Codificación de canal II: códigos convolucionales

Francesc Tarrés
Margarita Cabrera

PID_00185033

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Este módulo debe considerarse como una extensión y continuación del módulo anterior. Tal y como hemos visto, la codificación de canal consiste en definir con precisión el conjunto de símbolos con los que se enviarán los mensajes entre el emisor y el receptor. Esta definición tan genérica admite que el conjunto de reglas sistemáticas para determinar la secuencia de bits de salida en función de la secuencia de entrada pueda elegirse con el objetivo de solventar problemas de distinta naturaleza, teniendo una gran flexibilidad en función de las aplicaciones y de las posibilidades de proceso de datos que se disponga.

Al diseñar estrategias para proteger la información frente a eventuales errores, no tenemos demasiado margen de maniobra. En efecto, la única solución posible es que el receptor pueda distinguir entre un mensaje producido por el emisor y un mensaje en el que se han producido errores. Para ello, es necesario que introduzcamos elementos adicionales en la información que nos proporcionen pistas sobre la integridad de los mensajes. Esta introducción de información adicional recibe el nombre de *redundancia estructurada* y significa que el número de bits de información que finalmente se transmiten al canal sea superior al mínimo necesario. Así pues, la inserción de códigos de protección frente a errores supone la introducción de bits adicionales en el transmisor que representa un aumento del ancho de banda de la señal que transmitimos. El precio que debe pagarse para proteger el mensaje es, por tanto, aumentar el ancho de banda de la información.

El módulo empieza con una descripción de la estructura general de los códigos convolucionales. Este tipo de códigos son muy eficientes y representan una alternativa a los códigos de bloque utilizada en muchas aplicaciones. En la práctica, el interés de los códigos convolucionales se centra en que muchos sistemas de comunicaciones utilizan de forma simultánea códigos de bloque y códigos convolucionales. Veremos que, cuando estos dos tipos de códigos se combinan en la forma adecuada (concatenación de códigos), se pueden conseguir propiedades de protección que no serían posibles con el uso único de los códigos individuales. La concatenación de estos dos tipos de códigos se utiliza en varios sistemas de comunicaciones como la televisión digital terrestre (DVB-T), los sistemas WiFi, los sistemas de telefonía móvil, etc.

En la primera parte del módulo se analizan distintas formas de representar y analizar el funcionamiento de los códigos convolucionales, centrándose principalmente en los diagramas de Trellis. Una vez estudiado el proceso de codificación, se expone con detalle el proceso de descodificación de una secuencia, utilizando el conocido algoritmo de Viterbi, sobre el que trata la mayor parte

del contenido de este módulo. El algoritmo de Viterbi nos permite determinar cuáles son los bits correctos e incorrectos de la secuencia transmitida. El módulo continúa con una presentación general de los conceptos de perforación de códigos, concatenación de códigos y ejemplos de algunos sistemas de comunicaciones que utilizan códigos convolucionales. Finalmente, se presentan las características generales de los denominados turbo códigos. Estos códigos son bastante complejos desde un punto de vista matemático y sus detalles van más allá de los objetivos generales de este texto.

Se proporcionan algunos ejemplos y problemas resueltos a lo largo del texto y al final del módulo. También se incluyen algunos ejercicios para ser resueltos de forma autónoma por el alumno. Todos estos ejercicios y ejemplos constituyen los problemas más típicos que pueden realizarse sobre códigos convolucionales.

Objetivos

Los objetivos que debe alcanzar el estudiante con este módulo didáctico son los siguientes:

1. Comprender la necesidad de proteger la información frente a errores del canal.

2. Diferenciar entre los códigos de bloque y los códigos convolucionales.

3. Comprender los elementos matemáticos básicos para realizar códigos convolucionales.

4. Dominio del algoritmo de descodificación de Viterbi.

5. Comprender las necesidades y características de la concatenación de códigos.

6. Comprender los mecanismos de protección frente a errores utilizados en algunos sistemas de comunicación reales (tomando como ejemplo la DVB).

1. Códigos convolucionales

La característica principal de los códigos convolucionales es que el proceso de codificación tiene memoria, por lo que los bits en la salida del codificador dependen no sólo de la información actual sino también de los bits anteriores. La memoria de un código convolucional nos indica en qué grado los bits codificados dependen de los valores que ha tomado la información original en el pasado. En la figura 1 se muestra un diagrama de bloques muy sencillo correspondiente a un código convolucional que podemos utilizar como ejemplo para mostrar las características esenciales de estos códigos.

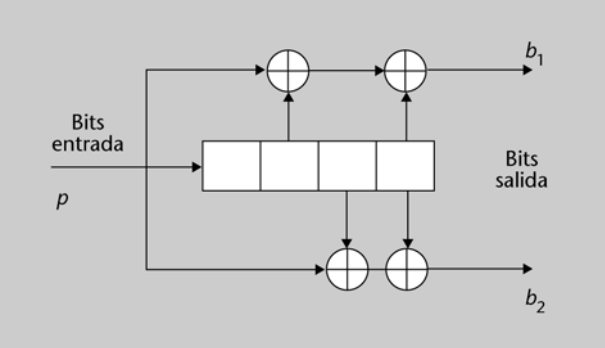

Figura 1. Diagrama de bloques de un ejemplo de código convolucional.

En este ejemplo se muestra un conjunto de registros de desplazamiento (en total 4 unidades de memoria) que almacenan los valores anteriores de la secuencia de entrada. Los bits del código se determinan teniendo en cuenta el bit actual y los bits retardados, realizando las operaciones de sumas exclusivas indicadas en el diagrama. Por cada bit de la secuencia de entrada se obtienen 2 bits en la secuencia de salida que deberán alternarse en dicha salida. Así pues, se trata de un código con una tasa $R = k/n = 1/2$. En este ejemplo, los bits de salida dependen del valor del bit actual de entrada y del valor de los 4 bits anteriores. Esta situación puede interpretarse como si los 4 bits anteriores definieran un estado del codificador y que la salida depende de este estado y del valor del bit actual. Además, cada vez que se recibe un nuevo bit, el estado del codificador varía. En nuestro ejemplo, como el número de bits que define los posibles estados del codificador es 4, el número total de estados del codificador es de 2^4. Finalmente, comentar que las características del código vienen determinadas por las operaciones de suma exclusiva y los bits retardados que se utilizan. Estas conexiones se indican mediante un polinomio generador. Así, el polinomio generador asociado a la rama inferior (b_2) se denota como $G_2 = X^4 + X^3 + 1$, indicando que se realiza la suma exclusiva entre los bits de las posiciones 4 y 3 (muestras retardadas) del registro de desplazamiento con el bit de la entrada. Los polinomios generadores pueden encontrarse en tablas y se proporcionan con notación octal. En nuestro ejemplo el polinomio expresado en notación octal sería $31_{OCT} = (11\ 001)$.*

*** La notación octal utiliza los dígitos del 0 al 7. Para pasar de esta notación a binario convencional, basta con sustituir cada dígito por su código binario de 3 bits. Así, por ejemplo, el número octal 541 será 101 100 001.**

A modo de resumen, podemos identificar las características del codificador convolucional de la figura 1 de la siguiente manera:

- Número de líneas de entrada: $k = 1$
- Número de líneas de salida: $n = 2$
- Tasa del código: $R = 1/2$
- Unidades de memoria: 4
- Número de posibles estados: $2^4 = 16$
- Longitud total de la convolución: 5
- Polinomio generador 1: $G_1 = X^4 + X^2 + 1 = 25_{OCT}$
- Polinomio generador 2: $G_2 = X^4 + X^3 + 1 = 31_{OCT}$

1.1. Diagrama de bloques genérico de un codificador convolucional

En la figura 3 se muestra el diagrama de bloques genérico de un codificador convolucional con una tasa de codigo $R = k/n$. En general, puede suponerse que existe un total de L etapas, cada una de ellas con k bits. Para obtener el código correspondiente a una determinada secuencia de bits, debemos suponer que en la entrada los datos se estructuran en paquetes de k bits, y que posteriormente se calculan los n bits correspondientes a la salida. En el ejemplo de la figura 3 el número total de etapas de memoria es $L \cdot k$.

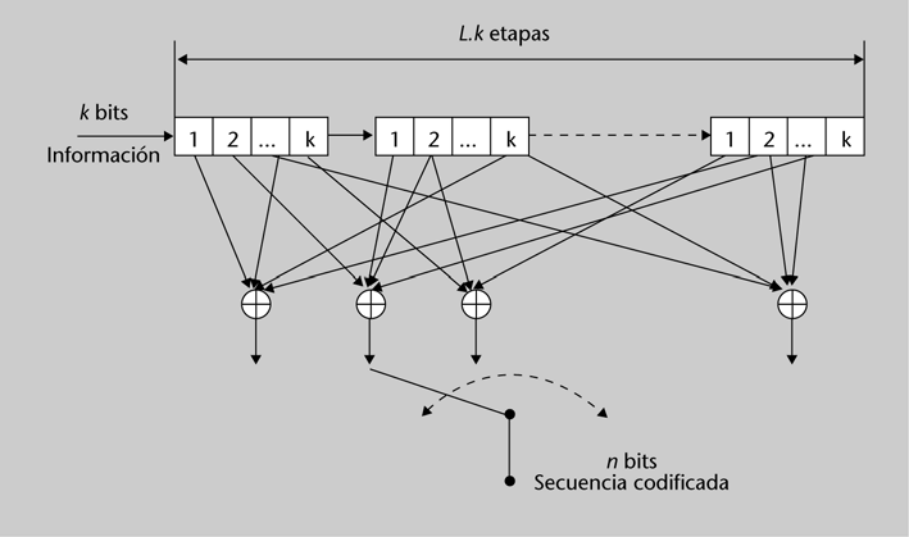

Figura 3. Diagrama genérico de un codificador convolucional.

1.2. Diagramas de estado y diagramas de Trellis

Un codificador convolucional tiene una memoria finita, por lo que es posible representar todos sus posibles estados y transiciones mediante un grafo denominado diagrama de estado. Para ver cómo se realiza la representación de un codificador mediante un diagrama de estado, consideraremos un ejemplo muy simple representado en la figura 4. En esta figura, la entrada se ha repre-

sentado a la derecha y la salida, a la izquierda para que los estados concuerden con el contenido de los registros de desplazamiento.

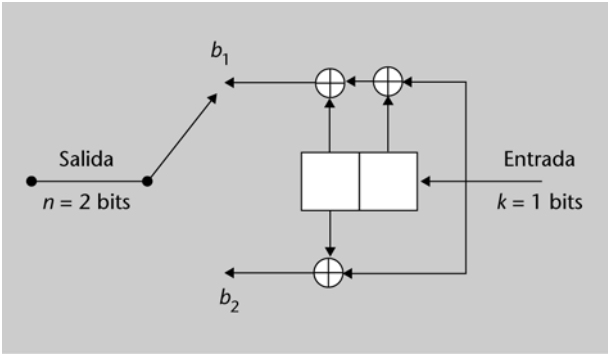

Figura 4. Codificador convolucional simple para ilustrar los diagramas de estado y de Trellis.

El codificador viene caracterizado por los parámetros siguientes:

- Número de líneas de entrada: $k = 1$
- Número de líneas de salida: $n = 2$
- Tasa del código: $R = 1/2$
- Unidades de memoria: 2
- Número de posibles estados: $2^2 = 4$
- Longitud total de la convolución: 3
- Polinomio generador 1: $G_1 = X^2 + X + 1 = 7_{OCT}$
- Polinomio generador 2: $G_2 = X^2 + 1 = 5_{OCT}$

Los posibles estados en los que puede estar el sistema se corresponden con los valores que pueden tomar los dos registros internos, es decir, {00, 01, 10 y 11}. En el diagrama de estado, los estados se indican con un círculo en cuyo interior aparecen los valores de los registros. Cada vez que entra un nuevo bit, el estado interno de los registros se modifica, por lo que es posible que se produzca una transición de estados. Las transiciones entre estados se indican con flechas que se rotulan con 3 bits en la forma X/XX. El primer bit indica el valor de la entrada que produce la transición y los dos bits después de la barra indican los dos valores que tomará la salida. En el caso de que el número de líneas de entrada y salida sea distinto, las flechas se rotulan con el número de bits apropiado. De cada estado pueden salir dos flechas dirigidas a otros estados (o a él mismo) y a cada uno llegan también dos flechas.

