UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SISTEMAS DE ECUALIZACIÓN TURBO, USANDO LOG-MAP Y LDPC NO BINARIO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

GONZALO ANTONIO YÁÑEZ AZÚA

PROFESOR GUÍA:
ISMAEL SOTO G.

MIEMBROS DE LA COMISIÓN:
HELMUTH THIEMER W.
BENJAMIN JACARD H.

SANTIAGO DE CHILE
MAYO 2008

Gracias a mi familia, mis amigos, mi polola Marilén y su familia, quienes me bridaron su apoyo durante los momentos difíciles y fueron capaces de darme fuerzas para sacar adelante este trabajo.

Gracias a mi profesor Ismael Soto y a Claudio Valencia, quienes me orientaron en esta investigación.

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: GONZALO YÁÑEZ AZÚA
FECHA: 18 DE ABRIL DE 2008
PROF. GUÍA: ISMAEL SOTO GÓMEZ

"Sistemas de ecualización Turbo, usando log-MAP y LDPC no binario"

El creciente avance de las telecomunicaciones durante estos últimos años, ha despertado el interés de muchos investigadores en conseguir más y mejores técnicas que permitan establecer comunicaciones inalámbricas robustas y que a su vez, soporten grandes flujos de información a alta velocidad. No obstante, la utilización de un canal inalámbrico para el envío y recepción de señales, introduce efectos indeseados tales como interferencia y ruido, debido a que entre el emisor y el receptor se interponen diferentes agentes que se comportan de aletoriamente. Es así como se han desarrollado métodos que han despertado gran interés entre los investigadores, debido a la obtención de resultados sorprendentes. Entre estos métodos destacan el algoritmo MAP y los códigos LDPC. El primero puede ser utilizado tanto en el aspecto de la ecualización de señales o en la decodificación de señales, mientras que el segundo se utiliza específicamente para codificar y decodificar señales.

De este modo, el objetivo principal de esta memoria es unir estos dos algoritmos para que trabajen de forma iterativa, utilizando la información proporcionada por el ecualizador para mejorar el proceso de decodificación y a su vez, realizar el proceso de ecualización utilizando la información del decodificador. A este sistema retroalimentado se le llama Esquema Turbo. Además, se plantea el algoritmo para lenguajes no binarios.

Para realizar este trabajo, primeramente se hizo un desglose de un sistema de comunicaciones actual, reconociendo cada uno de los bloques componentes, analizando su funcionamiento teórico, introduciendo modificaciones según corresponda y a partir de este análisis obtiene un diagrama de bloques que resume el funcionamiento general del algoritmo propuesto. Seguidamente se realizan pruebas a pequeña escala, utilizando un modelo de canal TDL Gaussiano de tres derivaciones, capaz de emular el comportamiento de una señal bajo interferencia y ruido, con el objetivo de demostrar el correcto funcionamiento del algoritmo y su convergencia, para posteriormente someter el esquema a transmisiones de grandes bloques de información, midiendo su desempeño bajo distintos escenarios y estudiando la tasa de símbolos errados (SER) variando la razón señal ruido (SNR). Los resultados obtenidos de esta investigación dicen que este algoritmo es capaz de lograr una corrección completa en señales pequeñas, incluso bajo condiciones en que de SNR se reduce hasta 5 dB para 7 iteraciones. Sin embargo, al momento de enviar bloques de información de tamaño mucho mayor (10^5 símbolos), el algoritmo presenta un piso de SNR de 10 dB, ya que al aumentarlo por sobre ese valor no se lograba mejoramiento de decodificación, debido a que se utilizó un esquema de codificación de bloques con matrices pequeñas. Esto se explica justamente porque la elaboración de matrices de codificación para códigos LDPC no binarios, para bloques grandes de información, es un tema de investigación que no se encuentra resuelto hasta la fecha.

Contenido

Índice de figuras

Índice de tablas

Índice de acrónimos

ASCII	American Standard Code for Information Interchange
AWGN	Additive white Gaussian noise
BER	Bit Error Rate
CISI	Controlled ISI
DFE	Decision-Feedback Equalizer
EXIT Charts	Extrinsic information transfer chart
FEC	Forward Error Correction
FFT	Fast Fourier Transform
GF(q)	Galois Field (q order)
GMSK	Gaussian minimum shift keying
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HSDPA	High-Speed Downlink Packet Access
IFFT	Inverse Fast Fourier Transform

IMT-2000	International Mobile Telecommunications-2000
ISI	Inter Symbol Interference
LDPC	Low Density Parity Check
log-MAP	Logaritmic MAP
LLR	Log Likelyhood Ratio
MAP	Maximum A Posteriori
MIMO	Multiple-input and multiple-output
ML	Maximum Likelihood
PCS	Personal Communications Service
PDF	Probability density function
PSK	Phase-shift keying
PSTN	Public switched telephone network
QAM	Quadrature amplitude modulation
RSC	Recursive Systematic Convolutional code
SER	Symbol Error Rate
SNR	Signal-to-Noise Ratio
SOVA	Soft-Output Viterbi Algorithm
TDMA	Time division multiple access
UMTS	Universal Mobile Telecommunications System
WCDMA	Wideband Code Division Multiple Access
WHM	Walsh Hadamard Matrix

Capítulo 1

Introducción

1.1 Motivación

En los últimos años, las tecnologías de telecomunicaciones han sufrido una evolución tremenda, debido a que la demanda por este tipo de servicios aumenta de forma sistemática y con ello, cada uno de los competidores de la industria busca satisfacer las necesidades de los clientes de modo de situarse por sobre sus competidores. Es por ello que la telefonía ha tenido saltos como la forma de transmisión, pasando del envío de datos de forma análoga, hasta el envío de ondas de forma digital (modulada en señales análogas), dado por los beneficios que se obtienen con este cambio de formato. Además, es posible diferenciar entre los medios físicos por donde se envían las señales, saltando desde la telefonía por cableado, hasta lo que actualmente se está utilizando de forma masiva; la transmisión inalámbrica, dado que los consumidores de telefonía necesitan estar comunicados todo el tiempo, independientemente del lugar donde se encuentren. Además, existen diversas formas de transmisión de datos por vía inalámbrica, donde destacan la telefonía celular, satelital y microondas, que se diferencian principalmente porque se ubican en distintos lugares del espectro radioeléctrico, es decir, son ondas con distintas frecuencias, y con ello presentan distintos alcances y limitaciones. Cabe señalar el papel importante que han jugado los gobiernos de los distintos países que lideran el desarrollo de tecnologías de telecomunicaciones, ya que éste regula el uso del espectro radioeléctrico para los distintos competidores, y no lo regulan las propias empresas proveedoras o los fabricantes de equipos.

El gran salto de la comunicación inalámbrica fue dado por los Laboratorios Bell en 1960 y 1970 con el desarrollo del concepto de telefonía celular, que principalmente plantea la idea de dividir un territorio en celdas, cada una con una estación base y que entrega una conexión inalámbrica a la PSTN (red telefónica pública switcheada o red telefónica cableada convencional) a algún usuario que se requiera comunicarse con algún teléfono fijo o móvil. Dentro de estas celdas, la estación base es capaz de entregar cierto número de canales de radio a quienes se encuentra dentro de una celda, de modo que estos mismos canales pueden ser reutilizados en otra celda que se ubique a una distancia que no interfiera con aquella que maneja los mismos canales de radio, este concepto llamado reutilización de frecuencias es la base del sistema celular. Además, si un usuario se mueve de una celda a otra, se produce un sofisticado proceso llamado *handoff,* donde se cambia el canal usado por el teléfono móvil y estación, manteniendo la calidad, haciendo de forma casi imperceptible el cambio de canal quienes está realizando el intercambio de datos. Además, los sistemas celulares entregan un servicio llamado *roaming*, el cual permite que un usuario pueda operar en áreas de servicio en las cuales no se encuentra suscrito. Con el pasar de los años, las empresas han creado una serie de estándares que utilizan cierta cantidad de canales en una porción de ancho de banda, además de introducir esquemas de modulación más eficiente para enviar un mayor número de símbolos en una misma onda portadora. Además, a medida que avanzan los estudios en esta área, se incorporan esquemas para transmitir señales de

manera más robusta, que puedan soportar mayor interferencia, mejorando la capacidad de los sistemas celulares. Para señales de voz, se han incorporado además las tecnologías de procesamiento de voz llamadas *vocoder*, que principalmente reducen la cantidad de bits requeridos para envío y recepción de voz.

A partir de 1990 se crearon servicios especializados que permitían hablar por teléfono, envío de mensajería y transmisión de datos, todos estos servicios usando la misma red celular. Entre éstos destacan los Servicios de Comunicación Personal (PCS), cuyo estándar más conocido es el GSM, cuya banda de frecuencias se ubica entre los 1.85 y los 1.99 GHz (específicamente en PCS-1900 se utilizan las bandas de frecuencia dividida entre 1.85–1.91 GHz para enlace de subida y 1.93–1.99 GHz para enlace de descarga), acceso múltiple vía TDMA y modulación GMSK, donde el ancho de banda asignado para cada canal es de 200 kHz.

La tecnología que predomina actualmente en Chile es principalmente GSM (anteriormente se usaban redes TDMA, pero se ha migrado a GSM, basado en tecnología TDMA) que cada día se transforma en el estándar más utilizado mundialmente. Si se quiere adoptar una ubicación temporal de acuerdo a la evolución de esta tecnología, se puede decir que ésta es la segunda generación o 2G, el cual permite velocidades de transmisión de 10 kbps. A partir de este punto, en la actualidad se encuentran desarrollando nuevas tecnologías de comunicaciones evolucionando a 2.5G a través de GPRS (Sistema de radio general de paquetes), el cual coexiste con GSM, que incorpora a este estándar una mayor velocidad de trasmisión, conexión permanente y tarificación por tráfico, servicios WAP y acceso a internet. La velocidad de transmisión de esta tecnología es de entre 64 y 144 kbps La tercera generación o 3G, especificado en el estándar IMT-2000, el cual incorpora mayor calidad, capacidad, movilidad (*roaming* mundial) e integración de servicios, pudiendo alcanzar velocidades de transferencia que varían entre los 384 kbps y los 2 Mbps, utilizando la tecnología espectral UMTS/WCDMA. Actualmente las mayores compañías en Chile de telefonía celular se encuentran realizando una marcha blanca para incorporar sistemas de 3.5G, que utilizan la tecnología HSDPA (*High Speed Downlink Packet Access* o Acceso de baja de paquetes de alta velocidad) la cual mejora aun más la velocidad de transferencia de paquetes de datos llegando hasta los 14 Mbps y se considera el paso previo para las redes de cuarta generación o 4G.

Es fácil ver que la tendencia de las redes celulares se centra en enviar la mayor de cantidad de datos posibles, optimizando la utilización del espectro, de forma confiable y con una cobertura que no deje a ningún cliente incomunicado.

Para agregar confiabilidad al envío de datos se introducen datos adicionales a la señal, a los cuales se les llama “datos redundantes”, además de codificar la señal, de manera de permitir al receptor recuperar una señal que fue enviada por un transmisor, y que ha sufrido los efectos del canal como ruido o interferencia. Es natural pensar entonces que si se quiere aumentar la confiabilidad de una señal, entonces la señal útil que desea enviar debe contener una mayor cantidad de bits de redundancia por lo que el ancho de banda efectivo se reduce, entonces se produce una disputa entre otorgar mayor prioridad a la cantidad de datos que se envían o a la confiabilidad de éstos. El pionero en trabajar en la codificación de canal o también llamada *Forward Error Correction* (FEC) o corrección de errores hacia delante fue Shannon en 1948, prediciendo que es posible enviar señales de forma confiable si se añade información redundante. Además establece un teorema que limita la cantidad máxima de datos digitales que es posible enviar, sobre un canal con ruido, dentro de un ancho de banda específico. Con los esquemas actuales de codificación (convolucional y turbo) se han alcanzado resultados que se acercan cada

vez más a los límites pronosticados por Shannon. Pero, no sólo el efecto del ruido es el que introduce el canal a las señales que viajan por el aire, sino además que sufre los efectos propios de las ondas que interactúan con el medio. Este tipo de interferencia es llamado desvanecimiento *(fading)* el cual debe ser disminuido por medio de ecualizadores y codificadores. En este contexto, la idea de este trabajo de título es incorporar un método para ecualizar señales binarias para disminuir los efectos producidos por la interferencia inter símbolo, trabajando de forma conjunta y concatenada con un par codificador-decodificador LDPC no binarios.

1.2 Alcances

En el contexto de esta memoria se hace un análisis teórico y práctico de los algoritmos que actualmente se ocupan sistemas de ecualización y decodificación, como son los algoritmos MAP y LDPC sobre lenguajes binarios, los cuáles se encuentran actualmente incorporados en sistemas de comunicaciones actuales con la idea principal de unir estos algoritmos de manera que puedan trabajar de forma conjunta para proponer una idea de algoritmo. El código propuesto en esta memoria, se basa en los códigos convolucionales turbo, pero a su vez incorpora otro algoritmo tipo bloques, como es el LDPC y que ha obtenido grandes resultados hasta la fecha. Además, el algoritmo propuesto extiende la idea actual a lenguajes no binarios y procesa la información sin hacer conversión intermedia de símbolos, lo que podría significar un cierto nivel de ganancia computacional versus algoritmos que deben descomponer la señal no binaria en binaria para poder procesarla.

Esta memoria pretende proponer un algoritmo basado en códigos turbo binarios, incorporando un elemento como los códigos LDPC y extendiendo los algoritmos existentes a lenguajes no binarios, realizando prototipos de pequeña escala, que en un futuro se propone extender estos modelos a mayor escala, de modo de lograr una optimización del modelo que pueda significar mejor desempeño para la transmisión de datos por vía inalámbrica.

1.3 Objetivos

1.3.1 Objetivos generales

El objetivo general de este trabajo es desarrollar, programar y simular un algoritmo de corrección de datos, basado en la combinación de codificación LDPC no binaria y ecualización log-MAP en lenguajes no binarios.

1.3.2 Objetivos específicos

El objetivo general de esta memoria de título se puede estructurar a través de los siguientes objetivos específicos:

- Entregar una visión general y actualizada tanto de los sistemas de telecomunicaciones actuales, como de los métodos de codificación-ecualización que se están desarrollando en el mundo, centrados especialmente en los que se usarán en este trabajo.
- Explicar y modelar en forma teórica un sistema codificación no binario, mezclando códigos log-MAP y LDPC dado que han arrojado grandes resultados de forma separada y que en este trabajo de título, se les hará trabajar de forma concatenada y coordinada, usando la idea propuesta para códigos turbo.
- Simular el algoritmo propuesto en esta memoria, en un contexto de pequeña escala, donde se pretende realizar un análisis de factibilidad del algoritmo, estudiar su comportamiento y su convergencia.
- Realizar un análisis en detalle de este algoritmo haciendo simulaciones a mayor escala, modificando parámetros del canal y enviando cantidades de datos superiores para obtener curvas características del código y su desempeño.
- Analizar comparativamente los resultados obtenidos, pudiendo visualizar las ventajas y desventajas, las cuales permitan extrapolar este diseño a futuras modificaciones y proponer posibles mejoras.

1.4 Plan de trabajo

Para poder cumplir de forma ordenada con los objetivos anteriormente planteados, es necesario establecer un plan de trabajo acorde con el tiempo y los alcances que se deben cumplir. Por ello, se estableció el siguiente plan de trabajo: recolectar

- Reunir los antecedentes que permitan desarrollar esta investigación. Para ello se deben estudiar los aspectos históricos, avances hasta la fecha, tendencias y los trabajos propuestos para el futuro.
- Investigar el modelo teórico del que están compuestos los algoritmos involucrados en el desarrollo del tema, con el fin de establecer el marco en que se pueden introducir mejoras.
- Llevar el modelo teórico (matemático) a un modelo compuesto por bloques que permitan abordar el problema de manera modular, dejando la posibilidad de que en trabajos futuros puedan mejorar cada uno de los bloques por separado, dividiendo el trabajo.
- Implementar el modelo a través de programas usando 2 plataformas: JGrasp y NetBeans 5.5.1, ya que esta última entrega la posibilidad de simular las clases java que usan los teléfonos celulares actuales.
- A través de los modelos presentados, realizar una comparación del mismo algoritmo bajo distintos escenarios, de manera de decidir qué ventajas y desventajas entrega cada uno.
- Obtener conclusiones a partir de esos resultados, y proponer desafíos futuros que puedan guiar investigaciones posteriores.

1.5 Estructura de la memoria

Para entender bien este trabajo es necesario esquematizar de buena manera cómo se desarrolla esta memoria, para ello se ha dividido este estudio en 5 capítulos. El primer capítulo es una introducción para internar al lector en el aspecto de las telecomunicaciones, mediante un contexto histórico, junto con delimitar el trabajo especificando los objetivos. Posteriormente, en el capítulo 2 se realizará una revisión bibliográfica, donde se estudiarán conceptos generales de los modelos utilizados en este trabajo, historia del avance de estas tecnologías y estado del arte presente en estos tópicos de investigación. En el capítulo 3 se presentarán los fundamentos teóricos sobre los cuáles se basa la memoria, revisar la base de los algoritmos de estudios, ecuaciones, esquemas y cuáles son las ventajas y desventajas que presentan. En el capítulo 4 se hará una descripción del sistema propuesto, en un principio será una descripción general pasando posteriormente a estudio en mayor detalle, explicando en tratamiento de señales y realizando modificaciones al lo que propone el planteamiento teórico, explicando con ecuaciones. Posteriormente se realizan mediciones y simulaciones del modelo propuesto, obteniendo gráficos de desempeño y pudiendo identificar ventajas y desventajas del modelo. En el capítulo 5, partir de los resultados obtenidos en el capítulo anterior, se obtendrán conclusiones del trabajo, para comentar las mejorías o aportes que se pudieran entregar al campo de las telecomunicaciones. Además se realizarán propuestas para quiénes se motiven en continuar con el tema y se especificarán que puntos mayormente conviene atacar.

Capítulo 2

Antecedentes: Historia de las telecomunicaciones y su evolución.

2.1 Introducción

Este capítulo tiene el objetivo de entregar al lector un contexto del desarrollo de los temas contenidos dentro de este trabajo, comenzando por temas generales como el descubrimiento de los algoritmos, su desarrollo en el tiempo, para finalmente entregar un estudio del estado del arte sobre el tema.

2.2 Generalidades

Se puede decir que el inicio del tema de corrección de errores en señales, se remonta a 1948, cuando Claude Shannon introdujo la teoría de información en [16]. Posteriormente, en 1949 Werren Weaver volvió a publicar este artículo, enfatizando algunos aspectos de esta propuesta. Esta teoría introduce los conceptos de fuente binaria y canal simétrico binario, donde el primero corresponde a un dispositivo que emite símbolos “0” y “1”, llamados bits, mientras que el segundo es el medio por donde se transmiten esos bits. El canal presentado por Shannon no es 100% confiable y existe una probabilidad de que el bit que llegue sea errado. Para ello también Shannon hace un análisis entre la comunicación de un receptor y transmisor, comunicados mediante un canal no confiable. Con ello mide principalmente la cantidad de información que puede enviar el transmisor de forma confiable. Estos fenómenos fabrican una teoría de la información basada en modelos estadísticos, lo que permitió, entre otras cosas, determinar la capacidad de un canal de comunicación en términos de los bits, establecer mejores métodos para separar las señales del ruido y cómo utilizar los mejores métodos para distintos sistemas de comunicación. Esta teoría propone principalmente 3 conceptos:

1. Definir una variable que permita medir la información, dependiente de las características del sistema. Esta variable se llama medida de información.
2. La información está relacionada con las fuentes que la generan. Para tratar esta información para mejor desempeño del esquema se deben introducir técnicas de compresión y encriptamiento.
3. Relación entre canal y la información que transmite. Acá aparece el concepto de ruido. Además aparece el parámetro denominado capacidad de información del canal y asociado con lo anterior, surge el tratamiento de la información usando métodos de codificación para control de errores.

En 1950 Richard Hamming [21] presentó el primer algoritmo de corrección de errores, el llamado código Hamming, el cual se sigue usando actualmente. Éste método permitía corregir sólo un bit errado por cada palabra codificada. Paralelamente Marcel Golay diseñó los códigos Golay [22] que permiten corregir hasta 3 bits errados por cada palabra codificada. En 1955 se introdujeron los códigos convolucionales en el trabajo de P. Elías [23]. Los códigos convolucionales han sido la base del estudio de los llamado códigos turbo, además de otras variaciones de estos códigos, que se han presentado como los más populares hasta esta fecha.

Los códigos BCH fueron introducidos en 1960 por R. C. Bose y D.K. Chaudhuri [24], que era un poderoso código que trabajaba por bloques de datos, capaces de corregir muchos datos a la vez. Posteriormente Berlekamp [25] obtuvo potentes algoritmos para decodificar códigos BCH a inicios de los 60. A su vez, ese mismo año aparecen los códigos Reed-Solomon[26], los que han sido altamente utilizados en la actualidad en comunicaciones satelitales o en el tratamiento de datos corruptos en el software y hardware de computadores. La idea de concatenar códigos la mencionó G.D Forney [27] el año 1966. Al año siguiente aparece uno de los más importantes descubrimientos de la mano de A. J. Viterbi [28], quien diseño el algoritmo que lleva su apellido para decodificar códigos convolucionales, maximizando la probabilidad de corrección.

Estos códigos tenían un alto poder correctivo, pero las limitaciones técnicas que había hasta ese entonces no hacían posible su implementación. En 1974 se presentaron unos códigos concatenados en cascada, que podían corregir errores en cascada y aleatorios, éstos fueron presentados en [29]. Iniciados los 70, un investigador de apellido Goppa [30] introdujo un método totalmente distinto de codificación por medio de curvas algebraicas, que facilitaban de gran manera la codificación y control de códigos. Pero sólo a finales de los 80 se encontraron métodos eficientes para decodificar los códigos de Goppa, ya que hay una profundidad matemática importante en el tratamiento de estos códigos.

En 1993 los investigadores Claude Berrou y Alain Glavieux [1] introdujeron los códigos turbo, que dejaron un gran impacto en el mundo de las telecomunicaciones, debido a los impresionantes resultados obtenidos, aproximándose a las tasa máximas de transmisión posibles que vienen de la teoría de Shannon. Estos códigos se utilizan actualmente en telefonía 3G y transmisión digital.

La “contraparte” de los códigos turbo son los códigos LDPC (Low-Density Parity Check), contraparte por el hecho que se han estudiando de forma paralela y se muestran resultados de uno en función del otro. Hay que mencionar que estos códigos fueron inventados en 1960 por Robert Gallager en [2], pero que en su tiempo no existían herramientas que permitieran su implementación práctica. En la publicación de D. J. C. Mackay et al.[5] del año 1997 se obtienen esos resultados y se descubren los alcances de esta técnica.

El año 1998, los investigadores M. C. Davey y D. J. C. MacKay diseñaron los códigos LDPC en lenguajes no binarios en [31] y posteriormente en 1999 implementan estos algoritmos para la corrección de errores. El mismo 1998 también se propuso el diseño de códigos LDPC irregulares, junto también con proponer la reducción de la complejidad del algoritmo usando transformadas de Fourier.

Hasta el momento se ha investigado mucho sobre el tema de los códigos LDPC. Investigadores como Richardson et al. el año 2001 [32], midieron el funcionamiento de los códigos LDPC sobre canales AWGN, posteriormente en los años 2001 y 2002 se determinaron los límites de tasa de los códigos LDPC, estudios que fueron realizados por Burshtein[33] y

paralelamente Chen y Fossorier [34] se encontraban mejorando estos límites experimentando con bloques de gran tamaño.

Por otra parte, la ecualización turbo fue propuesta por Douillard el año 1995, usando códigos convolucionales y modulación BPSK. En [35] demostró que el ecualizador turbo es capaz de mitigar los efectos de la interferencia inter símbolo (ISI), entregada por la respuesta al impulso del canal. Este resultado se obtuvo uniendo la ecualización de canal con la decodificación de canal, trabajando iterativamente. Gertsman y Lodge [36] demostraron que el principio turbo es capaz de compensar las degradaciones debidas a respuestas al impulso del canal estimadas imperfectamente.

2.3 Estado del arte

Actualmente se han realizador diversos estudios sobre aplicaciones de códigos LDPC no binarios, por ejemplo C. Poulliat et al.[37], el año 2006, proponen una implementación de códigos LDPC no binarios de modo de optimizar el modelo usando sus imágenes binarias. O también está algo mucho más concreto como el paper de Dai Kimura et al [7] implementan el algoritmo para comunicar vehículos en movimiento, consiguiendo resultados más que aceptables.

Las primeras implementaciones de esquemas de ecualización turbo fueron propuestas en 1997, en el trabajo de G. Bauch et al [38], que principalmente propone una implementación del esquema de Douillard para sistemas de telefonía móvil. El año 1998 se traspasa este modelo a algo más concreto como el sistema que actualmente se utiliza actualmente, el llamado GSM y los proponen los investigadores G. Bauch et al [39] combinaron esquemas de ecualización turbo y decodificación turbo. El 2000, los académicos L. Hanzo et al [41] introdujeron modulación adaptativa junto con ecualización turbo, usando estimación iterativa.

Una pequeña aplicación de ecualización turbo se puede encontrar en el paper de T. Okada et al [42] donde se implementa este algoritmo usando modulación GMSK para transmitir señales FM de banda angosta.

La publicación del año 2007 de Naeem Ramzan el at [45] se propone la concatenación de códigos LDPC y códigos turbo enfocado al envío de señales de video escalable, relacionando los esquemas de codificación de canal y de fuente. Este trabajo tiene gran similitud con el presentado en esta memoria, con la diferencia que se proponen estos esquemas bajo sistemas no binarios, pudiendo aprovechar de mejor forma el uso del canal. Anteriormente, en el año 2005 se había propuesto un esquema similar para el tratado de señales de video usando sólo códigos LDPC en la publicación M. Bansal y L. P. Kondi [44].

Existe un nuevo elemento que ha tomado gran interés en las telecomunicaciones actuales, que es la tecnología OFDM. Esta modulación presenta varios beneficios con respecto a la modulación convencional. Con ello, se han desarrollado varios trabajos que se complementan con ecualización turbo y LDPC, por ejemplo en la publicación de L. Hanzo et al [45] se propone un esquema que combinan tecnología OFDM con los códigos que se mencionan, con buenos resultados. En la práctica, usar la tecnología OFDM (multiplexación por división de frecuencias ortogonales) es aumentar la complejidad del modelo transmisor receptor, ya que se deben establecer esquemas de diversidad. Además, en la práctica es difícil generar señales que sean ortogonales, que son los tipos de señales que trabajan con OFDM, es por ello que en la actualidad

se han implementado esquemas de diversidad con pocas antenas. Se espera que en el futuro se puedan combinar técnicas OFDM y ecualización para eliminar las imperfecciones que presentan ambos modelos por separado.

Sobre esquemas de ecualización turbo no binarias sólo se han encontrado planteamientos netamente teóricos, pero que los resultados de sus simulaciones entregan un alentador primer paso para seguir con su investigación. Es por ello que se decidió usar la idea de esquemas turbo y combinar estas dos técnicas de corrección de señales para medir si trabajando juntas pueden entregar un resultado mejor.

Capítulo 3

Fundamentos de la Teoría de Codificación LDPC y Ecualización log-MAP

3.1 Estructura general de sistemas de comunicaciones

El principal objetivo de un sistema de telecomunicaciones es entregar información desde un punto a otro, los cuales se encuentran separados uno del otro. Mirando desde este punto de vista el modelo de un sistema de telecomunicaciones, se podría decir que existen tres principales agentes: el emisor (quién envía la señal), el canal (medio por el cual se transmite la señal) y el receptor (el destinatario al cual debe llegar el mensaje del emisor).

Figura 3.1. Modelo elemental de comunicación

Este simple modelo puede ser aplicado para distintas formas de comunicación a distancia, ya sea, radio, telegrafía, televisión, telefonía, etc. Al momento de querer llevar este simple modelo a un contexto real en telefonía celular, es necesario incorporar otros elementos, ya que ha habido aumento de la demanda por servicios de comunicación inalámbrica (telefonía, datos, etc.), tanto en la cantidad de usuario que desean utilizar este medio para el envío y recepción de sus datos, como en la cantidad de datos que cada uno de los usuarios requiere enviar a recibir. Entonces este modelo debe ahora soportar no sólo uno, sino varios usuarios y además debe soportar el envío y recepción de gran cantidad de datos.

Para lograr un ancho de banda más eficiente, lo que permite soportar envío de mayor cantidad de información y/o soportar a más usuarios, se usan esquemas de modulación, ya que la transmisión de datos en banda base (frecuencia a la cual es enviada la información por el emisor) es limitada en frecuencia, produciendo Interferencia Inter Simbólica o ISI (Criterio de Nyquist) e interferencias. Estos esquemas de modulación permiten enviar información sobre una sola onda portadora, de modo de aprovechar de mejor forma el canal, permitiendo transmitir más información de forma simultánea, además de permitir la protección de los datos enviados sobre posibles interferencias, ruido o desvanecimiento. Existe una gran cantidad de tipos de modulación, tanto analógicos como digitales y cada uno presentando distintas ventajas y desventajas. Para este estudio se utilizará modulación PSK (Phase-Shift keying), la cual es muy utilizada en esquemas turbo, primeramente utilizando modulación BPSK y posteriormente

extendiéndose a lenguajes no binarios MPSK. Entonces, incorporando este elemento, es posible establecer el siguiente modelo:

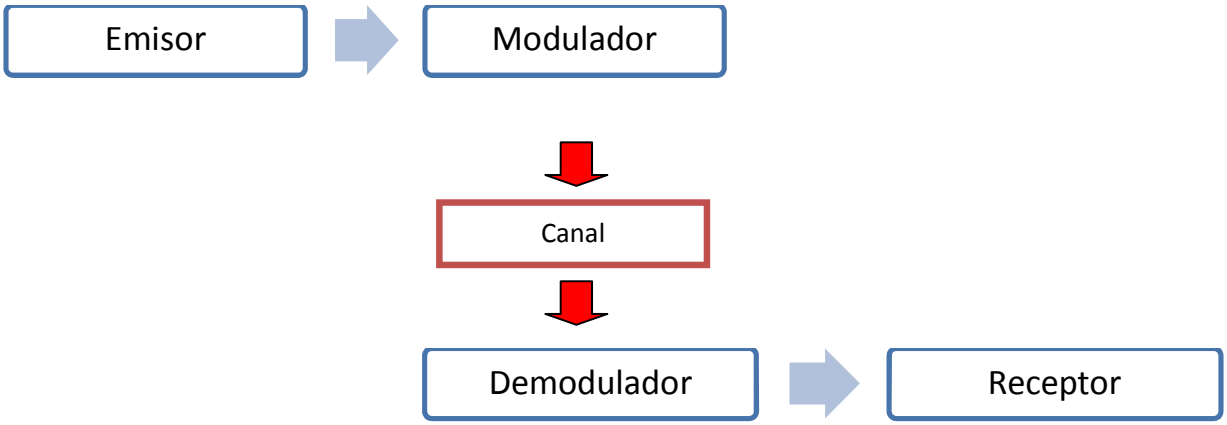


Figura 3.2. Modelo elemental de comunicación, incorporando modulación

La incorporación de un conjunto modulador-demodulador en el esquema de telecomunicaciones permite un mejor aprovechamiento del canal de comunicación lo que posibilita transmitir más información en forma simultánea, protegiéndola de posibles interferencias y ruidos.

Las ondas moduladas por las cuales viaja la información, también están sujetas a sufrir desvanecimientos e interferencias debido que en el canal por el cual viaja es el aire, y éste se encuentra sujeto a diversos fenómenos aleatorios que hacen cambiar los valores de potencia de la onda viajera. Uno de estos fenómenos es el de multitrayecto, el cual surge por el hecho de que la onda enviada llega al receptor desde varias direcciones o caminos. Estos caminos se originan por las múltiples reflexiones, difracciones y dispersiones, las que producen atenuaciones y retardos de forma aleatoria. Esto se puede observar en la Figura 3.3, donde la línea roja representa onda que llega con línea de vista (es decir, directa) y la de color morado representa ondas reflejadas o sin línea de vista:

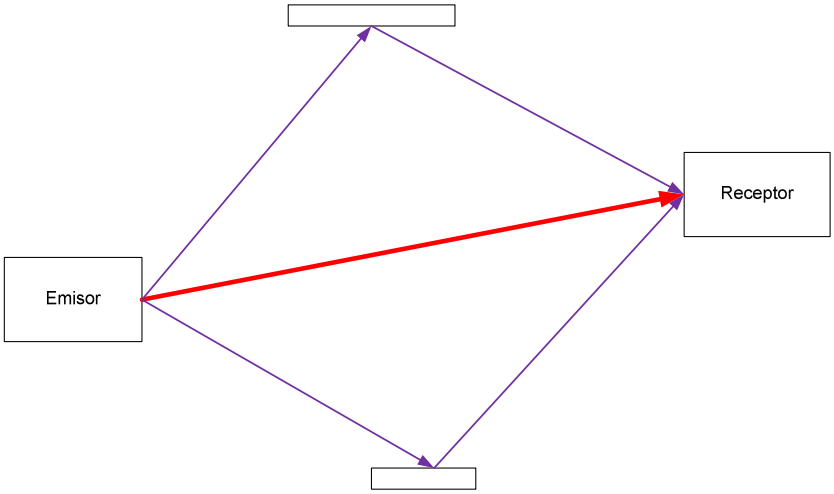


Figura 3.3. Fenómeno de multitrayectorias (dos trayectorias)

Como es de esperarse, debido al fenómeno de multitrayectorias, el aumento de la velocidad de transferencia de datos a través del aire produce fluctuaciones en la amplitud y fase de las señales recibidas, sobre todo si esos sistemas tienen un esquema de utilización espectral de alta eficiencia o utilizan esquemas de diversidad. Utilizar estos esquemas da pie a que se originen los siguientes fenómenos:

Desvanecimiento multitrayecto: Cuando las ondas de las señales multitrayecto tienen distinta fase con respecto a la onda de la señal con línea de vista (componente dominante), la señal resultante es una suma de una componente dominante con otras de menor intensidad, obteniendo una señal intensidad variable. A este fenómeno se le llama desvanecimiento tipo Rice o desvanecimiento Rápido. En el caso en que se reciben componentes similares (multitrayectorias de amplitud similar), ya que no existe línea de vista, el desvanecimiento se le llama de tipo Rayleigh.

Ensanchamiento en Retardos Multitrayecto: Como el trayecto por donde circula la señal contiene múltiples elementos que producen la reflexión de ésta, los símbolos transmitidos y reflejados llegan al receptor a distintos tiempos, lo que puede llegar a producir Interferencia Inter Simbólica o ISI. Esta ISI es la interferencia entre símbolos enviados en instantes distintos, pero que llegan al mismo tiempo dado por los retardos introducidos por las reflexiones. Para evitar que se produzca esta ISI, la velocidad de transmisión debe ser mucho más pequeña con respecto a la inversa de raíz cuadrática media de la medida del ensanchamiento de retardos. A esta medida se le llama ancho de banda de coherencia. Si se desea enviar información a una velocidad mucho menor que el ancho de banda de coherencia, se dice que el canal es Plano o de Ancho de Banda Estrecho. Pero en sistemas actuales, se desea enviar información a una velocidad mayor al ancho de banda de coherencia, por lo que se debe hacer el estudio en los denominados Canales Selectivos en Frecuencia o Canal de Banda Ancha. Para poder realizar una comunicación confiable en Canales de Banda Ancha se utilizan técnicas de **ecualización**, las cuales se encargan de disminuir el efecto del multitrayecto, el cual es uno de los objetivos de estudios en este trabajo, lo que modifica el modelo de sistema de telecomunicaciones presentado anteriormente:

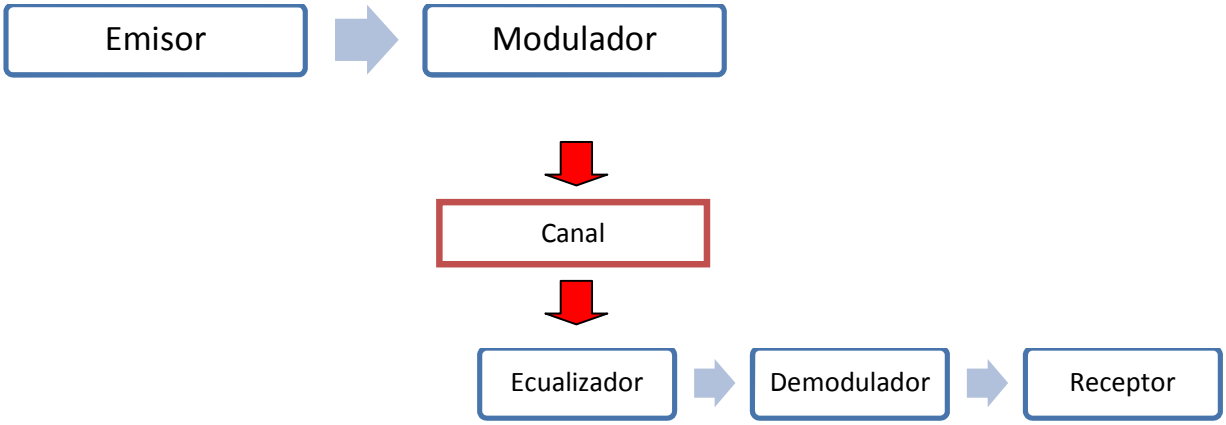
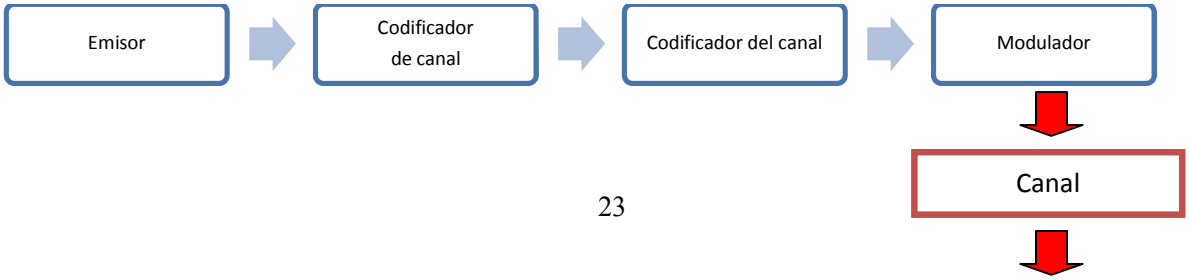


Figura 3.4. Modelo de un sistema de comunicación con ecualizador

El ecualizador tiene como principal objetivo mitigar la ISI. Existen varios tipos de ecualizadores como los de ML (Maximum Likelihood, los cuales maximizan la probabilidad de la secuencia de salida o minimizan la probabilidad de error en la salida) o los DFE (que intentan estimar parámetros que permitan eliminar la ISI). Para este estudio se considerarán ecualizadores de tipo ML, dado que son los que han presentado mejor desempeño frente a los demás, sobretodo en su uso en esquemas turbo.

Otra componente importante se presenta en el modelo de comunicación digital inalámbrica es el codificador-decodificador de canal, dado que es posible recibir diferencias entre las secuencias enviadas por un emisor y las que arriban al receptor, estas diferencias se llaman errores. Se producen principalmente por la presencia de ruido en el canal de comunicación existente entre el conjunto emisor-receptor. Para poder recuperar la secuencia original de datos enviada es necesario hacer una codificación a la entrada del canal, de modo que el receptor sea capaz de detectar y corregir los errores producidos durante la comunicación. El codificador introduce redundancia en los datos de forma que el receptor sea capaz de reconstruir (decodificar) la palabra original de manera confiable. Estos codificadores deben contar con la característica de introducir la menor cantidad de datos redundantes posibles para aprovechar de mejor manera el canal, que a su vez permitan reconstruir la palabra de forma eficaz y con un costo computacional razonable. Se pueden encontrar distintos tipos y algoritmos de codificación de canal, los cuales se dividen principalmente en 2 categorías: Codificación lineal en bloques y codificación convolucional. Ambas categorías han sido desarrolladas de forma separada obteniendo algoritmos muy eficientes y que actualmente están implementados en tecnologías de uso diario.

Así es como en el modelo presentado en la Figura 3.4 es necesario incorporarle el par codificador-decodificador para que la información enviada sea capaz de aguantar el ruido presente en el canal. Este modelo se observa en la Figura 3.5:



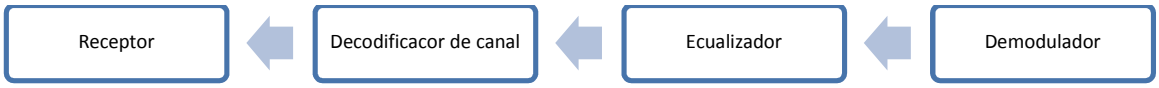


Figura 3.5. Modelo general de un sistema de telecomunicaciones usado en este estudio.

En los párrafos anteriores se explicó la función de los bloques: ecualizador, modulador y codificación-decodificación de canal para establecer un modelo general usado en telecomunicaciones, pero existen otros bloques que se podrían introducir en este modelo, los cuales no son de estudio para este trabajo y solamente se hará alusión a ellos para tener presente su existencia de manera de no pasarlos por alto. Además del codificador de canal en el receptor y el decodificador de canal en el receptor, existe otro tipo de par codificador-decodificador presente en los modelos actuales de telecomunicaciones; los llamados codificadores-decodificadores de fuente, los cuales se encargan de representar de manera eficiente los símbolos del alfabeto fuente en otro alfabeto. Esto se logra teniendo un conocimiento de las probabilidades de cada uno de los símbolos del alfabeto fuente. Algunos ejemplos de codificación de fuente son: el algoritmo de Huffman y el algoritmo de Lempel-Ziv. Otro conjunto presente en el modelo de telecomunicaciones es el compresor en el emisor y el descompresor en el receptor. El compresor cumple la tarea de reducir la longitud de datos de un conjunto de datos, sin modificar su significado y asimismo, el descompresor se encarga de deshacer el proceso realizado en el emisor.

El modelo de la Figura 3.5 ha sido con el que se había trabajado durante mucho tiempo, el que básicamente se caracteriza por realizar cada una de las tareas de los bloques de forma separada e independiente, por lo es posible introducir mejoras tecnológicas en cada uno de estos bloques. Es posible observar que éste modelo presenta cierto tipo de linealidad con respecto al recorrido que hace la señal al momento de ser emitida hasta ser recibida, debido a que una vez que ésta ha pasado por un bloque entonces hay vuelta atrás. Los códigos turbo introducen la idea de que un bloque puede mejorar su desempeño usando la información que ha realizado el bloque que le sigue, estableciendo una especie de retroalimentación entre bloques. Por ejemplo el ecualizador corrige algunos símbolos que venían errados por efectos del ISI y envía esta información al decodificador, el cual a su vez corrige otros símbolos afectados por el AWGN. Estos símbolos son enviados de vuelta al ecualizador para que vuelva a corregir otros símbolos errados, produciendo la retroalimentación. Este modelo será explicado en mayor detalle, enfocándose principalmente en el modelo presentado en este trabajo de título.

3.2 Introducción al concepto de códigos turbo.

El algoritmo presentado en este trabajo de título se basa en el estudio de ecualizadores turbo, los cuales a su vez, se basan en los códigos turbo. En esta sección se propone entregar una pequeña explicación del funcionamiento de los códigos turbo, para explicar más fácilmente como trabaja el ecualizador turbo, el cual es muy similar con el código turbo con algunas diferencias. Finalmente se explicará cómo funciona el esquema de ecualización log-MAP no binaria concatenado con un decodificador LDPC no binario, formando un esquema de turbo.

