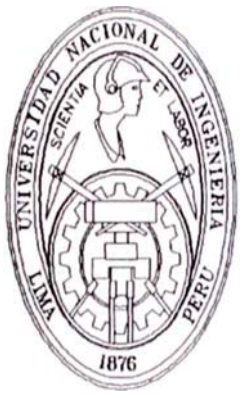
UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA ELECTRICA Y ELECTRONICA



APLICACIÓN DE TURBO CÓDIGOS EN SISTEMAS DE TRANSMISIÓN DE TRAMAS CORTAS

TESIS

PARA OPTAR EL GRADO DE MAESTRO EN CIENCIAS

MENCIÓN: TELECOMUNICACIONES

PRESENTADA POR:

HÉCTOR ARISTIDES TRIGOSO MEDINA

LIMA - PERU

2008

A Giovana, María Victoria, Daniel Victor y David Pablo, mi amada familia

Agradecimiento

Quisiera empezar agradeciendo al Señor Dios Todopoderoso que me dió la capacidad y fortaleza para poder concluir esta tesis. Mi sincero agradecimiento a mis asesores el MSc. Fernando Merchán y al Dr. Carlos Valdéz por su guía y aliento.

Quisiera también expresar mi agradecimiento a mi esposa Giovana y mis hijos, María Victoria, Daniel Victor y David Pablo por su incondicional amor, que fue una inagotable fuente de estímulo para seguir trabajando. A mis padres por haberme alentado y apoyado para poder llegar a finalizar este proyecto. Muchas gracias a todos.

UNIVERSIDAD NACIONAL DE INGENIERIA
FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

APLICACIÓN DE TURBO CÓDIGOS EN SISTEMAS DE TRANSMISIÓN DE TRAMAS CORTAS

TESIS

PARA OPTAR EL GRADO DE MAESTRO EN CIENCIAS
MENCIÓN: TELECOMUNICACIONES

PRESENTADA POR:

HÉCTOR ARISTIDES TRIGOSO MEDINA

LIMA - PERÚ

EXTRACTO

En el año de 1993 un grupo de investigadores franceses asombraron a la comunidad científica con un descubrimiento que marcaría un hito en el mundo de la codificación. Se trataba de los Turbo Códigos, los cuales demostraron un rendimiento muy cercano a los límites predichos por Shannon en su teoría matemática de las comunicaciones. Uno de los puntos fuertes en los que se basa su excelente rendimiento es, entre otros, las largas tramas de bits que se usan para la transmisión. Los resultados originales se obtivieron con una secuencia de 64 Kbits, en los que para una relación señal a ruido (SNR) de 0,7 dB la tasa de error de bit BER era 10^{-5}. Este fue un paso muy importante, tomado con excepticismo por muchos, pero ampliamente demostrado por otros. Se comprobó que a medida que disminuye la longitud de la trama, la eficiencia de este código, es decir, su capacidad para corregir errores, también lo hace |8, 22, 10, 9|. Dada la tendencia actual de buscar más y mejores medios de comunicaciones inalámbricas, como es el caso de los Sistemas de Comunicaciones Personales (PCS) que emplean tramas de 192 y 189 bits, por ejemplo, en los cuales se requiere una buena capacidad de corrección de errores, aún con una alta potencia de ruido presente en el canal, es de gran interés para los investigadores extender el uso de los Turbo códigos para que puedan emplearse en ellos o en otros sistemas de transmisión de tramas cortas |8, 19, 22|. Existen varias propuestas en este sentido, que presentan enfoques diferentes |8, 22, 10, 19, 31, 20|, pero quedando

todavía camino por recorrer para mejorar su rendimiento. Por lo que en esta tesis se estudiarán los principales aspectos que inciden en el rendimiento de los Turbo Códigos y se propondrá una alternativa de codificación eficiente para Sistemas de Comunicaciones de tramas cortas, empleando estos códigos.

UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

APPLICATION OF TURBO CODES IN SHORT FRAMES TRANSMISSION SYSTEMS

THESIS

REQUIREMENT FOR THE DEGREE OF MASTER IN SCIENCE
FIELD: TELECOMMUNICATIONS

BY:

HÉCTOR ARISTIDES TRIGOSO MEDINA

LIMA - PERÚ

ABSTRACT

In 1993 a group French researchers astonished the international scientific community with a discovery that become a landmark in the world of coding theory. It was about Turbo Codes, which demonstrated a performance very close to the limits predicted by Shannon in his mathematical theory of communications. One of the strong points in which their excellent performance yields is, among other, the long frames that are used for the transmission. The original results were obtained with a 64 Kbits'length interleaver, and for a signal noise ratio (SNR) of 0,7 dB the bit error rate (BER) obtained was 10^{-5}. This was a very important step, took with scepticism by many, but extensively demonstrated by others. It was demonstrated that as the frame's length decrease, the performance of this code, in other words, its capacity to correct errors, also decrease |8, 22, 10, 9|. Following the actual trend for seeking more an better media for wireless communications, like for example Personal Communication Systems (PCS) which uses 192 and 189 bits frames, a good error correction capacity is required. This is also true with a high noise power present in the channel. That's why it's a great interest for the researchers to extend the use of turbo codes in those Personal Communication Systems or in other short frame transmission systems |8, 19, 22|. There are several approaches in this sense, with different perspectives |8, 22 10, 19, 31, 20|, but remaining way to improve its performance. That's why in this thesis we are going to study the principal components which affect the performance

of the Turbo Codes and we re to propose an efficient coding scheme applied to short frame transmission systems, using these codes.

Contenido

Agradecimiento IV
Resúmen V
Abstract VII
Lista de Figuras XI
Lista de Tablas XIII
Introducción. 1

Capítulo 1.
Sistemas de comunicaciones 3
1.1. Modelos de canal 5
1.2. Capacidad de canal 7
1.3. Medidas de performance 9

Capítulo 2.
Codificación de canal 10
2.1. Códigos de bloque 12
2.1.1. Codificación de Códigos de Bloque 13
2.1.2. Códigos Cíclicos 14
2.1.3. Códigos sistemáticos 16
2.1.4. Códigos BCH y Reed Solomon 19
2.2. Códigos Convolucionales 20
2.2.1. Codificación de Códigos Convolucionales 21
2.2.2. Representación de Códigos convolucionales 23
2.2.3. Códigos Convolucionales Terminados 25
2.2.4. Códigos Convolucionales Puncturados 26
2.2.5. Códigos Convolucionales Sistemáticos Recursivos 27
2.3. Códigos Concatenados 29
2.4. Turbo Códigos 31
2.5. Comparación de los Sistemas de Codificación 33

Capítulo 3.
Fundamentos de turbo códigos 36
3.1. Estructura del Codificador 37
3.2. Factores que afectan la Performance de los Turbo Códigos 39
3.2.1. Códigos Constituyentes 39
3.2.2. Entrelazador 40

3.2.3. Puncturado 41
3.2.4. Terminación de Trellis 41
3.2.5. Algoritmo de Decodificación 43
3.3. Decodificación Iterativa 44
3.3.1. Turbo Decodificación 44
3.3.2. Clases de Algoritmos de Decodificación 45
3.4. Análisis de la Performance de los Turbo Códigos 46
3.4.1. Esquema de simulación 49
3.4.2. Influencia de la longitud de la trama 51
3.4.3. Influencia del número de iteraciones 52
3.4.4. Influencia de la tasa del código 54
3.4.5. Influencia del código generador 55
3.4.6. Influencia del diseño del entrelazador 56
3.4.7. Comparación entre el algoritmo de decodificación SOVA y Log-Map 58

Capítulo 4.
Diseño de TC en tramas cortas **59**
4.1. Diseño del entrelazador 59
4.2. Selección de los códigos constituyentes 63
4.3. Rendimiento del nuevo esquema propuesto para tramas cortas 67

Capítulo 5.
Conclusiones **70**

Capítulo 6.
Trabajos Futuros **73**

Apéndice A.
Simulación del Software **74**

Apéndice B.
Listado de Programas en Matlab **76**

Bibliografía. **93**

Lista de Figuras

1.1. Diagrama de bloques de un sistema de comunicaciones típico 4
1.2. Diagrama de transición de probabilidades en un canal binario simétrico 6
1.3. Modelo de comunicaciones donde el canal, el modulador y el demodulador son reemplazados por un canal de tiempo discreto 7
2.1. Codificador cíclico para $g(p) = p^2 + p + 1$ 15
2.2. Codificador cíclico sistemático para $g(p) = p^2 + p + 1$ 19
2.3. Codificador convolucional de tasa $r = 1/2$ con longitud de contención $K_c = 3$ 21
2.4. Diagrama de Estado para el código convolucional del ejemplo 24
2.5. Diagrama de Trellis para el código convolucional del ejemplo 25
2.6. Ejemplo de un codificador convolucional sistemático recursivo (CSR) de tasa $1/2$ y longitud de conteción $K_c = 3$ 27
2.7. Diagrama de Estado para el código CSR del ejemplo 28
2.8. Diagrama de Trellis para el código CSR del ejemplo 29
2.9. Código concatenado serial 30
2.10. Codificador para Turbo Códigos 32
2.11. Decodificador para Turbo Códigos 32
2.12. E_b/N_0 vs tasa de código para varios sistemas de codificación de corrección de errores, usando modulación BPSK en un canal AWGN . . 34
3.1. Codificador para el Turbo Código del ejemplo 38
3.2. Turbo codificador con terminación de Trellis forzada 43
3.3. Performance asintótica de un turbo código de tasa $r = 1/2$, longitud de contención $K_c = 15$ y de un código convolucional de los mismos parámetros 47
3.4. Bits errados por trama transmitida en función de la relación señal a ruido. Código de trama de 256 bits, códigos generadores $(13, 17)_8$, algoritmo Log-MAP 49
3.5. Performance asintótica y simulada de un turbo código de tasa $r = 1/2$ y longitud de contención $K_c = 5$ con varias medidas de entrelazadores 50
3.6. Performance del turbo código original en función del número de iteraciones 51
3.7. Performance en función del número de iteraciones. Código de trama de 20,049 bits, códigos generadores $(7, 5)_8$, algoritmo Log-MAP y relación señal a ruido de 1,5 db 53
3.8. Performance en función del número de iteraciones, código de trama de 400 bits 54
3.9. Performance en función de la tasa del código, código de trama de 400 bits 55

3.10. Performance en función de la longitud de contención, código de trama de 400 bits 56
3.11. Performance en función del entrelazador, código de trama de 400 bits 57
3.12. Performance en función del algoritmo de decodificación, código de trama de 400 bits 58
4.1. Performance en función del entrelazador, trama de 256 bits, con polinomios generadores $(15, 17)_8$ 60
4.2. Diagrama de Flujo del algoritmo IDS para diseño de un entrelazador 62
4.3. Performance simulada de turbo códigos de tasa $=1/3$ y longitud de trama de 4096 bits, con entrelazador aleatorio, que emplea polinomios generadores con diversos periodos de realimentación L, pero la misma longitud de contención $K_c = 5$ 64
4.4. Performance simulada de turbo códigos de tasa $=1/3$ con longitud de contención 4.4a $K_c = 4$ y 4.4b $K_c = 5$. La longitud del entrelazador es de 500 bits y cada entrelazador ha sido diseñado con el algoritmo IDS. 65
4.5. Performance simulada de turbo códigos de tasa $=1/3$ con entrelazadores de 500 bits y longitud de contención $K_c = 4$ y $K_c = 5$, con polinomios de realimentación primitivos. 4.5a Entrelazador Golden, 4.5b entrelazador aleatorio y 4.5c entrelazador IDS. 66
4.6. Modelo de esquema de codificación, que emplea códigos generadores $(15, 17)_8$, tasa $=1/3$, y un entrelazador especialmente diseñado con el algoritmo IDS 67
4.7. Performance simulada de turbo códigos de longitud de trama de 500 bits, con códigos generadores $(15, 17)_8$, analizada en función del algoritmo de codificación y de la tasa del código 69

Lista de Tablas

2.1. Tabla de síndromes para el código (7,4) 18

Introducción

La teoría de los códigos de canal o códigos de corrección de errores, que surgió a consecuencia del trabajo pionero de Shannon en su teoría matemática de la Información [30], ha dado como fruto una gran variedad de códigos que se encuentran presentes, desde hace más de cincuenta años, en casi todos los sistemas de transmisión y almacenamiento de información digital. Los turbo códigos pertenecen a este grupo, y han despertado un gran interés ya que su performance se acerca, de una manera nunca antes pensada posible, a los límites teóricos predichos por Shannon. Este, en [30] establece que se puede lograr una probabilidad de error tan pequeña como se desee, dada una tasa de transmisión (cociente entre el número de bits de entrada y el número de bits codificados) menor que la capacidad del canal, que es la medida de la información que puede ser transportada por dicho canal.

De esta misma teoría se desprende que la forma de aproximarse a éste límite es con códigos aleatorios y de gran longitud de bloque. Sin embargo, a medida que la longitud de bloque se hace mayor, la complejidad del decodificador óptimo se vuelve inmanejable. Es por eso que el reto constante de los teóricos de la codificación, ha sido diseñar códigos que se desempeñen próximos al límite de capacidad y al mismo tiempo tengan una modesta complejidad de codificación y decodificación. Es así como en 1993 se descubren los turbo códigos por un grupo de investigadores franceses [6], que asombraron a la comunidad ciéntifica internacional por haber logrado una probabilidad de error de bit (BER) de 10^{-5}, con tan solo una relación señal a ruido (SNR) de 0.7 dB. Este rendimiento fue obtenido con una longitud de bloque de 64 Kbs, una tasa de código $r = 1/2$ y un algoritmo de decodificación iterativo sub-óptimo.

Si embargo, se comprobó que estos resultados sólo eran posibles con longitudes de tramas de varios miles de bits, como la que se mencionó en el párrafo anterior. A medida que disminuye la longitud de la trama, la eficiencia de este código, es decir, su capacidad para corregir errores, también lo hace [8, 22, 10, 9].

No se puede negar el auge de las comunicaciones inalámbricas y la aparición de nuevos estándares para los sistemas de comunicaciones personales. Ya es una realidad la telefonía celular de tercera generación y está muy próxima a la aparición de la cuarta generación. Así que, como se ve, existe un gran interés en buscar más y mejores medios para las comunicaciones inalámbricas. Las personas quieren estar

"conectadas" en todo tiempo y en todo lugar. Dado que estos sistemas utilizan tramas cortas, en promedio menores a 1,000 bits, existe un gran interés por encontrar códigos robustos que tengan una buena capacidad de corrección de errores, aún con una alta potencia de ruido presente en el canal.

Dadas las capacidades de los turbo códigos, ampliamente demostradas actualmente a pesar de haber surgido como un descubrimirnto intuitivo, es de gran interés para los investigadores extender su uso para poder emplearlos en los sistemas de comunicaciones personales (PCS), o en sistemas de transmisión de datos en las bandas de V/UHF, o en otros sistemas que también empleen tramas cortas [8, 19, 22]. Existen varias propuestas en este sentido, con enfoques diversos [8, 22, 10, 19, 31, 20]. Pero sin embargo, todavía se puede mejorar su rendimiento, a medida que se entiendan mejor los parámetros y las propiedades que influyen en el, quedando todavía camino por recorrer.

Es así, como a través de un análisis de los diversos factores que influyen en el desempeño de éstos códigos, llevados al terreno de las tramas cortas, es decir empleando longitudes de trama menores a los 1,000 bits, se ha podido determinar en ésta tésis, la mejor configuración para potenciar su rendimiento.

Asimismo, se han analizado dos parámetros fundamentales que influyen directamente en la elección de los entrelazadores y los códigos constituyentes, éstos últimos, considerados los componentes principales de los turbo códigos. Los parámetros analizados son el espectro de distancia y la performance de la decodificación iterativa, los cuales se estudian en forma particular, ya que su tratamiento es diferente en cada uno de los componentes principales mencionados.

Con ésta información analizada y comprendida, hasta cierto punto, ya que todavía quedan algunas particularidades del desempeño de los turbo códigos por entender, se ha podido recomendar un diseño eficiente de éstos códigos para sistemas de transmisión de tramas cortas. Este diseño se ha obtenido en base a los resultados de las comparaciones de desempeño de sus diversos componentes. Sin embargo, todavía queda camino por recorrer para mejorar su desempeño y adaptarlos a los estándares de los sistemas de comunicaciones de última generación.

Capítulo 1

Sistemas de comunicaciones

Un sistema de comunicaciones tiene la finalidad de transportar información de una parte (Fuente) a otra (Destino). La mayoría de los sistemas de comunicaciones modernos operan en el dominio digital, es decir, emplean una secuencia de símbolos de un alfabeto finito para representar la información. La transmisión de datos en formato digital ofrece la posibilidad de emplear una gran variedad de técnicas de procesamiento de señal, que de otra forma no estarían disponibles, incluyendo, por supuesto, codificación de corrección de errores. Un modelo de un sistema de comunicaciones es el que se muestra en la Figura 1.1, cuyos bloques constitutivos son descritos brevemente.

La *fuente de información* genera mensajes que deben ser transmitidos al receptor. Estos mensajes pueden ser analógicos o digitales, dependiendo del tipo de fuente. Por ejemplo, la voz transmitida durante una conversación telefónica es representada por una señal de tiempo y amplitud continuos generada por un micrófono, que junto con el parlante constituye una fuente analógica. En un sistema de comunicaciones digitales, los mensajes emitidos por una fuente analógica son convertidos a una señal digital, usualmente representada por secuencias de dígitos binarios o bits. Esta operación es realizada por el *codificador de fuente*, el cual busca representar el mensaje de entrada con tan pocos bits como sea posible. Si la fuente de información es digital, la codificación de fuente no requiere una conversión analógica a digital (A/D). Sin embargo, la tarea de representar el mensaje con tan pocos bits como sea posible permanece. Este proceso es llamado compresión de datos, proceso durante el cual se reduce la redundancia presente en los mensajes de la fuente, o en forma ideal se prescinde completamente de ella.

El siguiente bloque en el modelo de comunicaciones es el *codificador de canal*. Mientras que el codificador de fuente es seleccionado con respecto a una fuente de información particular, el codificador de canal se elige generalmente con respecto al canal por el cual los mensajes van a ser transmitidos. El propósito del codificador de canal es darle un formato a los mensajes transmitidos, de tal manera de incrementar su inmunidad al ruido y a la interferencia introducida por el canal. Esto se realiza insertando redundancia controlada en el mensaje a ser transmitido, permitiendo al

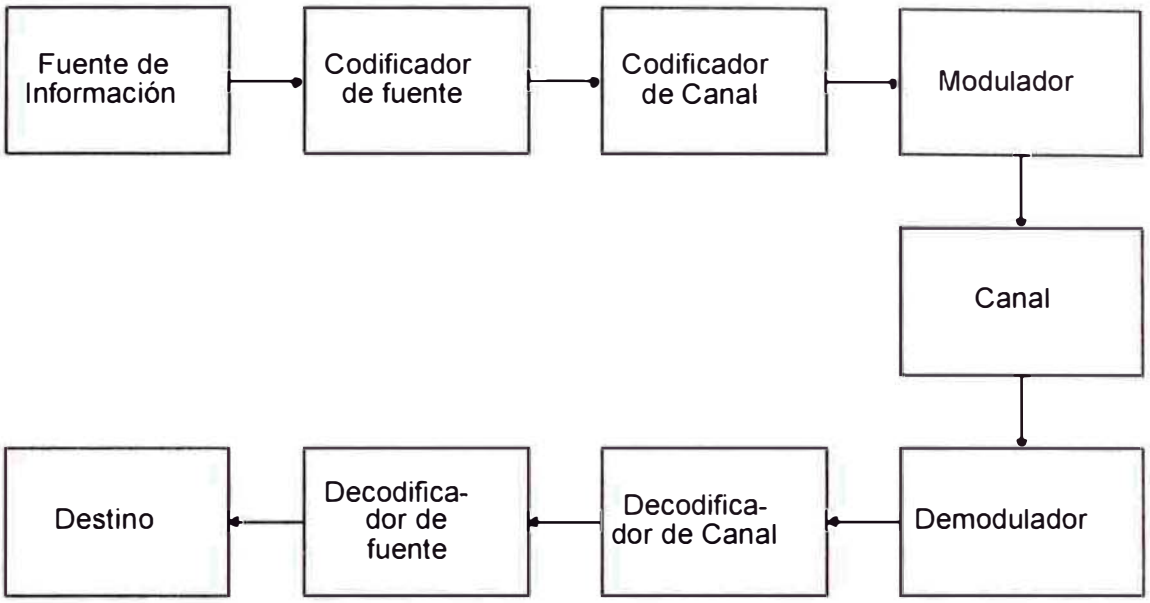


Figura 1.1: Diagrama de bloques de un sistema de comunicaciones típico

receptor detectar y, posiblemente, corregir errores. Los turbo códigos, el tema de esta tesis, pertenecen a esta clase de códigos, que son códigos de canal.

El *canal* sirve como un medio de unión por el cual los mensajes codificados son transmitidos. En muchas situaciones prácticas, por ejemplo en radio comunicaciones, el canal es de ondas. Sin embargo, éste no puede ser empleado para transmitir directamente las secuencias de dígitos binarios. Mas bien las secuencias digitales deben ser convertidas en formas de onda adecuadas a las características específicas del canal, tarea que es efectuada por el *modulador*. Las ondas de salida del modulador son afectadas entonces en una forma que depende de las características del canal. Estas ondas dañadas son la entrada del *demodulador*, cuya función es la inversa del modulador, convertir la onda recibida en una secuencia discreta en el tiempo, idealmente idéntica a la que entró al modulador. Aquí es donde el *decodificador de canal* entra en acción; su papel es contrarrestar las perturbaciones inducidas en el canal, valiéndose de la redundancia introducida por el codificador de canal. La secuencia estimada aqui es transformada en la salida estimada de la fuente en el *decodificador de fuente*, el cual envía este estimado al destino. Cuando la fuente es analógica, se debe efectuar una conversión digital/analógica (D/A). En un sistema bien diseñado, el estimado será una fiel reproducción de la salida de la fuente, excepto cuando el canal sea muy ruidoso.

1.1. Modelos de canal

El canal, como se dijo anteriormente, daña las ondas que viajan por el mismo de una manera que depende de sus características. Este daño es producido de dos maneras:

Definición 1.1.1 (Atenuación) *Causada generalmente por la absorción de energía y las multitrayectorias de la señal en el medio de propagación. En el caso de comunicaciones satelitales la atenuación se manifiesta como una pérdida de potencia en la señal transmitida, como función de la distancia, conocida como el efecto de pérdida de espacio libre. En el caso de comunicaciones móviles, la atenuación no es fija, y depende de la combinación del movimiento y de las numerosas reflexiones de la señal de radio transmitida, causadas por el ambiente de propagación entre la antena transmisora y la receptora. Tales canales de comunicaciones se conocen como canales de desvanecimiento por multitrayectoria.*

Definición 1.1.2 (Ruido) *Puede tomar diferentes formas. Las fuentes más comunes son el calor en el transreceptor y en el medio de propagación. El modelo más comunmente asumido de ruido es el modelo de ruido aditivo blanco gaussiano (AGWN, por sus siglas en inglés "Additive white Gaussian noise")*

Cuando se estudian y se comparan los códigos de canal es conveniente considerar al modulador y al demodulador como parte del canal [25]. Si este canal emplea sólo dos símbolos (0 y 1) y realiza detecciones duras en el receptor, se trata de una canal simétrico binario, como el que se ve en la Figura 1.2. Este es un canal compuesto con entradas y salidas discretas en el tiempo, caracterizado por las posibles entradas, conjunto $X = \{0, 1\}$, las posibles salidas, conjunto $Y = \{0, 1\}$, y las probabilidades de transición que relacionan las posibles salidas con las posibles entradas. Si en la secuencia binaria transmitida, el ruido y otras perturbaciones del canal causan errores estáticamente independientes, con probabilidad promedio p, las probabilidades de transición son

$$\begin{aligned} P(Y = 0|X = 1) &= P(Y = 1|X = 0) = p \\ P(Y = 1|X = 1) &= P(Y = 0|X = 0) = 1 - p \end{aligned} \tag{1.1}$$

En general estas probabilidades de transición pueden variar con el tiempo, y pueden estar también correlacionadas de un tiempo a otro. Además, las salidas del canal no dependen necesariamente de la entrada actual solamente, sino que pueden depender de entradas previas. En tales casos se dice que el canal tiene memoria. Los

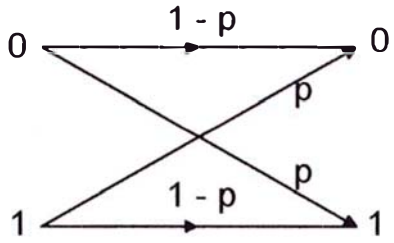


Figura 1.2: Diagrama de transición de probabilidades en un canal binario simétrico

canales encontrados en sistemas de comunicaciones móviles son ejemplos de canales que normalmente deben ser modelados como variantes en el tiempo y con memoria. Estos efectos surgen de la combinación del movimiento y de las numerosas reflexiones de las ondas de radio transmitidas, causadas por el ambiente de propagación entre las antenas transmisora y receptora.

Un modelo que se adapta con exactitud a varios sistemas de comunicaciones y que es el más comunmente usado, entre otras razones, por su simplicidad y su facilidad de análisis en diversos contextos, es el canal de ruido blanco aditivo gaussiano (AWGN). Si bien este modelo no se adapta exactamente al comportamiento de un canal de comunicaciones móviles, si lo hace para los de comunicaciones por satélite y para los terrestres en línea de vista. Es por eso que se adoptará este modelo en el desarrollo de esta tesis, ya que por tratarse de un estudio del funcionamiento de los turbo códigos no es conveniente añadir complicaciones al canal que dificulten su comprensión.

Este modelo AGWN es aditivo ya que la salida del canal Y se obtiene añadiendo a la entrada X la variable aleatoria gaussiana G, así

$$Y = X + G \tag{1.2}$$

donde G es una variable aleatoria gaussiana con media cero y varianza σ^2. Para un simbolo de entrada dado $X = x$, la salida del canal Y es una variable aleatoria gaussiana con media x y varianza σ^2, así

$$p(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-x)^2}{2\sigma^2}}, \tag{1.3}$$

donde $p(.)$ denota una función densidad de probabilidad de una variable aleatoria.

La porción blanca del AGWN se refiere a la densidad espectral del ruido, que se asume tiene el valor constante N_0 entre las frecuencias 0 a $+\infty$. Esto es claramente una imposibilidad física , porque se requeriría que la fuente de ruido tenga energía infinita; pero esta suposición brinda buenos resultados cuando se aplica al análisis de sistemas de comunicaciones de banda limitada.

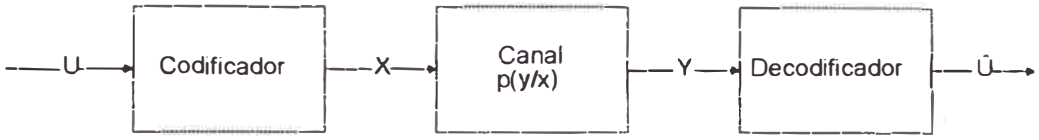


Figura 1.3: Modelo de comunicaciones donde el canal, el modulador y el demodulador son reemplazados por un canal de tiempo discreto

Un modelo de comunicaciones simplificado, donde el modulador, demodulador y canal de ondas es reemplazado por un canal de tiempo discreto se muestra en la Figura 1.3. En este modelo se han logrado las simplificaciones reemplazando la fuente y el codificador de fuente con una secuencia de símbolos binarios. Estos símbolos son variables aleatorias independientes e idénticamente distribuidas, con igual probabilidad de ser 0 ó 1. Esto corresponde a una situación ideal en la que al codificador de fuente se le ha removido toda la redundancia presente en la secuencia de información original. El resultado de estas simplificaciones es un modelo que es frecuentemente usado en estudios y comparaciones de códigos de canal, y es el que se usará en esta tesis.

1.2. Capacidad de canal

En [30] Shannon introdujo el concepto de *"capacidad del canal"* C, que es la máxima velocidad a la cual la información podría ser transmitida confiablemente en un canal de comunicaciones dado. Entiéndase entonces, que la capacidad del canal es una medida de la cantidad de información que puede ser transportada entre la entrada X y la salida Y de un canal. La medida de capacidad esta relacionada con la definición matemática de información. La información mutua promedio entre X e Y, está dada por

$$I(X,Y) = \begin{cases} \sum x \sum y p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} & \text{para un canal discreto,} \\ \int p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} & \text{para un canal continuo.} \end{cases} \tag{1.4}$$

donde $p(x)$ y $p(y)$ son las funciones de densidad de probabilidad marginales de X e Y respectivamente y $p(x,y)$ es la función densidad de probabilidad unida de X e Y. Entonces la capacidad del canal se define como el valor máximo de $I(X,Y)$, maximizado sobre la distribución de entrada $p(x)$, así

$$C = \max_{p(x)} I(X,Y), \tag{1.5}$$

medida en bits por transmisión. La importancia de la capacidad de canal C queda claramente establecida por el "teorema de codificación para canales continuos y ruidosos con limitaciones de potencia promedio". Este establece lo siguiente:

Teorema 1.2.1 (Codificación para canales ruidosos de Shannon) *A cada canal se le puede asociar una capacidad de canal C, de tal manera que existen códigos de control de errores tales que la información puede ser transmitida sobre el canal a velocidades menores que C con una probabilidad de error de bit (BER) arbitrariamente baja.*

La demostración de este teorema está fuera de los alcances de esta tesis. Sin embargo, de ella se desprende que para cualquier velocidad de transmisión r menor o igual a la capacidad del canal C, existe un esquema de codificación de corrección de errores que logra una muy pequeña probabilidad de error, la cual va disminuyendo a medida que la longitud del bloque n se aproxima a infinito. Contrariamente, si r es mayor que C, ningún esquema de codificación puede lograr esta performance. En este teorema queda sentada la existencia de estos códigos, pero no se da, sin embargo, ninguna guía para obtener esquemas de codificación apropiados. La búsqueda de estos esquemas sigue siendo el esfuerzo de estos más de 50 años, de ellos se dará un alcance en el siguiente capítulo.

Para un canal AGWN, la capacidad del canal es,

$$C = \frac{1}{2}\log_2\left(1 + \frac{\sigma_x^2}{\sigma^2}\right) \qquad \text{bits por transmisión,} \tag{1.6}$$

donde la potencia de entrada al canal está limitada por $E[X^2] \leq \sigma_x^2$, y σ^2 es la varianza del ruido. En la práctica, el canal físico de tiempo discreto es de banda limitada. Esto significa, que no es posible incrementar indefinidamente la velocidad de transmisión accesando al canal idefinidamente seguido. Para un canal AWGN limitado a un ancho de banda de W Hz y una densidad espectral de potencia de ruido de dos bandas $N_0/2$, la capacidad es [25]

$$C = W\log_2\left(1 + \frac{P}{WNo}\right) \qquad \text{(bps)}, \tag{1.7}$$

donde P es la potencia transmitida promedio. Asumiendo una perfecta señalización de Nyquist, esta ecuación se puede también expresar como

$$C = W\log_2\left(1 + \frac{E_s}{No}\right) \qquad \text{(bps)} \tag{1.8}$$

donde E_s es la energía promedio de la señal en cada intervalo de señalización de duración $T = 1/W$, $E_s = PT$ ó en forma equivalente $E_s = P/W$. E_s/N_0 es la relación señal a ruido (SNR).

1.3. Medidas de performance

Para un ancho de banda W fijo, C aumenta con un incremento en la potencia de la señal transmitida. Por otra lado si P es fijo, la capacidad se puede incrementar, aumentando el ancho de banda W. Cuando $W \rightarrow \infty$, la capacidad del canal se aproxima a su valor asintótico (capacidad de canal de ancho de banda infinito), hallado como

$$C_{\infty} = \frac{P}{No} \log_2 e. \tag{1.9}$$

La potencia de entrada al canal se puede expresar como $P = E_b r_b$, donde E_b es la energía transmitida por bit de información y r_b es la velocidad de información en bits/s. Combinando el teorema de codificación de canal con la capacidad de canal de ancho de banda infinito se tiene que $r_b < \frac{P}{N_0} \log_2 e$, lo que nos lleva a un requerimiento fundamental para la realización de las comunicaciones:

$$\frac{E_b}{N_0} > \frac{1}{\log_2 e} = \ln 2, \tag{1.10}$$

donde ln es el logaritmo natural. De la Ecuación 1.10 se desprende que no es posible diseñar un sistema de comunicaciones con una probabilidad de error arbitrariamente baja si $E_b/N_0 < -1{,}6$ dB. Por otro lado, mientras que $E_b/N_0 > -1{,}6$ dB, es posible en teoría lograr un probabilidad de error arbitrariamente baja, usando códigos suficientemente largos.

