ESTRATEGIAS DE DECODIFICACIÓN

Gabriela Sanromán

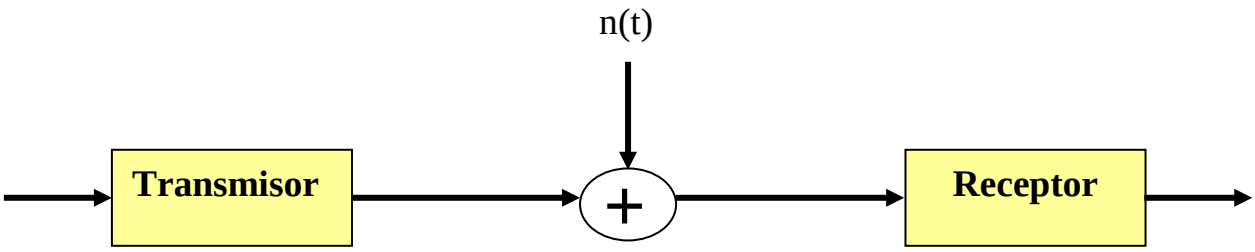
sanroman_gabriela@speedy.com.ar

Febrero de 2004

ÍNDICE

1. INTRODUCCIÓN

El problema fundamental de la comunicación es reproducir en un punto, de forma exacta o aproximada, un mensaje seleccionado en otro punto. El ruido es un valor aleatorio que se suma a la información útil transformando la señal que llega al receptor, también en aleatoria. Sabiendo que la información enviada pertenece a un conjunto finito de mensajes, el receptor debe detectar el mensaje recibido (no estimarlo), para lo cual, en caso de no poder traducirlo exactamente debido a la adición del ruido, deberá tomar la decisión por uno de los posibles mensajes válidos. Esto se logra al costo de cierta redundancia en la codificación y aumento de la complejidad.



Sistema de transmisión

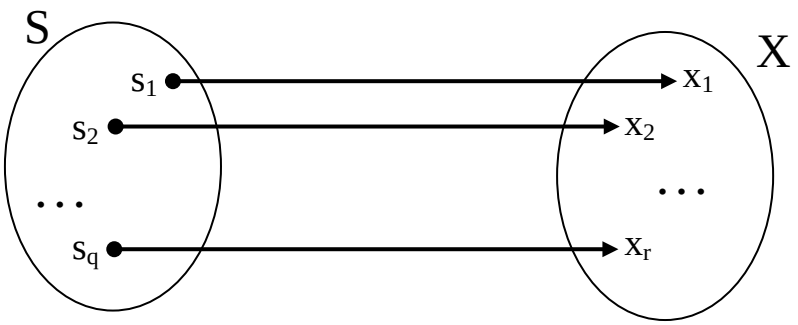
De alguna manera de lo mencionado surge que una buena técnica de decodificación permite obtener la decisión correcta.

El objetivo del siguiente trabajo es analizar las diferentes estrategias utilizadas en el diseño de codificadores/decodificadores, para luego realizar una comparación de las distintas técnicas de decodificación mencionadas.

2. CODIFICACIÓN

La codificación es la operación que permite pasar del alfabeto fuente al alfabeto código.

Siendo $S = \{s_1, s_2, \ldots, s_q\}$ el conjunto de símbolos del alfabeto fuente, se define un código como la correspondencia de todas las secuencias posibles de símbolos de S a secuencias de símbolos de algún otro alfabeto $X = \{x_1, x_2, \ldots, x_r\}$, alfabeto código.



Alfabeto fuente Alfabeto código

Esta definición es demasiado general para el estudio de la codificación, por lo tanto se tendrá en cuenta sólo aquellos códigos que posean ciertas propiedades suplementarias.

La primera de estas propiedades es que el código constituya un *bloque*. Esto es un código que asigna cada uno de los símbolos del alfabeto fuente S a una secuencia fija del alfabeto código X. Esas secuencias fijas (secuencias de x_j) reciben el nombre de *palabras código.* Se denominará X_i a la palabra de código que corresponde al símbolo s_i. Hay que notar que X_i constituye una secuencia de x_j 's.[1]

La segunda propiedad es que todas las palabras X_i sean distintas, lo que se denomina *no singular*. Se denomina código no singular aquel bloque de código en el cual a cada símbolo codificado le corresponde una única codificación.

Símbolos de la fuente	Código
s_1	0
s_2	11
s_3	00
s_4	01

Ejemplo de bloque de código no singular

La tercera de las propiedades es que el código sea *unívocamente decodificable.* Para poder definir esta propiedad, se necesita conocer una nueva definición. La extensión de orden n de un bloque de código que hace corresponder los símbolos s_i con las palabras de código X_i , es el bloque de código que hace corresponder las secuencias de símbolos de la fuente (s_{i1}, s_{i2},....., s_{in}) con las secuencias de las palabras de código (X_{i1}, X_{i2}, ..., X_{in}).

Símbolos de la fuente	Código	Símbolos de la fuente	Código
s_1s_1	00	s_3s_1	000
s_1s_2	011	s_3s_2	0011
s_1s_3	000	s_3s_3	0000
s_1s_4	001	s_3s_4	0001
s_2s_1	110	s_4s_1	010
s_2s_2	1111	s_4s_2	0111
s_2s_3	1100	s_4s_3	0100
s_2s_4	1101	s_4s_4	0101

Extensión de segundo orden del bloque de código no singular del ejemplo anterior

Ahora se puede decir que un código es unívocamente decodificable si, y solamente si, su extensión de orden n es no singular para cualquier valor finito de n. Según la definición se puede observar que el ejemplo citado anteriormente no satisface la condición de unívocamente decodificable.

Por último un código unívocamente decodificable se denomina *instantáneo* cuando es posible decodificar las palabras de una secuencia sin precisar el conocimiento de los símbolos que las suceden.

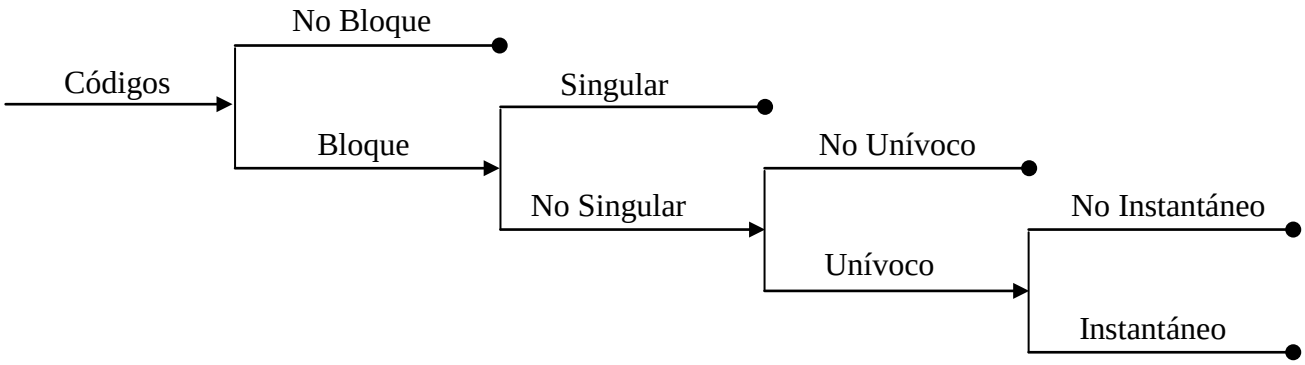
Símbolos de la fuente	Código *A*	Código *B*	Código *C*
s_1	00	0	0
s_2	01	10	01
s_3	10	110	011
s_4	11	1110	0111

Ejemplos de códigos unívocamente decodificables, *A* y *B* instantáneos, *C* no instantáneo.