Los códigos turbo fueron descubiertos en 1993 por Claude Berrou et al. en[1]. Estos códigos se clasifican como una evolución de los ya conocidos hasta esa fecha como los códigos convolucionales. El esquema turbo se basa principalmente en la concatenación de códigos convolucionales recursivos y el decodificador trabaja con decodificadores idénticos ubicados en forma paralela, usando una regla de retroalimentación. El codificador trabaja de la siguiente forma:

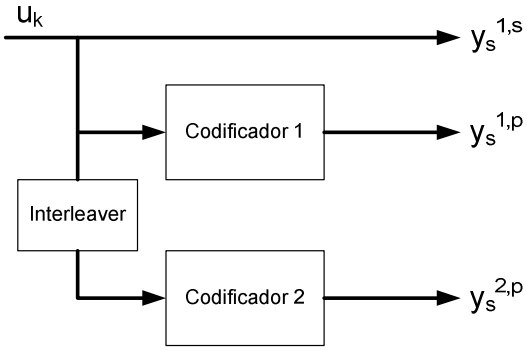


Figura 3.6. Codificador turbo

Entonces, la información enviada por cada bit o símbolo correspondería, según la Figura 3.6, a $y_s^{1,s}$ que es el bit sistemático y los bits $y_s^{1,p}$ y $y_s^{2,p}$ que son los bits de paridad. El término sistemático se refiere a aquellos códigos en los que una de sus ramas generadores de los símbolos de código emite justamente los bits de datos originales. Este esquema se podría extender para n codificadores, transmitiendo mayor cantidad de información adicional por cada bit sistemático.

Los bits transmitidos son modificados por el efecto del canal y llegan al extremo decodificador los bits correspondientes al bloque codificador. Éstos se disponen como en la Figura 3.7 para ejecutar el esquema de decodificación turbo, el que trabaja de forma iterativa, es decir, que la información que entrega el decodificador 1 pasa al decodificador 2(después de pasar por un interleaver, de acuerdo al esquema de codificación) y a su vez la información del decodificador 2 pasa al decodificador 1 (pasando ahora por un deinterleaver) para mejorar el poder de decodificación y corregir usando la información de la iteración anterior. Este esquema puede ejecutarse para una cantidad razonable de iteraciones, para que finalmente se haga la decisión sobre los bits que llegan. Como el objetivo de esta memoria no es precisamente explicar el funcionamiento de los códigos turbo, sino un esquema de ecualización basado en códigos turbo, se explicará esto de manera superficial. Puede encontrar mayor información en [14].

El esquema usando en la decodificación es el siguiente:

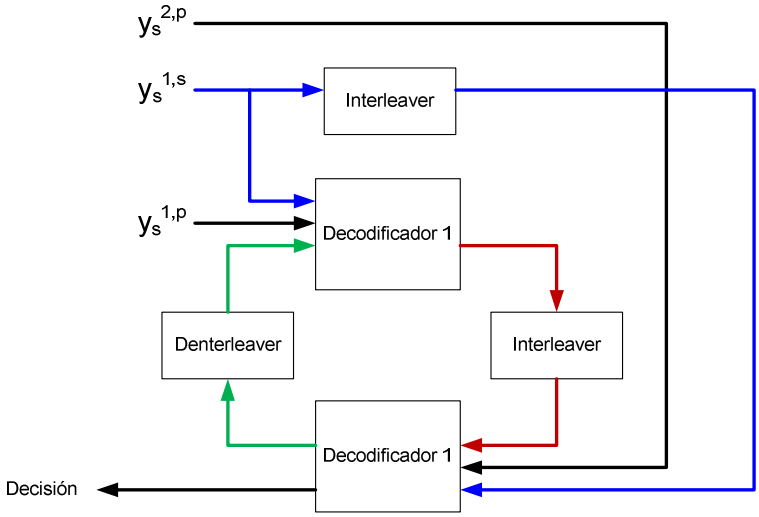


Figura 3.7. Decodificador turbo

Este esquema llamado iterativo puede usarse bajo cualquier código que se adecue, como por ejemplo el algoritmo MAP o el algoritmo SOVA (algoritmo de Viterbi con salidas soft). Lo importante de este esquema planteado es que se rompe con la linealidad del recorrido de la señal dentro del esquema general de telecomunicaciones mostrado en la Figura 3.5, ya que la información de decodificación retorna a los decodificadores para ser corregida. Este mismo esquema se utilizó para el diseño de los ecualizadores turbo.

3.3 Ecualizador turbo

La ecualización turbo fue primero propuesta por Douillard en 1995 para un sistema con codificación convolucional y modulación BPSK, como se puede ver en la Figura 3.8:

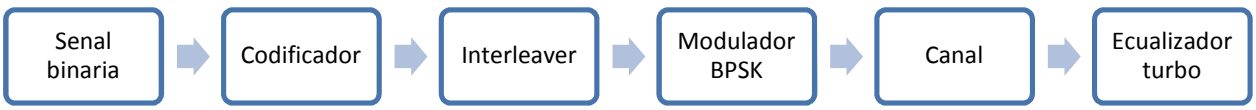


Figura 3.8. Esquema de un ecualizador Turbo propuesto por Douillard

En este esquema, la ecualización turbo sigue una estructura como la de la Figura 3.9:

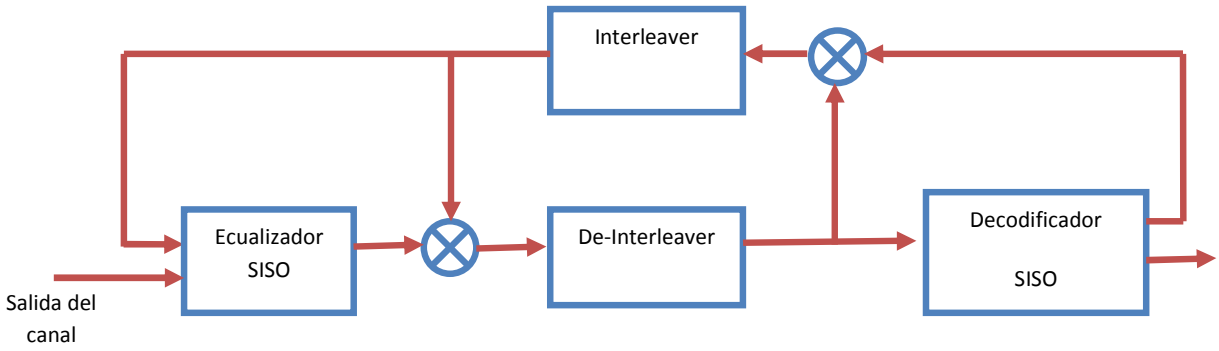


Figura 3.9. Esquema de ecualización turbo de Douillard

Como puede verse en la Figura 3.9, la filosofía del ecualizador turbo es realizar de forma iterativa el proceso de ecualización y decodificación usando información proporcionada por un bloque para que el otro bloque pueda hacer su tarea de manera mejor, utilizando un esquema de retroalimentación. Cabe señalar que la idea de códigos turbo se puede utilizar cambiando cualquiera de los bloques del esquema original propuesto por Douillard, utilizando códigos por bloques en vez de convolucionales por ejemplo, conservando la idea de retroalimentación entre los bloques. Para implementar el ecualizador SISO y el decodificador SISO es posible se implementar mediante varios tipos de algoritmos, como el Soft-Output Viterbi Algorithm (SOVA) o el Maximum A Posteriori (MAP). En este trabajo se utilizarán los bloques basados en el algoritmo MAP, dado que se adecúa de mejor al modelo planteado en este estudio y que ha presentado mejores resultados.

Para comprender el funcionamiento del esquema de ecualizador turbo, es necesario realizar un análisis más detenido acerca del cada uno de los bloques que componen este modelo, los que posteriormente se implementarán mediante simuladores computacionales y obtener resultados según distintas condiciones a las que se encuentra sometido el esquema.

3.4 Canal de transmisión

El canal es el medio por el cual se transmite la onda que lleva la información enviada por el transmisor. Este medio puede ser un cable de cobre, un cable de fibra óptica, o como en este caso de estudio, la atmósfera terrestre y los obstáculos presentes como montañas, edificios, automóviles, personas o cualquier objeto que se interponga entre el terminal emisor y receptor.

El desarrollo de un modelo de canal lo más realista posible ha sido uno de los desafíos más grandes para los investigadores en el terreno de las telecomunicaciones. Los modelos más

clásicos y que mayormente se han utilizado para simular los efectos producidos en la transmisión vía aérea son los modelos de Clarke y posteriormente de Jakes.

Para este trabajo de título se utilizará un modelo de canal tipo Tapped Delay Line (TDL) Gaussiano, pero se le aplicarán algunas modificaciones por motivos que se explicarán más adelante.

El modelo TDL Gaussiano corresponde a un tipo de canal propuesto por S. K. Leung-Yan-Cheong y Martin E. Hellman en [15] extendiendo a un contexto Gaussiano el modelo original de canales discretos sin memoria propuesto por Wyner's. Este modelo ha sido estudiado y perfeccionado principalmente en la universidad de Standford, debido a la simplicidad de implementación de éste, versus los resultados satisfactorios y muy similares que se obtienen comparados con modelos más complejos como el de Jakes. Un modelo TDL tiene al menos un bloque de retardo y un bloque ponderador. La señal de entrada pasa por un ponderador y al mismo tiempo pasa por un bloque de retardo (*delay*), obteniendo la señal anterior a la actual. Esta señal retardada también pasa por otro ponderador y se obtiene la suma de la señal actual ponderada y la señal anterior, también ponderada. Con esto se obtienen 2 salidas para una señal y a cada una de estas salidas se les denomina derivaciones (*taps*). Finalmente estas salidas se suman para obtener la señal resultante. El caso más general de un TDL incorpora una derivación después de cada bloque de retardo lo que entrega un filtro causal de respuesta al impulso finita o FIR. Éste es causal ya que la salida no depende de las muestras futuras. A la salida de este filtro FIR se le adiciona ruido blanco Gaussiano, lo que sirve para representar las interferencias de otros usuarios.

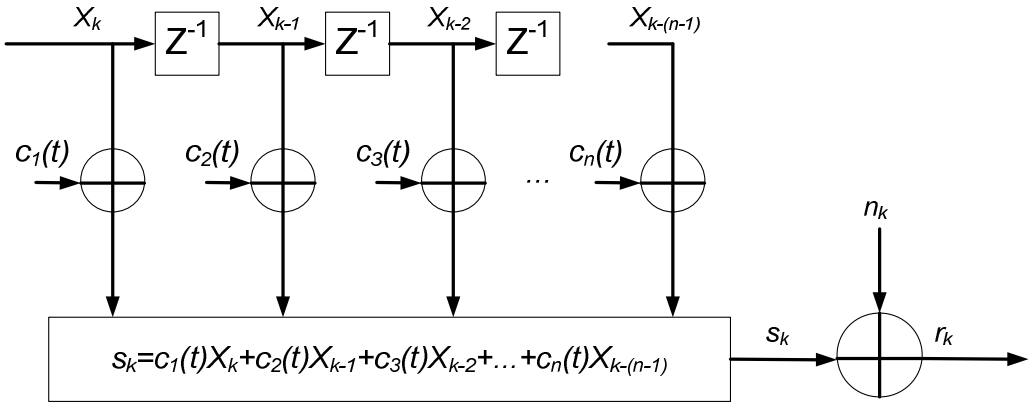


Figura 3.10. Modelo del canal TDL

El modelo de la Figura 3.10 representa de buena manera los fenómenos de multitrayectoria y las interferencias aleatorias que pudiera encontrar la señal recibida dentro del canal de comunicaciones, por lo cual este modelo se ha utilizado para simular algoritmos en comunicaciones Inalámbricas y Wimax, tal y como se muestra en [16].

En este trabajo, se asume conocimiento total del canal en el extremo receptor, supuesto que no se cumple en su totalidad, pero existen métodos para realizar estimaciones del canal según este modelo. Adicionalmente se asumirá un canal estático, de 3 caminos como se muestra en la

Figura 3.11. No obstante, a pesar de realizar tantos supuestos, este modelo ha presentado resultados muy satisfactorios comparados con modelos más sofisticados y cuya implementación puede llegar a ser muy compleja.

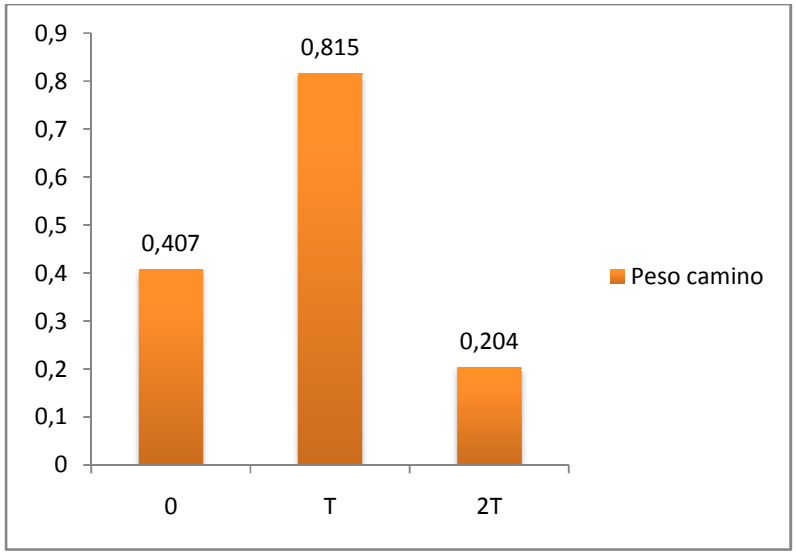


Figura 3.11. Canal de TDL con tres derivaciones

En la Figura 3.11 se puede ver un modelo estático de tres caminos. La manera de interpretar este modelo es que al enviar un dato, al extremo receptor llega el dato y adicionalmente llegan los rebotes de esta señal o reflexiones de ésta, con un retardo de T y 2T. Estos caminos adicionales se conocen como derivaciones, por lo que el modelo de canal usado en este trabajo consta de tres derivaciones no equiespaciados. En la Figura 3.12 que modela el funcionamiento de este canal:

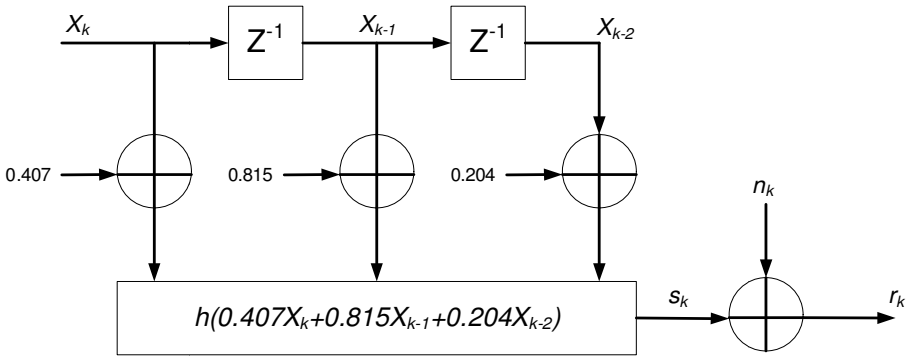


Figura 3.12. Funcionamiento canal

De este modo la señal a la salida del canal sería: $r_k = n_k + s_k$ donde n_k es el ruido blanco gaussiano y s_k es la señal recibida correspondiente a la suma de los 3 caminos ponderados por su peso.

El modelo TDL de canal puede ser visto como un codificador convolucional, donde la información se codifica no solamente tomando en cuenta la entrada actual de bits, sino también de los bits anteriores. Es por ello que es posible representar este canal usando los mismos modelos usados para la representación de codificación convolucional, estos modelos son los llamados diagramas de estado y diagramas de Trellis (mayor información ver [11]) Por ejemplo, para un canal de dos derivaciones, usando modulación BPSK (2 símbolos), tiene el siguiente modelo TDL:

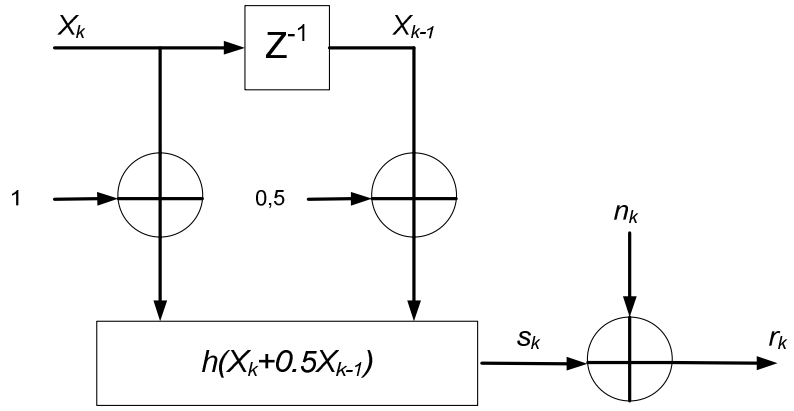


Figura 3.13. Modelo canal de dos derivaciones

Para dibujar el diagrama de Trellis de este canal es necesario tener en cuanta todas las salidas posibles del canal, en ausencia de ruido. Como se está usando modulación BPSK, los símbolos considerados son 0 y 1 (-1 y +1 en formato polar), y además como el modelo de canal toma la entrada actual de bits y la anterior para obtener la salida, entonces los estados posibles a considerar son cuatro: 00, 01, 10 y 11. Las salidas serán calculadas usando formato polar:

Tabla 3.1. Información de salida para canal con dos derivaciones

Entrada actual (x_k)	Entrada anterior (x_{k-1})	Salida del canal (sk)
-1	-1	1x(-1) + 0,5x(-1) = -1,5
-1	+1	1x(-1) + 0,5x(+1) = -0,5
+1	-1	1x(+1) + 0,5x(-1) = +0,5
+1	+1	1x(+1) + 0,5x(+1) = +1,5

Entonces, el diagrama de Trellis de este canal, resume todos los estados presentes dentro de un canal. El diagrama de Trellis de este canal se muestra en la Figura 3.14:

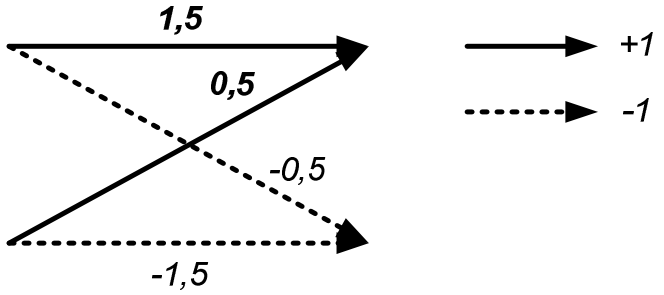


Figura 3.14. Diagrama de Trellis de un canal dos derivaciones

A diferencia del modelo de diagrama de estados, el uso de diagramas de Trellis puede utilizarse para representar la salida esperada de un canal, dada una secuencia recibida de bits o símbolos, la cual se puede extender todo lo que se requiera, entregando a su vez un diagrama de Trellis que se expande según esa secuencia recibida. En cambio, en el diagrama de estados no se podría observar bien cual es la secuencia enviada ni la salida recibida, ya que la notación se tornaría algo engorrosa y confusa. Por ejemplo, sea secuencia {0, 1, 1,0, 1, 0, 1}, la que enviada en formato polar correspondería a {-1, 1, 1, -1, 1, -1, 1}, entonces, el diagrama de Trellis correspondiente al envío de esta secuencia y la salida esperada (sin ruido) correspondiente se muestra en la Figura 3.15:

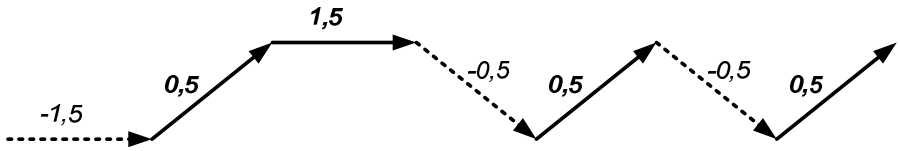


Figura 3.15. Diagrama de Trellis secuencia {-1,1,1,-1,1,-1,1}

Entonces, se espera que para la secuencia {-1, 1, 1, -1, 1, -1, 1}, la salida sea {-1.5, 0.5, 1.5, -0.5, 0.5, -0.5, 0.5} en ausencia de ruido y debido al efecto de multitrayectorias.

Es importante señalar que el diagrama de Trellis es único para cada secuencia y se puede extender conforme el largo de ésta. Además, esta representación puede ser utilizada para modelos de canal más complejos y con mayor cantidad de derivaciones, además de extender el modelo a un esquema de modulación que involucre una mayor cantidad de símbolos, permitiendo dibujar diagrama de Trellis de mayor tamaño que ayuden a comprender de forma más simple el funcionamiento de un código.

3.5 Codificador – Decodificador de Canal

3.5.1 Codificador

Un par codificador-decodificador tiene como función aplicar un tratamiento a una señal de modo que ésta pueda ser reconstruida después de haber sufrido alguna alteración debido a algún tipo de interferencia o ruido, y así reducir el número de errores en los bits recibidos. El tratamiento de esta señal se fundamenta en introducir bits redundantes de forma selectiva. El costo de introducir estos bits adicionales para proteger la información es reducir la tasa efectiva de información en el ancho de banda.

Existen dos principales tipo de codificadores de canal; los codificadores por bloques y los codificadores convolucionales y con ello, además existen muchos tipos de codificadores por bloques y convolucionales. Los codificadores por bloques principalmente toman bloques de información, por ejemplo k-bits, y producen otro bloque de n-bits. Estos bits son producidos usando propiedades aritméticas y algebraicas, a partir de ello, se usan diversas técnicas para decodificar los códigos y estimar la señal. Los tipos de códigos por bloques más conocidos son los tipo Hamming, Golay, BCH, Reed Solomon o los que se estudiarán en este trabajo, los llamados códigos de paridad de baja densidad o LDPC (*Low density parity check*). Los códigos convolucionales, a diferencia de loa anteriores, realizan la corrección de errores a "tiempo real", es decir, que realizan la corrección de errores a medida que va llegando la secuencia de bits. El esquema presentado en la Figura 3.7 para decodificación turbo está implementado usando el algoritmo de BAHL descrita en el artículo de L. R Bahl et al [19]. Este algoritmo también se denomina algoritmo BCJR (por la iniciales de sus autores) o con su nombre más conocido, el algoritmo MAP o de máxima probabilidad a posteriori.

Como se dijo anteriormente, un canal puede ser modelado a través de TDL, donde se puede comparar un cierto modelo de canal con un codificador convolucional, con algunas diferencias. Lo que intenta hacer un ecualizador es simplemente realizar una especie de "decodificación" de un canal, que puede variar en el tiempo. Entonces, como el ecualizador intenta hacer esta tarea, lo correspondiente entonces es tratar de igualar el modelo de un canal calculado en el extremo receptor, a la señal que se recibe realmente y tratar de maximizar la probabilidad de que la señal se parezca al canal. Para ello se puede utilizar en el ecualizador un algoritmo similar al de decodificación para códigos convolucionales. Es por ello que se ha pensado en usar el algoritmo log-MAP, que es una versión modificada del algoritmo MAP, pero que disminuye la carga computacional ya que usa las propiedades de los logaritmos.

3.6 Ecualización

3.6.1 Procesos *soft* y *hard*

Antes de explicar el proceso de ecualización realizado por el algoritmo log-MAP, es importante mencionar que este procedimiento se puede hacer de manera *soft* o *hard*. Estos términos se diferencian en el nivel de cuantización sobre el cual se está haciendo el manejo de decisiones sobre las señales que se están recibiendo y decodificando, es decir, si se toma un umbral dependiendo del signo para establecer si una señal es un cero o un uno (binario), entonces si en el receptor se recoge una muestra de valor -1.345, entonces el decodificador aceptará que se recibió un 0 (-1 en formato polar) o en cambio si se recibiera un 0.14, la decisión *hard* tomada diría que el bit recibido es un 1 (+1 en formato polar) y toma esa información adicional sobre qué tan alejado o no se está del umbral de decisión no se toma en cuenta. Lo mismo pasa en el decodificador. El proceso *hard* tiene la ventaja de entregar menor carga al momento de procesar

las señales. Los procesos de decodificación *soft* en cambio, toman en cuenta los números decimales y aprovecha la información adicional antes de tomar la decisión si un bit recibido era realmente el que determinó el umbral de decisión. El método de decodificación que se usará en este proyecto de trabajo de título será de tipo *soft*, dado que se han obtenido mejores resultados que con procesos *hard*, con un costo computacional mayor. El algoritmo log-MAP se basa en procesos de Markov y redes Bayesianas, es por ello que es necesario entregar una reseña de algunas propiedades de probabilidades que se usarán.

Revisión de probabilidades

El teorema de Bayes para probabilidad condicional de un evento viene dado por:

$$P(A,B) = P(A)P(B|A) \tag{1}$$

Que se puede leer como: La probabilidad de que ocurra A y B es igual a la probabilidad que ocurra A, es decir P(A), multiplicado por la probabilidad que ocurra B dado que el evento A ya ha ocurrido, o sea P(B|A). Si hay 3 eventos independientes, llamados A, B y C, entonces el teorema de Bayes entrega:

$$P(A,B|C) = P(A|B,C)P(B,C) \tag{2}$$

Que es una propiedad útil para el desarrollo del algoritmo.

Además, es posible observar que la probabilidad de que ocurra A y B es igual a la probabilidad que ocurra B y A, de este modo se puede reescribir la ecuación (1) para A o B:

$$P(A,B) = \underbrace{P(A|B)}_{\substack{\text{Pbb.}\\ \text{a-posteriori}\\ \text{evento A}}} \underbrace{P(B)}_{\substack{\text{Pbb.}\\ \text{a-priori}\\ \text{evento B}}} = \underbrace{P(B|A)}_{\substack{\text{Pbb.}\\ \text{a-posteriori}\\ \text{evento B}}} \underbrace{P(A)}_{\substack{\text{Pbb. a-}\\ \text{priori}\\ \text{evento A}}} \tag{3}$$

La probabilidad a priori, se refiere a un evento que ocurre de manera independiente, además de indicar que el evento es conocido independientemente de la experiencia, mientras que el término a posteriori hace referencia a un evento condicionado a la base de experiencia y no ocurre de manera independiente. De este modo, se pueden escribir las probabilidades a posteriori (o APP) como:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4}$$

El algoritmo MAP y log-MAP, ayudan a tomar las decisiones basándose en los conocimientos a priori y a posteriori basados en los bits recibidos, usando el indicador LLR, que tiene la particularidad de que su polaridad indica el signo del bit, mientras que su amplitud cuantifica la probabilidad de que la decisión sea correcta.

3.6.2 El algoritmo log-MAP

A continuación se describirá el funcionamiento del algoritmo log-MAP, donde es importante señalar que se le eligió dado que proporciona un mejor desempeño que el algoritmo MAP y el SOVA (*Soft Output Viterbi Algoritm*, basado en el algoritmo de Viterbi pero con salidas *soft*) además de entregar una menor carga computacional (comparado con otras versiones del algoritmo MAP) al transformar multiplicaciones y divisiones en sumas y restas (propiedades de los logaritmos), además de usar algunas aproximaciones. Este algoritmo puede ser usando tanto en el bloque de ecualizador SISO o el decodificador SISO que se muestra en el esquema de la Figura 3.9, teniendo sus pequeñas diferencias según la operación que se esté realizando.

Para comenzar a describir el algoritmo log-MAP hay que definir el Log-likehood Ratio (tasa de similitud logarítmica o LLR) utilizado para enviar información entre bloques, de forma iterativa. Este término se define como:

$$L(u_k) \triangleq \ln\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right) \tag{5}$$

Por lo tanto, el LLR de un bit de datos u_k, denotado por $L(u_k)$, se define como el logaritmo de la tasa entre las probabilidades de un bit tomando dos posibles valores +1 y -1. Este indicador es una función centrada en cero y que muestra qué tan predominante es una probabilidad de un valor sobre la otra, es decir, si $L(u_k) \gg 0$, entonces $P(u_k = +1) \gg P(u_k = -1)$ y se tiene la certeza que el bit u_k =+1, y viceversa. También es posible expresar las probabilidades en función de del LLR, lo que se indica en la ecuación (6):

$$P(u_k = \pm 1) = \left(\frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}}\right) \cdot e^{\pm L(u_k)/2} \tag{6}$$

De igual forma, es posible definir el LLR basado en probabilidades condicionales. Podemos así definir un indicador que defina la probabilidad de que un bit $u_k = \pm 1$, dada una secuencia recibida y. Este término es el que se llamara LLR condicional, definido en la ecuación (7):

$$L(u_k|y) \triangleq \ln\left(\frac{P(u_k = +1|y)}{P(u_k = -1|y)}\right) \tag{7}$$

Este indicador se utiliza para identificar el signo de una señal enviada, dado que la secuencia recibida toma cierto valor, el cual corresponde a un tipo de métrica usado en la mayoría de los esquemas de corrección de errores. Además, este algoritmo puede ser utilizado tanto en el ecualizador como en el decodificador, así que en este punto se tratará de generalizar el algoritmo para que sea extendido según el uso que se le quiera dar. Por ejemplo, en el ecualizador u_k representa la información del bit con codificación de canal, mientras que en el decodificador u_k es el bit de datos de la fuente.

Por otra parte, se puede definir el LLR para que ocurra una secuencia y_k, dado que se envió una secuencia x_k que puede ser +1 o -1. Entonces, el LLR para estos eventos se muestra en la ecuación 8:

$$L(u_k|y) \triangleq \ln\left(\frac{P(y_k|u_k=+1)}{P(y_k|u_k=-1)}\right) \tag{8}$$

El algoritmo MAP y log-MAP calcula para un bit u_k, la probabilidad que ese bit sea +1 o -1, dada una secuencia recibida y. Esto es equivalente a calcular el LLR $L(u_k|y)$ de la ecuación (7), utilizando el teorema de Bayes mencionado se puede reescribir:

$$L(u_k|y) \triangleq \ln\left(\frac{P(u_k=+1|y)}{P(u_k=-1|y)}\right) = \ln\left(\frac{P(u_k=+1,y)}{P(u_k=-1,y)}\right) \tag{9}$$

Esta formulación de LLR incluye las probabilidades conjuntas entre los bits recibidos y los bits de información (fuente). Si se utiliza un modelo de canal tipo TDL (o un codificador convolucional o recursivo RSC), debido a su naturaleza, es posible identificar cualquier camino utilizando estos tres términos:

1. Estado inicial o *s'*
2. Estado final o *s*
3. Bit de entrada, que se ha llamado *y*.

De este modo, se puede saber a qué camino recorrido por el diagrama de Trellis corresponde al bit que se recibió, si es que se saben estos parámetros o al menos, sólo dos de ellos.

Para un sistema que comprenda un número *X* de estados del diagrama de Trellis entonces habrán *X* transiciones para los bits u_k=+1 y u_k=-1, las cuales son mutuamente exclusivas (no pueden ocurrir simultáneas). En la ecuación (9) sólo se tiene conocimiento de la secuencia recibida *y*, pero, ahora se sabe que es válido decir que la probabilidad que un bit u_k = 1, es lo mismo decir que cual ha sido el camino recorrido por el diagrama de Trellis. De este modo, se puede replantear la ecuación para calcular el LLR como:

$$L(u_k|y) = \ln\left(\frac{(\sum_{(s',s)\Rightarrow u_k=+1}^{X} P(s',s,y))}{(\sum_{(s',s)\Rightarrow u_k=-1}^{X} P(s',s,y))}\right) \tag{10}$$

Como se dijo anteriormente, *X* denota la cantidad de transiciones del diagrama de Trellis correspondientes al bit u_k, por lo que se deben sumar todas transiciones correspondientes a ese bit para determinar el LLR correspondiente a esa secuencia. El término *(s',s)=>u_k=±1* corresponde al conjunto de transiciones del estado *s'* al estado *s* causado por el bit u_k=±*1*, según corresponda al caso.

Para explicar este algoritmo se usará un diagrama de Trellis Figura 3.16 donde se tienen las transiciones para los bits u_k=*+1* y u_k=*-1*, y los estados corresponden a 00, 01, 10 y 11 (indicados al principio y al final de las transiciones) y los números ubicados sobre las líneas de transición que tienen la forma *X/YY* corresponden al bit de entrada al codificador o canal (*X*) y al la salida entregada por el codificador o canal (*YY*, que puede variar según el modelo):

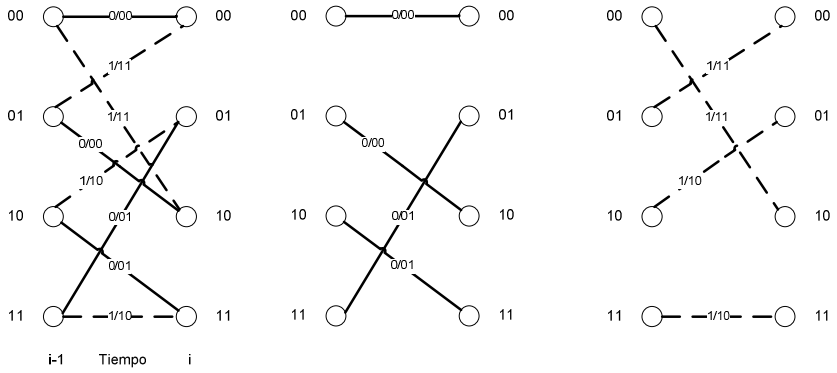


Figura 3.16. Diagrama de Trellis usado como ejemplo

Ahora, si se toma en cuenta los bits anteriores y los bits posteriores tomando como referencia un bit u_k, se obtiene el siguiente diagrama de Trellis:

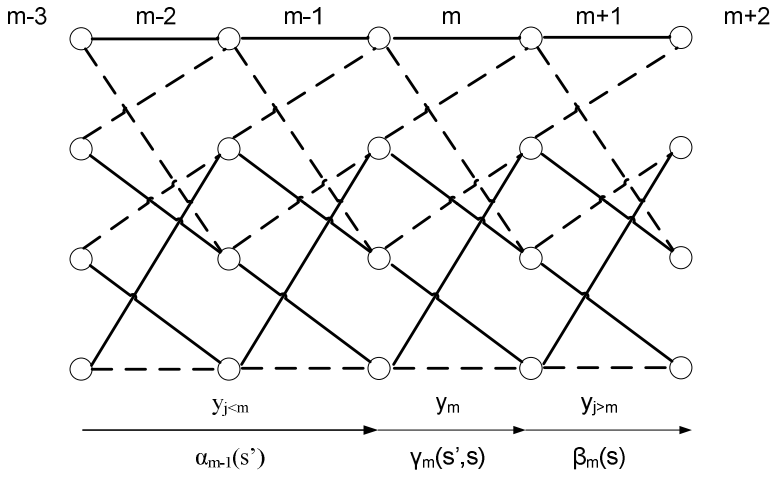


Figura 3.17. Diagrama de Trellis de ejemplo, usando entradas anteriores y posteriores

Por lo que para calcular el LLR del bit u_k, se deben tomar las secuencias a partir de u_{k-1} hacia atrás y de u_{k+1} en adelante. Para el sistema de la Figura 3.16, hay M=2 transiciones saliendo de cada estado, además de X=4 transiciones para el bit u_k =+1, cada una de estas transiciones vienen dadas por $(s',s) \Rightarrow u_k = +1$, de igual manera ocurre para u_k =-1. En la Figura 3.17 se puede observar que hay unos términos asociados a los bits anteriores y posteriores a u_m, a estos términos se les ha nombrado $\alpha_{k-1}(s')$, $\beta_m(s)$, y $\gamma_m(s',s)$. Es posible ver que los valores de los términos $\alpha_{m-1}(s')$ dependen de los posibles valores anteriores de $\alpha_{m-1}(s')$ o estados del trellis. Además, dependen de los valores de las entradas en el tiempo m-1, que viene dadas por los valores de $\gamma_{m-1}(s',s)$. Entonces, se puede decir que los $\alpha_{m-1}(s')$ se calculan "hacia delante", ya que sólo dependen de sus valores anteriores. Por otra parte los $\beta_m(s)$, dependen de sus posibles valores o entradas futuras, es decir $\beta_{m-1}(s)$ depende de los posibles valores de $\beta_m(s)$ y de la entrada en ese tiempo, es por ello que se puede decir que los $\beta_m(s)$ se calculan "hacia atrás".

En la publicación de L.R. Bahl et al [19], se demostró que usando el teorema de Bayes, se puede separar la secuencia recibida en tres términos, tomando como referencia un bit y_m:

- y_m representa el valor de un bit recibido en un tiempo presente m.
- $y_{j<m}$ representa el valor de los bits recibidos anteriormente al bit de referencia y_m.
- $y_{j>m}$ representa el valor de los bits que siguen el en tiempo al bit de referencia y_m.

Es así como se puede representar la ecuación 10 de la siguiente manera:

$$L(u_m|y) = \ln\left(\frac{(\sum_{(s',s)\Rightarrow u_m=+1}^{X} P(y_{j<m}, s')P(y_m, s|s')P(y_{j>m}, s)}{(\sum_{(s',s)\Rightarrow u_m=-1}^{X} P(y_{j<m}, s')P(y_m, s|s')P(y_{j>m}, s)}\right) \qquad (11)$$

Se tomará entonces de ahora en adelante la siguiente notación:

- $\alpha_{m-1}(s') = P(y_{j<m}, s')$: Se define como la probabilidad de estar en el estado *s'*, tomando en cuenta todos los bits recibidos antes de *m*, es decir, los bits de salida de secuencia $y_{j<m}$.
- $\gamma_m(s', s) = P(y_m, s|s')$: Se define como la probabilidad de haber recibido la secuencia y_m, al instante *m*, dado que estamos en el estado *s'* del diagrama de Trellis en el instante *m-1*.
- $\beta_m(s) = P(y_{j>m}, s)$: Se define como la probabilidad de recibir la secuencia futura $y_{j>m}$ en el instante *m*, dado que el estado presente es el estado *s*.

En la Figura 3.18 se puede esquematizar la forma en que dependen los valores de $\alpha_{m-1}(s')$ y $\beta_m(s)$ de los valores de $\gamma_m(s', s)$ y de sus valores anteriores y posteriores respectivamente.

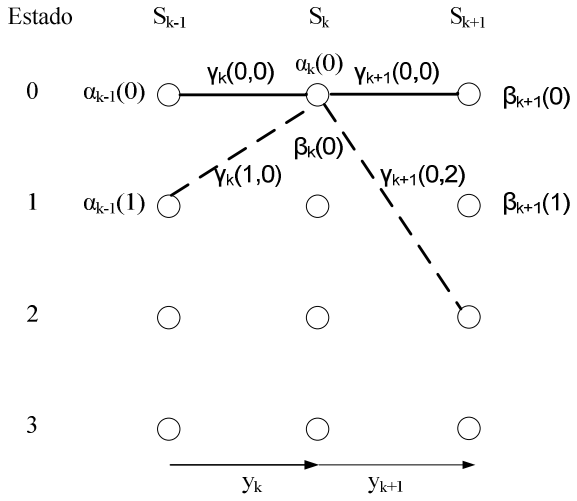


Figura 3.18. Dependencia de entradas anteriores y posteriores

Según la dependencia mostrada en la figura anterior, es posible calcular los valores de los $\alpha_{m-1}(s')$ y $\beta_m(s)$ usando las dependencias que presentan y la entrada de bit que podrían llevar a ese estado. Además, en vez de escribir los estados de trellis como 00, 01, 10 y 11, se toma una nueva notación de la forma 0, 1, 2 y 3 respectivamente, para reducir escritura y evitar confusiones.

En [18] se utilizan propiedades de las probabilidades para obtener los cálculos de los coeficientes $\alpha_k(s')$, $\gamma_k(s',s)$ y $\beta_k(s)$ lo que entrega los siguientes valores para estos parámetros (asumiendo que se tiene un canal Gaussiano y se está usando modulación BPSK):

$$\gamma_k(s',s) = C \cdot e^{(u_k L(u_k)/2)} \cdot \exp\left(\frac{E_b}{2\sigma^2} 2a \sum_{l=1}^{n} y_{kl} x_{yl}\right) \tag{12}$$

$$\alpha_k(s) = \sum_{todos\ los\ s\prime} \gamma_k(s',s) \cdot \alpha_{k-1}(s'), \quad \alpha_0(s) = 1 \tag{13}$$

$$\beta_{k-1}(s) = \sum_{todos\ los\ s\prime} \beta_k(s) \cdot \gamma_k(s',s), \quad \beta_N(s) = 1 \tag{14}$$

Donde los valores y_{kl} y x_{yl} corresponden a los bits individuales recibidos y transmitidos dentro de las palabras $y_k y$ x_y respectivamente.

Hasta aquí se está hablando básicamente del algoritmo MAP, el que necesita calcular los valores de $\alpha_k(s')$, $\gamma_k(s',s)$ y $\beta_k(s)$ para obtener los LLR operando con multiplicaciones, logaritmos naturales, etc. Estos cálculos sobrecargan de gran manera el algoritmo lo que puede tornarlo trabajoso y lento. Es por ello que el algoritmo log-MAP recurre a algunas propiedades para alivianar esta carga computacional.

La siguiente aproximación se usará con frecuencia en este algoritmo, llamada relación del logaritmo Jacobiano:

$$\ln(e^{x_1} + e^{x_2}) = \max(x_1 + x_2) + \ln\left(1 + e^{-|x_1 - x_2|}\right)$$

$$= \max(x_1 + x_2) + \ln\left(1 + e^{-f_{diff}}\right) \tag{15}$$

Para calcular el término $\ln\left(1 + e^{-f_{diff}}\right)$ se puede usar una tabla de consulta (*look-up table*) definida en [14], que se muestra en la Tabla 3.2:

Tabla 3.2. Tabla de consulta para logaritmo Jacobiano

Rango de f_{diff}	$\ln\left(1 + e^{-f_{diff}}\right)$
$f_{diff} > 3.70$	0.00
$3.70 \geq f_{diff} > 2.25$	0.05
$2.25 \geq f_{diff} > 1.50$	0.15
$1.50 \geq f_{diff} > 1.05$	0.25

$1.05 \geq f_{diff} > 0.70$	0.35
$0.70 \geq f_{diff} > 0.43$	0.45
$0.43 \geq f_{diff} > 0.20$	0.55
$f_{diff} \leq 0.20$	0.65

Con esta tabla de consulta se reduce la computación de logaritmos naturales y exponenciales a cálculos de máximos y mapeos de tablas. Ahora se hará la transformación al dominio logarítmico definiendo:

$$A_m(s) \triangleq \ln(\alpha_m(s)) \tag{16}$$

$$B_m(s) \triangleq \ln(\beta_m(s)) \tag{17}$$

$$\Gamma_m(s', s) \triangleq \ln(\gamma_m(s', s)) \tag{18}$$

Entonces, con esta transformación se usará la relación del logaritmo Jacobiano para evaluar la LLR a través de la tabla de consulta. Usando la definición recursiva de $\alpha_m(s)$ mostrada en (16), se obtiene:

$$\begin{aligned} A_m(s) &\triangleq \ln(\alpha_m(s)) \\ &= \ln\left(\sum_{todos\ los\ s'} \gamma_k(s', s) \cdot \alpha_{k-1}(s')\right) \\ &= \ln\left(\sum_{todos\ los\ s'} exp[A_{m-1}(s) + \Gamma_m(s', s)]\right) \\ &= \ln\left(\sum_{todos\ los\ s'} exp[\Upsilon_f(s', s)]\right) \end{aligned} \tag{19}$$

Donde $\Upsilon_f(s', s) = A_{m-1}(s) + \Gamma_m(s', s)$ se define como la métrica de transición para pasar de un estado *s'* a *s*. Además, se utilizará esta notación para que se entienda más la idea del uso de aproximaciones en este algoritmo. Ahora, es posible tomar la suposición de que para llegar a un estado *s*, pueden arribar dos métricas de transición, como en el diagrama de Trellis de la Figura 3.18, por lo que el cálculo de $A_m(s)$ puede escribirse como:

$$\begin{aligned} A_m(s) &= \ln\left(\sum_{todos\ los\ s'} exp[\Upsilon_f(s', s)]\right) \\ &= \ln(exp[\Upsilon_{f1}(s', s)] + exp[\Upsilon_{f2}(s', s)]) \end{aligned} \tag{20}$$

Como $A_m(s)$ es un logaritmo natural de una suma de exponenciales, entonces se puede usar la aproximación del logaritmo Jacobiano, por lo que el cálculo se reduce a lo siguiente:

$$A_m(s) = \ln\left(exp\left[\Upsilon_{f1}(s',s)\right] + exp\left[\Upsilon_{f2}(s',s)\right]\right)$$

$$= \max\left(\Upsilon_{f1}(s',s), \Upsilon_{f2}(s',s)\right) + \ln\left(1 + exp\left[-\left|\Upsilon_{f1}(s',s) - \Upsilon_{f2}(s',s)\right|\right]\right) \quad (21)$$

Si se utilizaran esquemas más complejos, la relación del logaritmo Jacobiano seguiría siendo útil, ya que el cálculo de $A_m(s)$ para mayor número de términos se puede calcular recursivamente, utilizando las mismas definiciones.