La Ecuación 1.10 se trata de un límite inferior de la relación señal a ruido (SNR) en términos de la relación energía por bit con respecto a la densidad espectral de ruido (E_b/N_0), requerida para lograr una comunicación libre de error. Sin embargo, los supuestos asumidos para derivar este límite son demasiado optimistas para la mayoría de las situaciones prácticas. Primeramente el ancho de banda del canal es limitado. Segundo se permite a la palabra de código transmitida ser infinitamente larga, lo que introduciría un retardo infinito en la transmisión. Como es de suponerse los sistemas de comunicaciones son diseñados para transmitir en un tiempo limitado. Además este límite es derivado asumiendo una entrada continua, sabiendose que los sistemas de comunicaciones digitales emplean esquemas de modulación con alfabetos finitos. Debido a estas suposiciones la (E_b/N_0) requerida es en realidad mayor que $-1{,}6$ dB. Existen otros límites teóricos que dan cotas inferiores más exactas, tomando en consideración algunas de estas suposiciones, como por ejemplo el límite de paquete esférico [28] y el límite esférico tangencial [17].

Capítulo 2

Codificación de canal

Los códigos de corrección de errores o de canal surgen a finales de la década de 1940 con el trabajo pionero de shannon [30], Hamming [14] y Golay [13]. En su Teoría matemática de las comunicaciones Shannon definió los conceptos teóricos de la codificación, basado en rigurosas demostraciones matemáticas que sentaron las bases de lo que hoy se conoce como la Teoría de la Información. En su trabajo dejó por sentado que es posible lograr comunicaciones con una muy pequeña probabilidad de error (tan pequeña como se desee), si se utiliza algún esquema de codificación, con una velocidad de transmisión r menor que la capacidad del canal C. Sin embargo no estableció ningún método práctico para lograr esta capacidad. Desde entonces se ha dado un gran esfuerzo por diseñar estos códigos que operen lo más cercano posible a los límites prodichos por este gran matemático.

Las funciones de codificación y decodificación involucran las operaciones aritméticas de adición y multiplicación modulares realizadas en las palabras de código. Estas operaciones matemáticas se realizan de acuerdo a las convenciones del campo algebraico que tiene como sus elementos los símbolos contenidos en el alfabeto que se va a usar. Aquí algunas definiciones y ejemplos de estas

Definición 2.0.1 (Adición Modular) *Se dice que c es el resultado de la* adición módulo *m de los enteros a y b, lo que se escribe como*

$$a + b \equiv c \text{ módulo } m, \tag{2.1}$$

si c es el resto positivo de la división por m de la suma $a + b$.

Ejemplo 2.0.1 – Adición Modular

$$\begin{aligned} 13 + 18 &\equiv 11 \text{ módulo } 20 & \qquad 3 + 8 &\equiv 11 \text{ módulo } 20 \\ 9 + 5 &\equiv 4 \text{ módulo } 10 & \qquad 2 + 4 &\equiv 0 \text{ módulo } 6 \\ 18 + 27 &\equiv 0 \text{ módulo } 5 & \qquad 15 + 100 &\equiv 50 \text{ módulo } 65 \end{aligned}$$

Definición 2.0.2 (Multiplicación Modular) *Se dice que c es el resultado de la* multiplicación módulo *m de los enteros a y b, lo que se escribe*

$$a.b \equiv c \text{ módulo } m, \tag{2.2}$$

si c es el resto positivo de la división por m de la operación de multiplicación entera a.b.

Ejemplo 2.0.2 – Multiplicación Modular

11,5	≡ 15 módulo 20	6,4	≡ 6 módulo 18
9,7	≡ 13 módulo 50	2,5	≡ 0 módulo 10
4,17	≡ 0 módulo 2	6,5	≡ 0 módulo 6

Normalmente se se tiene familiaridad con el campo de los números reales o el campo de los números complejos, que tienen un número infinito de elementos. Sin embargo, los códigos se construyen en campos que tienen un número finito de elementos. Un campo finito con q elementos es generalmente llamado campo de Galois y se denota por GF(q).

El más simple de los campos de Galois es GF(2). Este puede ser representado por el conjunto de dos elementos $\{0, 1\}$ bajo las operaciones de adición y multiplicación binaria estándares, como se muestra a continuación

+	0	1
0	0	1
1	1	0

.	0	1
0	0	0
1	0	1

que son operaciones módulo 2. El grupo multiplicativo dentro de GF(2) es particularmente simple, consistiendo sólamente del único elemento identidad multiplicativa 1. La tabla de multiplicación incluye el elemento 0 (el cual se encuentra fuera del grupo multiplicativo) debido a que las dos operaciones del campo son distributivas.

En forma similar, el campo GF(3) es un conjunto que consiste de los elementos $\{0, 1, 2\}$. Las tablas de adición y multiplicación para GF(3) son

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

.	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

En general, el campo finito GF(q) solo puede ser construido si q es un número primo o potencia de un primo. Cuando q es primo, la multiplicación y la adición se basan en la aritmética módulo q, ilustrada anteriormente.

Esta breve introducción a las operaciones aritméticas que se realizan con los elementos de las palabras de código, es necesaria para entender las características y el funcionamiento de los códigos de bloque que se verán en la siguiente sección.

El propósito de la codificación de canal o de corrección de errores es hacer los mensajes transmitidos lo menos suceptibles al ruido introducido por el canal. Esto se logra añadiendo redundancia estructurada a la secuencia transmitida, y consecuentemente incrementando el número de bits a transmitir. Existen dos clases básicas de códigos usados para añadir esta redundancia: *códigos de bloque* y *códigos convolucionales*. Es necesario precisar, asimismo, que los turbo códigos comparten muchos de los conceptos y de la terminología empleada por ambos esquemas de codificación, por lo que es necesario realizar una breve descripción de estos antes de entrar al tema de los turbo códigos.

2.1. Códigos de bloque

Al mismo tiempo que Shannon definía los límites teóricos de la comunicación confiable, Hamming [14] y Golay [13] estaban desarrollando los primeros esquemas prácticos de códigos correctores de errores. Su trabajo dió nacimiento a una floreciente rama de la matemática aplicada conocida como Teoría de la Codificación. Normalmente se le atribuye a Richard Hamming haber descubierto el primer código corrector de errores en 1948 [14], el cual lleva hoy su nombre. A éste se sumaron los códigos Golay, Reed-Muller, cíclicos, también conocidos como CRC (códigos de redundancia cíclica), BCH y Reed Solomon.

Un código de bloque C consiste en un conjunto de T vectores de longitud fija $\mathbf{C}_i, i = 0, ..., T-1$, llamados *palabras de código*, cada uno de la forma $\mathbf{C} = (c_0, c_1, \ldots, c_{n-1})$, que toma valores de un campo de Galois de q elementos, denotado por GF(q). Se dice entonces que el código C es q-ario. Cuando el alfabeto consiste de dos elementos, 0 y 1, se trata de un código binario y los elementos de cualquier palabra de código se llaman bits. La longitud de una palabra de código es el número de elementos en el vector y se denota por n. Existen entonces, q^n posibles palabras de código en un bloque de longitud n. De éstas q^n palabras de código se seleccionan $T = q^k$ palabras de código, siendo ($k < n$), para formar el código C.

Si los símbolos en el tren de datos a codificar toman valores de GF(q), entonces la colección de todos los posibles mensajes $\mathbf{X} = (X_0, X_1, \ldots, X_{k-1})$ forman un espacio vectorial sobre GF(q). Se puede ver entonces que, como se mencionó anteriormente, existen q^k posibles vectores de información. Así también la colección de todas las posibles palabras de código forman un espacio vectorial sobre GF(q) que contiene q^n vectores. Existen entonces ($q^n - T$) patrones que no están asociados con bloques

de datos y no son, por tanto, palabras de código válidas. Se puede decir entonces que el código C tiene redundancia si $n > k$. La redundancia R se expresa en forma logarítmica como

$$R = n - log_q T. \tag{2.3}$$

Si $T = q^k$, entonces R está dada simplemente por la diferencia $R = (n - k)$.

2.1.1. Codificación de Códigos de Bloque

El proceso de codificación consiste en partir el tren de datos de entrada en bloques de k símbolos de información llamados *mensajes*, y entonces realizar un mapeo uno–a–uno de cada mensaje

$$\mathbf{X}_m = \begin{bmatrix} x_{m1} & x_{m2} & \cdots & x_{mk} \end{bmatrix}$$

en una palabra de código

$$\mathbf{C}_m = \begin{bmatrix} c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}$$

de n símbolos. Se referirá al código de bloque resultante C, como un código (n, k), y el factor $k/n \equiv r$ se define como la *tasa* del código.

En este estudio se restringirá nuestra atención a códigos lineales, los cuales tienen un proceso de codificación que puede ser descrito por la multiplicación de matrices

$$\mathbf{C}_m = \mathbf{X}_m \mathbf{G}, \tag{2.4}$$

donde la multiplicación de matrices es en el campo GF(q) y $\mathbf{G}$ es la *matriz generadora* $k \times n$, que es de la forma

$$\mathbf{G} = \begin{bmatrix} \leftarrow \mathbf{g}_1 \rightarrow \\ \leftarrow \mathbf{g}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{g}_k \rightarrow \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix}, \tag{2.5}$$

Notese que cualquier palabra de código es simplemente la combinación lineal de los vectores $\{\mathbf{g}_i\}$ de $\mathbf{G}$, así

$$\mathbf{C}_m = x_{m1}\mathbf{g}_1 + x_{m2}\mathbf{g}_2 + \cdots + x_{mk}\mathbf{g}_k \tag{2.6}$$

Los códigos lineales poseen la propiedad que la suma (sobre $GF(q)$) de dos palabras de código es también una palabra de código, y así todos los códigos lineales

de bloque deben contener necesariamente la palabra de código con todos ceros, denominada 0. Sin perder la generalidad, se asume que $\mathbf{C}_0 = 0$.

Definición 2.1.1 (Distancia de Hamming) *Se representa por d_{ij}, donde $i \neq j$, es el número de elementos o posiciones correspondientes en las que difieren dos palabras de código. En el caso binario puede ser hallada tomando la suma* módulo 2 *de dos palabras de código y contando el número de unos en el resultado.*

Definición 2.1.2 (Distancia mínima) *La distancia mínima d_{min} de un código es la menor de las distancias de Hamming entre cualesquiera dos palabras de código*

$$d_{min} = \min_{i \neq j} d_{ij}. \tag{2.7}$$

Un código con distancia mínima d_{min} es capaz de corregir todas las palabras de código con t o menos errores, donde

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor, \tag{2.8}$$

donde $\lfloor . \rfloor$ denota el operador floor[1].

Definición 2.1.3 (Peso de Hamming) *De una palabra de código, w_i, es la distancia de Hamming entre ella misma y la palabra de código todos ceros*

$$w_i = d_{i0}. \tag{2.9}$$

El peso de Hamming se puede determinar simplemente contando el número de elementos no cero en una palabra de código. Para códigos lineales, la distancia mínima es el menor peso de Hamming de todas las palabras de código, excepto la palabra de código todos ceros

$$d_{min} = \min_{i>0} \{w_i\}. \tag{2.10}$$

2.1.2. Códigos Cíclicos

Un *código cíclico* es un subconjunto de la clase de los códigos lineales, en el que cada corrimiento cíclico de una palabra de código produce una palabra de código válida. Esto es, C es cíclico si para cada palabra de código $\mathbf{C} = (c_{n-1}, c_{n-2}, \cdots, c_1, c_0) \in C$ existe también una palabra de código $\mathbf{C}' = (c_{n-2}, c_{n-3}, \cdots, c_0, c_{n-1}) \in C$. Esto puede

[1]Este operador es usado para tomar el valor del resultado y los menores

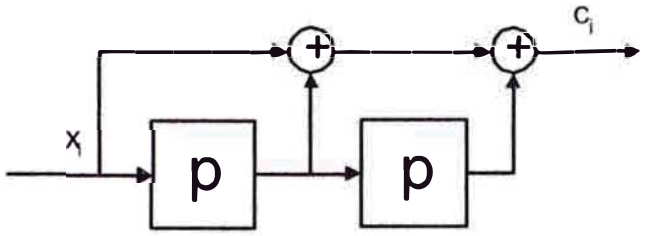


Figura 2.1: Codificador cíclico para $g(p) = p^2 + p + 1$

ser generalizado en que cada corrimiento cíclico de **C**, es una palabra de código. La matriz generadora de un código cíclico puede ser expresada de la forma

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k} & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_{n-k} \end{bmatrix}. \tag{2.11}$$

Además de su representación matricial un código cíclico puede ser especificado en forma compacta por polinomios. La palabra de código $\mathbf{C} = [c_{n-1}c_{n-2}\ldots c_1c_0]$ puede ser expresada como un *código polinómico* $C(p)$ de grado $\leqslant n-1$, de la forma $C(p) = cn-1p^{n-1} + c_{n-2}p^{n-2} + \ldots + c_1p + c_0$. El *polinomio generador* de un código cíclico (n,k) es $g(p) = p^{n-k} + g_{n-k-1}p^{n-k-1} + \ldots + g_1p + 1$. Asimismo, el mensaje puede ser expresado como un *mensaje polinómico* $X(p) = x_{k-1}p^{k-1} + x_{k-2}p^{k-2} + \ldots + x_1p + x_0$, donde $[x_{k-1}x_{k-2}\ldots x_1x_0]$ representan los k bits de información. La operación de codificación viene a ser entonces la multiplicación polinómica $C(p) = X(p)g(p)$. La multiplicación de polinomios es equivalente a la convolución de los coeficientes del polinomio, y entonces la operación de codificación puede ser expresada como

$$c_i = \sum_{j=0}^{n-k} x_{i-j}g_j. \tag{2.12}$$

La convolución de secuencias de valor discreto puede ser implementada por una red lineal de registros de desplazamiento, y así el codificador para códigos cíclicos es extremadamente simple. Un ejemplo de un codificador cíclico que usa registros de desplazamiento lineales se muestra en la Figura 2.1 para el polinomio generador $g(p) = p^2 + p + 1$.

2.1.3. Códigos sistemáticos

Se dice que un código es *sistemático* si el mensaje $\mathbf{X}_m$ está contenido en la palabra de código $\mathbf{C}_m$. La matriz generadora de un código sistemático (n, k) puede ser particionada como

$$\mathbf{G} = \left[\mathbf{I}_k \,\middle|\, \mathbf{P} \right] = \left[\begin{array}{ccccc|cccc} 1 & 0 & 0 & \cdots & 0 & p_{11} & p_{12} & \cdots & p_{1n-k} \\ 0 & 1 & 0 & \cdots & 0 & p_{21} & p_{22} & \cdots & p_{2n-k} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & p_{k1} & p_{k2} & \cdots & p_{kn-k} \end{array}\right], \tag{2.13}$$

donde $\mathbf{I}_k$ es la matriz identidad $k \times k$ y $\mathbf{P}$ es una matriz $k \times (n-k)$. La *matriz comprobadora de paridad* asociada con un código sistemático es

$$\mathbf{H} = \left[\mathbf{P}^T \,\middle|\, \mathbf{I_{n-k}} \right], \tag{2.14}$$

donde $\mathbf{P}^T$ es la traspuesta de $\mathbf{P}$. La matriz comprobadora de paridad brinda una manera eficiente de computar la distancia mínima, que es el menor número de columnas en $\mathbf{H}$ que suman cero.

Dado que cualquier palabra de código del código (n, k) es ortogonal a cada fila de la matriz $\mathbf{H}$ [25], se tiene

$$\mathbf{C}_m\mathbf{H}^T = \mathbf{0}, \tag{2.15}$$

donde $\mathbf{0}$ es un vector fila con $n-k$ elementos todos ceros, y $\mathbf{C}_m$ es una palabra de código del código (n, k). Ya que la Ecuación 2.15 se cumple para cada palabra de código del código (n, k), también se tiene que

$$\mathbf{G}\mathbf{H}^T = \mathbf{0}, \tag{2.16}$$

donde $\mathbf{0}$ es ahora una matriz $k \times (n-k)$ con todos sus elementos ceros.

Aprovechando la propiedad de la Ecuación 2.15, y definiendo a la palabra de código recibida a la salida del demodulador como $\mathbf{Y}$, se puede afirmar que para tener una comunicación sin error se debe cumplir que $\mathbf{Y}\mathbf{H}^T = \mathbf{0}$. Pero $\mathbf{Y}$ dificilmente es igual a $\mathbf{C}_m$, ya que el canal afecta a la palabra de código, añadiéndole, un vector de error binario $\mathbf{e}$. Entonces $\mathbf{Y}$ puede ser expresado como

$$\mathbf{Y} = \mathbf{C}_m + \mathbf{e} \tag{2.17}$$

Es así como el producto $\mathbf{Y}\mathbf{H}^T$ nos lleva a

$$\begin{aligned} \mathbf{Y}\mathbf{H}^T &= (\mathbf{C}m + \mathbf{e})\mathbf{H}^T \\ &= \mathbf{C}m\mathbf{H}^T + \mathbf{e}\mathbf{H}^T \\ &= \mathbf{e}\mathbf{H}^T = \mathbf{S} \end{aligned} \tag{2.18}$$

donde $\mathbf{S}$ es un vector de dimensión $(n-k)$ que se conoce como el síndrome del patrón de error. Los componentes de $\mathbf{S}$ son cero cuando los elementos de $\mathbf{Y}$ corresponden a los de una palabra de código válida, y son no cero cuando no corresponden. Es posible, entonces asociar a cada síndrome un patrón de error, el más probable, de tal manera que la decodificación consiste en, una vez calculado el síndrome $\mathbf{S}$, verificar el patrón de error $\hat{\mathbf{e}}$ asociado más aparente, y restarlo de la palabra recibida $\mathbf{Y}$. Dado que con códigos binarios la sustracción módulo 2 es idéntica a la adición módulo 2, esta operación se puede expresar como

$$\hat{\mathbf{C}}_m = \mathbf{Y} \oplus \hat{\mathbf{e}}_m \tag{2.19}$$

El siguiente es un ejemplo de codificación y decodificación de un código lineal sistemático de bloque (7,4).

Ejemplo 2.1.1 – Código de bloque (7,4)

$$\mathbf{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] = \left[\begin{array}{c|c} \mathbf{I}_4 & \mathbf{P} \end{array}\right] \tag{2.20}$$

Una palabra de código típica se expresaría de la forma

$$\mathbf{C}_m = [\, x_{m1} \quad x_{m2} \quad x_{m3} \quad x_{m4} \quad c_{m5} \quad c_{m6} \quad c_{m7} \,]$$

donde los $\{x_{mj}\}$ representan los cuatro bits de información y los $\{c_{mj}\}$ representan los tres bits comprobadores de paridad, dados por

$$\begin{aligned} c_{m5} &= x_{m1} + x_{m2} + x_{m3} \\ c_{m6} &= x_{m2} + x_{m3} + x_{m4} \\ c_{m7} &= x_{m1} + x_{m2} + x_{m4} \end{aligned} \tag{2.21}$$

De acuerdo a la Ecuación 2.14 se tiene para este código sistemático (7,4), que la matriz $\mathbf{H}$ es de la forma

$$\mathbf{H} = \left[\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{2.22}$$

El producto $\mathbf{C}_m\mathbf{H}^T$ lleva a estas tres ecuaciones

$$\begin{aligned} x_{m1} + x_{m2} + x_{m3} + c_{m5} &= 0 \\ x_{m2} + x_{m3} + x_{m4} + c_{m6} &= 0 \\ x_{m1} + x_{m2} + x_{m4} + c_{m7} &= 0 \end{aligned} \tag{2.23}$$

Empleando la Ecuación 2.18 se construye la tabla de síndromes para el código (7,4), como se muestra a continuación

Síndrome	Patrón de error(**e**)
0 0 0	0 0 0 0 0 0 0
0 0 1	0 0 0 0 0 0 1
0 1 0	0 0 0 0 0 1 0
1 0 0	0 0 0 0 1 0 0
0 1 1	0 0 0 1 0 0 0
1 1 0	0 0 1 0 0 0 0
1 1 1	0 1 0 0 0 0 0
1 0 1	1 0 0 0 0 0 0

Cuadro 2.1: Tabla de síndromes para el código (7,4)

Suponiendo que se recibe la palabra $\mathbf{Y} = [\ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ ]$, se desea saber si existe algún error, y de ser así, corregirlo.

El procedimiento para la decodificación se inicia hallando el síndrome $\mathbf{S}$ para la palabra recibida. De la Ecuación 2.18 se tiene que el síndrome es $\mathbf{S} = [\ 0\ \ 0\ \ 1\ ]$ y de la Tabla 2.1 se obtiene que el patrón de error que le corresponde a esta palabra es $\hat{\mathbf{e}} = [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ ]$. Entoces aplicando la Ecuación 2.19 para obtener la palabra de código, se tiene que

$$\begin{aligned} \hat{\mathbf{C}}_m &= [\ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ ] + [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ ] \\ &= [\ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ ] \end{aligned}$$

El Ejemplo 2.1.1 corresponde a un código Hamming[2], que tiene una $d_{min} = 3$ (esto se puede comprobar contando el número de columnas en $\mathbf{H}$ que suman cero), por lo que según la Ecuación 2.8, puede corregir un sólo error.

Cualquier código puede ser hecho sistemático sin modificar su distancia mínima por medio de operaciones de filas en $\mathbf{G}$. Los códigos cíclicos pueden hacerse sistemáticos multiplicando el mensaje polinómico $X(p)$ por p_{n-k} y calculando el resto

[2]Estos códigos llevan el nombre de su descubridor y tienen la propiedad que $(n, k) = (2^m - 1, 2^m - 1 - m)$, donde m es cualquier entero positivo. En este caso $m = 3$

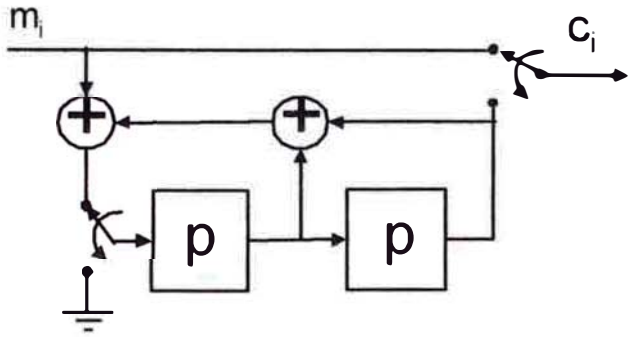


Figura 2.2: Codificador cíclico sistemático para $g(p) = p^2 + p + 1$

$r(p)$ de la división polinómica $p^{n-k}X(p)/g(p)$. La operación de codificación se vuelve entonces $C(p) = p^{n-k}X(p) - r(p)$ [35]. El resto de la división polinómica puede encontrarse empleando una red de registros de desplazamiento con realimentación, tal como el mostrado en la Figura 2.2. Los dos interruptores se encuentran en la posición arriba para las k primeras salidas $c_0, c_1, \cdots, c_{k-1}$, llamados los bits *sistemáticos*. Los interruptores se encuentran en la posición baja para los últimos $n - k$ salidas $c_k, \cdots, c_{n-1}$, llamados los bits de *paridad*.

2.1.4. Códigos BCH y Reed Solomon

Una importante subclase de los códigos cíclicos fue descubierta casi simultáneamente por Hoequenghem en 1959 [16] y por el equipo de Bose y Ray-Chaudhuri en 1960 [5, 4]. Estos códigos son conocidos como códigos BCH, por las iniciales de sus descubridores, y tienen una longitud $n = qm - 1$, donde m es un parámetro de diseño de valor entero. El número de errores que el código BCH binario ($q = 2$) puede corregir esta acotado por $t < (2m - 1)/2$. Los códigos BCH fueron extendidos para el caso no binario ($q > 2$) por Reed y Solomon en 1960 [27]. Los códigos Reed Solomon (RS) constituyeron un gran avance, ya que por su naturaleza no binaria permiten la protección contra ráfagas de errores[3]. Sin embargo, no fue hasta que Berlekamp introdujo un eficiente algoritmo de decodificación en 1967 que los códigos RS comenzaron a encontrar aplicaciones prácticas. Desde entonces los códigos RS han encontrado varias aplicaciones en sistemas tales como Reproductoras de Discos Compactos (CD), reproductoras de Discos Versátiles Digitales (DVD) y el estándar CDPD (Cellular Digital Packet Data).

[3]Un error en ráfaga es aquel que afecta a un grupo de bits contiguos en una trama.

2.2. Códigos Convolucionales

En el proceso de codificación, los códigos de bloque segmentan la información en bloques de longitud k y los mapean en palabras de código de longitud n. Ambas longitudes k y n son fijas. Esto hace que la palabra de código deba ser recibida completamente para poder decodificarla. Esto puede producir retardos excesivamente grandes cuando la longitud de bloque es larga, lo que resulta intolerable para algunos sistemas. Además, los códigos de bloque requieren una sincronización de marco muy precisa[4]. Otro inconveniente es que la mayoría de los decodificadores algebraicos para códigos de bloque trabajan con decisiones "duras", en vez de salidas no cuantificadas o "suaves" del demodulador. Con decodificación de "decisión dura", típica para códigos de bloque, la salida del canal se toma como binaria, mientras que con decodificación de "decisión suave" la salida del canal es un valor continuo. Sin embargo, para lograr las cotas de performance predichas por Shannon se requiere que la salida del canal sea de valor continuo, es decir, "suave"[5].

Los inconvenientes encontrados en los códigos de bloque pueden ser eliminados siguiendo un camino diferente en la codificación. Este camino fue trazado por Elias en 1955 [11], con la introducción de los *códigos convolucionales*. En lugar de segmentar los datos en distintos bloques, los codificadores convolucionales añaden redundancia a un tren continuo de bits de entrada usando un registro de desplazamiento lineal.

Cada conjunto de n bits de salida es una combinación lineal del conjunto de k bits de entrada y de los m bits que se encuentran en el registro de desplazamiento. El número total de bits de los que cada salida depende se conoce como la *longitud de contención* y se denota como K_c. *La tasa del codificador convolucional* r es el número de bits de datos k tomados por el codificador en un intervalo de codificación, dividido por el número de bits de código n de salida durante el mismo intervalo. Así como los datos son continuamente codificados, pueden ser también continuamente decodificados con sólo la latencia nominal. Además los algoritmos de decodificación pueden hacer uso de la información de decisión suave del demodulador.

El primer algorítmo práctico de decodificación fue el decodificador secuencial de Wozencraft y Reiffen en 1961, el cual fue luego modificado en 1963 por Fano y por Jelinek en 1969. Mientras que el trabajo de Fano y Jelinek representaba un avance en el algoritmo de decodificación, no fue sino hasta la introducción del algo-

[4]La sincronización de marco implica que el decodificador sabe cual es el primer símbolo en un palabra de código o marco recibido

[5]Usando "decisiones suaves", en lugar de "duras", la eficiencia en potencia puede ser mejorada en 2.5 dB para códigos de tasa 1/2

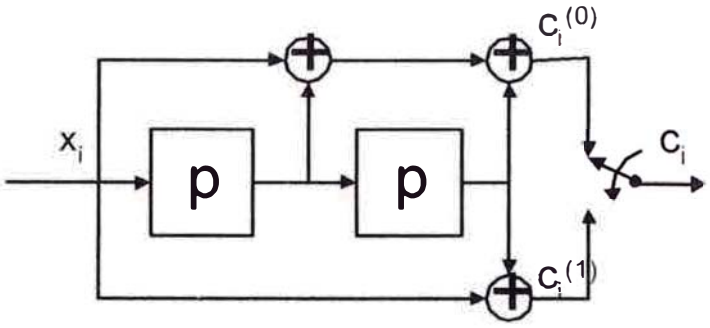


Figura 2.3: Codificador convolucional de tasa $r = 1/2$ con longitud de contención $K_c = 3$

ritmo de Viterbi en 1967 que una solución óptima se hizo realidad (en un sentido de maximum likelihood). Después del desarrollo del Algoritmo de Viterbi, la codificación convolucional comenzó a experimentar una extensiva aplicación en los sistemas de comunicaciones. La longitud de contención $K_c = 7$ (Códigos convolucionales de Odenwalder), que opera a tasas de $r = 1/3$ y $r = 1/2$, se ha vuelto un estándar para aplicaciones de comunicaciones en satélites comerciales. Los Códigos convolucionales fueron usados para muchas sondas espaciales como el Voyager y el Pioneer. Todos los estándares de celulares digitales de segunda generación incorporan codificación convolucional; GSM emplea un código convolucional con $K_c = 5$, $r = 1/2$, USDC emplea un código convolucional con $K_c = 6$, $r = 1/2$, y IS-95 emplea un código convolucional con $K_c = 9$, $r = 1/2$ para la señal de bajada y $r = 1/3$ para la de subida. Globalstar también usa un código convolucional con $K_c = 9$, $r = 1/2$, mientras que Iridium emplea un código convolucional con $K_c = 7$, $r = 3/4$. Además todos los estándares de tercera generación incorporan codificación convolucional para algunos modos de operación.

2.2.1. Codificación de Códigos Convolucionales

Para tener una idea de como se realiza el proceso de codificación y definir algunos conceptos básicos de los códigos convolucionales vease el siguiente ejemplo

Ejemplo 2.2.1 – Código convolucional (2,1,3)

En la Figura 2.3 se muestra un codificador convolucional lineal típico de *tasa* $k/n = 1/2$ (salen dos bits del codificador por cada bit de entrada) y de *longitud de contención* $K_c = 3$. La *longitud de contención* K_c del codificador se define más común-

mente como el máximo número de bits en un solo tren de salida que puede ser afectados por cualquier bit de entrada

$$K_c = 1 + \max_s M_s, \tag{2.24}$$

donde M_s es la longitud del registro de desplazamiento asociado con la entrada s. Nótese que este codificador consiste de $m = 2$ elementos de memoria en el registro de desplazamiento, con $n = 2$ sumadores módulo 2 y un multiplexor para serializar las salidas del codificador. Este código es no recursivo[6] y no sistemático, y puede ser descrito por vectores conocidos como secuencias generadoras, o simplemente generadores. La secuencia generadora es la respuesta impulso de la respectiva salida del codificador al aplicarsele un 1 seguido de un tren de 0's. Para este caso son

$$\begin{aligned} g^0 &= [\ 1 \quad 1 \quad 1\ ] \\ g^1 &= [\ 1 \quad 0 \quad 1\ ] \end{aligned} \tag{2.25}$$

donde cada 1 en la iésima posición del vector indica que el correspondiente estado en el registro de desplazamiento está conectado al sumador módulo 2 y un 0 indica que no existe conexión. También se pueden representar estos vectores en forma polinómica, como $g^{(0)} = p^2 + p + 1$ y $g^{(1)} = p^2 + 1$, o en forma octal $(7,5)_8$.

Suponiendo que se desea codificar la secuencia $\mathbf{x} = [\ 1 \quad 0 \quad 1\ ]$. Inicialmente se asume que el registro de desplazamiento está en el estado todos ceros. Luego del ingreso del primer bit (1), la salida es 11. Tras el ingreso del segundo bit (0), la secuencia de salida es 10. Finalmente para el tercer bit (1), la salida correspondiente es 00. Por lo tanto la secuencia de salida es $\mathbf{c} = [\ 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0\ ]$.

En forma general el proceso de codificación es el siguiente: el mensaje de entrada $\mathbf{x}$ es primeramente dividido en k entradas $\mathbf{x}^{(p)} = (x_p, x_{p+k}, x_{p+2k}, \cdots)$, donde $p \in (0, \cdots, k-1)$. Las k entradas pasan a través de un codificador convolucional, que produce n salidas, $\mathbf{c}^{(q)}$, $q \in (0, \cdots, n-1)$. Con cada nuevo valor de ingreso al registro de desplazamiento, los valores de los elementos de memoria son corridos una posición y sumados en un sumador módulo 2 de acuerdo en un forma que es determinada por la secuencia generadora. Las salidas son multiplexadas para formar la palabra de código $\mathbf{c} = (c_0^{(0)}, \cdots, c_0^{(n-1)}, c_1^{(0)}, \cdots, c_1^{(n-1)}, \cdots)$.