En el ejemplo aparecen tres códigos unívocamente decodificables. Los códigos *A* y *B* son códigos instantáneos, existe la capacidad de reconocer cuando una palabra código llega a su final. En el código *C* , en cambio, no se puede decodificar la sentencia en sus palabras según se van recibiendo, al recibir 01, por ejemplo, no se puede asegurar que corresponde al símbolo s_2 , en tanto no se haya recibido el símbolo siguiente. Por lo tanto el código *C* no es instantáneo.

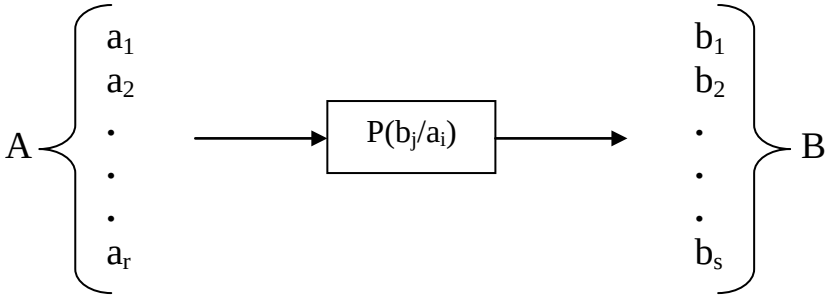
Sea $X_i = x_{i1}, x_{i2}, ..., x_{im}$ una palabra de un código, se denomina prefijo de esta palabra a la secuencia de símbolos $(x_{i1}, x_{i2}, ..., x_{ij})$, donde $j \leq m$. La condición necesaria y suficiente para que un código sea instantáneo es que ninguna palabra del código coincida con el prefijo de otra.

En el siguiente gráfico se muestra un resumen de las distintas clases de códigos tratadas precedentemente, hasta llegar a los códigos instantáneos y no instantáneos.



3. DECODIFICACIÓN

Un canal de información viene determinado por un alfabeto de entrada A = {a_1, a_2, ..., a_r}; un alfabeto de salida B = {b_1,b_2, ..., b_s}; y un conjunto de probabilidades condicionales $P(b_j/a_i)$. $P(b_j/a_i)$ es la probabilidad de recibir a la salida el símbolo b_j cuando se envía el símbolo de entrada a_i.



Canal de información

También se puede describir el canal de información disponiendo las probabilidades condicionales de la siguiente forma:

		Salidas			
		b_1	b_2	…….	b_s
Entradas	a_1	$P(b_1/a_1)$	$P(b_2/a_1)$	…….	$P(b_s/a_1)$
	a_2	$P(b_1/a_2)$	$P(b_2/a_2)$	…….	$P(b_s/a_2)$
	…….	…….	…….	…….	…….
	a_r	$P(b_1/a_r)$	$P(b_2/a_r)$	…….	$P(b_s/a_r)$

Cada fila corresponde a una entrada determinada siendo sus términos las probabilidades de obtener a la salida las diferentes b_j para una entrada fija. Se puede definir $P_{ij} = P(b_j/a_i)$. Por lo tanto la tabla de probabilidades se transforma en la matriz P:

$$\mathbf{P} = \begin{vmatrix} P_{11} & P_{12} & \ldots\ldots & P_{1s} \\ P_{21} & P_{22} & \ldots\ldots & P_{2s} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & & \\ P_{r1} & P_{r2} & \ldots\ldots & P_{rs} \end{vmatrix}$$

Cada fila de la matriz corresponde a una entrada del canal y cada columna a una salida.

3.1. PROBABILIDAD DE ERROR Y REGLAS DE DECISIÓN

Considerando un canal con un alfabeto de entrada $A = \{a_1, a_2, ..., a_r\}$ y un alfabeto de salida $B = \{b_1, b_2, ..., b_s\}$, se denomina regla de decisión, $d(b_j)$ a la función que especifica el símbolo de entrada único que corresponde a cada símbolo de salida: $d(b_j) = a_i$. Un canal de r entradas y s salidas admite r^s reglas de decisión diferentes. Se elegirá la regla de decisión que haga mínima la probabilidad de error.

La posibilidad de error de un canal será mínima con la regla de decisión que asigna a cada símbolo de salida el símbolo de entrada de mayor probabilidad. Esta regla de decisión recibe el nombre de *máxima*

posibilidad condicional y depende de las probabilidades a priori. La regla de decisión que se conoce como de *máxima posibilidad*, es independiente de las probabilidades a priori.

4. CÓDIGOS

4.1. RENDIMIENTO Y REDUNDANCIA DE UN CÓDIGO

El primer teorema de Shannon demuestra la existencia de una unidad común con la que puede medirse cualquier fuente de información. El valor de símbolo de una fuente *S* puede definirse en términos del número equivalente de dígitos binarios necesario para representarlo. El teorema establece que el valor medio de un símbolo *S* es *H(S)*, llamado *entropía de la fuente*.

Supongamos que *L* es la longitud media de un código de la fuente *S*, L no puede ser inferior a *H(S)*. Según esto, se define el rendimiento de un código como:

$$\eta = \frac{H(S)}{L}$$

La redundancia es la información superflua o innecesaria para interpretar el significado de los datos originales. La introducción de redundancia en la codificación tiene como finalidad mejorar la fiabilidad de la transmisión. Se puede definir como:

$$\mathrm{R} = 1 - \eta$$

4.2. CÓDIGOS COMPACTOS

Son códigos de longitud variable que minimiza la longitud promedio (entropía), se los llama código de alto rendimiento. Son códigos instantáneos, por lo que ninguna palabra de código puede aparecer como prefijo de una de mayor longitud.

La codificación consiste en asignar las palabras de menor cantidad de símbolos a aquel mensaje de mayor probabilidad, así sucesivamente:

$$P(1) \geq P(2) \geq \ ... \geq P(n-1) \geq P(n)$$

$$L(1) \leq L(2) \leq \ldots \leq L(n-1) \leq L(n)$$

Estos códigos presentan el inconveniente de que la distribución de la fuente puede no ser conocida cuando se diseña el código.

Si ninguna palabra es prefijo de otra, podrá decodificarse directamente a su recepción cualquier secuencia de símbolos formada por palabras de código. Para ello se observa una secuencia hasta reconocer una subsecuencia formada por una palabra código completa. La secuencia debe ser precisamente la palabra código, ya que no puede ser prefijo de otra palabra. De esta manera puede procederse a decodificar las palabras, una por una, sin pérdida de tiempo en la operación. Se podría agregar un bit de paridad para control de error.

4.2.1. CÓDIGO DE SHANNON – FANO

Se toman los mensajes y se los ordena con probabilidades decrecientes de acuerdo con un análisis de tráfico que debe ser realizado previamente. Se agrupan los mensajes de tal forma que la suma de las probabilidades de ocurrencia sean aproximadamente equivalentes. Cada grupo se lo designa mediante el símbolo 1 o 0 y así sucesivamente. A continuación se muestra con un ejemplo este código:

	Probabilidad	Secuencia de codificación				Código	Entropía
m_1	0,4	1	1			11	2 X 0,4
m_2	0,2		0			10	2 X 0,2
m_3	0,15	0	1	1		011	3 X 0,15
m_4	0,1			0		010	3 X 0,1
m_5	0,06		0	1	1	0011	4 X 0,06
m_6	0,04				0	0010	4 X 0,04
m_7	0,03			0	1	0001	4 X 0,03
m_8	0,02				0	0000	4 X 0,02

Entropía promedio = 2,55

Rendimiento = 95,5%

4.2.2. CÓDIGO DE HUFFMAN

Los mensajes se disponen en una columna en orden decreciente de probabilidades las que son tomadas iguales que las del código de Shannon-Fano para poder compararlas. Las dos probabilidades menores se suman para dar lugar a una nueva probabilidad, la que se sitúa con las demás en una nueva columna ordenada decrecientemente por probabilidad. Este procedimiento se repite hasta terminar con una probabilidad igual a 1. Los códigos se obtienen a partir del resultado hacia el mensaje, tomando los ceros y unos que deban pasarse en ese camino en el sentido contrario que se empleó anteriormente.