Esta relación también se puede aplicar para el cálculo de $B_m(s)$. Para el cálculo de $\Gamma_m(s',s)$ (necesario para calcular los términos anteriores), se puede usar la definición de $\gamma_m(s',s)$ y el teorema de Bayes:

$$\gamma_m(s',s) = P(y_m, s|s')$$

$$= P(y_m|\{s',s\}) \cdot P(s|s')$$

$$= P(y_m|\{s',s\}) \cdot P(u_m) \quad (22)$$

Donde u_m es el bit necesario para que ocurra la transición desde *s'* a *s*, y $P(u_m)$ es la probabilidad a priori de ese bit y $P(y_m|\{s',s\})$ es la probabilidad se reciba una secuencia y_m dado que ocurrió en el instante *m* una transición de *s'* a *s*. Si se asume que se recibe una señal $\widehat{y_m}$ sobre canal Gaussiano, con varianza σ^2 de ruido banco complejo Gaussiano con media cero se obtiene:

$$\Gamma_m(s',s) = ln(P(y_m|\{s',s\}) \cdot P(u_m))$$

$$= ln\left[\left(\frac{1}{\sqrt{2\pi\sigma^2}} exp\left[-\frac{1}{2\sigma^2}(y_m - \widehat{y_m})\right]\right)\right] \cdot P(u_m)$$

$$= ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}(y_m - \widehat{y_m}) + ln(P(u_m)) \quad (23)$$

Con esto, la expresión del LLR de $L(u_m|y)$, bajo el algoritmo log-MAP, queda representada con la ecuación 24:

$$L(u_m|y) = ln\left(\frac{\sum_{(s',s)\Rightarrow u_m=+1}^{X} \alpha_{m-1}(s') \cdot \gamma_m(s',s) \cdot \beta_m(s)}{\sum_{(s',s)\Rightarrow u_m=-1}^{X} \alpha_{m-1}(s') \cdot \gamma_m(s',s) \cdot \beta_m(s)}\right)$$

$$= ln\left(\frac{\sum_{(s',s)\Rightarrow u_m=+1}^{X} \exp\left(A_{m-1}(s) + \Gamma_m(s',s) + B_m(s)\right)}{\sum_{(s',s)\Rightarrow u_m=-1}^{X} \exp\left(A_{m-1}(s) + \Gamma_m(s',s) + B_m(s)\right)}\right)$$

$$= \ln\left(\sum_{(s',s)\Rightarrow u_m=+1}^{X} \exp\left(A_{m-1}(s) + \Gamma_m(s',s) + B_m(s)\right)\right)$$
$$- \ln\left(\sum_{(s',s)\Rightarrow u_m=-1}^{X} \exp\left(A_{m-1}(s) + \Gamma_m(s',s) + B_m(s)\right)\right) \qquad (24)$$

Considerando *X* como el número de estados de cada etapa del diagrama de Trellis. Así es como recursivamente se pueden calcular los valores de los $L(u_m|y)$ para posteriormente tomar la decisión sobre los resultados obtenidos.

3.6.3 Extensión a lenguajes no binarios.

Un campo finito o campo de Galois se puede definir en palabras simples (solamente para los alcances de este trabajo) como un conjunto finito de *k* elementos y que consta de dos operaciones: suma y multiplicación (+ y ·), con neutro aditivo y multiplicativo, además de contar con otros tipos de propiedades como la distributividad, etc. Es finito porque cualquier operación que se haga entre los elementos que componen el campo da como resultado un elemento perteneciente al campo. Un ejemplo de campo finito son los números binarios, que cuentan con dos elementos (0 y 1) y donde las operaciones entre ellos entrega un valor dentro del mismo campo. En la

Tabla 3.3 se resume la operatoria sobre números binarios:

Tabla 3.3. Operatoria sobre números binarios

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Entonces, el sistema algebraico ({0, 1}; + , ·) representa un campo finito o campo de Galois de dos elementos y se denota como *GF(2)*. Es posible construir campos con mayor número de elementos, que deben cumplir ciertos requisitos según la teoría de cuerpos finitos [8], los cuales pueden ser ocupados en sistemas de compresión y criptografía, ya que puede representar una cantidad mayor de bits mediante el uso de un símbolo. En este trabajo se harán pruebas con campos de orden cuatro o *GF(4)*, el cual consta de cuatro elementos (0, 1, 2 y 3). Es posible hacer un mapeo uno a uno de los elementos del *GF(4)* y los de *GF(2)* como se muestra en la Tabla 3.4:

Tabla 3.4. Mapeo de elementos de GF(4) en GF(2)

GF(4)	GF(2)
0	00
1	01
2	10
3	11

Como se ve en la Tabla 3.4, se pueden representar dos símbolos de *GF(2)* usando un solo símbolo de *GF(4)* por lo que el envío de señales por vía inalámbrica sería más eficiente. Para poder enviar información que pueda discriminar entre mayor número de símbolos se usan esquemas de modulación como se explicará en la sección 3.9, por lo que se usará de modo ilustrativo para esta memoria la modulación Q-PSK, que permita distinguir entre cuatro símbolos.

La operatoria de *GF(4)* se muestra en la Tabla 3.5:

Tabla 3.5. Operatoria sobre GF(4)

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Una vez definida la aritmética para la suma y la multiplicación para *GF(4)*, se explicará cómo cambian los diagramas de Trellis en lenguajes no binarios.

Los diagramas de Trellis en lenguajes no binarios dependen principalmente del modelo de canal que se tenga y de la cantidad de símbolos contenidos en el campo finito. Para el modelo de canal presentado en la sección 3.3, la máxima dispersión para la respuesta al impulso es $t_d = 2$ períodos y si se usan lenguajes binarios (modulación BPSK) se manejan dos símbolos (0 y 1), entonces $s_c = 2$ se tiene que el diagrama de Trellis cuenta con ${s_c}^{t_d} = 2^2 = 4$ estados, ya que el canal actúa como registros de memoria que debe guardar los dos estados anteriores para determinar el comportamiento del modelo. Otro aspecto importante en la extensión de log-MAP a lenguajes no binarios es que la definición de un indicador como el LLR se hace algo compleja ya que en la definición que se había planteado se podía definir una función centrada en cero (llamada LLR) y que su signo indicaba el posible valor de la probabilidad predominante entre dos posibles símbolos. Al tener más de dos símbolos, la definición de LLR debería llevarse a planos de más de dos dimensiones, lo que podría tornar más complejo el reconocimiento de una señal. En este trabajo se optó por no calcular el indicador LLR, sino que se trabajó directamente con las probabilidades a posteriori, de modo de determinar la decisión sobre esos términos. Además, para el envío de señales desde el ecualizador al decodificador o viceversa, también se puede trabajar directamente sobre los términos probabilísticos definidos en la ecuación (24) para cualquier símbolo de *GF(q)*, y hace más trasparente el traspaso de probabilidades a posteriori, sobre todo cuando se hace combinación de distintos tipos de códigos, como códigos convolucionales con códigos de bloques.

3.7 Códigos LDPC

Los códigos de paridad de baja densidad o LDPC (*Low density parity check*) fue presentado por primera vez por Gallager [2] durante sus estudios de Ph.D en MIT. Estos estudios no tuvieron mucha relevancia en esa época (1963), ya que se presentaba como un algoritmo complejo y con altos requerimientos de almacenaje. Durante los 90s hubo un resurgimiento de los códigos LDPC ya que el estado del arte permitía su implementación, su medición y con ello surgieron interesantes mejoras al algoritmo original. Este algoritmo ha evolucionado desde la idea original de Gallager, pasando por la representación gráfica del algoritmo, el importante reporte de su desempeño (cercano al límite de Shannon), aplicación en lenguajes no binarios y algunas simplificaciones de su algoritmo para reducir la complejidad de cálculo. El propósito de esta sección es explicar brevemente el funcionamiento del algoritmo de códigos LDPC para centrarse principalmente en su aplicación en lenguajes no binarios.

Los códigos LDPC son llamados así porque se basan en una matriz H de baja densidad, lo que significa que tiene pocos elementos distintos de cero y muchos ceros (en binario sería muchos "0" y pocos "1"). Los códigos LDPC que tienen un número fijo de "1" en las filas y columnas se llaman regulares y los que no, se llaman irregulares.

Este código es de tipo lineal de bloques, denotado por $C_b(n,k)$, y se especifica por una matriz generadora:

$$G = [P|I_k] \tag{25}$$

De modo que:

$$c = m \cdot G \tag{26}$$

Donde c es el vector que contiene la palabra codificada y m constituye el mensaje original y P es la matriz de paridad de dimensión $kx(n-k)$ e I_k es la matriz identidad de kxk. Es posible pasar a la forma sistemática de la matriz de paridad H, que viene dada por:

$$H = [I_{n-k}P^T] \tag{27}$$

La matriz H se construye de modo que sea ortogonal con G, de modo que:

$$G \cdot H^T = 0 \xrightarrow{implica} c \cdot H^T = m \cdot G \cdot H^T = 0 \tag{28}$$

La construcción de la matriz H debe cumplir con las siguientes restricciones:

- El número de unos presentes en filas y columnas debe ser constante, para todas las filas y columnas respectivamente.

- La cantidad de traslapes de unos por fila y columna debe ser a lo más de uno. Esto evita la aparición de ciclos en el grafo bipartito. Que exista un ciclo significa que es posible trazar una línea empezando por un uno y terminar en ese mismo, pasando por otros nodos. La existencia de ciclos merma el funcionamiento de los LDPC, por lo que debe ser evitados.

- El número de unos por fila y columna deben ser pequeños con respecto al tamaño de la palabra codificada.

Se puede modificar el planteamiento a partir del proceso inverso, es decir partir de la matriz H y obtener la matriz en forma sistemática, con lo que se construye la matriz G, lo que

permite modificar el planteo de ecuaciones (cambiar por traspuestas). Entonces la palabra codificada quedaría:

$$c = G^T \cdot m \tag{29}$$

Este algoritmo se basa en la decodificación por síndrome *S*, definido como:

$$S = H \cdot c \tag{30}$$

Con lo que la condición de paridad (que la palabra codificada será válida) queda definido por la expresión (31):

$$H \cdot c = 0 \tag{31}$$

Entonces, con lo anterior se tendría que cumplir:

$$H \cdot G^T = 0$$

La tesis de Gallager dice que un código LDPC se define por $C_{LDPC}(n,s,v)$, con n longitud de palabras de bloque, de modo que *H* tiene *s* unos por columna y *v* unos por fila. Para entender mejor el algoritmo, Tanner en 1981 definió los grafos bipartitos, que generaliza la idea de condición de paridad. Éstos se explicarán en el capítulo siguiente.

3.7.1 Decodificación de códigos LDPC

Un mensaje codificado se calcula con la ecuación:

$$c = G^T \cdot m \tag{32}$$

Este mensaje es un vector que viaja por un canal que introduce ruido, por lo que en el extremo receptor se recibe un vector que es la señal codificada más un ruido *n*: $r = c + n$. Este vector se utiliza para calcular el síndrome:

$$S = H \cdot r = H(G^T \cdot m + n) = H \cdot n \tag{33}$$

Es así como el método de decodificación debe calcular un vector *d*, tal que se anule el síndrome, o sea un *d* tal que $H \cdot d = 0$, el algoritmo que lo calcula se llama: algoritmo ***suma-producto***. La matriz H define dos tipos de nodo: los nodos símbolo (filas de H, denotados por d_j) y los nodos paridad (columnas de H, denotados por h_j). Estos nodos se relacionan por medio de la matriz H conectándolos entre sí, simbolizado en H por un "1". Estas relaciones se utilizan para calcular la probabilidad a posteriori de cada símbolo tomando en cuenta: canal, señal recibida y condiciones de paridad. En la Figura 3.19 se muestra un grafo bipartito:

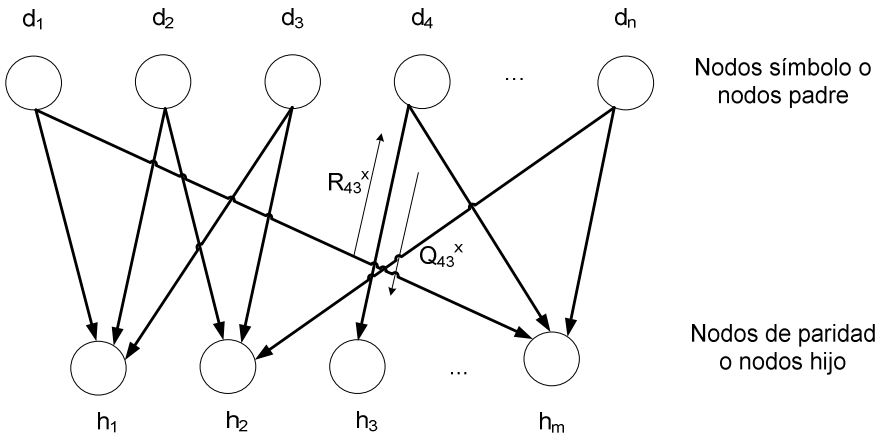


Figura 3.19. Grafo bipartito

En este algoritmo, cada nodo padre d_j, envía a su nodo hijo h_i unos índices Q_{ij}^X, que son información probabilística basada en lo que entregan los otros nodos hijo que se relacionan con el símbolo d_j. Esta información da cuenta del estado en que se encuentra ese símbolo. Adicionalmente, los nodos hijos (paridad) h_i mandan la información R_{ij}^X, a los nodos padre d_j, la que se basa en la información de los otros nodos padre ($\neq j$). El algoritmo se detiene cuando la palabra decodificada es válida, o sea cuando $H \cdot c = 0$.

Esa es principalmente la idea del algoritmo suma-producto, ahora se explicará el fundamento teórico.

Se comienza calculando los valores iniciales de Q_{ij}^X, de acuerdo a la probabilidad a priori de cada símbolo, denotada por f_j^X, es decir, la probabilidad de que el j-avo símbolo sea X. Si no se tiene información previa de la probabilidad a priori, (por ejemplo proveniente de otro decodificador o ecualizador) se estima esta probabilidad según el modelo de canal. Después se calculan los valores R_{ij}^X, correspondiente a la probabilidad que el nodo de paridad i se satisface, dado que el nodo padre j está en el estado X, o sea, que cumpla con la condición de síndrome, planteada en la ecuación (34):

$$P\left(h_i \middle| d_j = X\right) = \sum_{d:d_j=X} P(h_i|d)P(d|d_j = X) \tag{34}$$

La información que envía el nodo de paridad h_i al nodo símbolo d_j se calcula para el estado X como:

$$R_{ij}^X = \sum_{d:d_j=X} P(h_i|d) \cdot \prod_{k\epsilon N(i)\setminus j} Q_{ik}^{d_k} \tag{35}$$

Donde el primer término (sumatoria) se calcula sobre todos los valores del vector decodificado d, tal que se cumple que el nodo símbolo sea X, por la condición de paridad. El segundo término (pitatoria) se calcula sobre los $N(i)\backslash j$, o sea, los nodos padre conectados al nodo de paridad h_i, excluyendo al nodo j. El término $P(h_i|d)$ es 0 o 1 si es que se satisface o no el control de paridad. El término Q_{ij}^X es la probabilidad que el nodo símbolo esté en estado X, dada la información que envían los nodos de paridad, o sea, usando teorema de Bayes:

$$P\left(d_j = X \middle| \{h_i\}_{i \in M(j)\backslash i}\right) = \frac{P\left(d_j = X\right) P(\{h_i\}_{i \in M(j)\backslash i} | d_j = X)}{P(\{h_i\}_{i \in M(j)\backslash i})} \tag{36}$$

Éste se calcula para todos los nodos *M(j)\i*, o sea, para los nodos de paridad conectados al nodo símbolo d_j, excluyendo el nodo *i*.

Por lo tanto el término Q_{ij}^{X}, se calcula con la ecuación (37):

$$Q_{ij}^{X} = \alpha_{ij} f_j^{X} \prod_{k \in M(j)\backslash i} R_{kj}^{X} \tag{37}$$

Estas expresiones vienen dadas por los conceptos de redes bayesianas y probabilidades condicionales. El término α_{ij} es una constante de normalización. Entonces la decisión para el bit decodificado se toma sobre todas las probabilidades de Q_{ij}^{X} para cada símbolo y para todos los valores de *X*, de modo que se cumpla la condición de paridad.

3.8 LDPC no binario

Los códigos LDPC no binarios, o sea sobre *GF(q), q>2,* fueron inicialmente propuestos por Davey y Mackay en [5] y [6], logrando en su tiempo alcanzar mejor desempeño que los códigos LDPC binarios. En la construcción de códigos LDPC binario se insertan unos en la matriz H, en cambio, en LDPC no binario se insertan números en *GF(q)* distintos de cero, o sea valores entre *1,...,q-1*. Además, la matriz H debe seguir la misma condición para cada palabra válida codificada, es decir, $H \cdot c = 0$. Por otra parte, las operaciones de suma y multiplicación deben ser realizadas sobre el correspondiente campo finito *GF(q)*. Igualmente que en el caso binario, se puede representar por un gráfico de Tanner la relación entre los nodos símbolo y los nodos de paridad, pero se debe agregar un número llamado peso al camino que une los nodos (que es el número correspondiente en *GF(q)* que determina la unión entre los nodos símbolo con nodos de paridad). Además es posible encontrar una representación equivalente entre las matrices *G* y *H* en campos *GF(2)* y *GF(q)* donde se cumplen las mismas condiciones para estas matrices en el campo binario que para el campo de mayor tamaño, tal como se muestra en [8]. La representación de matrices binarias en campos no binarios representa una disminución en el número de ciclos, lo que significa un mejor desempeño del código. En la Figura 3.20, se muestra un ejemplo de disminución de ciclos al usar representaciones de matrices en *GF(2)* y su equivalente en *GF(4)*.

$$\begin{array}{c} \\ C_1 \\ C_2 \end{array} \begin{array}{c} \begin{array}{ccc} X_1 & X_2 & X_3 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \\ 3 & 0 & 2 \end{bmatrix} \end{array} \qquad \begin{array}{c} \begin{array}{cccccc} X_{11} & X_{12} & X_{21} & X_{22} & X_{31} & X_{32} \end{array} \\ \begin{array}{c} C_{11} \\ C_{12} \\ C_{21} \\ C_{22} \end{array} \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

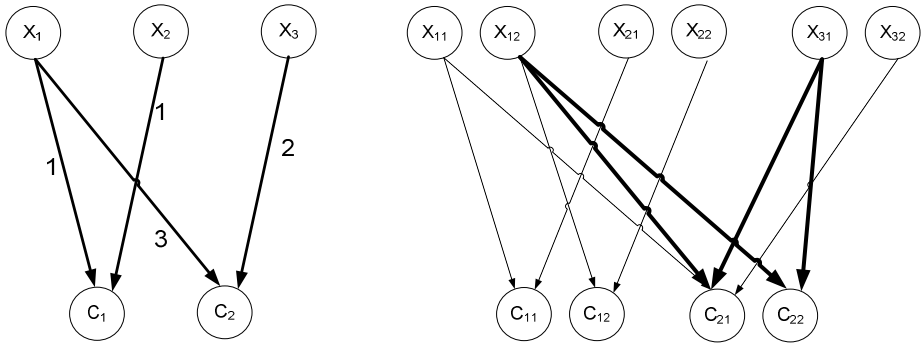


Figura 3.20. Disminución de ciclos usando GF(4)

En el ejemplo de la Figura 3.20, se muestra que el grafo para GF(4) no tiene ciclos, en cambio, el grafo equivalente para GF(2) tiene más de un ciclo, pero se marca uno para saber diferenciarlo. La eliminación de ciclos permite que las entradas distintas de cero puedan obtener información de sus nodos vecinos, lo que mejor el desempeño del algoritmo. El LDPC no binario también presenta algunas desventajas, como el incremento de estados en la matriz H, debido a mayor número de estados distintos de cero, lo cual hace más difícil para el decodificador tomar la decisión sobre el estado que debería tener cada símbolo. Además, con el número de estados, se incrementa de gran manera el nivel de complejidad de decodificación, ya que se debe hacer mayor cantidad de cálculos, además de definir operaciones sobre *GF(q)*.

3.8.1 Decodificación LDPC sobre *GF(q)*

El decodificador recibe una cantidad de p bits por cada símbolo a en *GF(q)*, entonces la probabilidad a priori está dada por:

$$f^a = \prod_{i=1}^{p} f_{x_i}^{a_i} \tag{38}$$

Siendo $f_{x_i}^{a_i}$ la probabilidad de que i-avo bit de q sea igual a a_i, perteneciente al campo finito.

El proceso de decodificación es muy similar al caso de LDPC binario, pero las operaciones matemáticas deben hacerse sobre *GF(q)*. Usando la ecuación (37) es posible calcular R_{ij}^X para *GF(q)*, de forma iterativa, pero a diferencia del caso binario, es necesario reordenar los valores de Q_{ij} antes de realizar el cálculo ya que esta operación se hace necesaria porque la palabra codificada tiene que ser multiplicada con la entrada de la matriz H, antes de la suma. Para realizar este reordenamiento hay que tener en cuenta los símbolos distintos de cero para cada fila de H, y reordenar los valores de Q_{ij} de acuerdo a la operación "inversa de multiplicación" sobre *GF(q)*(que normalmente se llama división pero en *GF(q)* a veces no se define como tal.

Igualmente se usará el símbolo divisor para identificar esta operación). Por ejemplo, si se tiene una matriz H de 5x10, revisando los valores de la primera fila se tiene que ésta vale: $[0, 1,2,3,0,0,0,3,0,0,]$, por lo tanto, los valores distintos de cero corresponden a los que están en las columnas 1, 2, 3 y 7 (contando desde cero). Esto significa que los valores de Q_{ij} para la primera fila deben ser reordenados según estos valores, por que lo se obtendrá una matriz de dimensión 4x4 (cuatro filas por cada elemento distinto de cero de la primera fila de la matriz H y cuatro columnas por cada elemento de *GF(4)*) y cada fila será ordenada según la operación "inversa de multiplicación" que indiquen los valores distintos de cero de la matriz H. Por lo tanto, el primer valor distinto de cero es 1, y se deben reordenar los valores según: $1 \div [0,1,2,3] = [0,1,2,3]$, se debe ordenar según: $2 \div [0,1,2,3] = [0,2,3,1]$, y lo mismo se debe hacer para las restantes entradas distintas de cero la fila de H. Este proceso se debe hacer también para todas las filas de la matriz H. Con esto, si la matriz H es de 5x10, se obtienen 5 matrices (como 5 nodos de chequeo) de dimensiones distintas, dadas por las condiciones de chequeo que impone H. Al hacer este reordenamiento se obtiene la PDF para el producto del nodo símbolo j-*ésimo*, en la i-*ésima* fila y su correspondiente número distinto de cero en la matriz $H_{ij'}$, o sea, $\left(\left(Q_{ij'} \cdot H_{ij'}\right)^0, \dots, \left(Q_{ij'} \cdot H_{ij'}\right)^{q-1}\right)$, correspondiente a la versión reordenada de $\left(Q_{ij'}^0, \dots, Q_{ij'}^{q-1}\right)$ tomando en cuenta la entrada de la matriz $H_{ij'}$.

Como la obtención de R_{ij}^X lleva consigo la operación de convolución, para este cálculo es necesario llevar previamente los valores de Q_{ij} al dominio de la frecuencia por lo que es preciso implementar una FFT sobre campos finitos. Para ello se usará la matriz de Walsh Hadamard (WHM), para campos de Galois de tamaño 2^p. Para *GF(2)* la FFT viene dada por la aproximación:

$$\begin{pmatrix} F^0 \\ F^1 \end{pmatrix} = \begin{pmatrix} f^0 \\ f^1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{39}$$

Donde la matriz de 2x2 puede ser usada para la construcción de WHM sobre *GF(q)*. Si se dobla el tamaño de la WHM, entonces ésta podrá usarse para *GF(4)*. Para construirla, se copia la matriz *GF(2)* en la partes superiores y en la parte inferior izquierda y en la parte inferior derecha se copia la matriz para *GF(2)* multiplicada por -1. Entonces la matriz WHM para *GF(4)* será la dada en (40):

$$\begin{pmatrix} F^0 \\ F^1 \\ F^2 \\ F^3 \end{pmatrix} = \begin{pmatrix} f^0 \\ f^1 \\ f^2 \\ f^3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{40}$$

De igual forma se puede tomar la matriz sobre *GF(4)* y realizar las mismas operaciones para obtener la WHM para *GF(8)*.

Una vez calculada la FFT de la PDF para el producto para el producto del nodo símbolo j-*ésimo*, en la i-*ésima* fila y su correspondiente número distinto de cero en la matriz $H_{ij'}$, se obtienen estos valores en el dominio de la frecuencia, denotados por $\left(\hat{Q}_{ij'}^0, \dots, \hat{Q}_{ij'}^{q-1}\right)$ y con ellos, es más fácil obtener la convolución en el dominio de la frecuencia, según la ecuación (41):

$$\left(\prod_{k \in C(i) \setminus j} \hat{Q}_{ik}^{0}, \prod_{k \in C(i) \setminus j} \hat{Q}_{ik}^{1}, \ldots, \prod_{k \in C(i) \setminus j} \hat{Q}_{ik}^{q-1} \right) \tag{41}$$

Esto entrega la convolución de todas las cantidades $Q_{ij'} \cdot H_{ij'}$participando en la i-*ésima* fila, con excepción del nodo símbolo *j*. Estos vectores se denotarán como $\left(\hat{R}_{ij}^{0}, \ldots, \hat{R}_{ij}^{q-1}\right)$, siendo esta la FFT de la secuencia $\left(\left(R_{ij} \cdot H_{ij}\right)^{0}, \ldots, \left(R_{ij} \cdot H_{ij}\right)^{q-1}\right)$, entonces se debe calcular la IFFT de los $\left(\hat{R}_{ij}^{0}, \ldots, \hat{R}_{ij}^{q-1}\right)$ para obtener los $\left(\left(R_{ij} \cdot H_{ij}\right)^{0}, \ldots, \left(R_{ij} \cdot H_{ij}\right)^{q-1}\right)$, llevando los vectores al dominio del tiempo, pero donde está incluida la contribución de los índices de la matriz H_{ij} por lo que se debe invertir el reordenamiento (o sea, reordenar esta vez según la multiplicación de los números distintos de cero en la fila de la matriz H) para volver al estado inicial para obtener finalmente los vectores $\left(R_{ij'}^{0}, \ldots, R_{ij'}^{q-1}\right)$. Con ello, es posible obtener los valores actualizadores de los Q_{ij}, se normalizan para finalmente calcular las probabilidades a posteriori por cada símbolo usando la ecuación (42):

$$P_n^a = \propto_n f_n^a \prod_{k \in M(n)} R_{k,n}^a \tag{42}$$

Siendo *M(n)* un set de índices de las columnas de las entradas distintas de cero en la i-ésima columna de H y siendo a el símbolo correspondiente dentro del alfabeto.

Esta información puede usarse para tomar la decisión sobre qué símbolo tiene la mayor probabilidad de ser efectivamente, dada una secuencia recibida o también puede usarse como información a priori, en un esquema turbo para posteriormente realizar una mayor cantidad de iteraciones.

3.9 Interleaver-Deinterleaver

El interleaver tiene como entrada cierto bloque de datos, de un largo determinado y permuta los bits de ese bloque de forma pseudo-aleatoria para principalmente entregarle aleatoriedad a las señales que entran a este bloque. Esto tiene una serie de ventajas que permiten incrementar el desempeño del esquema presentado. Las principales son: evitar pérdidas importante de información cuando ocurre que una ráfaga de datos sean corrompidos, es decir, cuando varios bits seguidos se pierden. Así no se pierde la correlación existente entre los datos en un tiempo actual y el anterior. Existen varios tipos de interleavers en la literatura, cada una con sus ventajas y desventajas. Se explicarán algunos para entender el funcionamiento de este bloque presente en la Figura 3.9:

3.9.1 Interleaver de Bloques

Este interleaver es ampliamente usado en sistemas de comunicaciones. Como su nombre lo indica, éste funciona tomando bloques de datos (como forma de matriz) y lee las filas de izquierda a derecha y las escribe de arriba hacia abajo, tal y como se muestra en la Figura 3.21:

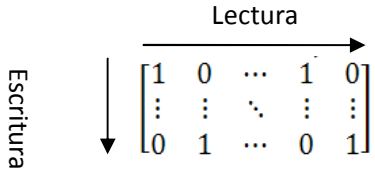


Figura 3.21. Interleaver de bloques

3.9.2 Interleaver pseudo-aleatorio

El interleaver pseudo-aleatorio usa una permutación aleatoria fija y mapea la secuencia de bits que entran al interleaver de acuerdo al orden preestablecido de permutación. Esta permutación se hace para una secuencia fija de bits, tal y como se muestra en la Figura 3.22:

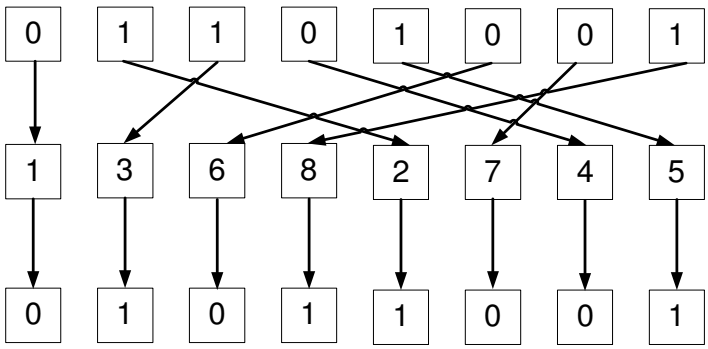


Figura 3.22. Interleaver pseudo-aleatorio

Existen otros tipos de interleavers, algunos que utilizan métodos muy elaborados para conseguir mejor desempeño como por ejemplo los interleavers de permutación circular, los semialeatorios, diseñados usando topología par e impar o el interleaver óptimo (o más bien cercano al óptimo), pero como este tema puede ser extendido ampliamente, se remitirá bajo el contexto de este trabajo de título el uso de un interleaver pseudo-aleatorio debido a la forma en que se hará el tratamiento de las secuencias de bits.

El deinterleaver se encarga de invertir el proceso de permutación del interleaver, es decir, si un bloque de datos para primero por un interleaver y después pasa por un deinterleaver, entonces se obtiene la secuencia de datos sin cambios. Esta definición se aplica para todos los tipos de interleavers.

3.10 Modulación PSK

El término modulación se refiere al hecho de imprimir información a una onda portadora. La modulación por desplazamiento de fase o modulación PSK (*phase-shift keying* en inglés) actualmente encuentra un uso amplio en sistemas de telecomunicaciones militares y comerciales, como comunicaciones satelitales y terrestres. PSK es considerado eficiente para la modulación de datos porque entrega la más baja probabilidad de error para un nivel de señal dado(comparado con QAM, por ejemplo), midiendo sobre el período de un símbolo. El funcionamiento de PSK se basa en tomar una señal de entrada y se desplaza la fase de la salida a un número fijo de estados. La señal puede ser escrita como:

$$v_0(t) = \sqrt{2A}\sin\left(w_0 \cdot t + \frac{2\pi(i-1)}{M}\right)$$

$$i = 1, 2, \dots, M \tag{43}$$

$$t \in [\frac{-T_s}{2}, \frac{T_s}{2}]$$

Donde A es la potencia promedio de la señal sobre un intervalo T_s y M es el número de fases permitidos o estados. Es importante señalar que el número de estados M simboliza la cantidad de estados que puede presentar la señal a medida que se cambia su fase. Las versiones comunes de PSK es el BPSK (*binary* PSK, M=2 y los estados se representan por 0 y 1), QPSK (*quadrature* PSK, M=4 y los estados se representan por 0, 1, 2, 3) y en general se pueden establecer modulaciones que tengan una cantidad de 2^N estados. Además, una mayor cantidad de estados representa un mejor uso espectral ya que se envía una mayor cantidad de bits por cada símbolo (BPSK necesita 1 bit para representar cada estado, en cambio QPSK necesita 2 bits), pero en los esquemas de ecualización turbo representan una mayor complejidad en el tratamiento de señales, sobretodo por el aumento de estados en el trellis y con ello una mayor cantidad de cálculos en cada iteración. Es por ello que no es conveniente utilizar una modulación con una cantidad de estados muy grande, además del hecho que una mayor cantidad de símbolos en la constelación representa mayor susceptibilidad a ruido. Por otra parte, la cantidad de bits usados en cada esquema involucra operatoria en campos de Galois de orden q (o *GF(q)*) cuando se utiliza en algoritmos como el LDPC no binario, lo que puede complicar aun más su implementación. La operatoria sobre *GF(q)* y el algoritmo LDPC no binario se presenta en la sección 3.7.1.

3.11 Consideraciones para ecualización turbo

Volviendo al esquema de la Figura 3.9, propuesto por Douillard, se puede decir que la ecualización turbo se basa principalmente del esquema de decodificación turbo, consistente en usar dos decodificadores SISO, que intercambian información de forma iterativa, donde los decodificadores SISO pueden estar implementados con algún tipo de algoritmo afín. En la ecualización turbo, se cambia un decodificador SISO (de los dos que hay) por un ecualizador SISO, dejando un decodificador SISO y estos dos bloques son los que trabajan de forma iterativa. Antes de seguir en mayor profundidad, es necesario definir algunos términos que se usarán con frecuencia. Estos términos son: información a priori, información extrínseca e información a posteriori:

A priori: Es la información asociada a un bit v_m, que es conocida antes de empezar el proceso de ecualización o decodificación, recibida desde una fuente distinta a la secuencia recibida. En la literatura también se le suele llamar información intrínseca, comparándola con la información extrínseca.

Extrínseca: Es la información asociada a un bit v_m entregada por el ecualizador o decodificador, basados en la secuencia recibida y la información a priori de todos los bits, con excepción de la información recibida y a priori relacionada con el bit v_m. La información extrínseca existe por varios factores. Al usar un codificador que tenga memoria, cada bit influye en la salida del codificador para los demás bits, ya que en teoría, el codificador convolucional, tiene respuesta al impulso infinita. En la práctica, la respuesta es acortada, pero igualmente es prolongada. Para códigos turbo, la respuesta al impulso igualmente se hace prolongada, pero para códigos que no presenten memoria. La presencia de este fenómeno justifica el uso de interleavers de gran profundidad, pero el hecho de que una señal influye en la codificación de las demás, puede entregar una herramienta poderosa para un decodificador o ecualizador turbo, ya que iterativamente, éste puede usar información de los otros bits para saber si la decisión tomada es la correcta o no. Para ello, la información intrínseca y extrínseca debe ser tratada de forma separada y no correlacionada.

Otra fuente de información extrínseca, debido a que expresa memoria, es la memoria del canal que introduce dispersión en el tiempo, el que anteriormente se llamó ISI o también, con un esquema conocido se puede llamar CISI (*Controlled* ISI o interferencia intersímbolo controlada) que presentan algunos moduladores como GMSK.

A posteriori: Es la información entregada por el algoritmo SISO, relacionada a un bit, tomando en cuenta todas las fuentes de información disponibles acerca de ese bit v_m.

La importancia de la información a priori, extrínseca y a posteriori tienen vital importancia para el buen funcionamiento de esquemas turbo que exhiben memoria, ya que el ecualizador SISO trabaja recibiendo señales corruptas e información a priori del decodificador SISO y a partir de eso genera la información a posteriori. Esta información a posteriori debería ser la información a priori que usará el decodificador, pero primero debe removerse la información acumulada de la iteración anterior para entregar información extrínseca combinada con la de canal. Remover la información a priori es necesaria para prevenir un fenómeno llamado "retroalimentación positiva" ya que el bloque estaría recibiendo información de sí misma, lo que podría afectar su estabilidad. En la Figura 3.23 se muestra el esquema de un ecualizador turbo, tomando las consideraciones anteriormente expuestas:

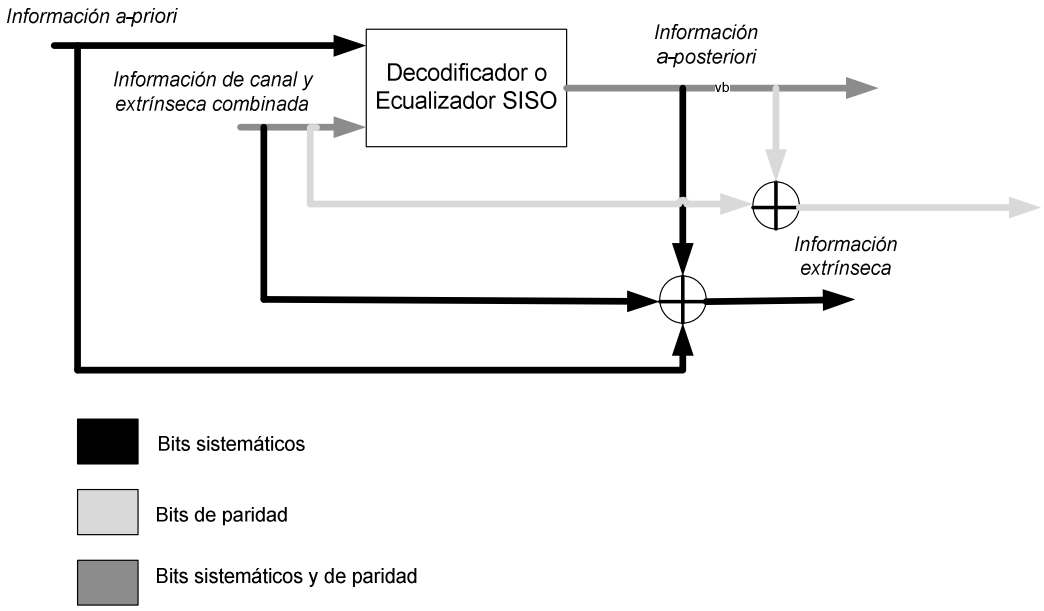


Figura 3.23. Esquema ecualización turbo, tomando en cuenta información a priori, a posteriori.

Posteriormente la información extrínseca pasa por un deinterleaver para llegar al decodificador y consecuentemente éste calcula sus valores a posteriori, los cuales pasan por un interleaver y se enviadas al ecualizador SISO para completar una iteración. Esto se debe hacer tanto para los bits correspondientes a información de canal y paridad. Antes de enviar la información al interleaver se debe remover la información de canal y extrínseca combinada de la información entregada por el decodificador. Esto se hace restando los LLR sistemáticos de canal y extrínsecos, y los LLR a priori con los bits sistemáticos de los LLR a posteriori. Esto entrega sólo los LLR de los bits sistemáticos. Para obtener los bits de paridad se debe restar los LLR de la información de canal y extrínseca combinada de los bits de paridad con los LLR a posteriori de los bits de paridad. Toda esta información (bits sistemáticos y de paridad) es utilizada para tomar la decisión final o bien, pueden enviarse de vuelta al ecualizador y con ello realizar otra iteración. Este mismo esquema presentado en la Figura 3.23, puede ser incrementado usando mayor cantidad de codificadores y decodificadores, como en esquemas de codificación turbo, lo que a su vez conlleva a una mayor complejidad en la computación de cálculos e implementación. En el contexto de esta memoria se usará un codificador y un decodificador, de tal modo de centrarse en la idea del esquema original, pero con algunas modificaciones. Se propone para trabajos posteriores trabajar con esquemas de mayor complejidad.

Capítulo 4

Descripción del modelo propuesto y simulaciones asociadas

4.1 Esquema general

Una vez explicados los algoritmos de la sección anterior, la ilustración del modelo presentado en esta memoria de título se hace una tarea de menor complejidad, dado que éstos se basan en los algoritmos anteriores. El modelo presentado en este tema y que titula esta memoria, se trata de la concatenación de dos códigos anteriormente presentados: el código log-MAP en su versión no binaria y el código LDPC no binario, los cuales se implementarán bajo *GF(4)*, para introducir el modelo. Se propone para trabajos posteriores la implementación sobre campos más extensos para extraer resultados. El algoritmo log-MAP no binario será utilizado para ecualizar el canal y reducir los efectos producidos por el ISI y el LDPC no binario será utilizado para reducir los efectos del canal AWGN. Entonces el proceso de codificación se hará usando el algoritmo LDPC no binario a través de la matriz H no binaria, tal como se explicó en la sección 2.6. En resumidas cuentas, el lado del emisor, se compondrá del siguiente esquema:

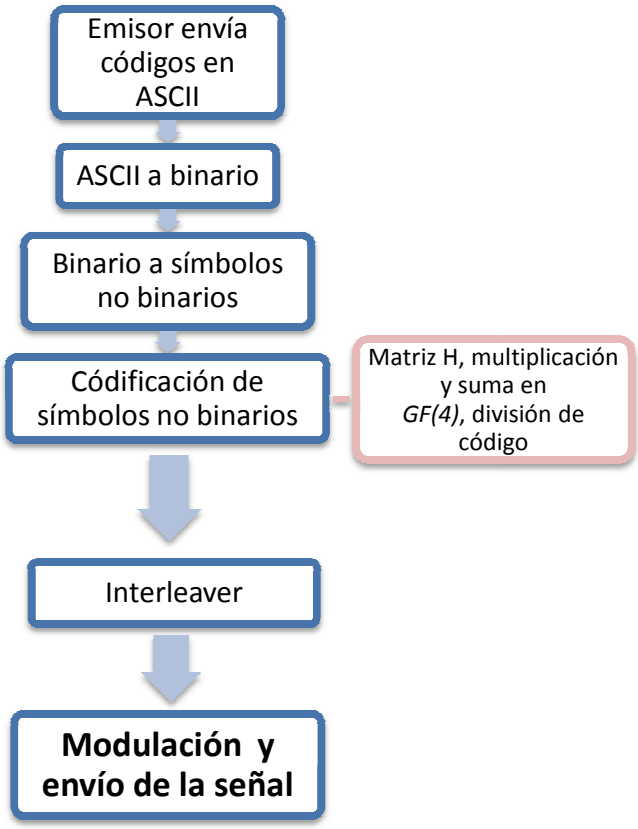


Figura 4.1. Esquema general del proyecto

El método de codificación se explicó en la sección 2.6, donde se puede resumir diciendo que se debe generar una matriz H, según la cantidad de nodos de chequeo y nodos de símbolo, la cual multiplicada al vector de símbolos no binarios, para obtener una secuencia de mayor tamaño que incluye los bits de redundancia. Esta secuencia codificada pasa por un canal que introduce ISI, junto con adherir ruido blanco Gaussiano. Este modelo de canal estático es muy usado en la actualidad para validar modelos, en que, si es cierto que los canales reales no son efectivamente estáticos, los resultados que entrega este canal se asemejan a modelos más reales que existen en la actualidad, además que su implementación se hace menos compleja.