Mientras que los códigos cíclicos pueden ser especificados completamente por un simple polinomio generador $g(p)$, los códigos convolucionales deben ser especificados por un conjunto de $n \times k$ polinomios generadores $g^{(s,g)}(p)$. El codificador tiene

[6]No recursivo significa que sus generadores no tienen realimentación. Este concepto será explicado con mayor detalle en la subsección 2.2.5

k registros de desplazamiento, uno para cada salida. La longitud del registro de desplazamiento asociado con la entrada s se denota como M_s, y la *memoria total* en el codificador es

$$M_c = \sum_{s=0}^{k-1} m_s. \tag{2.26}$$

La codificación procede primero convolucionando los coeficientes de cada generador polinómico con la entrada correspondiente:

$$v_i^{(s,q)} = \sum_{j=0}^{M_s} x_{i-j}^{(s)} g_j^{(s,g)}. \tag{2.27}$$

El tren de salida $c^{(q)}$ es entonces formado por

$$\mathbf{c}^{(q)} = \sum_{s=0}^{k-1} \mathbf{c}^{(s,g)}. \tag{2.28}$$

Este estudio estará abocado a los códigos convolucionales con solo un tren de bits de entrada $k = 1$ y tasa $r = 1/n$. Cuando se consideran códigos convolucionales de tasa $1/n$, el superíndice s del polinomio generador $g^{(s,q)}$ es siempre cero y usualmente se omite.

2.2.2. Representación de Códigos convolucionales

Una forma de ver los codificadores convolucionales es como máquinas de estado finito, representados por diagramas de estado, gráficos y diagramas de trellis. La representación de los codificadores convolucionales como máquinas finitas es esencial para comprender los algoritmos de decodificación que se verán más adelante.

El *estado* de un codificador convolucional es determinado por el contenido de sus registros de desplazamiento. Para un codificador convolucional de tasa $r = 1/n$ con memoria M_c, existen 2^{M_c} estados internos posibles. Una vez que el codificador recibe un nuevo bit de entrada, realiza una transición a uno de dos posibles estados, dependiendo si el bit de entrada es un 1 o un 0. Cuando el codificador realiza una transición de un estado a otro, se obtiene como salida una secuencia particular de n bits.

La operación de un codificador convolucional puede ser especificada sin ambigüedad por un *diagrama de estado*. Los diagramas de estado consisten de un conjunto de nodos $S_j, j \in (0, \cdots, 2^{M_c} - 1)$, correspondiendo a los posibles estados internos del codificador. Los nodos se conectan por ramas que están rotuladas con la notación x/c que especifica el bit de entrada x requerido para hacer la transición y

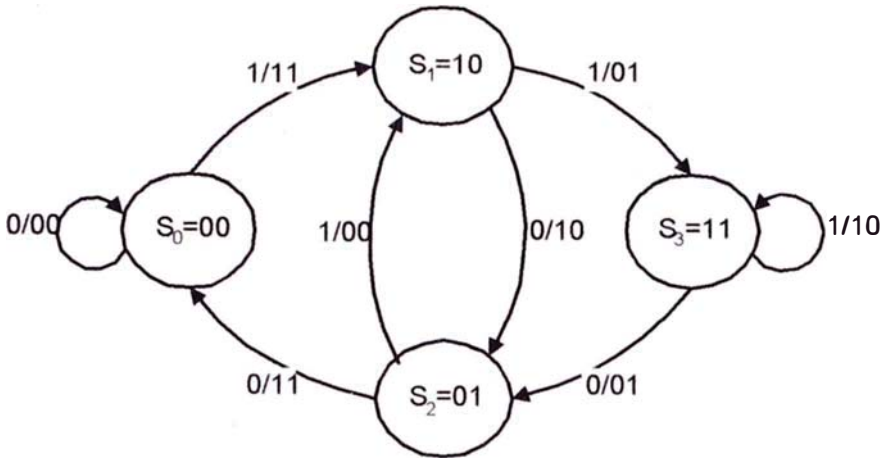


Figura 2.4: Diagrama de Estado para el código convolucional del ejemplo

el correspondiente bit $c = (c^{(0)}, \cdots, c^{(n-1)})$ que son las salidas durante la transición de estado. Un ejemplo de diagrama de estado para el codificador de la Figura 2.3 se muestra en la Figura 2.4.

El diagrama de estado puede ser expandido a un *diagrama de trellis*, el cual muestra explícitamente el paso del tiempo. Los nodos en un diagrama de trellis corresponden no solo a un estado en particular, sino también a un instante discreto de tiempo. Los nodos son rotulados $S_{j,i}$, donde j es el estado e i es el índice de tiempo. Las ramas conectan los nodos del tiempo i con nodos del tiempo $i + 1$. Un ejemplo de diagrama de trellis se muestra en la Figura 2.5.

Para los códigos convolucionales, cada palabra de código se asocia con una única ruta a traves del trellis. Esta ruta se conoce como una *secuencia de estado* y se denota $\mathbf{s} = (s_0, \cdots, s_L)$. Una ruta completa comienza en el estado $s_0 = S_{0,0}$ y termina en el estado $s_L = S_{0,L}$. Hay una correspondencia uno a uno entre cada ruta $\mathbf{s}$, la palabra de código $\mathbf{c}$ asociada con la ruta, y el mensaje $\mathbf{x}$ que produce la palabra de código.

La distancia mínima de un código convolucional es llamada la *distancia libre mínima* y se denota por d_{free}. Para los códigos lineales,

$$d_{free} = \min w(\mathbf{c}) : c_0 = 1. \tag{2.29}$$

Esto es, la d_{free} es el más pequeño de los pesos de Hamming de todas las palabras de código cuya ruta correspondiente $\mathbf{s}$ difiere de la ruta de todos ceros en el tiempo cero. Así como d_{min}, d_{free} es una medida de performance de primer orden.

La representación de trellis no es solamente para los códigos convolucionales. Los códigos lineales de bloque pueden ser representados, también, por un trellis.

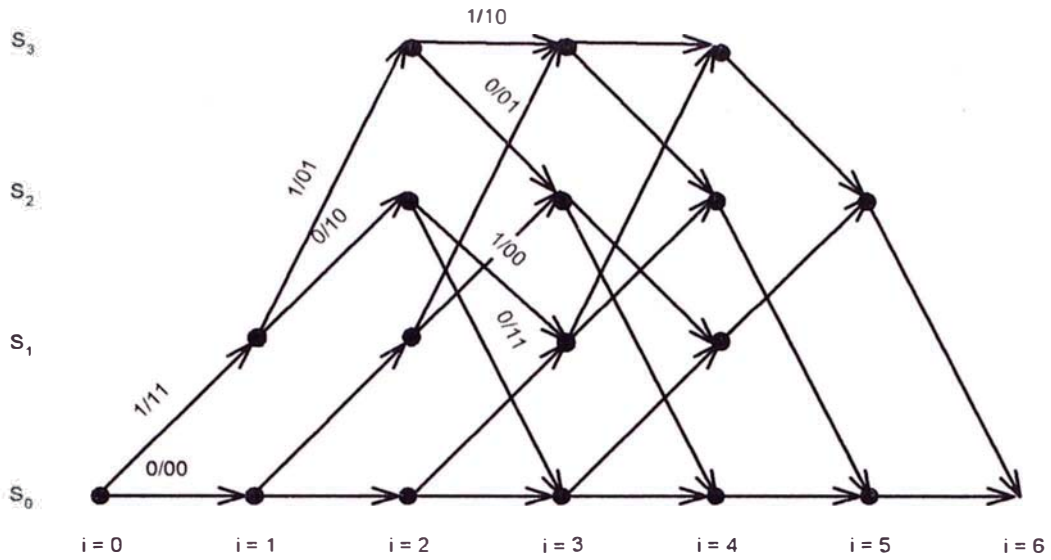


Figura 2.5: Diagrama de Trellis para el código convolucional del ejemplo

Los códigos de bloque tienen trellises que son generalmente variables con el tiempo. Sin embargo, algunas clases de códigos de bloque, como los códigos cíclicos, tienen trellises de tiempo invariable. El número de estados para un trellis de un código de bloque es a lo más 2^{n-k}, lo cual para muchos códigos puede ser muy grande. Históricamente, los trellises no han sido usados frecuentemente para representar códigos de bloque debido a su estructura irregular y gran número de estados. Sin embargo, con la reciente introducción de los turbo códigos compuestos de simples códigos de bloque componentes, existe un renovado interés en representar los códigos de bloque con trellises.

2.2.3. Códigos Convolucionales Terminados

A diferencia de los códigos de bloque, la palabra de código de un código convolucional no tiene longitudes fijas. Sin embargo para muchas aplicaciones es deseable forzar a que la palabra de código convolucional tenga una longitud máxima limitada. Esto se puede realizar por un proceso llamado *terminación de trellis*. Para que la palabra de código termine en el estado $S_{0,L}$ los últimos M_c bits de mensaje deben ser todos ceros. Así el trellis puede ser forzado a regresar al estado todos ceros cambiando a cero los últimos M_c bits del mensaje de entrada. La terminación de trellis cambia un código convolucional en un código de bloque con tasa $r' = k(L - Mc)/(nL)$. Hay que notar que la tasa del código convolucional terminado es menor que la tasa $r = k/n$ del codificador. La reducción de la tasa esta dada por la *tasa de pérdida*

fraccional, definida por [35]

$$\xi = \frac{r - r'}{r} \tag{2.30}$$

$$= \left(\frac{n}{k}\right)\left(\frac{k}{n} - \frac{k(L - M_c)}{nL}\right) \tag{2.31}$$

$$= \frac{M_c}{L}, \tag{2.32}$$

donde r es la tasa del codificador convolucional y r' es la tasa del código terminado. Note que a medida que L crece, la tasa de pérdida fraccional se vuelve insignificante.

2.2.4. Códigos Convolucionales Puncturados

Si se utilizan códigos convolucionales de tasa $1/n$, la tasa más alta que se puede obtener es $r = 1/2$. Para muchas aplicaciones puede que se requieran tasas más altas tales como $r = 2/3$ o $r = 3/4$. Una forma de obtener tasas más altas es usar codificadores que acepten más de un solo tren de bits de entrada y tengan la forma $r = k/n$, $k > 1$. Sin embargo la complejidad del circuito decodificador Agrega - Compara - Selecciona (ACS) crece exponencialmente con el número de trenes de entrada. Entonces es deseable mantener el número de trenes de entrada en el codificador lo más bajo que sea posible, y preferentemente este número debe ser solo uno. Una alternativa a usar múltiples trenes de entrada en el codificador es usar un solo tren de entrada en el codificador y un proceso llamado *puncturado*.

El puncturado es el proceso por el cual se borran sistemáticamente bits del tren de salida de un codificador. El número de bits borrados determina la nueva tasa del código. Por ejemplo suponga que a la salida de un codificador convolucional de tasa $r = 1/2$ uno de cada 4 bits de salida es borrado. Entonces para cada 2 bits en la entrada del codificador, tres permanecen luego del puncturado. Entonces la tasa puncturada es $r = 2/3$. El mismo codificador puede ser usado para generar un código de tasa 3/4 borrando dos de cada seis bits. Uno de los principales beneficios del puncturado es que permite una amplia gama de tasas de codificación para el mismo codificador, simplemente cambiando el número de bits borrados.

Cuando se usa el puncturado la localización de los bits borrados debe ser establecida explícitamente. Se puede usar una *matriz de puncturado* para especificar cuales bits son borrados previos a la transmisión. La matriz de puncturado es $n \times P_p$, donde P_p es el *período* de la matriz de puncturado. Los bits que son transmitidos se especifican por un "1", mientras que los bits puncturados por un "0". Por ejemplo, la

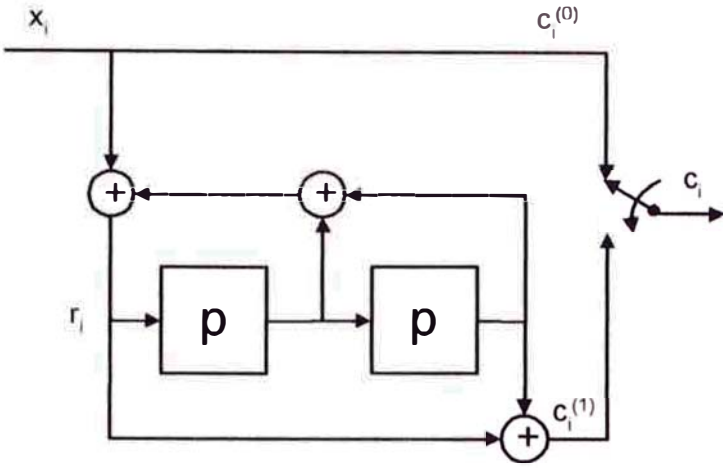


Figura 2.6: Ejemplo de un codificador convolucional sistemático recursivo (CSR) de tasa $1/2$ y longitud de conteción $K_c = 3$

siguiente matriz puede ser usada para incrementar un código de tasa $1/2$ a tasa $2/3$

$$P_M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.33}$$

Si se utiliza tal esquema de puncturado, entonces la palabra de código

$$\mathbf{c} = (c_0^{(0)}, c_0^{(1)}, c_1^{(0)}, c_1^{(1)}, c_2^{(0)}, c_2^{(1)}, c_3^{(0)}, c_3^{(1)}, \cdots) \tag{2.34}$$

se convierte en la salida

$$\mathbf{c}' = (c_0^{(0)}, c_0^{(1)}, c_1^{(0)}, E, c_2^{(0)}, c_2^{(1)}, c_3^{(0)}, E, \cdots), \tag{2.35}$$

donde se usa E para representar los bits puncturados. La transmisión se produce como antes excepto que ahora los bits puncturados no son enviados sobre el canal.

2.2.5. Códigos Convolucionales Sistemáticos Recursivos

Así como los códigos cíclicos pueden ser hechos sistemáticos sin reducir la distancia mínima de Hamming, así también los códigos convolucionales pueden hacerse sistemáticos sin reducir su d_{free} mínima. Un código convolucional de tasa $1/2$ se hace sistemático calculando primeramente el resto $r(p)$ de la división polinómica $x(p)/g^{(0)}(p)$. El polinomio de paridad de salida se encuentra por la multiplicación polinómica $c^{(1)}(p) = r(p)g^{(1)}(p)$, y la salida sistemática es simplemente $c^{(0)}(p) = x(p)$. El resto $r(p)$ puede ser hallado usando una red de registros de desplazamiento con realimentación, en cuyo caso $g^{(0)}(p)$ es llamado el *polinomio de realimentación*

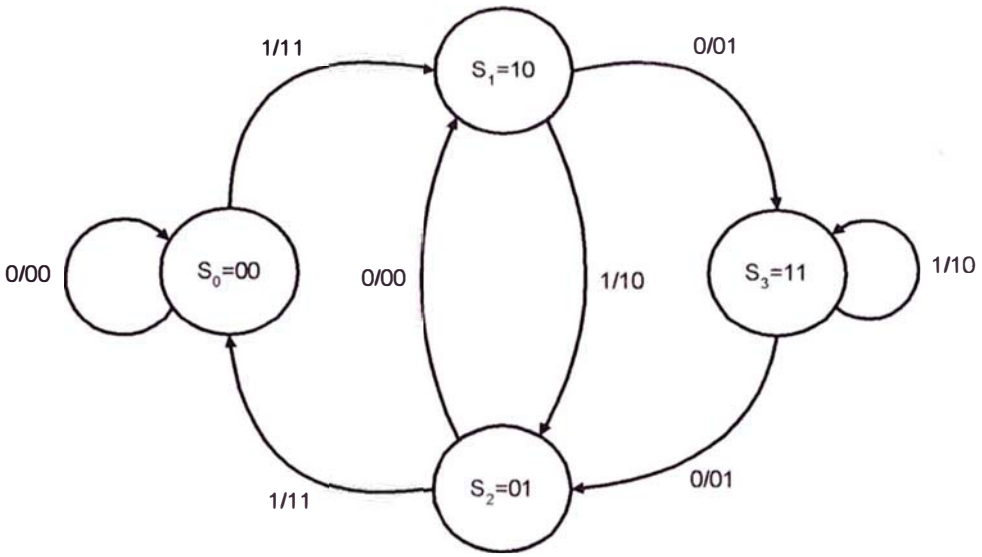


Figura 2.7: Diagrama de Estado para el código CSR del ejemplo

y $g^{(1)}(p)$ es llamado el *polinomio de realimentación hacia delante.* Los códigos generados de esta manera son llamados códigos convolucionales sistemáticos recursivos o códigos CSR. La codificación CSR computa primero la variable de realimentación

$$r_i = x_i + \sum_{j=1}^{M_c} r_{i-j} g_j^{(0)}, \tag{2.36}$$

y luego encontrando la salida de paridad

$$c_i^{(1)} = \sum_{j=0}^{M_c} r_{i-j} g_j^{(1)}. \tag{2.37}$$

Mientras que los codificadores convolucionales convencionales son filtros de respuesta impulso finita, los codificadores CSR son filtros de respuesta impulso infinita (IIR). Un ejemplo de codificador CSR basado en el codificador convolucional convencional del ejemplo previo se muestra en la Figura 2.6. A menos que se especifique, la frase "código convolucional" se refiere a los códigos convolucionales no sistemáticos convencionales, mientras que "códigos CSR" se refiere a códigos convolucionales sistemáticos recursivos.

Los codificadores CSR son máquinas de estado finito y pueden ser representados por diagramas de estado y trellis. El diagrama de estado para el código CSR del ejemplo se muestra en la Figura 2.7 y el diagrama de trellis para este código se muestra en la Figura 2.8. Note que los diagramas de estado y Trellis para los códigos CSR son casi idénticos a los de sus códigos convolucionales no sistemáticos relacionados. De hecho, la única diferencia entre los dos diagramas de estado es que los bits de entrada que dan nombre a las ramas que salen de los nodos S_1 y S_2 son

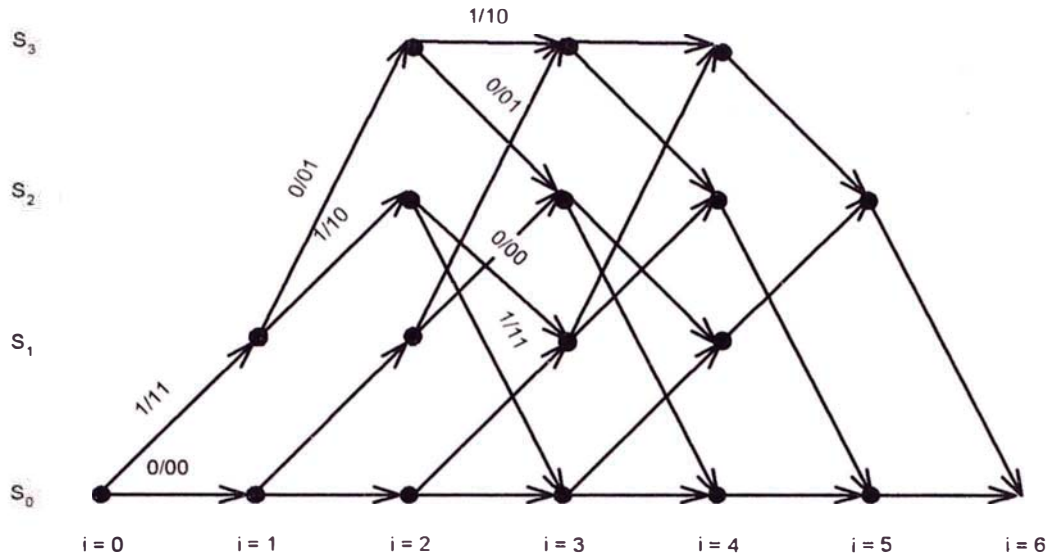


Figura 2.8: Diagrama de Trellis para el código CSR del ejemplo

complementos uno del otro. Con los códigos convolucionales convencionales, los bits de entrada que dan nombre a las dos ramas que entran en cualquier nodo son los mismos. Sin embargo con códigos CSR, los bits de entrada que dan nombre a las dos ramas que entran en cualquier nodo son complementos uno de otro. Como la estructura del Trellis y los bits de salida que dan nombre a las ramas permanecen iguales cuando el código se hace sistemático, la dfree mínima permanece igual.

Con códigos convolucionales convencionales, el trellis puede ser forzado a regresar al estado todos ceros rellenando el mensaje con M_c ceros. Debido a su respuesta impulso infinita, no se puede decir lo mismo de los códigos CSR. Para terminar el Trellis de un código CSR las entradas de mensaje m_i deben escogerse de tal manera que $r_i = 0$ para $L - Mc \leq i \leq L - 1$. Así de (2.21) los últimos M_c bits del mensaje de entrada deben satisfacer

$$m_i = \sum_{j=1}^{M_c} r_{i-j} g_j^{(0)}. \tag{2.38}$$

2.3. Códigos Concatenados

Una debilidad de los códigos convolucionales es su susceptibilidad a los errores en ráfaga. Esta debilidad puede ser superada usando un entrelazador, que altera el orden de los bits del código antes de ser transmitidos. Un de-entrelazador en el receptor pone los bits recibidos nuevamente en el orden apropiado después de la demodulación y previo a la decodificación. Alterando el orden de los bits del código

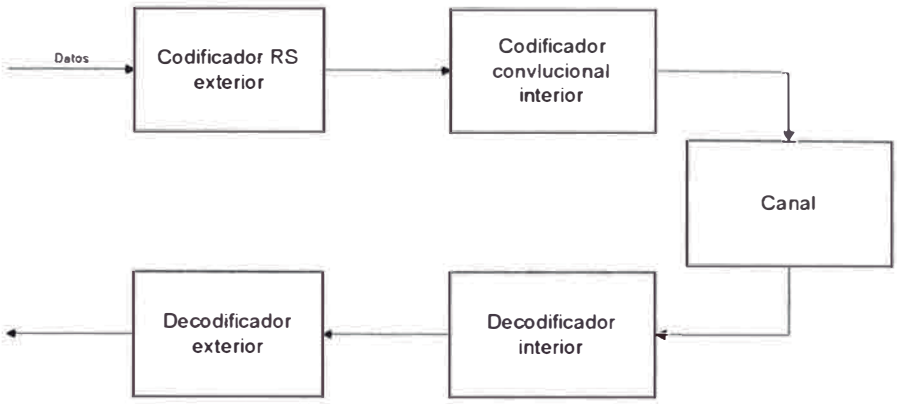


Figura 2.9: Código concatenado serial

en el transmisor y luego aplicando el proceso inverso en el receptor, los errores en ráfaga pueden ser esparcidos de tal forma que parezcan independientes en el decodificador. El tipo más común de entrelazador es el entrelazador de bloque, el cual es simplemente un arreglo de $M_b \times N_b$ bits. Los datos se colocan en el arreglo por columnas y se leen por filas. Un error en ráfaga de longitud hasta de N_b bits puede ser dispersado por un entrelazador de bloque tal que sólo un error ocurra cada M_b bits. Todos los estándares de celulares digitales de segunda generación usan alguna forma de entrelazador de bloque. Un segundo tipo de entrelazador es el cruzado o convolucional, que permite el entrelazado y de-entrelazado continuo y está idealmente preparado para usarse con códigos convolucionales.

Debe notarse que los códigos convolucionales tienen muchas propiedades que son complementarias a las de los códigos Reed-Solomon. Mientras que los códigos convolucionales son susceptibles a errores en ráfaga, los códigos RS manejan estos errores bastante bien. Sin embargo, los códigos convolucionales con codificación de decisión suave generalmente tienen mejor performance que códigos RS de similar complejidad a bajas relaciones señal a ruido (SNR) [35]. En canales limitados en potencia, un diseño de un sistema eficiente e interesante puede ser obtenido utilizando ambos códigos, un código RS y un código convolucional concatenados en serie como se muestra en la Figura 2.9. Los datos son primero codificados por el código exterior RS el cual alimenta a un codificador convolucional interior. En el receptor el decodificador convolucional interior limpia los datos recibidos por el canal con ruido. La salida del decodificador convolucional tiene una mejor relación señal a ruido, pero debido a la naturaleza de los códigos convolucionales los errores están típicamente agrupados en ráfagas. El decodificador exterior RS completa el proceso de decodificación haciendo lo propio con los datos de salida del decodificador convolucional. De esta manera cada decodificador trabaja con el tipo de datos apropiado

- el decodificador convolucional trabaja a bajas relaciones señal a ruido con errores independientes en su mayoría, mientras que el decodificador RS trabaja a altas relaciones señal a ruido con errores en ráfaga mayormente. Este tipo de concatenación de códigos en serie fue propuesto inicialmente por David Forney en 1966, y es usado por la "NASA" y por la "ESA" como estándares para redes espaciales a partir de 1987. En este estándar se emplea un código RS ($q = 8, n = 255, k = 223, t = 16$) con el código convolucional de Odenwalder. En casos extremos como las misiones Galileo y Giotto de la NASA y ESA respectivamente, se puede colocar entre los codificadores convolucional y RS un entrelazador de bloque para dispersar ráfagas largas de errores en varias palabras de código RS.

2.4. Turbo Códigos

En 1993 un descubrimiento revolucionó a todo el mundo de la codificación. Tres investigadores franceses Berrou, Glavieux y Thitimajshima [6] asombraron a la comunidad científica con una nueva técnica de decodificación para códigos concatenados paralelos. Estos son los llamados "Turbo códigos", que en su forma original su codificador esta formado por dos o más códigos CSR, separados por un entrelazador. Para estos códigos se requiere, solamente, una relación señal a ruido (SNR) de 0.7 dB para lograr una tasa de error de bit (BER) de 10^{-5}. El código tiene una tasa $r = 1/2$ y se requiere una longitud de bloque de 64 Kbits para obtener estos resultados. Como se ve en la Figura 2.10, los datos de entrada son entrelazados antes de alimentar al codificador inferior. Debido a que los codificadores son sistemáticos (una de las salidas es la entrada misma) y recibe la misma entrada (aunque en un orden diferente), la salida sistemática del codificador inferior es completamente redundante y no necesita ser transmitida. La tasa resultante del código concatenado paralelo es $r = 1/3$, aunque tasas mayores pueden ser obtenidas por "puncturado" (eliminar bits en forma selectiva del flujo de transmisión) de la salida de paridad, con un circuito multiplexor (MUX).

Debido a la presencia del entrelazador, la decodificación óptima (maximal likelihood) es muy compleja y por ende impracticable. Sin embargo un algorítmo subóptimo de decodificación iterativa se presentó en [6], el cual ofrece una buena performance a una mucha menor complejidad. La idea de esta estrategia de decodificación es descomponer el problema de decodificación en conjunto, en dos problemas más pequeños (decodificar cada uno de los códigos constituyentes) con soluciones ópti-

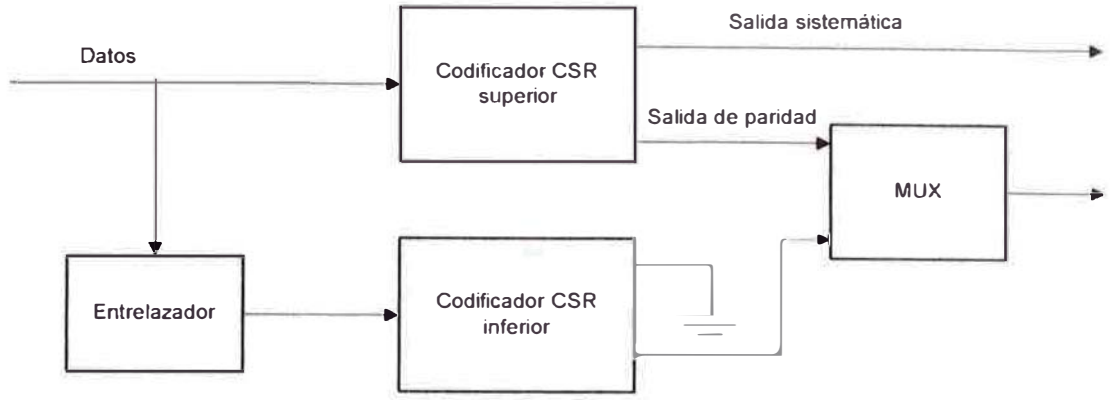


Figura 2.10: Codificador para Turbo Códigos

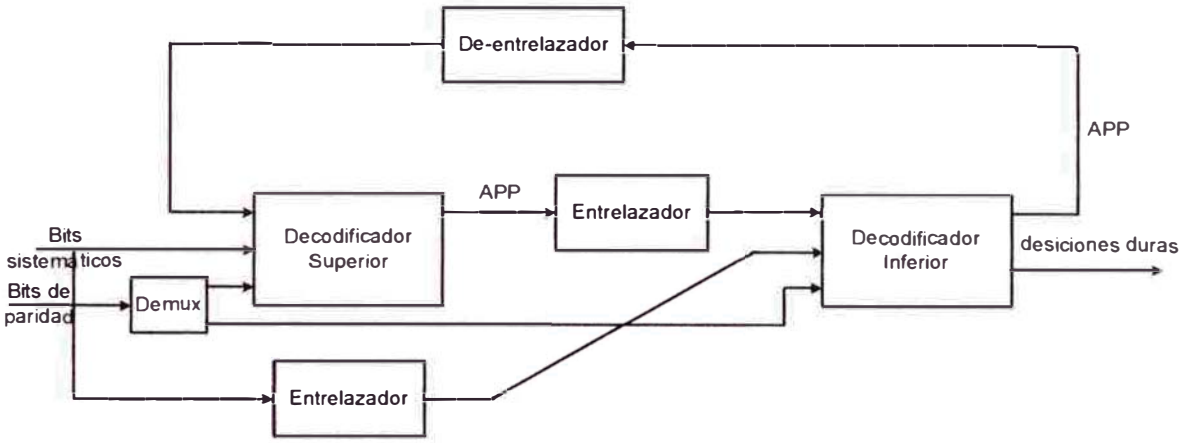


Figura 2.11: Decodificador para Turbo Códigos

mas localmente y compartir información de una manera iterativa. El decodificador asociado a cada uno de los códigos constituyentes esta modificado de tal manera que produce salidas suaves en la forma de probabilidades a posteriori (APPs) de los bits de datos. Los dos decodificadores están colocados en cascada como se muestra en la Figura 2.11 tal que el decodificador inferior recibe la salida suave del decodificador superior. Al final de la primera iteración, la salida suave del decodificador inferior alimenta al decodificador superior y es usada como información a priori durante la siguiente iteración. La decodificación continua en una forma iterativa hasta que se obtiene la performance deseada. Sin embargo, la decodificación iterativa obedece a una ley de disminuir retornos y así la ganancia incremental de cada iteración adicional es menor que la de la iteración previa. Es el método de decodificación lo que le da a los turbo códigos su nombre, ya que la acción de realimentación del decodificador es recordatoria del turbo motor.

El turbo código original de [6] usaba una longitud de contención $Kc = 5$, codificadores CSR y un entrelazador de 65,536 bits. Los bits de paridad eran "puncturados" de tal forma que el código resultante era un código lineal de bloque con $n = 131,072$,

$k = 65,532$. Los resultados de las simulaciones mostraron que una tasa de error de bit de 10^{-5} se podía obtener a una relación Eb/N_0 de sólo 0.7 decibeles, después de 18 iteraciones de decodificación. Así los autores declararon que los turbo códigos estaban a 0.7 dB del límite de Shannon. Mientras que, al principio, existía cierto escepticismo sobre estos resultados, otros investigadores lograron obtener resultados similares para una gran variedad de longitudes de bloque. Sin embargo, se determinó que la performance de los turbo códigos se degradaba a medida que la longitud del código n decrecía (o en forma equivalente a medida que la dimensión del entrelazador disminuía). Otros investigadores pronto comenzaron a considerar usar otras configuraciones de concatenación y otros tipos de códigos componentes. Se encontró que códigos concatenados en forma serial ofrecen performances comparables, y en algunos casos superiores a las de los códigos concatenados paralelos. También, se encontró que la performance con componentes convolucionales se puede lograr o superar con códigos componentes de bloque como son Códigos Hamming, códigos BCH, y códigos Reed-Solomon. Como consecuencia de esto, rápidamente se volvió aparente, que el verdadero avance de la introducción de los turbo códigos no fue la construcción del código, sino más bien el método de decodificación iterativa.