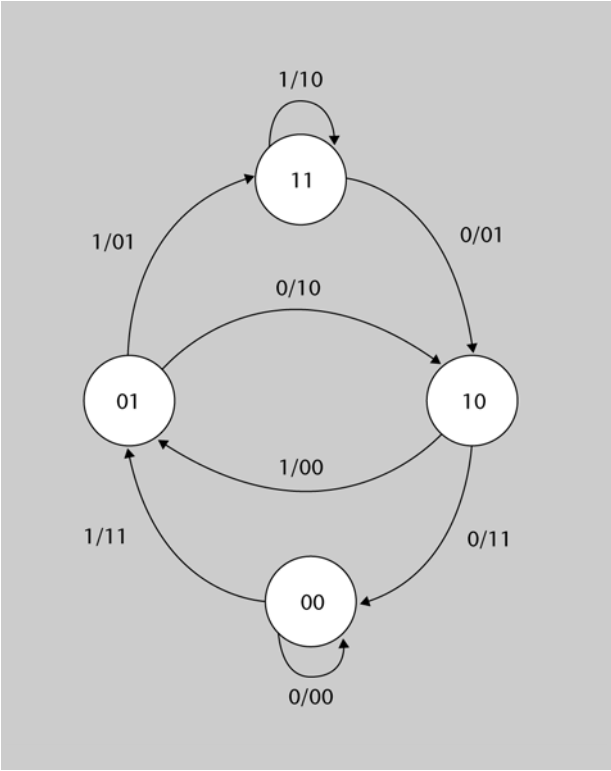

Figura 5. Diagrama de estado correspondiente al codificador convolucional de la figura 4.

Actividad

Comprobad que el diagrama de estado representado en la figura 5 concuerda con el codificador convolucional representado en la figura 4.

El diagrama de Trellis es una representación alternativa que también nos indica los diferentes estados del codificador y sus posibles transiciones en función de la información de entrada. La diferencia más importante es que en esta representación se muestra la evolución de las transiciones en el tiempo. De hecho, el diagrama de Trellis es una secuencia en el eje temporal de las posibles transiciones que pueden producirse en el codificador a medida que va entrando nueva información. En el eje vertical se representan todos los posibles estados del codificador. Las transiciones se representan mediante flechas. En el caso concreto en que sólo exista una línea de entrada, se suele utilizar una línea continua para representar la entrada de un cero y una línea discontinua para un uno.

En la figura 6 se representa el diagrama de Trellis asociado al codificador de la figura 4. Obsérvese que suponemos que el estado inicial del codificador es el {00}, por lo que los posibles estados que puede tomar el sistema evolucionan a lo largo del tiempo hasta llegar a una misma representación gráfica que se repite en cada transición (cuando todos los estados han podido ser alcanzados). En cada flecha se indican los valores que toman las líneas de salida (en nuestro caso las dos líneas). Si el número de líneas de entrada fuera mayor, es conveniente rotular las flechas utilizando la misma notación que en el diagrama de estado.

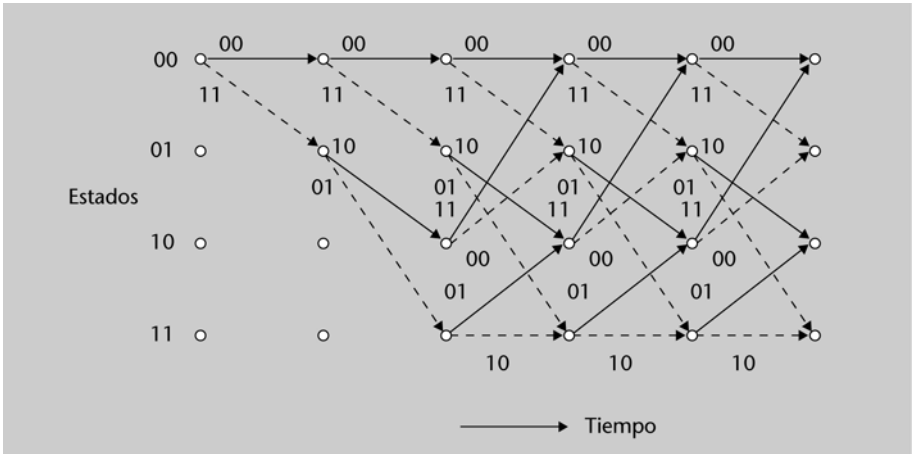

Figura 6. Diagrama de Trellis del codificador convolucional de la figura 4.

Actividad

Comprobad que el diagrama de Trellis de la figura 6 se corresponde con el codificador de la figura 4.

1.3. Proceso de codificación de una secuencia

La sistemática para codificar una secuencia de información mediante un código convolucional es muy simple. Puede realizarse directamente a partir del diagrama de bloques del codificador, viendo cómo evolucionan los registros internos a medida que se introduce la información y realizando las operaciones de sumas exclusivas indicadas. No obstante, resulta mucho más simple efectuar el cálculo de una forma más sistemática utilizando los diagramas de estado o los diagramas de Trellis. El diagrama de Trellis proporciona una valiosa información de cómo han ido evolucionando los estados del codificador en el tiempo.

En general, para determinar la secuencia codificada debemos introducir un bloque de k bits de la información original (suponemos que el codificador tiene k líneas), y en función de estos k bits determinar los n bits que obtendremos en la salida y el nuevo estado al que accederemos. Habitualmente, se supone que el codificador en su estado inicial tiene todos los registros a cero. Además, cuando determinamos la salida a una determinada secuencia de entrada, siempre añadimos suficientes ceros a la información de entrada para que el codificador regrese al estado inicial cero al finalizar la codificación.

Consideremos un ejemplo sencillo con el codificador que estamos utilizando en este apartado (figura 4). Supongamos que la secuencia que queremos codificar es: {0, 1, 1, 1, 0, 1, 1}. En nuestro caso, como sólo tenemos una línea de entrada, los bits entrarán al codificador de uno en uno. Debido a que los dos últimos bits son 1, el estado final en el que quedará el codificador es el {11}. Deberemos pues extender la secuencia de entrada con ceros para que el estado final sea el {00}. De esta forma, la secuencia de entrada definitiva será: {0, 1, 1, 1, 0, 1, 1, 0, 0}. Veamos cómo se realiza la codificación de la secuencia con ayuda de la siguiente tabla en la que utilizamos la información del diagrama de estados de la figura 5.

Estado inicial	Entrada	Salida	Estado final
00	0	00	00
00	1	11	01
01	1	01	11
11	1	10	11
11	0	01	10
10	1	00	01
01	1	01	11
11	0	01	10
10	0	11	00

Tabla 1. Sucesivos estados del codificador para la secuencia de entrada del ejemplo.

La forma de construir la tabla es muy simple. Inicialmente estamos en el estado 00 y recibimos un cero, por lo que permaneceremos en el mismo estado obteniendo una salida de 00. Cuando la siguiente entrada vale la unidad, pasamos al estado 01 y, en esta transición, la salida del sistema es 11. De esta forma vamos completando la tabla obteniendo la siguiente salida del codificador: {00 11 01 10 01 00 01 01 11}.

El diagrama de Trellis nos proporciona una visión más gráfica de cómo se va realizando la codificación de la secuencia. En la figura adjunta se han representado en trazo grueso las transiciones de la tabla anterior sobre el diagrama de Trellis. Las transiciones que se corresponden con un cero en la entrada se indican con trazo continuo. De modo análogo, los unos se indican usando trazo discontinuo.

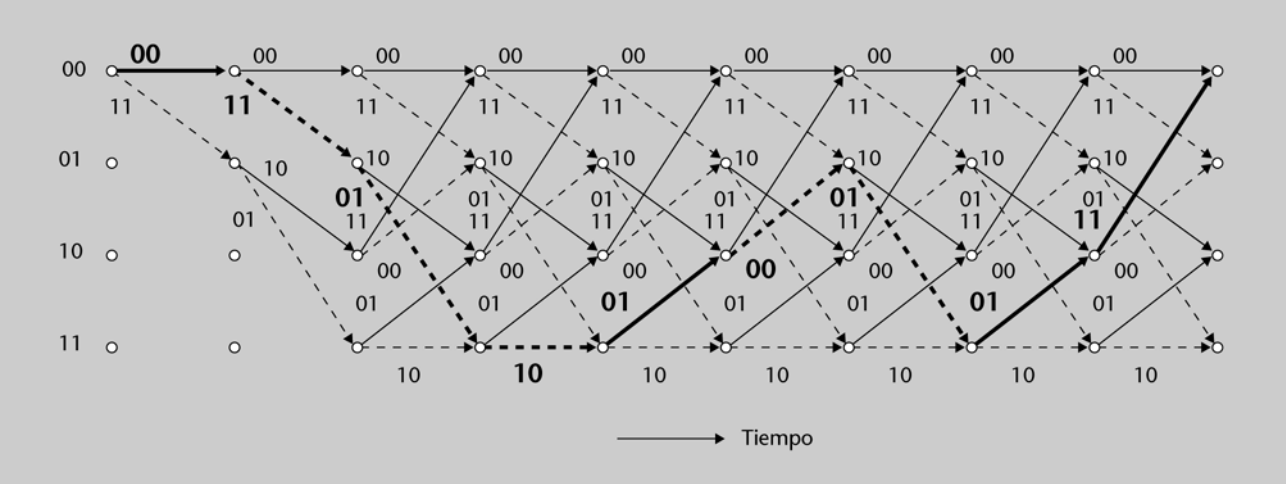

Figura 7. Evolución en el diagrama de Trellis al codificar un mensaje.

Con el diagrama de Trellis es fácil seguir cómo van evolucionando los estados en el codificador a medida que aparecen nuevos datos. Con una simple inspección del diagrama, tenemos información de cuál ha sido la secuencia de entrada (0 si la línea es continua o 1 si es discontinua), de cuál es la secuencia de estados internos del codificador (el nivel en la posición vertical) y de los bits codificados en la salida del sistema (pares de bits en negrita).

1.4. Descodificación de una secuencia. Algoritmo de Viterbi

El proceso de descodificación de una secuencia es algo más complejo que el de codificación debido a que si se producen errores en la secuencia no resulta trivial recuperar cuál es la información original. Para realizar la descodificación suele utilizarse el algoritmo de Viterbi, un algoritmo muy conocido que tiene aplicación en diversas áreas del procesado de señal como, por ejemplo, en los esquemas de demodulación PCM, en la estimación de máxima verosimilitud (ML) de secuencias transmitidas en canales con interferencia intersimbólica, en algunos métodos para el reconocimiento de voz y en diversos esquemas para la clasificación de patrones. En todos estos problemas se trata de buscar un camino óptimo (o el camino más probable) dentro de un diagrama de Trellis.

El algoritmo de Viterbi tiene un enunciado formal bastante complejo y los detalles de las demostraciones escapan a los objetivos de este texto. Por ello presentaremos el algoritmo mediante un ejemplo y posteriormente intentaremos generalizar los resultados de forma que quede clara la esencia del algoritmo y el procedimiento que debe seguirse para descodificar una secuencia mediante el mismo.

Para empezar con el ejemplo supondremos que se ha transmitido la misma secuencia que hemos codificado en el ejemplo del apartado anterior y que se han producido 2 errores en las posiciones que se muestran en la figura 8.

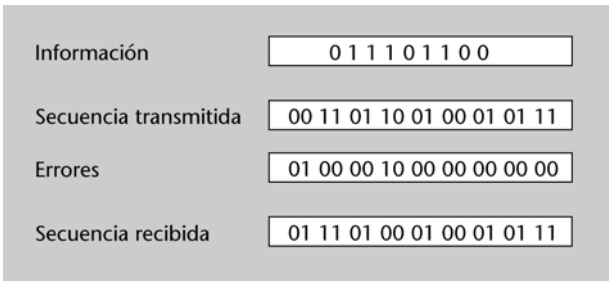

Figura 8. Secuencia del ejemplo recibida después de haberse producido dos errores en el canal.

Para empezar con el algoritmo para la descodificación de la secuencia, los bits recibidos se agrupan en paquetes de n bits ($n = 2$ en nuestro ejemplo). Para proceder con la descodificación dibujaremos un diagrama de Trellis con una profundidad (número de etapas) igual al número de paquetes de n bits de la secuencia recibida (en nuestro caso tenemos 9 paquetes de 2 bits).

Una clave importante para la descodificación es que conocemos el estado inicial del codificador (siempre empezamos la codificación de una secuencia con todos los registros con valor inicial cero) y también su estado final (ya que hemos añadido ceros a la secuencia de entrada hasta garantizar que dejábamos el codificador en su mismo estado inicial). Por ello, sabemos que el camino que ha seguido el codificador en el diagrama de Trellis empieza y finaliza en el estado {00}, sólo nos queda conocer cuáles han sido las transiciones intermedias. De hecho, tenemos alguna pista adicional. En efecto, si el estado final es {00}, significa que los dos últimos bits recibidos han sido cero.

Para empezar con el proceso de descodificación, vemos que desde el estado inicial {00} podemos llegar a los estados {00} o {01}. En el primer caso la salida del sistema hubiera sido 00 y en el segundo hubiera sido 11. Así pues, tanto en un caso como en otro, lo que hemos recibido no coincide con lo que deberíamos haber recibido. En los dos casos tenemos un bit correcto y un bit incorrecto. Para seguir con el procedimiento de descodificación anotaremos el valor 1 (número de bits coincidentes) en los dos nodos de llegada. En la segunda etapa, si partimos del estado {00} podemos llegar al {00} o al {01} y si partimos del {01} podemos ir al {10} o al {11}, con lo cual ya podemos haber alcanzado cualquiera de los 4 estados del codificador. Ahora procedemos a rotular cada estado con el número de coincidencias entre la secuencia recibida y la secuencia que hubiéramos obtenido para llegar a ese estado. Todas estas etapas de la fase de descodificación se muestran en la figura 9. Obsérvese, por ejemplo, que en la etapa 2, el estado {01} está rotulado con el número 3, ya que se producen tres coincidencias entre la secuencia recibida hasta este momento {01 11} y la secuencia que hubiéramos recibido para llegar a este estado {00 11}. Análogamente, pueden calcularse los valores y las coincidencias de los otros estados.