Entonces, el diagrama de Trellis que entrega el modelo anterior se muestra en la Figura 4.2:

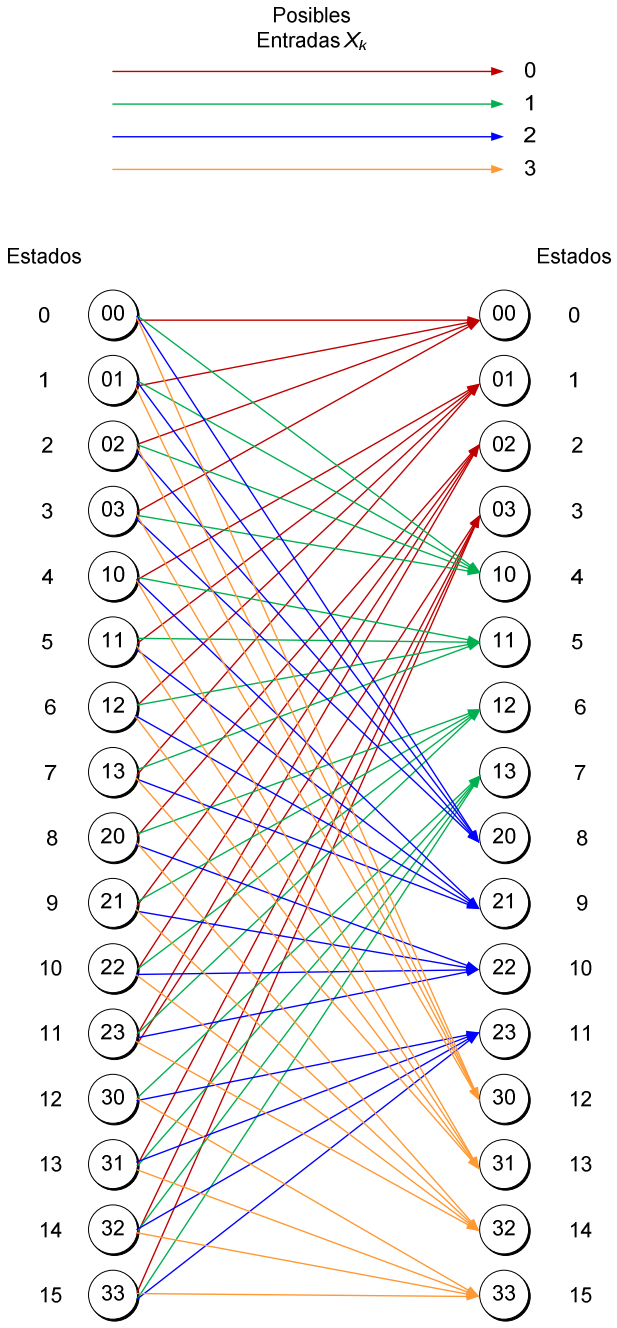


Figura 4.2. Diagrama de Trellis utilizado en GF(4)

Este diagrama de Trellis será utilizado para realizar el proceso de ecualización. Se han omitido las salidas esperadas por cada camino del diagrama de Trellis para no complicar la notación. La cantidad de estados del diagrama de Trellis está directamente relacionada con el modelo del canal y con la cantidad de elementos en el campo que se está trabajando. El canal tiene dos derivaciones y se está trabajando sobre *GF(4)*, por lo que la cantidad de estados viene dada por $4^2 = 16$, denotados por números del 0 al 15. Los estados se denotan por un buffer de

dos símbolos, que representan todas las combinaciones posibles en de los 4 bits y cada línea representa la llegada de un símbolo.

Una vez que la señal codificada pasa por el canal de la Figura 3.11, se comienza el proceso de ecualización. Al comienzo de este proceso se recibe la señal afectada por el canal y se introduce ISI y ruido AWGN, entonces, el esquema general que usa el modelo se muestra en la Figura 4.3:

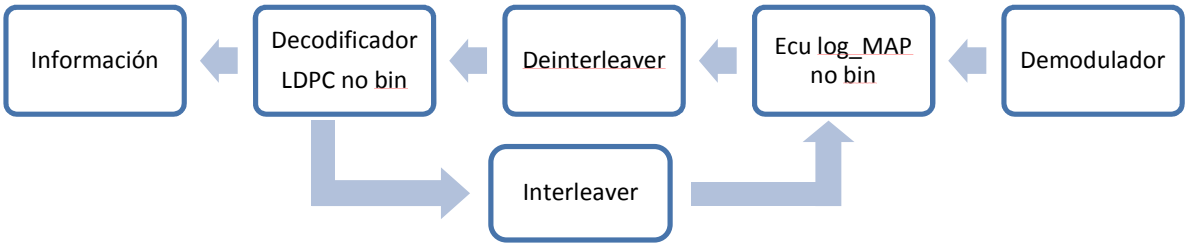


Figura 4.3. Esquema general al lado del receptor

Este esquema general puede ser de ayuda para comprender el funcionamiento del sistema como un todo, pero para entender a fondo cómo funciona esto, hay que empezar a descomponer cada bloque. El demodulador por se encarga de hacer un muestreo de la señal y mapearla según el esquema que se utilizará para la ecualización y la decodificación. Para esto se usarán señales con valores en los reales, donde el valor 0 indica que se trata de una señal de valor cero. De igual forma se tratarán los otros valores del campo. Entonces la cuantización será según el Figura 4.4:

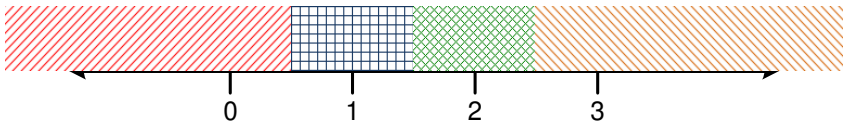


Figura 4.4. Nivel de cuantización

4.2 Modificación al modelo matemático

Supongamos que se recibe un tren de N símbolos consecutivos, por lo que el ecualizador debe eliminar el ISI de cada uno de los símbolos que dependen de las entradas anteriores y futuras. Entonces, se procede a calcular los valores de probabilidad a posteriori de la ocurrencia de un símbolo $u_m = a$, dada una secuencia recibida y, como se define en la ecuación (43) (de igual manera que para campos binarios):

$$P(u_m = a|y) = \sum_{(s',s)\Rightarrow u_m=a} \frac{P(u_m = a, y)}{P(y)} \tag{43}$$

Esta última igualdad definida por teorema de Bayes.

Entonces en la publicación de L. R. Bahl et al [19], se puede usar su razonamiento del cálculo de las probabilidades a posteriori para extrapolarlo a lenguajes no binarios. La sumatoria de la ecuación (43) significa que engloba a todos los términos que involucran un camino que va desde s' a s donde la entrada del bit corresponde a $u_m = a$. Entonces, el razonamiento se desarrolla como sigue:

$$\sum_{(s',s)\Rightarrow u_m=a} \frac{P(u_m = a, y)}{P(y)} = \sum_{(s',s)\Rightarrow u_m=a} \frac{P(y_{j<m}, s') \cdot P(y_m, s|s') \cdot P(y_{j>m}|s)}{P(y)} \tag{44}$$

Es posible usar otra vez el teorema de Bayes para obtener la relación de la ecuación (45):

$$\begin{aligned} P(y_m, s|s') &= P(y_m|s', s) \cdot P(s', s) \\ &= P(y_m|s', s) \cdot P(u_m) \end{aligned} \tag{45}$$

Esto se puede utilizar para calcular las probabilidades a posteriori usando el algoritmo log-MAP como sigue:

$$P(u_m = a|y) = \sum_{(s',s)\Rightarrow u_m=a} \frac{P(y_{j<m}, s') \cdot P(y_{j>m}|s) \cdot P(y_m|s', s) \cdot P(s', s)}{P(y)} \tag{46}$$

Usando el hecho que se usa un canal Gaussiano, se obtiene:

$$\begin{aligned} &\sum_{(s',s)\Rightarrow u_m=a} \frac{P(y_{j<m}, s') \cdot P(y_{j>m}|s) \cdot P(y_m|s', s) \cdot P(s', s)}{P(y)} \\ &= \sum_{(s',s)\Rightarrow u_m=a} \frac{\alpha_{m-1}(s') \cdot \beta_m(s)}{P(y)} \cdot \underbrace{\left[\frac{1}{\sqrt{2\pi\sigma}} exp\left[\frac{-1}{2\sigma^2}(y_m - \hat{y}_m)^2\right]\right]}_{\gamma_m(s',s)} \cdot P(u_m) \end{aligned} \tag{47}$$

Entonces, tomando el logaritmo a este término (ya que se trata del algoritmo log-MAP) y usando la definición para este algoritmo, entonces se obtiene:

$$\ln\big(P(u_m = a|y)\big) = \ln\left(\sum_{(s',s)\Rightarrow u_m=a} \exp\left(A_{m-1}(s') + \Gamma_m(s', s) + B_m(s)\right)\right) \tag{48}$$

Donde se define el término $\Gamma_m(s', s)$ como:

$$\Gamma_m(s', s) = -\frac{1}{2\sigma^2}(y_m - \hat{y}_m)^2 + \ln\big(P(u_m)\big) \tag{49}$$

Y los términos ln $(P(y))$ y ln $\left(\frac{1}{\sqrt{2\pi\sigma}}\right)$ han sido obviados por ser constantes para todos los símbolos.

El término $\ln\big(P(u_m)\big)$ se refiere a la información a priori que se tiene sobre si un símbolo corresponde según la secuencia recibida o no. En el esquema presentado, esta información corresponde a la que se envía a partir de la segunda iteración por parte del decodificador LDPC, donde se explicará más adelante la forma en que éste envía esa información. Cabe recalcar que como se dijo anteriormente, no es posible usar un indicador como el LLR del esquema binario ya que la mayor cantidad de símbolos en el alfabeto presenta una dificultad al representarlos y con ello es más fácil trabajar directamente con las probabilidades.

Con esto, se obtienen los cálculos de probabilidades a posteriori para cada uno de los N símbolos y para cada uno de las cuatro posibilidades de símbolos presentes en el alfabeto, en este caso se obtendrán Nx4 probabilidades a posteriori.

4.3 Modelo específico

Es más fácil visualizar este razonamiento haciendo un esquema de cómo funciona el algoritmo en el extremo receptor, según la Figura 4.5:

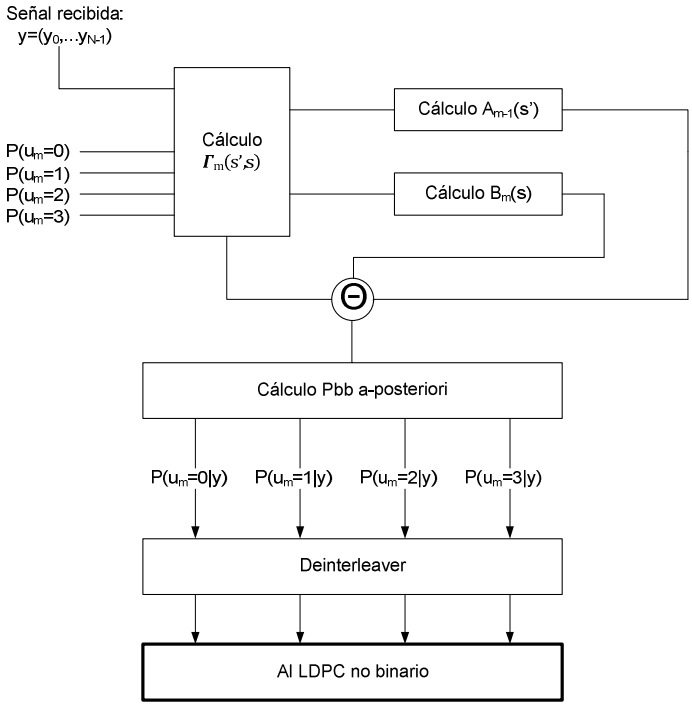


Figura 4.5. Esquema de funcionamiento algoritmo log-MAP propuesto

Entonces, se puede usar la información a posteriori del bloque que se encarga de ecualizar la señal como información a priori del decodificador LDPC. Para esto es necesario usar un deinterleaver entre cada bloque y con ello se puede iniciar el proceso de decodificación. Además el operador Θ del esquema corresponde a la operación realizada en la ecuación (48).

Para iniciar el algoritmo LDPC no binario se usará el mismo razonamiento utilizado en la sección 2.6, por lo que ahora solamente se esquematizará lo visto anteriormente y se adecuará al modelo presentado. El decodificador trabaja según el esquema mostrado en la Figura 4.6:

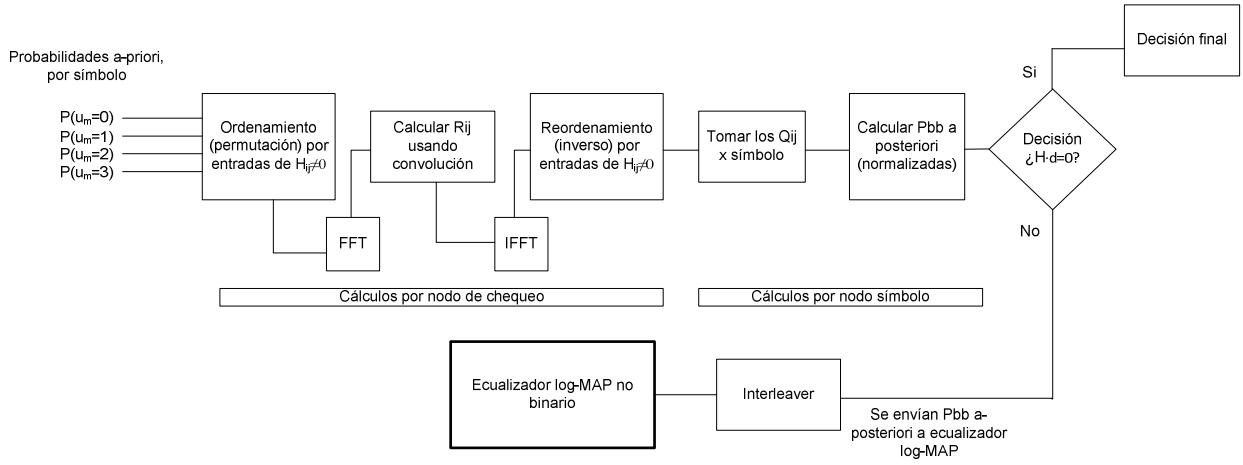


Figura 4.6. Esquema de funcionamiento algoritmo LDPC no binario propuesto

Los bloques componentes de este esquema ya fueron explicados anteriormente en el Capítulo 2. Lo único que falta por explicar es la realización de las próximas iteraciones usando la información del decodificador LDPC no binario. Las probabilidades a posteriori calculadas para cada símbolo permiten tomar la decisión sobré que bit debió haberse recibido dada una secuencia de arribo. Si bien, al no cumplir el criterio de la matriz de chequeo se puede decir que el bit decodificado efectivamente está erróneo, pero el receptor, al no tener información sobre qué bits realmente fueron enviados, entonces no se tiene certeza sobre qué bits están erróneos y cuáles no. Esta información a posteriori entrega información sobre las probabilidades de cada símbolo dentro de la secuencia y aunque la decisión entregue un vector decodificado erróneo, se puede corregir en cada iteración el valor de las probabilidades, lo que después de un número determinado de iteraciones, puede entregar realmente un vector correcto. Así es como se han logrado concatenar dos algoritmos que por sí solos han entregado excelentes resultados en telecomunicaciones y que trabajando en forma conjunta podrían incrementar ese potencial.

4.4 Implementación del modelo

El modelo planteado en la Sección 4.3 fue implementado usando el lenguaje Java, por medio del modelo cliente- servidor. El servidor trabaja de la siguiente manera:

Éste es capaz de mantenerse escuchando a hasta que uno o más clientes realicen una petición de conexión. Al realizarse esta petición, el servidor le envía una respuesta para establecer un diálogo. El servidor es quién envía un flujo o stream de datos al cliente, donde se da la opción de generar estos datos de forma aleatoria o que se puede especificar qué datos envía el servidor al cliente.

En el simulador desarrollado en este trabajo, los eventos mencionados anteriormente ocurren de la siguiente manera:

Cuando el servidor (corriendo bajo entorno de JGrasp) se encuentra esperando a algún cliente se visualiza el siguiente cuadro mostrado en la Figura 4.7:

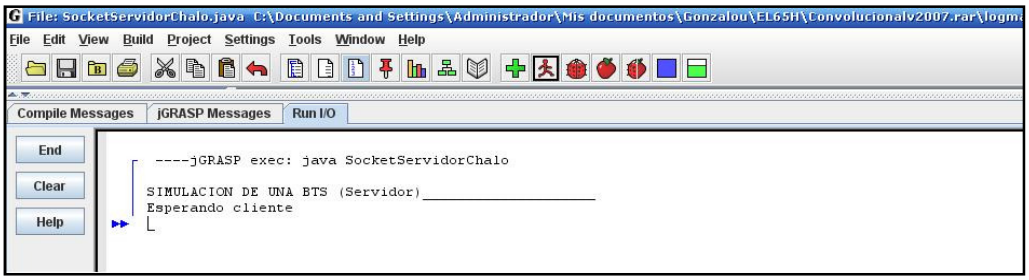


Figura 4.7. Servidor esperando

El cliente a su vez (corriendo en Netbeans), al momento de iniciar el programa intenta comunicarse con el servidor. Si la petición de comunicación se realiza correctamente, entonces se desplegará lo siguiente:

Figura 4.8. Cliente conectado

El servidor, al momento de recibir la petición del cliente, lanza un mensaje que verifica el éxito de la operación y a su vez entrega la posibilidad de introducir una palabra que sería el mensaje a transmitir al cliente, como se muestra en la Figura 4.9:

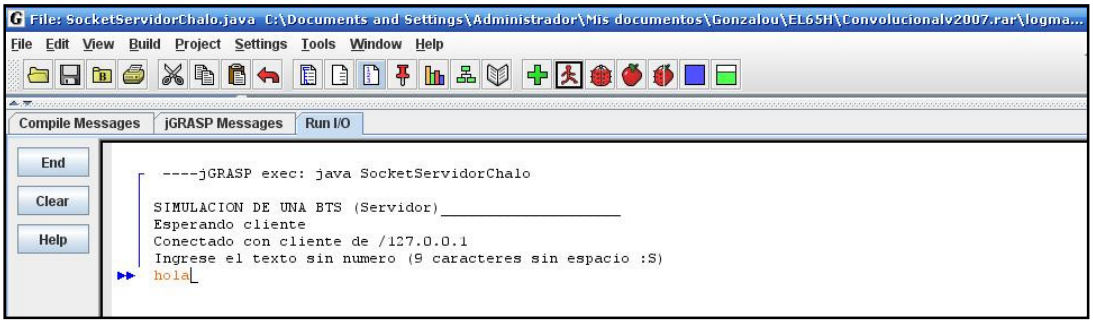


Figura 4.9. Servidor conectado y despliegue mensaje

A modo de ejemplo se ha introducido la palabra “hola”, la que dentro del servidor, se pasará a código ASCII para posteriormente traducirla a lenguaje binario y seguidamente a no binario. Esta palabra en lenguaje no binario dividirá en bloques que serán codificados mediante LDPC no binario, lo que finalmente serán enviados al cliente. Estas operaciones se pueden observar en la información que entrega el servidor, mostrada en la Figura 4.10:

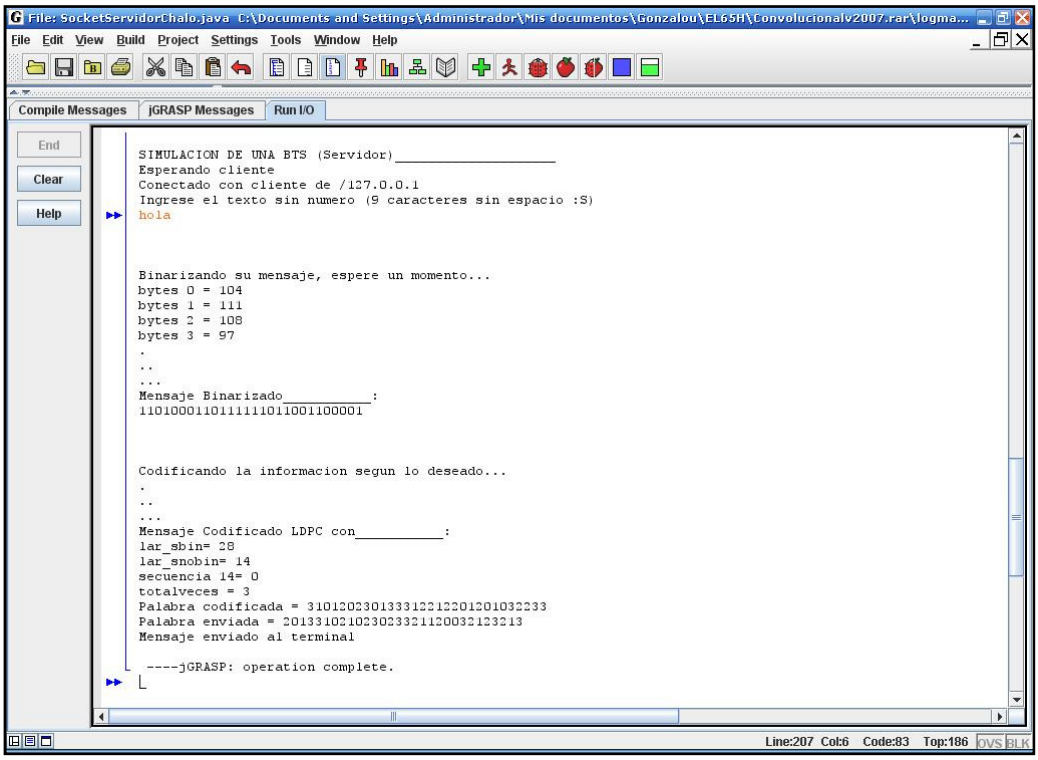


Figura 4.10. Sumario de operaciones del servidor

El mensaje enviado por el servidor llega al cliente, el cual despliega la información que le ha llegado, como se muestra en la Figura 4.11:

Figura 4.11. Mensaje recibido

Dentro del cliente se simula los efectos producidos por el canal, el cual introduce ISI y AWGN. A su vez, es posible modificar el SNR sobre el cual se establece la comunicación dentro del mismo programa. En la Figura 4.11, se puede ver que en la esquina inferior derecha de la pantalla del teléfono se visualiza un botón rotulado como “Decodificar”, al presionar ese botón la pantalla cambia y despliega el mensaje reconstruido, el cual ha pasado por el ecualizador LDPC no binario y el LDPC no binario, donde se produce la retroalimentación del par ecualizador-decodificador siguiendo el esquema turbo, donde también es posible modificar el número de iteraciones. Al presionar el botón se despliega la siguiente pantalla:

Figura 4.12. Mensaje decodificado

Esta es la pantalla con la cual finaliza el programa, donde en la esquina inferior derecha se puede salir, pulsando justamente el botón “Salir”. Es importante señalar que, la apariencia de celular del receptor no es sólo algo estético, sino que equivale a decir que el programa que se encarga de reconstruir la señal en el extremo receptor está programado usando J2ME Wireless Toolkit 2.2, que es el entorno que se utiliza en los celulares actuales comunes, es decir, el programa descrito se podría traspasar a un celular real para realizar pruebas. La Figura 4.13 resume el funcionamiento de este conjunto de programas:

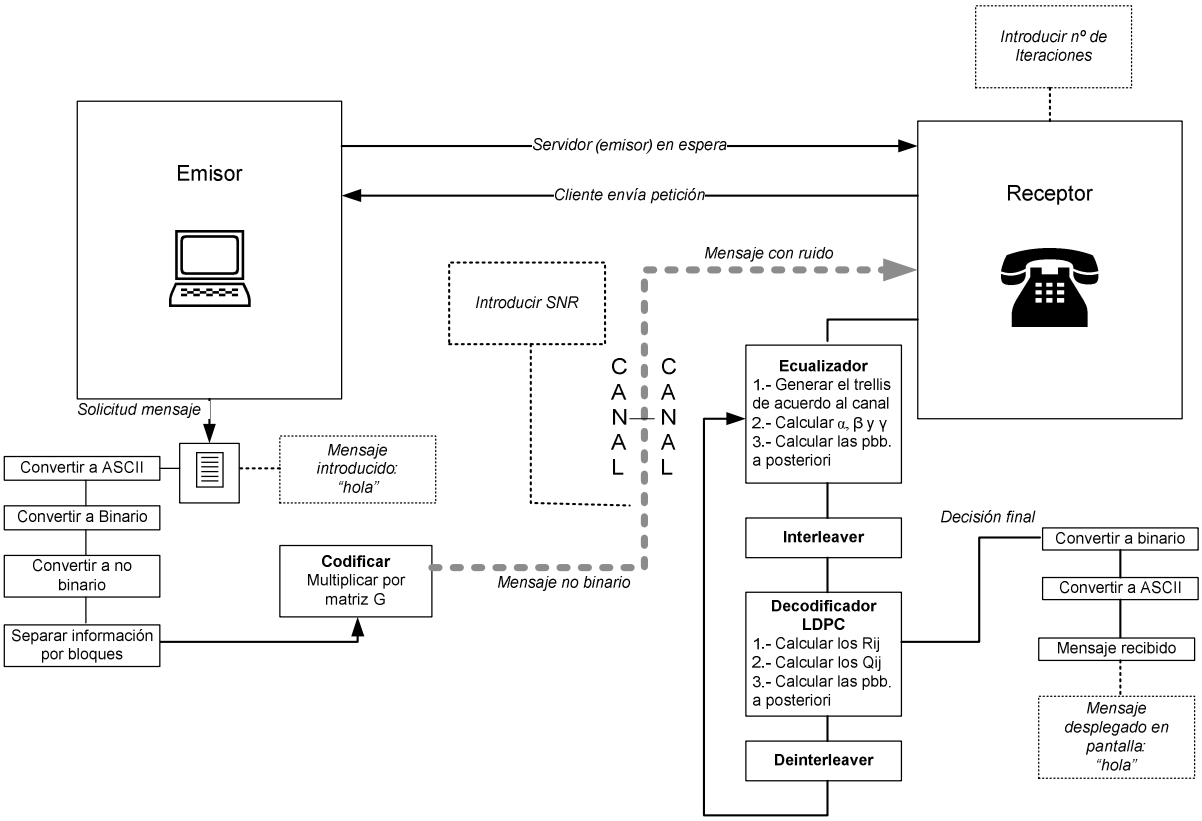


Figura 4.13. Esquema funcionamiento simulador

Para medir el funcionamiento del sistema se generaron tramas de datos aleatorias de bloques de tamaño 10^n, con n=1,…,6 y a partir de ellos se observó la tasa de símbolos erróneos encontrados en el receptor.

4.5 Explicación "paso a paso" del algoritmo propuesto y ejemplos

El objetivo principal de esta sección es entregar una forma estructurada del trabajo de manera que el lector pueda reconstruir fácilmente los cálculos y curvas expuestas en las secciones 4.6 y 4.7 a modo de permitir la construcción de algoritmos de mayor tamaño que ayuden a profundizar este estudio. Para cumplir esto, se ha dividido el algoritmo en distintos pasos a seguir, acompañado por ejemplos que complementen el aprendizaje, además, esta sección sirve de guía para entender mejor el esquema general de las Figura 4.5 y Figura 4.6.

Los pasos a seguir en este algoritmo son los siguientes.

a) Codificación y envío

- Paso 1: Encontrar una matriz H válida.
- Paso 2: Obtener matriz G.

- Paso 3: Ya con la matriz G calculada, comienza el proceso de codificación.
- Paso 4: Aplicar interleaver y enviar palabra codificada.
- Paso 5: Paso del mensaje por el canal

b) Estimación del canal

- Paso 6: Trazado del diagrama de Trellis.

c) Algoritmo log-MAP no binario

- Paso 7: Cálculo de las métricas de rama.
- Paso 8: Calcular los valores "hacia delante" $A_{k-1}(s')$.
- Paso 9: Calcular los valores "hacia atrás" $B_k(s)$.
- Paso 10: Cálculo de las posibilidades a priori, en dominio logarítmico.
- Paso 11: Enviar las probabilidades a posteriori como probabilidades a priori al decodificador.

d) Algoritmo LDPC no binario

- Paso 12: Inicio algoritmo LDPC no binario.
- Paso 13. Cálculo de los valores R_{ij}^X en el dominio de la frecuencia.
- Paso 14. Transformación de los R_{ij}^X al dominio del tiempo.
- Paso 15: Reordenamiento de los valores R_{ij}^X.
- Paso 16: Cálculo de las probabilidades a posteriori.

e) Siguiente Iteración

- Paso 17: Envío de la señal al ecualizador (retroalimentación).
- Paso 18: Cálculo de métricas de rama en el ecualizador, usando información a priori proveniente del decodificador LDPC.
- Paso siguiente: Volver al Paso 8

Ahora, se explicará detalladamente cada uno de los pasos, ayudándose con ejemplos numéricos.

Paso 1: Encontrar una matriz H válida. Una matriz de paridad H, tiene que cumplir con una serie de requerimientos para que el funcionamiento del algoritmo de codificación y decodificación sea óptimo.
Esta matriz debe cumplir con las condiciones mencionadas en la Sección 3.7 y en las publicaciones [2] y [18] se explican métodos constructivos para estas matrices.

A modo de ilustración se puede tomar la siguiente matriz H:

$$H = \left[\begin{array}{ccccc:ccccc} 0 & 1 & 2 & 3 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 3 & 0 & 0 & 3 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 2 & 3 \\ 2 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 2 \\ 2 & 0 & 3 & 0 & 1 & 0 & 2 & 0 & 2 & 0 \end{array}\right]$$

El grafo bipartito correspondiente a esta matriz de codificación corresponde al que se muestra en la Figura 4.14:

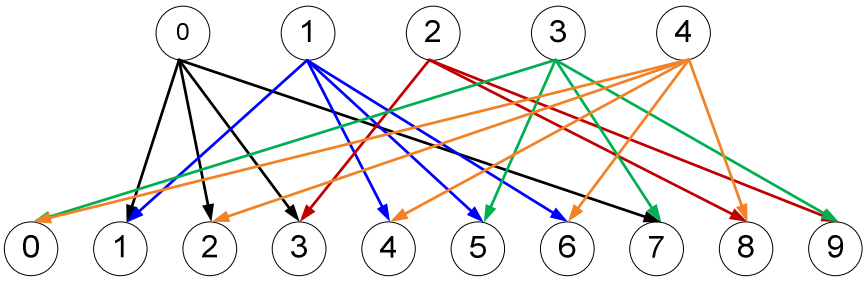


Figura 4.14. Grafo bipartito matriz H de ejemplo

Paso 2: Obtener matriz G. A partir de la matriz H escogida anteriormente se debe encontrar una matriz generadora G, como se explica anteriormente en la sección 3.7, donde a partir de la matriz H se obtiene la forma sistemática. La matriz H de forma sistemática tiene la siguiente estructura $H = [P \quad \vdots \quad I]$

Este proceso se refleja simplemente pivoteando la matriz para encontrar la forma requerida siguiendo la misma matriz del ejemplo se obtiene.

$$H \rightarrow H' = \left[\begin{array}{ccccc:ccccc} 1 & 2 & 2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 3 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 3 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Estamos tomando la nueva matriz H' de forma sistemática, es fácil determinar la matriz generadora G, imponiendo que

$$G = \left[\begin{array}{c} I \\ \cdots \\ P \end{array}\right] = \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 2 & 1 & 0 \\ 1 & 1 & 2 & 1 & 3 \\ 0 & 2 & 3 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 3 & 0 & 2 & 0 \end{array}\right]$$

Paso 3: Ya con la matriz G calculada, comienza el proceso de codificación. Para ello, se toma una secuencia cualquiera que será el mensaje que será enviado al extremo receptor. Por ejemplo se tomará la secuencia $m^T = [2\ 0\ 1\ 2\ 2]$ y a partir de la multiplicación de este vector con la matriz generadora se obtiene la secuencia codificada, por ejemplo:

$$c = G\,.m = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 2 & 1 & 0 \\ 1 & 1 & 2 & 1 & 3 \\ 0 & 2 & 3 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 3 & 0 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 0 \\ 1 \\ 2 \\ 2 \end{bmatrix} = [2 \quad 0 \quad 1 \quad 2 \quad 2 \quad 2 \quad 3 \quad 1 \quad 2 \quad 3]$$

Es importante señalar que la palabra codificada contiene una parte sistemática, correspondiente a los cinco primeros símbolos y una parte correspondiente a la información de paridad, representada por los cinco últimos símbolos.

Paso 4: Aplicar interleaver y enviar palabra codificada. Como se explicó en la Sección 3.7, es necesario aplicar un interleaver a la señal para mejorar el desempeño del esquema. Por ejemplo, al mensaje codificado obtenido en el Paso 3, se le puede aplicar un interleaver pseudo-aleatorio, que aplica el siguiente patrón de permutación.

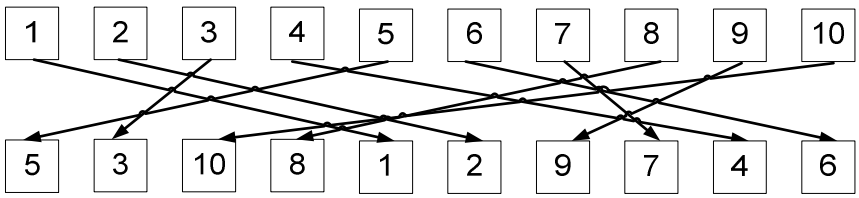


Figura 4.15. Patrón de permutación interleaver pseudo-aleatorio para ejemplo

Por lo que los símbolos de la señal codificada son permutados y se obtiene la siguiente palabra:

$$m_p = [2 \quad 1 \quad 3 \quad 1 \quad 2 \quad : \quad 0 \quad 2 \quad 3 \quad 2 \quad 2]$$

Ahora el mensaje está listo para ser enviado.

Paso 5: Paso del mensaje por el canal. El canal mostrado en la sección 3.4 se ajusta al modelo TDL Gaussiano y es el medio sobre el cual la señal sufre distorsión. El modelo mostrado en la Figura 3.12 además de producir ISI también adiciona AWGN, donde el simulador permite introducir un valor correspondiente a la amplitud de este ruido. Fijando la amplitud de la señal y

modificando la amplitud del ruido, permite modificar el SNR, que es un parámetro que se variará en los análisis posteriores.

La Tabla 4.1 muestra como se produce la distorsión de la señal digital, asumiendo estado inicial $x_{-1} = x_{-2} = 0$

Tabla 4.1. Distorsión de la señal usando canal de tres derivaciones del ejemplo.

Índice	X_k	X_{k-1}	X_{k-2}	Resultado	Ruido(SNR)	Enviada = Resultado + Ruido
0	2	0	0	0,814	0,042	0,856
1	1	2	0	2,037	0,075	2,112
2	3	1	2	2,444	-0,026	2,418
3	1	3	1	3,056	0,006	3,062
4	2	1	3	2,241	0,132	2,373
5	0	2	1	1,834	0,075	1,909
6	2	0	2	1,222	-0,061	1,161
7	3	2	0	2,851	0,103	2,954
8	2	3	2	3,667	0,018	3,685
9	2	2	3	3,056	0,074	3,130

Esta señal distorsionada es la que finalmente llega al receptor.

Paso 6: Trazado del diagrama de Trellis. En el extremo receptor, es necesario estimar el canal mediante el mismo modelo TDL explicado en la sección 3.4. El diagrama de Trellis que traza el receptor, asumiendo el canal de la Figura 3.11 se dibuja de la siguiente forma:

- Como el canal es de tres derivaciones, para entregar una salida, se debe tener información sobre el estado anterior (x_{k-1}) y el estado precedente al anterior (x_{k-2}) del símbolo de entrada. Por ejemplo, se puede asumir un estado inicial donde los estados $x_{k-1} = 0$ y $x_{k-2} = 0$. Entonces se deben considerar todas las entradas posibles teniendo en cuenta los estados anteriores y aplicar el modelo del canal, por ejemplo:

Tabla 4.2. Valores salidas de diagrama de Trellis para $x_{k-1} = x_{k-2} = 0$

	X_k	X_{k-1}	X_{k-2}	Resultado= 0,407*X_k + 0,815*X_{k-1} + 0,407*X_{k-2}
Caso 1	0	0	0	0
Caso 2	1	0	0	0,407
Caso 3	2	0	0	0,814
Caso 4	3	0	0	1,221

Los valores calculados en la Tabla 4.2 corresponden a los casos mostrados en la Figura 4.16:

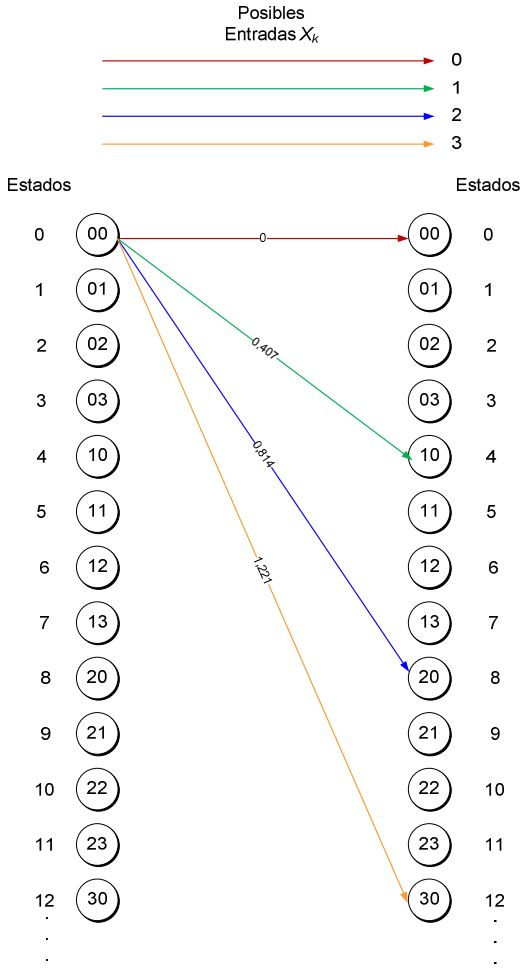


Figura 4.16. Valores de cálculo para ramas de diagrama de Trellis, para el caso de estado inicial $x_{k-1} = x_{k-2} = 0$

Este mismo proceso se debe repetir considerando todas las combinaciones entre los estados posibles x_{k-1} y x_{k-2}. Finalmente se obtiene un diagrama de Trellis como el de la Figura 3.20, con los valores correspondientes para cada salida posible indicados en el Anexo 1.

Paso 7: Cálculo de las métricas de rama. Este cálculo representa la similitud entre la secuencia recibida y alguna posible secuencia representada en el diagrama de Trellis, de manera de estimar la palabra enviada. Estas métricas de rama se calculan mediante la ecuación (49), que para la primera iteración se asume que la información a priori ($P(u_m)$) tiene igual probabilidad para todos los símbolos, por lo que se puede despreciar. Esta fórmula se repite aquí por conveniencia:

$$\Gamma_m(s', s) = -\frac{1}{2\sigma^2}(y_m - \hat{y}_m)^2 + \ln\big(P(u_m)\big)$$

El valor de σ se asume igual a 0,5. Esta fórmula dice que se debe calcular la distancia entre cada uno de los símbolos recibidos y los distintos estados del diagrama de Trellis. Por

ejemplo, el diagrama de Trellis calculado en el paso 6 contiene 64 estados, entonces se debe comparar el primer símbolo recibidos con cada uno de los 64 estados del diagrama.

Siguiendo el ejemplo se calculan las matrices de rama para los primeros cuatro estados, donde x_{k-1} y $x_{k-2} = 0$ y las entradas x_k varían entre 0 y 3 con lo que se obtiene:

Tabla 4.3. Métricas de rama, para cuatro primeros casos

	k=1
$\boldsymbol{\Gamma_k(0,0)}$	-1,466
$\boldsymbol{\Gamma_k(0,4)}$	-0,404
$\boldsymbol{\Gamma_k(0,8)}$	-0,004
$\boldsymbol{\Gamma_k(0,12)}$	-0,266

Este proceso se debe repetir para cada símbolo recibido y para los 64 caminos del diagrama resultando un total de 640 valores de métricas de rama (64 estados x 10 símbolos recibidos), los que se muestran en el Anexo 2.

Paso 8: Calcular los valores "hacia delante" $A_{k-1}(s')$. Para calcular estos valores, se debe tener clara su definición explicada en la Sección 3.6.2, donde se visualiza que los valores $A_k(s)$ tienen directa dependencia en los valores $A_{k-1}(s')$ y con las métricas de rama $\Gamma_k(s', s)$. Estas dependencias están ligadas al diagrama de Trellis donde se analizan todos los caminos posibles $\Gamma_k(s', s)$ para llegar a otro estado $A_k(s)$ a partir de un estado anterior $A_{k-1}(s')$.

Para ejemplificar este cálculo, es preciso observar el diagrama de la Figura 4.17 donde se analizan todos los estados anteriores y caminos posibles para llegar al estado $A_k(s) = 00$.

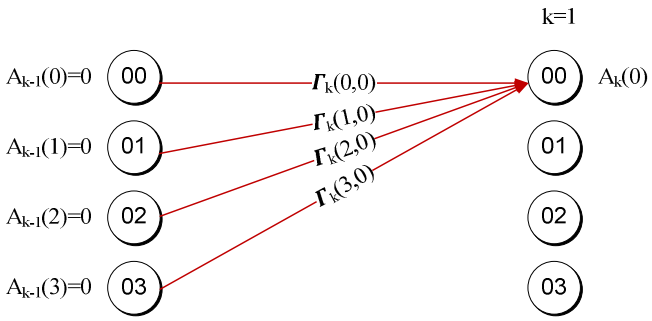


Figura 4.17. Dependencia de caminos y estados anteriores para el cálculo de $A_1(0)$

El algoritmo impone que los valores de $A_k(s)$ para $k = 0$ son todos cero, por lo que se obtiene que $A_1(0)$ vale:

$$A_1(0) = \ln\big(\exp(A_0(0) + \Gamma_1(0,0)) + \exp\big(A_0(1) + \Gamma_1(1,0)\big) + \exp\big(A_0(2) + \Gamma_1(2,0)\big) + \exp\big(A_0(3) + \Gamma_1(3,0)\big)\big)$$

$$= \ln(\exp(0 - 1{,}466) + \exp(0 - 0{,}851) + \exp(0 - 0{,}402) + \exp(0 - 0{,}119))$$

$$= 0{,}795$$

Igualmente para calcular $A_1(1)$ se observan las dependencias según los posibles estados anteriores.

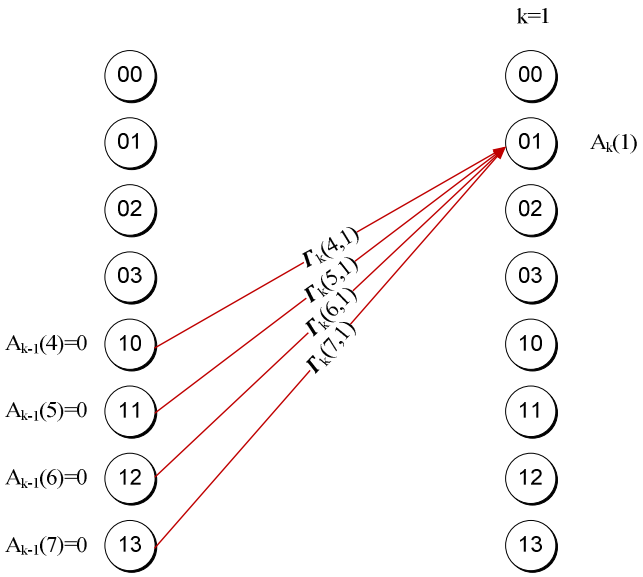


Figura 4.18. Dependencia de caminos y estados anteriores para el cálculo de $A_1(1)$

Calculando este valor se obtiene:

$$A_1(0) = \ln\big(\exp(A_0(4) + \Gamma_1(4,1)\,) + \exp\big(A_0(5) + \Gamma_1(5,1)\big) + \exp\big(A_0(6) + \Gamma_1(6,1)\big) + \exp\big(A_0(7) + \Gamma_1(7,1)\big)\big)$$

$$= \ln(\exp(0 - 0{,}003) + \exp(0 - 0{,}053) + \exp(0 - 0{,}269) + \exp(0 - 0{,}651))$$

$$= 1{,}173$$

Asimismo se continúa el cálculo para todos los estados de $A_k(j)$ en el diagrama hasta llegar a $j = 15$ y posteriormente repitiendo el cálculo para todos los símbolos de la palabra recibida hasta a $k = 10$ Con ellos se obtienen los valores mostrados en el Anexo 3.