2.5. Comparación de los Sistemas de Codificación

Una comparación de la performance de los códigos correctores de error encontrados en varios sistemas se muestra en la Figura 4.4. Aquí el eje "x" mide la eficiencia en potencia en términos de la relación de energía por bit con densidad espectral de potencia de ruido de una banda E_b/N_0, mientras que el eje "y" mide la eficiencia espectral en términos de la tasa del código r. Se asume un canal con ruido blanco aditivo gausiano. Todos los puntos en la curva asumen modulación BPSK y una tasa de error de bit de $Pb = 10^{-5}$. La curva llamada "Cota de Capacidad de Shannon" es la E_b/N_0 mínima requerida para lograr una comunicación confiable como una función de la tasa del código. Como se asume modulación BPSK, el límite a considerarse es la curva rotulada como "Cota de Capacidad BPSK".

Cuando se usa modulación BPSK sin ninguna codificación de corrección de errores, se requiere una $E_b/N_0 = 9{,}6$ dB para lograr una $Pb = 10^{-5}$. Si se usa codificación el valor requerido de E_b/N_0 puede ser reducido, con el costo, sin embargo, de reducir la eficiencia espectral e incrementar la complejidad del receptor. La diferencia entre la E_b/N_0 requerida cuando se usa un código y la E_b/N_0 requerida para BPSK no

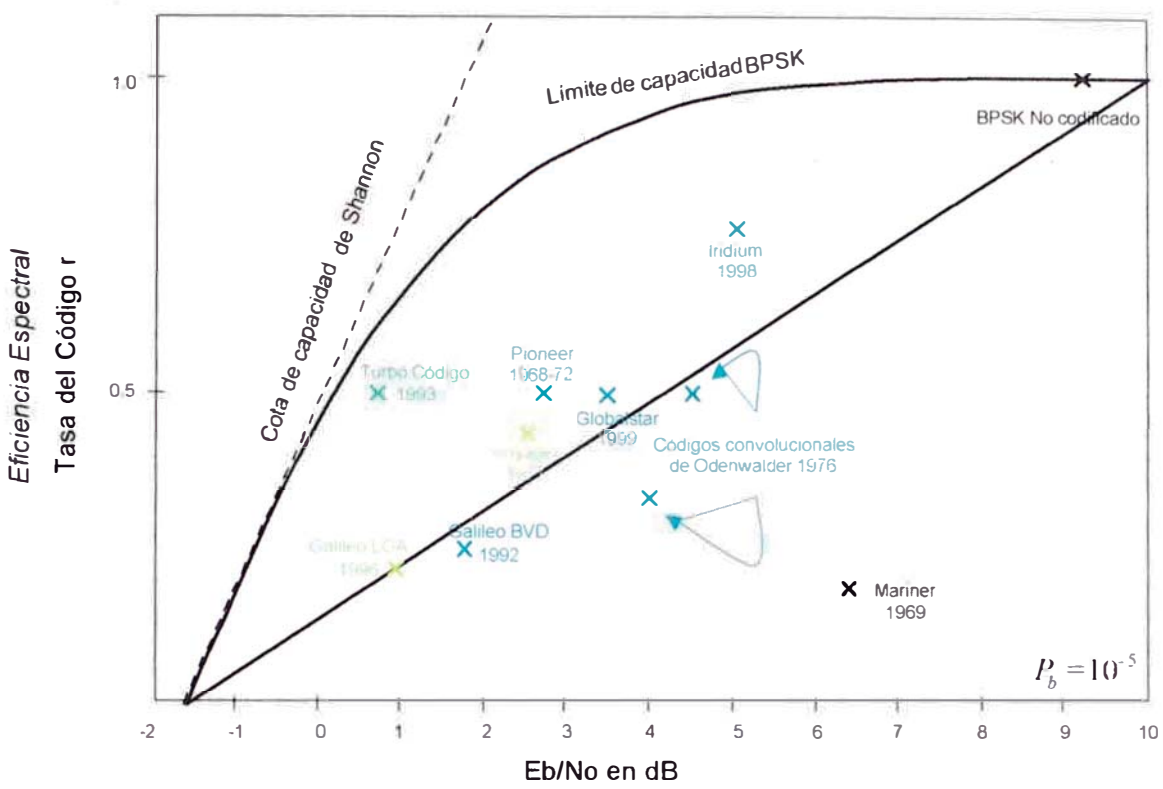


Figura 2.12: E_b/N_0 vs tasa de código para varios sistemas de codificación de corrección de errores, usando modulación BPSK en un canal AWGN

[Gráfico tomado de [33]]

codificado es llamada la ganancia de codificación. El primer código operacional en la Figura 4.4 fue el código Reed Muller (32,6) usado durante la misión del Mariner a Marte en 1969. Este código brindaba 3.2 dB de ganancia de codificación, pero la tasa del código era sólo de $r = 0{,}1875$. Aunque ésta era una modesta ganancia de codificación, fue significativo porque en ese tiempo cada decibel en ganancia de codificación reducía el costo total del sistema en cerca de un millón de dolares. El código usado por las misiones 10, 11 y 12 del Pioneer fue significativamente más potente que el del Mariner. El Pioneer empleó un código convolucional decodificado secuencialmente, de tasa $r = 1/2$ y longitud de contención $K_c = 32$, el cual brindaba una ganancia de codificación de 6.9 dB.

Muchos de los sistemas mostrados en la Figura 4.4 emplean códigos convolucionales. Los códigos Odenwalder de longitud $K_c = 7$ de tasas $1/2$ y $1/3$ han sido adoptados por la NASA y la ESA como parte de los estándares de la DSN, y se han convertido en un estándar industrial de facto para muchas aplicaciones de comunicaciones por satélite. El código Odenwalder de tasa $1/2$ proporciona 5.1 dB de ganancia de codificación, mientras que el código Odenwalder de tasa $1/3$ proporciona 5.6 dB de ganancia de codificación. El código GSM con $K_c = 5$, el código

USDC con $K_c = 6$ y el código IS-95 con $K_c = 9$, proporcionan ganancias de codificación de 4.3, 4.6, y 6.1 dB respectivamente. El código usado por Globalstar es idéntico al de IS-95, mientras que el código convolucional de Iridium con $r = 3/4$ y $K_c = 7$ logra 4.6 dB de ganancia de codificación. El sistema del gran decodificador de Viterbi fue desarrollado para la misión Galileo a Júpiter, y esta'compuesto por un código convolucional de tasa 1/4 y longitud $K_c = 15$ capaz de lograr una ganancia de codificación de 7.9 dB. El sistema usado por las misiones Voyager fue una concatenación en serie de un código Odenwalder con r = 1/2 y un código Reed-Solomon con $q = 28, n = 255, k = 223$ y $t = 16$. El sistema lograba una ganancia de codificación de 7.1 dB a una tasa de código de $r = 0{,}44$. Para compensar los largos errores en ráfaga producidos por el decodificador convolucional, se colocaba entre ambos codificadores un entrelazador de bloque con un espesor de entre 2 y 8 bloques de código exterior. Una concatenación entre un código convolucional y el código RS (28, 255, 223, 16) fue también usada por la misión Galileo a Júpiter, aunque para esta misión se usó un código convolucional BVD con $r = 1/4$. El sistema concatenado probó ser esencial para el éxito de Galileo porque, debido a fallas de la antena de alta ganancia, sólo se pudo usar la antena de baja ganancia. También se muestra en la Figura 4.4 la performance del turbo código original. Como se puede ver los turbo códigos se aproximan al límite de capacidad mucho más cercanamente que cualquiera de los otros códigos. Además, la complejidad del decodificador usado por el turbo código es aproximadamente la misma que la del decodificador BVD. Es precisamente debido a esta extraordinaria performance y razonable complejidad que los turbo códigos han sido un foco de atención de la comunidad de codificación.

Capítulo 3

Fundamentos de turbo códigos

En el capítulo anterior se introdujo el tema de los turbo códigos y se vió como estos tienen un desempeño muy superior a todos los códigos de corrección de errores conocidos hasta antes de su descubrimiento en 1993. En éste capítulo se explicará el funcionamiento de los componentes que tienen una ingerencia directa en este desempeño y se dará una luz de como mejorarlo para sistemas de tramas cortas, dando un punto de partida para el trabajo del siguiente capítulo, en el que se propondrá ya un diseño más adecuado para aquellos sistemas.

Desde su aparición, hace ya 13 años, los turbo códigos han despertado un gran interés en la comunidad internacional de codificación, generando un gran esfuerzo de investigación, que ha dado como resultado un notorio desarrollo en el conocimiento de sus fundamentos teóricos y en el descubrimiento de diversas aplicaciones prácticas, que han redundado en su adopción en varios sistemas. Por ejemplo, han sido incorporados en estándares usados por la NASA para comunicaciones del espacio profundo (CCSDS), difusión de video digital (DVB-T), y en los estándares de telefonía celular de tercera generación UMTS y cdma2000.

Siguiendo el planteamiento de Shannon en su teorema de codificación para canales ruidosos [30], un grupo de investigadores franceses, empleando códigos largos y casi aleatorios[1], asociados a un procedimiento de decodificación iterativo, descubrieron los turbo códigos en 1993 [6]. Estos demostraron tener una excelente performance, que se acerca a los límites teóricos predichos por Shannon, de una forma nunca antes sospechada posible.

Esta performance superior de los "Turbo códigos" con respecto a la de los códigos convolucionales, es sólo alcanzada cuando la longitud de la trama es bastante larga, en el orden de varios miles de bits. Se determinó, asimismo, que la performance de los turbo códigos se degrada a medida que la longitud del código n decrece (o en forma equivalente a medida que la dimensión del entrelazador disminuye).

En el caso de aplicaciones como la telefonía celular la longitud de las tramas

[1]No es posible trabajar con códigos perfectamente aleatorios, ya que no se podrían implementar en la práctica. Los códigos deben poseer alguna estructura, que les permita contar con algorítmos de codificación y decodificación realizables

es por lo general inferior a los 400 bits. Para estas aplicaciones, tramas tan largas como las normalmente usadas en los Turbo códigos no son apropiadas. Es así como el diseño de Turbo códigos para tramas de bits de corta longitud, ha generado un gran interés dada la tendencia mundial hacia las comunicaciones móviles, y es por eso que este es el tema de esta tesis.

Para lograr este objetivo se puede optar por tres alternativas, que son:

1. Un apropiado diseño del entrelazador para obtener ventaja de la ganancia de codificación que ofrecen los Turbo códigos[18, 15, 19]

2. La búsqueda de códigos constituyentes buenos[21, 17]

Es así como en este capítulo la atención estará centrada en los turbo códigos, los que comparten varios de los mismos conceptos y terminología de los códigos de bloque y los convolucionales. Por esta razón, se expusieron estos en el capítulo precedente, buscando establecer una base teórica que nos permita iniciar la discusión de éstos códigos revolucionarios.

3.1. Estructura del Codificador

En su forma original, los turbo códigos tienen un codificador formado por dos o más códigos constitutivos que son una subclase de los códigos convolucionales conocidos como códigos convolucionales recursivos sistemáticos (CSR), separados por un entrelazador, ver Figura 3.1. El descubrimiento de los Turbo códigos fue tomado con ecepticismo por muchos debido a la excepcional performance demostrada. Se requería solamente de una relación señal a ruido de 0.7 dB para lograr una probabilidad de error de bit (BER) de 10^{-5}, y esto con una tasa de código $r = 1/2$.

Estos mismos resultados fueron posteriormente obtenidos por varios investigadores alrededor del mundo [1, 2]. Sin embargo, esta performance se obtuvo con una longitud de bloque de 64 Kbs, demostrándose que cuando la longitud del entralazador es corta, la performance del Turbo código se degrada sustancialmente hasta el punto que su probabilidad de error de bit es peor que la de los códigos convolucionales [18].

El turbo código original de [6] usaba una longitud de contención $K_c = 5$, codificadores CSR y un entrelazador de 65,536 bits. Los bits de paridad eran "puncturados" de tal forma que el código resultante era un código lineal de bloque con tasa $r = 1/2$, $n = 131,072$ y $k = 65,532$. Los resultados de las simulaciones mostraron que una

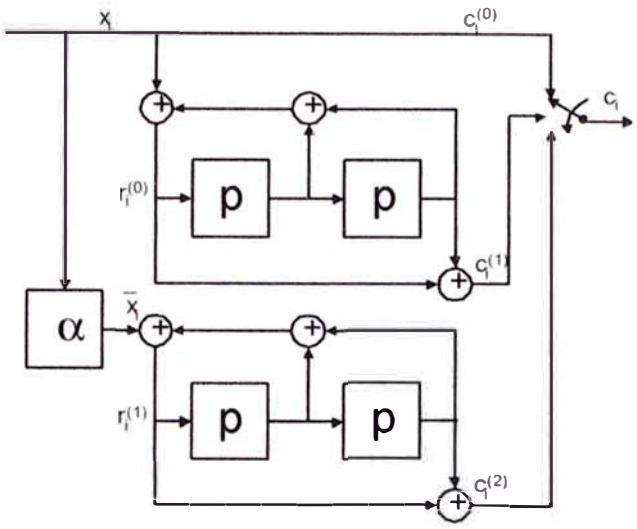


Figura 3.1: Codificador para el Turbo Código del ejemplo

tasa de error de bit de 10^{-5} se podía obtener a una relación E_b/N_0 de sólo 0.7 decibeles, después de 18 iteraciones de decodificación. Así los autores declararon que los turbo códigos estaban a 0.7 dB del límite de Shannon.

Debido a que los códigos convencionales de bloque y convolucionales son altamente estructurados, tienen codificadores y decodificadores que pueden ser implementados con una complejidad razonable. Sin embargo, la misma estructura que facilita la implementación práctica resulta en una performance significativamente inferior a los límites de la codificación aleatoria predichos por Shannon. Mientras que estos contienen suficiente estructura como para permitir algoritmos prácticos de codificación y decodificación, los turbo códigos poseen algunas propiedades de tipo aleatorias. Como consecuencia la performance de los turbo códigos se acerca mucho más a los límites de Shannon que la de los más tradicionales códigos convolucionales y de bloque.

Un comentario final en cuanto a la terminología de los Turbo códigos. No hay una definición exacta de lo que son Turbo códigos. Por ejemplo se pueden sustituir los códigos constitutivos por cualquier otro tipo de código, por ejemplo códigos de bloque. Tales construcciones se denominan Turbo códigos de bloque [26]. Además es posible concatenar varios códigos constitutivos insertando codificadores adicionales en paralelo [8]. Tales construcciones son aún llamadas Turbo códigos múltiples. En las investigaciones que se reporten en esta tesis, se hará referencia exclusivamente a la estructura original del codificador presentado por Berrou et al. en [6], esto es,

concatenación en paralelo de dos codificadores convolucionales recursivos sistemáticos.

3.2. Factores que afectan la Performance de los Turbo Códigos

3.2.1. Códigos Constituyentes

Un diagrama general para el turbo codificador se dió en la Figura 2.10. El turbo codificador se compone de dos codificadores CSR, los cuales son usualmente idénticos. Los dos codificadores reciben los mismos bits, pero el segundo codificador el más bajo, recibe los bits luego de haber sido permutados por un entrelazador. Es el entrelazado lo que hace que los turbo códigos parezcan aleatorios. Debido a que los codificadores son sistemáticos (una de las salidas es la entrada misma) y recibe la misma entrada (aunque en un orden diferente), la salida sistemática del codificador inferior es completamente redundante y no necesita ser transmitida. La tasa resultante del código concatenado paralelo es $r = 1/3$, aunque tasas mayores pueden ser obtenidas por "puncturado" (eliminar bits en forma selectiva del flujo de transmisión) de la salida de paridad con un circuito multiplexor (MUX).

Debido a que el entrelazador debe tener una estructura fija y como generalmente trabaja los datos en forma de bloques, los turbo códigos son por necesidad códigos de bloque. Si el entrelazador tiene unas dimensiones fijas y ambos codificadores CSR comienzan en el estado todos ceros, entonces el turbo código es un código lineal de bloque.

No hay que olvidar que la distancia mínima de un código lineal de bloque es una buena medida de la performance del código. Para estos, la d_{min} es la palabra de código válida con el menor peso de Hamming diferente de cero. La combinación del entrelazado y la codificación CSR asegura que la mayoría de las palabras de código producidas por un turbo código tengan un peso de Hamming alto. Debido a su respuesta impulso infinita, la salida de un codificador CSR generalmente tiene un alto peso de Hamming. Existen, sin embargo, algunas secuencias de entrada que causan que un codificador CSR produzca salidas de bajo peso. Debido al entrelazador, los dos codificadores CSR no reciben sus entradas en el mismo orden. Así si uno de los codificadores recibe una entrada que origina una salida de bajo peso, entonces es improbable que el otro codificador también reciba una entrada que produzca una

salida de bajo peso. Desafortunadamente, siempre existirán algunos pocos mensajes de entrada que causan que ambos codificadores CSR produzcan salidas de bajo peso y así la distancia mínima de un turbo código no es, en general, particularmente alta. Pero la *multiplicidad* de las palabras de código de bajo peso en turbo códigos bien diseñados es baja.

Los turbo códigos tienen buena performance en relaciones señal a ruido bajas debido a la poca cantidad de palabras de código de bajo peso. Sin embargo, la performance de los turbo códigos a relaciones señal a ruido más altas se vuelve limitada por la relativamente pequeña distancia mínima del código. Mientras que la meta del diseño tradicional de códigos es incrementar la distancia mínima del código, la meta del diseño de los turbo códigos es reducir la multiplicidad de las palabras de código de bajo peso. Esta nueva manera de ver la codificación es resumida por David Forney: "Contrario al conocimiento convencional, en lugar de atacar exponentes de error, atacan multiplicidades".

3.2.2. Entrelazador

Un turbo codificador basado en el codificador CSR del ejemplo se muestra en la Figura 3.1. El bloque entrelazador se denota por α y su salida es $\bar{\mathbf{x}}_\mathbf{i}$. La función a especifica al entrelazador de acuerdo a

$$\bar{\mathbf{x}}_{\alpha(\mathbf{i})} = \mathbf{x}_\mathbf{i}, \tag{3.1}$$

donde $i \in (0, \ldots, L-1)$. El proceso de *deentrelazado* puede ser definido en forma similar como

$$\mathbf{x}_{\alpha^{-1}(\mathbf{i})} = \bar{\mathbf{x}}_\mathbf{i}. \tag{3.2}$$

El entrelazado que se realiza antes del segundo codificador es un reordenamiento de los bits de información. La combinación de los dos codificadores CSR y el entrelazador brinda la solución a dos aspectos importantes en la codificación:

1. La creación de códigos con buenas propiedades de distancia
2. Que pueden ser eficientemente decodificados con decodificación iterativa [17]

La razón por la que los entrelazadores permiten el uso de decodificación iterativa es que decorrelaciona las entradas de bits contiguos al codificador, incrementando las propiedades de distancia del sistema. Esto es fundamental, debido a que si existe correlación entre los bits contiguos de una trama se produce un deterioro en la performance de la decodificación.

3.2.3. Puncturado

La salida sistemática del turbo codificador $\mathbf{c}^{(0)}$ se toma del codificador CSR superior. Las dos salidas de paridad $\mathbf{c}^{(1)}$ y $\mathbf{c}^{(2)}$ se toman de las salidas de paridad de los codificadores CSR superior e inferior respectivamente. Los tres trenes de salida se multiplexan para formar la palabra de código $\mathbf{c} = (c_0^{(0)}, c_0^{(1)}, c_0^{(2)}, \ldots, c_{L-1}^{(0)}, c_{L-1}^{(1)}, c_{L-1}^{(2)})$. La tasa resultante de un turbo código (despreciando la pérdida de tasa fraccional debido a la terminación de trellis) es 1/3. Así como con los códigos convolucionales, esta tasa puede ser incrementada por puncturado. En particular, un turbo código de tasa 1/2 puede ser obtenido de un turbo código de tasa 1/3 usando la siguiente matriz de puncturado

$$P_M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.3}$$

En general como se ha podido apreciar, y se explicó en el capítulo precedente, el puncturado es el proceso por el cual se borran sistemáticamente ciertos bits del tren de salida de un codificador. El número de bits borrados determina la nueva tasa del código.

3.2.4. Terminación de Trellis

Un punto que afecta la performance de los turbo códigos es la terminación de trellis. Mientras que es posible terminar el trellis de cualquiera de los codificadores CSR componentes con una cola de M_c bits de código, la terminación simultánea de ambos trellises no es una tarea sencilla. Esto es por dos puntos complicados:

1. Debido al entrelazador, los bits de la cola no están localizados necesariamente al final del mensaje, así como en el caso de los códigos convolucionales convencionales y los CSR.

2. Debido a la combinación del entrelazador y la naturaleza recursiva de los codificadores, es difícil computar los valores de los bits de la cola.

Mientras que los bits de la cola usados para terminar el trellis del codificador superior se localizan al final del mensaje, el entrelazador origina que los bits de la cola usados para terminar el trellis del codificador inferior sean dispersados a traves del mensaje. A pesar que este hecho nos lleva a una implementación hacia adelante no ocasiona por si solo un problema significativo. El segundo aspecto, sin embargo,

crea un dilema. Debido a la naturaleza recursiva del codificador componente, los bits de la cola para cada codificador no son conocidos antes que ese codificador haya terminado de codificar sus datos. Pero debido al entrelazador, los bits de la cola de un codificador se convierten en datos para el otro codificador, influenciando así en el valor de los bits de cola de los otros codificadores. Así para computar la cola para el primer codificador, la cola para el segundo codificador debe ser conocida primero y viceversa. Este problema hace que sea difícil computar una cola que termine ambos trellis.

Hay muchas soluciones para el problema de la terminación trellis. Una opción es terminar uno de los trellis (usualmente el del codificador superior) y dejar el otro "abierto". Otra opción es diseñar el entrelazador de tal forma que los dos trellis puedan ser terminados al mismo tiempo con una sola cola de M_c bits. Puede demostrarse que la respuesta impulso para cada codificador componente CSR es periódica con período $p \leq 2^{M_c} - 1$. Si el polinomio de realimentación es primitivo con grado M_c, entonces la respuesta impulso es una secuencia de máxima longitud con período $p = 2^{M_c} - 1$. La condición para el entrelazador que permite que ambos codificadores CSR terminen con la misma cola es

$$i \bmod p = \alpha(i) \bmod p \quad \forall i. \tag{3.4}$$

Si el entrelazador es diseñado usando 3.4, entonces la misma cola de M_c bits que termina el codificador superior, termina también el codificador inferior. Sin embargo, esta regla de diseño reduce el número de posibles entrelazadores e impone un nivel extra de estructura al turbo codificador. Debido a su estructura, los turbo códigos que son terminados por una sola cola tienden a comportarse más pobremente que los turbo códigos que usan una aproximación diferente al problema de la terminación trellis.

Otra solución para el problema de la terminación es forzar ambos codificadores al estado todos ceros usando el circuito codificador mostrado en la Figura 3.2. Los dos interruptores están en la posición arriba antes del término de la trama de datos, en cuyo momento ambos se colocan en la posición abajo. Cada codificador entonces genera en forma independiente los bits de cola requeridos para terminar. La cola para el primer codificador se incluye al final de la salida sistemática $x_i^{(0)}$, pero la cola para el segundo codificador $x_i^{(1)}$ debe ser transmitida separadamente. Aunque esta aproximación termina ambos codificadores adolece del hecho que el conjunto de bits en la entrada de ambos codificadores no son los mismos. Esto reduce la eficiencia del algoritmo de decodificación, el cual se deriva con la suposición que

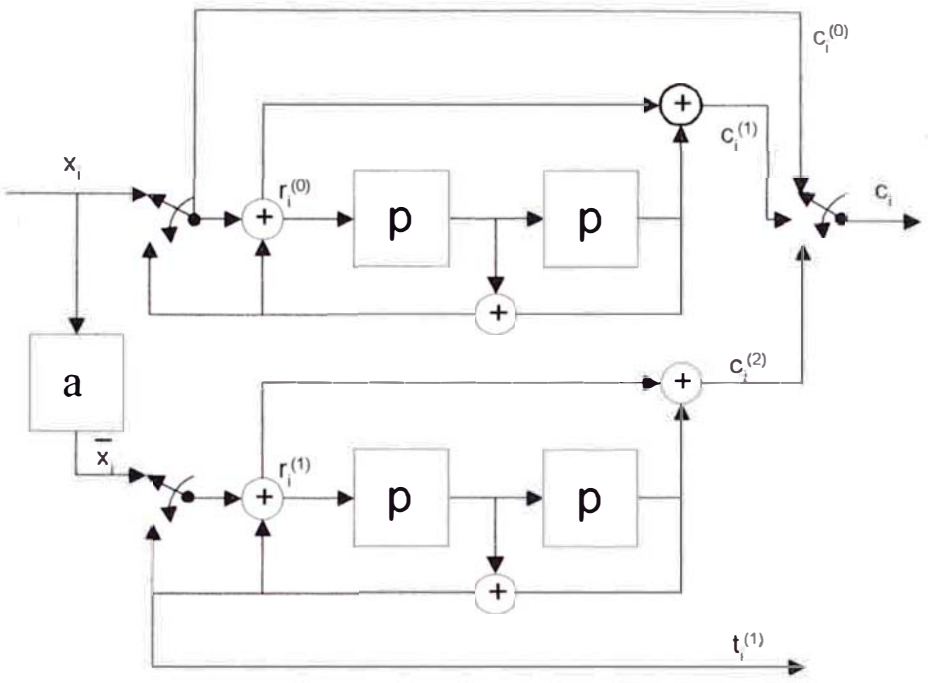


Figura 3.2: Turbo codificador con terminación de Trellis forzada

ambos codificadores reciben los mismos datos de entrada, solo que en diferente orden.

Otra solución al problema de la terminación del trellis es usar "bits precursores" y codificar los datos dos veces. Un primer paso para codificar llevando los $2M_c$ primeros bits, *bits precursores*, a cero. Luego se examinan los estados finales de los dos codificadores y se usa una tabla para determinar cuales deben ser los bits precursores para terminar los trellises de los codificadores. Los bits precursores son entonces colocados en este valor y se realiza un segundo paso de codificación en cuyo punto los codificadores estarán en el estado todos ceros. Esta técnica requiere que exista una correspondencia uno a uno entre el valor de los bits precursores y el estado final de los codificadores , los cuales establecen una restricción en el diseño del entrelazador. Sin embargo, esta restricción es menos severa que 3.4, y por lo tanto la performance de esta técnica es mejor que la propuesta en [34].

3.2.5. Algoritmo de Decodificación

El número de iteraciones tiene gran impacto en la performance debido a que la mayoría de los algoritmos de decoficación son iterativos. Este concepto de decodificación iterativa se explicará en la siguiente sección.

3.3. Decodificación Iterativa

Debido a la presencia del entrelazador, la decodificación óptima (maximal likelihood) es increíblemente compleja y por ende impracticable. Sin embargo un algoritmo sub-óptimo de decodificación iterativa se presentó en [24], el cual ofrece una buena performance a una mucha menor complejidad. La idea de esta estrategia de decodificación es descomponer el problema de decodificación en conjunto, en dos problemas más pequeños (decodificar cada uno de los códigos constitutivos) con soluciones óptimas localmente y compartir información de una manera iterativa. El decodificador asociado a cada uno de los códigos constitutivos esta modificado de tal manera que produce salidas suaves en la forma de probabilidades a posteriori (APPs) de los bits de datos. Los dos decodificadores están colocados en cascada como se muestra en la Figura 2.11 tal que el decodificador inferior recibe la salida suave del decodificador superior. Al final de la primera iteración, la salida suave del decodificador inferior alimenta al decodificador superior y es usada como información a priori durante la siguiente iteración. La decodificación continua en una forma iterativa hasta que se obtiene la performance deseada. Sin embargo, la decodificación iterativa obedece a una ley de disminuir retornos y así la ganancia incremental de cada iteración adicional es menor que la de la iteración previa. Es el método de decodificación lo que le da a los turbo códigos su nombre, ya que la acción de realimentación del decodificador es recordatoria del turbo motor.

Hasta ahora se ha visto que los turbo códigos pueden lograr una excelente performance a bajas relaciones señal a ruido (SNR) debido, en primer lugar, a la baja multiplicidad de las palabras de código de distancia mínima. Los algoritmos de decodificación para turbo códigos son iterativos y su complejidad se incrementa solo linealmente con la longitud de la trama. De hecho el nombre "turbo" se usa porque el decodificador tiene realimentación como un motor turbo. Dado que la complejidad de un Turbo Código recae en su decodificador iterativo, es muy importante entender el algoritmo de decodificación y encontrar la forma más simple de realizar el decodificador sin disminuir la performance.

3.3.1. Turbo Decodificación

Se ha mostrado anteriormente que los turbo códigos pueden alcanzar una gran performance a bajas SNR debido principalmente a la baja multiplicidad de las palabras de código de distancia mínima. Sin embargo, el algoritmo usado para decodificar

los turbo códigos no se ha descrito hasta ahora. En esta sección se presentará una introducción al proceso de turbo decodificación. Se mostrará que un turbo decodificador de dos módulos de entrada y salida suaves que trabajan juntos de una forma iterativa.

3.3.2. Clases de Algoritmos de Decodificación

El problema de estimar la sequencia de estado **s** de un proceso de Markov observado a través de ruido tiene dos soluciones basadas en trellis bien conocidas - El algoritmo de Viterbi (VA) [12] y el Algoritmo maximun a posteriori (MAP) [24]. Una vez que la secuencia de estado es estimada, es una tarea trivial determinar el mesaje **m** o la palabra de código **x** asociada con el.

El VA y el algoritmo MAP difieren en su criterio de optimización. El algoritmo de Viterbi encuentra la secuencia de estado más probable **s** dada la secuencia recibida **y**

$$\hat{\mathbf{s}} = \arg\{\max_{\mathbf{s}} P[\mathbf{s}|\mathbf{y}]\}. \tag{3.5}$$

El algoritmo MAP, por otra parte, trata de encontrar el estado individual s_i más aparente dado **y**

$$\hat{s}_i = \arg\{\max_{s_i} P[s_i|\mathbf{y}]\}. \tag{3.6}$$

La diferencia clave entre algoritmos es que los estados estimados por el VA deben formar una ruta conectada a traves del trellis, mientras que los estados estimados por el algoritmo MAP no necesitan conectarse. Cuando se aplica a sistemas de transmisión digital, el VA minimiza la tasa de error de trama (FER), mientras que el algoritmo MAP minimiza la tasa de error de bit (BER).

El problema de decodificación de turbo códigos involucra la estimación conjunta de dos procesos de Markov, uno para cada código constitutivo. Mientras que en teoría es posible modelar un turbo código como un proceso de Markov simple, tal representación es extremadamente compleja y no permite ser expresada por algoritmos de decodificación computacionalmente realizables. La turbo decodificaión procede más bien estimando independientemente los procesos de Markov individuales. Debido a que los dos procesos de Markov son conducidos con el mismo conjunto de datos, las estimaciones pueden ser afinadas intercambiando información entre los dos decodificadores de una manera iterativa. Más específicamente, la salida de un decodificador puede ser usada como información a priori por el otro decodificador. Si las salidas de los decodificadores individuales están en la forma de decisiones duras de bit,

entonces existe muy poca ventaja de intercambiar información. Sin embargo, si se producen decisiones suaves de bit por los decodificadores individuales, se pueden lograr considerables ganancias en la performance ejecutando múltiples iteraciones de decodificación.