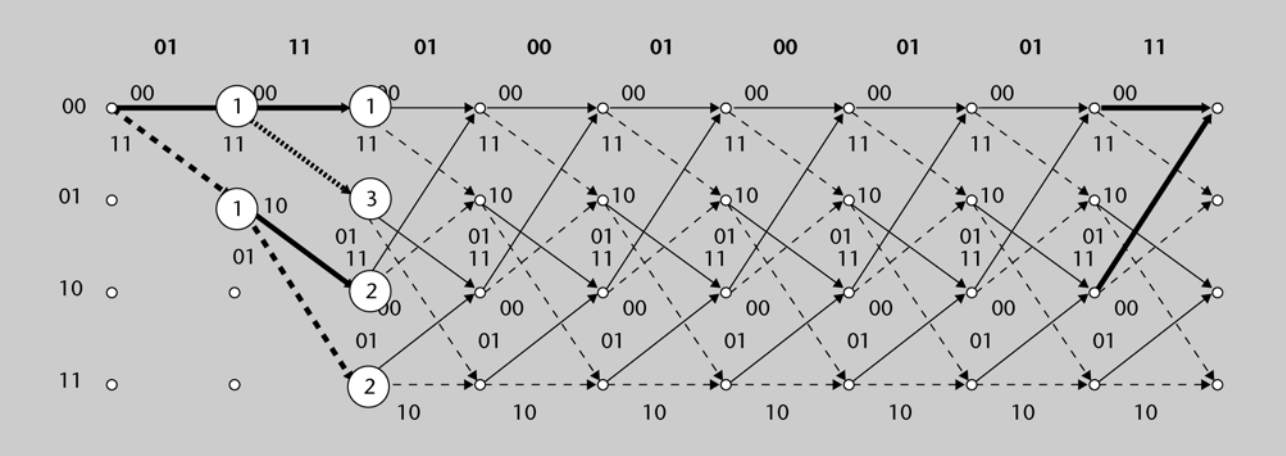

Figura 9. Primeras etapas del proceso de descodificación de una secuencia.

En este punto, cuando ya hemos alcanzado todos los posibles estados, es donde empieza el algoritmo de Viterbi propiamente dicho. En la etapa siguiente, cada estado puede ser alcanzado desde 2^k estados distintos (en nuestro caso 2). El algoritmo de Viterbi propone que se calculen el número de coincidencias que se producen desde cada una de las posibles transiciones y que se tome el camino que produce el mayor número de coincidencias, descartando los demás.

Veamos qué ocurre en nuestro ejemplo. Para alcanzar el estado {00}, en la tercera etapa podemos proceder del estado {00} o del estado {10} en la etapa previa. Si llegamos desde el estado {00}, la salida actual será 00 por lo que tiene una sola coincidencia con la salida de la tercera etapa 01. El número total de coincidencias asignadas a esta rama es por tanto de dos (las acumuladas en la etapa anterior, que aparecen en el rótulo, más la actual). En cambio, si llegamos al estado {00} desde el {10}, ya tenemos 2 coincidencias acumuladas. Ade-

más, ahora la salida es 11, por lo que también se produce una coincidencia con la secuencia recibida, de manera que el número total de coincidencias es tres. Como resultado final nos quedamos con la rama procedente del estado {10} y rotulamos el nuevo estado con el número de coincidencias total obtenido desde esta rama, es decir, tres. Realizamos esta misma operación con el resto de estados de la tercera etapa con la misma idea en mente: de los dos posibles caminos de llegada a un estado, debemos sólo considerar aquél que presente un mayor número de coincidencias y descartar el otro. El resultado final después de realizar todas estas operaciones para la etapa tres se muestra en la figura 10.

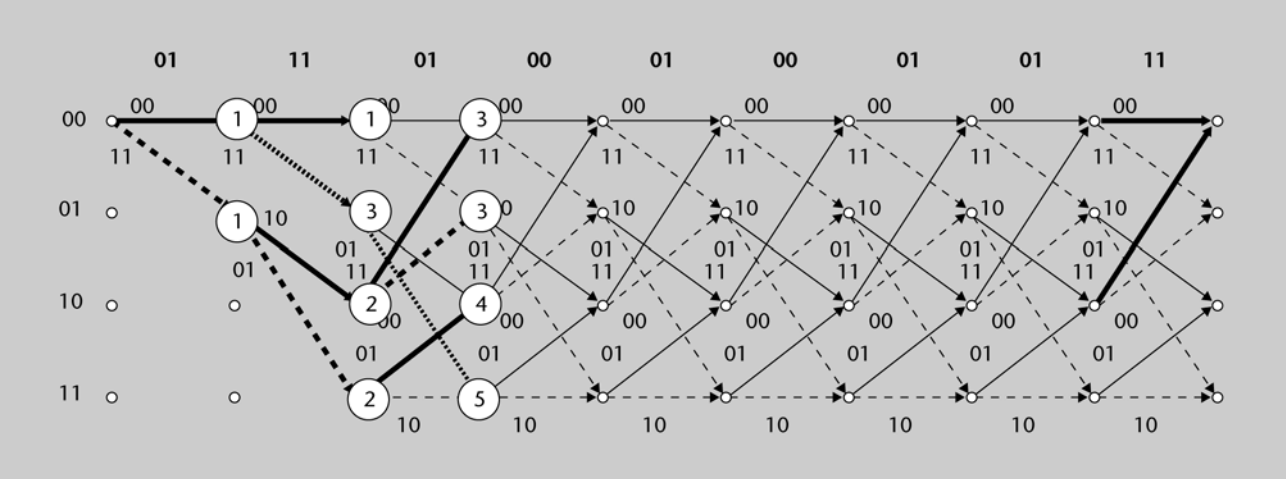

Figura 10. Proceso de selección de los caminos óptimos en la tercera etapa.

Ahora podemos eliminar todas las ramas (posibles caminos) que han sido descartadas y aquellas de las etapas anteriores que también quedan en vía muerta. Después de realizar este proceso, obtenemos el diagrama representado en la figura 11.

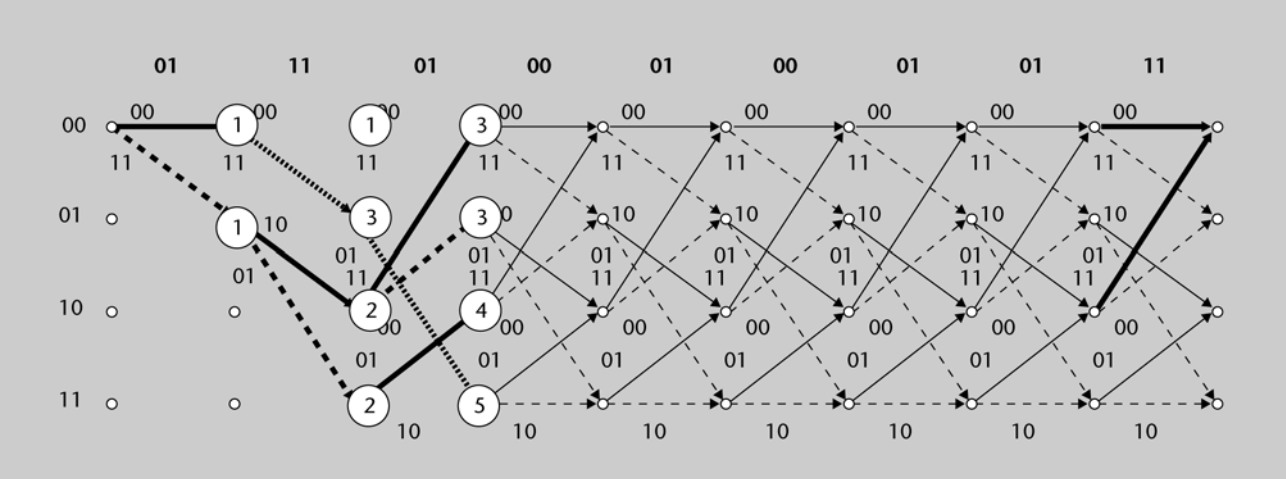

Figura 11. Eliminamos las ramas que no conducen a ningún nodo de destino de la tercera etapa.

La idea básica del esquema de Viterbi es que, para un estado en la etapa n al que se puede llegar desde dos (en un caso genérico serían 2^k) estados de la etapa anterior (n – 1), debemos considerar únicamente el camino que presenta la

métrica más alta, es decir, el mayor número de coincidencias. La demostración de que este procedimiento es óptimo no es evidente, pero pueden encontrarse los detalles en un artículo clásico (Viterbi, 1967) y en algunos textos avanzados de comunicaciones.

Podéis observar que las métricas o coincidencias calculadas en la etapa $n - 1$ pueden utilizarse para calcular las coincidencias de la etapa actual. Únicamente debemos acumular las coincidencias de la rama que estamos evaluando con las coincidencias que teníamos en el nodo del que partimos. Así pues, para cada nodo de la etapa n evaluaremos su coste (número de coincidencias) partiendo de cada uno de los nodos de la etapa anterior que llegan a él, quedándonos con la rama que presenta mayores coincidencias y eliminando el resto de ramas. Podría ocurrir que el valor máximo se alcanzara para dos ramas distintas, ya que el número de coincidencias por los dos caminos es el mismo. En este caso, el algoritmo de Viterbi permite elegir arbitrariamente cualquiera de las dos ramas. En efecto, es posible demostrar que no existe forma de solventar este tipo de ambigüedades, por lo que las dos soluciones son equivalentes (igual de malas). En las figuras 12, 13, 14 y 15 se muestran las etapas sucesivas que van obteniéndose al aplicar el algoritmo en las etapas 4, 5, 6 y 7 respectivamente.

Actividad

Se propone como ejercicio que se verifiquen los detalles de cómo se van obteniendo los estados en las diferentes etapas.

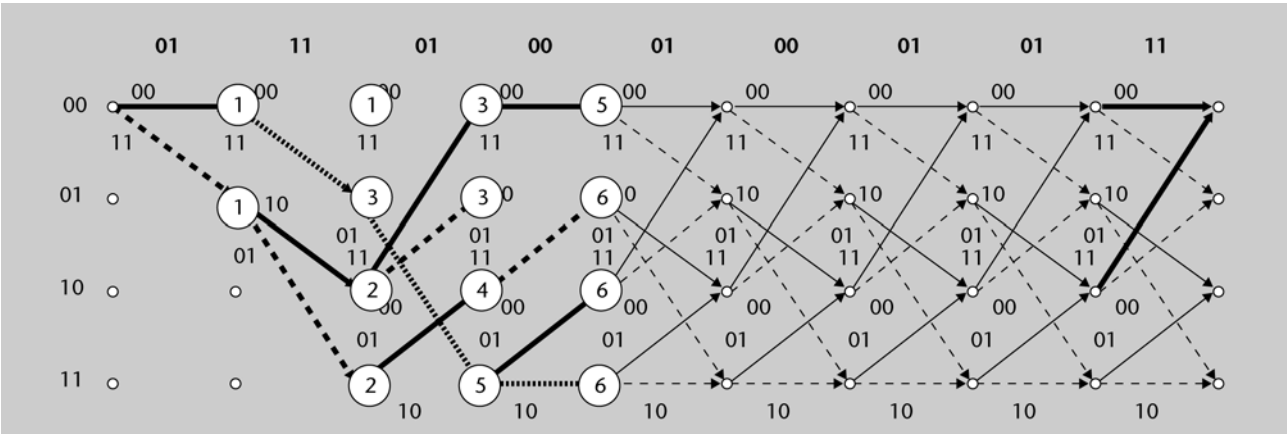

Figura 12. Estado de la descodificación al alcanzar la cuarta etapa.