Paso 9: Calcular los valores "hacia atrás" $B_k(s)$. Tal como lo explica su nombre, estos valores se calculan "hacia atrás". Esta denominación se explica en la sección 3.6.2 donde los valores $B_{k-1}(s')$ tienen directa dependencia con los valores $B_k(s)$ y las métricas de rama $\Gamma_k(s', s)$. Esta dependencia está ligada al diagrama de Trellis donde se analizan los caminos$\Gamma_k(s', s)$ para llegar desde un estado actual $B_k(s)$ a un estado $B_{k-1}(s')$.

Para ilustrar este cálculo, es de gran ayuda observar la Figura 4.19, en la cual se muestran todos los caminos posibles para los estados $B_{10}(s)$ donde el estado anterior fue $B_9(0)$.

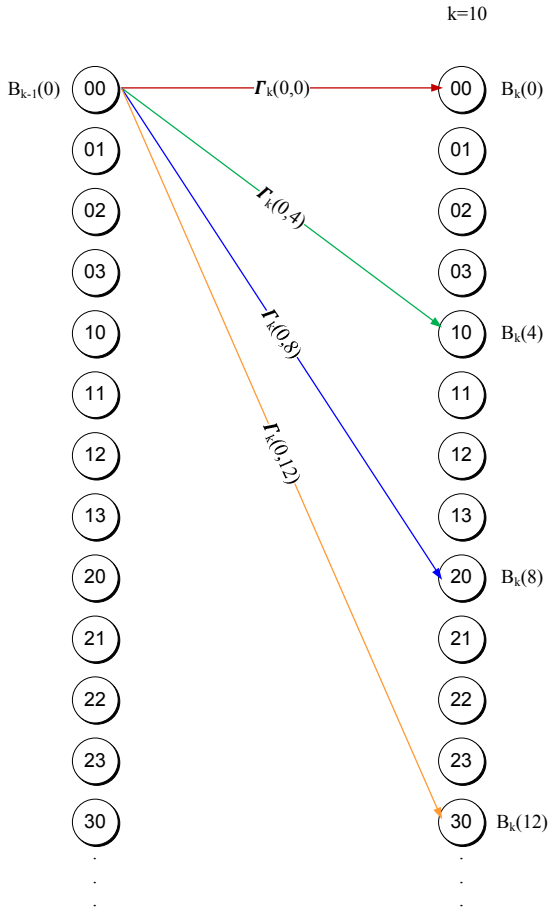


Figura 4.19. Caminos posibles para los estados $B_{10}(s)$ donde el estado anterior fue $B_9(0)$

Similarmente al cálculo de los valores $A_k(s)$, el algoritmo impone que los valores de $B_k(s)$ para el último símbolo (en este caso $k = 10$) son todos cero. Consecuentemente, se calcula el valor de $B_9(0)$ usando la siguiente lógica.

$$B_9(0) = \ln\left(\exp(B_{10}(0) + \Gamma_{10}(0,0)) + \exp\left(B_{10}(4) + \Gamma_{10}(0,4)\right) + \exp\left(B_{10}(8) + \Gamma_{10}(0,8)\right) + \exp\left(B_{10}(12) + \Gamma_{10}(0,12)\right)\right)$$

$$= \ln\left(\exp(0 - 19{,}578) + \exp(0 - 14{,}833) + \exp(0 - 10{,}731) + \exp(0 - 7{,}291)\right)$$

$$= -7{,}259$$

De la misma manera para calcular $B_9(1)$, se observan las dependencias posibles de los estados futuros $B_{10}(k)$

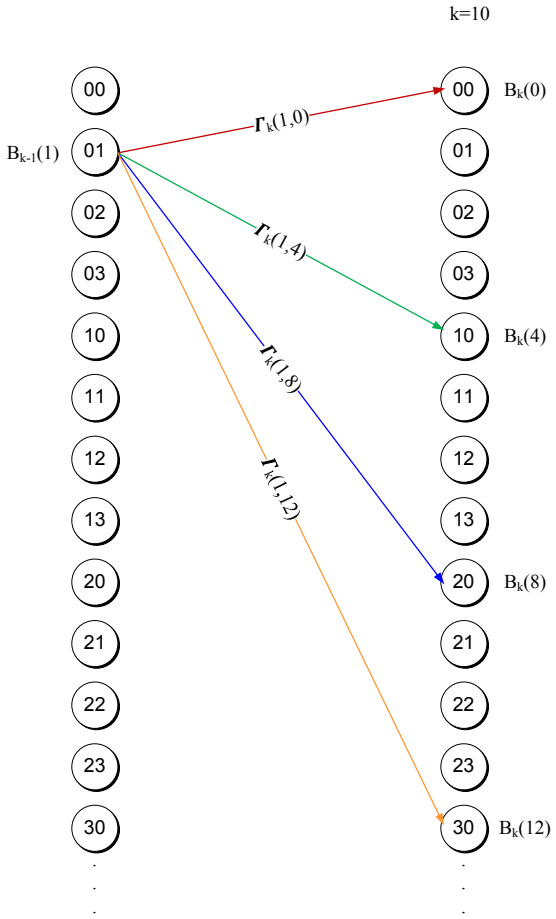


Figura 4.20. Caminos posibles para los estados $B_{10}(s)$ donde el estado anterior fue $B_9(1)$

Calculando este valor se obtiene:

$$B_9(1) = \ln\left(\exp(B_{10}(0) + \Gamma_{10}(1,0)) + \exp\left(B_{10}(4) + \Gamma_{10}(1,4)\right) + \exp\left(B_{10}(8) + \Gamma_{10}(1,8)\right) + \exp\left(B_{10}(12) + \Gamma_{10}(1,12)\right)\right)$$

$$= \ln\left(\exp(0 - 17{,}127) + \exp(0 - 12{,}693) + \exp(0 - 8{,}924) + \exp(0 - 5{,}816)\right)$$

$$= -5{,}772$$

Posteriormente se continúa con el cálculo pasado por todos los posibles estados anteriores $B_k(j)$, con $j = 0, \ldots, 15$ y para todos los símbolos, o sea de $k = 10, 9, \ldots, 1$ y con ellos se obtienen los resultados del Anexo 4.

Paso 10: Cálculo de las posibilidades a priori, en dominio logarítmico. En la sección 4.2 se muestra el procedimiento para el cálculo de estas probabilidades que se realiza una vez calculados los valores de $A_k(s')$, $\Gamma_k(s', s)$ y $B_k(s)$ para todos los símbolos y todos los estados del diagrama de Trellis. La forma para calcularla se muestran en (48), pero se repite acá por conveniencia:

$$\ln\big(P(u_m = a|y)\big) = \ln\left(\sum_{(s',s)\Rightarrow u_m=a} \exp\left(A_{m-1}(s') + \Gamma_m(s',s) + B_m(s)\right)\right)$$

Esta fórmula dice que el logaritmo de la probabilidad a posteriori de que un símbolo u_m sea igual al símbolo a, dado que la secuencia recibida es y, se calcula considerando todos los estados anteriores ($A_{m-1}(s')$) las transiciones ($\Gamma_m(s',s)$) y los estados posteriores $B_m(s)$, sobre los cuales el símbolo recibido pudo haber sido a. En la Figura 4.2, se observa las transiciones para cada uno de los símbolos, las cuales han sido separadas por color (rojo: símbolo cero, verde: símbolo uno, azul: símbolo dos, naranjo: símbolo tres) Para ejemplificar esto se muestra un diagrama en la Figura 4.21 para el caso del símbolo cero:

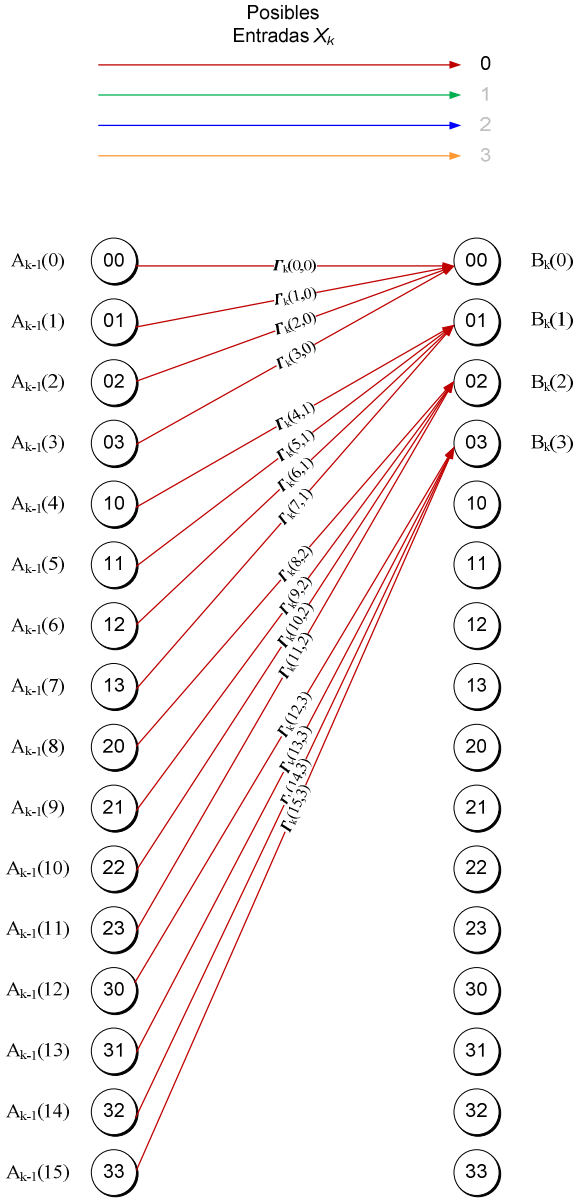


Figura 4.21. Estados anteriores (A_{m-1} (s')) las transiciones (Γ_m(s',s)) y los estados posteriores B_m(s) para el símbolo cero de GF(4)

Por lo tanto las probabilidades para cada símbolo se calculan de la siguiente forma.

$$\begin{aligned}\mathrm{Ln}\big(P(u_m = 0|0{,}856)\big) &= \ln\big(\exp\big(A_0(0) + \Gamma_1(0,0) + B_1(0)\big) + \exp\big(A_0(1) + \Gamma_1(1,0) + B_1(0)\big) \\ &+ \exp\big(A_0(2) + \Gamma_1(2,0) + B_1(0)\big) + \exp\big(A_0(3) + \Gamma_1(3,0) + B_1(0)\big) \\ &+ \exp\big(A_0(5) + \Gamma_1(5,1) + B_1(1)\big) + \exp\big(A_0(6) + \Gamma_1(6,1) + B_1(1)\big) \\ &+ \exp\big(A_0(7) + \Gamma_1(7,1) + B_1(1)\big) + \exp\big(A_0(8) + \Gamma_1(8,1) + B_1(1)\big) \\ &+ \exp\big(A_0(8) + \Gamma_1(8,2) + B_1(2)\big) + \exp\big(A_0(9) + \Gamma_1(9,2) + B_1(2)\big) \\ &+ \exp\big(A_0(10) + \Gamma_1(10,2) + B_1(2)\big) + \exp\big(A_0(11) + \Gamma_1(11,2) + B_1(2)\big) \\ &+ \exp\big(A_0(12) + \Gamma_1(12,3) + B_1(3)\big) + +\exp\big(A_0(13) + \Gamma_1(13,3) + B_1(3)\big) \\ &+ \exp\big(A_0(14) + \Gamma_1(14,3) + B_1(3)\big) + \exp\big(A_0(15) + \Gamma_1(15,3) + B_1(3)\big) \\ &= 4{,}087\end{aligned}$$

Asimismo para los demás símbolos se obtienen los siguientes valores de logaritmos de probabilidades:

$$\mathrm{Ln}\big(P(u_m = 1|0{,}856)\big) = 5{,}578$$

$$\mathrm{Ln}\big(P(u_m = 2|0{,}856)\big) = 5{,}668$$

$$\mathrm{Ln}\big(P(u_m = 3|0{,}856)\big) = 4{,}040$$

Extendiendo este cálculo para todos los símbolos recibidos se obtienen las probabilidades del Anexo 5.

A partir de esta información se puede determinar posible secuencia recibida eligiendo el valor máximo entre los logaritmos de probabilidad entre los cuatro símbolos posibles. Para el primer símbolo se tiene que el máximo corresponde a $u_m = 2$ por lo que el símbolo recibido concuerda con el símbolo enviado por el emisor. En la Tabla 4.4 se muestra la decisión sobre los símbolos enviados una vez aplicado el algoritmo log-MAP y los errores encontrados. Cabe señalar que en los gráficos que se mostrarán posteriormente el valor del SER (Symbol Error Rate) se calcula contabilizando los símbolos errados y se dividen por el total de símbolos.

Tabla 4.4. Probabilidades a posteriori ecualizador log-MAP

	k=1	**k=2**	**k=3**	**k=4**	**k=5**	**k=6**	**k=7**	**k=8**	**k=9**	**k=10**
Pbb(0)	4,087	3,937	1,779	3,631	3,954	6,093	1,676	-2,737	-0,134	4,972
Pbb(1)	5,572	5,565	4,495	5,477	5,522	5,295	4,806	2,616	4,053	5,229
Pbb(2)	5,678	5,607	5,628	5,669	5,718	3,349	5,961	5,286	5,698	5,219
Pbb(3)	4,040	4,466	5,697	4,653	4,175	-0,064	5,049	6,132	5,752	5,049
Decisión	2	2	3	2	2	0	2	3	3	1
Correcto	2	1	3	1	2	0	2	3	2	2
Error?	correcto	**error**	correcto	**error**	correcto	correcto	correcto	correcto	**error**	**error**
Antes de EQ	1	2	3	3	3	2	2	3	3	3

Error pre EQ	**error**	**error**	correcto	**error**	**error**	**error**	correcto	correcto	**error**	**error**

Por lo mostrado en la Tabla 4.4 se contabiliza un SER de $\frac{4}{10} = 0{,}4$.

En la casilla "Antes de EQ" se muestra la secuencia recibida antes de aplicar ecualización y en la casilla "Error pre EQ" se contabilizan los errores en la secuencia recibida antes de aplicar ecualización, por lo que si no se aplicara el proceso, se contabilizaría un SER de $\frac{8}{10} = 0{,}8$.

Paso 11: Enviar las probabilidades a posteriori como probabilidades a priori al decodificador. Este paso es sencillo, pero sumamente importante para que el proceso iterativo se produzca exitosamente. Las probabilidades a posteriori del ecualizador deben pasar por un deinterleaver, aplican función exponencial (para obtener probabilidades), normalizarlas y transponerlas (este último pero se hace por conveniencia).

Como consecuencia se obtienen las siguientes probabilidades a priori para el decodificador LDPC no binario (el resumen del cálculo se muestra en el Anexo 6)

Tabla 4.5. Probabilidades a priori para LDPC no binario

Índice	0	1	2	3
0	0,078	0,373	0,453	0,097
1	0,659	0,297	0,042	0,001
2	0,076	0,389	0,405	0,130
3	0,001	0,086	0,444	0,469
4	0,089	0,391	0,435	0,085
5	0,215	0,278	0,275	0,232
6	0,000	0,020	0,294	0,685
7	0,056	0,356	0,431	0,156
8	0,008	0,182	0,578	0,232
9	0,009	0,133	0,414	0,444

Paso 12: Inicio algoritmo LDPC no binario. Obtención de matrices de permutación según matriz H, tal como se explicó en la sección 3.7.1, el primer paso de este algoritmo corresponde al cálculo de los Q_{ij}^{X}. Como se tiene información previa (probabilidad a priori), el cálculo de los Q_{ij}^{X} se facilita al utilizar esta información. Como se tienen cinco chequeos de paridad (determinado según la matriz H) entonces se obtendrán cinco matrices Q_{ij}^{X}, una por cada chequeo. Para ejemplificar se calculará la primera.

Al observar la matriz H del ejemplo se observa en el primer chequeo de paridad (primera fila de H) que existen cuatro índices distintos de cero, en cada columna 1, 2, 3, 7 y estos valores corresponden a 1, 2, 3 y 3, respectivamente.

De este modo, es como deben tomarse las filas de la matriz de probabilidades a priori según los índices en que H_{ij} es distinto de cero, es decir, para el primer chequeo serían las filas 1, 2, 3 y 7 y

se deben reordenar según la operación "inversa de multiplicación" aplicada a los valores $(1, 2, 3, 3)$. Por lo tanto el reordenamiento que se hará para obtener la matriz Q_{ij}^{X} será:

$$1 \div (0, 1, 2, 3) = (0, 1, 2, 3)$$

$$2 \div (0, 1, 2, 3) = (0, 3, 1, 3)$$

$$3 \div (0, 1, 2, 3) = (0, 2, 3, 1)$$

$$3 \div (0, 1, 2, 3) = (0, 2, 3, 1)$$

Esto implica que la matriz Q_{ij}^{X} para el primer chequeo de paridad, o sea, Q_{ij}^{0} corresponde a:

Tabla 4.6. Cálculo de Q_{ij}^{0}

	0	1	2	3
1	0,659	0,297	0,042	0,001
2	0,076	0,130	0,389	0,405
3	0,001	0,444	0,469	0,086
7	0,056	0,431	0,156	0,356

Paso 12: Transformar los resultados de Q_{ij}^{X} al dominio de la frecuencia. Este paso es relativamente sencillo y corresponde a multiplicar cada una de las matrices Q_{ij}^{X} por la matriz WHM para *GF(4)*. Haciendo este cálculo resulta:

$$\widehat{Q_{ij}^{0}} = Q_{ij}^{0} \cdot WHM(4) = \begin{bmatrix} 0{,}659 & 0{,}297 & 0{,}042 & 0{,}001 \\ 0{,}076 & 0{,}130 & 0{,}389 & 0{,}405 \\ 0{,}001 & 0{,}444 & 0{,}469 & 0{,}086 \\ 0{,}056 & 0{,}431 & 0{,}156 & 0{,}356 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0{,}404 & 0{,}912 & 0{,}322 \\ 1 & -0{,}070 & -0{,}588 & -0{,}037 \\ 1 & -0{,}059 & -0{,}109 & -0{,}826 \\ 1 & -0{,}575 & -0{,}025 & -0{,}175 \end{bmatrix}$$

Posteriormente se realiza este mismo cálculo para las demás matrices de chequeo, cuyos resultados se muestran en el Anexo 9.

Paso 13. Cálculo de los valores $\widehat{R_{ij}^{X}}$ en el dominio de la frecuencia. El valor de $\widehat{R_{ij}^{X}}$ corresponde a la información que envía el nodo de paridad h_i al nodo símbolo d_j.

Pasar los valores Q_{ij}^{X} al dominio de la frecuencia trajo el beneficio de facilitar el cálculo para obtener los valores R_{ij}^{X}. Consecuentemente el procedimiento de cálculo procede como sigue:

En el dominio de la frecuencia la convolución (es decir, el cálculo de los valores $\widehat{R_{ij}^{0}}, \dots, \widehat{R_{ij}^{q-1}}$) se obtiene por medio de multiplicaciones como en la ecuación (41), por lo que en el ejemplo se obtiene que:

$$\widehat{R_{00}^{0}} = \widehat{Q_{10}^{0}} \cdot \widehat{Q_{20}^{0}} \cdot \widehat{Q_{30}^{0}} = 1 \cdot 1 \cdot 1 = 1$$

Este cálculo se puede visualizar con el siguiente esquema:

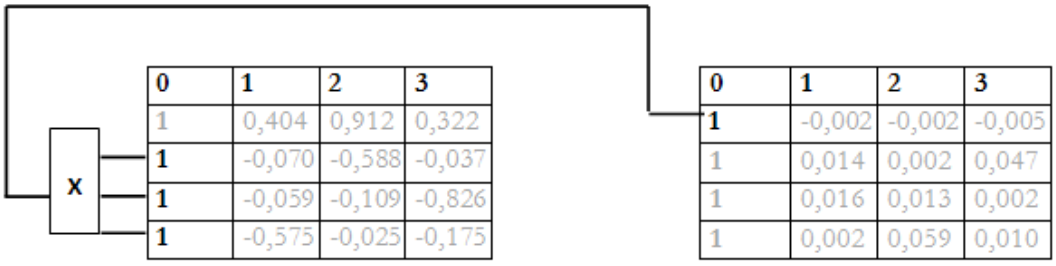


Figura 4.22. Esquema para cálculo de R_{00}^{0}

Asimismo, el valor de $\widehat{R_{01}^{0}}$ se calcula como:

$$\widehat{R_{01}^{0}} = \widehat{Q_{11}^{0}} \cdot \widehat{Q_{21}^{0}} \cdot \widehat{Q_{31}^{0}} = -0{,}070 \cdot -0{,}059 \cdot -0{,}575 = -0{,}002$$

Lo que se visualizaría como:

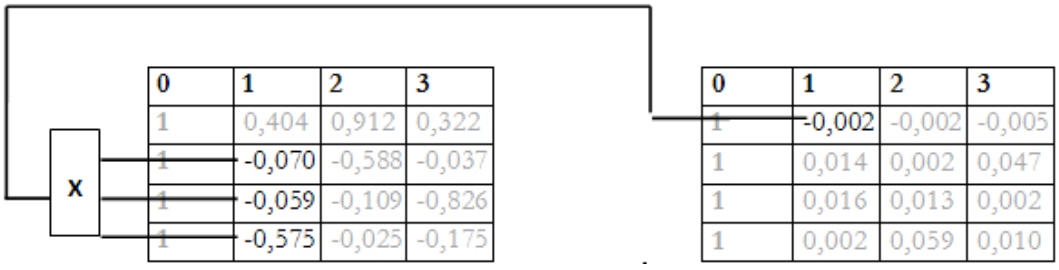


Figura 4.23. Esquema para cálculo de R_{01}^{0}

Este procedimiento se repite para calcular todos los valores de $\widehat{R_{ij}^{X}}$, con i=0,…,3 (símbolos de *GF(4)*), *j*=0,…,3 (cantidad de nodos padre unidos al nodo hijo *X*) y para *X*=0,…,4 , correspondiente a la cantidad de nodos de paridad o nodos hijo.

De este modo se obtienen los valores de $\widehat{R_{ij}^{X}}$ mostrados en el Anexo 10.

Paso 14. Transformación de los $\widehat{R_{ij}^{X}}$ al dominio del tiempo. Este paso es relativamente sencillo y consiste en multiplicar las matrices $\widehat{R_{ij}^{X}}$ por la inversa de la matriz WHM, determinada por:

$$\begin{bmatrix} 0{,}25 & 0{,}25 & 0{,}25 & 0{,}25 \\ 0{,}25 & -0{,}25 & 0{,}25 & -0{,}25 \\ 0{,}25 & 0{,}25 & -0{,}25 & -0{,}25 \\ 0{,}25 & -0{,}25 & -0{,}25 & 0{,}25 \end{bmatrix}$$

Haciendo esta multiplicación con la matriz se obtiene el valor de R_{ij}^{0}:

$$R_{ij}^{0} = \widehat{R_{ij}^{0}} \cdot WHM^{-1} = \begin{bmatrix} 1 & -0{,}002 & -0{,}002 & -0{,}005 \\ 1 & 0{,}014 & 0{,}002 & 0{,}047 \\ 1 & 0{,}016 & 0{,}013 & 0{,}002 \\ 1 & 0{,}002 & 0{,}059 & 0{,}010 \end{bmatrix} \cdot \begin{bmatrix} 0{,}25 & 0{,}25 & 0{,}25 & 0{,}25 \\ 0{,}25 & -0{,}25 & 0{,}25 & -0{,}25 \\ 0{,}25 & 0{,}25 & -0{,}25 & -0{,}25 \\ 0{,}25 & -0{,}25 & -0{,}25 & 0{,}25 \end{bmatrix}$$

$$= \begin{bmatrix} 0{,}248 & 0{,}252 & 0{,}251 & 0{,}250 \\ 0{,}266 & 0{,}236 & 0{,}241 & 0{,}258 \\ 0{,}258 & 0{,}249 & 0{,}250 & 0{,}243 \\ 0{,}268 & 0{,}262 & 0{,}233 & 0{,}237 \end{bmatrix}$$

Paso 15: Reordenamiento de los valores R_{ij}^{X}. Según como se dijo en el Paso 11, los valores de Q_{ij}^{X} fueron ordenados según la operación "inversa de multiplicación", entonces en este Paso se debe volver al orden inicial. Esto se realiza reordenando según la operación multiplicación, procedimiento que sigue el siguiente patrón para R_{ij}^{0}:

$$(0,1,2,3) \times 1 = (0,1,2,3)$$

$$(0,1,2,3) \times 2 = (0,2,3,1)$$

$$(0,1,2,3) \times 3 = (0,3,1,2)$$

$$(0,1,2,3) \times 3 = (0,3,1,2)$$

Por lo que se obtienen los valores de R_{ij}^{X} mostrados en el Anexo 12.

Paso 16: Cálculo de las probabilidades a posteriori. Este procedimiento se realiza por medio de la ecuación (42), teniendo en cuenta que:

f_n^a corresponde a las probabilidades a priori provenientes del ecualizador.

$R_{k,n}^{a}$ corresponde a los valores R_{ij}^{X} calculados en el Paso 15

$\propto_n$ son constantes de normalización

n varía entre 0 y , el cual se encuentra determinado por la cantidad de símbolos enviados.

a varía desde 0 a 3 (según los símbolos de *GF(4)*)

La multiplicación de los valores $R_{k,n}^{a}$ se realiza para todos los nodos de paridad conectados a un nodo símbolo d_j. Esta dependencia se observa en el grafo bipartito Figura 4.14.

Por ejemplo, la probabilidad de que el primer símbolo sea cero se calcula:

$$P_0^0 = R_{00}^3 \cdot R_{00}^4 \cdot f_0^0 = 0{,}005$$

Igualmente se puede calcular la probabilidad de que el segundo símbolo sea cero, de la siguiente manera:

$$P_1^0 = R_{00}^0 \cdot R_{00}^1 \cdot f_1^0 = 0{,}041$$

Este procedimiento se repite para todos los símbolos (0 a 9) y para todos los elementos del campo de Galois de orden 4, o sea, 0, 1, 2 y 3, para con ello obtener las probabilidades a posteriori, las cuales finalmente se normalizan para tomar la decisión sobre la secuencia recibida. Este cálculo se muestra en el Anexo 13 y a modo de ejemplo en la Tabla 4.7, donde también se indica el número de símbolos errados.

Tabla 4.7. Cálculo probabilidades a posteriori decodificador LDPC

	0	**1**	**2**	**3**	**Sobrev.**	**Correcto**	**Resultado**
0	0,076	0,368	0,458	0,099	2	2	Correcto
1	0,654	0,302	0,043	0,001	0	0	Correcto
2	0,081	0,380	0,417	0,122	2	1	**Error**
3	0,001	0,057	0,466	0,476	3	2	**Error**
4	0,089	0,358	0,469	0,085	2	2	Correcto
5	0,225	0,253	0,313	0,210	2	2	Correcto
6	0,000	0,020	0,294	0,686	3	3	Correcto
7	0,060	0,337	0,456	0,147	2	1	**Error**
8	0,012	0,066	0,832	0,090	2	2	Correcto
9	0,008	0,095	0,447	0,451	3	3	Correcto

Es importante señalar que el SER ha disminuido a 0,3.

Segunda Iteración

Paso 17: Envío de la señal al ecualizador (retroalimentación). Para que se produzca la retroalimentación es necesario enviar las probabilidades calculadas por el decodificador LDPC. Para ello se debe aplicar nuevamente un interleaver, que tiene la misma forma del que se utilizó en el Paso 4, al momento de enviar la señal. Es así como se obtienen las probabilidades de la Tabla 4.8:

Tabla 4.8. Información a priori ecualizador, segunda iteración

Símbolo	**PBB(Um=0)**	**PBB(Um=1)**	**PBB(Um=2)**	**PBB(Um=3)**
0	0,089	0,358	0,469	0,085
1	0,081	0,380	0,417	0,122
2	0,008	0,095	0,447	0,451
3	0,060	0,337	0,456	0,147
4	0,076	0,368	0,458	0,099
5	0,654	0,302	0,043	0,001
6	0,012	0,066	0,832	0,090
7	0,000	0,020	0,294	0,686
8	0,001	0,057	0,466	0,476
9	0,225	0,253	0,313	0,210

Cable señalar que esta matriz se traspone por conveniencia, lo que se verá en el Paso siguiente, donde se calculan las métrica de rama utilizando esta información. La matriz traspuesta entonces quedaría de la siguiente forma:

Tabla 4.9. Información a priori segunda iteración, traspuesta

Símbolo	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
PBB(Um=0)	0,089	0,081	0,008	0,060	0,076	0,654	0,012	0,000	0,001	0,225
PBB(Um=1)	0,358	0,380	0,095	0,337	0,368	0,302	0,066	0,020	0,057	0,253
PBB(Um=2)	0,469	0,417	0,447	0,456	0,458	0,043	0,832	0,294	0,466	0,313
PBB(Um=3)	0,085	0,122	0,451	0,147	0,099	0,001	0,090	0,686	0,476	0,210

Paso 18: Cálculo de métricas de rama en el ecualizador, usando información a priori proveniente del decodificador LDPC. Esta información proveniente del LDPC pasa a ser la señal a priori del ecualizador, pero se debe tener clara la forma en que ésta debe ser introducida.

Este Paso es muy similar al Paso 7, donde se debe usar la ecuación (49) para calcular estas métricas de rama, con la diferencia que en este último no se tenía información a priori, por lo que fue fijada en cero, pero en este caso esta información a priori viene dada por el término ln $(P(u_m))$ y es distinta de cero.

Ahora, para calcular las métricas de rama, se debe sumar esta información a priori al cálculo de las métricas ya explicado en el Paso 7, dependiendo de la entrada esperada según el diagrama de Trellis. Este cálculo se hace más fácil al visualizar la Figura 4.24, que ejemplifica el cálculo de las métricas de rama para los cuatro primeros estados del diagrama de Trellis:

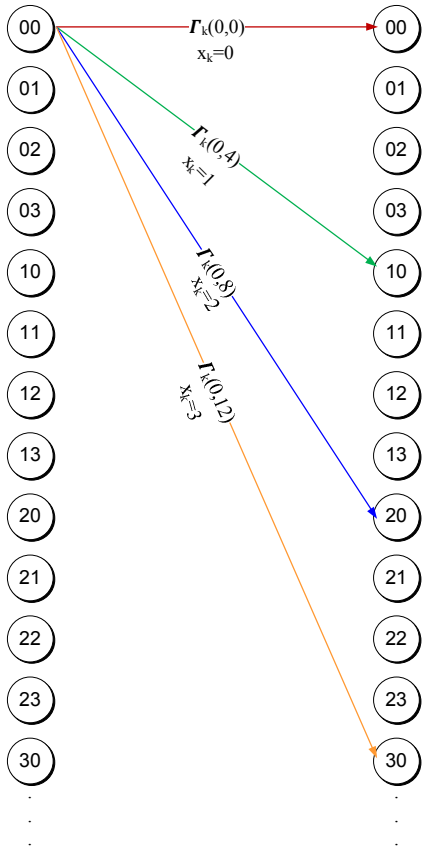


Figura 4.24. Ejemplo de cálculo de métricas de rama con información a priori

Es así, como se obtienen los nuevos valores de métricas de rama para estos cuatro casos:

$$\Gamma_1(0,0) = -2 \cdot (0{,}857 - 0)^2 + (-2{,}423) = -3{,}889$$

$$\Gamma_1(0,4) = -2 \cdot (0{,}857 - 0{,}407)^2 + (-2{,}513) = -1{,}430$$

$$\Gamma_1(0,8) = -2 \cdot (0{,}857 - 0{,}814)^2 + (-4{,}877) = -0{,}762$$

$$\Gamma_1(0,12) = -2 \cdot (0{,}857 - 1{,}221)^2 + (-2{,}815) = -2{,}736$$

El cálculo completo de estas métricas de rama se encuentra en el Anexo 14. De aquí en adelante el procedimiento de cálculo sigue idénticamente como en el Paso 8 y se continúa dependiendo de la cantidad de iteraciones se fije. Con esto termina la explicación paso a paso del algoritmo.

Siguiendo con el ejemplo, al terminar el proceso de ecualización log-MAP se obtienen las probabilidades a posteriori de la Tabla 4.10:

Tabla 4.10. Probabilidades a posteriori ecualizador log-MAP, segunda iteración

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Pbb(0)	-6,749	-6,702	-12,774	-7,315	-7,850	-2,649	-12,254	-21,784	-14,833	-4,107
Pbb(1)	-3,456	-3,188	-6,526	-3,373	-3,696	-4,413	-6,549	-10,588	-6,781	-3,737
Pbb(2)	-3,017	-3,248	-3,179	-3,086	-2,890	-9,697	-2,542	-4,451	-3,066	-3,543
Pbb(3)	-6,713	-6,464	-3,224	-6,092	-6,160	-18,396	-5,889	-2,642	-3,348	-4,290
Decisión	2	1	2	2	2	0	2	3	2	2
Correcto	2	1	3	1	2	0	2	3	2	2
Error?	correcto	Correcto	**error**	**Error**	correcto	correcto	correcto	correcto	correcto	correcto
Antes EQ	1	2	3	3	3	2	2	3	3	3
Error pre EQ	**error**	**error**	correcto	**Error**	**error**	**error**	correcto	correcto	**error**	**error**

La Tabla 4.10 da cuenta que ha vuelto a bajar el SER de la señal enviada, que ahora es de 0,2. Estos datos son pasados nuevamente al decodificador, el cual entrega los siguientes valores probabilísticos y las decisiones tomadas sobre cada símbolo recibido.

Tabla 4.11. Probabilidades a posteriori decodificador LDPC, segunda iteración

	0	1	2	3	Sobrev	Correcto	Resultado
0	0,004	0,223	0,746	0,027	2	2	Correcto
1	0,869	0,131	0,001	0,000	0	0	Correcto
2	0,017	0,650	0,321	0,012	1	1	Correcto
3	0,000	0,001	0,511	0,488	2	2	Correcto
4	0,012	0,160	0,820	0,009	2	2	Correcto
5	0,225	0,147	0,552	0,076	2	2	Correcto
6	0,000	0,000	0,112	0,888	3	3	Correcto
7	0,007	0,479	0,483	0,031	2	1	**Error**
8	0,000	0,001	0,998	0,001	2	2	Correcto
9	0,000	0,001	0,445	0,554	3	3	Correcto

La corrección de la palabra sigue mejorando y ahora el SER es de 0,1. Realizando otra iteración sobre el esquema ecualizador-decodificador, se tiene que ahora el ecualizador entrega los siguientes valores:

Tabla 4.12. Probabilidades a posteriori ecualizador log-MAP, tercera iteración

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Pbb(0)	-8,023	-6,613	-21,379	-7,865	-10,651	-0,876	-17,066	-32,522	-22,015	-2,537
Pbb(1)	-3,117	-1,060	-10,586	-1,343	-3,159	-4,215	-10,120	-14,186	-9,549	-2,687
Pbb(2)	-0,952	-2,490	-1,975	-1,779	-0,953	-13,701	-0,842	-4,077	-1,272	-1,350
Pbb(3)	-7,453	-8,616	-1,230	-7,124	-5,895	-29,281	-9,140	-0,882	-1,892	-3,703
Decisión	2	1	3	1	2	0	2	3	2	2
Correcto	2	1	3	1	2	0	2	3	2	2

Error?	correcto	correcto	correcto	correcto	correcto	correcto	correcto	correcto	correcto	correcto
Antes EQ	1	2	3	3	3	2	2	3	3	3
Error pre EQ	**error**	**error**	correcto	**error**	**error**	**error**	correcto	correcto	**error**	**error**

En la Tabla 4.12 se muestra que el SER ha disminuido a cero, es decir, se ha producido la corrección total de la palabra, igualmente, la información será enviada al decodificador para comprobar si el esquema converge. Al enviar finalmente esta información al decodificador se obtiene la siguiente Tabla de probabilidades y decisiones:

Tabla 4.13. Probabilidades a posteriori decodificador LDPC, tercera iteración

	0	**1**	**2**	**3**	**Sobrev**	**Correcto**	**Resultado**
0	0,000	0,002	0,997	0,001	2	2	Correcto
1	0,995	0,005	0,000	0,000	0	0	Correcto
2	0,000	0,994	0,006	0,000	1	1	Correcto
3	0,000	0,000	0,957	0,043	2	2	Correcto
4	0,000	0,001	0,999	0,000	2	2	Correcto
5	0,023	0,004	0,970	0,002	2	2	Correcto
6	0,000	0,000	0,001	0,999	3	3	Correcto
7	0,000	0,957	0,042	0,000	1	1	Correcto
8	0,000	0,000	1,000	0,000	2	2	Correcto
9	0,000	0,000	0,073	0,927	3	3	Correcto

Este resultado finalmente deja la conclusión que el esquema funciona y que en el ejemplo se pudo corregir totalmente una palabra de diez símbolos.

4.6 Análisis de factibilidad

En la literatura actual existen diversas discusiones acerca si los ecualizadores realmente pueden ser implementados en sistemas actuales, dado su alta complejidad de cálculo, donde se han cuestionado los resultados mostrados en los estudios relacionados con esta materia e incluso se ha cuestionado la convergencia de este tipo de códigos. En la publicación [17] donde se hace una comparación y un análisis de complejidad de códigos convolucionales y turbo identificando la cantidad de operaciones que se ejecutan y su desempeño. En este trabajo, se parte desde el punto de vista que la complejidad no es excusa para dejar de estudiar un cierto tipo de código, sobretodo por el hecho del acelerado crecimiento de las compañías fabricantes de hardware, que compiten constantemente en lanzar máquinas de procesamiento más potentes y a menor precio.

Para los análisis de este trabajo se utilizará el indicador SER (tasa de símbolos errados), mientras se varía el valor del SNR (razón señal a ruido) entre la señal enviada y el ruido introducido en el modelo de canal mostrado en la Sección 4.1.

En algunas publicaciones se utiliza el indicador E_b/N_0 que es la tasa de densidad espectral por bit sobre ruido, el cual se usa cuando se analizan principalmente esquemas de modulación. Como en este caso se analiza el poder correctivo de un código y un ecualizador se utiliza el indicador SNR. En todo caso, la relación entre el CNR (*Carrier to noise ratio*) o sea el SNR de la señal recibida después del filtro de detección es: $\frac{C}{N} = \frac{E_b}{N_0} \cdot \frac{f_b}{B}$, con f_b la tasa de datos del canal y B en ancho de banda del canal.

Para salir de dudas, primeramente se realizará un programa que funcione a pequeña escala, que permitirá determinar si este tipo de algoritmos realmente permiten la corrección de errores en canales que introducen ruido e ISI y si el algoritmo propuesto en este trabajo converge y puede ser llevado a esquemas de mayor tamaño.

Este análisis de factibilidad se llevó a cabo realizando un programa que permite enviar secuencias de palabras en lenguaje no binario de largo 10, que las son captadas por un receptor, encargado de corregir los efectos del canal y decodificar la palabra, donde es posible variar el número de iteraciones sobre el cual se produce el esquema de retroalimentación entre el ecualizador log-MAP no binario y el decodificador LDPC no binario.

Para empezar, se analizará la tasa de símbolos errados versus el nivel de ruido presente. El gráfico de este proceso se observa en la Figura 4.25:

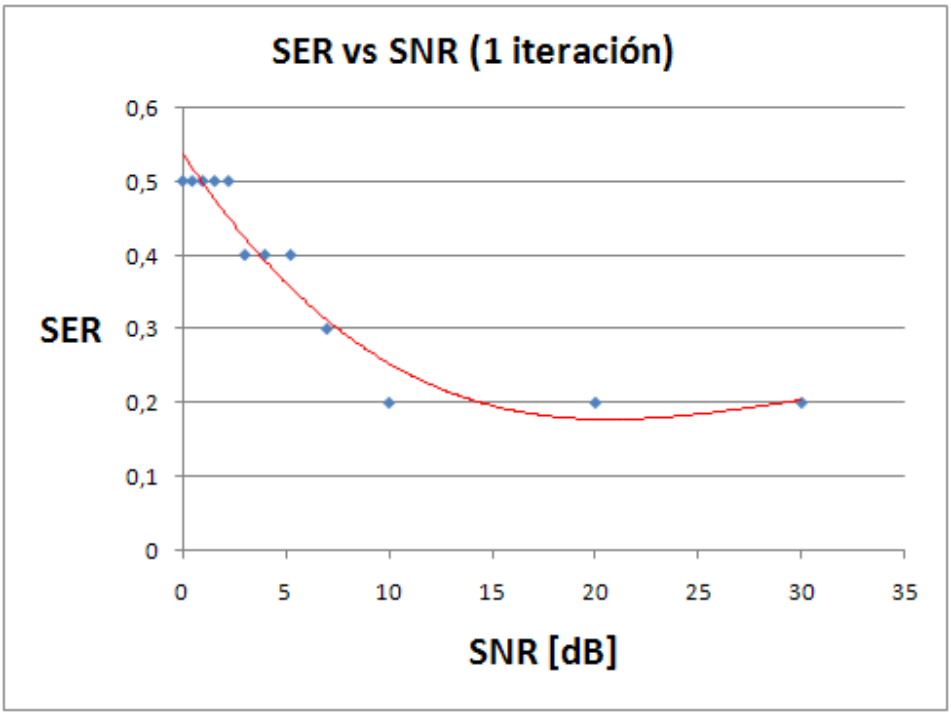


Figura 4.25. SER vs SNR (Una iteración)

Usar una iteración para realizar este proceso no es suficiente para corregir el tren de símbolos enviados por el emisor. En el límite máximo de SNR de 30 dB usado en este análisis

sostiene un nivel de error de dos símbolos por cada diez que se envían, hecho que se observa a partir de un SNR de 10 dB y se mantiene al aumentar la potencia de la señal por sobre el ruido. Para un nivel de SNR menor de 3 dB se puede observar que el nivel de símbolos errados por cada diez símbolos enviados es de 5, o sea, solamente con una iteración se puede recuperar la mitad de los símbolos enviados, incluso cuando se tiene un SNR de 0 dB, lo que demuestra un buen precedente para el estudio del algoritmo presentado.