Código								
0	m_1	0,4	0,4	0,4	0,4	0,4	0,4	**0, 6 1**
111	m_2	0,2	0,2	0,2	0,2	**0,25**	**0,35 1**	0,4 **0**
110	m_3	0,15	0,15	0,15	0,15	0,2 **1**	0,25 **0**	
100	m_4	0,1	0,1	0,1	**0,15 1**	0,15 **0**		
1010	m_5	0,06	0,06	**0,09 1**	0,1 **0**			
10110	m_6	0,04	**0,05 1**	0,06 **0**				
101111	m_7	0,03 **1**	0,04 **0**					
101110	m_8	0,02 **0**						

Rendimiento = 98%

4.3. CÓDIGOS DE BLOQUE

Son códigos no instantáneos donde cada palabra tiene exactamente n dígitos binarios, de los cuales:

- m dígitos se asocian con la información
- los restantes k = n – m dígitos se usan para detección y corrección de errores.

4.3.1. DETECCIÓN Y CORRECCIÓN DE ERRORES POR CHEQUEO DE PARIDAD

4.3.1.1. CÓDIGOS DE DETECCIÓN DE UN ERROR

En las primeras *n*-1 posiciones ponemos *n*-1 dígitos de información. En la *n*-ésima posición ponemos un 1 o un 0, de modo que las *n* posiciones en total tengan un número par de dígitos (chequeo de paridad par). Cualquier error simple en la transmisión producirá un número impar de 1’s en el símbolo recibido.

La redundancia de estos códigos es:

$$R = \frac{n}{n-1} = 1 + \frac{1}{n-1}$$

Pareciera que para lograr una redundancia baja deberíamos hacer que *n* sea muy grande. Sin embargo al aumentar *n*, la probabilidad de que exista por lo menos un error en un símbolo aumenta, y el riesgo de un doble error, el cual pasaría inadvertido, también. El método usado para cualquier error simple se denomina chequeo de paridad.

4.3.1.2. CÓDIGOS DE CORRECCIÓN DE UN ERROR SIMPLE

Se asignan *m* de las *n* posiciones disponibles como posiciones de información. Se asignan las *k* posiciones restantes como posiciones de chequeo. Los valores en estas *k* posiciones se determinan en el proceso de codificación por chequeos de paridad par sobre posiciones de información seleccionadas. Se aplican los *k* chequeos de paridad, en orden, y cada vez que el chequeo de paridad produce el valor observado en su posición de chequeo se escribe un 0, mientras que cada vez que los valores asignado y observado no coinciden, se escribe un 1. Cuando se escribe de derecha a izquierda esta secuencia de *k* 0's y 1's se forma un número binario denominado *número de chequeo*. Queremos que este número indique la posición de cualquier error simple. El valor cero significa que no existe error.

El número de chequeo debe describir $m + k + 1$ posiciones diferentes. De modo que:

$$2^k \geq m + k + 1$$

Como $n = m + k$, la expresión anterior se transforma en.:

$$2^m \leq \frac{2^n}{n+1}$$

n	m	k correspondiente
1	0	1
2	0	2
3	1	2
4	1	3
5	2	3
6	3	3
7	4	3
8	4	4
9	5	4
10	6	4
11	7	4
12	8	4
13	9	4
14	10	4
15	11	4
	Etc.	

Tabla 1: Máximo *m* para un *n* dado

Ya que el número de chequeo debe entregar la posición de cualquier error en un símbolo del código, cualquier posición que tenga un 1 a la derecha de su representación binaria debe hacer que el primer chequeo falle:

1 = 1

3 = 11

5 = 101

7 = 111

9 = 1001 Etc.

Por lo tanto el primer chequeo de paridad debe usar las posiciones 1, 3, 5, 7, 9,...

El segundo chequeo de paridad debe usar aquellas posiciones que tienen 1 en el segundo dígito de derecha a izquierda de su representación binaria.

2 = 10
3 = 11
6 = 110
7 = 111
10 = 1010
11 = 1011
Etc.

Y el tercer chequeo de paridad:

4 = 100
5 = 101
6 = 110
7 = 111
12 = 1100
13 = 1101
14 = 1110
15 = 1111
20 = 10100
Etc.

Falta decidir para cada chequeo de paridad qué posiciones van a contener información y cuáles el número de chequeo. Escoger las posiciones 1, 2, 4, 8.. como posiciones de chequeo tiene la ventaja de hacer que cada posición de chequeo sea independiente de las otras, es decir, que sólo dependan de los bits de información.

Número de chequeo	Posiciones de chequeo	Posición chequeada
1	1	1, 3, 5, 7, 9, 11, 13, 15, 17, ...
2	2	2, 3, 6, 7, 10, 11, 14, 15, 18, ...
3	4	4, 5, 6, 7, 12, 13, 14, 15, 20, ...
4	8	8, 9, 10, 11, 12, 13, 14, 15, 24, ...
...	**...**	**...**

Tabla II: Posiciones de chequeo

Ejemplo:

De la Tabla I: para $n = 7$, $m = 4$ y $k = 3$

De la Tabla II:

Posición de Chequeo	Posiciones chequeadas
1	1, 3, 5, 7
2	2, 3, 6, 7
4	4, 5, 6, 7

Posiciones de información: 3, 5, 6, 7

El código completo puede verse en la siguiente tabla:

Posición							Valor decimal del símbolo
1	2	3	4	5	6	7	
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1
0	1	0	1	0	1	0	2
1	0	0	0	0	1	1	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	5
1	1	0	0	1	1	0	6
0	0	0	1	1	1	1	7
1	1	1	0	0	0	0	8
0	0	1	1	0	0	1	9
1	0	1	1	0	1	0	10
0	1	1	0	0	1	1	11
0	1	1	1	1	0	0	12
1	0	1	0	1	0	1	13
0	0	1	0	1	1	0	14
1	1	1	1	1	1	1	15

Para este código existen 2^7 - 16 = 112 símbolos sin significado. En ocasiones se elimina el primer símbolo del código para evitar la combinación de todos ceros como un símbolo del código (o un símbolo de código más un error simple) ya que éste podría ser confundido con ausencia de mensaje. Así, quedarían todavía 15 símbolos “útiles” de código.

Ejemplo de Corrección:

Suponga que el símbolo 0111100 se contamina en su quinta posición quedando 0111000. Los chequeos entregan la secuencia 101, la cual indica que el bit erróneo está en la posición 5. El símbolo correcto se obtiene por tanto cambiando el 0 en la quinta posición a un 1.

4.3.1.3. CÓDIGOS DE CORRECCIÓN DE UN ERROR SIMPLE MÁS DETECCIÓN DE UN ERROR DOBLE

Se inicia con un código de corrección de un error simple. A éste código se agrega una posición más para chequear todas las posiciones anteriores, usando un chequeo de paridad par.

Existen tres casos:

1. No hay errores. Todos los chequeos de paridad incluyendo el último se satisfacen.
2. Error simple. El último chequeo falla ya sea que el error está en la información, las posiciones de chequeo original, o en la última posición de chequeo. El número de chequeo original da la posición del error, pero ahora el valor cero significa la última posición de chequeo.
3. Dos Errores. En todos los casos el último chequeo de paridad se satisface, y el número de chequeo indica alguna clase de error.