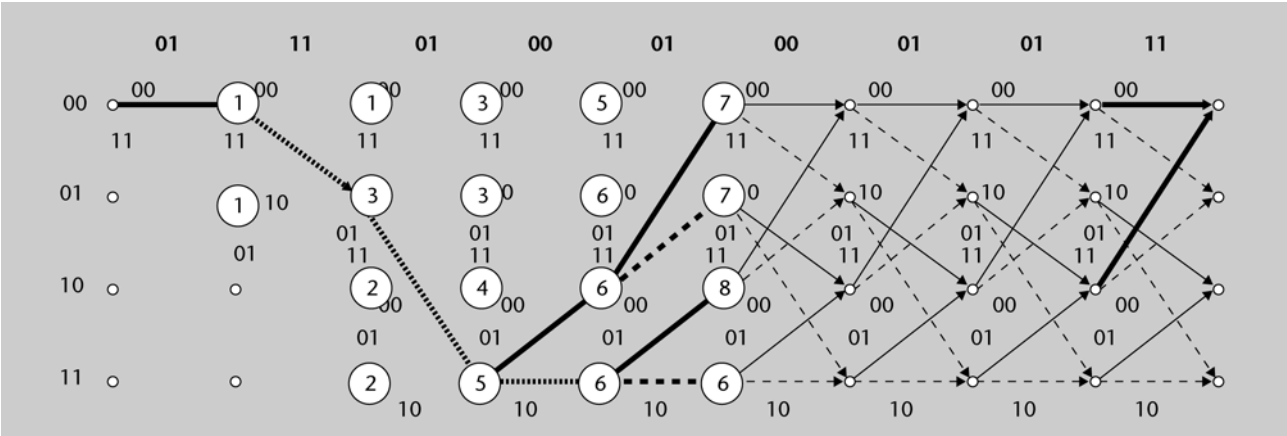

Figura 13. Estado de la descodificación al alcanzar la quinta etapa.

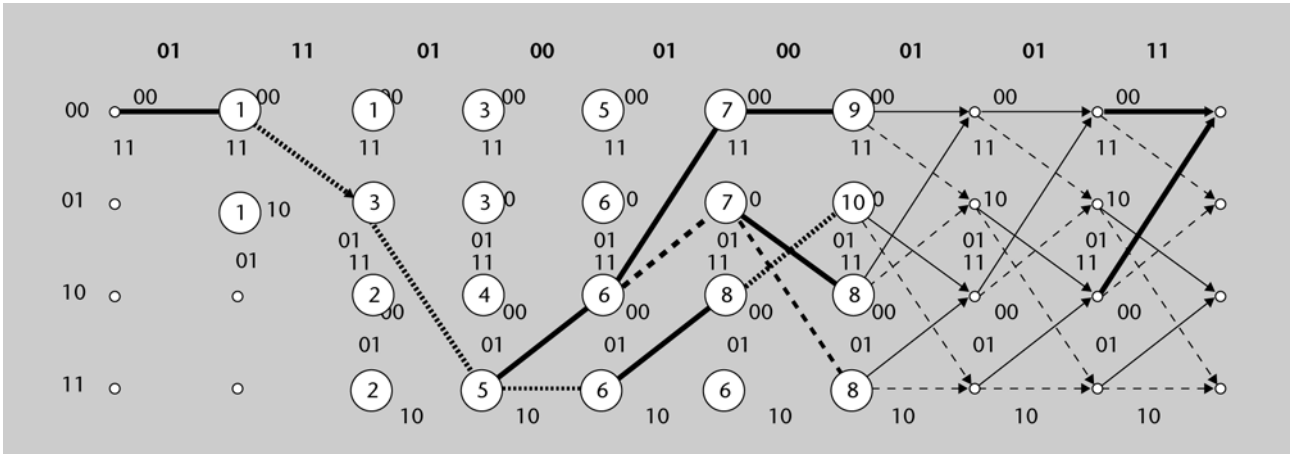

Figura 14. Estado de la descodificación al alcanzar la sexta etapa.

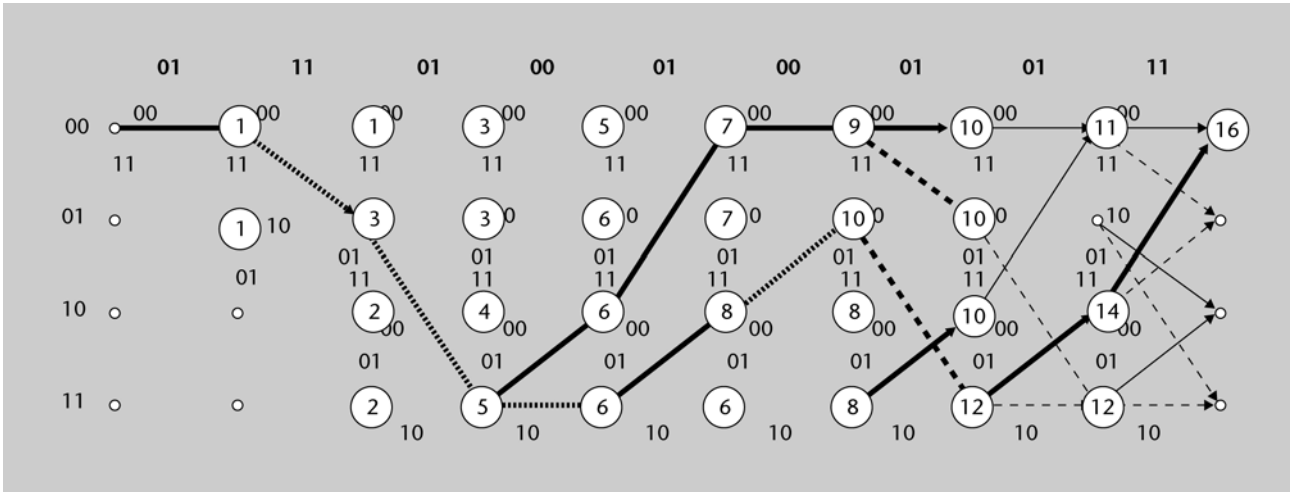

Figura 15. Estado de la descodificación en la última etapa.

En la figura 15 se muestra la resolución completa del algoritmo de Viterbi sobre el diagrama de Trellis. El resultado final obtenido por el algoritmo consiste en extraer el camino que originalmente había utilizado el codificador. Para recuperar la secuencia de bits de información original, debemos empezar a recorrer el diagrama de Trellis siguiendo el convenio utilizado para la codificación (un 'cero' se representa con una transición continua mientras que los 'unos' se representan con líneas discontinuas). De este modo, la secuencia de bits originales que obtenemos es {0 1 1 1 0 1 1 0 0} que, como podemos comprobar, coincide con la de la figura 8. Análogamente, podemos obtener la corrección de la secuencia recibida sin más que seguir en el diagrama de Trellis las salidas que se obtienen en cada transición. Finalmente, observar que el nodo final siempre queda rotulado con el número de bits de la secuencia recibida que han coincidido con el número de bits que hemos descodificado. En nuestro ejemplo el rótulo que aparece en el nodo final es 16, y esto significa que se han producido y corregido dos errores en la secuencia original de 18 bits y que los 16 bits restantes eran correctos.

El algoritmo de Viterbi obtiene la estimación de máxima verosimilitud (ML, *maximum likelihood*) de la secuencia transmitida. Su complejidad es proporcio-

nal al número de estados que existen en el diagrama de Trellis. Esto significa que la complejidad del algoritmo aumenta de forma exponencial con el número de muestras de entrada (unidades de memoria) que utiliza el código convolucional para determinar las salidas (el producto $L \cdot k$ de la figura 3). En la práctica, la complejidad del algoritmo de Viterbi puede ser asumida sólo cuando el número de unidades de memoria del codificador no es excesivo. Cuando la memoria del codificador es considerable, deberán valorarse alternativas al algoritmo de Viterbi que producen descodificaciones no optimizadas. Existen muchas variantes de estos descodificadores alternativos entre las que debemos destacar los algoritmos de Fano (1963), Jelinek (1969) y Heller (1975).

En el algoritmo de Viterbi que hemos visto en este apartado se ha utilizado una descodificación dura (*hard decision*), en la que el receptor decide el valor lógico de cada uno de los bits recibidos antes de iniciar el proceso de descodificación de la secuencia recibida. No obstante, el algoritmo puede generalizarse de forma bastante directa para realizar también una descodificación blanda (llamada *soft*). Existen varias estrategias para generalizar este algoritmo para la descodificación utilizando decisiones blandas (*soft decision*).

La alternativa más utilizada consiste en modificar la métrica de decisión y utilizar el valor absoluto de las diferencias entre la señal recibida y la que teóricamente deberíamos haber recibido en ausencia de ruido en el canal. En este caso, la mejor rama es la que tenga una métrica menor, por lo que deberemos descartar siempre aquellos caminos que presenten el rótulo con valores mayores. Otra posible forma de utilizar el algoritmo es utilizando las diferencias elevadas al cuadrado en vez del valor absoluto. También es posible definir una nueva métrica que permita medir el grado de coincidencia entre la secuencia que idealmente deberíamos haber recibido y la que realmente se ha recibido.

Para ilustrar el mecanismo mediante el cual puede utilizarse el algoritmo de Viterbi utilizando una descodificación blanda, se incluye un ejemplo en el apartado de actividades, análogo al realizado en este apartado, en el que se utilizan las diferencias en valor absoluto entre los valores ideales y los recibidos.

1.5. Elección de los códigos convolucionales

Los códigos convolucionales combinan mediante operaciones de sumas exclusivas los últimos bits de la secuencia de información, es decir, los que están en la memoria del codificador. El nombre de estos códigos se debe a que la operación que realizan es equivalente a la operación de convolución que se utiliza para expresar la salida de un sistema lineal e invariante en función de la entrada y la respuesta impulsional. En nuestro caso, la convolución se realiza entre la secuencia de bits de información y un polinomio que explicita cómo deben realizarse las operaciones de sumas exclusivas entre los bits actuales y los bits almacenados en la memoria del codificador. El polinomio que define el codificador es, por tanto, análogo a la respuesta impulsional que define el filtro lineal.

Aunque parece que puede establecerse un cierto paralelismo entre los sistemas lineales y los códigos convolucionales, en la práctica las técnicas de análisis y diseño son muy distintas. De hecho, existen muchas técnicas

matemáticas para el diseño de filtros que nos permiten determinar los coeficientes de los filtros en función de las características que queremos que tenga nuestro sistema lineal. En cambio, para el diseño de códigos convolucionales no existen estas técnicas sistemáticas en el diseño de los coeficientes del polinomio. La elección de un polinomio u otro se reduce generalmente a la realización de simulaciones y pruebas con distintos polinomios y a la confección de tablas donde se incluyen aquellos polinomios que se han comprobado que tienen 'buenas propiedades'. Además, el concepto de 'buenas propiedades' tampoco está claramente definido, ya que depende de la aplicación y de las características que queremos que tenga el código.

En general, con un código convolucional buscamos que una secuencia de bits de entrada (generalmente larga) se transforme en una palabra código para transmitirla a través del canal. El propósito es que esta palabra código esté más protegida frente al ruido del canal y pueda ser identificada de forma más eficiente en el receptor. Según este principio, esperamos que dos secuencias de información que sean muy distintas produzcan también palabras código que sean muy diferentes. Si esto no fuera así, el código tendría poca utilidad práctica.

1.5.1. Códigos catastróficos

Pongamos un ejemplo de códigos convolucionales inadecuados: los códigos catastróficos, que presentan las características indeseables descritas en los párrafos anteriores. Formalmente, un código se denomina catastrófico cuando para dos secuencias de entrada que difieren en un número infinito de posiciones, se producen dos secuencias de salida que sólo difieren en un número finito de posiciones. Según esta definición, es posible que, si se produce un número finito de errores en la recepción de una secuencia, el descodificador produzca un número infinito de errores en la secuencia descodificada. Sin duda, esta definición puede parece un poco abstracta, por lo que es mejor utilizar un ejemplo de código catastrófico para entender más claramente cuáles son sus efectos.

Consideremos como ejemplo el siguiente código, cuya realización y diagrama de estados se representa en la figura 16.

- Número de líneas de entrada: $k = 1$
- Número de líneas de salida: $n = 2$
- Tasa del código: $R = 1/2$
- Unidades de memoria: 2
- Número de posibles estados: $2^2 = 4$
- Longitud total de la convolución: 3
- Polinomio generador 1: $G_1 = X^2 + X = 6_{OCT}$
- Polinomio generador 2: $G_2 = X + 1 = 3_{OCT}$

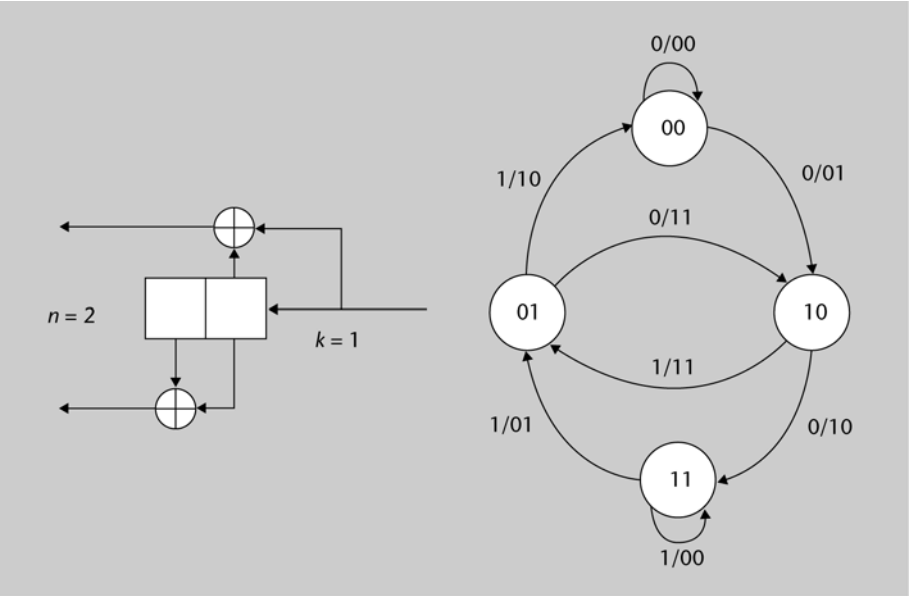

Figura 16. Ejemplo de un código convolucional catastrófico.