Para el caso en que se realizan dos iteraciones, se puede observar el comportamiento del algoritmo según la Figura 4.26:

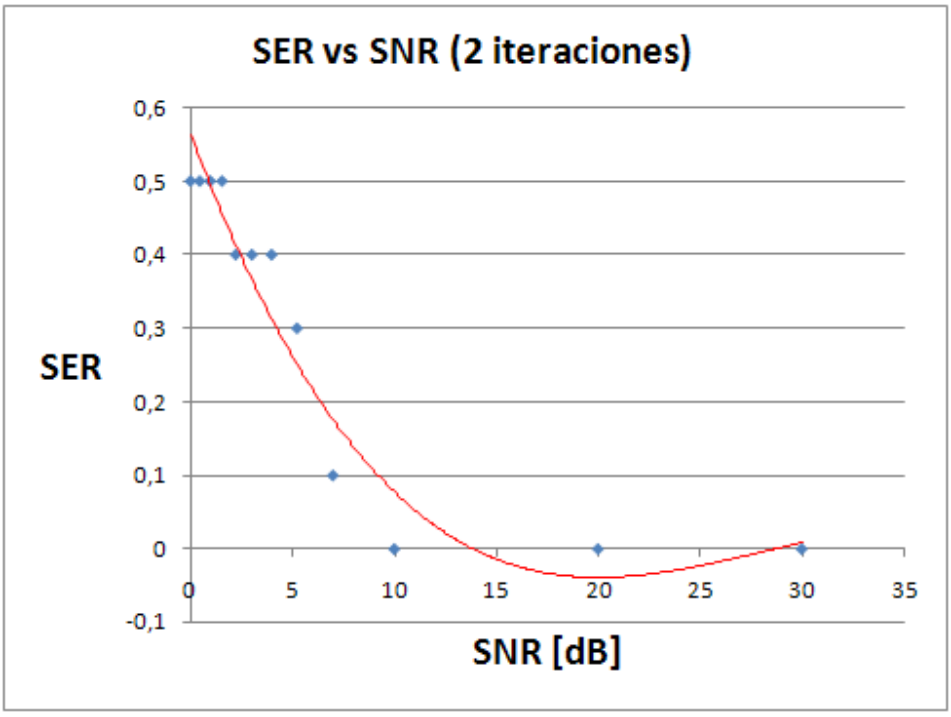


Figura 4.26. SER vs SNR (Dos iteraciones)

A diferencia del gráfico anterior, en la Figura 4.26, o sea, cuando se realizan dos iteraciones del algoritmo en vez de una, se puede ver que a partir de un nivel de SNR de 10 dB es posible realizar la corrección completa del tren de símbolos, es decir, es posible conseguir una SER de 0 (cero símbolos errados) por cada tren de 10 símbolos en GF(4). Por otra parte, en el otro extremo, es posible observar que al disminuir el SNR de 10 a 0 dB es posible ver que se pierde el poder correctivo, llegando a un tope de una tasa de cinco símbolos errados por cada tren de diez símbolos enviados, de igual manera que en el caso anterior. Con esto, se puede determinar que este algoritmo realmente converge y puede ser utilizado para corregir completamente un tren pequeño de diez símbolos y que con sólo dos iteraciones se puede conseguir esto. Ahora se analizará el efecto aumentando la cantidad de iteraciones, primeramente para observar si el algoritmo es estable y si el poder correctivo aumenta con el número de iteraciones.

Cuando se realizan tres iteraciones se puede observar un gráfico como el de la Figura 4.27:

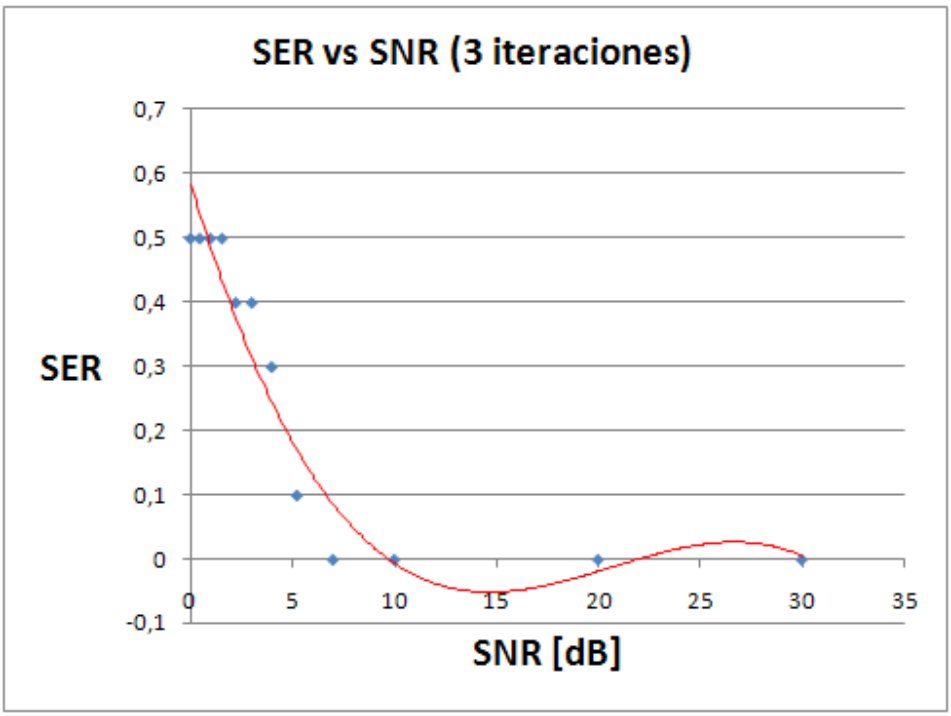


Figura 4.27. SER vs SNR (Tres iteraciones)

Para este caso se puede observar un comportamiento similar al anterior, pero con la diferencia que ahora el nivel donde el SER llega a 0, o sea corrección total, no es precisamente los 10 dB de SNR sino se produce alrededor de los 6,98 dB, es decir, se está cumpliendo el precedente que dice que este tipo de algoritmos iterativos, el número de iteraciones incrementa el poder correctivo del esquema.

Cuando el número de iteraciones se incrementa a cuatro se puede observar un comportamiento como el de la Figura 4.28:

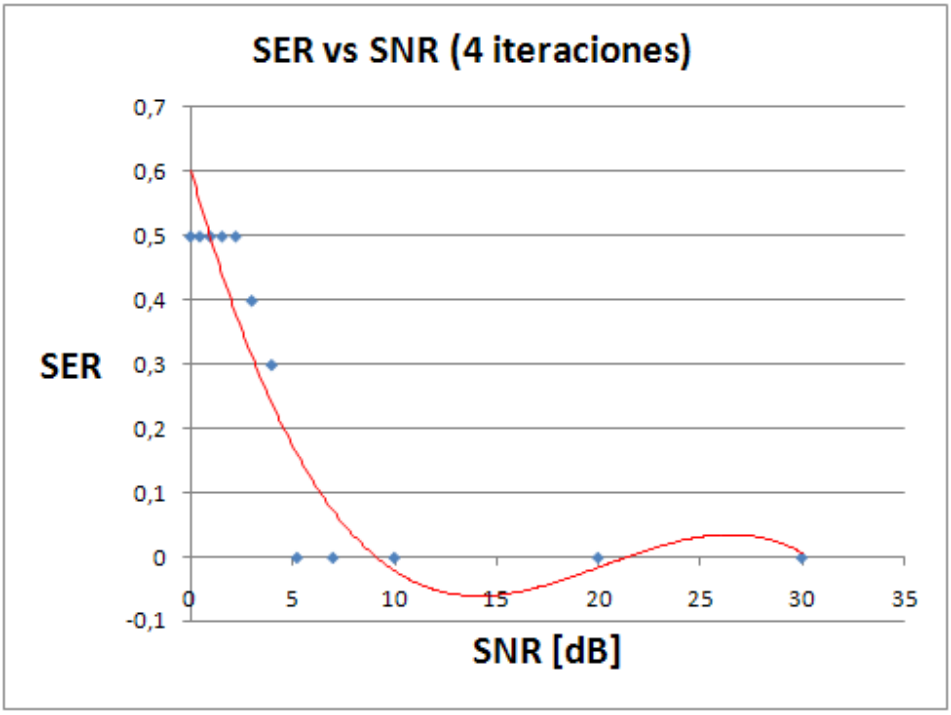


Figura 4.28. SER vs SNR (Cuatro iteraciones)

La Figura 4.28 muestra que el incremento en una iteración con respeto al gráfico anterior permite reconstruir la señal de forma total teniendo una SNR de 5,2 dB, lo cual es una mejora considerable con respecto al esquema anterior, o sea, a partir de cuatro iteraciones, se es capaz de incrementar la tasa de ruido en 1,78 dB con respecto a la potencia de la señal. Para los casos en que se incrementan las iteraciones a cinco o seis, se aprecia un comportamiento muy similar al esquema de cuatro iteraciones, por lo que se pasará directamente al estudio de cuando se utilizan siete iteraciones. El comportamiento de este esquema se puede visualizar en la Figura 4.29:

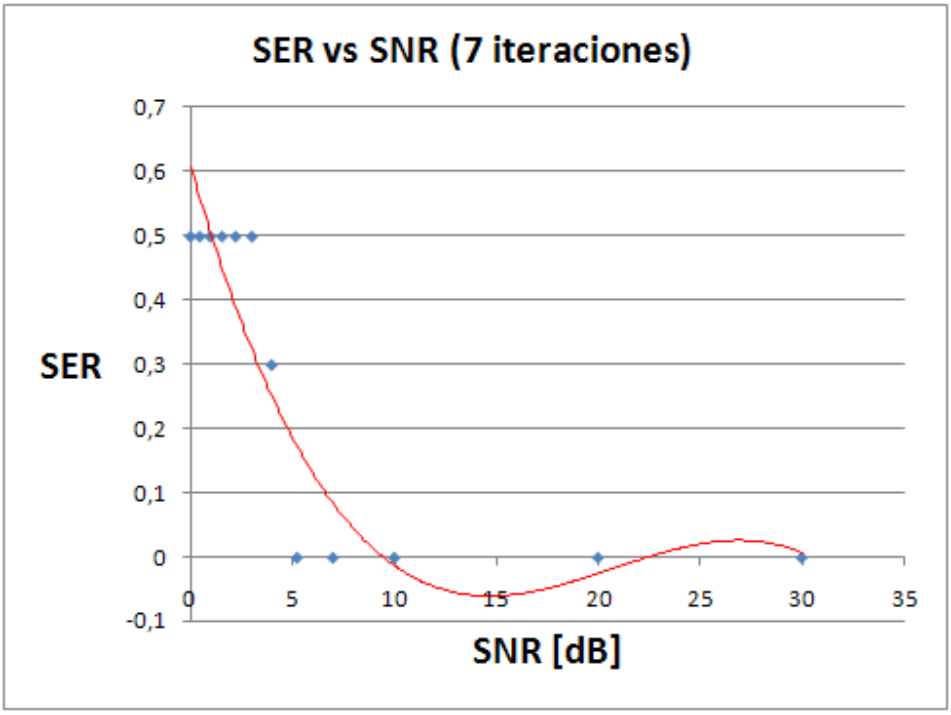


Figura 4.29. SER vs SNR (Siete Iteraciones)

A simple vista, se observa un comportamiento muy similar al anterior, pero a partir de los 5,2 dB hacia abajo se puede ver que existe un pequeño salto de la SER que pasa por los 0,3 a los 0,5, que es el límite anteriormente encontrado, sin pasar por otro valores de SER, lo que puede interpretarse como que existe un límite de SNR que tiene el esquema de 5,2 dB, ya que de ahí hacia valores menores, el esquema es incapaz de corregir completamente el tren de símbolos, sin importar el incremento en el número de iteraciones, ya que se ha probado en este estudio que un mayor número de iteraciones presenta un comportamiento idéntico al esquema de siete iteraciones. Con esto, se puede decir que un esquema conveniente sería usar entre tres o cuatro iteraciones y que el límite de SNR capaz de soportar el algoritmo llega a los 5,2 dB. Cabe recordar que el estudio se realizó usando los siguientes supuestos:

- Se utilizó un modelo de canal aproximado de dos derivaciones, similar al de Standford, pero se cambiaron algunos valores de derivaciones debido a que el modelo de cuatro símbolos tendía a confundirse al observar varios estados diferentes como equivalentes. Este supuesto es en cierta manera válido, ya que en la realidad el entorno es algo variante que difícilmente puede presentar el traslape de símbolos de manera equivalente, es por eso que una pequeña variación los pesos de las derivaciones no es un supuesto que invalide el estudio.
- Otro supuesto importante es que se tiene conocimiento total del canal en el extremo receptor, es decir, que la estimación de derivaciones del diagrama de Trellis ideal para el cálculo del algoritmo log-MAP es perfecta, y el estimador corresponde exactamente a las derivaciones del canal. Este supuesto tampoco invalida la respuesta ya que actualmente se han realizado diversos estudios acerca de estimadores de estado, los cuales se acercan a la estimación ideal (no es el caso de estudio de esta memoria). Una estimación aproximada

en vez de las derivaciones del modelo podrían aumentar muy poco el SNR límite para el cual el sistema puede decodificar de manera perfecta la palabra.

Los demás supuestos de estudio fueron mostrados en el Capítulo 2 cuando se revisó el marco teórico, estos supuestos ayudan de gran manera al planteamiento teórico del algoritmo dada la complejidad de éste.

Los resultados anteriores dicen que el algoritmo tiene un buen funcionamiento bajo un esquema de pequeña escala, donde es posible lograr la corrección completa de un tren de 10 símbolos dada una SNR razonable. Además se pudo ver que el algoritmo puede seguir iterando después de haber logrado la corrección total del tren de símbolos y sigue entregando el resultado correcto, vale decir, el algoritmo converge hacia el resultado correcto. Con este análisis, se tiene una base para comenzar a construir un programa a mayor escala, el cual se presenta en el capítulo siguiente.

4.7 Análisis detallado

El análisis detallado se refiere a la construcción de un programa que sea capaz de corregir una cantidad considerable de información, de modo de construir la curva característica del programa, es decir, la curva SNR vs SER. La construcción de este programa se basa principalmente en el mismo esquema anterior, donde existe una matriz de codificación para el código LDPC, lo que separa la señal en bloques según el tamaño de esta matriz para realizar a codificación, que posteriormente toma todos los bloques y los envía dentro de un solo tren de símbolos. Estos símbolos son enviados desde un emisor al receptor, pero en el trayecto sufre modificaciones debido a que se encuentra un canal que introduce ISI y ruido AWGN, por lo cual en el receptor se recoge una señal distorsionada, la que se separa también en bloques debido a que la matriz de codificación es fija. De este modo, la ecualización y decodificación se realiza por bloques de código y no por el código completo de una sola vez. Este proceso a su vez se repite tantas veces, como separaciones de código se hayan hecho, además de variar el SNR del canal para la corrección de cada tren de símbolos enviado. Es de vital importancia explicar que en esta primera parte se implementó el algoritmo sin usar la expresión del logaritmo Jacobiano explicado en la Sección 3.5.2, específicamente en la Tabla 3.2, por lo cual, se obtienen expresiones exactas de los términos probabilísticos involucrados en el algoritmo, a cambio de una alta carga computacional para su cálculo. Posteriormente se analizará este algoritmo usando las aproximaciones para el logaritmo Jacobiano. Entonces, a partir de esta explicación se puede comenzar el análisis.

El análisis comienza generando bloques de datos de forma aleatoria, que en este caso se utilizaron bloques de tamaño de 10^4 y 10^5 símbolos, que al ser codificados duplica el tamaño del bloque de símbolos enviado. Iniciaremos analizando el envío de estos bloques haciendo dos iteraciones en el esquema ecualizador-decodificador, donde se puede observar que los gráficos de SER vs SNR para bloques de tamaño 10^4 y 10^5 símbolos, muestran en las Figura 4.30 a) y Figura 4.30 b) respectivamente:

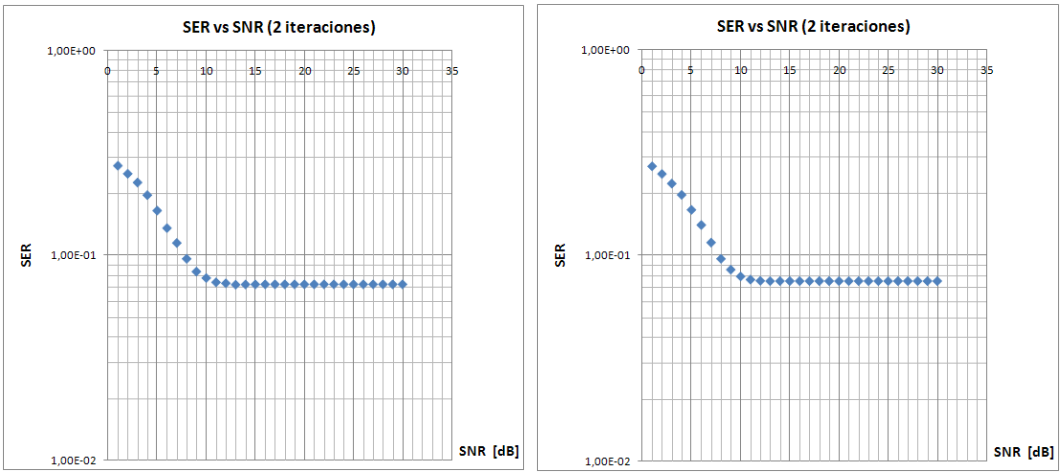


Figura 4.30. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Dos iteraciones.

En las Figura 4.30 a) y b) se puede ver que el código sigue las curvas típicas de SER vs SNR, es decir, a medida que se aumenta el SNR, aumenta el poder correctivo del código presentado. Además, se puede ver que el código presenta un piso, o sea, aunque se aumenta la tasa entre la potencia de le señal y el ruido, no se obtiene ninguna mejora sobre el desempeño del código. Otro punto que vale destacar, es que aunque se envíen bloques de datos de 10^4 o 10^5 símbolos, el piso del código es prácticamente el mismo, o equivalentemente, el esquema corrige la misma cantidad de símbolos dentro de 10000 o 100000.

Este mismo hecho se presenta en las Figura 4.31 a) y b) donde se encuentra nuevamente el piso del código y en el mismo valor que en las Figura 4.30 a) y b), es decir, alrededor de los 10 dB, sin embargo se observa que el piso del código se desplaza en el eje vertical al disminuir el SER, por lo que se tiene que el aumento de dos a cuatro iteraciones produce beneficios en el código.

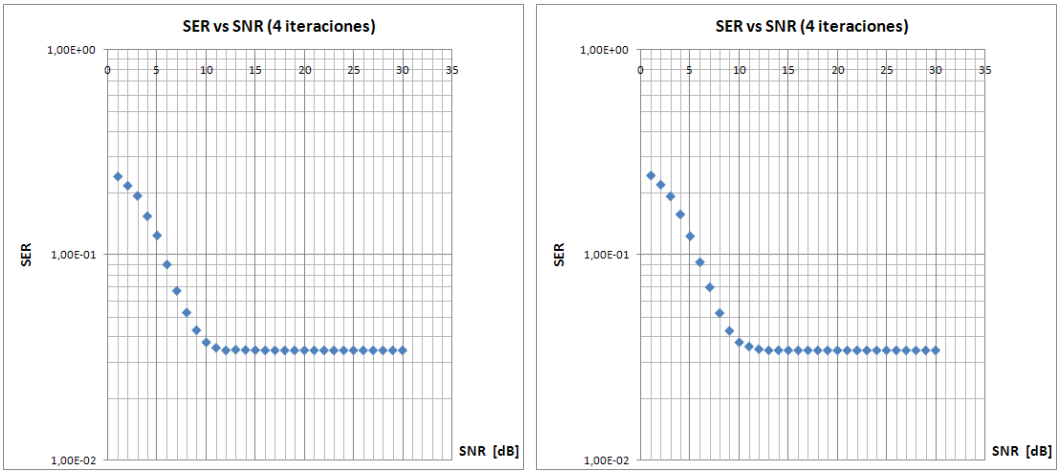


Figura 4.31. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Cuatro iteraciones.

En la Figura 4.32 se observa panorama casi idéntico al de la Figura 4.30 y la Figura 4.31, donde el piso del código se mantiene en los 10 dB, pero continúa bajando sobre el eje vertical, donde un mayor número de símbolos son corregidos conforme se aumenta el SNR, hasta llegar al piso del código. Igualmente al caso anterior, no se aprecia ninguna diferencia significativa al momento de enviar bloques de datos de mayor tamaño. Esta misma tendencia se observa en las Figura 4.33, Figura 4.34.

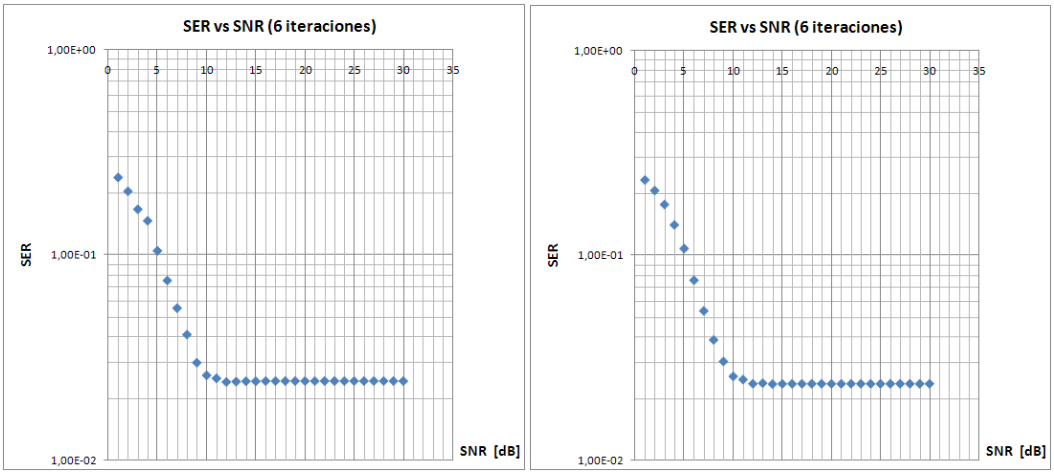


Figura 4.32. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Seis iteraciones.

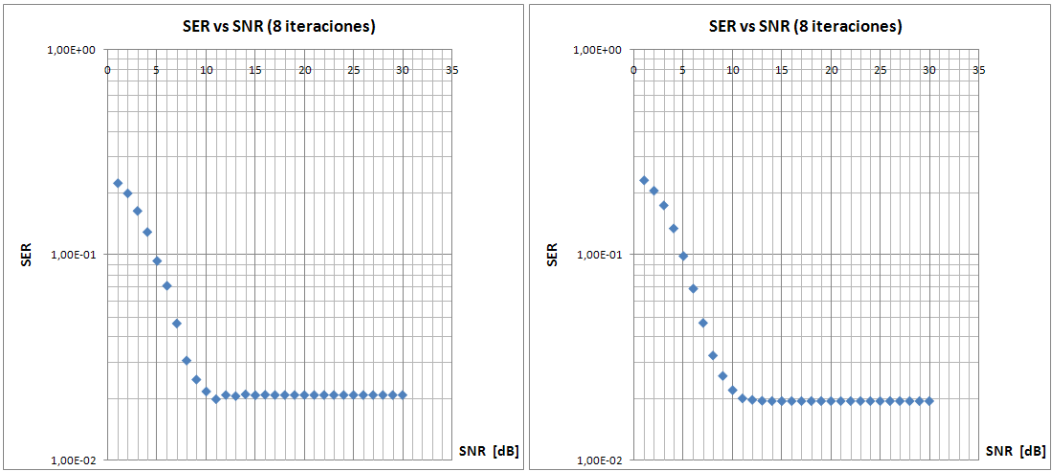


Figura 4.33. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Ocho iteraciones.

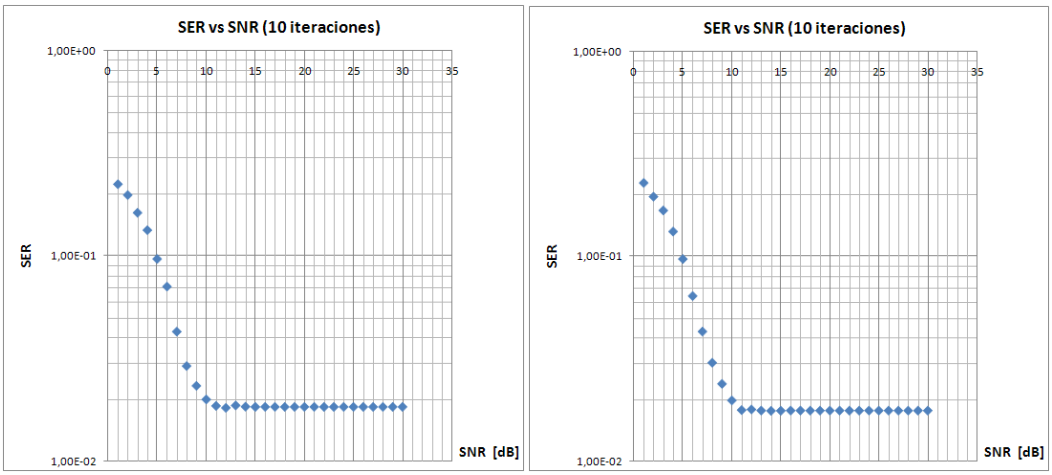


Figura 4.34. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Diez iteraciones.

En los últimos dos figuras, se puede observar que, aunque se haya aumentado el número de iteraciones de ocho a diez, el piso se desplazó hacia abajo sólo una pequeña porción en el eje vertical, lo que podría interpretarse en que se podría encontrar el piso del código en cuanto al número de iteraciones, asunto que hasta el momento no se tiene completa certeza.

Ahora se estudiará el funcionamiento y desempeño de este algoritmo usando las aproximaciones del logaritmo Jacobiano, el cual aliviana en gran manera la carga computacional de la máquina que ejecuta el algoritmo, en desmedro de obtener expresiones aproximadas de los términos probabilísticos envueltos en este algoritmo. Los resultados de esta simulación, cuando el número de iteraciones es de diez, se muestra en las Figura 4.35 a) y b):

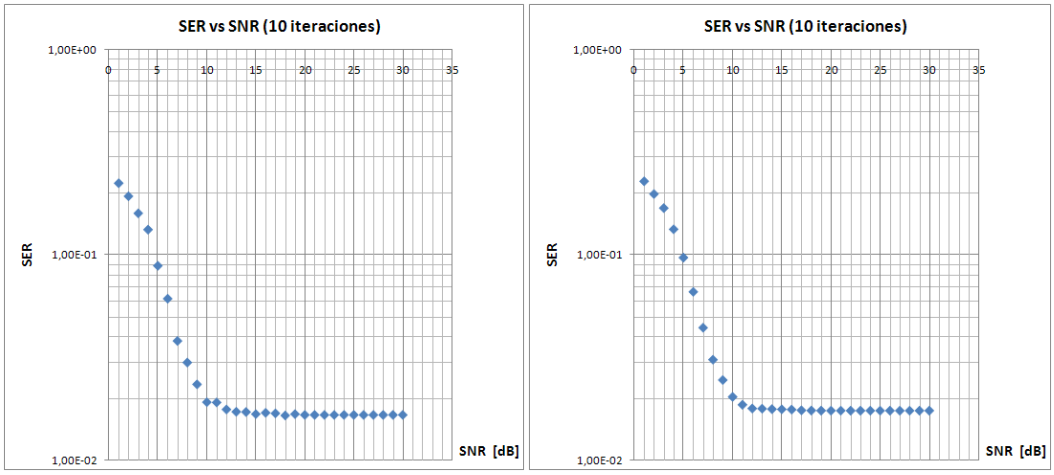


Figura 4.35. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Diez iteraciones y usando aproximación de logaritmo Jacobiano.

Contrariamente a lo que se puede haber presagiado, el comportamiento del algoritmo cuando se utilizan expresiones aproximadas, es muy similar a cuando se utilizan expresiones exactas e incluso experimenta una leve mejoría, reflejada en un pequeño desplazamiento vertical del piso del algoritmo hacia abajo. Sin embargo este piso se desplaza levemente desde los 10 dB una pequeña fracción a la derecha de las Figura 4.35 a) y b), es decir, el piso del código se ubica alrededor de los 12 dB y sobre ese nivel de SNR no se observa ninguna mejoría en el desempeño del código. Ahora se seguirá revisando el efecto del número de iteraciones en el algoritmo y se analizará hasta qué punto puede llegar a bajar el piso del código (dentro de un contexto razonable de análisis). Ahora se revisará el desempeño aumentando el número de iteraciones a veinte y siguiendo con el uso del algoritmo Jacobiano para alivianar la carga computacional del programa. El resultado de esta simulación se puede ver en las Figura 4.36 a) y b):

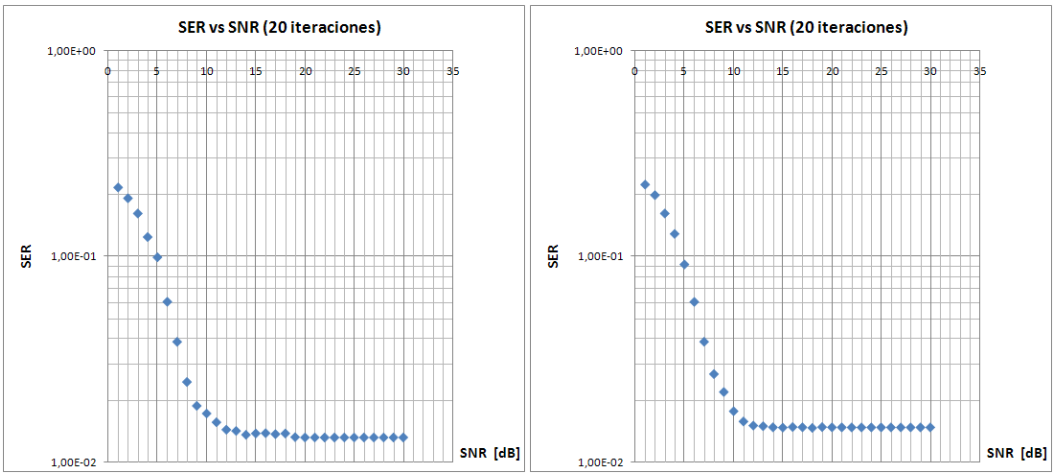


Figura 4.36. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Veinte iteraciones y usando aproximación de logaritmo Jacobiano.

Analizando las Figura 4.36 a) y b) se puede ver que el aumento del número de iteraciones de diez a veinte experimenta una pequeña mejoría en cuanto al desempeño del código, donde el piso del código se ha desplazado levemente en el eje de SER, que numéricamente hablando se ha desplazado desde un SER de $1{,}7\text{x}10^{-2}$ a un SER de $1{,}48\text{x}10^{-2}$ para el caso en que se envía una palabra de tamaño 10^5, es decir se ha experimentado un pequeño mejoramiento, pero no es muy significativo. En caso de incrementar a cuarenta el número de iteraciones, se puede ver el comportamiento mostrado en las Figura 4.37 a) y b):

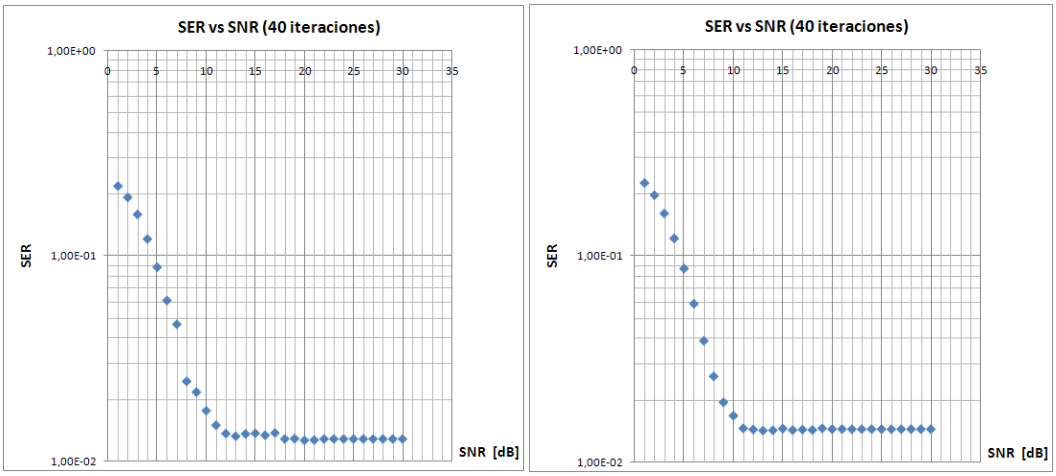


Figura 4.37. a) SER vs SNR, bloque de 10^4 datos. b) SER vs SNR, bloque de 10^5 datos. Cuarenta iteraciones y usando aproximación de logaritmo Jacobiano.

El comportamiento ya se repite de forma gradual, donde un gran aumento en el número de iteraciones, entrega una leve mejoría en el desempeño del código, que esta vez pasa de tener un SER de $1{,}48x10^{-2}$ a obtener finalmente un SER de alrededor de $1{,}3x10^{-2}$. Con eso se puede decir que el algoritmo se encuentra llegando a un límite en donde un aumento de iteraciones no entrega un resultado muy distinto, en cambio, el tiempo de ejecución del programa aumenta en forma importante, considerando la gran cantidad de cálculos que se deben hacer por cada iteración.

Ahora, ¿qué pasaría si al esquema mostrado se le eliminara el bloque de ecualizador no binario? ¿Es posible para el decodificador LDPC no binario sea capaz de soportar y corregir los efectos de un canal que introduce ISI? Para investigar esto se eliminó el bloque ecualizador del esquema presentado y se dejó solamente el LDPC no binario, de modo de probar si es capaz de corregir señales que han pasado por un canal que introduce ISI y ruido AWGN, por lo que la señal recibida pasa directamente al decodificador (obviamente pasando antes por un deinterleaver). Este experimento se realizó bajo un esquema pequeño como el que se presentó en el capítulo 4.5. Se tomó una señal pequeña de diez símbolos y se introdujo al decodificador, analizando los resultados en cada una de sus iteraciones. Primeramente se observó que el decodificador alcanzaba a corregir sólo unos pocos símbolos, donde se recibían específicamente entre siete u ocho símbolos errados, que al subir el número de iteraciones entregaba la corrección de uno o dos símbolos. Además, se realizó un análisis de desempeño del LDPC no binario (sin ecualizador), pero enviando señales en un canal que solamente introduce AWGN (sin ISI), también a pequeña escala, obteniendo un comportamiento mejor al esquema anterior, entregando corrección completa de la palabra a pequeña escala.

Posteriormente se hizo este mismo experimento, pero a gran escala, enviando señales de largo 10^5 símbolos. Los resultados de estas simulaciones se pueden ver en la Figura 4.38:

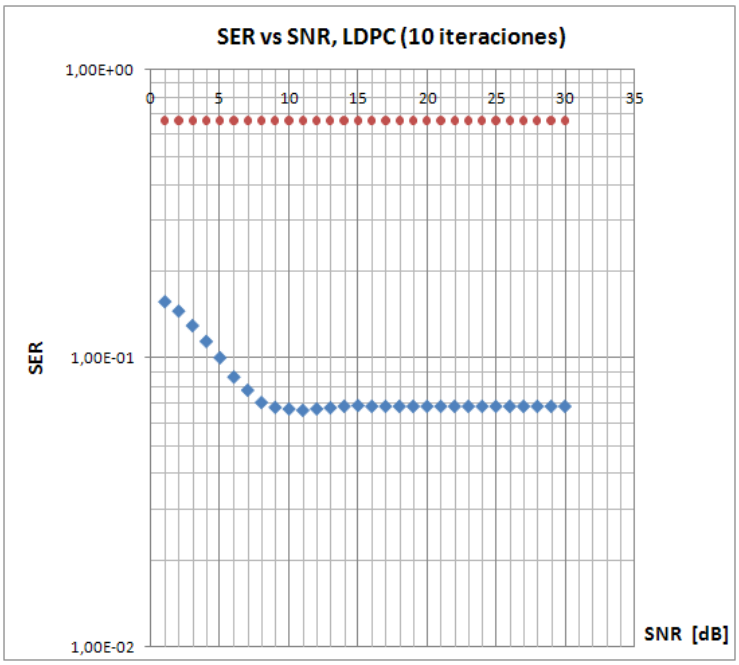


Figura 4.38. a) SER vs SNR, bloque de 10^5, Cuarenta iteraciones. Rojo: Desempeño del LDPC trabajando sin ecualizador y con canal AWGN + ISI. Azul: Desempeño del LDPC trabajando sin ecualizador y con canal AWGN.

Como era de esperarse, la curva en rojo indica que el decodificador LDPC no cuenta con protección de señales para cuando se les introduce ISI, por lo que el ecualizador conforma una pieza fundamental en el algoritmo. Lo visto en las Figura 4.38 muestra que el LDPC es capaz de corregir la palabra hasta cierto punto y a partir de cierto número de iteraciones no consigue mejorar la corrección, independiente del nivel de SNR que existan entre la señal enviada y el ruido del canal, ya que el problema de la introducción de ISI es independiente al nivel de ruido. Esto también se puede ver en la curva en azul de la Figura 4.38, donde el decodificador LDPC es capaz de corregir de mejor forma el tren de símbolos enviados.

Es importante también analizar los tiempos de ejecución del algoritmo, debido a que se necesita que exista coherencia entre las velocidades de envío de una señal o los bits por segundo (bps) y la velocidad que demora una máquina en ejecutar este algoritmo, de modo de encontrar los límites en que se pueda realizar una decodificación a "tiempo real". Primero se analizará la dependencia entre el tiempo y el número de iteraciones, para un bloque pequeño de envío de datos, fijado en diez símbolos y variando el número de iteraciones del algoritmo. Para estudiar la complejidad del modelo se realizó un análisis de tiempos de ejecución usando un computador personal[1]. Este estudio se observa en la Figura 4.39:

[1] Características principales del computador usado en las simulaciones: **Procesador:** AMD Sempron Mobile 3100+ LV (Venice-256)de 1,8 Ghz, Caché L1: 64 kB, Caché L2: 256 kB. **Memoria RAM:** 896 DDR 333 Mhz. **Placa base:** Acer Aspire 3000, Bus AMD Hammer de 64 bits, reloj efectivo de 320 Mhz.

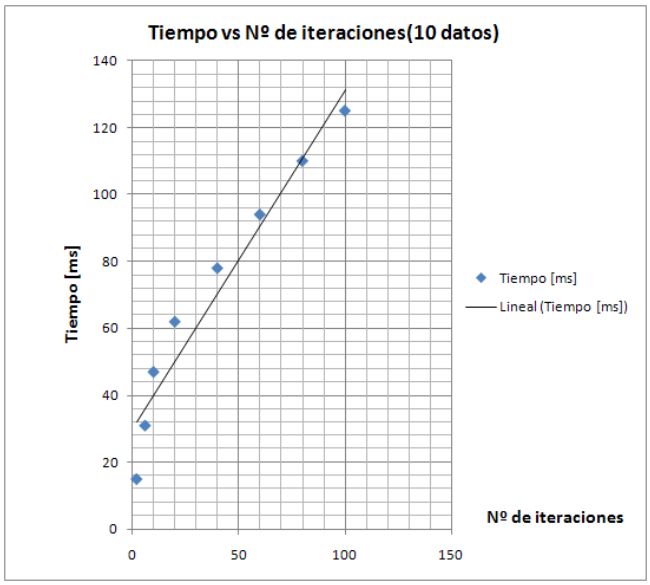


Figura 4.39. Tiempo vs Nº de iteraciones (diez datos)

Tabla 4.14. Tiempo vs número de iteraciones

Nº iteraciones	Tiempo [ms]
2	15
6	31
10	47
20	62
40	78
60	94
80	110
100	125

Como era de esperarse, el comportamiento del algoritmo muestra que si se aumenta el número de iteraciones para un cierto bloque de datos, entonces el tiempo de ejecución aumenta de forma lineal. Para ver el desempeño de estos algoritmos en torno al número de cálculos (multiplicaciones, sumas u otros cálculos) se puede encontrar este estudio en [17]. Ahora, si se mantiene el número de iteraciones fijo, pero se varía el tamaño de la trama de datos que se envía dentro de este esquema, se puede hacer un análisis de tiempos de ejecución en función del número de datos. Este análisis, dejando el número de iteraciones fijado en diez, se puede ver en la Figura 4.40:

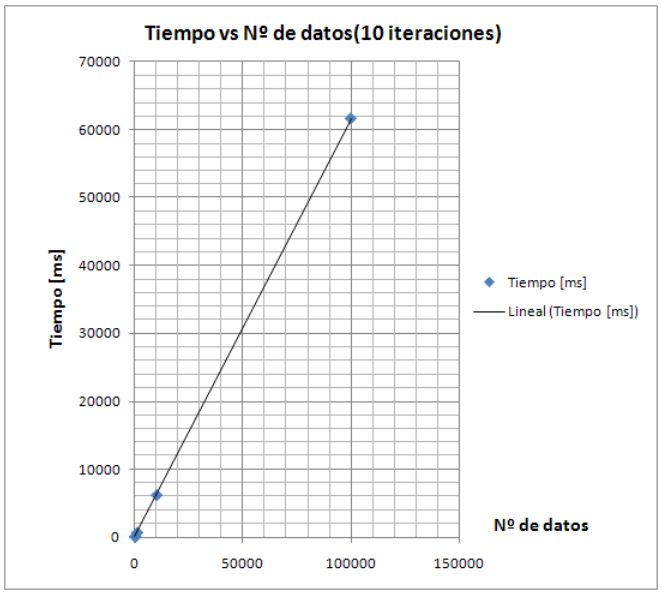


Figura 4.40. Tiempo vs Nº de datos (Diez iteraciones)

Tabla 4.15. Tiempo vs número de datos

Nº datos	Tiempo [ms]
10^1	47
10^2	110
10^3	671
10^4	6188
10^5	61547

Como era de esperarse, estos tiempos también se comportan de manera lineal, es decir, mientras aumente el tamaño de datos, se aumentará proporcionalmente el tiempo de ecualización y decodificación.

Todo este análisis permite dictaminar que no existen fenómenos del tipo "cuello de botella" que demoren la ejecución del programa más de la cuenta, ya que el aumento de tanto las iteraciones del esquema, como de la cantidad de información que se envía, producen un incremento lineal del tiempo de ejecución.

Finalmente, es posible mejorar de gran manera estos tiempos de ejecución optimizando las rutinas de programación, las rutinas de cálculo, desarrollando plataformas de hardware y software especializadas para la realización de estos cálculos, entre otras soluciones que quedan a la opinión del lector. Queda propuesto para un trabajo futuro la optimización de este proceso, que es el mayor impedimento para la utilización de este algoritmo en plataformas actuales.

Capítulo 5

Conclusiones y desafíos futuros.

5.1 Conclusiones

Considerando los resultados derivados de la propuesta teórica y la implementación bajo un ambiente simulado, es posible decir que el planteamiento de esta memoria cumple con los primeros objetivos trazados al momento de comenzar este trabajo, consistentes en analizar dos métodos existentes en el campo de telecomunicaciones para la corrección de señales afectas a un canal que introduce ruido blanco Gaussiano e interferencia intersímbolo. Estos métodos son el algoritmo log-MAP para ecualizar señales y el algoritmo LDPC para codificar y decodificar las señales. Para ambos algoritmos se trabajó para realizar su extensión a lenguajes no binarios para finalmente diseñar un método y modificar los algoritmos existentes para que permitir su funcionamiento en forma concatenada. El planteamiento teórico fue realizado a partir de la base de los algoritmos mencionados, utilizando teoría Bayesiana y extendiendo las ideas planteadas en publicaciones a lenguajes con mayor número de símbolos. Con ello, se llegó a un modelo a pequeña escala, que fue primeramente implementado bajo una plataforma, con señales determinísticas. Posteriormente se implementó este modelo en una plataforma que permitiera el ingreso de señales de tipo aleatorias, pero de tamaño fijo, que permitiera realizar distintos tipos de pruebas, de las cuáles, principalmente importaba estudiar la convergencia de este algoritmo. Esta etapa fue llamada "Análisis de factibilidad" y arrojó buenos resultados, dado que el algoritmo llegó a la convergencia el 100% de los casos, entregando la mayoría de los casos la corrección completa de una palabra de diez símbolos, que fue afectada por los efectos del canal. Además se estudió el comportamiento del algoritmo cuando se modifica el número de iteraciones en este algoritmo, entregando resultados acorde a los modelos iterativos; al incrementar el número de iteraciones, mejora el poder correctivo del algoritmo. Con esto, se puede decir que el modelo planteado en esta primera etapa cumple con los objetivos propuestos.

Además, según este planteamiento, es posible extender a lenguajes no binarios mucho más grandes como $GF(2^n)$, ya que la formulación teórica presentada en este trabajo así lo permite. No obstante, la extensión de la cantidad de símbolos en el campo de Galois incrementa la complejidad del algoritmo siguiendo la misma tasa, y lo que pudiera significar un mejor aprovechamiento del canal al tener mayor cantidad de símbolos en una transmisión, se ve truncado porque la complejidad del algoritmo frena la tasa de transferencia a tiempo real que idealmente se debiera alcanzar. La mayor cantidad de cálculos que debiera realizar tanto la estación base al codificar, como el receptor al decodificar y ecualizar, es el punto donde se produce un *trade-off*. Es por ello que es indispensable establecer un método de cálculo que permita reducir la complejidad de estos algoritmos.