A modo de ejemplo de código de corrección de un error simple y uno doble, se tomará el código de siete posiciones del ejemplo anterior y se le

agregará una octava posición de modo que exista un número par de 1’s en las ocho posiciones. La siguiente tabla muestra el código:

Posiciones							
1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1
1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	1
0	1	0	0	1	0	1	1
1	1	0	0	1	1	0	0
0	0	0	1	1	1	1	0
1	1	1	0	0	0	0	1
0	0	1	1	0	0	1	1
1	0	1	1	0	1	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	0	0	0
1	0	1	0	1	0	1	0
0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1

4.3.2. MODELO GEOMÉTRICO

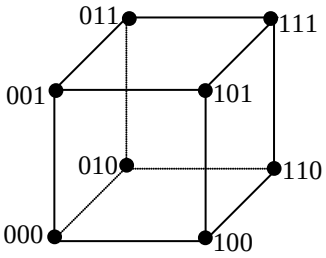
Consiste en identificar las distintas secuencias de 0's y 1's que forman los símbolos de un código con los vértices de un cubo *n*-dimensional unitario. Los símbolos del código en el modelo geométrico se convierten en puntos en los vértices de un cubo *n*-dimensional unitario. Se define una distancia o métrica, $D(a_i,b_i)$, conocida como distancia Hamming. Se define la distancia $D(a_i,b_i)$ entre dos puntos a_i y b_i como el número de coordenadas para las cuales a_i y b_i son diferentes. Por ejemplo:

$a_i = 101111$ $\quad b_i = 111100$ $\quad D(a_i,b_i) = 3$

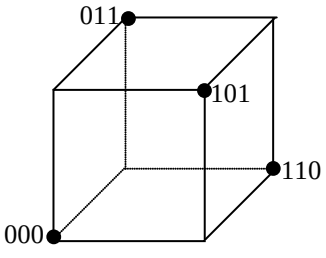
Aplicando estos conceptos en los siguientes códigos:

	Código *A*	Código *B*	Código *C*
	000	000	000
	001	011	111
	010	101	
	011	110	
	100		
	101		
	110		
	111		
Número de mensajes:	8	4	2

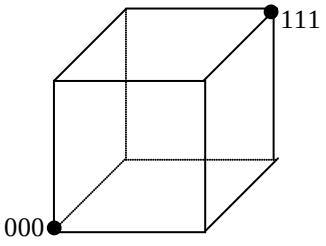
Representando los códigos en cubos tridimensionales:



Código *A*



Código *B*



Código *C*

La distancia mínima entre palabras de un código está íntimamente relacionada con su probabilidad de error, a mayor distancia mínima, la probabilidad de error será menor. Como es lógico, cuanto mayor es la distancia mínima, el número de palabras que puede alojarse en lo vértices de un cubo de n dimensiones es menor. La ventaja de poder representar un gran número de mensajes con un código, por un lado se equilibra, por el otro, con la de tener un canal con baja probabilidad de error. Las distancias mínimas en los códigos *A*, *B* y *C* son, respectivamente, 1, 2 y 3.

Los errores surgidos en la transmisión de una secuencia a_i de n bits dan lugar a que la secuencia recibida b_j sea distinta de ella. Si han aparecido *D* errores, la distancia de Hamming entre a_i y b_j será *D*. La probabilidad de recibir b_j al enviar a_i es la de que aparezcan un error en cada uno de los *D* en que difieren y no se produzca en los $n - D$ restantes. Para la decodificación, la regla de decisión consiste en seleccionar la palabra más cercana a b_j según el concepto de distancia de Hamming.

Por lo tanto, si la distancia mínima de Hamming es *D* el código es capaz de detectar *D* errores y de corregir $D - 1$ errores.

4.3.3. CÓDIGO DE HAMMNING

Podemos denotar un código de Hamming mediante un par *(n,m)*, donde:

- *n* es la longitud palabra de código
- *m* es el número de bits de datos de la palabra original sin codificar
- el número de bits de paridad será $k = n - m$.

La forma de una palabra de código es la siguiente:

$$M_0, M_1, ..., M_{m-1}, B_0, B_1, ….., B_{n-m-1}$$

De esta forma podemos expresar una palabra de código como

$$C_0, C_1, ..., C_{n-m-1}, C_{n-m}....., C_{n-1}$$

Por lo tanto a cada palabra original se le asignará unos bits de paridad para obtener la palabra de código, de forma que estos bits de paridad sirvan posteriormente para encontrar y corregir errores que se produzcan en la transmisión.

Cada uno de los bits de paridad que se añaden a una palabra original va a afectar a unas determinadas posiciones de la nueva palabra de código, de forma que tomarán un valor adecuado para que se cumpla el criterio de paridad (par o impar) preestablecido en las subcombinaciones afectadas por cada bit de paridad en la forma que se analizó la corrección de un error simple.

4.4.CÓDIGOS DE REDUNDANCIA CÍCLICA

Para el manejo de estos códigos se utiliza una notación polinómica, de forma que una palabra de código:

$$C = (c_0, ..., c_{n-1})$$

se interpretará como un polinomio de grado $n - 1$ (n cantidad de bits de la palabra codificada), y cada uno de los bits de la palabra de código será uno de los coeficientes de este polinomio:

$$C(x) = c_0 + c_1 x + ...+ c_{n-1} x^{n-1}$$

A su vez, una palabra original:

$$M = (m_0, ..., m_{k-1})$$

la interpretaremos como el polinomio de grado $k -1$, siendo k la cantidad de bits de la palabra original:

$$m(x) = m_0 + m_1 x + ...+ m_{k-1} x^{k-1}$$

Se define un polinomio divisor $g(x)$, también conocido como generador, de grado n-k.

Se genera $C(x)$ a partir del mensaje a transmitir en forma de polinomio $m(x)$ tal que sea divisible en forma exacta por el polinomio generador $g(x)$.

A su vez los bits de paridad forman una palabra que también se interpreta en forma de polinomio:

$$b(x) = b_0 + b_1 x + ….. + b_{n-k-1} x^{n-k-1}$$

Por lo tanto la palabra de código será:

$$C(x) = x^{n-k} m(x) - b(x)$$

Los pasos para obtener el código son:

- multiplicar la palabra original *m(x)* por x^{n-k}
- dividir $x^{n-k}\ m(x)$ por *g(x)*, el resto es el polinomio con bits de paridad

la palabra codificada será: $C(x) = x^{n-k}\ m(x) - b(x)$

Para la decodificación

La palabra recibida por el receptor la denotaremos por:

$$r(x) = r_0 + r_1\ x + \ldots\ldots + r_{n-1}\ x^{n-1}$$

realizando la siguiente división:

$r(x)$	$g(x)$
$s(x)$	$q(x)$

si no se producen errores en la transmisión $s(x) = 0$.

Existen distintos polinomios generadores en base a pruebas:

CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-ITU	$x^{16} + x^{12} + x^5 + 1$

o CRC- CCITT

CRC-12 se usa con palabras de datos de 6 bits, CRC-16 y CRC-ITU con palabras de datos de 8 bits.

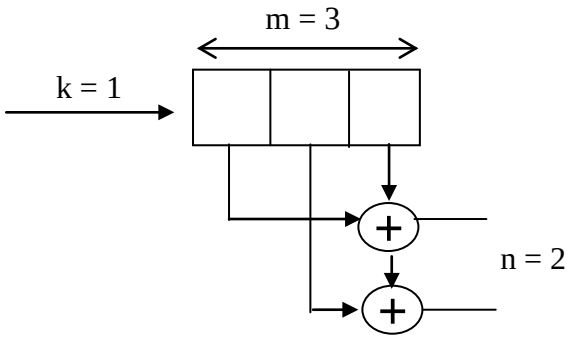
Según el polinomio generador usado detecta todos los errores simples, cualquier número de errores impares, todos los errores dobles y ráfagas de error.

4.5.CÓDIGOS CONVOLUCIONALES

Un código convolucional queda especificado por tres parámetros (n, k, m):

- *n* es el número de bits de la palabra codificada.
- *k* es el número de bits de la palabra de datos.
- *m* es la memoria del código o longitud restringida

Se define como ejemplo un codificador convolucional (2,1,3), es decir: un bit para representar la palabra de datos, dos bits de palabra de codificación por cada bit de palabra de datos y tres bits de longitud de registro.