Para comprobar que el código es catastrófico, supongamos que la secuencia de información es una secuencia con todos los símbolos igual a la unidad {1 1 1 1 1 …. 1 1 1}. La secuencia que obtenemos en la salida del codificador puede calcularse muy fácilmente a partir del diagrama de estados:

{10 01 00 00 00 00 00 00 … 00 00 10 01}

Por otra parte, si la secuencia de entrada son todo ceros { 0 0 0 0 0 0 ….. 0 0 }, la salida que obtenemos es:

{ 00 00 00 00 00 00 …. 00 00 00}

Este resultado nos indica que aunque las secuencias de entrada difieren en todos los bits (infinitos, si las secuencias tienen una longitud infinita), las secuencias de salida sólo difieren en un número finito de bits, en concreto 4 bits para nuestro ejemplo.

Los códigos catastróficos pueden detectarse de forma más o menos inmediata a partir de su diagrama de estados. La forma más sencilla de detectarlos es observar que un código es catastrófico cuando tiene algún estado en el que sus salidas son siempre cero cuando la secuencia de entrada tiene bits diferentes de cero. En nuestro ejemplo, el problema está en el estado {11}, ya que aunque la secuencia de entrada sea distinta de cero {1 1 1 1 1….}, puede producir una salida en la que todo son ceros.

1.5.2. Distancia libre de un código

Una característica que suele utilizarse para indicar que un código tiene buenas prestaciones es la distancia libre del código (d_{free}). La distancia libre se define como el número de 'unos' que tiene la secuencia de salida **más corta** tal que

partiendo del estado {00} vuelve al estado {00}. En el ejemplo del código catastrófico anterior, la distancia libre del código es 4. En efecto, el camino más corto que sale y retorna al estado {00} pasa por los estados {00} => {01} => {10} => {00}, y las salidas que obtenemos son {10 11 01}. La distancia libre puede interpretarse como un equivalente a la palabra código de peso mínimo que se utiliza en los códigos de bloque. En este caso, la salida que hemos obtenido representa la secuencia de bits de entrada más próxima a la palabra todo ceros. Es importante notar que una distancia libre grande no nos garantiza que el código tenga buenas prestaciones. Es fácil imaginar un diagrama de estados en el que la distancia libre es muy grande y, en cambio, existen estados que provocan que el código resultante sea catastrófico.

En la práctica existen tablas en las que se proporcionan polinomios generadores que se sabe a partir de simulaciones que tienen buenas propiedades de distancia. Estas tablas dependen de las tasas del código. En la tabla siguiente se proporcionan los polinomios generadores (en notación octal) para códigos de tasa 1/2.

Tasa código = 1/2; Códigos con d_{free} máxima			
L Long. conv.	**Polinomios generadores octal**		**d_{free}**
3	5	7	5
4	15	17	6
5	23	35	7
6	53	75	8
7	133	171	10
8	247	371	10
9	561	753	12
10	1167	1545	12
11	2335	3661	14
12	4335	5723	15
13	10533	17661	16
14	21675	27123	16

Tabla 2. Tabla de códigos convolucionales de tasa 1/2 con distancias libres máximas.

1.5.3. Longitud de convolución

Un parámetro muy importante es la longitud de la convolución que afecta a las capacidades correctoras del código. Así, por ejemplo, para un sistema de comunicaciones que trabaje con una relación E_0/N_0 de 5 dB, con modulaciones QPSK, podemos obtener una probabilidad de error típica (BER, *Bit Rate Error*) de $5 \cdot 10^{-3}$, cuando no utilizamos ningún código convolucional. Si mantenemos la relación E_0/N_0 pero se utiliza un código convolucional

con una tasa de código $R = 1/2$ y una longitud de convolución de 2, obtenemos una reducción en la BER (después del código) de aproximadamente $3 \cdot 10^{-5}$, lo que nos da una idea de la mejora introducida por el código convolucional. Si la longitud de convolución aumenta hasta 7, la tasa de error se reduce hasta $5 \cdot 10^{-7}$.

Para poner un ejemplo, los sistemas de difusión de televisión digital (DVB, *digital video broadcasting*) usan códigos convolucionales de tasa $R = 1/2$ y longitud de convolución de 7. Con ello, trabajando con una relación E_0/N_0 típica de 3,2 dB se consigue obtener una tasa de error de $2 \cdot 10^{-4}$. Esta tasa de error puede parecer excesiva para un sistema de comunicaciones real, pero debemos tener en cuenta que en DVB se utilizan dos códigos concatenados, que complementan la corrección de errores. En este ejemplo, además, juntamente con el código convolucional se utiliza un código de Reed-Solomon que reducirá la tasa de error hasta aproximadamente 10^{-11}.

1.6. *Puncturing*

Una de las desventajas de los códigos convolucionales es que sus tasas de código son muy bajas. En efecto, el codificador que hemos estado considerando en nuestro ejemplo tiene una tasa $R = 1/2$, lo que significa que un 50% de los bits en la secuencia codificada corresponden a redundancia.

Una técnica directa para mejorar la tasa del código consiste en realizar la 'perforación' (*puncturing*) de la secuencia codificada resultante. Esta técnica consiste en dejar de transmitir sistemáticamente algunos bits de la secuencia resultante. Así, por ejemplo, podríamos dejar de transmitir uno de cada tres bits, con lo que un código que tuviera una tasa $R = 1/2$ pasaría a tener una tasa $R = 1/2 \times 3/2 = 3/4$. A pesar de que las capacidades correctoras del código se ven alteradas al realizar la perforación, con algunos pequeños cambios en el procedimiento, podemos seguir utilizando el algoritmo de descodificación de Viterbi.

En efecto, en la figura 17 se muestra un ejemplo de una secuencia de información que se codifica con un código de tasa $R = 1/2$. Cada bit de la información original da lugar a dos bits de la secuencia codificada. Antes de realizar la transmisión, el código se 'perfora' (*puncture*) con una tasa de 3/2, es decir, de cada 3 bits de la secuencia de datos sólo transmitimos los dos primeros. Si suponemos que la secuencia se recibe sin errores, el descodificador puede insertar bits desconocidos (indicados con X) en aquellas posiciones que sabe que el codificador no ha transmitido. Para realizar el proceso de descodificación con el algoritmo de Viterbi, las X son ahora valores desconocidos en la señal recibida que no se utilizan para incrementar o decrementar las coincidencias entre la secuencia descodificada y la secuencia recibida.

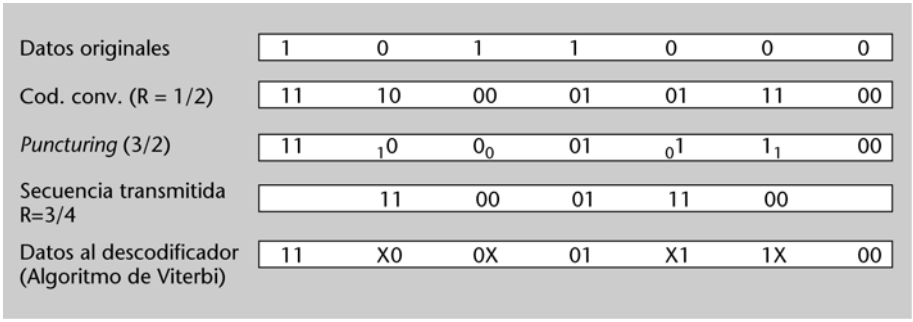

Figura 17. Proceso de codificación, *puncturing* y obtención de la secuencia de entrada al algoritmo de Viterbi.

2. Concatenación de códigos

La concatenación de códigos consiste en aplicar un código de protección de errores a una secuencia de bits a la que ya se ha aplicado otro código de protección. El concepto puede parecer muy trivial y sugiere que posiblemente se trata de una simplificación para no utilizar códigos de protección más complejos. No obstante, veremos que con la concatenación de códigos pueden obtenerse características muy específicas, que no pueden obtenerse con un único código.

En la figura 18 se muestra un ejemplo en el que se concatenan dos códigos de protección de errores. En el emisor, la secuencia de bits originales se pasa a través de un primer código llamado código externo. El resultado de esta primera codificación se pasa a través de otro descodificador que denominamos código interno. Los datos obtenidos tras la concatenación de los dos códigos se envían al canal. En el receptor, la secuencia se descodifica en el sentido inverso, es decir, primero se aplica el descodificador interno y posteriormente el externo. La tasa del código resultante total es el producto de las tasas de los dos códigos. Esto es, si la tasa de bit en la entrada del primer código es R_0, la tasa de bit en la salida será R_0/R_1, donde R_1 es la tasa del código externo. Esta es la tasa de bits que entra al segundo codificador, obteniendo una tasa de bit final R_0/R_1R_2, por lo que puede considerarse que la tasa del código concatenado es el producto de las tasas individuales de cada código.

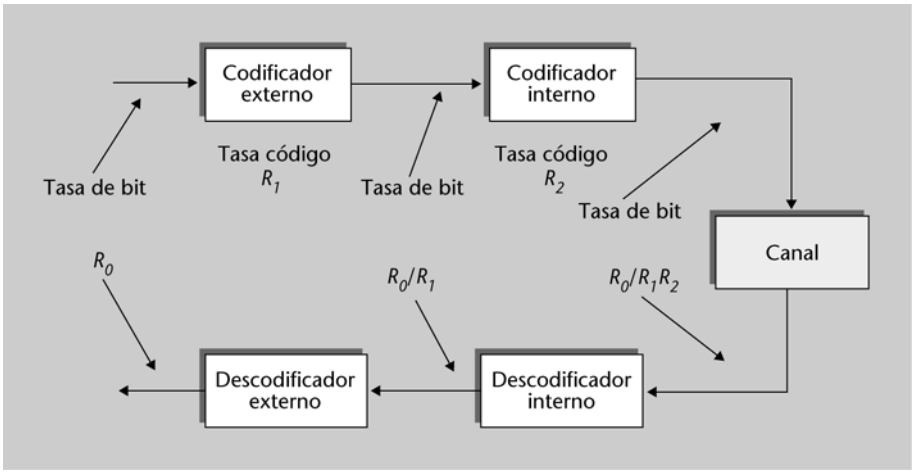

Figura 18. Diagrama de bloques de la concatenación de dos códigos.

2.1. Ventajas de la concatenación de códigos

Para ver claramente la ventaja que supone la concatenación de dos códigos, vamos a poner un ejemplo concreto en el que se utiliza un código de Reed-Solomon (código RS) como código externo y un código de Hamming como có-

digo interno. Tal y como se muestra en la figura 19, el código RS toma un conjunto de *K* símbolos y añade *R* símbolos de redundancia. Vamos a suponer que puede llegar a corregir hasta un total de *C* símbolos, donde cada símbolo está compuesto de 8 bits. El código interno, de Hamming, toma un paquete de 4 bits (la mitad de un símbolo) y le añade 3 bits de redundancia, pudiendo llegar a corregir 1 bit por cada paquete.

La concatenación de los dos códigos presenta las ventajas derivadas de cada uno de los códigos individuales. En efecto, si en el canal se producen muchos errores aislados (de un bit por cada paquete), estos pueden ser corregidos por el código interno (de Hamming), por lo que la información se recibe sin errores en el código externo. En cambio, si se producen ráfagas de errores, que afectan a varios bits consecutivos, no podrán ser corregidos por el código interno pero, en este caso, entrará en acción el código RS. En consecuencia, gracias a la concatenación de los dos códigos podemos llegar a obtener un sistema con capacidad para corregir errores aislados y ráfagas de errores.

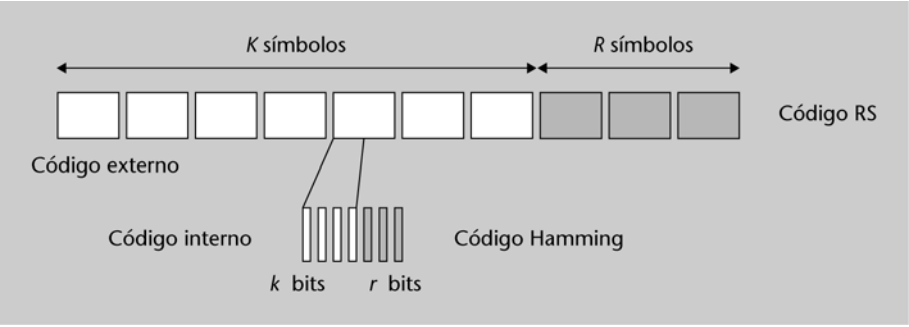

Figura 19. Concatenación de un código de Reed-Solomon con un código de Hamming.