La segunda etapa de análisis comienza en construir un modelo a mayor escala (basado en el modelo anterior, de pequeña escala) que permitiera enviar una cantidad de datos significativa, y

que sometiera el algoritmo propuesto a condiciones que lo llevaran al límite. Es así como se realizaron varias simulaciones variando distintos parámetros del modelo, de modo de obtener las curvas características del código. En el total de simulaciones se llegó a que el código implementado presentaba un piso, es decir, que aunque se incremente el SNR entre la señal enviada y el canal, no se aprecia mayor cantidad de símbolos corregidos, entonces el SER se mantiene constante. Además, un incremento en el número de iteraciones produce un pequeño desplazamiento en el SER, disminuyendo la tasa de símbolos errados, sobre la palabra enviada. Este desplazamiento es pequeño, teniendo un límite inferior del SER de alrededor de 10^{-2}. Por otra parte, el límite superior del SNR para este piso se encuentra fijo en los 10 dB, independiente del número de iteraciones.

Además, el LDPC no binario trabajando sin ecualizador, no presenta un esquema de protección en canales donde se introduce ISI, corroborando esto haciendo trabajar el LDPC no binario bajo un canal que introduce solamente ruido blanco aditivo gaussiano. Este último esquema también presenta un piso, el cual se centra alrededor de los $6x10^{-1}$, contrastando con el del esquema LDPC + ecualizador, fijado en un piso de alrededor de los 10^{-2}. Esto quiere decir que los esquemas se complementan.

Adicionalmente, se puede decir que el piso del código viene entregado por el código LDPC usado, que utiliza una matriz de codificación de dimensión 5x10, tomando una palabra sin codificar de largo cinco y entregando una palabra codificada de largo diez. Esto quiere decir que para enviar palabras de mayor longitud, es necesario dividir la palabra enviada en bloques de cinco símbolos, enviarla en bloques de diez símbolos y aplicar el algoritmo para cada uno de estos bloques. Es por ello que se pierde la correlación entre las palabras que son separadas, haciendo que el esquema introduzca errores en los inicios y finales de los bloques de las palabras recortadas. Es por eso que para mejorar el desempeño de este esquema es necesario construir matrices de dimensiones mucho mayores, que puedan codificar una mayor cantidad de símbolos.

Finalmente se realizó un análisis de tiempos de ejecución del programa, también modificando el número de iteraciones y el largo de las palabras enviadas, encontrando que estos tiempos son algo extensos y se deberían mejorar, para que se permitiese el envío y corrección de señales a una mayor velocidad.

5.2 Desafíos futuros

Según los resultados obtenidos, para futuras investigaciones que quisieran continuar con este estudio, es posible determinar cuáles son los puntos en que se debe mejorar en primera instancia este trabajo de título, de modo de tener una primera visión sobre los tópicos en que se debe poner hincapié. Estos puntos son los siguientes:

- Optimizar el cálculo matemático del algoritmo de modo de reducir los tiempos de ejecución y junto con ello, dar pié a que este algoritmo pueda alcanzar altas tasas de transferencia, ya que una de las principales limitantes encontradas en este trabajo es el tiempo de ejecución del programa.
- Extender este trabajo a esquemas de modulación con mayor cantidad de símbolos de modo de optimizar de mayor manera el ancho de banda, pero como se dijo anteriormente, el incremento de símbolos en este esquema produce un aumento en la complejidad del

algoritmo, sobretodo en lo que respecta a la parte de ecualización, ya que los diagramas de Trellis que se deben analizar crecen en tamaño y la cantidad de caminos a seguir que debe discriminar el ecualizador también aumentan. Este punto va de la mano con el anterior, ya que al optimizar el cálculo del algoritmo se produce una ganancia en los tiempos de ejecución. Una solución que podría ser estudiada investigaciones posteriores podría ser la posibilidad de colocar bits de cabeza, cola o bits intermedios que posibiliten al ecualizador acotar la cantidad de caminos que podría seguir el diagrama de Trellis de la palabra de arribo. Esta solución se contrapone eso si al hecho de optimizar el ancho de banda, ya que se enviarían bits que no corresponden precisamente a información. Se debe hacer un estudio para cuantificar la ganancia al implementar esta solución.

- Realizar el mismo estudio de esta memoria pero modificando algunos bloques y medir resultados. Como se dijo anteriormente, la implementación del algoritmo se hizo a pequeña escala, lo que quiere decir que si se quieren enviar grandes bloques de información, se debe descomponer la palabra en bloques de diez símbolos, para con ello decodificar cada bloque por separado. Esta podría ser una de las principales razones por las cuales el código presenta un piso, ya que al separar las palabras se pierde la "correlación" entre los símbolos que terminan en una palabra y continúan la siguiente. Lo que se propone es implementar métodos que permitan trabajar con matrices generadoras de LDPC de gran cantidad de filas y columnas, además de poder trabajar con bloques de información variable y que permitan crear las matrices de codificación y decodificación según el tamaño de bloques de símbolos de arribo. Actualmente existen estudios al respecto que involucran distintos métodos de creación de matrices LDPC. Entre éstos destaca un estudio muy reciente, tanto que se agregó poco antes de entregar este trabajo, por lo cual no fue incluido en este estudio pero se deja el puerta abierta para quienes deseen seguir investigando este tema. Este estudio se encuentra en [18] que presenta principalmente varios métodos de construcción de códigos LDPC no binarios de alto desempeño y eficiencia, basado en campos finitos. Todos estos códigos son simulados en canales con AWGN, por lo que su comportamiento en canales con ISI es incierto. Esto se podría implementar usando el algoritmo propuesto en este trabajo y extenderlo a lenguajes con mayor cantidad de símbolos.

- Implementar este algoritmo en plataformas reales celulares (estación base y receptor) para verificar su funcionamiento en un ambiente real y no simulado para así comprar resultados con respecto a los presentados en este trabajo.

Referencias

[1] Berrou, C. Glavieux, A. Thitimajshima, P. "Near Shannon limit error-correcting coding and decoding:Turbo-codes" Ecole Nat. Superieure des Telecommun. de Bretagne; Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference. Publication Date: 23-26 May 1993 Volume: 2, Pages: 1064-1070 vol.2.

http://ieeexplore.ieee.org/iel2/3171/8993/00397441.pdf?tp=&isnumber=&arnumber=397441

[2] R.G. Gallagher, "Low-density parity-check codes" IRE Trans. Inform. Theory, Volume. IT-8, Pages: 21-28, Jan 1962.

[3] Andrea Goldsmith "Wireless Communications" Published by Standford University, 2003. Pages: 147 – 181.

[4] "Wireless Communications Principles and Practice" Editorial Prentice Hall 2002. Pages: 57 – 248.

[5] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low-density parity-check codes," Electronics Letters, Volume 32, Issue 18, 29 Aug 1996 Page(s): 1645 - Digital Object Identifier
http://ieeexplore.ieee.org/iel1/2220/11500/00533358.pdf?tp=&isnumber=11500&arnumber=533358

[6] Matthew C. Davey "Error-correction using Low-Density Parity-Check Codes" A disertation submited in candidature for the degree of Doctor of Philosophy, University of Cambridge. Dec 1999.
http://www.inference.phy.cam.ac.uk/mcdavey/papers/davey_phd.ps

[7] Dai Kimura, Frédéric Guilloud and Ramesh Pyndiah "Application of Non-binary LDPC codes for Small Packet Transmission in Vehicle Communications" The 5th International Conference on ITS Telecommunications, Jun 27-29. Brest, France 2005, Pages: 109-112.
http://public.enst-bretagne.fr/~fguillou/publi/Kimura-ITST05.pdf

[8] Feng Guo, and Lajos Hanzo, "A Purely Symbol-Based Non-Binary LDPC-Aided, Non-Binary Modulated MIMO Scheme", manuscript for IEEE communications letters.

[9] Feng Guo "Low Density Parity Check Code Study" Mini thesis of requirement for the award of doctor of Philosophy University of Southampton, Jun 2002.

[10] Deepak Gilra "A class of Non-Binary Low Density Parity Check Codes" Thesis for the degree of Master of Science, University of Texas. Mayo 2003.
http://handle.tamu.edu/1969.1/67

[11] Jorge Castiñeira Moreira, Patrick Guy Farrell, "Essentials of Error-Control Coding" John Wiley & Sons, Ltd, 2006. Pages: 41-55, 157-324.

[12] Fu-hua Huang "Evaluation of Soft Output Decoding for Turbo Codes" Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of science in Electrical Engineering, May 29, 1997.
http://scholar.lib.vt.edu/theses/available/etd-71897-15815/unrestricted/etd.pdf

[13] K. Fagervik, A. Larssen, "Performance And Complexity Comparison Of Low Density Parity Check Codes And Turbo Codes", Paper presented at NORDIC SIGNAL PROCESSING SYMPOSIUM, NORSIG 2003.

[14] L. Hanzo, T. H. Liew, B.L. Yeap, Libro "Turbo Coding, Turbo Equalization and Space Time Coding for Transmission over Fading Channels" Editorial Wiley & Sons, Ltd, 2002. Páginas 11-32, 105-209,317-390, 511-709.

[15] Leung-Yan-Cheong, S.; Hellman, M. "The Gaussian wire-tap Channel Information Theory", IEEE Transactions on Volume 24, Issue 4, Jul 1978 Page(s): 451 -456.
http://ieeexplore.ieee.org/iel5/18/22703/01055917.pdf?tp=&isnumber=22703&arnumber=1055917

[16] Apuntes curso "Comunicaciones móviles" Universidad de Rosario, Octubre 2002.

[17] Chatzigeorgiou, I. A. Rodrigues, M. R. D. Wassell, I. J. Carrasco, R. A. "Comparison of Convolutional and Turbo Coding for Broadband FWA Systems", Broadcasting, IEEE Transactions on Volume 53, Issue 2, June 2007 Page(s):494 – 503.
http://ieeexplore.ieee.org/iel5/11/4215101/04215104.pdf?tp=&isnumber=&arnumber=4215104

[18] Zhou, Bo Tai, Ying Y. Lan, Lan Song, Shumei Zeng, Lingqi Lin, Shu "CTH08-1: Construction of High Performance and Efficiently Encodable Nonbinary Quasi-Cyclic LDPC Codes" Global Telecommunications Conference, 2006. GLOBECOM apos;06. IEEE, Nov. 2006 Page(s):1 – 6.

http://ieeexplore.ieee.org/iel5/4150629/4150630/04150705.pdf?tp=&isnumber=4150630&arnumber=4150705

[19] Bahl, L. Cocke, J. Jelinek, F. Raviv, J. "Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)" Information Theory, IEEE Transactions on. Publication Date: Mar 1974 Volume: 20, Issue: 2 Pages: 284- 287.

[20] C. E. Shannon, "A mathematical theory of communication" Bell System Technical Journal, vol. 27, Page:. 379-423 and 623-656, Jul and Oct, 1948.

http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

[21] Hamming, Richard W., "Error Detecting and Error Correcting Codes" The Bell System Technical Journal 26, 2 (April 1950), Pages: 147–160.

http://guest.engelschall.com/~sb/hamming/

[22] M. J. E. Golay, “Notes on digital coding” Proc. IEEE, vol. 37, 1949. Pages: 657.

[23] P. Elias, “Coding for noisy channels”, IRE Conv. Record, vol. 4, 1955. Pages: 37– 47.

[24] R. C. Bose and D. K. Ray-Chaudhuri, “On a class of error correcting binary group codes”, Information and Control, vol. 3, Mar 1960. Pages: 68–79.

[25] E. R. Berlekamp, “Nonbinary BCH decoding” in IEEE Int. Symp. on Inform. Theory, (San Remo, Italy), 1967.

http://ieeexplore.ieee.org/iel5/18/22638/01054109.pdf?tp=&isnumber=&arnumber=1054109

[26] I. S. Reed and G. Solomon, “Polynomial codes over cerain finite fields” SIAM Journal on Applied Mathematics, vol. 8, 1960. Pages: 300–304.

http://www.cs.cornell.edu/Courses/cs722/2000sp/ReedSolomon.pdf

[27] G. D. Forney, “Concatenated Codes”, Cambridge, MA: MIT Press, 1966.

[28] A. J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”, IEEE Trans. Inform. Theory, vol. 13, Apr. 1967. Pages: 260–269.

http://ieeexplore.ieee.org/iel5/18/22634/01054010.pdf?tp=&isnumber=22634&arnumber=1054010

[29] V.A. Zinoviev, “Generalized cascade codes”, Problems of Inform. Trans.12, 1976. Pages: 2-9.

http://www.springerlink.com/content/y4q2g454801411m2/

[30] V. D. Goppa 1981. "Codes on algebraic curves" In Dokl. Akad. Nauk. SSSR, 259. Pages: 1289-1290.

[31] Davey, M.C.; MacKay, D. “Low-density parity check codes over GF(q)” Communications Letters, IEEE Volume 2, Issue 6, Jun 1998 Pages:165 – 167.

http://ieeexplore.ieee.org/iel4/5724/15314/00706440.pdf

[32] T. J. Richardson and R. Urbanke, “The capacity of low-density parity-check codes under message-passing decoding,” IEEE Trans. Inform. Theory, vol. 47, no. 2, Feb. 2001. Pages: 599-618.

http://ieeexplore.ieee.org/iel5/18/19638/00910577.pdf?tp=&isnumber=19638&arnumber=910577

[33] D. Burshtein, M. Krivelevich, S. Litsyn and G. Miller, “Upper bounds on the rate of LDPC codes,” IEEE Transactions on Information Theory, volume 48, no. 9, September 2002. Pages: 2437-2449.

http://www.eng.tau.ac.il/~burstyn/Publications/univ_up_bounds.pdf

[34] J. Chen and M.P.C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," IEEE Trans. Commun., vol.50, no.3, March 2002. Pages: 406–414.

http://ieeexplore.ieee.org/iel5/26/21364/00990903.pdf?tp=&isnumber=21364&arnumber=990903

[35] C. Douillard, A. Picart, P. Didier, M. Jzquel, C. Berrou and , A. Glavieux, "Iterative Correction of Intersymbol Interference : Turbo Equalization", ETT, Vol.6, N5, september/october 1995. Pages: 507-511.

[36] M. J. Gertsman and J. H. Lodge, "Symbol-by-symbol MAP demodulation of CPM and PSK signals on Rayleigh flat-fading channels" IEEE Trans. Commun., vol. 45, pp. 788–799, July 1997.

http://ieeexplore.ieee.org/iel1/26/13231/00602584.pdf?tp=&isnumber=13231&arnumber=602584

[37] C. Poulliat, M. Fossorier and D. Declercq, "Design of regular (2,dc)-LDPC codes over GF(q) using their binary images", accepted in IEEE Trans. Commun., 2008

http://perso-etis.ensea.fr/~declercq/PDF/JournalPapers/Poulliat_2007_IEEETransCommun.pdf.gz

[38] Gerhard Bauch, Houman Khorram and Joachim Hagenauer, "Iterative Equalization and Decoding in Mobile Communications Systems", VDE-Verlag. Ntg-Fachberichete, No. 145, 1997. Pages: 307-312.

http://citeseer.ist.psu.edu/118293.html

[39] Gerhard Bauch, Volker Franz, "Iterative Equalization and Decoding for the GSM-System", Vehicular Technology Conference, 1998. VTC 98. 48th IEEE. Pages: 2262-2266.

http://ieeexplore.ieee.org/iel4/5616/15057/00686160.pdf?tp=&isnumber=&arnumber=686160

[40] Dan Raphaeli and Yoram Zarai, "Combined Turbo Equalization and Turbo Decoding", IEEE Communications Letters, vol. 2, No. 4, Apr. 1998. Pages: 107-109.

http://ieeexplore.ieee.org/iel3/4234/14569/00664220.pdf?tp=&isnumber=14569&arnumber=664220

[41] Yeap, B. L., Wong, C. H. and Hanzo, L. (2002) "Wideband Burst-by-Burst Adaptive Modulation Using Reduced-Complexity In-Phase/Quadrature-Phase Turbo Equalization", Proceedings of the 4th ITG Colloquium on Source and Channel Coding, 2002.

http://eprints.ecs.soton.ac.uk/6377/1/itg-jan2002.pdf

[42] Okada Tomoya, Iwanami Yasunori "Turbo Equalization of GMSK signals using Limiter-Discriminator Detection" IEICE technical report. Information theory Vol.101, No.70, July 2001. Pages: 13-18.

http://iwanami-web.elcom.nitech.ac.jp/iwanami/ISSSE01.pdf

[43] Ramzan, N. Shuai Wan Izquierdo, E. "Error Robustness Scheme for Scalable Video Based on the Concatenation of LDPC and Turbo Codes" Queen Mary Univ. of London, London; Image Processing, 2007. ICIP 2007. IEEE International Conference. Sept. 16 2007-Oct. 19 2007 Volume: 6. Pages: volume VI - 521-VI – 524.

http://www.ieeexplore.ieee.org/iel5/4378863/4379494/04379636.pdf?tp=&isnumber=4379494&arnumber=4379636

[44] Manu Bansal and Lisimachos P. Kondi "Scalable video transmission over Rayleigh fading channels using LDPC codes" Proceedings of SPIE -- Volume 5685 Image and Video Communications and Processing 2005, Amir Said, John G. Apostolopoulos, Editors, March 2005. Pages: 390-401.

http://spiedl.aip.org/getpdf/servlet/GetPDFServlet?filetype=pdf&id=PSISDG005685000001000390000001&idtype=cvips&prog=normal

[45] Alias, M.Y.; Guo, F.; Ng, S.X.; Liew, T.H.; Hanzo, L. "LDPC and turbo coding assisted space-time block coded OFDM" Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual, Volume 4, Issue , 22-25 April 2003. Pages: 2309 - 2313 vol.4.

http://ieeexplore.ieee.org/iel5/8574/27214/01208801.pdf?tp=&isnumber=27214&arnumber=1208801

Anexos

ANEXO 1. CÁLCULO DE SALIDAS ESPERADAS DEL DIAGRAMA DE TRELLIS. PRIMERA ITERACIÓN

Trellis del canal, usando símbolos 0,1,2,3						
Camino	**Xk**	**Xk-1**	**Xk-2**	**Eº inicial**	**Eº final**	**Salida**
γ(0,0)	0	0	0	0	0	0
γ(0,4)	1	0	0	0	4	0,407
γ(0,8)	2	0	0	0	8	0,814
γ(0,12)	3	0	0	0	12	1,221
γ(1,0)	0	0	1	1	0	0,204
γ(1,4)	1	0	1	1	4	0,611
γ(1,8)	2	0	1	1	8	1,018
γ(1,12)	3	0	1	1	12	1,425
γ(2,0)	0	0	2	2	0	0,408
γ(2,4)	1	0	2	2	4	0,815
γ(2,8)	2	0	2	2	8	1,222
γ(2,12)	3	0	2	2	12	1,629
γ(3,0)	0	0	3	3	0	0,612
γ(3,4)	1	0	3	3	4	1,019
γ(3,8)	2	0	3	3	8	1,426
γ(3,12)	3	0	3	3	12	1,833
γ(4,1)	0	1	0	4	1	0,815
γ(4,5)	1	1	0	4	5	1,222
γ(4,9)	2	1	0	4	9	1,629
γ(4,13)	3	1	0	4	13	2,036
γ(5,1)	0	1	1	5	1	1,019
γ(5,5)	1	1	1	5	5	1,426
γ(5,9)	2	1	1	5	9	1,833
γ(5,13)	3	1	1	5	13	2,24
γ(6,1)	0	1	2	6	1	1,223
γ(6,5)	1	1	2	6	5	1,63
γ(6,9)	2	1	2	6	9	2,037
γ(6,13)	3	1	2	6	13	2,444
γ(7,1)	0	1	3	7	1	1,427
γ(7,5)	1	1	3	7	5	1,834
γ(7,9)	2	1	3	7	9	2,241
γ(7,13)	3	1	3	7	13	2,648
γ(8,2)	0	2	0	8	2	1,63
γ(8,6)	1	2	0	8	6	2,037
γ(8,10)	2	2	0	8	10	2,444

γ(8,14)	3	2	0	8	14	2,851
γ(9,2)	0	2	1	9	2	1,834
γ(9,6)	1	2	1	9	6	2,241
γ(9,10)	2	2	1	9	10	2,648
γ(9,14)	3	2	1	9	14	3,055
γ(10,2)	0	2	2	10	2	2,038
γ(10,6)	1	2	2	10	6	2,445
γ(10,10)	2	2	2	10	10	2,852
γ(10,14)	3	2	2	10	14	3,259
γ(11,2)	0	2	3	11	2	2,242
γ(11,6)	1	2	3	11	6	2,649
γ(11,10)	2	2	3	11	10	3,056
γ(11,14)	3	2	3	11	14	3,463
γ(12,3)	0	3	0	12	3	2,445
γ(12,7)	1	3	0	12	7	2,852
γ(12,11)	2	3	0	12	11	3,259
γ(12,15)	3	3	0	12	15	3,666
γ(13,3)	0	3	1	13	3	2,649
γ(13,7)	1	3	1	13	7	3,056
γ(13,11)	2	3	1	13	11	3,463
γ(13,15)	3	3	1	13	15	3,87
γ(14,3)	0	3	2	14	3	2,853
γ(14,7)	1	3	2	14	7	3,26
γ(14,11)	2	3	2	14	11	3,667
γ(14,15)	3	3	2	14	15	4,074
γ(15,3)	0	3	3	15	3	3,057
γ(15,7)	1	3	3	15	7	3,464
γ(15,11)	2	3	3	15	11	3,871
γ(15,15)	3	3	3	15	15	4,278

ANEXO 2. CÁLCULO DE LAS MÉTRICAS DE RAMA $\Gamma_K(S',S)$. PRIMERA ITERACIÓN

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
$\Gamma_K(0,0)$	-1,46642	-8,92052	-11,6977	-18,7553	-11,2614	-7,29201	-2,69442	-17,4526	-27,1604	-19,5979
$\Gamma_K(0,4)$	-0,4037	-5,81359	-8,09182	-14,1012	-7,72961	-4,51472	-1,13611	-12,9748	-21,4923	-14,833
$\Gamma_K(0,8)$	-0,00357	-3,36926	-5,14848	-10,1097	-4,86041	-2,40003	-0,24039	-9,15949	-16,4868	-10,7307
$\Gamma_K(0,12)$	-0,26605	-1,58752	-2,86775	-6,78074	-2,6538	-0,94793	-0,00727	-6,00681	-12,1439	-7,29104
$\Gamma_K(1,0)$	-0,85093	-7,28042	-9,80753	-16,3397	-9,40835	-5,81713	-1,83053	-15,1254	-24,2366	-17,1268
$\Gamma_K(1,4)$	-0,12032	-4,5056	-6,53371	-12,0177	-6,20866	-3,37195	-0,60433	-10,9796	-18,9006	-12,694
$\Gamma_K(1,8)$	-0,05231	-2,39338	-3,92249	-8,35829	-3,67157	-1,58937	-0,04072	-7,49645	-14,2272	-8,92383

Γ_K (1,12)	-0,64689	-0,94375	-1,97386	-5,36147	-1,79707	-0,46939	-0,13971	-4,67589	-10,2164	-5,81627
Γ_K (2,0)	-0,4019	-5,80678	-8,08377	-14,0906	-7,72175	-4,50871	-1,1331	-12,9646	-21,4792	-14,8221
Γ_K (2,4)	-0,00341	-3,36407	-5,14207	-10,1007	-4,85417	-2,39565	-0,23901	-9,15093	-16,4754	-10,7215
Γ_K (2,8)	-0,26751	-1,58396	-2,86296	-6,77337	-2,64919	-0,94518	-0,00752	-5,99988	-12,1341	-7,28341
Γ_K (2,12)	-1,19421	-0,46645	-1,24645	-4,10867	-1,10681	-0,15731	-0,43862	-3,51143	-8,45539	-4,50795
Γ_K (3,0)	-0,11934	-4,4996	-6,52648	-12,0079	-6,20162	-3,36676	-0,60213	-10,9702	-18,8883	-12,6839
Γ_K (3,4)	-0,05296	-2,38901	-3,91689	-8,35011	-3,66615	-1,58581	-0,04015	-7,48871	-14,2166	-8,91539
Γ_K (3,8)	-0,64917	-0,94101	-1,96989	-5,35492	-1,79328	-0,46745	-0,14077	-4,66977	-10,2074	-5,80945
Γ_K (3,12)	-1,90798	-0,15561	-0,68549	-3,02233	-0,58301	-0,01169	-0,90399	-2,51343	-6,86081	-3,3661
Γ_K (4,1)	-0,00341	-3,36407	-5,14207	-10,1007	-4,85417	-2,39565	-0,23901	-9,15093	-16,4754	-10,7215
Γ_K (4,5)	-0,26751	-1,58396	-2,86296	-6,77337	-2,64919	-0,94518	-0,00752	-5,99988	-12,1341	-7,28341
Γ_K (4,9)	-1,19421	-0,46645	-1,24645	-4,10867	-1,10681	-0,15731	-0,43862	-3,51143	-8,45539	-4,50795
Γ_K (4,13)	-2,7835	-0,01153	-0,29253	-2,10656	-0,22702	-0,03203	-1,53232	-1,68557	-5,4393	-2,39509
Γ_K (5,1)	-0,05296	-2,38901	-3,91689	-8,35011	-3,66615	-1,58581	-0,04015	-7,48871	-14,2166	-8,91539
Γ_K (5,5)	-0,64917	-0,94101	-1,96989	-5,35492	-1,79328	-0,46745	-0,14077	-4,66977	-10,2074	-5,80945
Γ_K (5,9)	-1,90798	-0,15561	-0,68549	-3,02233	-0,58301	-0,01169	-0,90399	-2,51343	-6,86081	-3,3661
Γ_K (5,13)	-3,82939	-0,0328	-0,06368	-1,35234	-0,03533	-0,21853	-2,3298	-1,01969	-4,17683	-1,58536
Γ_K (6,1)	-0,26897	-1,5804	-2,85818	-6,76601	-2,64459	-0,94243	-0,00776	-5,99295	-12,1242	-7,27578
Γ_K (6,5)	-1,1973	-0,46452	-1,24329	-4,10294	-1,10383	-0,15619	-0,4405	-3,50613	-8,44716	-4,50195
Γ_K (6,9)	-2,78822	-0,01123	-0,291	-2,10246	-0,22567	-0,03254	-1,53582	-1,6819	-5,4327	-2,39072
Γ_K (6,13)	-5,04174	-0,22054	-0,00131	-0,76458	-0,01011	-0,57149	-3,29375	-0,52027	-3,08083	-0,94208
Γ_K (7,1)	-0,65145	-0,93827	-1,96592	-5,34838	-1,7895	-0,46552	-0,14184	-4,66366	-10,1984	-5,80263
Γ_K (7,5)	-1,91189	-0,15449	-0,68315	-3,01742	-0,58085	-0,01139	-0,90668	-2,50895	-6,85341	-3,36092
Γ_K (7,9)	-3,83492	-0,03332	-0,06297	-1,34905	-0,0348	-0,21985	-2,33412	-1,01683	-4,17106	-1,5818
Γ_K (7,13)	-6,42055	-0,57474	-0,10539	-0,34328	-0,15135	-1,09091	-4,42416	-0,18731	-2,1513	-0,46527
Γ_K (8,2)	-1,1973	-0,46452	-1,24329	-4,10294	-1,10383	-0,15619	-0,4405	-3,50613	-8,44716	-4,50195
Γ_K (8,6)	-2,78822	-0,01123	-0,291	-2,10246	-0,22567	-0,03254	-1,53582	-1,6819	-5,4327	-2,39072
Γ_K (8,10)	-5,04174	-0,22054	-0,00131	-0,76458	-0,01011	-0,57149	-3,29375	-0,52027	-3,08083	-0,94208
Γ_K (8,14)	-7,95785	-1,09244	-0,37421	-0,08929	-0,45714	-1,77303	-5,71427	-0,02123	-1,39156	-0,15604
Γ_K (9,2)	-1,91189	-0,15449	-0,68315	-3,01742	-0,58085	-0,01139	-0,90668	-2,50895	-6,85341	-3,36092
Γ_K (9,6)	-3,83492	-0,03332	-0,06297	-1,34905	-0,0348	-0,21985	-2,33412	-1,01683	-4,17106	-1,5818
Γ_K (9,10)	-6,42055	-0,57474	-0,10539	-0,34328	-0,15135	-1,09091	-4,42416	-0,18731	-2,1513	-0,46527
Γ_K (9,14)	-9,66878	-1,77875	-0,81041	-0,00011	-0,93049	-2,62457	-7,17679	-0,02039	-0,79414	-0,01135
Γ_K (10,2)	-2,79294	-0,01093	-0,28948	-2,09836	-0,22433	-0,03305	-1,53933	-1,67823	-5,42611	-2,38635
Γ_K (10,6)	-5,04809	-0,22187	-0,00141	-0,76211	-0,01039	-0,57363	-3,29888	-0,51823	-3,07587	-0,93934
Γ_K(10,10)	-7,96583	-1,0954	-0,37594	-0,08845	-0,45905	-1,7768	-5,72103	-0,02082	-1,38823	-0,15493
Γ_K(10,14)	-11,5462	-2,63153	-1,41307	-0,07739	-1,57031	-3,64257	-8,80577	-0,18601	-0,36318	-0,03311
Γ_K (11,2)	-3,84046	-0,03383	-0,06227	-1,34577	-0,03428	-0,22118	-2,33844	-1,01398	-4,16528	-1,57824
Γ_K (11,6)	-6,42772	-0,57688	-0,10631	-0,34163	-0,15245	-1,09387	-4,43011	-0,18609	-2,14715	-0,46335
Γ_K(11,10)	-9,67757	-1,78252	-0,81295	-7,9E-05	-0,93322	-2,62915	-7,18437	-0,02079	-0,79162	-0,01105
Γ_K(11,14)	-13,59	-3,65076	-2,18219	-0,32113	-2,37659	-4,82703	-10,6012	-0,51809	-0,09869	-0,22135
Γ_K(12,3)	-5,04809	-0,22187	-0,00141	-0,76211	-0,01039	-0,57363	-3,29888	-0,51823	-3,07587	-0,93934

$\Gamma_K(12,7)$	-7,96583	-1,0954	-0,37594	-0,08845	-0,45905	-1,7768	-5,72103	-0,02082	-1,38823	-0,15493
$\Gamma_K(12,11)$	-11,5462	-2,63153	-1,41307	-0,07739	-1,57031	-3,64257	-8,80577	-0,18601	-0,36318	-0,03311
$\Gamma_K(12,15)$	-15,7891	-4,83025	-3,11279	-0,72892	-3,34416	-6,17093	-12,5531	-1,01379	-0,00073	-0,5739
$\Gamma_K(13,3)$	-6,42772	-0,57688	-0,10631	-0,34163	-0,15245	-1,09387	-4,43011	-0,18609	-2,14715	-0,46335
$\Gamma_K(13,7)$	-9,67757	-1,78252	-0,81295	-7,9E-05	-0,93322	-2,62915	-7,18437	-0,02079	-0,79162	-0,01105
$\Gamma_K(13,11)$	-13,59	-3,65076	-2,18219	-0,32113	-2,37659	-4,82703	-10,6012	-0,51809	-0,09869	-0,22135
$\Gamma_K(13,15)$	-18,1651	-6,1816	-4,21403	-1,30477	-4,48255	-7,68751	-14,6807	-1,67799	-0,06835	-1,09424
$\Gamma_K(14,3)$	-7,97381	-1,09836	-0,37768	-0,08761	-0,46097	-1,78057	-5,7278	-0,02042	-1,3849	-0,15382
$\Gamma_K(14,7)$	-11,5558	-2,63612	-1,41643	-0,07817	-1,57385	-3,64797	-8,81417	-0,18723	-0,36148	-0,03363
$\Gamma_K(14,11)$	-15,8003	-4,83647	-3,11778	-0,73134	-3,34933	-6,17796	-12,5631	-1,01664	-0,00066	-0,57604
$\Gamma_K(14,15)$	-20,7075	-7,69941	-5,48173	-2,04709	-5,78741	-9,37055	-16,9747	-2,50865	-0,30243	-1,78104
$\Gamma_K(15,3)$	-9,68637	-1,7863	-0,81551	-5,6E-05	-0,93595	-2,63374	-7,19195	-0,0212	-0,78911	-0,01075
$\Gamma_K(15,7)$	-13,6005	-3,65617	-2,18637	-0,32273	-2,38095	-4,83325	-10,6104	-0,52013	-0,0978	-0,22268
$\Gamma_K(15,11)$	-18,1771	-6,18863	-4,21983	-1,30801	-4,48854	-7,69535	-14,6915	-1,68166	-0,06909	-1,0972
$\Gamma_K(15,15)$	-23,4164	-9,38369	-6,91589	-2,95588	-7,25873	-11,22	-19,4352	-3,50578	-0,70298	-2,63432

ANEXO 3. CÁLCULO DE LOS VALORES "HACIA DELANTE" $A_K(S)$. PRIMERA ITERACIÓN

Ak(s)	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Ak(0)	0	0,79495	-5,4687	-5,8413	-9,9169	-4,1651	-0,4289	2,65237	-9,3466	-15,82	-9,5435
Ak(1)	0	1,17263	-1,2061	-1,1194	-3,5972	0,67138	2,44504	3,84131	-3,185	-6,6269	-1,6386
Ak(2)	0	-0,6301	0,91559	1,46642	0,20651	2,73575	3,22994	3,11673	1,04318	-0,4962	3,05852
Ak(3)	0	-4,7743	0,36915	2,02134	2,00065	2,6235	2,0263	0,22079	3,05862	3,13735	4,8108
Ak(4)	0	1,25262	-2,7888	-3,1274	-6,2327	-1,6158	1,45748	3,51755	-5,5417	-11,144	-5,7737
Ak(5)	0	0,55249	0,3741	0,4583	-1,1463	1,94847	3,2	3,63824	-0,4535	-3,2582	0,81034
Ak(6)	0	-2,395	1,31396	1,93323	1,54317	2,85276	2,90162	1,82855	2,68616	1,63895	4,21288
Ak(7)	0	-7,7738	-0,5145	1,38825	2,26969	1,66561	0,59595	-2,2514	3,50913	4,14548	4,75976
Ak(8)	0	1,17348	-0,7262	-1,0488	-3,2016	0,27616	2,71009	3,76727	-2,3539	-7,1295	-2,6661
Ak(9)	0	-0,6265	1,3278	1,40923	0,67397	2,58457	3,3369	2,82133	1,66427	-0,5436	2,59945
Ak(10)	0	-4,7677	1,06368	1,77907	2,26478	2,35175	1,95359	-0,0853	3,70245	3,14134	4,71815
Ak(11)	0	-11,411	-2,058	0,13066	1,9238	0,09417	-1,46	-5,3733	3,30925	4,53046	4,08213
Ak(12)	0	0,55469	0,71838	0,39807	-0,8204	1,51257	3,33328	3,40281	0,21823	-3,7754	-0,2204
Ak(13)	0	-2,3899	1,65139	1,73411	1,86799	2,58461	2,85639	1,38945	3,16755	1,51982	3,72978
Ak(14)	0	-7,7656	0,1615	1,00427	2,37235	1,23489	0,38422	-2,6292	4,08729	4,01462	4,57812
Ak(15)	0	-15,693	-4,262	-1,7531	0,96166	-2,0919	-4,1442	-9,1479	2,45599	4,29353	2,78079

ANEXO 4. CÁLCULO DE LOS VALORES "HACIA DELANTE" $B_K(S)$. PRIMERA ITERACIÓN.

Bk(s)	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Bk(0)	4,88676	1,64142	0,61669	-4,2679	-0,8373	-1,6885	2,2924	-4,555	-11,116	-7,259	0
Bk(1)	4,87984	2,42114	1,52733	-2,8265	0,09912	-0,7128	2,36105	-3,2212	-9,1859	-5,7715	0
Bk(2)	4,75552	3,06804	2,27775	-1,5434	0,89227	0,14468	2,29874	-2,0526	-7,4214	-4,4456	0
Bk(3)	4,51449	3,58594	2,87011	-0,4161	1,54624	0,88362	2,10623	-1,049	-5,8221	-3,2791	0
Bk(4)	4,38284	3,76845	3,36381	0,21482	1,90162	1,15532	1,93725	-0,0643	-4,3935	-2,2741	0
Bk(5)	3,97257	4,09693	3,67337	1,07274	2,35016	1,71523	1,54964	0,6195	-3,1184	-1,4166	0
Bk(6)	3,44332	4,30481	3,8409	1,7932	2,66823	2,15004	1,03473	1,14339	-2,005	-0,7069	0
Bk(7)	2,79331	4,39216	3,87136	2,38093	2,85625	2,4565	0,39239	1,50961	-1,0517	-0,1378	0
Bk(8)	2,43582	4,36206	3,90034	2,60484	2,95874	2,42823	0,00344	1,82417	-0,2964	0,29861	0
Bk(9)	1,58347	4,26988	3,72477	2,99848	2,95338	2,49842	-0,8335	1,91443	0,35818	0,61896	0
Bk(10)	0,59459	4,05324	3,42829	3,2722	2,8229	2,4205	-1,8031	1,86318	0,8651	0,83584	0
Bk(11)	-0,5358	3,71082	3,0125	3,42725	2,56842	2,19047	-2,9059	1,67429	1,22901	0,96545	0
Bk(12)	-1,3464	3,45514	2,80397	3,44089	2,4549	1,58558	-3,5557	1,55985	1,44434	1,02177	0
Bk(13)	-2,7577	2,91856	2,21622	3,42653	2,01513	1,05059	-4,8616	1,1606	1,57456	1,01321	0
Bk(14)	-4,327	2,24946	1,50971	3,29365	1,44903	0,3532	-6,3028	0,63274	1,58165	0,93803	0
Bk(15)	-6,0565	1,44407	0,68311	3,04113	0,75535	-0,5077	-7,8821	-0,0233	1,46897	0,7863	0

ANEXO 5. CÁLCULO DE LOGARITMO NATURAL DE PROBABILIDADES A POSTERIORI Y DECISIONES DEL ECUALIZADOR. PRIMERA ITERACIÓN

Ln( )	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Pbb(0)	4,0869	3,937128	1,779084	3,630611	3,953856	6,093222	1,675882	-2,73683	-0,1336	4,972036
Pbb(1)	5,5717	5,564518	4,494822	5,477005	5,52208	5,294652	4,806153	2,61588	4,052988	5,22854
Pbb(2)	5,6776	5,606626	5,628096	5,668951	5,717711	3,349254	5,961039	5,28555	5,697558	5,219048
Pbb(3)	4,0402	4,466354	5,696841	4,653367	4,174923	-0,06407	5,049322	6,131937	5,752417	5,049437
Decisión	2	2	3	2	2	0	2	3	3	1
Correcto	2	1	3	1	2	0	2	3	2	2
Error?	correcto	error	correcto	error	correcto	correcto	correcto	correcto	error	error
Antes EQ	1	2	3	3	3	2	2	3	3	3
Error pre EQ	error	error	correcto	error	error	error	correcto	correcto	error	error

ANEXO 6. PREPARACIÓN DE LA INFORMACIÓN PARA QUE SEA ENVIADA A DECODIFICADOR LDPC.