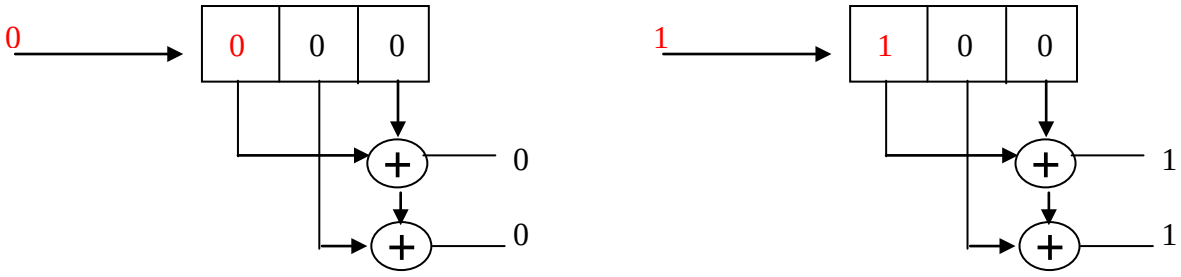
Codificador convolucional

Cuanto mayor sea *n*, más segura será la transmisión y menor será la probabilidad de error, eso sí, del mismo modo, más complejo será el diagrama de estados del codificador y más difícil su posterior decodificación. Hay que encontrar, la relación ideal entre calidad y complejidad.

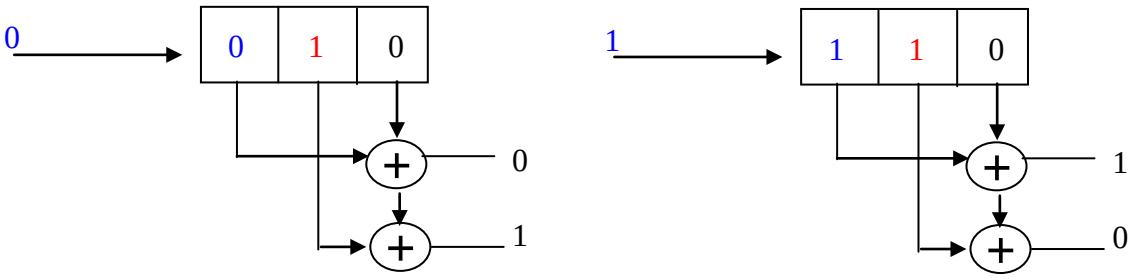
Un codificador convolucional puede ser fácilmente representado por un diagrama de estados de transición ya que tiene memoria finita. Para ello, procedemos al desarrollo de los diferentes estados de la máquina. Fijamos

que en el instante inicial el codificador está cargado con todo ceros, veamos los distintos estados posibles:

Cargado todo con ceros, ingresa un cero y un uno:



Ahora en la situación (1,0,0) se observa cómo codifica el sistema si se ingresa un cero y cómo si es un uno el bit que entra al canal:



Y así sucesivamente hasta completar un ciclo y con él, la máquina de estados. En este diagrama, cada estado del codificador convolucional se representa mediante una caja y las transiciones entre los estados vienen dadas por líneas que conectan dichas cajas. Para saber de una manera rápida en que estado se encuentra el codificador, basta con observar los dos bits del registro de memoria más cercanos de la entrada. Puede comprobarse interpretando la máquina de estados

correspondiente. Para el codificador de este ejemplo, el diagrama de estados correspondiente es:

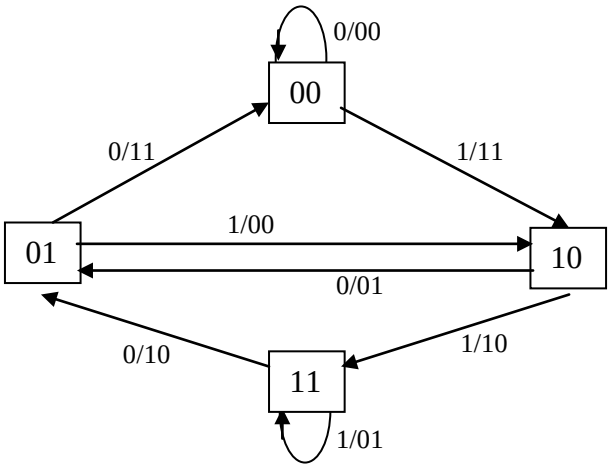


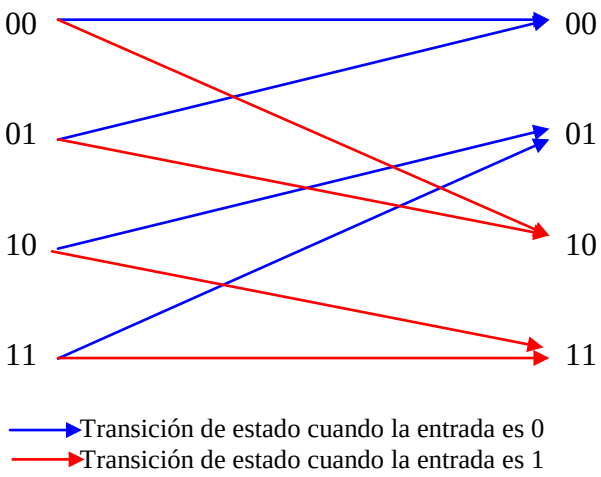
Diagrama de estados

En cada línea está representada la salida en función de la entrada y apunta al próximo estado. El número de líneas que salen de cada estado es, por lo tanto, igual al número de posibles entradas al codificador en ese estado, que es igual a 2k.

Una manera de representar las distintas transiciones y los caminos que éstas describen es mediante un Diagrama de Trellis. Una descripción de Trellis de un codificador convolucional muestra cómo cada posible entrada al codificador influye en ambas salidas y a la transición de estado del codificador. Un código de longitud restringida *m* tiene un Trellis con 2m-1 estados en cada intervalo t_i. Así que tendremos cuatro estados en t_i. De cada estado parten otros dos, uno si el bit enviado es un ‘1’ y otro si es un ‘0’.

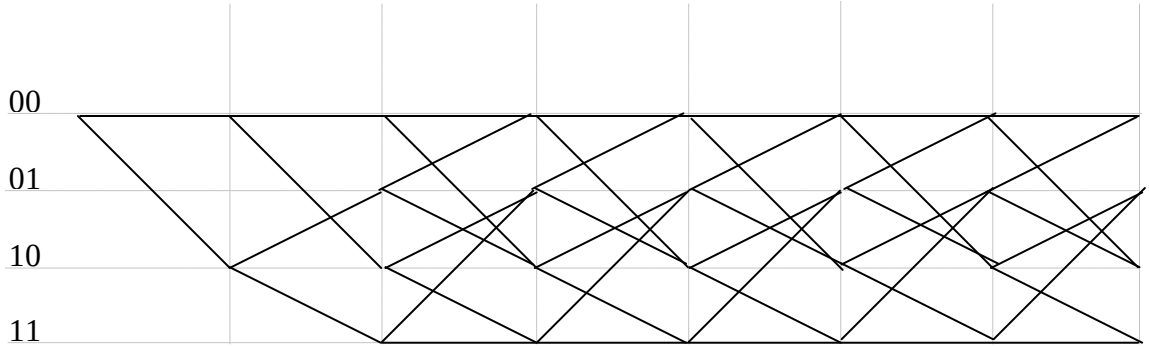
Estas son las transiciones de estados correspondientes al Diagrama de Trellis del ejemplo. Las flechas azules muestran cómo el codificador

cambia su estado si la entrada es un cero, y las flechas rojas cómo lo hace si la entrada es un uno:



Transiciones

El sistema tiene memoria: la codificación actual depende de los datos que se envían ahora y que se enviaron en el pasado. Por lo tanto, el diagrama completo que se obtiene para el ejemplo es:



El proceso de codificación evoluciona según el diagrama de estados. Entran al codificador *k* bits a la vez y las correspondientes *n* salidas se transmiten por el canal. Este procedimiento se repite hasta llegar el último grupo de *k* bits que se codifican en una salida de *n* bits. Para una mayor sencillez a la hora de decodificar, después del último envío de *k* bits, entran al codificador un grupo de *k(m-1)* ceros, y sus correspondientes salidas son enviadas. Así, queda listo para otro envío futuro y se facilita por lo tanto el proceso.