2.2. Intercalador

La concatenación de códigos estudiada en el apartado anterior nos permite corregir gran cantidad de errores de bits aislados, gracias al código de Hamming, y ráfagas de errores que involucren hasta un total de *C* símbolos en la trama RS. El número de símbolos que pueden corregirse depende de las características del código RS. Si una ráfaga de errores afecta a más de *C* símbolos, no podrá realizarse la corrección de forma adecuada.

Una posible estrategia para mejorar el comportamiento de los códigos concatenados es añadir un intercalador entre los dos códigos. Con ello conseguimos protección frente a ráfagas de errores que afecten a más de *C* símbolos. En la figura 21 se representa un esquema básico de un código concatenado que contiene un intercalador. El intercalador consiste en alterar las posiciones, el orden en el que se transmiten los bits, de manera que aquellos bits que en la trama original aparecen consecutivos quedan desordenados en la trama transmitida. En el receptor el desintercalador se encarga de restaurar el orden original de los bits. La ventaja de desordenar los bits (o los símbolos) antes de

transmitirlos permite dispersar los errores de ráfaga, tal y como se ilustra en la figura 21.

Figura 20. Uso de un intercalador en códigos concatenados.

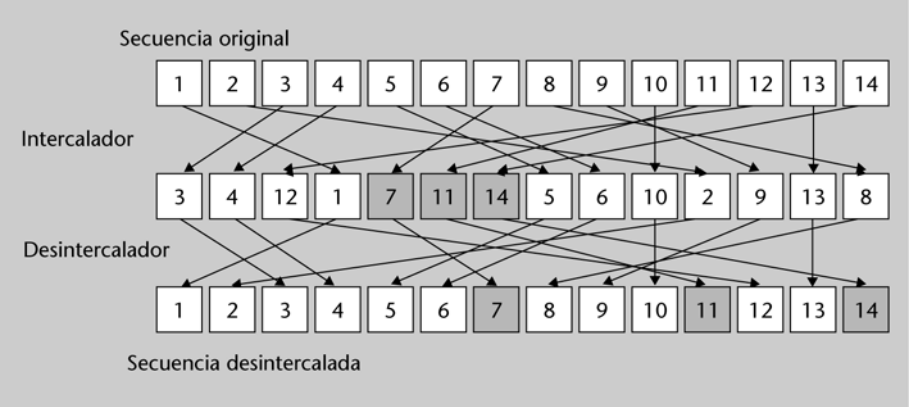

Figura 21. Dispersión de los errores de ráfaga mediante el uso de un intercalador y un desintercalador.

Existen dos alternativas para realizar un intercalador:

- el intercalador de bloques (*block interleaver*),
- el intercalador convolucional (*convolutional interleaver*).

El principio de funcionamiento del intercalador de bloques es muy simple. Se trata de una memoria organizada en forma de matriz (*array*) en la que la secuencia de símbolos de entrada se escribe fila a fila, tal y como se indica en la figura 22. Una vez que todo el bloque de memoria ha sido escrito se realiza la lectura y se transmiten los símbolos en el sentido de las columnas. En el receptor, los símbolos recibidos se escriben en la memoria columna a columna. Por lo tanto, si se ha producido una ráfaga de errores durante la transmisión de los datos, todos los símbolos erróneos estarán en la misma columna (o en columnas adyacentes). La lectura de la memoria se realiza ahora por filas, de manera que la ráfaga de errores que afectaba a varios símbolos ahora sólo afecta a un número reducido de símbolos, por lo que podrán ser corregidos por códigos no excesivamente complejos.

Las técnicas de IL de bloque generan inevitablemente un retraso en la desmodulación de los bits que depende de las dimensiones de la matriz de entrelazado. Aproximadamente, el retraso es del tiempo de N bits en el modulador más el tiempo de N bits en el desmodulador, de manera que se acumula un retraso total de $2N$ bits. Para disminuir el retraso por entrelazado, se utilizan las técnicas de IL convolucionales, mediante las que el retraso máximo a cau-

sa del entrelazado queda dividido por dos y es siempre menor que un retraso de *N* bits.

Los intercaladotes convolucionales se basan en pasar los símbolos de entrada a través de un sistema formado por *L* unidades de retardo, cada una de ellas con retardos distintos, e ir conmutando la entrada y salida de los símbolos, de forma sincronizada, tal y como se indica en la figura 23. La principal ventaja de esta tecnología de intercalación es que la sincronización de los bloques de código en el receptor es más simple.

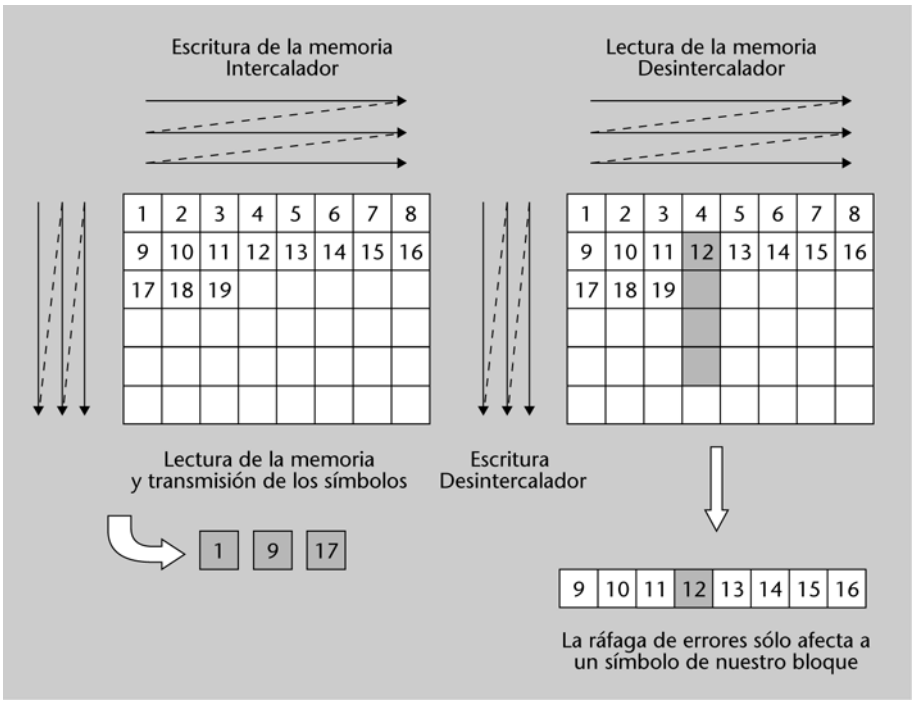

Figura 22. Esquema de funcionamiento de un intercalador y un desintercalador de bloque.

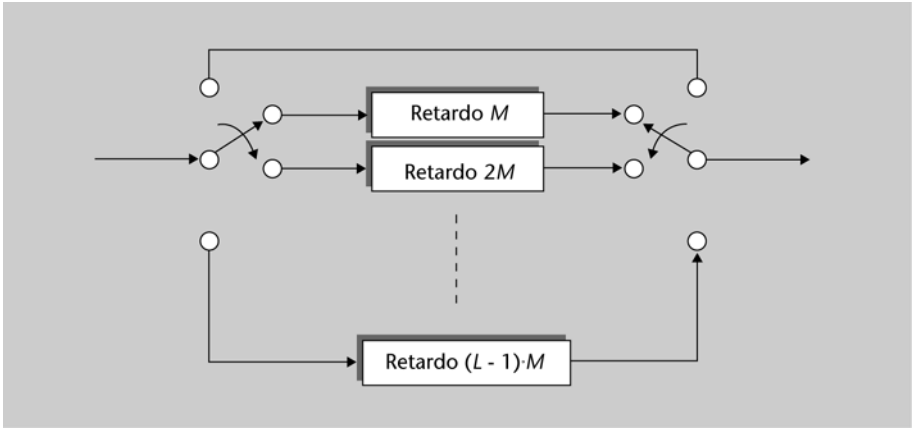

Figura 23. Diagrama de un intercalador convolucional.

A continuación, se explica la manera de procesar los bits en un IL convolucional.

1) En la operación de IL. A partir de la secuencia de bits, se procesa secuencialmente una trama de K bits, de manera que K es un múltiplo entero de un parámetro B, también entero.

- El primer bit de la trama no se retrasa,
- el segundo bit de la trama sufre un retraso igual a K períodos de bit,
- el tercer bit de la trama sufre un retraso igual a 2K períodos de bit,
- ...,
- el B-ésimo bit de la trama sufre un retraso de (B – 1)K períodos de bit,
- el B + 1-ésimo bit de la trama no se retrasa,
- el B + 2-ésimo bit de la trama sufre un retraso igual a K períodos de bit,
- etc.

2) En la operación de desentrelazado (DeIL) del desmodulador se complementan los retrasos. A partir de la secuencia de bits, se procesa secuencialmente una trama de K bits.

- El primer bit de la trama sufre un retraso de (B – 1)K períodos de bit,
- el segundo bit de la trama sufre un retraso igual a (B – 2)K períodos de bit,
- el tercer bit de la trama sufre un retraso igual a (B – 3)K períodos de bit,
- ...,
- el B-ésimo bit de la trama no se retrasa,
- el B + 1-ésimo bit de la trama sufre un retraso de (B – 1)K períodos de bit,
- el B + 2-ésimo bit de la trama sufre un retraso igual a (B – 2)K períodos de bit,
- etc.

Cada bit en particular, sumando el retraso que ha sufrido en el modulador con el retraso que ha sufrido en el desmodulador, ha acumulado un retraso de (B – 1)K bits.

2.3. Concatenación de códigos en DVB

Un ejemplo típico de concatenación de códigos son los sistemas de radiodifusión de señales de televisión digitales, tanto en su modalidad de televisión digital por satélite como la de televisión digital terrestre. Estos dos sistemas están regulados por los estándares ETS 421 (satélite) y ETS 744 (terrestre), y la arquitectura de los componentes que intervienen en la codificación de canal es idéntica en ambos casos. La normativa de codificación de canal y transmisión de estos sistemas se establece en el foro internacional DVB (*digital video broadcasting*). La codificación de canal en los sistemas terrestre y satélite consiste en la concatenación de un código de Reed-Solomon (externo) con un código convolucional (interno) y un intercalador también convolucional.

El código de Reed-Solomon utiliza símbolos de 8 bits. Está diseñado para corregir un total de 8 símbolos en cada paquete de datos utilizando un código con 239 símbolos de información y 16 símbolos de redundancia. Así pues, el código RS de partida es un código (255,239).

No obstante, la trama de transporte del MPEG-2 utiliza paquetes de datos de 188 bytes. Esto condiciona que los bloques de información del código RS deban tomarse en módulos de 188 bytes. La solución consiste en utilizar un código (255,239) en el que, de los 239 bytes de información, los 51 primeros se ponen a cero, y aunque no es necesario enviarlos, se utilizan para calcular los 16 símbolos de redundancia que se añaden. El código RS resultante es una versión recortada del código (255,239) que se denota como (204,188).

El intercalador utilizado en el DVB tiene una arquitectura convolucional y utiliza un total de 12 unidades de retardo (L) con un retardo de cada una que es un múltiplo entero de 17 símbolos (M) (véase la figura 23).

El código convolucional tiene una tasa $R = 1/2$ con una longitud de la convolución de 7 bits. Los polinomios generadores vienen dados por $G_1 = 171_{OCT}$ y $G_2 = 133_{OCT}$. Opcionalmente, podemos aplicar *puncturing* al código convolucional resultante eligiendo unas tasas de código de 2/3, 3/4, 5/6 o 7/8.

La codificación de televisión digital por cable es algo más simple debido a que la transmisión por cable está más protegida frente a errores e interferencias. En este caso, no se utiliza el código convolucional interno.

3. Turbo códigos

Los turbo códigos pueden considerarse un caso especial de códigos concatenados, con ciertas peculiaridades que los hacen muy efectivos en cuanto a su capacidad de corrección de errores y la simplicidad en la implementación del codificador. Sin embargo, en su contra está la complejidad computacional de los algoritmos de descodificación. Debido a estas características, son especialmente utilizados en aquellas aplicaciones en las que el transmisor tiene una complejidad computacional limitada y un coste reducido, mientras que el receptor puede ser complejo. Un escenario de aplicación típico son los equipos o terminales de telefonía móvil, generalmente con una capacidad de procesamiento numérico reducida pero que se comunican con una estación central en la que pueden realizarse procesos de cálculo complejos.