3.4. Análisis de la Performance de los Turbo Códigos

La probabilidad de error de bit de un código de bloque lineal (n, k) observada a través de un canal con ruido aditivo blanco Gausiano (AWGN) cumple con la cota

$$P_b \leq \sum_{i=1}^{2^k-1} \frac{w(\mathbf{m}_i)}{k} Q\left(\sqrt{w(\mathrm{x_i})\frac{2rE_b}{N_0}}\right), \tag{3.7}$$

donde la función Q se define por

$$Q(z) = \frac{1}{\sqrt{2\pi}} \int_z^\infty e^{x^2/2} \mathrm{d}x, \tag{3.8}$$

y asumiendo decodificación maximum likelihood (ML). E_b es la energía por bit y N_0 es la densidad espectral de ruido de una banda. Para computar 3.7 se requiere conocimiento de los pesos de todas las $2^k - 1$ palabras de código diferentes de cero y sus correspondientes mensajes, y tiene una complejidad computacional que limita su uso a los valores más pequeños de k. El número de términos en 3.7 puede ser grandemente reducido agrupando términos que producen los mismos pesos de palabras de código

$$P_b \leq \sum_{d=d_{min}}^{n} \frac{\tilde{w}_d N_d}{k} Q\left(\sqrt{d\frac{2rE_b}{N_0}}\right), \tag{3.9}$$

donde N_d es el número de palabras de código de peso d y $\tilde{w}_d$ es el peso promedio de los N_d mensajes que producen palabras de código de peso d. A altas relaciones señal a ruido la probabilidad de error de bit es aproximada por el primer término de 3.9

$$P_b \approx \frac{\tilde{w}_{d_{min}} N_{d_{min}}}{k} Q\left(\sqrt{d_{min}\frac{2rE_b}{N_0}}\right), \tag{3.10}$$

que se conoce como la *asíntota de distancia mínima.*

La meta del diseño de códigos ha sido tradicionalmente maximizar d_{min}. Con turbo códigos, sin embargo, el énfasis está más bien en minimizar el coeficiente $\tilde{w}_{d_{min}} N_{d_{min}}/k$. Considere, por ejemplo, el turbo código compuesto por dos codificadores CSR con $K_c = 5$ con generadores $(21)_O$ y $(37)_O$. La longitud del entrelazador es $L = 65,536$. La matriz de puncturado de la Ecuación 3.3 se usa para incrementar

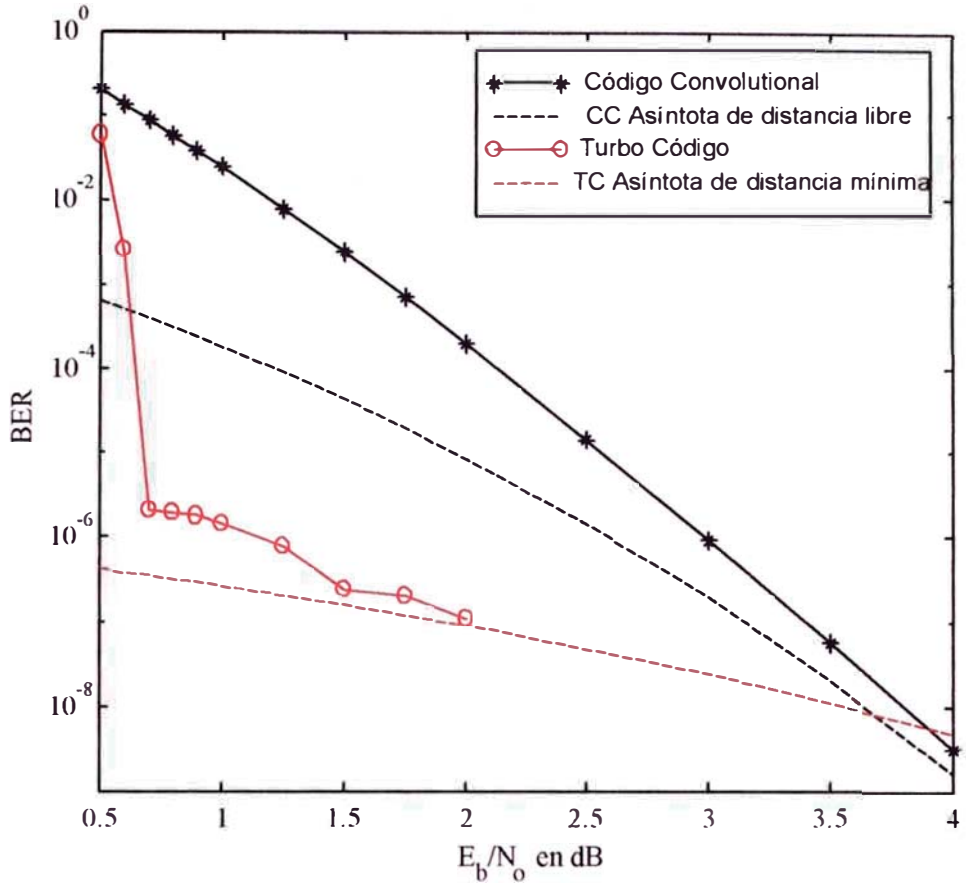


Figura 3.3: Performance asintótica de un turbo código de tasa $r = 1/2$, longitud de contención $K_c = 15$ y de un código convolucional de los mismos parámetros [Gráfico tomado de [33]]

la tasa del código a $r = 1/2$ y se asume que se termina sólo el trellis del codificador superior. Este es un código de bloque (131,072; 65,532). En [29] se reportó que la distancia mínima para éste código es $d_{min} = 6$, el número de las palabras de código de distancia libre es en promedio $N_{d_{min}}$ y el peso promedio de los mensajes asociados con las palabras de código de distancia mínima es $\tilde{w}_{d_{min}} = 2$. De esta manera la asíntota de distancia mínima para éste código es

$$P_b \approx \frac{(4,5)(2)}{65532} Q\left(\sqrt{6\frac{E_b}{N_0}}\right). \tag{3.11}$$

Ahora considere la performance asintótica de un código convolucional de similar complejidad de codificador. Dado que la longitud de bloque de los códigos convolucionales no esta limitada, la Ecuación 3.9 no es aplicable. En lugar de ello, la siguiente cota es usada para la decodificacion Maximal Likelihood de códigos convolucionales [23]

$$P_b = \sum_{d=d_{free}}^{\infty} W_d^0 Q\left(\sqrt{d\frac{2rE_b}{N_0}}\right), \tag{3.12}$$

donde W_d^0 es la suma de todos los mensajes con $m_0 = 1$ y cuyas palabras de código asociadas tienen peso d. A relaciones señal a ruido altas la probabilidad de error de bit de un código convolucional se aproxima a la asíntota de distancia libre

$$P_b \approx W_{d_{free}}^0 Q\left(\sqrt{d_{free}\frac{2rE_b}{N_0}}\right). \tag{3.13}$$

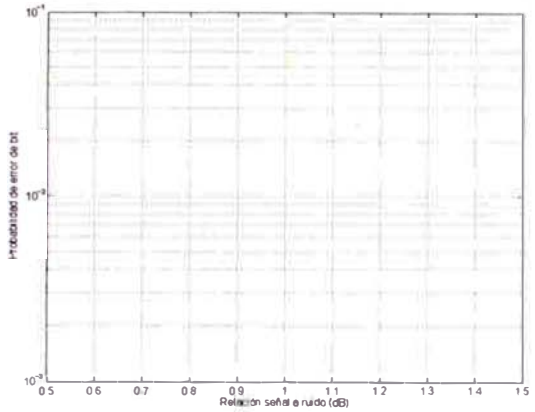
La complejidad del codificador del código convolucional $r = 1/2$ y $K_c = 15$ presentado en [7] es aproximadamente la misma que la del turbo código con los mismos valores de r y K_c. Para este código convolucional $d_{free} = 18$ y $W_{d_{free}}^0 = 187$ y así la asíntota de distancia libre es

$$P_b \approx 187Q\left(\sqrt{18\frac{E_b}{N_0}}\right). \tag{3.14}$$

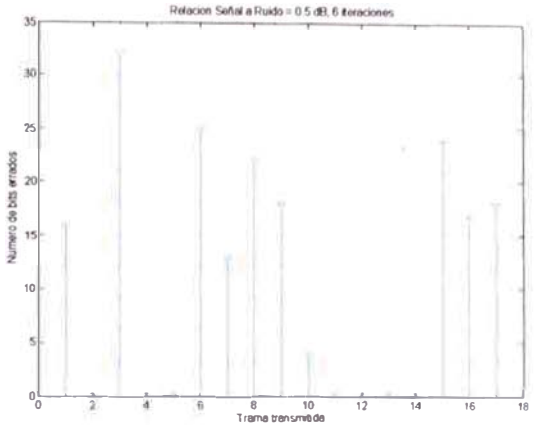
Note la diferencia entre la Ecuaciones 3.11 y 3.14. El argumento de la función Q es mayor para el código convolucional que para el turbo código, por lo que se puede esperar que el trazado de la asíntota del código convolucional sea mucho más inclinada que la del turbo código. sin embargo el coeficiente que precede a la función Q es mucho menor para el turbo código que para el código convolucional. De esta manera, para valores de E_b/N_0 suficientemente pequeños, puede esperarse que la asíntota de los turbo códigos esté más abajo que la de los códigos convolucionales.

En la Figura 3.3, la asíntota de distancia mínima para el turbo código se muestra juntamente con su performance simulada de error de bit. La simulación se llevó a cabo usando 18 iteraciones del algoritmo de decodificación log-MAP, y se ejecutó hasta que 40 palabras de código fueron detectadas en error para cada valor E_b/N_0. Note que para grandes valores de E_b/N_0, la cota sirve como una buena aproximación de la performance actual. También note que a medida que E_b/N_0 se incrementa la inclinación de la curva simulada comienza a perderse. Este es un punto clave de los turbo códigos, y es frecuentemente referido en la literatura de la turbo codificación como el piso de la tasa de error de bit.

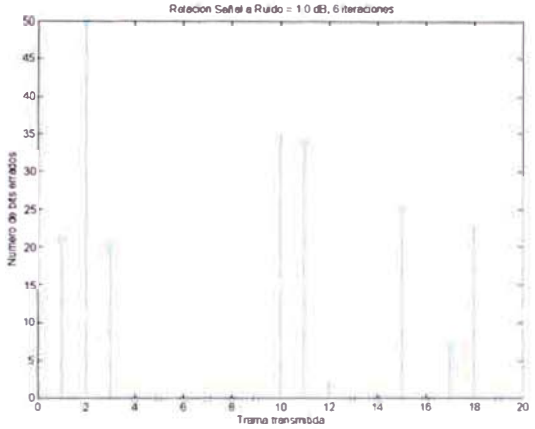
También se muestra en la Figura 3.3 la asíntota de distancia libre del código convolucional junto a la cota de la Ecuación 3.12 computadas sobre los ocho primeros términos de la sumatoria. Debido a la gran distancia libre del código convolucional, la asíntota es mucho más inclinada que para el turbo código. Sin embargo, debido a que los coeficientes precedentes a la función Q son mucho más pequeños para la asíntota de turbo código que para la de códigos convolucionales, la performance del turbo código es superior a niveles bajos de E_b/N_0. Sin embargo, a medida que E_b/N_0 crece, las dos asíntotas se acercan cada vez más hasta que finalmente se cruzan en $E_b/N_0 =$



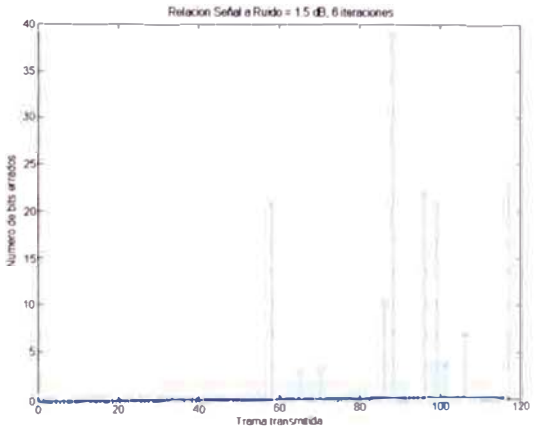
(a) Performance



(b) Nro. de bits errados para $S/N = 0{,}5$ dB



(c) Nro. de bits errados para $S/N = 1{,}0$ dB



(d) Nro. de bits errados para $S/N = 1{,}5$ dB

Figura 3.4: Bits errados por trama transmitida en función de la relación señal a ruido. Código de trama de 256 bits, códigos generadores $(13, 17)_8$, algoritmo Log-MAP

3,5. Esto implica que mientras los turbo códigos ofrecen performance superior (con respecto a los códigos convolucionales) a una baja SNR y una BER moderada, los códigos convolucionales podrían tener performances superiores a valores altos de SNR y valores bajos de BER.

3.4.1. Esquema de simulación

En esta sección se realizará la simulación de los turbo códigos en MATLAB 6.5 ®, para lo que se emplearán los programas del Apéndice. Los parámetros que se han fijado para las simulaciones son los siguientes:

1. Canal con ruido blanco aditivo Gaussiano (AWGN).

2. Modulación BPSK (Binary Phase Shift Keying).

3. Símbolos transmitidos +1/-1 correspondientes a los bit del código 1/0.

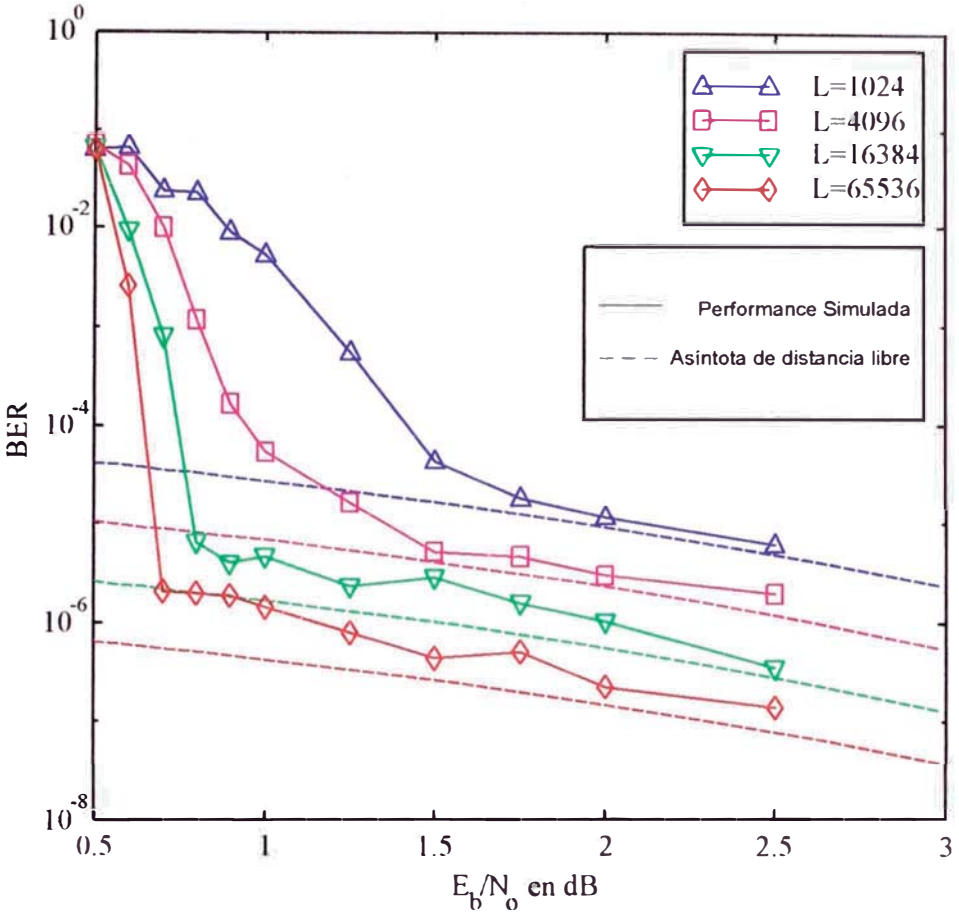


Figura 3.5: Performance asintótica y simulada de un turbo código de tasa $r = 1/2$ y longitud de contención $K_c = 5$ con varias medidas de entrelazadores
[Gráfico tomado de [33]]

4. Reseteo del codificador al estado 0 antes de codificar cada trama.

5. Los resultados se expresarán en valores de Probabilidad de Error de bit.

Para ilustrar el concepto de probabilidad de error de bit, se ha elaborado el ejemplo de la figura 3.4 en el que se muestran los resultados obtenidos con un código de trama $L = 256$ bits, polinomios generadores $(13, 17)_8$ y algoritmo Log-MAP. Se aprecia que para una relación señal a ruido de 0,5 dB, figura 3.4b, en la primera trama transmitida se producen 16 bits errados, en la segunda ninguno, en la tercera 32, etc, siendo el número total de bits errados 189 bits. Como se transmitieron para este valor de S/N, 17 tramas de 256 bits cada una (se considera una longitud efectiva de 253 bits, ya que los tres últimos bits son para la terminación trellis), se tiene, aplicando la siguiente fórmula

$$P_b = \frac{errs}{nframe(L\_total - m)}, \tag{3.15}$$

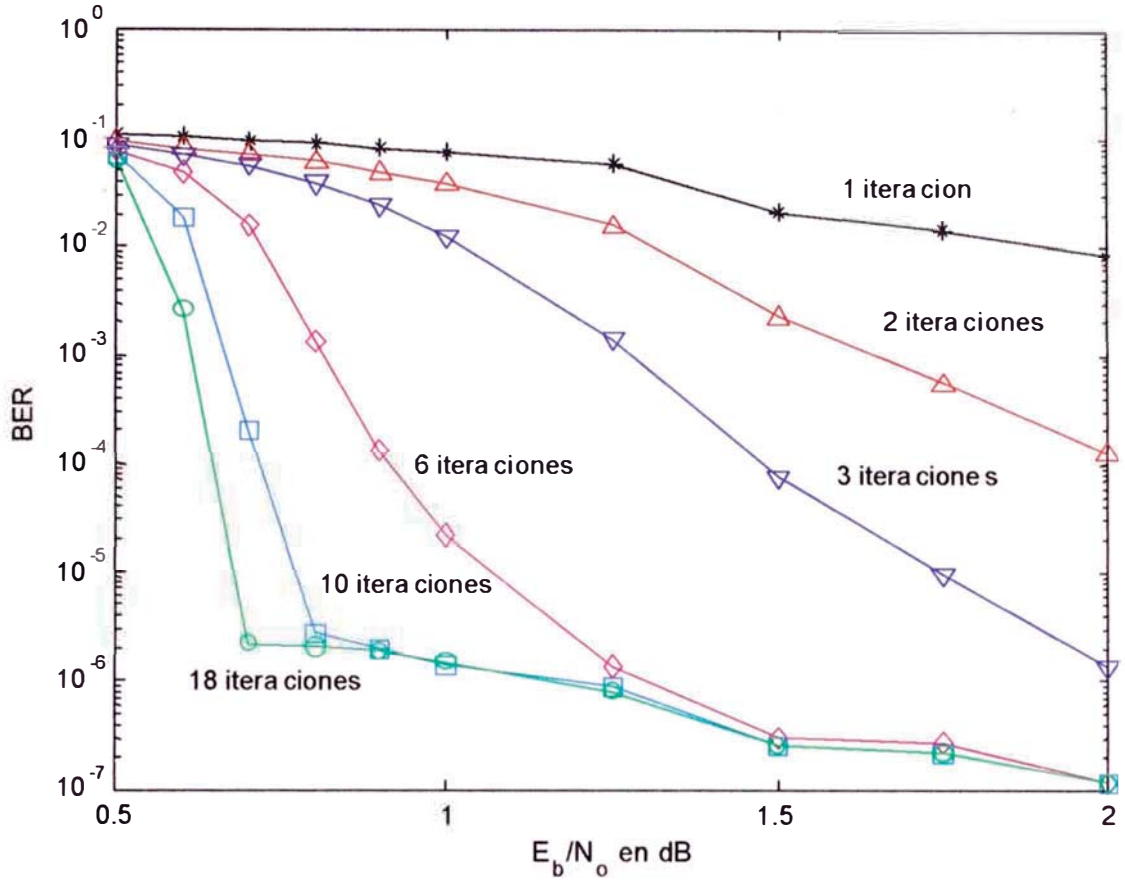


Figura 3.6: Performance del turbo código original en función del número de iteraciones

que la Probabilidad de error de bit es $P_b = 0{,}043943$, valor que coincide con el indicado en la figura 3.4a, para el mismo valor de relación señal a ruido. La ecuación 3.15 se ha elaborado con la misma notación empleada en los programas de simulación. Se tiene que *errs* es el número total de errores de la iteración presente, en este caso se aplicaron 6 iteraciones, $nframe$ es el número de tramas transmitidas, L_total es la longitud de la trama y m es el número de bits de terminación de trellis.

3.4.2. Influencia de la longitud de la trama

En la Ecuación 3.10, la longitud del mensaje k aparece en el denominador del coeficiente principal. La longitud del mensaje esta relacionada con la longitud del entrelazador por $k = L - K_c + 1$, y para entrelazadores suficientemente largos, $k \approx L$. Así, la BER asintótica de un turbo código es inversamente proporcional a la medida de su entrelazador. La asíntota de distancia libre de un turbo código puede ser de

esta manera, bajada, incrementando la medida del entrelazador, o inversamente, elevada, disminuyendo su medida. La Figura 3.5 muestra la performance asintótica y simulada para cuatro diferentes tamaños de entelazador: $L = 1024$, $L = 4096$, $L = 16,384$ y $L = 65,536$. Se ejecutan 18 iteraciones de decodificación log-MAP, y se fijan 40 palabras de código erróneas para cada valor de E_b/N_0. Nótese como la performance de un turbo código está fuertemente relacionada con el tamaño del entrelazador. De hecho, el tamaño del entrelazador es uno de los factores que más afectan la performance de los turbo códigos. Para una discusión de los puntos claves involucrados en la seleción del tamaño del entrelazador y como los turbo códigos pueden ser diseñados para satisfacer la calidad de servicio requerida en los sistemas inalámbricos multimedia, vea [32].

Cómo hacer, entonces para mejorar la performance en tramas cortas, o lo que es equivalente, reducir la probabilidad de error de bit debajo de los niveles obtenidos en la figura 3.5? Para resolver ésta pregunta es necesario analizar otros parámetros que influyen en el rendimiento de los turbo códigos, ya que en su diseño apropiado hay grandes posibilidades como se verá más adelante.

3.4.3. Influencia del número de iteraciones

Como ya se mencionó anteriormente el número de iteraciones es muy importante en el diseño de turbo códigos, ya que como se puede apreciar en la figura 3.6 la performance del turbo código original mejora en función de que aumenta el número de iteraciones.

Lo dicho se puede ilustrar mejor con un ejemplo. Para el caso de una longitud de trama larga $L = 20,049$, se puede apreciar en la figura 3.7 como varía el desempeño de la decodificación a medida que se incrementa el número de iteraciones. La imagen original, figura 3.7a es una imagen en blanco y negro puros, cuyos valores son un arreglo lógico de ceros y unos, contenidos en una matriz de $m \times n = 123 \times 163 = 20,049$ elementos, la cual se ha codificado con el algoritmo Log-MAP en un diferente número de iteraciones. Se puede apreciar que el número de errores en la figura va disminuyendo hasta llegar a la figura 3.7f, donde ya el error se convierte en cero.

En cuanto a turbo códigos de tramas cortas, se puede ver en la figura 3.8 que la tendencia es la misma. Se puede ver que de la iteración 1 a la 2 la mejora de la performance es notoria, sin embargo a medida que se aumenta el número de iteraciones, la mejora existe, pero va siendo cada vez más pequeña. De esta característica se desprende que ya no es conveniente aumentar el número de iteraciones a más de 10, ya

(a) Imagen Original

(b) Una iteración

(c) Dos iteraciones

(d) Tres iteraciones

(e) Cuatro iteraciones

(f) Cinco iteraciones

Figura 3.7: Performance en función del número de iteraciones. Código de trama de 20,049 bits, códigos generadores $(7,5)_8$, algoritmo Log-MAP y relación señal a ruido de 1,5 db

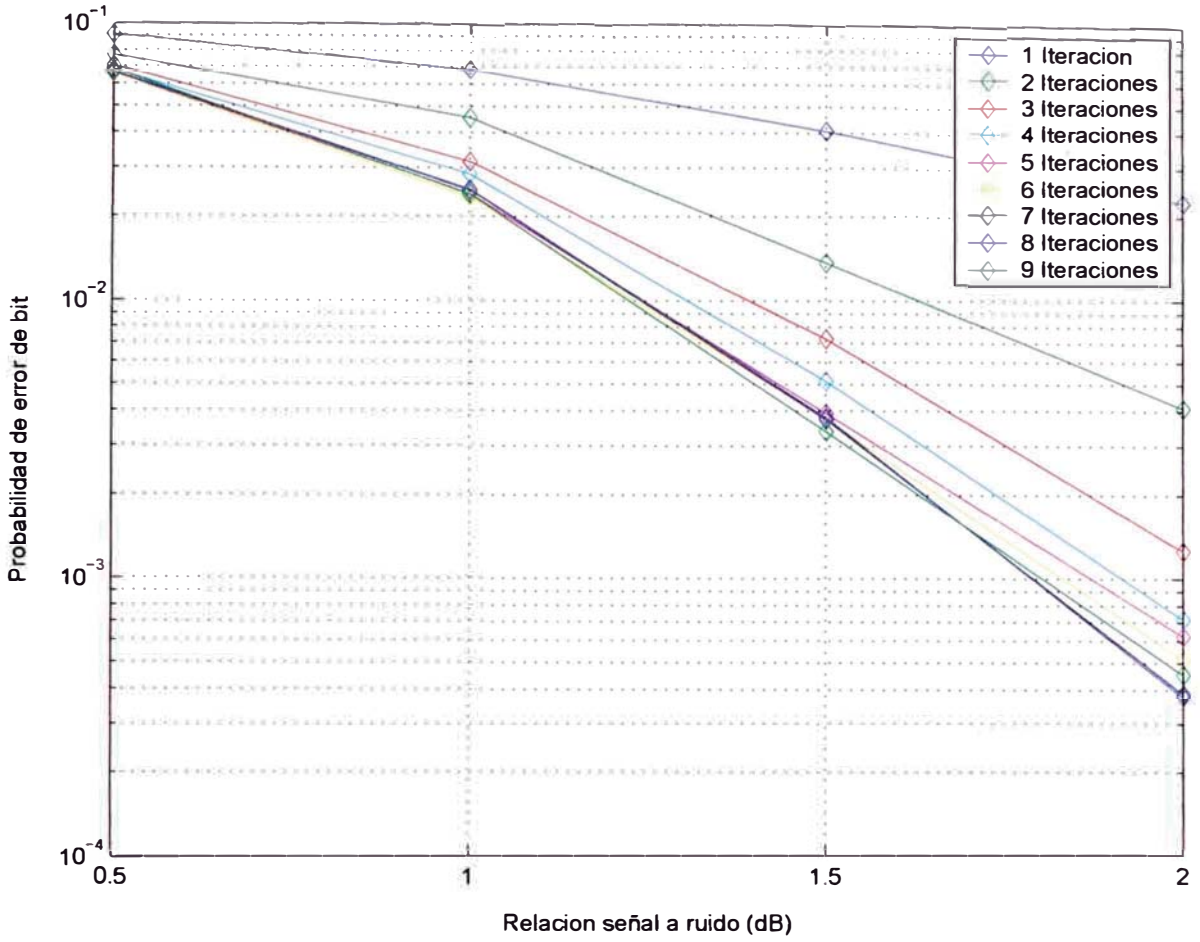


Figura 3.8: Performance en función del número de iteraciones, código de trama de 400 bits

que la mejora en el rendimiento(disminución de la probabilidad de error de bit) es muy pequeña, y sin embargo, el retardo adicional introducido en la comunicación por añadir más iteraciones es muy elevado.

3.4.4. Influencia de la tasa del código

Para estas simulaciones se emplearán tasas de código de $r = 1/2$ y $r = 1/3$. Como ya se mencionó anteriormente los códigos de tasa $r = 1/2$ se logran puncturando los bits de paridad de los codificadores constituyentes de los de tasa $r = 1/3$. La performance de los códigos de tasa $r = 1/3$ comparada a la de los códigos de tasa $r = 1/2$ se muestra en la figura 3.9, donde se puede apreciar que a medida que aumenta la tasa del código, la probabilidad de error de bit (BER) disminuye.