A modo de ejemplo, se aplicará el codificador convolucional expuesto anteriormente para la siguiente secuencia de entrada:

1 1 0 0 1 0 1 1

Como se ha dicho anteriormente, el codificador parte del estado *todo cero*. Según los diagramas, si entra un 1, la salida codificada es 11 y el estado al que se pasa es 10. A este nuevo estado le llamaremos estado presente. En el ejemplo:

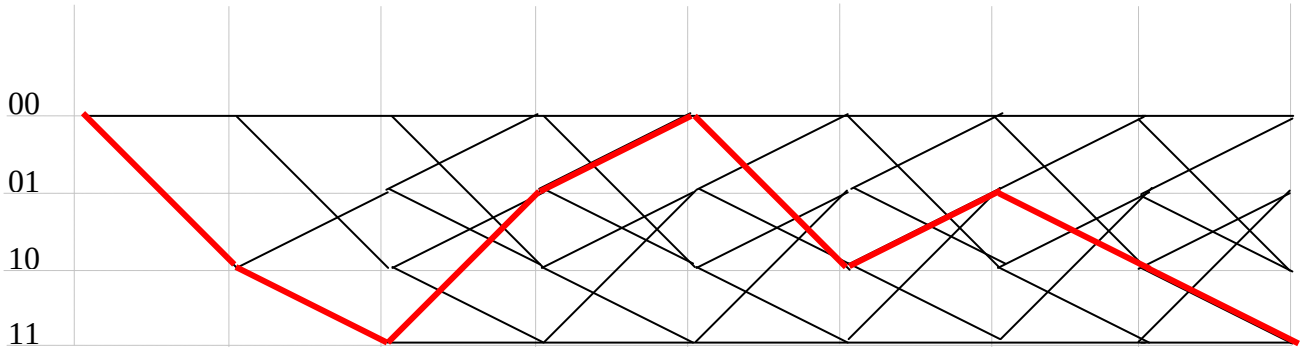
1º Dato entrada: 1	1º Dato codificado: 11	Estado presente: 10
2º Dato entrada: 1	2º Dato codificado: 10	Estado presente: 11
3º Dato entrada: 0	3º Dato codificado: 10	Estado presente: 01

Continuando el proceso llegamos a un código de estados presentes y a la secuencia codificada que entra al canal:

Estado presente: 10 11 01 00 10 01 10 11

Entrada codificada: 11 10 10 11 11 01 00 10

El camino de Trellis correspondiente a los datos correctos ya codificados:



DECODIFICACIÓN DE VITERBI

Para elegir la señal transmitida a partir de la recibida se toma la salida contaminada con ruido y se compara directamente con todos los caminos posibles obtenidos en el Diagrama de Trellis, hallar la distancia Hamming a cada uno de estos (número de bits en que se diferencian) y elegir el camino que menos distanciado esté. Al decir los posibles, nos referimos a que no todas las uniones que se pueden hacer en un Diagrama de Trellis son viables, ya que vienen determinados por las transiciones descritas en el diagrama de estados. Se graficará la decodificación mediante una simplificación del *Maximum Likelihood Decoder (ML),* el cual selecciona, por definición, el valor estimado de la salida del decodificador que maximiza la probabilidad de haber transmitido una entrada concreta. Esta simplificación del decodificador ML es el *Algoritmo de Decodificación de Viterbi*. El proceso consiste en desechar algunos de

todos los caminos posibles. Lo que se consigue aplicando este método es reducir el número de cálculos.

Según el algoritmo de Viterbi, para reducir el número de cálculos, cada vez que dos trayectos (también llamados ramas) se junten en un estado en el Diagrama de Trellis, el de mayor métrica acumulada, o el inferior si tienen la misma métrica, se desecha en la búsqueda del trayecto óptimo. Esto se debe a que el estado actual resume la historia de todos los estados anteriores en cuanto a su influencia en los estado posteriores. Esto se hace en los 2k-1 estados, se pasa al intervalo t_{i+1} y se repite el proceso. Realmente, esta eliminación de posibles caminos no comienza hasta el tercer nivel de representación. Esto se debe a que hasta ese instante, no han podido converger dos ramas en un estado ya que por cada *k* bits de palabra de datos (*k =1* en el ejemplo) surgen 2k estados. Es decir, del primer estado (el 00 por definición) tendremos dos estados en el siguiente nivel de representación, y por cada uno de estos, otros dos en el posterior nivel. Por tanto será en el tercer nivel cuando empecemos a tomar decisiones porque ahora sí convergerán dos ramas en cada estado.

EJEMPLO: Una vez conocida la secuencia de 16 bits correspondientes a la palabra codificada en el ejemplo anterior, se supone una secuencia recibida con una serie de errores producidos por el ruido en el canal (en azul los bits erróneos en la recepción). Dado que también que se conoce el Trellis que sigue la palabra enviada, se podrá observar gráficamente en el proceso de decodificación de Viterbi cómo se van obteniendo los caminos que conducen al resultado óptimo.

De esta manera:

Datos: 1 1 0 0 1 0 1 1

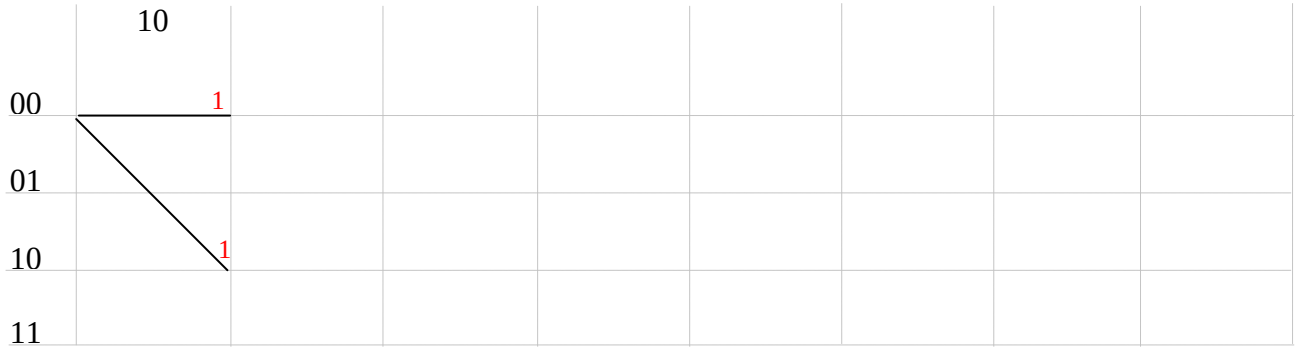
Estado presente: 10 11 01 00 10 01 10 11

Codificado: 11 10 10 11 11 01 00 10

Recibido: 10 10 11 11 01 01 00 10

Se representará gráficamente el desarrollo del proceso. Sobre cada línea de unión de estados se colocará en rojo el numero de errores acumulados en relación con la señal recibida (la distancia de Hamming entre lo recibido y lo que debería haber recibido. En el momento de hacer la elección se pondrá en azul el número de errores de los caminos desechados:

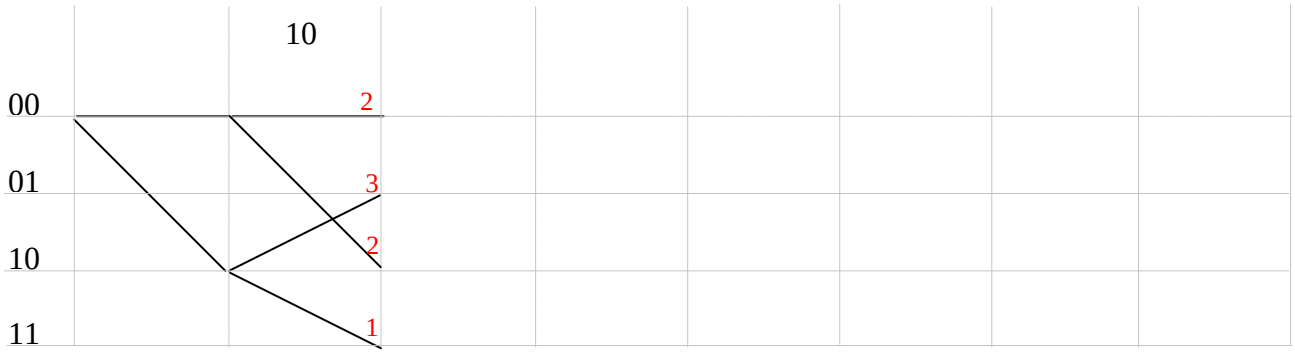
El primer y segundo nivel son fijos para cualquier entrada. Del estado 00 tenemos dos posibles ramas, hacia el estado 00 y hacia el 10.



