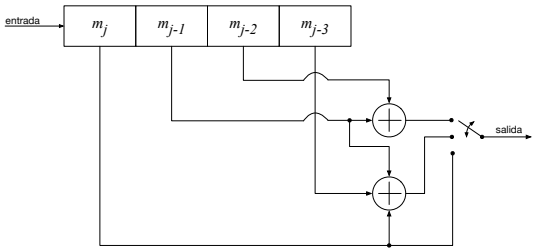
Determine la salida del codificador $(3, 2, 1)$ de la figura para la entrada (1101011101110000).



(c)



(d)



RESULTADO:

$x'_j = m_{j-1} \oplus m_{j-2} =$ 0101111001100100
$x''_j = m_j \oplus m_{j-1} \oplus m_{j-3} =$ 1010011000100110
$x'''_j = m_j =$ 1101011101110000

Problema 2.12

Para el codificador utilizado en los ejemplos del texto:

$$x'_j = m_j \oplus m_{j-2}$$
$$x''_j = m_j \oplus m_{j-1} \oplus m_{j-2}$$

Obtenga, mediante el algoritmo de Viterbi, la secuencia decodificada siendo ahora la secuencia recibida: 10 10 10 01 01 01 01 11.

RESULTADO:

11 10 10 00 01 00 01 11

Problema 2.13

Un código Golay (23,12) es un código perfecto capaz de corregir 3 errores. Este código se puede definir de forma cíclica mediante cualquiera de los siguientes polinomios generadores:

$$g_1(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1$$
$$g_2(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$$

a. Verifique que efectivamente se trata de un código perfecto.

b. Codifique la palabra $\mathbf{b} = [010001100001]$ de forma sistemática empleando $g_1(x)$ y de forma directa multiplicando $b(x)$ por $g_2(x)$.

.

RESULTADO:

De forma sistemática: $\mathbf{c} = [111100001010100011000001]$