La Figura 3.9 muestra la performance simulada para dos diferentes tasas de código: $r = 1/2$ y $r = 1/3$. Se ejecutan 9 iteraciones de decodificación log-MAP, y

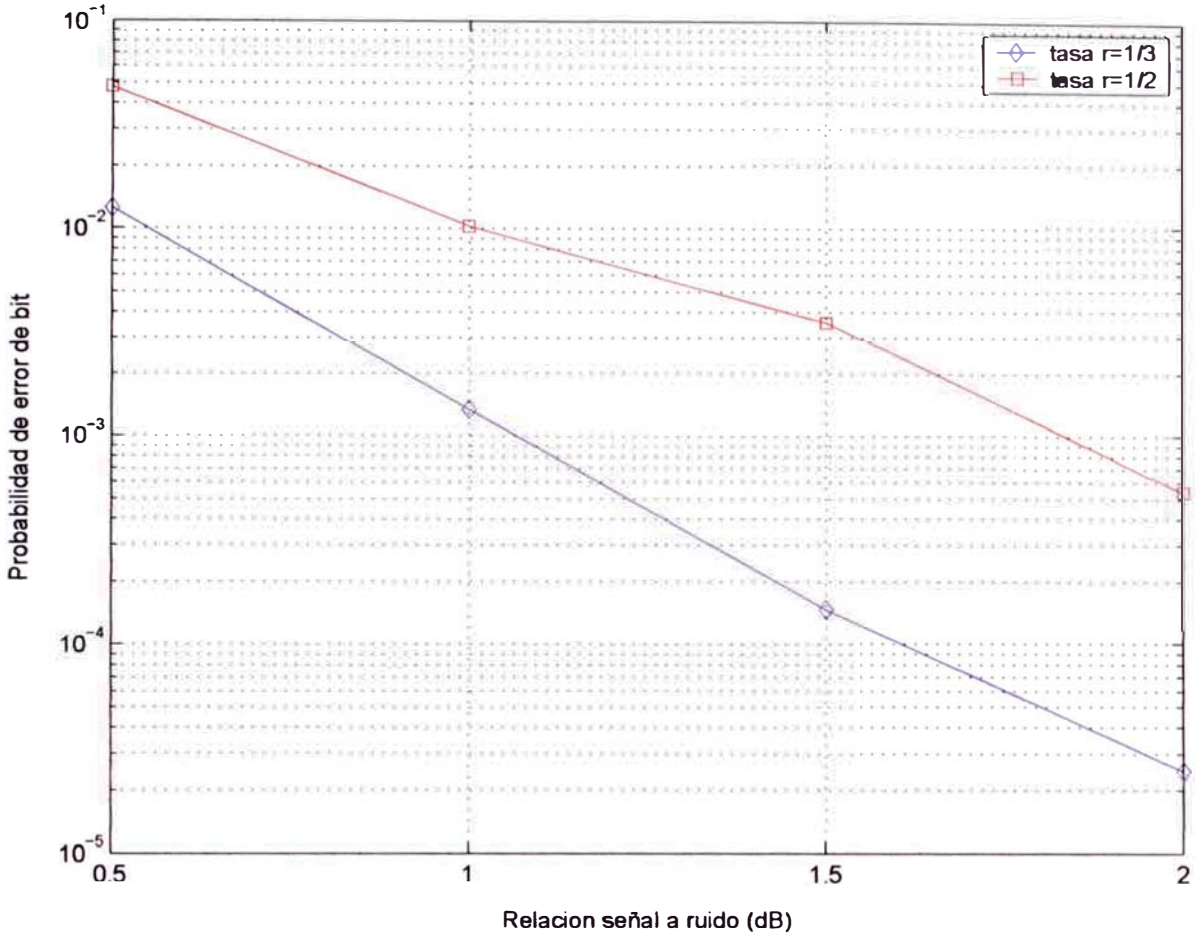


Figura 3.9: Performance en función de la tasa del código, código de trama de 400 bits

se fijan 15 palabras de código erróneas para cada valor de E_b/N_0

3.4.5. Influencia del código generador

En la figura 3.10 se puede apreciar que al aumentar la longitud de contención para iguales relaciones señal a ruido la probabilidad de error de bit (BER), disminuye. Se puede apreciar que cuando se aumenta la longitud de contención de $K_c = 3$ a $K_c = 4$ y de $K_c = 4$ a $K_c = 5$, mejora la performance de los turbo códigos, aunque en valores pequeños (aproximadamente 0.3 db). Sin embargo, la complejidad se incrementa en un factor de 2. Esto se debe a que al aumentar la longitud de contención, como se desprende de lo dicho en la sección 2.2, los bits de los que depende dicha salida aumentan también, por lo tanto el número de operaciones que se deben realizar tanto en el codificador como en el decodificador, se incrementan, proporcionándole al sistema una mayor carga de procesamiento y requiriendo, por tanto, un mayor

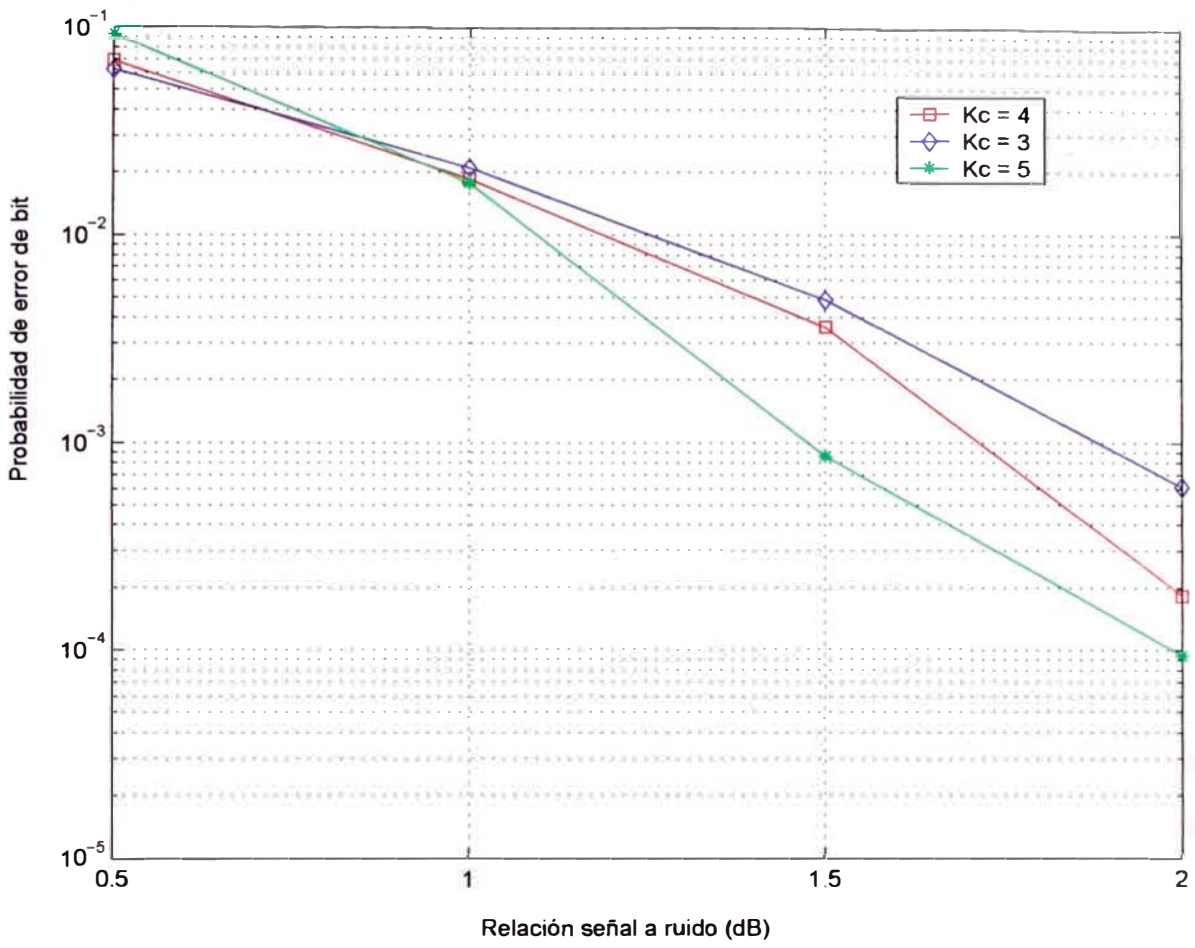


Figura 3.10: Performance en función de la longitud de contención, código de trama de 400 bits

tiempo para efectuar estas operaciones. Este incremento en la complejidad es de màs o menos dos veces entre $K_c = 3$ y $K_c = 4$, y entre $K_c = 4$ y $K_c = 5$, por lo que se recomienda que, dada la poca mejora lograda, la longitud de contención óptima sea de un valor máximo de $K_c = 4$.

Cabe señalar que las simulaciones se efectuaron con una longitud de trama $L = 400$, 9 iteraciones de decodificación Log-MAP y 15 palabras de código erróneas para cada valor de E_b/N_0.

3.4.6. Influencia del diseño del entrelazador

Se ha simulado el desempeño para diversos tipos de entrelazadores como se puede apreciar en la figura 3.11, donde se han empleado 9 iteraciones de decodificación Log-MAP y 15 palabras de código erróneas para cada valor de E_b/N_0. Se puede que la performance de los turbo códigos es afectada por el entrelazador empleado.

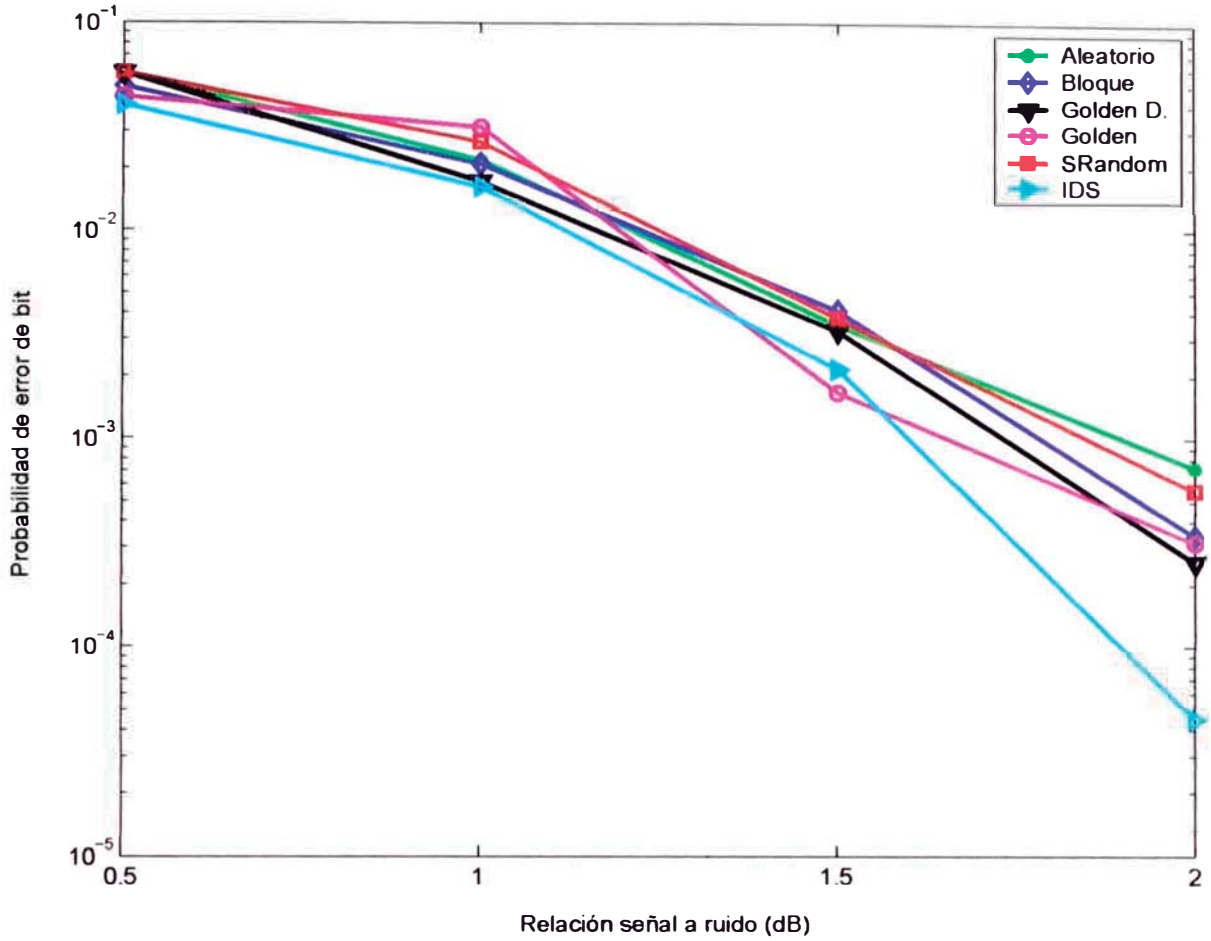


Figura 3.11: Performance en función del entrelazador, código de trama de 400 bits

El rendimiento del entrelazador y su contribución a la performance de los turbo códigos depende de dos aspectos fundamentales: el espectro de distancia del código y la correlación entre los datos de entrada y la salida suave de cada decodificador correspondiente a sus bits de paridad. En este sentido se pueden diseñar entrelazadores donde se maximizen estos dos aspectos, logrando, por lo tanto, un desempeño óptimo [17, 18].

Para el caso de los entralazadores para tramas largas la mayoría de los entrelazadores aleatorios funcionan adecuadamente. Esto concuerda con los resultados obtenidos por Berrou y su equipo de investigadores franceses [6]. Sin embargo para tramas cortas los entrelazadores determinísticos, diseñados especialmente para cada codificador, siguiendo para tal fin los parámetros mencionadas en el párrafo anterior, son los que han demostrado mejores resultados.

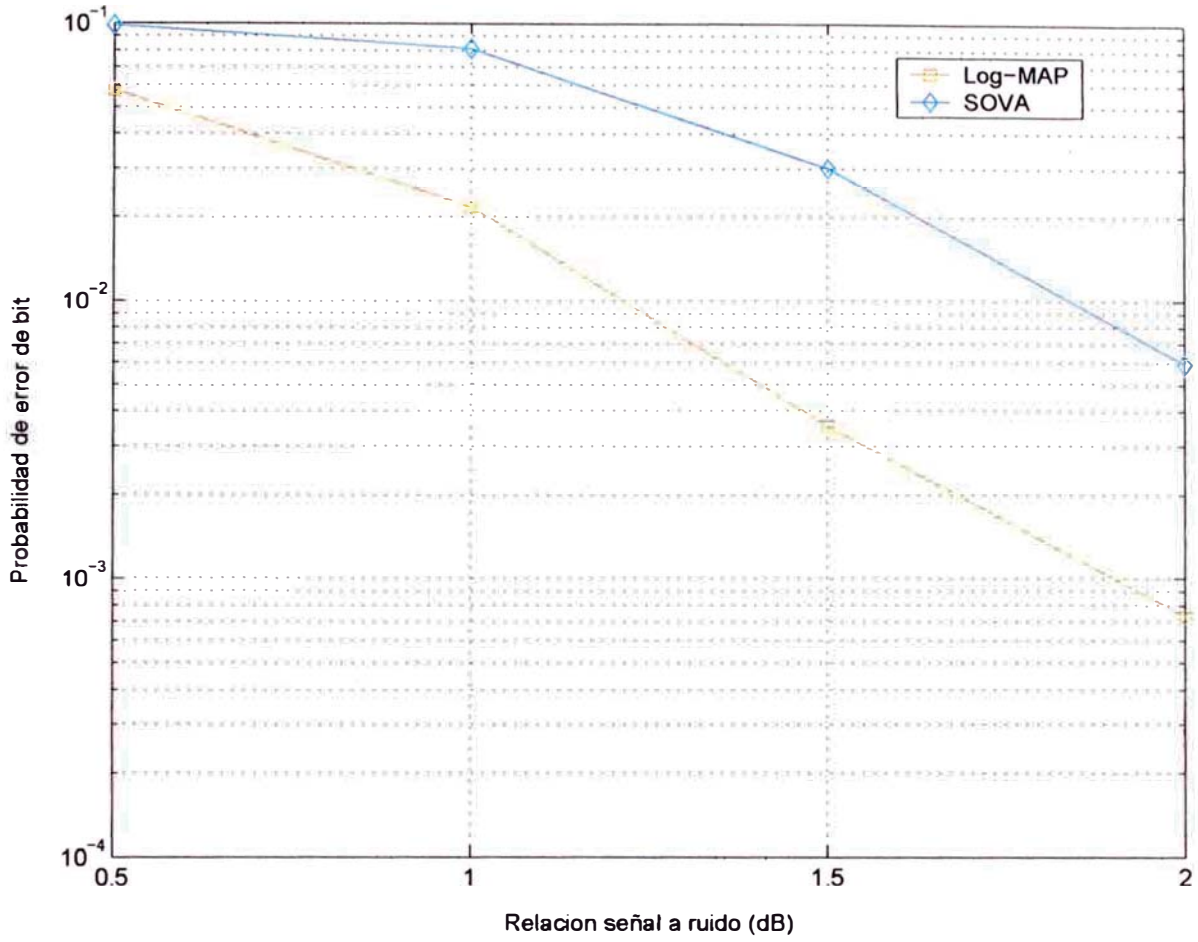


Figura 3.12: Performance en función del algoritmo de decodificación, código de trama de 400 bits

3.4.7. Comparación entre el algoritmo de decodificación SOVA y Log-Map

Se puede apreciar que para el algoritmo Log-Map se producen valores menores de probabilidad de error de bit, por tanto logrando una mejor performance como se puede apreciar en la figura 3.12. Para la simulación se han empleado 9 iteraciones y 15 palabras de código erróneas para cada valor de E_b/N_0, empleando para ambos casos códigos generadores $g^{(0)} = p^3 + p^2 + p + 1$ y $g^{(1)} = p^3 + p + 1$

Si bien el algoritmo SOVA tiene un desempeño inferior al Log-MAP, su implementación es más sencilla, y su tiempo de cálculo es menor, por lo que puede usarse para sistemas que requieren una mayor velocidad de transmisión y tengan un menor costo de implementación por consumir menos recursos computacionales.

Capítulo 4

Diseño de TC en tramas cortas

Como se dijo anteriormente el propósito de esta tesis es proponer un diseño eficiente de turbo códigos para sistemas de transmisón de tramas cortas. Por esta razón el análisis se centrará en los aspectos, vistos en el capítulo previo, que contribuyen a su desempeño, como son: el entrelazador y los códigos constitutivos. Ahí se vió que en la comparación de rendimiento en función del algoritmo empleado, Log-Map y SOVA, el primero fué el que mostró un mejor desempeño, por lo que éste será el que se considere para las simulaciones en este capítulo. Además se considerará una longitud de trama $L = 256$, $L = 500$ ó $L = 4096$, 9 iteraciones de decodificación, una tasa de código $r = 1/3$, y se empleará terminación de trellis en el primer codificador.

Para efecto de las simulaciones se ha empleado, como se dijo anteriormente, MATLAB 6.5 ®, utilizando como base de algunos de los archivos empleados, los desarrollados por Yufei Wu de la Universidad Virginia Tech [36], con modificaciones introducidas por el autor para efectos del empleo y selección de diversos entrelazadores y la presentación de los gráficos; y agregando otros.

4.1. Diseño del entrelazador

Como ya se vió en el capítulo anterior el entrelazador cumple la función de reordenar la secuencia de bits de información que salen del primer codificador y se dirigen al segundo. La combinación de estos dos codificadores y el entrelazador da una solución a los siguientes problemas:

1. La creación de códigos con buenas propiedades de distancia que,
2. pueden ser decodificados eficientemente con un algoritmo iterativo

El entrelazador es un componente clave de los turbo códigos, y su diseño es escencial para lograr un buen desempeño. En su diseño se debe considerar las propiedades de distancia, es decir se debe buscar que las palabras de código mantengan un peso de Hamming alto, como se vió en 3.2.1. Para ello existen métodos para calcular el espectro de distancia promedio como por ejemplo en [2, 3], empleando codificadores constitutivos específicos, y fijando la longitud del entrelazador en **N**

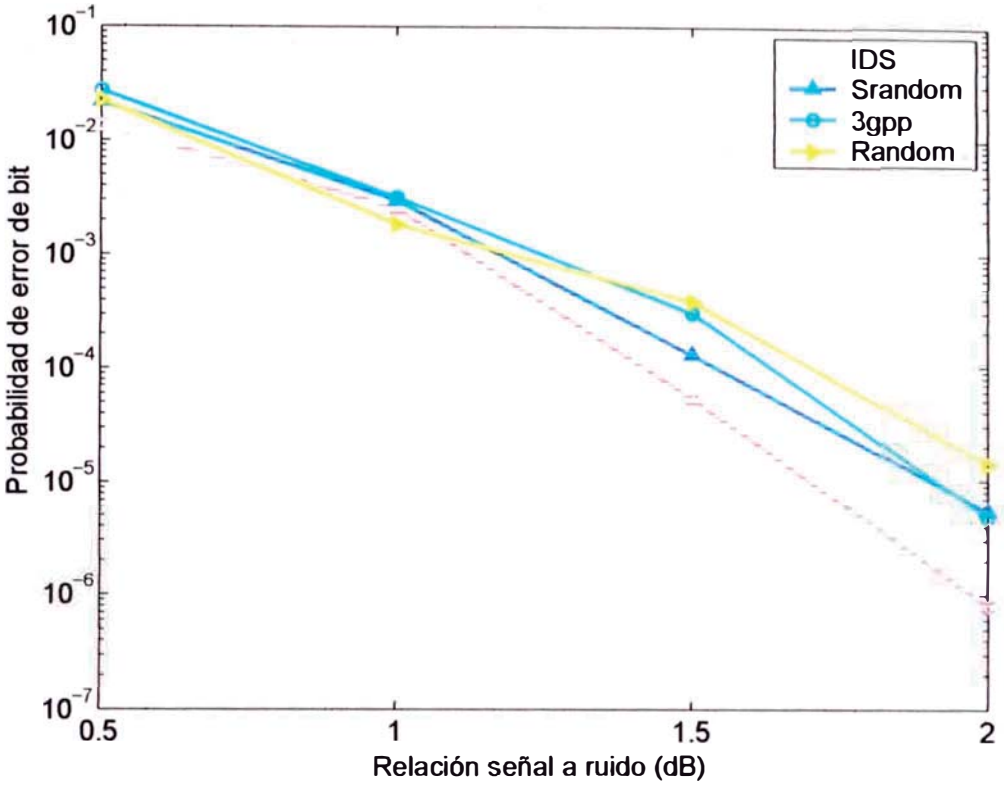


Figura 4.1: Performance en función del entrelazador, trama de 256 bits, con polinomios generadores $(15, 17)_8$

Otro criterio empleado para el diseño del entrelazador es aprovechando su capacidad de ser decodificado en forma iterativa, para lo que buscará disminuir lo máximo posible la correlación entre los datos de entrada y la salida suave de cada decodificador correspondiente a sus bits de paridad [17, 18].

Por lo tanto la mejor opción es emplear una combinación de ambos criterios para diseñar el entrelazador, por lo que se considera el algoritmo IDS (Iterative Decoding Suitability), propuesto por Johan Hokfelt de la Universidad de Lund, Suecia [17] que contiene los dos criterios mencionados y brinda el mejor desempeño.

En cuanto a la figura 4.1 se puede apreciar que para una longitud de trama $L = 256$, 9 iteraciones de decodificación Log-MAP y empleando un código generador $g^{(0)} = p^3 + p2 + 1$ y $g^{(1)} = p^3 + p^2 + p + 1$, o en forma octal $(15, 17)_8$, se puede ver que el entrelazador diseñado con el criterio IDS tiene una performance muy superior al resto.

En este algoritmo el entrelazador es descrito como un vector de longitud N, definido por los elementos $d(m), m = 1, \ldots, N$. Los elementos en éste vector son valores que se van asignando uno a uno, hasta que todo el vector quede definido. Para cada nuevo valor se escoge una nueva posición basada en los dos criterios

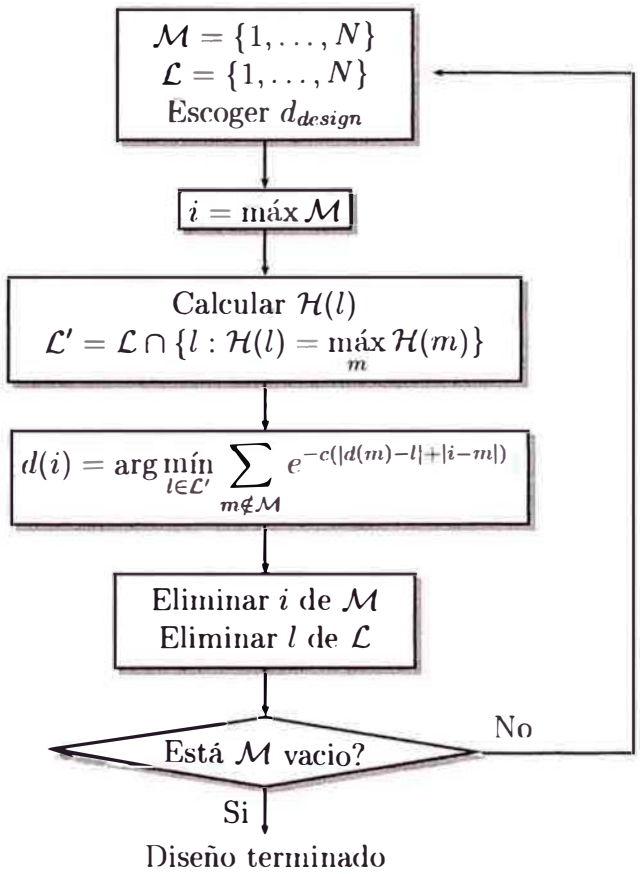


Figura 4.2: Diagrama de Flujo del algoritmo IDS para diseño de un entrelazador

2. Escoger una distancia de diseño d_{design}.

Loop de diseño

1. Escoger una posición $i \in \mathcal{M}$, para la cual se va a asignar $d(i)$. Con un diseño de orden inverso del entrelazador $i = \max \mathcal{M}$. $d(i)$ se escoge entre las posiciones $l \in \mathcal{L}$.

2. Calcular el menor peso de palabra de código que resulta de cada una de las elecciones $d(i) = l, l \in \mathcal{L}$, que toman pesos de Hamming menores o iguales a d_{design} considerado. Así, para cada posición $l \in \mathcal{L}$ existirá un mínimo peso de palabra de código en el código, dado que la elección $d(i) = l$ ha sido hecha. Este conjunto de pesos se almacena en el vector $\mathcal{H}$. Si para la elección $d(i) = l$ no se encuentra ninguna palabra de código de bajo peso ($\leq d_{design}$), se le asigna a $\mathcal{H}(l)$ un número grande (arbitrario, pero mayor que d_{design} e igual para todos estos valores de l).

mencionados, es decir el espectro de distancia y la correlación. En cuanto al espectro de distancia, el algoritmo está diseñado para eliminar mapeos del entrelazador que produzcan palabras de código con peso de Hamming menor o igual que cierto valor, llamado la *distancia de diseño*, denotada por d_{design}.

Como no existe una forma sencilla de ponderar la importancia del espectro de distancia del código y la performance de la decodificación iterativa, en el diseño del algoritmo se ha adoptado la siguiente estrategia:

para asignar un nuevo elemento del entrelazador, se debe escoger una posición con buenas propiedades de correlación entre las posiciones que satisfacen los requerimientos de espectro de distancia impuestos por d_{design}.

Los elementos en el vector del entrelazador se asignan en orden ascendente desde la posición 1 a la N.

A continuación se darán algunas definiciones de las variables empleadas en el algoritmo:

d: Es el vector que define el entrelazador. $\mathbf{d} = [d(1)d(2)\ldots d(N)]$, donde el m_{esimo} elemento encaja en la posición de entrada, $d(m)$, que es entrelazada a la m_{esima} posición en la secuencia entrelazada.

d_{design}: El máximo peso de Hamming de una palabra de código que es descartada por el algoritmo de diseño del entrelazador.

$\mathcal{M}$: El conjunto de posiciones en la secuencia entrelazada que todavía no están asociadas con una posición en la secuencia original. Es decir, las posiciones i para las cuales $d(i)$ no está definido todavía.

$\mathcal{L}$: El conjunto de posiciones en la secuencia original que todavía no han sido asignadas a las posiciones entrelazadas. Esta es la contraparte de $\mathcal{M}$.

$\mathcal{H}$: Un vector de longitud N que contiene los pesos de Hamming de las palabras de código de menor peso que resultan de las posibles elecciones de $d(i)$. En particular, $\mathcal{H}(l)$ almacena el peso de la palabra de código de menor peso que resulta de la elección $d(i) = l, l \in \mathcal{L}$.

Algoritmo:

Inicio

1. Sea $\mathcal{M} = \{1, \ldots, N\}$ y $\mathcal{L} = \{1, \ldots, N\}$.

3. Sea que las posiciones en $\mathcal{L}$ que brindan las palabras de código de mayor peso formen el conjunto $\mathcal{L}'$, así

$$\mathcal{L}' = \mathcal{L} \cap \{l : \mathcal{H}(l) = \max_{m} \mathcal{H}(m)\}. \tag{4.1}$$

4. Escoja la salida extrínseca del entrelazador $Le_i^{(2)}$ para la posición entre las posiciones en $\mathcal{L}'$ con la que tiene la menor correlación. Así, escoger $d(i)$ como

$$d(i) = \arg \min_{l \in \mathcal{L}'} \sum_{m \notin \mathcal{M}} e^{-c(|d(m)-l|+|i-m|)}. \tag{4.2}$$

   Si màs de una de las posiciones $l \in \mathcal{L}'$ tienen la misma correlación mínima, entonces escoger una aleatoriamente.

5. Borrar i y $d(i)$ de $\mathcal{M}$ y $\mathcal{L}$, respectivamente.

6. Si $\mathcal{M}$ no está vacio, ir a 1.

Este proceso de diseño se ilustra en el diagrama de flujo de la figura 4.2

4.2. Selección de los códigos constituyentes

En el caso de los entrelazadores se dijo que existían dos propiedades fundamentales que gobiernan el desempeño de los turbo códigos: el espectro de distancia y la performance de la decodificación iterativa. Para el caso de selección de los códigos constituyentes también se emplearán estos mismos criterios.

En este estudio se pretende indicar las propiedades que deben considerarse en el diseño de los códigos constituyentes de los turbo códigos, que a diferencia de los entrelazadores, que pueden ser diseñados para poseer buenas propiedades de distancia de Hamming y de correlación simultáneamente, existe un compromiso entre éstas dos características cuando se trata de elegir los códigos constituyentes, ya que ambas no son compatibles.

En cuanto al espectro de distancia de un turbo código, este es influenciado en gran manera por el período del código generador, como se puede apreciar en la figura 4.3, donde se han emplado códigos de tasa 1/3 con un entrelazador aleatorio de 4096 bits, y diferentes codificadores con diferentes periodos de realimentación L, pero todos con $K_c = 5$. En general, cuando se emplean entrelazadores aleatorios, mientras mayor sea el período, mejor será el espectro de distancia del turbo código.

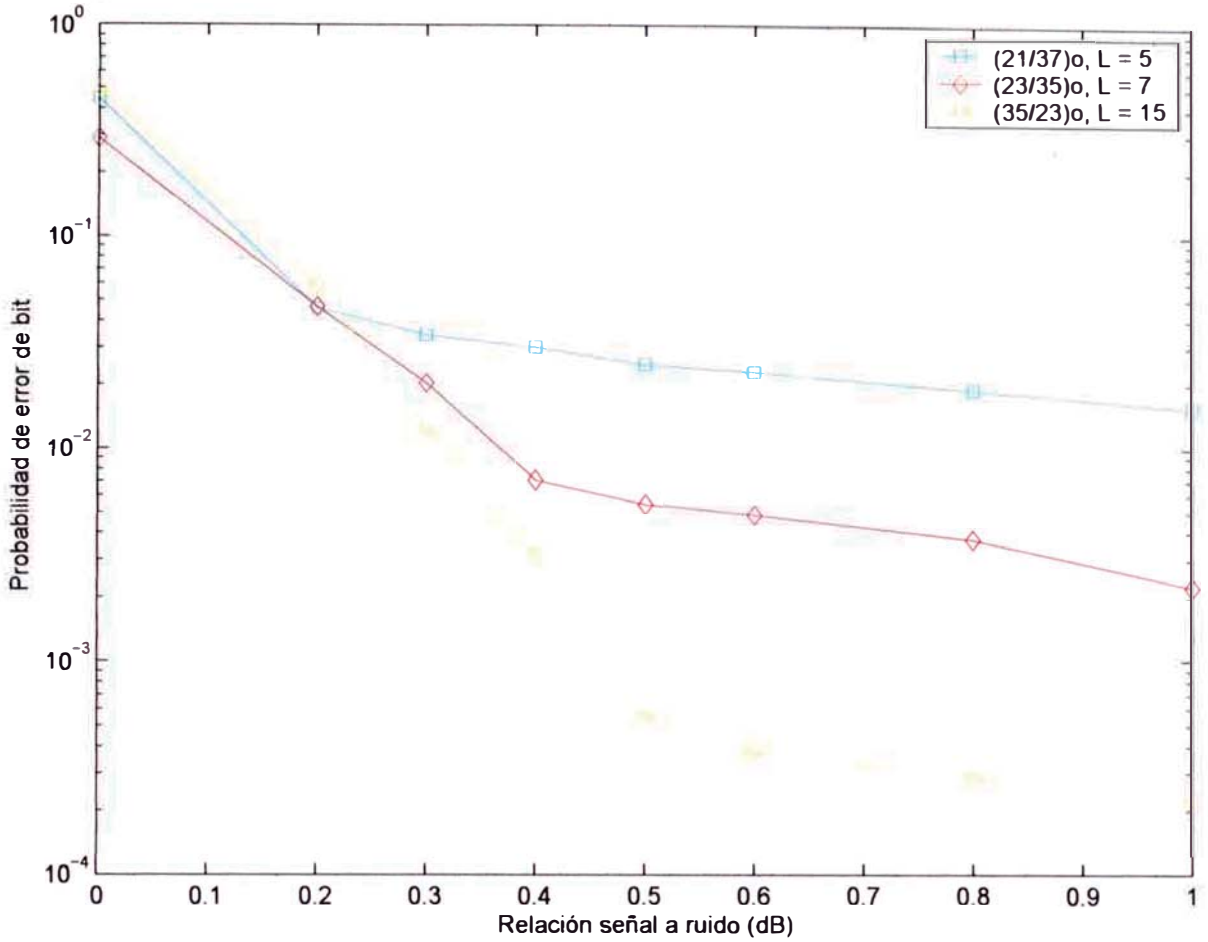
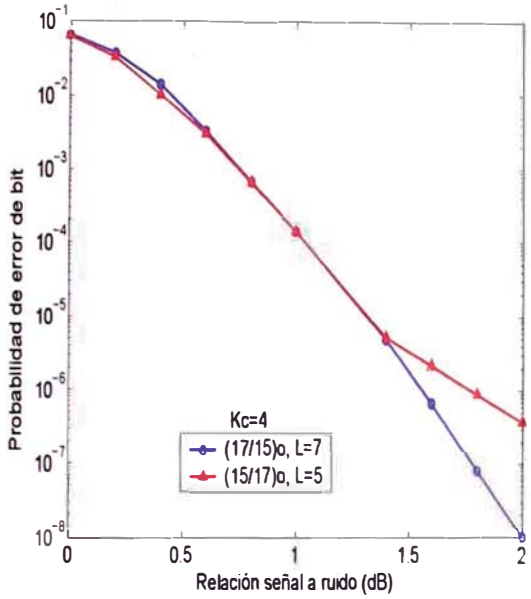


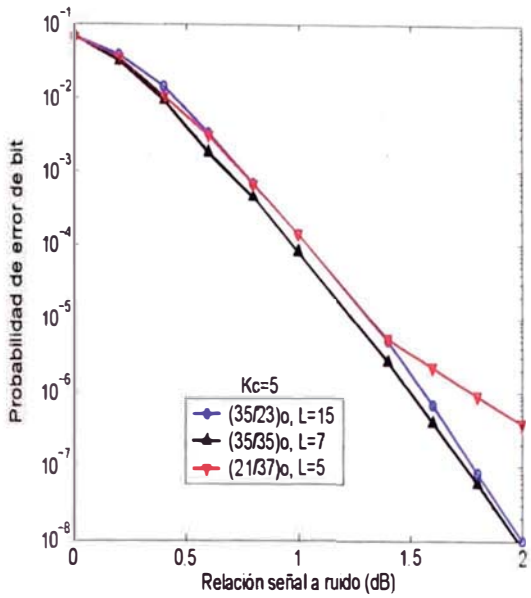
Figura 4.3: Performance simulada de turbo códigos de tasa $=1/3$ y longitud de trama de 4096 bits, con entrelazador aleatorio, que emplea polinomios generadores con diversos periodos de realimentación L, pero la misma longitud de contención $K_c = 5$

Específicamente, los parámetros que son directamente influenciados por la longitud de este período son la *distancia mínima*, denotada por d_{min} y la *distancia libre mínima*, denotada por d_{free}, como se vió en 2.2.2.