Nivel 1

D (10,00) = 1

D (10,11) = 1

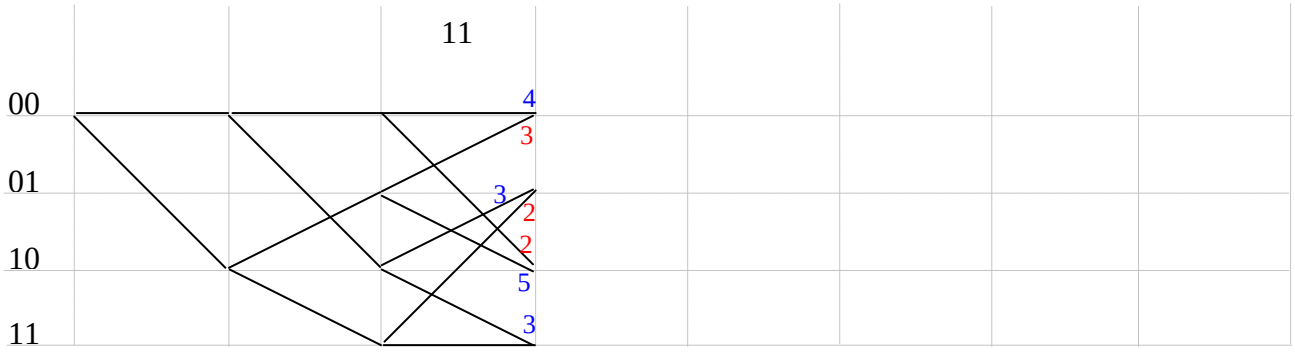


Nivel 2

$1 + D(10,00) = 1 + 1 = 2$ $1 + D(10,01) = 1 + 2 = 3$

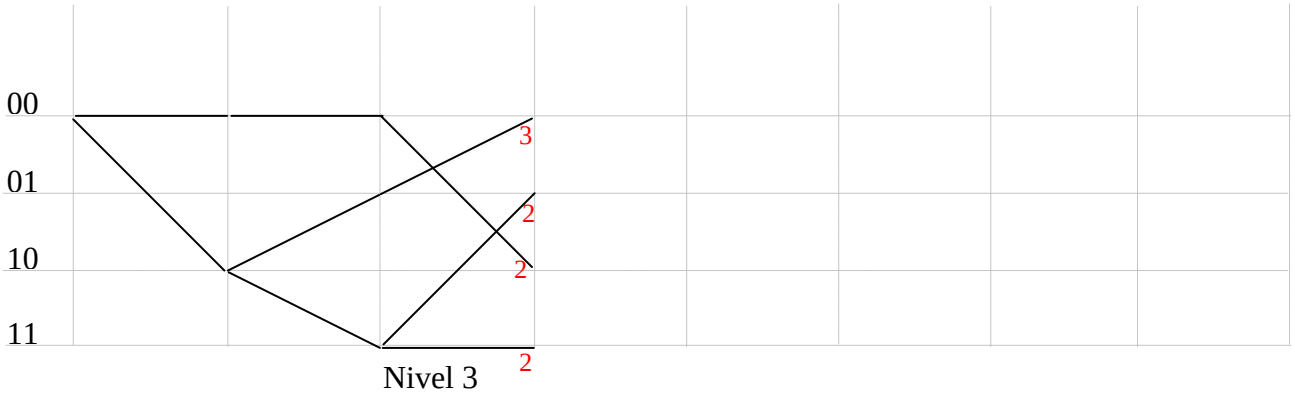
$1 + D(10,11) = 1 + 1 = 2$ $1 + D(10,10) = 1 + 0 = 1$

A partir del tercer nivel, en cada estado convergerán dos ramas.



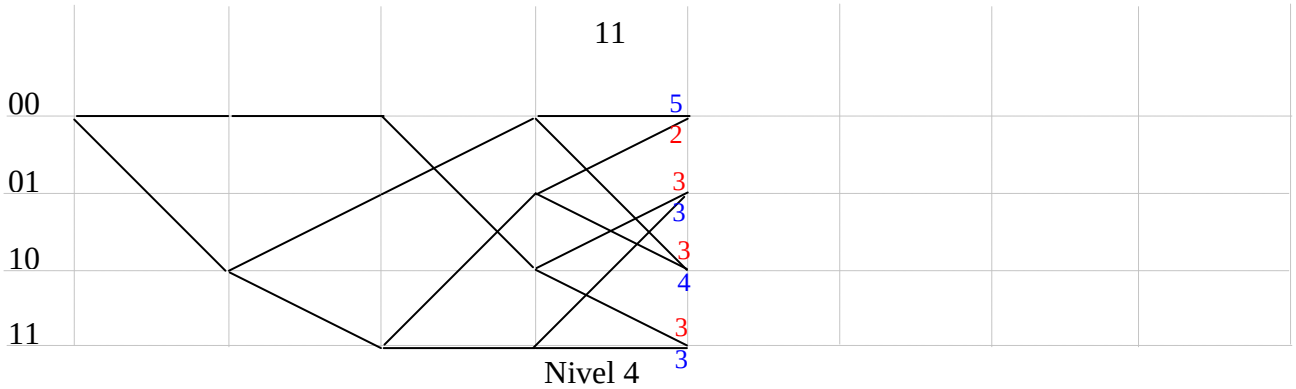
Nivel 3

Según el número de errores acumulados en relación con la señal recibida, se elige un camino u otro, quedando el de menor error acumulado. Así, se parte de menos ramas para el siguiente nivel del proceso:



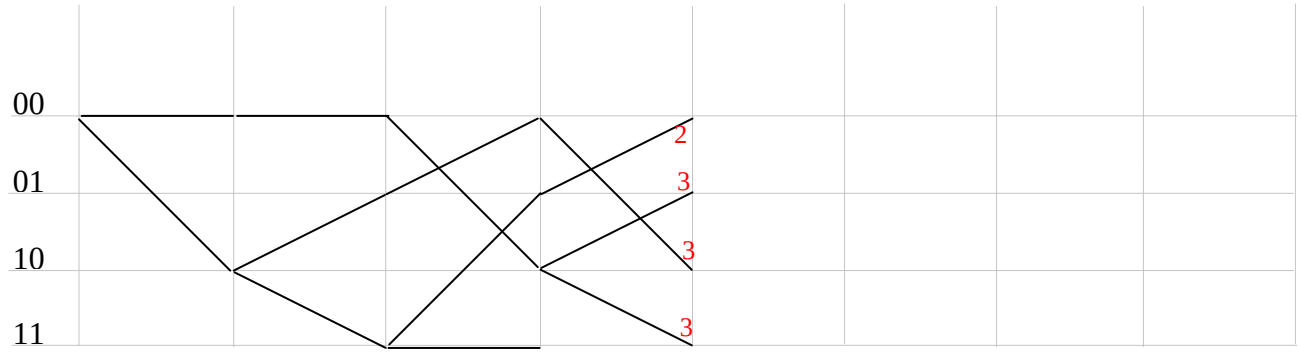
Nivel 3

De cada nodo parten de nuevo otras dos ramas.