Existen distintos tipos de configuraciones cuyo estudio detallado escapa a los propósitos de este texto. La configuración más utilizada es la representada en la figura 24 que supondremos que trabaja con paquetes de información de N bits. Existen dos codificadores convolucionales con tasas de código $R = 1/2$ que transmiten en paralelo y están conectados a través de un intercalador. Los códigos RSCC que se utilizan son **códigos convolucionales sistemáticos y recursivos** (RSCC, *recursive systematic convolutional code*). La propiedad de que sean *sistemáticos* es equivalente a la definición que utilizábamos para los códigos de bloque, es decir, los primeros bits de la secuencia codificada coinciden con los bits de información. La *recursividad* hace referencia a la estructura del generador del código. Hasta ahora, todos los códigos convolucionales que hemos estudiado son no recursivos, de manera que el bit de salida sólo depende de los bits de entrada retardados. En los codificadores recursivos aparecen realimentaciones entre los registros de desplazamiento. Puede verse la realización de un código convolucional recursivo en la figura 25. Obsérvese que es bastante directo establecer una relación entre los códigos convolucionales recursivos y los filtros de respuesta impulsional infinita (IIR, *Infinite Impulse Response*), en los que la salida del filtro se realimenta para obtener las sucesivas salidas, del mismo modo que los códigos no recursivos se relacionan con los filtros de respuesta impulsional finita (FIR, *Finite Impulse Response*), en los que la salida se calcula a partir de las muestras de la señal de entrada. Por tanto, la recursividad en la implementación del código supone que la longitud de la convolución puede considerarse infinita. Así pues, con un procesado relativamente reducido podemos implementar filtros con una longitud de convolución equivalente muy grande y con buenas prestaciones para su protección frente a errores.

En la implementación del turbo código representado en la figura 24, los N bits de información se transmiten directamente al canal, de manera que cada uno

de los códigos convolucionales recursivos, con $R = 1/2$, sólo transmite los N bits de redundancia. El número total de bits que se envían al canal es, por tanto, de $3N$, por lo que la tasa del código final es $R = 1/3$. Para hacer estos turbo códigos algo más eficientes, es habitual utilizar técnicas de *puncturing*.

Una característica de los turbo códigos, que afecta directamente a sus prestaciones, es la longitud equivalente sobre la que se realiza el entrelazado. En las implementaciones más habituales, este bloque suele afectar del orden de unos 1.000 bits o más.

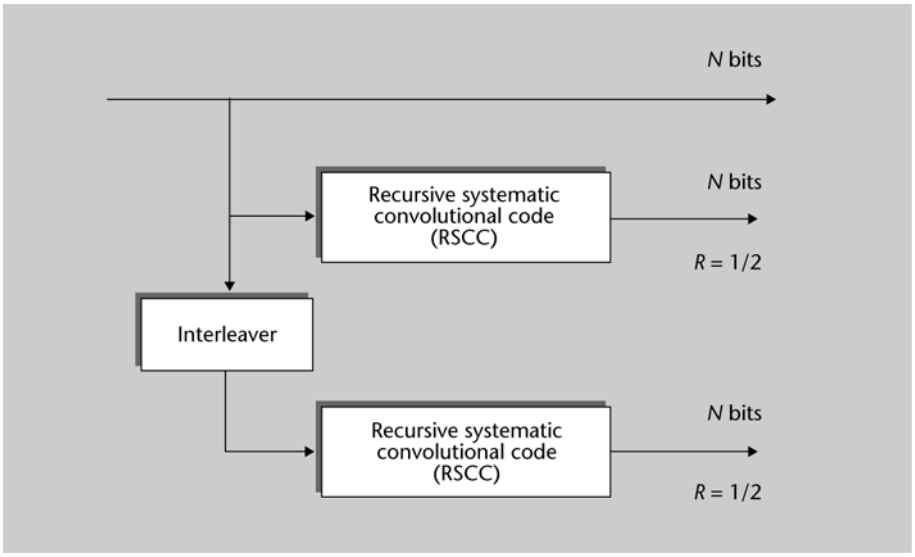

Figura 24. Diagrama de bloques de una implementación de un turbo código.

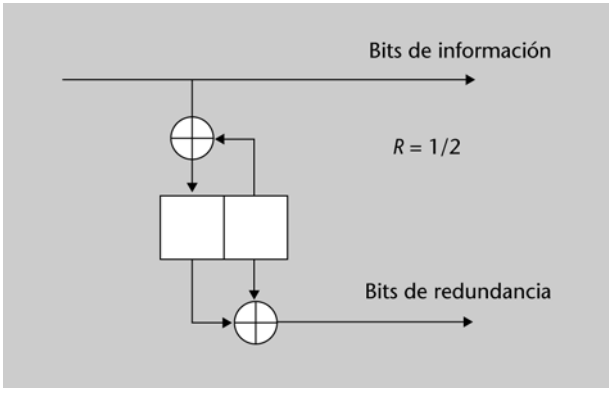

Figura 25. Ejemplo de un código convolucional sistemático (transmite los bits de información) y recursivo.

Actividades

Descodificación de Viterbi con decisiones *soft*

Consideremos el codificador de la figura 4, cuyo diagrama de estado y diagrama de Trellis están representados en las figuras 5 y 6. Supongamos que transmitimos la secuencia de información que se muestra en la figura 26 a través de un canal gaussiano y sabemos que el detector producirá una tensión de valor medio 1 voltio para el bit '1' y un valor de tensión medio de –1 voltio para el valor lógico '0'. La secuencia recibida se muestra también en la figura 27. Determinaremos la secuencia corregida que obtendremos utilizando un descodificador de Viterbi que utilice una descodificación blanda (*soft*).

Información	0 1 1 1 0 1 1 0 0
Secuencia transmitida	00 11 01 10 01 00 01 01 11
Secuencia detectada	{-0.8,1.2; 0.9, 0.8; -1.1,0.9; -0.6,-1.2; -1, 1.1; -0.8,-0.6 -0.9, 0.9; -1.3,0.7; 1.1,0.9}

Figura 26. Ejemplo de descodificación software mediante el algoritmo de Viterbi.

Solución

El proceso de descodificación de la secuencia es parecido al que se ha realizado para la descodificación hardware aunque ahora la forma como se contabiliza la métrica es distinta. En la figura 27 se muestran los resultados obtenidos hasta alcanzar la segunda etapa del proceso de descodificación. Veamos con detalle cómo se han rotulado los dos nodos de la primera etapa.

El nodo correspondiente al estado {00} de la primera etapa ha sido rotulado con un 2.4. Para llegar a este nodo deberíamos haber tenido una salida {00} que se correspondería con unas tensiones de –1 V, –1 V. Las tensiones que hemos recibido son (–0,8, 1,2), por lo que la diferencia en valores absolutos entre la señal recibida y la que deberíamos haber recibido es de: abs(–1 – (–0,8)) + abs(1,2 – (–1)) = 2,4. Análogamente, si realizamos un cálculo parecido para el estado {01} de la primera etapa, obtenemos una diferencia en valor absoluto de 2. Para la segunda etapa, en cada nodo debemos sumar el error absoluto acumulado hasta el nodo anterior con el error cometido en esta transición.

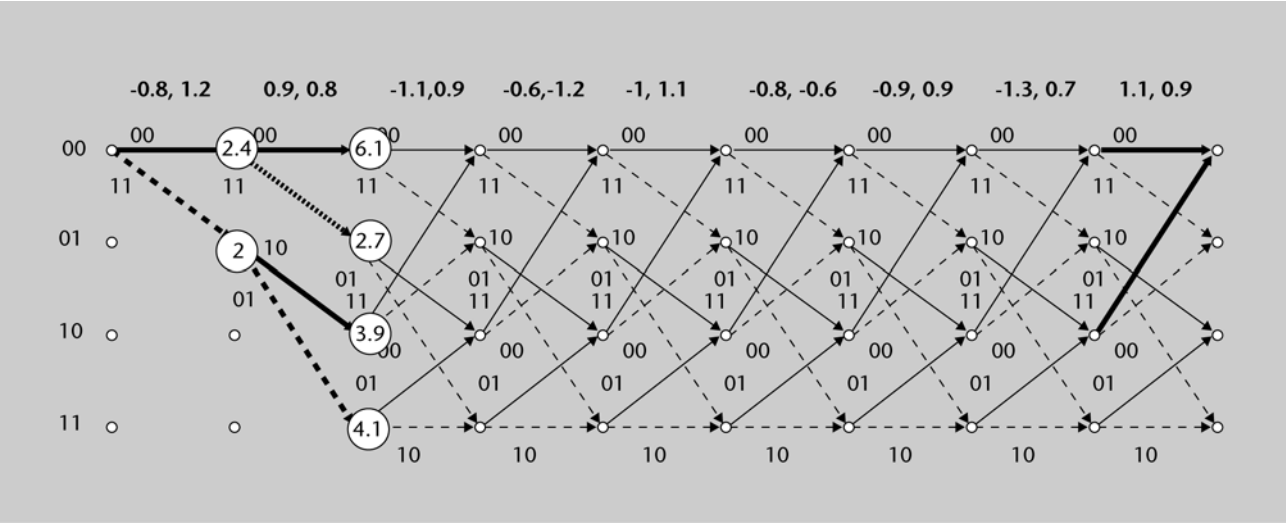

Figura 27. Primeras dos etapas de la descodificación software de una secuencia mediante el algoritmo de Viterbi.

Para continuar con la tercera etapa debemos valorar los dos caminos que llegan a cada uno de los nodos, quedándonos ahora con el que tenga el coste más pequeño. Igual que antes, eliminaremos todos aquellos caminos que queden cortados en un nodo de la etapa anterior. Los resultados obtenidos después de la tercera etapa se muestran en la figura 28. En las siguientes ilustraciones se muestran los resultados obtenidos en las etapas 4, 5, 6, 7 y, final-

mente, las etapas 8 y 9. Obsérvese cómo el camino óptimo final coincide con el del ejemplo que se ha analizado en el texto, por lo que los errores han sido corregidos correctamente.

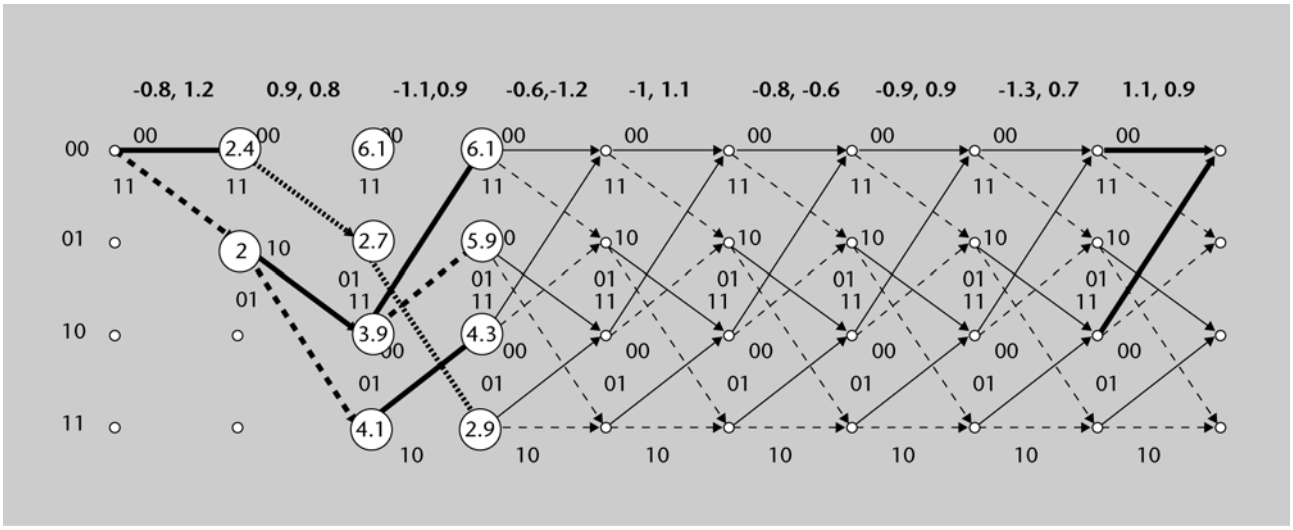

Figura 28. Descodificación software en la tercera etapa con el algoritmo de Viterbi.

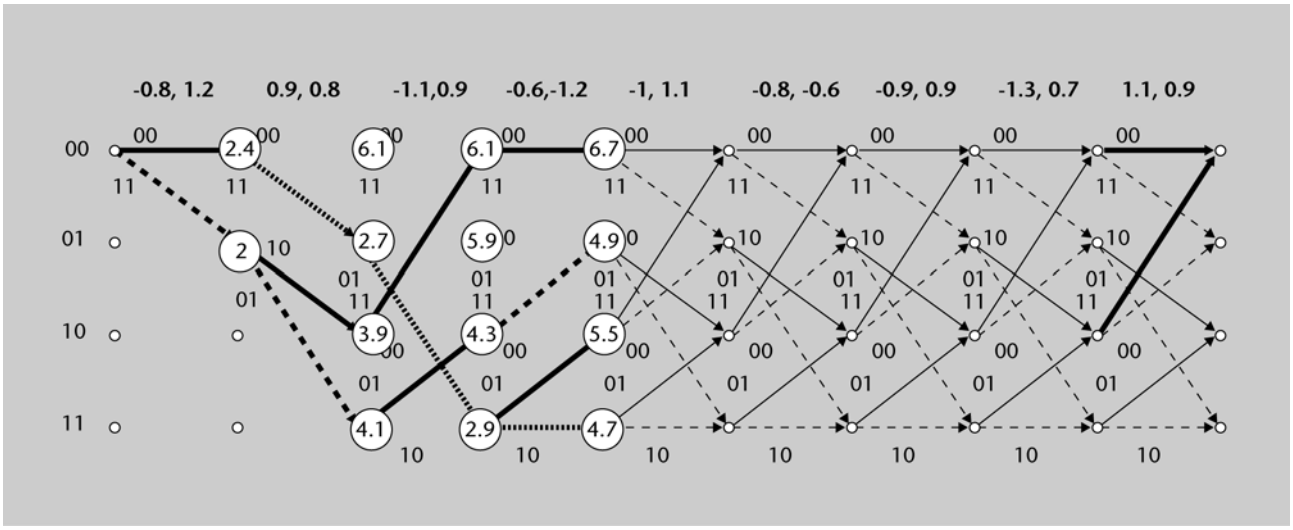

Figura 29. Descodificación software en la cuarta etapa con el algoritmo de Viterbi.