En la figura 4.3 se puede apreciar, que a pesar de tener todos los polinomios generadores la misma longitud de contención $K_c = 5$, la longitud del período de realimentación marca la diferencia en la performance cuando se emplean entrelazadores aleatorios. Por ésta razón, es razonable escoger polinomios de realimentación con el período lo más largo que sea posible, para lo cual se deberían emplear polinomios primitivos. Sin embargo cuando se emplean entrelazadores diseñados específicamente, el período de realimentación deja de ser tan importante. Con el adecuado diseño del entrelazador, el espectro de distancia se puede hacer lo suficientemente bueno como para permitir el empleo de polinomios de realimentación con períodos



(a) Código con $K_c = 4$



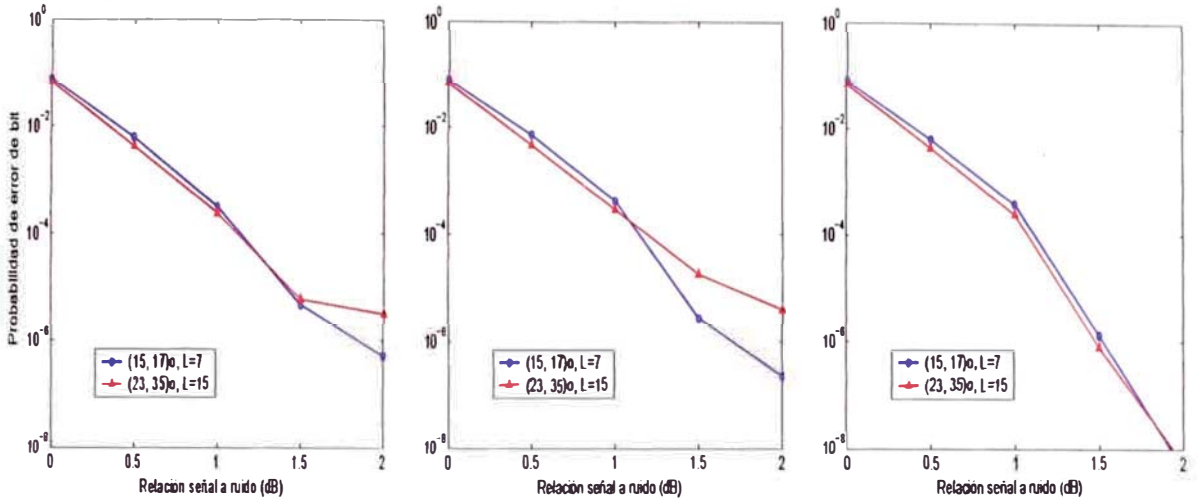
(b) Código con $K_c = 5$

Figura 4.4: Performance simulada de turbo códigos de tasa =1/3 con longitud de contención 4.4a $K_c = 4$ y 4.4b $K_c = 5$. La longitud del entrelazador es de 500 bits y cada entrelazador ha sido diseñado con el algoritmo IDS.

más cortos. Por lo tanto debe existir otro criterio para mejorar la performance de los códigos constitutivos, que se comentará a continuación.

Este criterio es el de la performance de la decodificación iterativa, ya que a medida que avanza el número de iteraciones en la decodificación iterativa aumenta la dependencia estadística entre las salidas y las entradas del entrelazador. Empleando este critero el enfoque estará centrado en la elección de los polinomios de realimentación, ya que ellos son los que ejercen mayor influencia en la performance de la decodificación iterativa.

En la figura 4.4 se puede apreciar que a diferencia del caso cuando se emplean entrelazadores aleatorios, que se pudo apreciar en las figuras 3.10 y 4.3, la longitud de contención K_c y aún la longitud del período de realimentación L, dejan de ser ya elementos fundamentales cuando se trata de entrelazadores especialemente diseñados. Se puede apreciar que en la figura 4.4a, para una misma longitud de contención $K_c = 4$, existe una variación en el desempeño cuando se varia el valor de L. Con un período de realimentación $L = 7$ el rendimiento es mejor que con un período $L = 5$, y además éstos dos tienen rendimientos casi iguales a los correspondientes períodos L en la figura 4.4b, a pesar de que en aquella $K_c = 5$. Además se puede ver que el mejor desempeño en esta última figura, no corresponde al período L de mayor valor, que produce un polinomio de realimentación primitivo, 15 en este caso, sino a un



(a) Golden (b) Aleatorio (c) IDS

Figura 4.5: Performance simulada de turbo códigos de tasa =1/3 con entrelazadores de 500 bits y longitud de contención $K_c = 4$ y $K_c = 5$, con polinomios de realimentación primitivos. 4.5a Entrelazador Golden, 4.5b entrelazador aleatorio y 4.5c entrelazador IDS.

periodo más corto, asociado a un polinomio no primitivo.

Finalmente, hay que enfatizar la importancia de emplear entrelazadores particularmente diseñados conjuntamente con codificadores constituyentes con períodos cortos. Para este propósito se comparan dos entrelazadores adicionales: los entrelazadores aleatorios y los entrelazadores golden. En estos casos particulares, los entrelazadores aleatorios no tienen un buen espectro de distancia, ni tampoco buenas propiedades de correlación. Por otro lado, los entrelazadores golden tienen muy buenas propiedades de correlación, pero sufren de grandes multiplicidades ocasionadas por palabras de código de relativo bajo peso. La figura 4.5 muestra la performance en función a la probabilidad de error de bit, empleando polinomios generadores $(23, 35)_8$ y $(15, 17)_8$ y un entrelazador de 500 bits. Como se ve, para los entrelazadores aleatorios y golden el uso de codificadores con $K_c = 4$ proporciona una mejor performance para las probabilidades de error de bit mayores a 10^{-4}, mientras que el límite correspondiente al entrelazador especialmente diseñado está en el orden de 10^{-7} en términos de probabilidad de error de bit. Además, como se intuyó, los beneficios de utilizar un polinomio de realimentación con periodo corto son válidos para un mayor rango de probabilidades de error cuando se emplean entrelazadores especialmente diseñados.

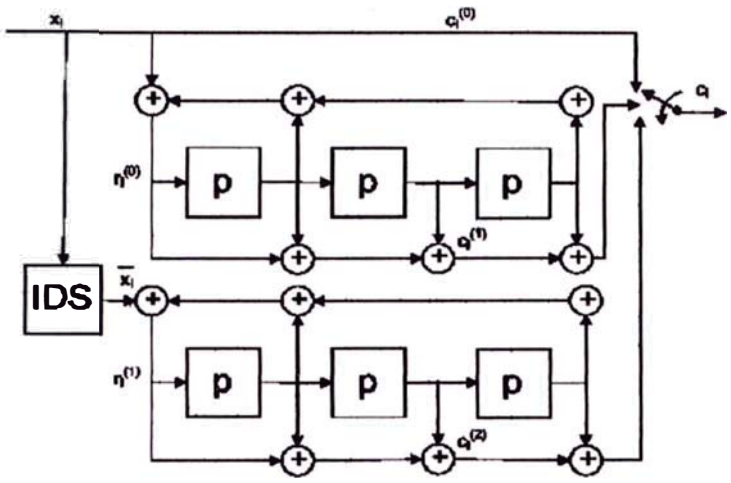


Figura 4.6: Modelo de esquema de codificación, que emplea códigos generadores $(15, 17)_8$, tasa $=1/3$, y un entrelazador especialmente diseñado con el algoritmo IDS

4.3. Rendimiento del nuevo esquema propuesto para tramas cortas

Dados los resultados obtenidos en las secciones precedentes se ha podido determinar la mejor configuración de esquema de codificación para tramas cortas, empleando turbo códigos, que es el siguiente:

1. Códigos constituyentes de tasa $r = 1/3$, dado que su rendimiento es mejor que cuando se punctura y se tiene tasa $r = 1/2$. La longitud de contención fijada máximo en $K_c = 4$, ya que no se aprecia una mejora notoria en el rendimiento como para aumentar la complejidad por su incremento, como se apreciar en la figura 4.5c, donde esta longitud de contención corresponde al ploteo en color rojo. Para esta caso los polinomios generadores son $(15, 17)_8$, pudiendo ser otros no primitivos, dado como se vió para tramas cortas y valores bajos de la relación señal a ruido, el hecho de que sea primitivo ya no es conveniente, sino al contrario, siendo en esta región del rendimiento más importante la construcción del entrelazador.

2. El algoritmo a emplear se recomienda sea el Log-MAP, dado que es el que posee mejor rendimiento y no es de gran complejidad. Sin embargo si se desea una menor retardo y una mayor rapidez en la respuesta, y la necesidad de contar con una muy baja probabilidad de error de bit (BER) no es tan apremiante, se debería emplear el algoritmo SOVA, ya que su demanda de recursos computacionales es mucho menor, por tanto es más rápido y menos

oneroso para su implementación. La comparación entre ambos se realizó en 3.4.7 y se muestra nuevamente en 4.7.

3. El entrelazador debe ser uno diseñado específicamente como puede comprobarse a partir de la figura 4.5, donde el rendimiento del esquema, cuando se emplea este tipo de componente, es mucho mejor que el de los otros con los que se comparó, en este caso el aleatorio y el golden, pero como se vió en la figura 4.1 es superior a otros. Para su diseño debe tenerse en consideración dos aspectos claves: el espectro de distancia de las palabras de código, es decir que mantengan un peso de Hamming alto, antes y después de ser codificadas; y que reduzcan al máximo la correlación entre los datos de entrada y la salida suave de cada decodificador correspondiente a sus bits de paridad. Por lo tanto el mejor procedimiento de creación de entrelazadores es el IDS de J.Hokfelt [17].

4. Otras consideraciones son ejecutar un máximo de 9 ó 10 iteraciones, ya que al aumentar este número, la mejora en el rendimiento es muy poca comparada con el gran aumento en el retardo y la longitud de trama aplicable para éstos resultados es de máximo de 1000 bits. si se tratara de una longitud mayor habría que pensar en usar otro tipo de entrelazador, ya como se vió anteriormente, los aleatorios o seudoaleatorios se desempeñan muy bien.

En La figura 4.1 se puede apreciar la performance del sistema propuesto empleando una trama de 256 bits, polinomios generadores $(15, 17)_8$, con una longitud de contención $K_c = 4$, que emplea el entrelazor especialmente diseñado, viéndose que es superior a esquemas similares que emplean otros entrelazadores. Un modelo de este esquema, cuya gráfica de BER versus S/N se analizó en la figura 4.5, comparándola con esquemas similares en los que se variaba el entrelazador y el periodo de realimentación L del código generador, es el que se muestra en la figura 4.6. Aquí, la combinación de los componentes mencionados en los párrafos anteriores logran este buen desempeño.

Finalmente en la figura 4.7 se hace una comparación doble del desempeño de este esquema. Primeramente es comparado variando únicamente la tasa del código r. En este caso se hace evidente que la performance correspondiente a una tasa 1/2 es inferior a la de tasa 1/3, como ya se había visto en 3.4.4. Sin embargo hay que resaltar que la performance correspondiente a esta tasa, empleando el esquema propuesto, es superior a la que corresponde cuando se emplea otro entrelazador,

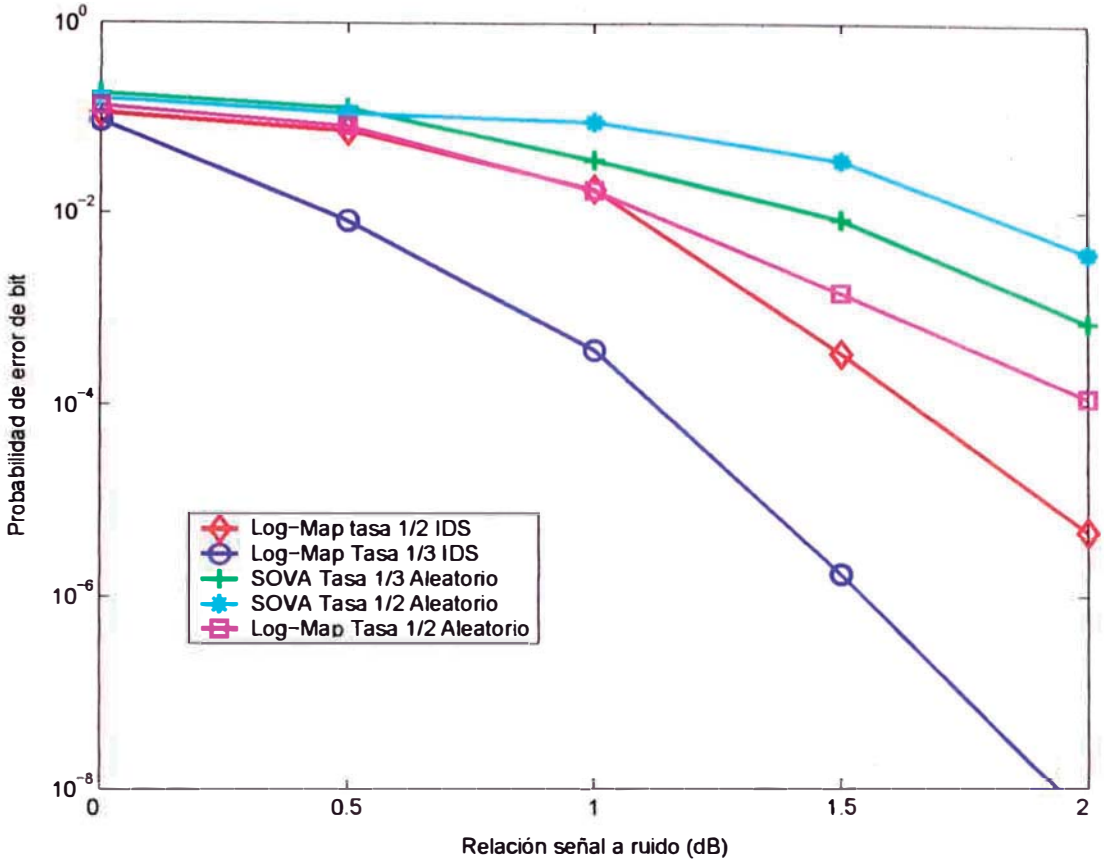


Figura 4.7: Performance simulada de turbo códigos de longitud de trama de 500 bits, con códigos generadores $(15, 17)_8$, analizada en función del algoritmo de codificación y de la tasa del código

como por ejemplo en este caso, el aleatorio; por lo tanto este esquema es válido para tasas de código mayores a 1/3. Asimismo, en segundo lugar, se muestra que el desempeño del algoritmo SOVA es bastante inferior. Para este caso se emplearon 16 iteraciones de codificación, ya que por requerir menos recursos computacionales, se puede aumentar el número de iteraciones sin superar el tiempo de proceso requerido por el algoritmo Log-Map. Pero a pesar del incremento en el número de iteraciones su rendimiento sigue siendo menor. Sin embargo, el algoritmo SOVA es apropiado cuando se requiere una mayor rapidez en la codificación de la señal, un menor costo para su implementación y no existe una demanda demasiado grande de confiabilidad.

Capítulo 5

Conclusiones

El propósito de esta tesis fue doble. Primero dar una introducción general a la teoría de códigos y a los turbo códigos específicamente, explicando las características que influyen en su rendimiento y; en segundo lugar, proponer un esquema de codificación eficiente para Sistemas de Transmisión de Tramas Cortas.

En cuanto al primer objetivo, sobre la teoría de códigos se puede decir que todavía es posible seguir mejorando el rendimiento de los esquemas de codificación, acercándolo aún más a los límites de capacidad de canal predichos por Shannon [30]. Los turbo códigos originales demostraron estar a sólo 0.7 dB del límite de Shannon, derribando el mito que ya no era posible seguir mejorando el rendimiento de los códigos prácticos, que se tenía hasta ese entonces.

Esta performance superior de los turbo códigos es alcanzada sólo cuando la longitud del entrelazar, o equivalentemente la trama, es muy larga, en el orden de varios miles de bits. Cuando se trata de turbo códigos que emplean tramas cortas (en general por debajo de los 1000 bits), se pudo ver que este rendimiento tan admirable decae debido principalmente a las características del entrelazador y los códigos constituyentes, que son los componentes principales y los que se han evaluado principalmente en esta tesis. En segundo lugar se tiene la terminación de trellis y el puncturado, de los que se ha brindado su concepción teórica.

Una de las mayores revoluciones en cuanto a la introducción de los turbo códigos es el proceso de decodificación iterativa, por medio de la cual dos decodificadores constituyentes se turnan para decodificar el mensaje recibido. En cada nuevo intento cada decodificador emplea la salida del otro decodificador. Si es adecuadamente formulado y con la suficiente relación señal a ruido, el proceso de decodificación iterativa es exitoso, y logra refinar las salidas del decodificador hasta que los dos codificadores convergen en una misma decisión.

La razón por la cual se emplea la decodificación iterativa se asocia a la todavía grande complejidad de emplear decodificación Maximum Likelihood (ML). La decodificación iterativa provee una decodificación eficiente de un código complejo a costa de ser un proceso sub óptimo comparado con la decodificación Maximum Likelihood.

En cuanto al diseño de turbo códigos para tramas cortas, se ha visto que para

mejorar su rendimiento se debe proporcionar el mejor espectro de distancia posible. Esta característica unida al hecho de que se emplee un algoritmo de decodificación iterativo, que es sub óptimo, y que por tanto, deba mantener en el valor más bajo posible la correlación entre los datos de entrada y la salida suave de cada decodificador correspondiente a sus bits de paridad, hizo que éstos dos criterios sean los principales que se consideraron para su diseño.

Empleando los dos criterios señalados anteriormente, que son: el espectro de distancia y la correlación entre las entradas del decodificador y la salida suave de cada decodificador correspondiente a sus bits de paridad; se pudo determinar que éstos son fuertemente influenciados por la elección del entrelazador. Esto deja por sentado que no es necesario el empleo de entrelazadores aleatorios, sino por el contrario los diseñados especialmente para satisfacer estos dos criterios, son los más eficientes.

También se revisó la elección de los códigos constituyentes y se determinó que uno de los parámetros más influyentes de los codificadores convolucionales recursivos es el período del polinomio de realimentación. Nuevamente el espectro de distancia y la performance de la decodificación iterativa son influenciados por éste parámetro. Aunque para relaciones señal a ruido altas conviene un período alto del polinomio de realimentación, para bajas relaciones señal a ruido, conviene mas bien que el período sea corto y se cuente con un entrelazador especialmente diseñado.

Se vió que el algoritmo Log-Map tiene un desempeño superior al SOVA, pero su demanda de recursos computacionales es mucho mayor, por lo que su procesamiento es más lento. Por esta causa el algoritmo SOVA puede emplearse en sistemas que requieran una mayor rapidez de transmisión y no necesiten una muy alta confibialidad. Debido a esta gran velocidad de procesamiento es posible aumentar el número de iteraciones (se evaluó con 16) sin superar el tiempo requerido por un esquema similar que emplee el algoritmo Log-Map. Se vió también, que el algoritmo SOVA responde mejor cuando se emplea un entrelazador aleatorio.

Se probó también el esquema propuesto, con el algoritmo Log-Map, para la tasa $r = 1/2$, es decir, puncturando uno de los bits de la salida de paridad. En este caso se pudo apreciar que su rendimiento sigue siendo superior a otros esquemas que trabajan con esta misma tasa y emplean otros modelos de entrelazador.

Finalmente se puede concluir que los turbo códigos, por su gran performance son los códigos que están ya marcando la pauta en los sistemas de comunicacciones de tercera y de cuarta generación y con aplicaciones en muchas ramas de la ciencia, por lo que entenderlos y estar listos para emplearlos en sistemas prácticos de

comunicación es una tarea urgente.

Capítulo 6

Trabajos Futuros

Los entrelazadores todavía se pueden mejorar siguiendo el criterio de optimizar:

1. El espectro de distancia.
2. La correlación entre las entradas del decodificador y la salida de cada decodificador correspondiente a sus bits de paridad

Estudiar la factibilidad de emplear esquemas de codificación ARQ para turbo códigos.

Estudiar, asimismo, la factibilidad de enlazar este código con un código CRC, de tal manera que éste último se coloque en serie con el turbo código y que todo este código concatenado sea el que se transmita por el canal. En el receptor se realizaría el proceso inverso, decodificándose primero el CRC y si, habrían más errores de los que puede corregir éste, pasaría al decodificador Turbo. Si los errores se pueden corregir en la primera etapa, ya no haría falta de decodificarlo en la segunda, con el consecuente ahorro de tiempo de procesamiento. En este último caso, en el receptor, se recuperaría la información del tren de datos de la salida sistemática del codificador que estaría inalterado.

Asismo, emplear este modelo de codificación en un sistema de comunicaciones de datos en las bandas de V/UHF, con aplicaciones en comunicaciones rurales o marinas, e incluso empleando el electrochorro ecuatorial para lograr alcances de propagación más allá del horizonte.

Apéndice A

Simulación del Software

Las simulaciones se han realizado en MATLAB 6.5 ®, empleando los programas que se encuentran en el Apéndice B. A continuación se enumeran las funciones de MATLAB empleadas que permiten simular los Turbo Códigos clásicos:

- **bin_state.m**: Utilidad para convertir un vector de enteros en una matriz; la i-esima fila es la forma binaria de m bits para el i-esimo entero.
- **demultiplex.m**: Demultiplexion serie a paralelo para obtener la palabra de codigo de cada codificador en el terminal del receptor.
- **encode_bit.m**: Dos codificadores (Convolucionales Recursivos Sistemáticos) CRS para codificar las tramas de bits de información. Un sólo bit. Esta función tiene como entrada un solo bit a ser codificado, asi como los coeficientes del polinomio generador y el actual vector de estado. Retorna como salida n bits de data codificados, donde 1/n es la tasa del código. La tasa es $1/n$. Donde k es la longitud de contención y m es la cantidad de memoria.
- **encoderm.m**: Proceso de turbo codificación. Emplea el mapa del entrelazador álpha'. Si puncture = 1, no puncturado, produce una salida de tasa 1/3 de longitud fija. Si puncture = 0, puncturado, produce una salida de tasa 1/2. El multiplexor escoge bits de verificación impares del RSC1 y bits de verificación pares del RSC2.
- **int_state.m**: Utilidad que convierte un vector fila de m bits en un entero (base 10).
- **logmapo.m**: Algoritmo Log_MAP empleando método straightforward para calcular las métricas de las ramas. No se emplea aproximación. Se puede simplificar a Max-Log-MAP usando la aproximación $ln(e^x + e^y) = \max(x, y)$.
- **rsc_encode.m**: Codifica un bloque de datos x (0/1) empleando un codificador CRS con vectores generadoresa en g, y retorna la salida en y (0/1). Si ed>0, el trellis es terminado perfectamente y si ed<0, se deja no terminado;

- **sova0.m**: Decodificador componente SOVA (Soft Output Viterbi Algorithm).
- **trellis.m**: Establece el trellis para un generador de código dado g en la forma de matriz binaria. Por ejemplo g = [1 1 1; 1 0 1].
- **turbo_sys_demo.m**: Simulación del sistema de turbo codificación y decodificación. Función principal. Aquí inicia el programa.
- **back_block.m**: Entrelazador de bloque reverso.
- **Entrelazador_3gpp.m**: Entrelazador de los sist. de 3ra. generación.
- **odd_even_interleaver.m**: Entrelazador odd_even (par impar).
- **goldenint.m**: entrelazador golden y golden dithered.

Se pueden seleccionar los siguientes parámetros:

- Algoritmo de decodificación (Log-MAP / SOVA)
- Longitud de la trama
- Generador del código (en forma vectorial binaria)
- puncturado / no puncturado
- Entrelazador
- número máximo de iteraciones por cada trama
- número de tramas erradas para terminar
- relación señal a ruido (Eb / No)

Apéndice B

Listado de Programas en Matlab

Programas principales

Obtenidos de internet de [36]

turbo_sys_demo.m

Este archivo ha sido modificado de su versión original, añadiendole varias funcionalidades:

1. La posibilidad de seleccíonar entre diversos tipos de entrelazadores y realizar los calculos y graficas con cada uno de ellos.

2. Los valores de relación señal a ruido han sido modificados.

3. La gráfica del número de bits errados por trama transmitida de la iteracion final para cada uno de los valores de la relacion señal a ruido (EbN0db).

4. La gráfica final de las curvas de probabilidad de error de bit versus relación señal a ruido para los diferentes valores de relación señal a ruido.

```
% Este script simula el sistema de turbo codigos clasico
% Simula codigos convolucionales concatenados paralelos.
% Se asumen dos codificadores componentes CSR (Convolucionales Sistematicos
% Recursivos) de tasa 1/2.
% El primer codificador es terminado con bits de cola. Los Bits (Info + cola)
$ son 'mezclados' y pasados al segundo codificador, mientras el
% segundo codificador es dejado abierto sin bits de cola.
% Bits aleatorios de info. se modulan en +1/-1, y se transmiten en canal AWGN.
% Los entrelazadores se pueden seleccionar de diversos tipos.
%
% Se emplea el algoritmo Log-MAP sin cuantizacion o aproximacion.
% Usando ln(e^x+e^y) = max(x,y) + ln(1+e^(-abs(x-y))), Se puede simplificar
% el Log-MAP con una tabla de busqueda para la funcion de correccion.
% Si se usa la aproximacion ln(e^x+e^y) = max(x,y), se tiene MAX-Log-MAP.

clear all

% Escribe los mensajes del Command Window en un archivo de texto
diary turbo_logmap.txt

% Escoger el algoritmo de decodificacion

dec_alg = input(' Ingrese el algoritmo de decodificacion.
(0:Log-MAP, 1:SOVA)  default 0    '); if isempty(dec_alg)
   dec_alg = 0;
```

```
end

% Longitud de la trama
L_total = input(' Ingrese la longitud de trama (= info + cola,
default: 400)   '); if isempty(L_total)
   L_total = 400;    % bits de infomación mas bits de cola
end

% Generador del codigo
g = input(' Ingrese el generador de codigo: ( default: g = [1 1 1; 1
0 1 ] )      '); if isempty(g)
   g = [ 1 1 1;
         1 0 1 ];
end
%g = [1 1 0 1; 1 1 1 1];
%g = [1 1 1 1 1; 1 0 0 0 1];

[n,K] = size(g); m = K - 1; nstates = 2^m;

%puncture = 0, puncturado tasa 1/2;
%puncture = 1, no puncturado tasa 1/3
puncture = input(' Escoja puncturado / no puncturado (0/1): default
0     '); if isempty(puncture)
    puncture = 0;
end

% Tasa del codigo
rate = 1/(2+puncture);

%Amplitude de desvanecimiento Fading; a=1 en canal AWGN
a = 1;

% Escoja el tipo de entrelazador
method = input(' Escoja el entrelazador (0:Golden, 1:G. Dithered,
2:Aleatorio, 3:Bloque, 4:Bloque rev, 5:odd-even, 6:3gpp, 7:SRandom)
default 2    '); if isempty(method)
   method = 2;   % entrelazador aleatorio
end

% Número de iteraciones
niter = input(' Ingrese número de iteraciones por cada trama:
default 5       '); if isempty(niter)
   niter = 5;
end
% Numero de errores de trama a contar como criterio de parada
ferrlim = input(' Ingrese el número de errores de trama para
terminar: default 15        '); if isempty(ferrlim)
   ferrlim = 15;
end

EbN0db = input(' Ingrese Eb/N0 en dB : default [0.5 1 1.5 2.0 2.5
3.0]    '); if isempty(EbN0db)
   EbN0db = [0.5 1 1.5 2.0 2.5 3.0];
end

fprintf('\n\n-------------------------------------------------------\n');
if dec_alg == 0
   fprintf(' === Decodificador Log-MAP === \n');
```

```
else
   fprintf(' === Decodificador SOVA === \n');
end
fprintf(' Longitud de trama = %6d\n',L_total);
fprintf(' Generador de codigo: \n'); for i = 1:n
    for j = 1:K
        fprintf( '%6d', g(i,j));
    end
    fprintf('\n');
end if puncture==0
   fprintf(' Puncturado, tasa del codigo = 1/2 \n');
else
   fprintf(' No puncturado, tasa del codigo = 1/3 \n');
end

if method == 8
    fprintf(' Entrelazador IDS \n');
elseif method == 7
    fprintf(' Entrelazador SRandom \n');
elseif method == 6
    fprintf(' Entrelazador 3gpp \n');
elseif method == 5
    fprintf(' Entrelazador odd-even \n');
elseif method == 4
    fprintf(' Entrelazador de bloque reverso \n');
elseif method == 3
    fprintf(' Entrelazador de bloque \n');
elseif method == 2
    fprintf(' Entrelazador aleatorio \n');
elseif method == 1
    fprintf(' Entrelazador Golden Dithered \n');
else
    fprintf(' Entrelazador Golden \n');
end

fprintf(' Numero de iteraciones =  %6d\n', niter);
fprintf(' Errores de trama para terminar = %6d\n', ferrlim);
fprintf(' Eb / N0 (dB) = '); for i = 1:length(EbN0db)
    fprintf('%10.2f',EbN0db(i));
end
fprintf('\n------------------------------------------------------------\n\n');

fprintf('+ + + + Por favor, espere mientras se da el resultado. + +
+ +\n');

for nEN = 1:length(EbN0db)
   en = 10^(EbN0db(nEN)/10);  % convierte Eb/N0 de unidades db a numeros normales
   L_c = 4*a*en*rate;   % valor de fiabilidad del canal
   sigma = 1/sqrt(2*rate*en);   % desviacion estandar del ruido AWGN

% Aclarar los contadores de error de bit y de error de trama
   errs(nEN,1:niter) = zeros(1,niter);
   nferr(nEN,1:niter) = zeros(1,niter);

   nframe = 0;    % resetear el contador de tramas transmitidas
   while nferr(nEN, niter)<ferrlim
      nframe = nframe + 1;
      x = round(rand(1, L_total-m));    % info. bits
```

```
    if method == 8
    alpha = load('IDS256_1317');  % distribucion de bits en el entrelazador IDS
elseif method == 7
    alpha = load('imapSrandom256'); % dist. de bits en el entrelazador Srandom
elseif method == 6
    alpha = entrelazador_3gpp(L_total) +1; % dist. de bits en entrelazador 3gpp
elseif method == 5
    alpha = odd_even_interleaver(L_total-m,1,2,m); % bits en entrelazador odd_even
elseif method == 4
    e = sort(randperm(L_total));
    alpha = back_block(e, 50, 8); % bits en entrelazador de bloque reverso
elseif method == 3
    e = sort(randperm(L_total));
    alpha = interleave(e, 50, 8); % bits en entrelazador de bloque
elseif method == 2
    [temp, alpha] = sort(rand(1, L_total));   % bits en el entrelazador aleatorio
elseif method == 1
    alpha = goldenint(L_total,1); % bits en el entrelazador Golden Dithered
else
    alpha = goldenint(L_total,0);  % bits en el entrelazador Golden
end

     en_output = encoderm( x, g, alpha, puncture ) ; % salida del codificador (+1/-1)

      r = en_output+sigma*randn(1,L_total*(2+puncture)); % bits recibidos
% demultiplexacion para obtener entrada para decodificadores 1 y 2
      yk = demultiplex(r,alpha,puncture);

% Poner escala a los bits recibidos
      rec_s = 0.5*L_c*yk;

% Inicializar informacion extrinsica
      L_e(1:L_total) = zeros(1,L_total);

      for iter = 1:niter
% Decodificador Uno
         L_a(alpha) = L_e;  % info a priori.
         if dec_alg == 0
            L_all = logmapo(rec_s(1,:), g, L_a, 1);  % info completa.
         else
            L_all = sova0(2*rec_s(1,:), g, L_a, 1);  % info completa.
         end
         L_e = L_all - 2*rec_s(1,1:2:2*L_total) - L_a;  % info extrinseca.

% Decodificador Dos
         L_a = L_e(alpha);  % info a priori.
         if dec_alg == 0
            L_all = logmapo(rec_s(2,:), g, L_a, 2);  % info completa.
         else
            L_all = sova0(rec_s(2,:), g, L_a, 2);  % info completa.
         end
         L_e = L_all - 2*rec_s(2,1:2:2*L_total) - L_a;  % info extrinseca.

% Estima los bits de informacion.
         xhat(alpha) = (sign(L_all)+1)/2;

% Numero de errores de bit en la iteracion presente
```

```
         err(iter) = length(find(xhat(1:L_total-m)~=x));
% Cuenta errores de trama para la iteracion presente
         if err(iter)>0
            nferr(nEN,iter) = nferr(nEN,iter)+1;
         end
      end   %iter

% Grafica el numero de bits errados por cada trama transmitida de la
% iteracion final para cada uno de los valores de la relacion señal
% a ruido (EbN0db)

    stem(nframe, err(niter))
    xlabel('Trama transmitida');
    ylabel('Numero de bits errados');
    hold on;

% Numero total de errores de bits para todas las iteraciones
      errs(nEN,1:niter) = errs(nEN,1:niter) + err(1:niter);

      if rem(nframe,3)==0 | nferr(nEN, niter)==ferrlim
% Tasa de error de bit (BER)
         ber(nEN,1:niter) = errs(nEN,1:niter)/nframe/(L_total-m);
% Tasa de error de trama (FER)
         fer(nEN,1:niter) = nferr(nEN,1:niter)/nframe;

% Muestra resultados intermedios en proceso
fprintf('************** Eb/N0 = %5.2f db ***************\n', EbN0db(nEN));
fprintf('Longitud de Trama = %d, tasa 1/%d. \n', L_total, 2+puncture);
fprintf('%d tramas transmitidas, %d tramas erradas.\n', nframe, nferr(nEN, niter));
fprintf('Tasa de Error de Bit (de la iteracion 1 a la iteracion %d):\n', niter);
for i=1:niter
   fprintf('%8.4e    ', ber(nEN,i));
end
fprintf('\n');
fprintf('Tasa de error de trama (de la iteracion 1 a la iteracion %d):\n', niter);
for i=1:niter
   fprintf('%8.4e    ', fer(nEN,i));
end
fprintf('\n');
fprintf('************************************************\n\n');

 % Grabar resultados intermedios
         save turbo_sys_demo EbN0db ber fer
      end

   end      %while
   figure
end         %nEN

% Elaboracion de los graficos. Esta parte es variada de acuerdo a lo que se
% necesite
hold on; BER = ber(1:length(EbN0db), niter);
semilogy(EbN0db,BER,'rs-')
grid xlabel('Relacion señal a
ruido(dB)');
ylabel('Probabilidad de error de bit');

diary off
```