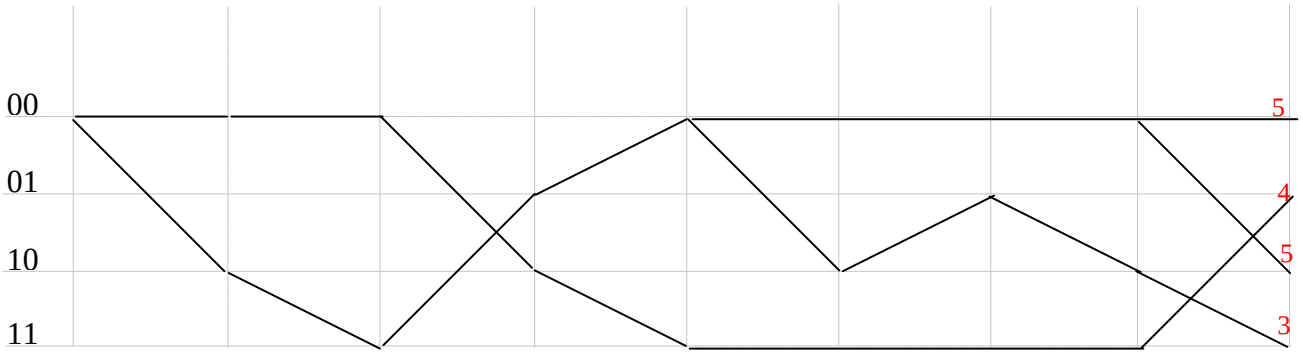
Nivel 4

Procedemos del mismo modo que para el tercer nivel y nos quedamos con las ramas de menor error acumulado:

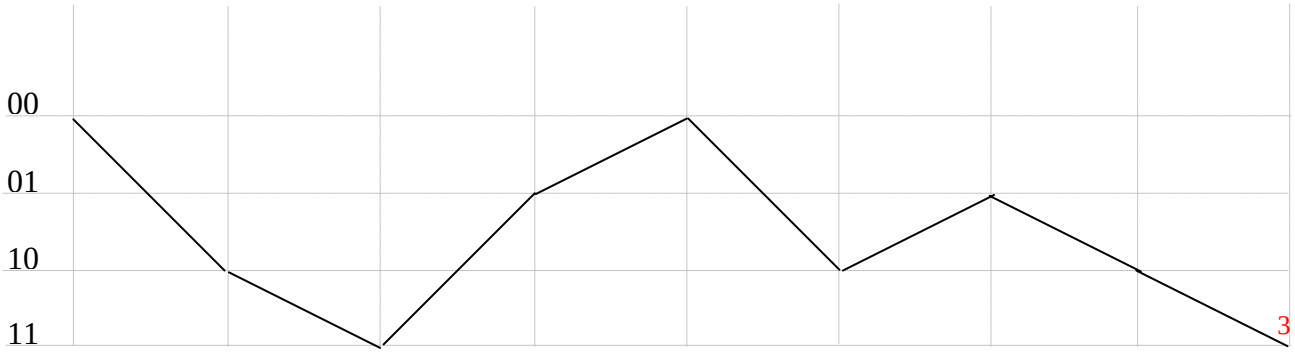


Nivel 4

Con esta metodología aplicada al resto de niveles, las cuatro ramas seleccionadas que llegarían a los cuatro últimos estados, dada su menor acumulación de error, serán:



Por lo tanto, el trayecto óptimo es aquel que finaliza en el estado 11, ya que tiene tres errores acumulados frente a los de los otros estados. Así pues, el trayecto recorrido óptimo es:



Observamos que coincide exactamente con la secuencia de datos enviada y codificada. Mediante el Algoritmo de Viterbi se reconstruyó perfectamente la secuencia de datos a partir de una secuencia codificada con errores introducidos por el ruido del canal.

5. CONCLUSIONES

Los códigos compactos, de longitud variable o de alto rendimiento son inequívocamente descifrables y no tienen redundancia, pero, como se dijo en el apartado donde fueron tratados estos códigos, presentan el inconveniente de que la distribución de la fuente puede no ser conocida cuando se diseña el código.

Los códigos de bloque suelen tener limitada capacidad de corrección de errores alrededor de 1 o 2 bits erróneos por palabra de código. Son códigos de bajo rendimiento debido a que tienen una gran redundancia (una palabra de código de Hamming tiene más bits de paridad que de información). Se podría pensar que si el número de bits de paridad es incrementado esto debería posibilitar la corrección de más y más errores. Sin embargo, agregando más y más bits de paridad, el ancho de banda aumenta igualmente. Debido al aumento de ancho de banda, más ruido es introducido y la probabilidad de error aumenta. Por lo tanto, estos códigos son buenos para utilizar en canales con baja probabilidad de error.

Los códigos de redundancia cíclica (CRC) son muy buenos para la detección de errores, son capaces de detectar muchas combinaciones. Su implementación práctica es sencilla, son muy usados, pero también se deben agregar bits de redundancia aumentando el número de bits del mensaje que se desea transmitir.

En los códigos convolucionales no se consideran los bloques individuales de bits como palabras de código. Un flujo continuo de bits de información se opera en la forma de mensaje codificado. La fuente genera

una secuencia continua de 1’s y 0’s. Los bits de mensaje son combinados de una manera particular con el fin de generar cada bit transmitido, y ésta es la secuencia transmitida, logrando ganancia de codificación sin expansión en el ancho de banda. Los códigos convolucionales son adecuados para usar sobre canales con mucho ruido (alta probabilidad de error) ya que el proceso de codificación convolucional y decodificación de Viterbi es válido para reconstruir con una probabilidad de error mínima una secuencia de bits transmitida con errores, y prácticamente, sin redundancia, lo que eleva el rendimiento del código.

6. BIBLIOGRAFÍA

- Castro Lechtaler, Antonio Ricardo y Fusario, Rubén Jorge: "Teleinformática para Ingenieros en Sistemas de Información" Editorial Reverté; 1999, Segunda edición.

- Schlegel, Christian: TRELLIS CODING
  IEEE PRESS, 1997

- Abramson, Norman: "Teoría de la Información y Codificación". Editorial Paraninfo, Madrid, 1986.

- Viterbi , A. J. y Omura, J. K.: "Principles of Comunications and Coding".
  Editorial Mc Graw Hill, Nueva York, 1990.

- "50 Years Information Theory".
  Editorial Verdu – McLaughlim, IEEE, 1999.

- http://www.isa.cie.uva.es

- http://www.tecnun.es

7. FUTURAS LÍNEAS DE INVESTIGACIÓN

- "Códigos convolucionales: Estrategias de decodificación"

En el presente trabajo fueron analizadas las distintas estrategias de codificación / decodificación para varios códigos, llegando a la conclusión que el código convolucional es uno de los más eficientes. En una próxima investigación, junto al equipo de investigación liderado por el Ingeniero Oscar Segre dentro del proyecto "Teoría de la codificación, simulación e implementación de códigos de alto rendimiento para se aplicados en redes de datos utilizando TCM", se analizarán otras estrategias de decodificación, además del algoritmo de Viterbi, para códigos convolucionales.

- "Transformación de los códigos de bloque en códigos de Trellis"

Análisis de la transformación de códigos de bloque en códigos de Trellis verificando la complejidad de los decodificadores a utilizar.

Tailbiting trellis son representaciones de códigos de bloque recientemente propuestos. Se propone comprobar si estos "nuevos" códigos permiten una más eficiente decodificación.

[1] Norman Abramson, Teoría de la Información y Codificación, Paraninfo, lo llama código de bloque. Otros autores definen los códigos de bloque como aquellos en que todas las palabras contienen un mismo número de símbolos, definición usada ampliamente.