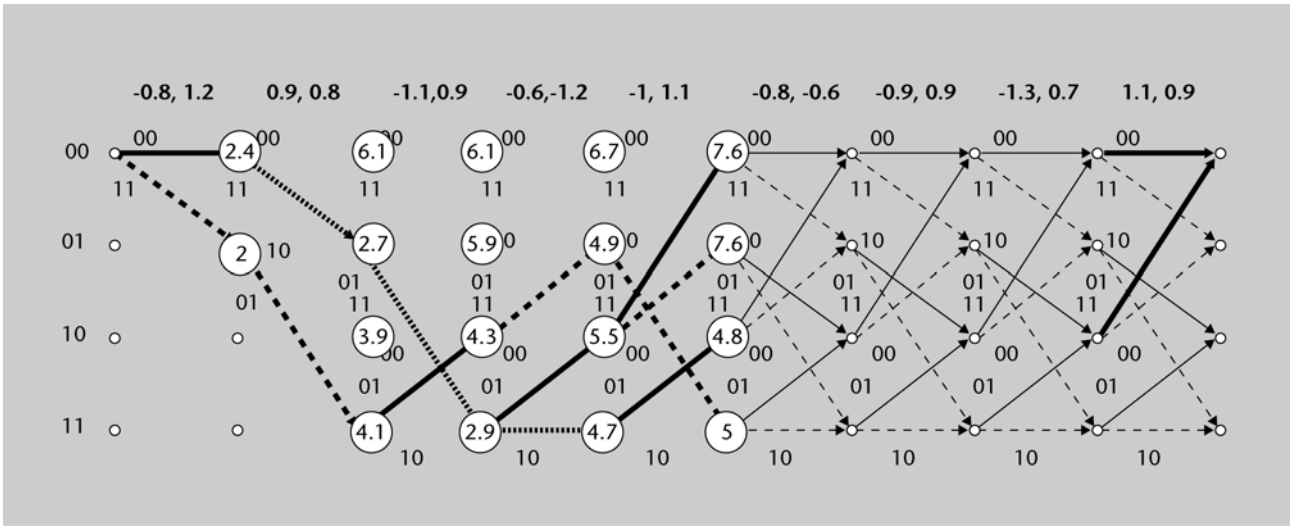

Figura 30. Descodificación software en la quinta etapa con el algoritmo de Viterbi.

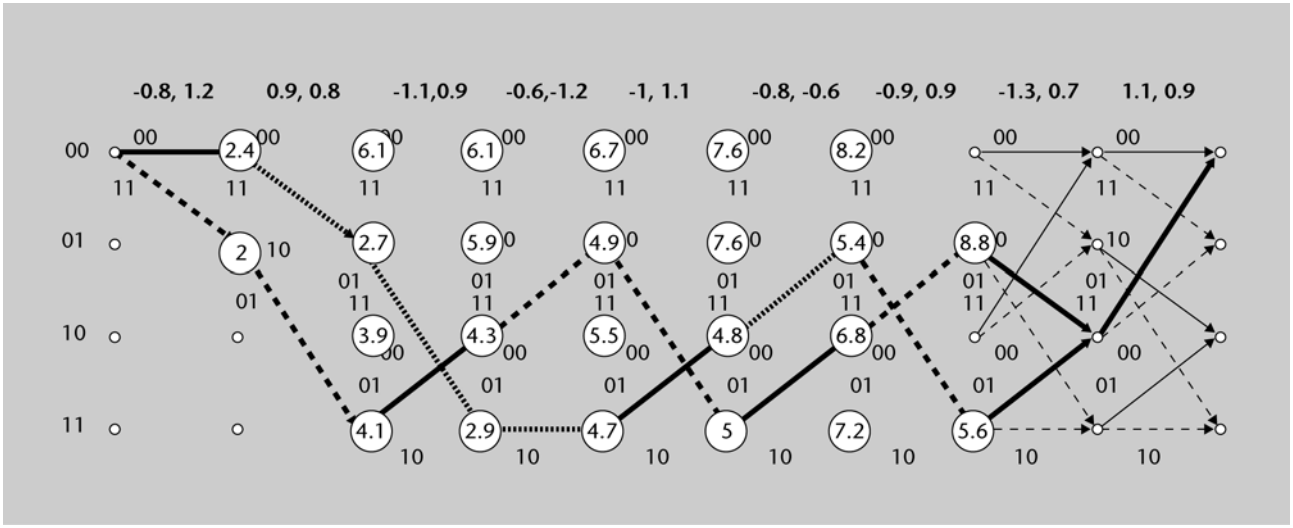

Figura 31. Descodificación software en la séptima etapa con el algoritmo de Viterbi.

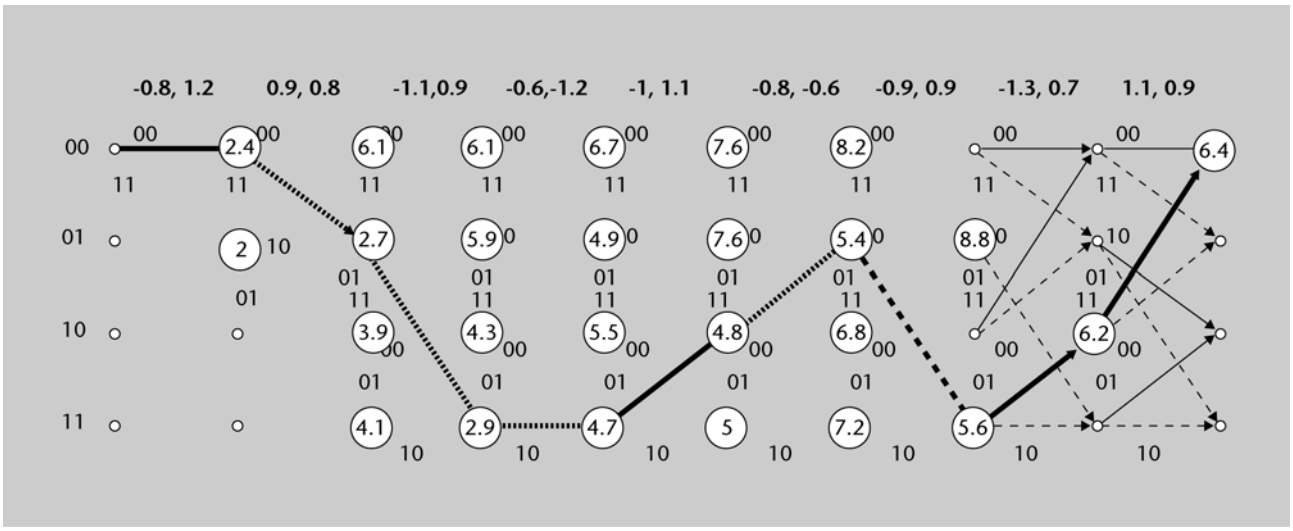

Figura 32. Descodificación software con el algoritmo de Viterbi. Resultado final.

Ejercicios de autoevaluación

1. Un código convolucional queda descrito por los siguientes parámetros:

- Número de líneas de entrada: $k = 1$
- Número de líneas de salida: $n = 3$
- Tasa del código: $R = 1/2$
- Unidades de memoria: 2
- Número de posibles estados: $2^2 = 4$
- Longitud total de la convolución: 3
- Polinomio generador 1: $G_1 = 1 = 1_{OCT}$
- Polinomio generador 2: $G_2 = X^2 + 1 = 5_{OCT}$
- Polinomio generador 3: $G_3 = X^2 + X + 1 = 7_{OCT}$

A partir de estos datos:

a) Determinad el diagrama de bloques del codificador.
b) Determinad y representad el diagrama de estado.
c) Representad el diagrama de Trellis.
d) Verificad si se trata de un código catastrófico.
e) Calculad su distancia libre.

2. Para los datos del ejercicio anterior, determinad:

a) La secuencia codificada que obtenemos cuando la entrada es {1 0 1 1 1}.
b) Tomando el resultado del apartado anterior, supongamos que se ha producido el siguiente patrón de error {001 000 100 000 000 000 000}. ¿Cuál es la palabra que recibiremos?
c) Utilizando el resultado del apartado anterior determinad, utilizando el algoritmo de Viterbi, la estimación ML de la secuencia que se ha transmitido originalmente.

3. Considerad un código convolucional cuyo diagrama de bloques se representa en la figura 33.

a) Determinad la tasa del código, los polinomios generadores y el número de estados necesarios para representarlo.
b) Representad el diagrama de estados del código.
c) Representad el diagrama de Trellis.
d) Determinad la salida del sistema para la siguiente secuencia de entrada {1 1 0 1 1}.
e) Supongamos que se transmite una secuencia en un canal con ruido y que se recibe el siguiente resultado: {101 011 101 100 001 101 011}. Utilizando el algoritmo de Viterbi realizad una estimación ML de la secuencia original que se ha enviado. Asimismo, calculad el número de errores que se estima que se han producido y determinad la secuencia de datos original.

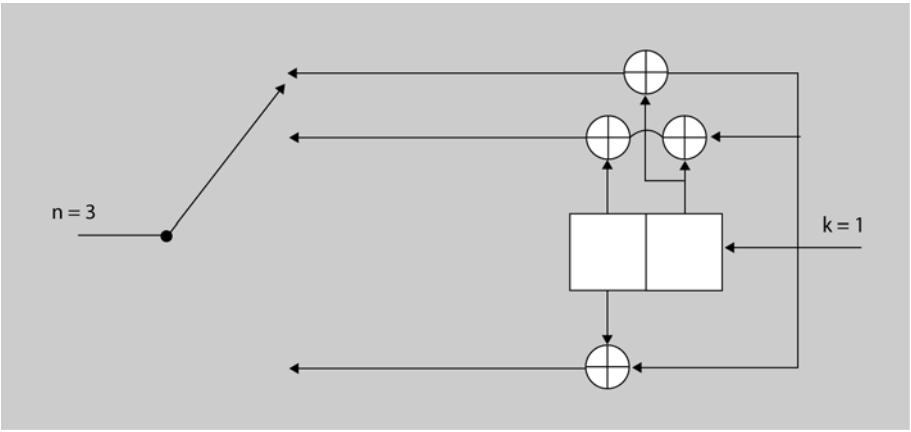

Figura 33. Diagrama de un código convolucional con tasa 1/3.

Bibliografía

Bibliografía básica

Proakis, J. G.; **Salehi, M.** (2002). *Communication Systems Engineering* (2.ª ed.). Prentice Hall.

Benedetto, S.; Biglieri, E. (1999). *Principles of Digital Transmission*. Kluwer Academic Press / Plenum publishers.

Proakis, J. G. (2003). *Digital Communications* (4.ª ed.). McGraw-Hill.

Bibliografía complementaria

Carlson, A. B. (2001). *Communication Systems: An Introduction to Signals and Noise in Electrical Communication* (4.ª ed.). McGraw Hill.

Viterbi, A. J. (1967). "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm". *IEEE Trans. on Information Theory* (núm. 2, pág. 260-269).

Fano, R. M. (Abr., 1963). "A Heuristic Discussion of Probabilistic Decoding". *IEEE Trans. on Information Theory* (vol. 9, pág. 64-74).

Jelinek, F. (nov., 1969). "Fast Sequential Decoding Algorithm Using a Stack". *IBM J. Res. Dev.* (vol. 13, pág. 675-685).

Heller, J. A. (1975). "Feedback Decoding of Convolutional Codes". *Advances in Communications Systems* (vol. 4.). J. Viterbi (ed.). Nueva York: Academic Press.

Stix, G. (sept., 1991). "Encoding the Neatness of Ones and Zeroes". *Scientific American* (pág. 54-58).

Segal, J. "El geómetra de la información". *Temas Investigación y Ciencia: La Información* (núm. 36, pág. 12-15).

Bekenstein, J. D. "La información en el Universo Holográfico". *Temas Investigación y Ciencia: La Información* (núm. 36, pág. 16-23).

Gibson, Jerry D. *et al.* (1998). *Digital Compression for Multimedia: Principles & Standards*. Morgan Kauffman.

Tarrés, F. (2001). *Sistemas Audiovisuales I: Televisión Analógica y Digital*. Ediciones UPC.