trellis.m

```
function [next_out, next_state, last_out, last_state] = trellis(g)
% Establece el trellis dado el generador de codigo g
% g dado en la forma de matriz binaria. Por ejemplo g = [ 1 1 1; 1 0 1 ];

% next_out(i,1:2): next_out del trellis cuando input = 0, state = i; next_out(i,i) = 0,1
% next_out(i,3:4): next_out del trellis cuando input = 1, state = i;
% next_state(i,1): proximo estado cuando input = 0, state = i; next_state(i,i) = 1,...2^m
% next_state(i,2): proximo estado cuando input = 1, state = i;
% last_state(i,1): estado previo que llega al estado i cuando info. bit = 0;
% last_state(i,2): estado previo que llega al estado i cuando info. bit = 1;

[n,K] = size(g); m = K - 1; max_state = 2^m;

% Establece las matrices next_out y next_state para el codigo sistematico
for state=1:max_state
   state_vector = bin_state( state-1, m );

   % Cuando se recibe un 0
   d_k = 0;
   a_k = rem( g(1,:)*[0 state_vector]', 2 );
   [out_0, state_0] = encode_bit(g, a_k, state_vector);
   out_0(1) = 0;

   % Cuando se recibe un 1
   d_k = 1;
   a_k = rem( g(1,:)*[1 state_vector]', 2 );
   [out_1, state_1] = encode_bit(g, a_k, state_vector);
   out_1(1) = 1;
   next_out(state,:) = 2*[out_0 out_1]-1;
   next_state(state,:) = [(int_state(state_0)+1) (int_state(state_1)+1)];
end

% Encuentra cual de los dos estados previos puede llegar al estado presente
last_state = zeros(max_state,2); for bit=0:1
   for state=1:max_state
      last_state(next_state(state,bit+1), bit+1)=state;
      last_out(next_state(state, bit+1), bit*2+1:bit*2+2) ...
         = next_out(state, bit*2+1:bit*2+2);
   end
end
```

logmapo.m

```
function L_all = logmapo(rec_s,g,L_a,ind_dec)
% Algoritmo Log_MAP empleando metodo straightforward para calcular las
% metricas de las ramas. No se emplea aproximacion.
% Se puede simplificar a Max-Log-MAP usando la aproximacion ln(e^x+e^y) = max(x,y).
% Ingresa: rec_s: Bits recibidos a escala.
%            rec_s = 0.5 * L_c * yk = ( 2 * a * rate * Eb/N0 ) * yk
% g: generador de codigo para el codigo constituyente RSC, en forma de matriz binaria.
% L_a: info. a priori para el decodificador actual,
%       version permutada de Inftyo. extrinseca del decodificador previo.
% ind_dec: indice del decodificador. Puede ser 1 o 2.
%  Se asume que el Codificador 1 es terminado,
%  mientras que codificador 2 esta abierto.
```

```
%
% salida: L_all: tasa log-likelihood de los simbolos. Informacion completa.

% Numero total de bits: Inftyo. + tail
L_total = length(rec_s)/2; [n,K] = size(g); m = K - 1;
nstates = 2^m;          % numero de estados en el trellis

% Set up the trellis
[next_out, next_state, last_out, last_state] = trellis(g);

Infty = 1e10;

% Inicializacion de Alpha
Alpha(1,1) = 0; Alpha(1,2:nstates) = -Infty*ones(1,nstates-1);

% Inicializacion de Beta
if ind_dec==1
   Beta(L_total,1) = 0;
   Beta(L_total,2:nstates) = -Infty*ones(1,nstates-1);
elseif ind_dec==2
   Beta(L_total,1:nstates) = zeros(1,nstates);
else
   fprintf('ind_dec se limita a 1 y 2!\n');
end

% calculo de Alpha
for k = 2:L_total+1
 for state2 = 1:nstates
   gamma = -Infty*ones(1,nstates);
   gamma(last_state(state2,1)) = (-rec_s(2*k-3)+rec_s(2*k-2)*last_out(state2,2))....
        -log(1+exp(L_a(k-1)));
   gamma(last_state(state2,2)) = (rec_s(2*k-3)+rec_s(2*k-2)*last_out(state2,4))....
        +L_a(k-1)-log(1+exp(L_a(k-1)));

     if(sum(exp(gamma+Alpha(k-1,:)))<1e-300)
        Alpha(k,state2)=-Infty;
     else
        Alpha(k,state2) = log( sum( exp( gamma+Alpha(k-1,:) ) ) );
     end
  end
  tempmax(k) = max(Alpha(k,:));
  Alpha(k,:) = Alpha(k,:) - tempmax(k);
end

% Calculo de Beta
for k = L_total-1:-1:1
 for state1 = 1:nstates
  gamma = -Infty*ones(1,nstates);
  gamma(next_state(state1,1)) = (-rec_s(2*k+1)+rec_s(2*k+2)*next_out(state1,2))....
        -log(1+exp(L_a(k+1)));
  gamma(next_state(state1,2)) = (rec_s(2*k+1)+rec_s(2*k+2)*next_out(state1,4))....
        +L_a(k+1)-log(1+exp(L_a(k+1)));
  if(sum(exp(gamma+Beta(k+1,:)))<1e-300)
     Beta(k,state1)=-Infty;
  else
     Beta(k,state1) = log(sum(exp(gamma+Beta(k+1,:))));
  end
 end
```

```
 Beta(k,:) = Beta(k,:) - tempmax(k+1);
end

% Calcular la salida suave, tasa log-likelihood de los simbolos en la trama
for k = 1:L_total
 for state2 = 1:nstates
  gamma0 = (-rec_s(2*k-1)+rec_s(2*k)*last_out(state2,2))....
        -log(1+exp(L_a(k)));
  gamma1 = (rec_s(2*k-1)+rec_s(2*k)*last_out(state2,4))...
        +L_a(k)-log(1+exp(L_a(k)));
  temp0(state2) = exp(gamma0 + Alpha(k,last_state(state2,1)) + Beta(k,state2));
  temp1(state2) = exp(gamma1 + Alpha(k,last_state(state2,2)) + Beta(k,state2));
 end
 L_all(k) = log(sum(temp1)) - log(sum(temp0));
end
```

sova0.m

```
function L_all = sova(rec_s, g, L_a, ind_dec)
% Esta funcion implementa el Soft Output Viterbi Algorithm (SOVA) en el modo trace back
% Entradas:
% rec_s: bits recibidos a escala. rec_s(k) = 0.5 * L_c(k) * y(k)
%      L_c = 4 * a * Es/No, valor de confiabilidad del canal
%      y: bits recibidos
% g:  matriz generadora del codificador en forma binaria,
% g(1,:) feedback, g(2,:) feedforward
% L_a: informacion a priori de los bits info. Info Extrinseca del
% decodificador constituyente previo
% ind_dec: indice del decodificador componente.
%   =1: decodificador constituyente 1; El trellis es terminado al estado todos ceros
%   =2: decodificador constituyente 2; El trellis no esta perfectamente terminado.
% Salida:
%   L_all: log ( P(x=1|y) ) / ( P(x=-1|y) )

% Tamano de la trama, bits info. + cola
L_total = length(L_a); [n,K] = size(g); m = K - 1; nstates = 2^m;
Infty = 1e10;

% Tamaño de la ventana SOVA. Toma la decision luego de un retardo 'delta'.
% Decide bit k cuando se procesan los bits recibidos (k+delta).
% Rastro hacia atras de (k+delta) a k.
delta = 30;

% Elaborar el trellis definido por g.
[next_out, next_state, last_out, last_state] = trellis(g);

% Initializa metricas de los caminos a -Infty
for t=1:L_total+1
   for state=1:nstates
      path_metric(state,t) = -Infty;
   end
end

% Rastro hacia adelante para calcular todas las metricas de los caminos
path_metric(1,1) = 0; for t=1:L_total
   y = rec_s(2*t-1:2*t);
   for state=1:nstates
```

```
      sym0 = last_out(state,1:2);
      sym1 = last_out(state,3:4);
      state0 = last_state(state,1);
      state1 = last_state(state,2);
      Mk0 = y*sym0' - L_a(t)/2 + path_metric(state0,t);
      Mk1 = y*sym1' + L_a(t)/2 + path_metric(state1,t);

      if Mk0>Mk1
         path_metric(state,t+1)=Mk0;
         Mdiff(state,t+1) = Mk0 - Mk1;
         prev_bit(state, t+1) = 0;
      else
         path_metric(state,t+1)=Mk1;
         Mdiff(state,t+1) = Mk1 - Mk0;
         prev_bit(state,t+1) = 1;
      end

   end
end

% Para decodificador 1, rastro hacia atras de todos los estados cero,
% Para decodificador 2, rastro hacia atras del estado mas aparente
if ind_dec == 1
   mlstate(L_total+1) = 1;
else
   mlstate(L_total+1) = find( path_metric(:,L_total+1)==max(path_metric(:,L_total+1)) );
end

% Rastro hacia atras para obtener los bits estimados, y el camino mas aparente
for t=L_total:-1:1
   est(t) = prev_bit(mlstate(t+1),t+1);
   mlstate(t) = last_state(mlstate(t+1), est(t)+1);
end

% Encontrar el delta minimo que corresponde a un camino competidor con
% diferente estimacion de bits de informacion. Dar la salida suave
for t=1:L_total
   llr = Infty;
   for i=0:delta
      if t+i<L_total+1
         bit = 1-est(t+i);
         temp_state = last_state(mlstate(t+i+1), bit+1);
         for j=i-1:-1:0
            bit = prev_bit(temp_state,t+j+1);
            temp_state = last_state(temp_state, bit+1);
         end
         if bit~=est(t)
            llr = min( llr,Mdiff(mlstate(t+i+1), t+i+1) );
         end
      end
   end
   L_all(t) = (2*est(t) - 1) * llr;
end
```

rsc_encode.m

```
function y = rsc_encode(g, x, ed)
```

```
% Codifica un bloque de data x (0/1) con un codigo recursivo sistematico
% convolucional con vectores generadoresa en g, y
% retorna la salida en y (0/1).
% Si ed>0, el trellis es terminado perfectamente
% Si ed<0, se deja no terminado;

% Determina la longitud de contencion (K), memoria (m), y tasa (1/n)
% y numero de bits de informacion.
[n,K] = size(g); m = K - 1; if ed>0
  L_info = length(x);
  L_total = L_info + m;
else
  L_total = length(x);
  L_info = L_total - m;
end

% inicializa los vectores de estado
state = zeros(1,m);

% Genera la palabra de codigo
for i = 1:L_total
   if ed<0 | (ed>0 & i<=L_info)
      d_k = x(1,i);
   elseif ed>0 & i>L_info
      % termina el trellis
      d_k = rem( g(1,2:K)*state', 2 );
   end

   a_k = rem( g(1,:)*[d_k state]', 2 );
   [output_bits, state] = encode_bit(g, a_k, state);
   % Como es sistematico, la primera salida es bit de entrada
   output_bits(1,1) = d_k;
   y(n*(i-1)+1:n*i) = output_bits;
end
```

int_state.m

```
function int_state = int_state( state )
% convierte un vector fila de m bits en un entero (base 10)

[dummy, m] = size( state );

for i = 1:m
   vect(i) = 2^(m-i);
end

int_state = state*vect';
```

encoderm.m

```
function en_output = encoderm( x, g, alpha, puncture )
% emplea el mapa del entrelazador 'alpha'
% Si puncture = 1, no puncturado, produce una salida de tasa 1/3 de longitud fija
% Si puncture = 0, puncturado, produce una salida de tasa 1/2
% multiplexor escoge bits de verificacion impares del RSC1 y bits de verificacion
```

```
% pares del RSC2

% Determinar la longitud de contencion (K), memoria (m)
% y numero de bits de informacion + bits de cola.

[n,K] = size(g); m = K - 1; L_info = length(x); L_total = L_info +
m;

% generar la palabrs de codigo correspondiente al primer codificador RSC
% ed = 1, perfectamente terminado;
input = x; output1 = rsc_encode(g,input,1);

% Crear una matriz con la primera fila correspondiendo a la secuencia de informacion
% segunda fila correspondiendo a los bits de verificacion del RSC #1.
% tercera fila correspondiendo a los bits de verificacion del RSC #2.

y(1,:) = output1(1:2:2*L_total); y(2,:) = output1(2:2:2*L_total);

% Entrada entrelazada al segundo codificador
for i = 1:L_total
   input1(1,i) = y(1,alpha(i));
end output2 = rsc_encode(g, input1(1,1:L_total), -1 ); y(3,:) =
output2(2:2:2*L_total);

% multiplexacion paralelo a serie para obtener el vector de salida
% puncture = 0: la tasa aumenta de 1/3 a 1/2;
% puncture = 1; no puncturado, tasa = 1/3;

if puncture > 0     % no puncturado
   for i = 1:L_total
       for j = 1:3
           en_output(1,3*(i-1)+j) = y(j,i);
       end
   end
else            % puncturado tasa 1/2
   for i=1:L_total
       en_output(1,n*(i-1)+1) = y(1,i);
       if rem(i,2)
      % bits de verificacion impares del RSC1
           en_output(1,n*i) = y(2,i);
       else
      % bits de verificacion pares del RSC2
           en_output(1,n*i) = y(3,i);
       end
    end
end

% modulacion antipodal: +1/-1
en_output = 2 * en_output - ones(size(en_output));
```

encode_bit.m

```
function [output, state] = encode_bit(g, input, state)
% Esta funcion tiene como entrada un solo bit a ser codificado,
% asi como los coeficientes del polinomio generador y
% el actual vector de estado.
% Retorna como salida n bits de data codificados, donde 1/n es la tasa del codigo.
```

```
% La tasa es 1/n
% k es la longitud de contencion
% m es la cantidad de memoria
[n,k] = size(g); m = k-1;

% determinar el proximo bit de salida
for i=1:n
   output(i) = g(i,1)*input;
   for j = 2:k
      output(i) = xor(output(i),g(i,j)*state(j-1));
   end;
end

state = [input, state(1:m-1)];
```

demultiplex.m

```
function subr = demultiplex(r, alpha, puncture);
% Demultiplexion serie a paralelo para obtener la palabra de codigo de
% cada codificador en el terminal del receptor
% alpha: mapeo del entrelazador
% puncture = 0: usa puncturado para incrementar la tasa a 1/2;
% puncture = 1; no puncturado,tasa 1/3;

% Longitud de la trama, que incluye bits de informacion y bits de cola
L_total = length(r)/(2+puncture);

% Extraer los bits de paridad para ambos decod
if puncture == 1        % no puncturado
  for i = 1:L_total
      x_sys(i) = r(3*(i-1)+1);
      for j = 1:2
          subr(j,2*i) = r(3*(i-1)+1+j);
      end
   end
else            % puncturado
   for i = 1:L_total
       x_sys(i) = r(2*(i-1)+1);
       for j = 1:2
          subr(j,2*i) = 0;
       end
       if rem(i,2)>0
          subr(1,2*i) = r(2*i);
       else
          subr(2,2*i) = r(2*i);
       end
   end
end

% Extraer los bits sistematicos de ambos decodificadores
for j = 1:L_total
% Para el decodificador uno
   subr(1,2*(j-1)+1) = x_sys(j);
% Para el decodificador dos: entrelazar lo bits sistematicos
   subr(2,2*(j-1)+1) = x_sys(alpha(j));
end
```

bin_state.m

```
function bin_state = bin_state( int_state, m)

% Convierte un vector de enteros en una matriz; la i-esima fila es la forma binaria
% de m bits para el i-esimo entero

for j = 1:length( int_state )
   for i = m:-1:1
       state(j,m-i+1) = fix( int_state(j)/ (2^(i-1)) );
       int_state(j) = int_state(j) - state(j,m-i+1)*2^(i-1);
   end
end

bin_state = state;
```

Programas adicionales correspondientes a los entrelazadores

Programas añadidos por el autor.

odd_even_interleaver.m

```
function alpha_rsc =
odd_even_interleaver(DLength,TerminateF,NEncoders,m);

% Esta funcion implementa un entrelazador odd_even(impar_par)
% alpha_rsc = odd_even_interleaver(DLength, TerminateF, NEncoders, m)

if TerminateF
  index_odd = [1:2:DLength+m];
  index_even = [2:2:DLength+m];
else
  index_odd = [1:2:DLength];
  index_even = [2:2:DLength];
end % if TerminateF
Lodd = length(index_odd); Leven = length(index_even); for i1 =
1:NEncoders-1
  [dummy, alpha_tmp1(1,:)] = sort(rand(1,Lodd));
  [dummy, alpha_tmp2(1,:)] = sort(rand(1,Leven));
  if (Lodd>Leven)
    alpha_rsc(i1,:) = ...
    reshape([index_odd(alpha_tmp1);index_even(alpha_tmp2), NaN], ...
        1,2*Lodd);
  else
    alpha_rsc(i1,:) = ...
    reshape([index_odd(alpha_tmp1);index_even(alpha_tmp2)],1,2*Lodd);
  end % if (Lodd>Leven)
end % for i1 = 1:NEncoders-1

if (Lodd>Leven)
  alpha_rsc = alpha_rsc(:,1:end-1);
end % if (Lodd>Leven)
```

goldenint.m

```
% goldenint.m
% Esta funcion genera los indices para un entrelazador Golden.
% i = goldenint(N, method)
% N > longitud de entrada del indice generado
% Method > Selecciona entre usar el entrelazador Golden o el dithered golden
% s = punto de inicio para el generador de indices
% m = cualquier entero mayor que 0 (1 o 2 normalmente)
% r = El espaciamiento de indice entre elementos cercanos para lograr la maxiima dispersion
% D = parametro dithered para turbo codigo tipicamente 0.01
% En elementos adyacentes tipicos a maxima dispersion j = 0 and r = 1
% Mayores valores de j y r pueden ser usados para obtener
% el mejor espaciamiento para elementos espaciados r lugares
% Para turbo Codigos con 16 estados j = 9 y r = 15 con repeticion rsc
% periodo de r = 15

function i = goldenint(N, method) m = 2; j = 0; r = 1; if method ==
0
    D = 0;
else
    D = 0.01;
end s = 1;

% Usado para entrelazador dithered golden
ND = N*D; d = rand(1,N)*ND;
% Calculo de la seccion Golden
g = (sqrt(5)-1)/2;
% Calculor el incremento real
C = N*(g.^m+j)/r;
% Calculo del Vector real v
for n = 0:N-1,
    v(n+1) = mod(s + n*C+d(n+1), N);
end
% Encontrar el vector ordenado
[a,b] = sort(v); n = 1:length(b);
% indices del vector
i(b) = n;
```

entrelazador_3gpp.m

```
% Entrelazador interno para el turbo codificador 3gpp

%clear all;
function Table = InternalInterleaver(K)
%K=100;
if (K < 40) | (K > 5114)
    fprintf(1,'Cuidado: La longitud de trama debe estar entre 40 y 5114!\n');
    return;
end

% Determinar el numero de filas de la matriz rectangular
if (40 <= K) & (K <= 159)
    R = 5;
else if ((160 <= K) & (K <= 200)) | ((481 <= K) & (K <= 530))
        R = 10;
    else
        R = 20;
    end
```

```
end

% El numero primo p y su raiz primitiva asociada v
p_tmp = primes (257); p_table = p_tmp(4:length(p_tmp)); v_table =
[3, 2, 2, 3, 2, 5,  2, 3, 2, 6, 3, ...
          5, 2, 2, 2, 2, 7,  5, 3, 2, 3, 5, ...
          2, 5, 2, 6, 3, 3,  2, 3, 2, 2, 6, ...
          5, 2, 5, 2, 2, 2, 19, 5, 2, 3, 2, ...
          3, 2, 6, 3, 7, 7,  6, 3];

% Determine el numero primo a ser usado en la permutacion interna, p, y
% el numero de columnas de la matri rectangular C
if (481 <= K) & (K <= 530)
    p = 53;
    v = 2;
    C = p;
else
    for i = 1 : length(p_table)
        if (R * (p_table(i) + 1)) >= K
            p = p_table(i);
            v = v_table(i);
            break;
        end
    end
    % Determinar C
    if K <= (R*(p-1))
        C = p - 1;
    else if ((R*(p-1)) < K ) & (K <= R*p)
            C = p;
        else if (R*p) < K
                C = p + 1;
            end
        end
    end
end

y = [1:1:K, zeros(1,R*C - K)]; y = reshape(y,C,R)';

s(1) = 1; for j = 1: (p-2)
    s(j+1) = mod(v*s(j), p);
end

q(1) = 1; for i = 2: R
    for j = 1: length(p_table)
        if (gcd(p_table(j), p-1) ==1) & (p_table(j) > 6) & (p_table(j) > q(i-1))
            q(i) = p_table(j);
            break;
        end
    end
end

% Setup T
if (40 <= K) & (K <= 159)
    T = 4:-1:0;
else if ((160 <= K) & (K <= 200)) | ((481 <= K) & (K <= 530))
        T = 9:-1:0;
    else if ((2281 <= K) & (K <= 2480)) | ((3161 <= K) & (K <= 3210))
            T = [19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 16, 13, 17, 15, 3, 1, 6, 11, 8, 10];
```

```
        else
            T = [19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10, 8, 13, 17, 3, 1, 16, 6, 15, 11];
        end
    end
end

r(T + 1) = q;

% Efectuar la i-esima permutacion entre filas interna
U = zeros(1,C); for i = 0: (R-1)
    if C == p
        for j = 0 : (p-2)
            U(j + 1) = s ( mod(j * r(i+1), p-1) + 1);
        end
        U(p) = 0;
    end
    if C == p + 1
        for j = 0 : (p-2)
            U(j + 1) = s ( mod(j * r(i+1), p-1) + 1);
        end
        U(p) = 0;
        U(p + 1) = p;
        if (K == (R * C)) & (i == (R-1))
            tmp = U(1);
            U(1) = U(p+1);
            U(p+1) = tmp;
        end
    end
    if C == p - 1
        for j = 0:(p-2)
            U(j + 1) = s ( mod(j * r(i+1), p-1) + 1) - 1;
        end
    end
    for j = 1 : C
        y_permu1(i + 1, j) = y(i + 1, U(j)+1);
    end
end

% Efectuar la permutacion entre filas externa
for i = 0 : (R-1)
    y_permu2(i + 1,:) = y_permu1 (T(i + 1) + 1, :);
end

x = reshape (y_permu2, 1, R*C); Table = x(x~=0)-1;
```

back_block.m

```
function y = back_block(x, n, m)

% Esta funcion implementa un entrelazador de bloque reverso nxm.
% La data de entrada se escribe en una matriz nxm de fila en fila, y se lee
% de columna en columna de la ultima a la primera columna

% Escribe en la matriz fila por fila
counter = 1; for i=1:n
   for j=1:m
      M(i,j) = x(counter);
```

```
      counter = counter + 1;
   end
end

% Lee de la matriz columna por columna desde la ultima a la primera
counter = 1; for j=m:-1:1
   for i=1:n
      y(counter) = M(i,j);
      counter = counter + 1;
   end
end
```

interleave.m

```
function y = rot_block(x, n, m)

% Esta funcion implementa un entrelazador de bloque nxm.
% La data de entrada se escribe en una matriz de fila en fila,
% y se lee de columna en columna.

% deentrelazado : x = interleave (y, m, n)

% Escribir en la matriz
counter = 1; for i=1:n
   for j=1:m
      M(i,j) = x(counter);
      counter = counter + 1;
   end
end

% Leer de la matriz columna por columna
counter = 1; for j=1:m
   for i=1:n
      y(counter) = M(i,j);
      counter = counter + 1;
   end
end
```

Bibliografía

[1] Andersen, J. D. "Turbo Coding for Deep Space Applications," *Proceedings of the 1995 IEEE International Symposium on Information Theory*, 36 (Setiembre 1995).

[2] Benedeto, S. and G. Montorsi. "Performance Evaluation of Turbo Codes," *Electronics Letters*, *31*(3):163–165 (Febrero 1995).

[3] Benedeto, S. and G. Montorsi. "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, *42*(2):409–428 (Marzo 1996).

[4] Bose, R. C. and D. K. Ray-Chaudhuri. "Further results on error correcting binary group codes," *Information and control*, *3*:279–290 (Setiembre 1960).

[5] Bose, R. C. and D. K. Ray-Chaudhuri. "On a class of error correcting binary group codes," *Information and control*, *3*:68–79 (Marzo 1960).

[6] C. Berrou, A. Glavieux and P. Thitimajshima. "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proceeding of IEEE ICC 93*, 1064–1070 (1993).

[7] Collins, O. M. "The subleties and intracies of building a constraint length 15 convolutional decoder," *IEEE Transactions on Communications*, *40*:1810–1819 (Diciembre 1992).

[8] Divsalar, D. and F. Pollara. "Multiple turbo codes," *Proc., IEEE MILCOM*, 279–285 (Noviembre 1995).

[9] Divsalar, D. and F. Pollara. "On the Design of Turbo Codes," *The Telecommunications and Data Adquisition Progress Report*, *42-123* (Julio-setiembre 1995).

[10] Divsalar, D. and F. Pollara. "Turbo codes for deep-space communications," *The Telecommunications and Data Adquisition Progress Report*, *42-120*:29–39 (febrero 1995).

[11] Elias, P. "Coding for noisy channels," *IRE Conv. Record*, *4*:37–47 (1955).

[12] Forney, G. D. "The Viterbi algorithm," *Proc., IEEE*, *61*:268–278 (Marzo 1973).

[13] Golay, M. J. E. "Notes on digital coding," *Proceedings of the Institute of Electrical and Electronic Engineers*, *37*:657 (1949).

[14] Hamming, R. W. "Error Detecting and Correcting Codes," *Bell Sys. Tech. J.*, *29*:147–160 (1950).

[15] Herzberg, H. "Multilevel Turbo Coding with Short Interleavers," *IEEE Journal on Selected Areas in Communications*, *16*(2):303–309 (Febrero 1998).

[16] Hocquenghem, A. "Codes Correcteurs dÉrreurs," *Chiffres*, *2*:147–156 (1959).

[17] Hokfelt, Johan. *On the Design of Turbo Codes*. PhD Dissertation, Lund University, Lund, Suecia, 2000.

[18] H.R. Sadjadpour, N.J.A. Sloane, M. Salehi and G. Nebe. "Interleaver Design for Turbo Codes," *IEEE Journal on Selected Areas in Communications*, *19*(5):831–837 (Mayo 2001).

[19] J. Hokfelt, O. Edfors and T. Maseng. "Interleaver Design for Turbo Codes Based on the Performance of Iterative Decoding," *Proceeding of IEEE ICC'99, Vancouver, Canada*, *1*:93–97 (1999).

[20] J. Hokfelt, O. Edfors and T. Maseng. "Turbo codes: Correlated Extrinsic Information and its Impact on Iterative Decoding Performance," *Proceeding of IEEE VTC'99, Houston, Texas*, *3*:1871–1875 (1999).

[21] J. Yuan, B. Vucetic and W. Feng. "Combined Turbo Codes and Interleaver Design," *IEEE Transactions on Communications*, *47*(4):484–487 (Abril 1999).

[22] Jung, P. "Comparison of Turbo-code decoders applied to short frame transmission systems," *IEEE Journal on Selected Areas in Communications* *14*(3) (abril 1996).

[23] L. C. Perez, J. Seghers and D. J. Costello. "A distance spectrum interpretation of turbo codes," *IEEE Transactions on Information Theory*, *12*:1698–1708 (Noviembre 1996).

[24] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv. "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, 284–287 (marzo 1974).

[25] Proakis, John G. *Digital Communications*. McGraw-Hill, 1995.

[26] Rainer Lucas, Martin Bossert and Markus Breitbach. "On iterative Soft-Decision Decoding of Linear Binary Block Codes and Product Codes," *IEEE Journal on Selected Areas in Communications*, *16*(2):276–296 (Febrero 1998).

[27] Reed, I. S. and G. Solomon. "Polimials codes over certain finite fields," *SIAM Journal on applied mathematics*, *8*:300–304 (1960).

[28] S. Dolinar, D. Divsalar and F. Pollara. "Turbo Code performance as a function of code block size," *International Symposium of Information Theory* (Abril 1998). Boston, USA.

[29] Seghers, J. *On the free distance of turbo codes and related product codes*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Suiza, Agosto 1995.

[30] Shannon, C. E. "A mathematical theory of communication," *Bell Sys. Tech. J.*, *27*:379–423 y 623–656 (1948).

[31] Takeshita, O.Y. and D.J. Costello Jr. "New Classes of Algebraic Interleavers for Turbo Codes," *International Symposium on Information Theory*, 419 (Agosto 1998).

[32] Valenti, M. C. and B. D. Woerner. "Variable latency turbo codes for wireless multimedia applications," *Proc. Int. Symp. on turbo codes and related topics*, 216–219 (Setiembre 1997).

[33] Valenti, Matthew C. *Iterative Detection and Decoding for Wireless Communications*. PhD Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1999.

[34] W. J. Blackert, E. K. Hall and S. G. Wilson. "Turbo code termination and interleaver conditions," *Electronics Letters*, *31*:2082–2083 (Noviembre 1995).

[35] Wicker, Stephen B. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, Inc., 1995.

[36] Wu, Yufei, "Turbo Code Demo." http://www.ee.vt.edu/ yufei/turbo.html, Junio 1999.