Aplicación deinterleaver										
	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Pbb(0)	3,9539	6,093222	3,937128	-0,1336	4,086922	4,972036	-2,73683	3,630611	1,675882	1,779084
Pbb(1)	5,5221	5,294652	5,564518	4,052988	5,571707	5,22854	2,61588	5,477005	4,806153	4,494822
Pbb(2)	5,7177	3,349254	5,606626	5,697558	5,677633	5,219048	5,28555	5,668951	5,961039	5,628096
Pbb(3)	4,1749	-0,06407	4,466354	5,752417	4,040202	5,049437	6,131937	4,653367	5,049322	5,696841

Cálculo de exponencial										
Pbb(0)	52,136	442,8458	51,27115	0,874942	59,5563	144,3205	0,064776	37,73588	5,343504	5,924426
Pbb(1)	250,15	199,2684	260,9993	57,56919	262,8825	186,5202	13,67925	239,1295	122,2603	89,55222
Pbb(2)	304,21	28,48149	272,2243	298,1386	292,2569	184,7582	197,4627	289,7305	388,0129	278,1321
Pbb(3)	65,035	0,937939	87,03879	314,9509	56,83781	155,9347	460,3269	104,9377	155,9168	297,9248

Trasposición				
k=1	52,136	250,1548	304,2078	65,03483
k=2	442,85	199,2684	28,48149	0,937939
k=3	51,271	260,9993	272,2243	87,03879
k=4	0,8749	57,56919	298,1386	314,9509
k=5	59,556	262,8825	292,2569	56,83781
k=6	144,32	186,5202	184,7582	155,9347
k=7	0,0648	13,67925	197,4627	460,3269
k=8	37,736	239,1295	289,7305	104,9377
k=9	5,3435	122,2603	388,0129	155,9168
k=10	5,9244	89,55222	278,1321	297,9248

Normalización				
k=1	0,077637	0,372513	0,453005	0,096845
k=2	0,659454	0,296736	0,042413	0,001397
k=3	0,076349	0,388662	0,405377	0,129612
k=4	0,001303	0,085728	0,443967	0,469002
k=5	0,088687	0,391466	0,435208	0,084639
k=6	0,214912	0,277753	0,275129	0,232207
k=7	9,65E-05	0,02037	0,294047	0,685486
k=8	0,056194	0,356095	0,431446	0,156266
k=9	0,007957	0,182061	0,577801	0,23218
k=10	0,008822	0,133355	0,414175	0,443648

ANEXO 7. PROBABILIDADES INICIALES (A PRIORI) DECODIFICADOR LDPC. PRIMERA ITERACIÓN

	0	**1**	**2**	**3**	**Sobrev**	**Correcto**	**Resultado**
0	0,07764	0,37251	0,453	0,09685	2	2	Correcto
1	0,65945	0,29674	0,04241	0,0014	0	0	Correcto
2	0,07635	0,38866	0,40538	0,12961	2	1	Error
3	0,0013	0,08573	0,44397	0,469	3	2	Error
4	0,08869	0,39147	0,43521	0,08464	2	2	Correcto
5	0,21491	0,27775	0,27513	0,23221	1	2	Error
6	9,6E-05	0,02037	0,29405	0,68549	3	3	Correcto
7	0,05619	0,35609	0,43145	0,15627	2	1	Error
8	0,00796	0,18206	0,5778	0,23218	2	2	Correcto
9	0,00882	0,13335	0,41417	0,44365	3	3	Correcto

ANEXO 8. PRIMERA ITERACIÓN CÁLCULO DE VALORES Q_{ij}^{X}

Q_{ij}^{0}

	0	**1**	**2**	**3**
1	0,65945	0,29674	0,04241	0,0014
2	0,07635	1,30E-01	0,38866	4,05E-01
3	0,0013	0,44397	4,69E-01	8,57E-02
7	0,05619	0,43145	0,15627	3,56E-01

Q_{ij}^{1}

	0	**1**	**2**	**3**
1	0,65945	0,04241	0,0014	0,29674
4	0,08869	0,43521	0,08464	3,91E-01
5	0,21491	0,27775	0,27513	2,32E-01
6	9,6E-05	0,02037	0,29405	0,68549

Q_{ij}^{2}

	0	**1**	**2**	**3**
3	0,0013	0,44397	4,69E-01	8,57E-02
8	0,00796	0,23218	0,18206	0,5778
9	8,82E-03	0,41417	0,44365	0,13335

Q_{ij}^{3}

	0	**1**	**2**	**3**
0	0,07764	0,09685	0,37251	0,453
5	0,21491	0,23221	0,27775	0,27513
7	0,05619	0,35609	0,43145	0,15627
9	8,82E-03	0,44365	0,13335	0,41417

Q_{ij}^{4}

	0	**1**	**2**	**3**
0	0,07764	0,09685	0,37251	0,453
2	0,07635	4,05E-01	0,12961	0,38866
4	0,08869	3,91E-01	0,43521	0,08464
6	9,6E-05	0,68549	0,02037	0,29405
8	0,00796	0,23218	0,18206	0,5778

ANEXO 9. CÁLCULO DE LOS VALORES $\widehat{Q_{ij}^{X}}$ (Q_{ij}^{X} EN DOMINIO DE LA FRECUENCIA). PRIMERA ITERACIÓN

$\widehat{Q_{ij}^{0}}$

0	**1**	**2**	**3**
1	0,403734041	0,912381364	0,321702256
1	-0,06997811	-0,58807736	-0,03654713
1	-0,05938937	-0,10946072	-0,82593831

1	-0,57508141	-0,02472065	-0,1754236

$\widehat{Q_{ij}^{1}}$

0	1	2	3
1	0,321702256	0,403734041	0,912381364
1	-0,65334834	0,047790409	-0,03969404
1	-0,01991882	-0,01467129	-0,10576289
1	-0,41171232	-0,95906676	0,371164916

$\widehat{Q_{ij}^{2}}$

0	1	2	3
1	-0,05938937	-0,10946072	-0,82593831
1	-0,61996294	-0,51972528	0,171516888
1	-0,09505874	-0,15400639	-0,71564593

$\widehat{Q_{ij}^{3}}$

0	1	2	3
1	-0,09969985	-0,65103497	0,061283927
1	-0,01467129	-0,10576289	-0,01991882
1	-0,02472065	-0,1754236	-0,57508141
1	-0,71564593	-0,09505874	-0,15400639

$\widehat{Q_{ij}^{4}}$

0	1	2	3
1	-0,09969985	-0,65103497	0,061283927
1	-0,58807736	-0,03654713	-0,06997811
1	0,047790409	-0,03969404	-0,65334834
1	-0,95906676	0,371164916	-0,41171232
1	-0,61996294	-0,51972528	0,171516888

ANEXO 10. CÁLCULO DE VALORES $\widehat{R_{ij}^{X}}$ (R_{ij}^{X} EN EL DOMINIO DE LA FRECUENCIA) .PRIMERA ITERACIÓN

$\widehat{R_{ij}^{0}}$

0	1	2	3
1	-0,00239001	-0,0015913	-0,00529528
1	0,013789021	0,002468849	0,04661114
1	0,016247514	0,013263884	0,002062507
1	0,001677901	0,058731238	0,009710799

$\widehat{R_{ij}^{1}}$

0	1	2	3
1	-0,00535799	0,000672447	0,001558208
1	0,002638224	0,00568084	-0,03581596
1	0,086535192	-0,01850482	-0,01344215

1	0,00418661	-0,00028308	0,00383032

$\widehat{R_{ij}^{2}}$

0	1	2	3
1	0,058932896	0,080041014	-0,12274536
1	0,005645479	0,01685765	0,591079392
1	0,036819209	0,056889502	-0,14166237

$\widehat{R_{ij}^{3}}$

0	1	2	3
1	-0,00025955	-0,00176365	-0,00176413
1	-0,00176381	-0,01085636	0,005427685
1	-0,00104679	-0,0065453	0,000187996
1	-3,616E-05	-0,01207885	0,000702004

$\widehat{R_{ij}^{4}}$

0	1	2	3
1	-0,01671051	-0,00027985	-0,00322855
1	-0,00283302	-0,00498506	0,002827432
1	0,034861296	-0,00458985	0,000302837
1	-0,00173714	0,000490859	0,000480574
1	-0,00268731	-0,00035055	-0,00115358

ANEXO 11. TRANSFORMACIÓN VALORES $\widehat{R_{ij}^{X}}$ AL DOMINIO TEMPORAL. PRIMERA ITERACIÓN

R_{ij}^{0}

0	1	2	3
0,247680851	0,251523498	0,251124142	0,249671509
0,265717252	0,235517172	0,241177258	0,257588317
0,257893476	0,248738466	0,250230281	0,243137777
0,267529984	0,261835634	0,233308966	0,237325415

R_{ij}^{1}

0	1	2	3
0,249218165	0,251118058	0,248102838	0,251560939
0,243125775	0,259714645	0,258193337	0,238966243
0,263647056	0,227100532	0,27962054	0,229631872
0,251933463	0,247924998	0,250159842	0,249981697

R_{ij}^{2}

0	1	2	3
0,254057137	0,285963371	0,275409311	0,184570182
0,40339563	0,105033195	0,099427109	0,392144066
0,238011586	0,290433165	0,280398019	0,19115723

R_{ij}^{3}

0	1	2	3
0,249053165	0,250065008	0,250817059	0,250064768
0,248201877	0,246369941	0,250916216	0,254511965
0,248148975	0,248578374	0,251327628	0,251945023
0,247146748	0,246813826	0,252835172	0,253204254

R_{ij}^{4}

0	1	2	3
0,244945272	0,254914805	0,246699471	0,253440452
0,248752338	0,248755132	0,249831152	0,252661378
0,257643571	0,240061505	0,259787077	0,242507847
0,249808573	0,250436857	0,249322856	0,250431714
0,248952139	0,250872586	0,249704204	0,250471071

ANEXO 12. REGRESO AL ORDENAMIENTO INICIAL VALORES R_{ij}^{X} SEGÚN MATRIZ H. PRIMERA ITERACIÓN

R_{ij}^{0}

0	0	1	2	3
1	0,247680851	0,251523498	0,251124142	0,249671509
2	0,265717252	0,241177258	0,257588317	0,235517172
3	0,257893476	0,243137777	0,248738466	0,250230281
7	0,267529984	0,237325415	0,261835634	0,233308966

R_{ij}^{1}

1	0	1	2	3
1	0,249218165	0,251560939	0,251118058	0,248102838
4	0,243125775	0,238966243	0,259714645	0,258193337
5	0,263647056	0,227100532	0,27962054	0,229631872
6	0,251933463	0,247924998	0,250159842	0,249981697

R_{ij}^{2}

2	0	1	2	3
3	0,254057137	0,184570182	0,285963371	0,275409311
8	0,40339563	0,099427109	0,392144066	0,105033195
9	0,238011586	0,19115723	0,290433165	0,280398019

R_{ij}^{3}

3	0	1	2	3
0	0,249053165	0,250817059	0,250064768	0,250065008
5	0,248201877	0,250916216	0,254511965	0,246369941
7	0,248148975	0,248578374	0,251327628	0,251945023
9	0,247146748	0,252835172	0,253204254	0,246813826

R_{ij}^{4}

4	0	1	2	3
0	0,244945272	0,246699471	0,253440452	0,254914805
2	0,248752338	0,252661378	0,248755132	0,249831152

4	0,257643571	0,240061505	0,259787077	0,242507847
6	0,249808573	0,249322856	0,250431714	0,250436857
8	0,248952139	0,249704204	0,250471071	0,250872586

ANEXO 13. CÁLCULO DE PROBABILIDADES A POSTERIORI DECODIFICADOR LDPC. PRIMERA ITERACIÓN

Pbb a posteriori, SIN NORMALIZAR					
	0	**1**	**2**	**3**	**Suma**
0	0,004736215	0,023049764	0,028709864	0,006173426	0,06266927
1	0,040705857	0,018775538	0,002674616	8,65183E-05	0,06224253
2	0,005046523	0,023683552	0,025975111	0,007626307	0,062331493
3	8,53657E-05	0,003847126	0,031579394	0,032321653	0,067833539
4	0,005555336	0,02245707	0,029363718	0,005299568	0,062675692
5	0,014063329	0,015827234	0,019580032	0,013136958	0,062607554
6	6,07068E-06	0,001259148	0,018421468	0,042914598	0,062601285
7	0,00373054	0,021007434	0,02839197	0,00918546	0,062315403
8	0,000799108	0,004520104	0,056752072	0,006117936	0,06818922
9	5,19E-04	6,45E-03	3,05E-02	3,07E-02	0,068125291

Pbb a posteriori, NORMALIZADAS							
	0	**1**	**2**	**3**	**Sobrev**	**Correcto**	**Resultado**
0	0,075574762	0,36780011	0,458117101	0,098508028	2	2	Correcto
1	0,653987839	0,301651275	0,042970868	0,001390018	0	0	Correcto
2	0,080962654	0,379961252	0,416725311	0,122350782	2	1	Error
3	0,001258458	0,056714215	0,465542487	0,47648484	3	2	Error
4	0,08863621	0,3583059	0,468502498	0,084555391	2	2	Correcto
5	0,224626717	0,25280071	0,312742325	0,209830247	2	2	Correcto
6	9,69737E-05	0,020113778	0,294266613	0,685522636	3	3	Correcto
7	0,059865456	0,337114624	0,455617204	0,147402717	2	1	Error
8	0,011718979	0,066287667	0,832273362	0,089719992	2	2	Correcto
9	0,007617689	0,094608139	0,447087233	0,450686939	3	3	Correcto

ANEXO 14. CÁLCULO DE LAS MÉTRICAS DE RAMA $\boldsymbol{\Gamma}_K(S',S)$. SEGUNDA ITERACIÓN

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
$\boldsymbol{\Gamma}_K(0,0)$	-4,5931	-0,9086	-0,868	-2,1352	-0,8154	-3,3671	-2,5177	-2,2401	-4,9356	-2,7442
$\boldsymbol{\Gamma}_K(0,4)$	-8,8909	-2,6756	-0,9024	-2,2579	-2,469	-7,6693	-6,8352	-0,5649	-2,8926	-2,0267
$\boldsymbol{\Gamma}_K(0,8)$	-3,6205	-2,9783	-6,1206	-6,9186	-3,6865	-0,5809	-4,887	-12,747	-15,125	-5,9953
$\boldsymbol{\Gamma}_K(0,12)$	-3,8146	-0,9789	-2,649	-3,1898	-1,2259	-1,231	-4,2496	-5,5883	-8,3024	-3,7659

$\Gamma_K(1,0)$	-5,7999	-1,0959	-0,8063	-1,5507	-0,7907	-3,7187	-3,4773	-1,7435	-3,8454	-2,1045
$\Gamma_K(1,4)$	-10,428	-3,1933	-1,1712	-2,0039	-2,7748	-8,3515	-8,1253	-0,3988	-2,1329	-1,7175
$\Gamma_K(1,8)$	-4,3351	-2,6683	-5,5604	-5,8331	-3,1635	-0,4361	-5,3532	-11,75	-13,531	-4,8542
$\Gamma_K(1,12)$	-4,8613	-1,001	-2,421	-2,4364	-1,035	-1,4183	-5,0479	-4,9232	-7,0408	-2,957
$\Gamma_K(2,0)$	-7,1788	-1,4501	-0,9104	-1,1294	-0,932	-4,2381	-4,6077	-1,4106	-2,9159	-1,6277
$\Gamma_K(2,4)$	-12,139	-3,8796	-1,6074	-1,9147	-3,2481	-9,203	-9,5878	-0,398	-1,5355	-1,5728
$\Gamma_K(2,8)$	-5,2162	-2,5247	-5,1668	-4,914	-2,807	-0,4577	-5,9859	-10,919	-12,104	-3,8797
$\Gamma_K(2,12)$	-6,0745	-1,1896	-2,3594	-1,8494	-1,0106	-1,7721	-6,0126	-4,4246	-5,9456	-2,3145
$\Gamma_K(3,0)$	-8,724	-1,9707	-1,1809	-0,8746	-1,2397	-4,924	-5,9046	-1,2441	-2,1528	-1,3173
$\Gamma_K(3,4)$	-14,017	-4,7324	-2,21	-1,992	-3,8879	-10,221	-11,217	-0,5636	-1,1045	-1,5946
$\Gamma_K(3,8)$	-6,2637	-2,5476	-4,9395	-4,1614	-2,6169	-0,6458	-6,785	-10,255	-10,843	-3,0716
$\Gamma_K(3,12)$	-7,4541	-1,5446	-2,4643	-1,429	-1,1527	-2,2923	-7,1439	-4,0924	-5,0169	-1,8385
$\Gamma_K(4,1)$	-10,436	-2,6579	-1,618	-0,7862	-1,7139	-5,7764	-7,368	-1,2441	-1,5562	-1,1734
$\Gamma_K(4,5)$	-16,06	-5,7516	-2,9792	-2,2357	-4,6942	-11,405	-13,012	-0,8957	-0,84	-1,7828
$\Gamma_K(4,9)$	-7,4713	-2,7356	-4,8787	-3,5778	-2,593	-0,9983	-7,7454	-9,7593	-9,7537	-2,4327
$\Gamma_K(4,13)$	-8,9922	-2,0631	-2,734	-1,1758	-1,4593	-2,9753	-8,4348	-3,9272	-4,258	-1,5301
$\Gamma_K(5,1)$	-12,304	-3,5069	-2,2181	-0,8635	-2,3509	-6,7898	-8,9894	-1,4093	-1,1277	-1,1955
$\Gamma_K(5,5)$	-18,259	-6,9311	-3,9098	-2,6435	-5,6618	-12,749	-14,964	-1,3914	-0,7421	-2,1354
$\Gamma_K(5,9)$	-8,8509	-3,0906	-4,9836	-3,1573	-2,7351	-1,5185	-8,8767	-9,4272	-8,825	-1,9567
$\Gamma_K(5,13)$	-10,704	-2,7502	-3,171	-1,0874	-1,9334	-3,8276	-9,8981	-3,9271	-3,6614	-1,3862
$\Gamma_K(6,1)$	-14,348	-4,5261	-2,9872	-1,1072	-3,1572	-7,9743	-10,785	-1,7414	-0,8632	-1,3837
$\Gamma_K(6,5)$	-20,635	-8,2825	-5,011	-3,2194	-6,8002	-14,266	-17,092	-2,0556	-0,8097	-2,6557
$\Gamma_K(6,9)$	-10,397	-3,6121	-5,255	-2,9033	-3,0436	-2,2052	-10,174	-9,2615	-8,0628	-1,6471
$\Gamma_K(6,13)$	-12,582	-3,6038	-3,7744	-1,1655	-2,5741	-4,8464	-11,528	-4,0936	-3,2312	-1,4088
$\Gamma_K(7,1)$	-16,559	-5,7118	-3,9228	-1,5174	-4,13	-9,3252	-12,747	-2,2399	-0,7652	-1,7384
$\Gamma_K(7,5)$	-23,178	-9,8003	-6,2787	-3,9617	-8,105	-15,949	-19,386	-2,8862	-1,0438	-3,3425
$\Gamma_K(7,9)$	-12,11	-4,3001	-5,6928	-2,8157	-3,5186	-3,0584	-11,638	-9,2623	-7,467	-1,5041
$\Gamma_K(7,13)$	-14,627	-4,6239	-4,5444	-1,4101	-3,3812	-6,0317	-13,324	-4,4265	-2,9675	-1,5978
$\Gamma_K(8,2)$	-18,935	-7,064	-5,0248	-2,0941	-5,2692	-10,843	-14,875	-2,9049	-0,8336	-2,2596
$\Gamma_K(8,6)$	-25,887	-11,485	-7,7129	-4,8705	-9,5763	-17,798	-21,846	-3,8833	-1,4443	-4,1958
$\Gamma_K(8,10)$	-4,5931	-0,9086	-0,868	-2,1352	-0,8154	-3,3671	-2,5177	-2,2401	-4,9356	-2,7442
$\Gamma_K(8,14)$	-8,8909	-2,6756	-0,9024	-2,2579	-2,469	-7,6693	-6,8352	-0,5649	-2,8926	-2,0267
$\Gamma_K(9,2)$	-3,6205	-2,9783	-6,1206	-6,9186	-3,6865	-0,5809	-4,887	-12,747	-15,125	-5,9953
$\Gamma_K(9,6)$	-3,8146	-0,9789	-2,649	-3,1898	-1,2259	-1,231	-4,2496	-5,5883	-8,3024	-3,7659
$\Gamma_K(9,10)$	-5,7999	-1,0959	-0,8063	-1,5507	-0,7907	-3,7187	-3,4773	-1,7435	-3,8454	-2,1045
$\Gamma_K(9,14)$	-10,428	-3,1933	-1,1712	-2,0039	-2,7748	-8,3515	-8,1253	-0,3988	-2,1329	-1,7175
$\Gamma_K(10,2)$	-4,3351	-2,6683	-5,5604	-5,8331	-3,1635	-0,4361	-5,3532	-11,75	-13,531	-4,8542
$\Gamma_K(10,6)$	-4,8613	-1,001	-2,421	-2,4364	-1,035	-1,4183	-5,0479	-4,9232	-7,0408	-2,957
$\Gamma_K(10,10)$	-7,1788	-1,4501	-0,9104	-1,1294	-0,932	-4,2381	-4,6077	-1,4106	-2,9159	-1,6277
$\Gamma_K(10,14)$	-12,139	-3,8796	-1,6074	-1,9147	-3,2481	-9,203	-9,5878	-0,398	-1,5355	-1,5728
$\Gamma_K(11,2)$	-5,2162	-2,5247	-5,1668	-4,914	-2,807	-0,4577	-5,9859	-10,919	-12,104	-3,8797
$\Gamma_K(11,6)$	-6,0745	-1,1896	-2,3594	-1,8494	-1,0106	-1,7721	-6,0126	-4,4246	-5,9456	-2,3145

$\Gamma_K(11,10)$	-8,724	-1,9707	-1,1809	-0,8746	-1,2397	-4,924	-5,9046	-1,2441	-2,1528	-1,3173
$\Gamma_K(11,14)$	-14,017	-4,7324	-2,21	-1,992	-3,8879	-10,221	-11,217	-0,5636	-1,1045	-1,5946
$\Gamma_K(12,3)$	-6,2637	-2,5476	-4,9395	-4,1614	-2,6169	-0,6458	-6,785	-10,255	-10,843	-3,0716
$\Gamma_K(12,7)$	-7,4541	-1,5446	-2,4643	-1,429	-1,1527	-2,2923	-7,1439	-4,0924	-5,0169	-1,8385
$\Gamma_K(12,11)$	-10,436	-2,6579	-1,618	-0,7862	-1,7139	-5,7764	-7,368	-1,2441	-1,5562	-1,1734
$\Gamma_K(12,15)$	-16,06	-5,7516	-2,9792	-2,2357	-4,6942	-11,405	-13,012	-0,8957	-0,84	-1,7828
$\Gamma_K(13,3)$	-7,4713	-2,7356	-4,8787	-3,5778	-2,593	-0,9983	-7,7454	-9,7593	-9,7537	-2,4327
$\Gamma_K(13,7)$	-8,9922	-2,0631	-2,734	-1,1758	-1,4593	-2,9753	-8,4348	-3,9272	-4,258	-1,5301
$\Gamma_K(13,11)$	-12,304	-3,5069	-2,2181	-0,8635	-2,3509	-6,7898	-8,9894	-1,4093	-1,1277	-1,1955
$\Gamma_K(13,15)$	-18,259	-6,9311	-3,9098	-2,6435	-5,6618	-12,749	-14,964	-1,3914	-0,7421	-2,1354
$\Gamma_K(14,3)$	-8,8509	-3,0906	-4,9836	-3,1573	-2,7351	-1,5185	-8,8767	-9,4272	-8,825	-1,9567
$\Gamma_K(14,7)$	-10,704	-2,7502	-3,171	-1,0874	-1,9334	-3,8276	-9,8981	-3,9271	-3,6614	-1,3862
$\Gamma_K(14,11)$	-14,348	-4,5261	-2,9872	-1,1072	-3,1572	-7,9743	-10,785	-1,7414	-0,8632	-1,3837
$\Gamma_K(14,15)$	-20,635	-8,2825	-5,011	-3,2194	-6,8002	-14,266	-17,092	-2,0556	-0,8097	-2,6557
$\Gamma_K(15,3)$	-10,397	-3,6121	-5,255	-2,9033	-3,0436	-2,2052	-10,174	-9,2615	-8,0628	-1,6471
$\Gamma_K(15,7)$	-12,582	-3,6038	-3,7744	-1,1655	-2,5741	-4,8464	-11,528	-4,0936	-3,2312	-1,4088
$\Gamma_K(15,11)$	-16,559	-5,7118	-3,9228	-1,5174	-4,13	-9,3252	-12,747	-2,2399	-0,7652	-1,7384
$\Gamma_K(15,15)$	-23,178	-9,8003	-6,2787	-3,9617	-8,105	-15,949	-19,386	-2,8862	-1,0438	-3,3425

ANEXO 15. PROGRAMA SERVIDOR

```
 /*
*
* Binarizar-Codificar
* Implementación de Código convolucional sobre un socket servidor
*/

  import java.net.*;
  import java.io.*;
  import prueba.Canal;
  import java.util.Random;

   public class SocketServidorChalo
 {

     String men,fin,Fading_Binario,Gauss_Binario;
     double [] gau;
     double velocidad,frecuencia,SNRNodB;
     int [] palabt;
     int [] cod;
     int O[] = new int[2];
     int Calc[][] = new int [2][3];
     static BufferedOutputStream outputStream;
```

```
    public static void main (String [] args)
   {
      new SocketServidorChalo();
   }

    public SocketServidorChalo()
   {
      try
      {

         ServerSocket socket = new ServerSocket (8000);
         System.out.println ("SIMULACION DE UNA BTS (Servidor)____________________");
        System.out.println ("Esperando cliente");
         Socket cliente = socket.accept();
         System.out.println ("Conectado con cliente de " + cliente.getInetAddress());

         DataOutputStream buffer = new DataOutputStream (cliente.getOutputStream());

         outputStream = new BufferedOutputStream (cliente.getOutputStream());
         // Se envia un entero y una cadena de caracteres.
         String alfa="";
         int largo=21;
         while(largo>=21){
            System.out.println ("Ingrese el texto sin numero (9 caracteres sin espacio
:S)");

            BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));
            alfa = teclado.readLine();
            largo=alfa.length();
         }


      //MODIFICACION*****************************************************************+

      //BINARIZAR
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Binarizando su mensaje, espere un momento...");

         byte[] bytes=alfa.getBytes(); //trasnformacion a codigo ASCII
         for (int i=0;i<bytes.length;i++){
            System.out.println("bytes " + i+" = "+bytes[i]);

         }

         int aa=bytes[0];   //largo del texto
         int oto=alfa.length();
         String texto2=Integer.toString(aa);
         int bi=Canal.binario(Integer.parseInt(texto2));
         String sa=Integer.toString(bi);
         fin=sa;
         for(int h=1;h<oto;h++){
            aa=bytes[h];
            texto2=Integer.toString(aa);
            bi=Canal.binario(Integer.parseInt(texto2));
            sa=Integer.toString(bi);
            fin+=sa;
         }
         System.out.println(".");
         System.out.println("..");
         System.out.println("...");
```

```
   System.out.println("Mensaje Binarizado__________:");
   System.out.println(fin);



//CODIFICACION*************************************************************

   System.out.println("");
   System.out.println("");
   System.out.println("");
//BufferedReader menu = new BufferedReader(new InputStreamReader(System.in));
   System.out.println("Codificando la informacion segun lo deseado...");
   System.out.println(".");
   System.out.println("..");
   System.out.println("...");
   System.out.println("Mensaje Codificado LDPC con__________:");

  String sal;
   largo=fin.length();
   int [] bit=new int[largo];
   int [] sec;
   int [] pal_cod;
   int lar_pal_cod;
   bit=Canal.mensaje(fin); //Se pasa mensaje de string a un arreglo de int
   sec=bin2nobin(bit); //pasar de binario a no binario
   int lar_sec=sec.length;
   for (int i=0;i<lar_sec;i++){
   //System.out.println("sec " + i+ " = " + sec[i]);
   }

  int[][] G= new int[10][5];// matriz generadora
   G[0][0]=1; G[0][1]=0; G[0][2]=0; G[0][3]=0; G[0][4]=0;
   G[1][0]=0; G[1][1]=1; G[1][2]=0; G[1][3]=0; G[1][4]=0;
  G[2][0]=0; G[2][1]=0; G[2][2]=1; G[2][3]=0; G[2][4]=0;
   G[3][0]=0; G[3][1]=0; G[3][2]=0; G[3][3]=1; G[3][4]=0;
   G[4][0]=0; G[4][1]=0; G[4][2]=0; G[4][3]=0; G[4][4]=1;

   G[5][0]=1; G[5][1]=2; G[5][2]=2; G[5][3]=1; G[5][4]=0;
   G[6][0]=1; G[6][1]=1; G[6][2]=2; G[6][3]=1; G[6][4]=3;
   G[7][0]=0; G[7][1]=2; G[7][2]=3; G[7][3]=1; G[7][4]=0;
   G[8][0]=0; G[8][1]=1; G[8][2]=0; G[8][3]=1; G[8][4]=0;
   G[9][0]=0; G[9][1]=3; G[9][2]=0; G[9][3]=2; G[9][4]=0;

   pal_cod=codLDPC(sec,G);
   sal="";
   lar_pal_cod=pal_cod.length;
  for (int i=0; i<lar_pal_cod ; i++){
      sal=sal + pal_cod[i];
   }

   System.out.println("Palabra codificada = " + sal);
   fin=sal;

   //Aplico interleaver
   sal="";
   int[] bits_inter=interleaver(pal_cod);
   int lar_inter=bits_inter.length;
   for (int i=0; i<lar_inter ; i++){
      sal=sal + bits_inter[i];
   }
   System.out.println("Palabra enviada = " + sal);
   //Enviar al terminal
```

```
            buffer.writeUTF(sal);

            System.out.println ("Mensaje enviado al terminal");
           //Hacer funcion exponencial chanta

            cliente.close();
            socket.close();
      }
            catch (Exception e)
            {
               e.printStackTrace();
            }
      }



   //MODIFICACION____ FUNCIONES AUX.*****************************

      static int[] codLDPC(int[] secuencia, int[][] G){
      //llenar con ceros
      int largosec=secuencia.length;
      int diferencia=5-(largosec%5);

      if (largosec%5!=0){
         largosec=largosec+diferencia;
         int[] temp=secuencia;
         secuencia = new int[largosec];
         int lar_ant=largosec-diferencia;
         System.arraycopy(temp,0,secuencia,0,lar_ant);
            for (int i=lar_ant; i<largosec; i++){
               secuencia[i]=0;
               System.out.println("secuencia "+ i +"= " + secuencia[i]);
            }
         }
      int totalveces=largosec/5;
      System.out.println("totalveces = " + totalveces);
      int nfil=10;
      int ncol=5;
      int[] resu=new int[10*totalveces];
      int[] aux=new int[5];
      for (int j=0; j<nfil;j++){
         for (int k=0;k<totalveces;k++){
                        aux[0]=multgf4(G[j][0],secuencia[0+5*k]);
                        aux[1]=multgf4(G[j][1],secuencia[1+5*k]);
                        aux[2]=multgf4(G[j][2],secuencia[2+5*k]);
                        aux[3]=multgf4(G[j][3],secuencia[3+5*k]);
                        aux[4]=multgf4(G[j][4],secuencia[4+5*k]);

resu[j+k*10]=sumagf4(aux[4],sumagf4(aux[3],sumagf4(aux[2],sumagf4(aux[1],aux[0]))));
         }
      }
      int largoresu=resu.length;
      for (int i=0;i<largoresu;i++){
         //System.out.println("palabra codif = "+ i + " " + resu[i]);
         }
         return resu;
      }

      static int[] bin2nobin(int[] sbin){
      int lar_sbin=sbin.length; System.out.println("" + "lar_sbin= " +lar_sbin);
      int[] sbin2;
      if (lar_sbin%2!=0){
         sbin2=new int[lar_sbin+1];
```

```
    int[] temp=sbin;
    System.arraycopy(temp,0,sbin2,0,lar_sbin);
    sbin2[lar_sbin]=0;
    lar_sbin++;
    }
 else {sbin2=sbin;}
 int lar_snobin=lar_sbin/2; System.out.println("" + "lar_snobin= " +lar_snobin);
 int[] snobin=new int[lar_snobin];
 for (int i=0;i<lar_sbin;i++){
    //System.out.println("en ESTE i se cae"+i);
    if(sbin2[i]==0 && sbin2[i+1]==0) {snobin[i/2]=0; i++;}
    else if(sbin2[i]==0 && sbin2[i+1]==1) {snobin[i/2]=1; i++;}
    else if(sbin2[i]==1 && sbin2[i+1]==0) {snobin[i/2]=2; i++;}
    else if(sbin2[i]==1 && sbin2[i+1]==1) {snobin[i/2]=3; i++;}
 }
 return snobin;
 }

 static int sumagf4(int p1, int p2){
 int rsum;
 int suma=p1+p2;
 if(p1==0 || p2==0) rsum=p1+p2;
 else if(suma==2 && p1==p2) rsum=0;
 else if(suma==3) rsum=3;
 else if(suma==4 && p1!=p2) rsum=2;
 else if(suma==4 && p1==p2) rsum=0;
 else if(suma==5) rsum=1;
 else if(suma==6) rsum=0;
 else rsum=5;
 return rsum;
 }

 static int multgf4(int p1, int p2){
 int rmult=p1*p2;
 int multip=p1*p2;
if (p1<2 || p2<2) rmult=p1*p2;
 else if(multip==4) rmult=3;
 else if(multip==6) rmult=1;
 else if(multip==9) rmult=2;
 return rmult;
 }

 static int tobinario(int uno){
    int resultado;
    if(uno==1) resultado=1;
    else resultado=0;
    return resultado;
 }


static int[] interleaver(int[] pal_cod){
 int largo=pal_cod.length;
 int nveces=largo/10;
 int[] pc_inter= new int[largo];
 for (int i=0;i<nveces;i++){
    pc_inter[0+10*i]=pal_cod[4+10*i];
    pc_inter[1+10*i]=pal_cod[2+10*i];
    pc_inter[2+10*i]=pal_cod[9+10*i];
    pc_inter[3+10*i]=pal_cod[7+10*i];
    pc_inter[4+10*i]=pal_cod[0+10*i];
    pc_inter[5+10*i]=pal_cod[1+10*i];
    pc_inter[6+10*i]=pal_cod[8+10*i];
    pc_inter[7+10*i]=pal_cod[6+10*i];
```

```
        pc_inter[8+10*i]=pal_cod[3+10*i];
        pc_inter[9+10*i]=pal_cod[5+10*i];
      }
   return pc_inter;
     }

   }
```

ANEXO 16. PROGRAMA CLIENTE

```
/*
 *
 *Decodificación convolucional en terminal
 *
 *
 */

import java.util.Random;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.lang.*;
import java.util.*;
import java.util.Random;
import prueba.Canal;
import javax.microedition.io.*;
import java.io.*;
import prueba.Float11;

/**
 * An example MIDlet with simple "midvit" text and an Exit command.
 * Refer to the startApp, pauseApp, and destroyApp
 * methods so see how each handles the requested transition.
 *
 * @author  Cristian chong
 * @version
 */
public class el65h extends MIDlet implements CommandListener {

    SocketConnection socket;

    private Command exitCommand; // The exit command
    private Display display;    // The display for this MIDlet
    Command Decodificar;
    TextField mensajein, coddec, mensajedec, string_item_form,string_item_form_msg;
    Form form,form1;
    String men,out,decod;
    StringBuffer buffer;

    int[] llegada = new int[1000];
    int[][] ordenados = new int[4][300];
    int[] orden_final=new int[300];


    public el65h() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Salir", Command.SCREEN, 2);
```

```
        form = new Form("Recibiendo Mensaje del BTS");
        form1 = new Form("Decodificación");


        coddec = new TextField("Código Decodificado","",300,TextField.ANY);
        mensajedec = new TextField("Mensaje Recibido","",300,TextField.ANY);

        string_item_form = new TextField("Mensaje","", 300, TextField.ANY);
        string_item_form_msg = new TextField("Mensaje Recibido","", 300, TextField.ANY);
//
        Decodificar=new Command("Decodificar",Command.OK,1);

//
        form.append(string_item_form);
        form.append(string_item_form_msg);
        form.addCommand(Decodificar);
        form.setCommandListener(this);
//
        form1.append(coddec);
        form1.append(mensajedec);
        form1.addCommand(exitCommand);
        form1.setCommandListener(this);


    }


    public void startApp() {

        display.setCurrent(form);


          try{
            socket = (SocketConnection)Connector.open("socket://localhost:8000");
            string_item_form.setString("Conexion establecida con el BTS");


            int ch=0;
            buffer = new StringBuffer();
            InputStream is = socket.openInputStream();
            men="";
            while ((ch = is.read()) != -1) {
               men=men+(char)ch;
            }

            int k=men.length();
            men=men.substring(2,k);
            // muestra el mensaje recibido del servidor
            //Pasar el string a arreglo de ints
            int largo_men=men.length();
            int num_arreglos=largo_men/10;
            int[][] men_num=new int[num_arreglos][10];
            int lar_str=0;
            for (int j=0;j<num_arreglos;j++){
                for (int i=0;i<10;i++){
                    men_num[j][i]=Integer.parseInt(men.substring(lar_str,lar_str+1));
                    lar_str++;
                }
            }
            double[][] fad_pru=new double[num_arreglos][10];
            for (int kn=0;kn<num_arreglos;kn++){
                fad_pru[kn]=canalfad(men_num[kn],10);
```

```
            }

            double[][] sen_ext={{0,0,0,0,0,0,0,0,0,0},
                                {0,0,0,0,0,0,0,0,0,0},
                                {0,0,0,0,0,0,0,0,0,0},
                                {0,0,0,0,0,0,0,0,0,0}};
            double[][][] sen_final=new double[num_arreglos][][];
            double tiempoactual=0;
            double tiempotranscurrido=0;
            int[] bit_deci = new int[10];
            int n_itera=4;
            for (int kn=0;kn<num_arreglos;kn++){
                for (int kc=0;kc<n_itera;kc++){
                    //System.out.println("RESULTADOS ITERACION "+kc);
                    double[][][] iteracion=new double[n_itera][][];
                    tiempoactual=System.currentTimeMillis();
                    iteracion[kc]= ecu_ldpc_nobin(fad_pru[kn],sen_ext);
                    sen_ext=iteracion[kc];
                    sen_final[kn]=sen_ext;
                    sen_ext=trasponer(sen_ext);
                    for (int i=0;i<sen_ext.length;i++){
                        sen_ext[i]=interleaver(sen_ext[i]);
                    }
                    tiempotranscurrido=System.currentTimeMillis()-tiempoactual;
                    //System.out.println("Tiempo transcurrido iteracion "+kc+" [ms] = "+
tiempotranscurrido);
                }
            }
            int[][] deci_final=new int[sen_final.length][];
            for (int ki=0;ki<sen_final.length;ki++){
                deci_final[ki]=decision(sen_final[ki]);
                System.out.println("DECISION FINAL" + ki);
                //int[] resu_probar=Mult_mat_vectgf4(H,decipri);
                for (int i=0;i<deci_final[ki].length;i++){
                    System.out.print("b"+i+"= "+deci_final[ki][i]+"  ");
                }System.out.println("");
            }
            String pruebapalabra;
            pruebapalabra=nobin2bin(deci_final);
            System.out.println("Palabra de prueba = " + pruebapalabra);
            buffer.append(men);
            string_item_form_msg.setString(men);
            men=pruebapalabra;

           // cerramos flujo entrante
            is.close();
        }
             catch(Exception e){ System.out.println("ERROR = " + e);
            }

    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command c, Displayable s) {

                if(c==Decodificar){
         // display.setCurrent(form1);
            // men=buffer.toString();
```

```
            // coddec.setString(men);

                 display.setCurrent(form1);
                 coddec.setString(men);

                int dadm=men.length();
                byte[] sali=new byte[dadm/7];
                String ciad;
                sali=convert_byte(men);
                ciad=byte_to_String(sali);
                mensajedec.setString(ciad);
        }

         else if (c == exitCommand) {

            //
            try {
               // cerrar conexion y salir de la aplicacion
               socket.close();
               System.out.println("Cerrado");
               destroyApp(false);
               notifyDestroyed();
               System.out.println("Cerrado");
            //
            }
                catch(Exception e){
               }
         }
    }

     static String viterbi (String palabra){
            System.out.println("palabra "+ palabra);
       int largo = palabra.length();
            System.out.println("largo "+ largo);
          //  double paldob[]=new double[4]; paldob[0]=1.802500 ; paldob[1]=1.247500;
paldob[2]=1.662500; paldob[3]=-1.867500;
          //  paldob=normavect(paldob);
          //  double residuoini[]=new
double[4];residuoini[0]=0;residuoini[1]=0;residuoini[2]=0;residuoini[3]=0;
          //  ecualiza(20, paldob, residuoini);
         return null;
        }


    public double[] creatrellis(){
        double[] salida=new double[64];
        int flag=0;
        int p=0;
        int j=0;
        for (int i=0;i<16;i++){
            salida[p]=0.407*0+0.815*j+0.204*flag; p++;
            salida[p]=0.407*1+0.815*j+0.204*flag; p++;
            salida[p]=0.407*2+0.815*j+0.204*flag; p++;
            salida[p]=0.407*3+0.815*j+0.204*flag; p++;
            flag++;
            if (flag%4==0) {
                flag=0;
                j++;
                if (j==4) j=0;
            }
        }
        return salida;
    }
```

```
    public double[][] calc_gama(double[] trellis, double[] recib, double Le[][]){
        int lrec=recib.length;
        int largo_tre=trellis.length;
        double gama[][]=new double[largo_tre][lrec];
        int p=0;
        for (int i=0;i<lrec;i++){
            for (int k=0;k<largo_tre;k++){
                gama[k][i]=-2*(trellis[k]-recib[i])*(trellis[k]-recib[i])+Le[p][i];
                p++;
                if (p%4==0) p=0;
            }
        }
        return gama;
    }

    public double[][] calcalfa(double[][] gama){
        int ncolgama=gama[0].length;
        double[][] alfa= new double[16][ncolgama+1];
        int k=0;int l=0;
        for (int i=0;i<16;i++){
            alfa[i][k]=0;
        }
        for(k=1;k<=ncolgama;k++){
            for(int p=0;p<4;p++){

                alfa[l][k]=Float11.log(Float11.exp(alfa[0][k-1]+gama[p][k-1])+
                        Float11.exp(alfa[1][k-1]+gama[p+4][k-1])+
                        Float11.exp(alfa[2][k-1]+gama[p+8][k-1])+
                        Float11.exp(alfa[3][k-1]+gama[p+12][k-1]));
                l++;
                alfa[l][k]=Float11.log(Float11.exp(alfa[4][k-1]+gama[p+16][k-1])+
                        Float11.exp(alfa[5][k-1]+gama[p+20][k-1])+
                        Float11.exp(alfa[6][k-1]+gama[p+24][k-1])+
                        Float11.exp(alfa[7][k-1]+gama[p+28][k-1]));
                l++;
                alfa[l][k]=Float11.log(Float11.exp(alfa[8][k-1]+gama[p+32][k-1])+
                        Float11.exp(alfa[9][k-1]+gama[p+36][k-1])+
                        Float11.exp(alfa[10][k-1]+gama[p+40][k-1])+
                        Float11.exp(alfa[11][k-1]+gama[p+44][k-1]));
                l++;
                alfa[l][k]=Float11.log(Float11.exp(alfa[12][k-1]+gama[p+48][k-1])+
                        Float11.exp(alfa[13][k-1]+gama[p+52][k-1])+
                        Float11.exp(alfa[14][k-1]+gama[p+56][k-1])+
                        Float11.exp(alfa[15][k-1]+gama[p+60][k-1]));
                l++;


                /*
                alfa[l][k]=log_exp(alfa[0][k-1]+gama[p][k-1],alfa[1][k-1]+gama[p+4][k-
1],alfa[2][k-1]+gama[p+8][k-1],alfa[3][k-1]+gama[p+12][k-1]);
                l++;
                alfa[l][k]=log_exp(alfa[4][k-1]+gama[p+16][k-1],alfa[5][k-1]+gama[p+20][k-
1],alfa[6][k-1]+gama[p+24][k-1],alfa[7][k-1]+gama[p+28][k-1]);
                l++;
                alfa[l][k]=log_exp(alfa[8][k-1]+gama[p+32][k-1],alfa[9][k-1]+gama[p+36][k-
1],alfa[10][k-1]+gama[p+40][k-1],alfa[11][k-1]+gama[p+44][k-1]);
                l++;
                alfa[l][k]=log_exp(alfa[12][k-1]+gama[p+48][k-1],alfa[13][k-1]+gama[p+52][k-
1],alfa[14][k-1]+gama[p+56][k-1],alfa[15][k-1]+gama[p+60][k-1]);
                l++; */
                } if (l%16==0) l=0;
        }
```

```
        return alfa;
    }

    public double[][] calcbeta(double[][] gama){
        int ncolgama=gama[0].length;
       double[][] beta=new double[16][ncolgama+1];
        int k=ncolgama;
        int l=0;
        int t;
        for (int i=0;i<16;i++){
            beta[i][k]=0;
        }
        k--;
        for (;k>=0;k--){
            t=0;
            for (int p=0; p<4;p++){

                beta[l][k]=Float11.log(Float11.exp(beta[p+0][k+1]+gama[t+0][k])+
                        Float11.exp(beta[p+4][k+1]+gama[t+1][k])+
                        Float11.exp(beta[p+8][k+1]+gama[t+2][k])+
                        Float11.exp(beta[p+12][k+1]+gama[t+3][k]));
                l++;
                beta[l][k]=Float11.log(Float11.exp(beta[p+0][k+1]+gama[t+4][k])+
                        Float11.exp(beta[p+4][k+1]+gama[t+5][k])+
                        Float11.exp(beta[p+8][k+1]+gama[t+6][k])+
                        Float11.exp(beta[p+12][k+1]+gama[t+7][k]));
                l++;
                beta[l][k]=Float11.log(Float11.exp(beta[p+0][k+1]+gama[t+8][k])+
                        Float11.exp(beta[p+4][k+1]+gama[t+9][k])+
                        Float11.exp(beta[p+8][k+1]+gama[t+10][k])+
                        Float11.exp(beta[p+12][k+1]+gama[t+11][k]));
                l++;
                beta[l][k]=Float11.log(Float11.exp(beta[p+0][k+1]+gama[t+12][k])+
                        Float11.exp(beta[p+4][k+1]+gama[t+13][k])+
                        Float11.exp(beta[p+8][k+1]+gama[t+14][k])+
                        Float11.exp(beta[p+12][k+1]+gama[t+15][k]));
                l++;


/*beta[l][k]=log_exp(beta[p+0][k+1]+gama[t+0][k],beta[p+4][k+1]+gama[t+1][k],beta[p+8][k+1]+gam
a[t+2][k],beta[p+12][k+1]+gama[t+3][k]);
                l++;

beta[l][k]=log_exp(beta[p+0][k+1]+gama[t+4][k],beta[p+4][k+1]+gama[t+5][k],beta[p+8][k+1]+gama[
t+6][k],beta[p+12][k+1]+gama[t+7][k]);
                l++;

beta[l][k]=log_exp(beta[p+0][k+1]+gama[t+8][k],beta[p+4][k+1]+gama[t+9][k],beta[p+8][k+1]+gama[
t+10][k],beta[p+12][k+1]+gama[t+11][k]);
                l++;

beta[l][k]=log_exp(beta[p+0][k+1]+gama[t+12][k],beta[p+4][k+1]+gama[t+13][k],beta[p+8][k+1]+gam
a[t+14][k],beta[p+12][k+1]+gama[t+15][k]);
                l++;*/
                t=t+16;
            } if (l%16==0) l=0;
        }
        return beta;
    }

    public double[][] pbbapos(double[][] alfa, double[][] beta, double[][] gama){
        //calcular sigma
        int ncolgama=gama[0].length;
```

```
        int q;
        int p;
        int t;
        int flag;
        int flag2;
        double[][] sigma=new double[64][ncolgama];
        for (int i=0;i<ncolgama;i++){
            q=0; p=0; t=0; flag=0; flag2=0;
            for (int k=0;k<64;k++){
                sigma[k][i]=alfa[q][i]+beta[p+t][i+1]+gama[k][i];
                flag++;
                if(flag==4){q++;flag=0;}
                p=p+4;
                if (p==16){p=0;flag2++;}
                if (flag2==4){t++;flag2=0;}
                if (t==4) t=0;
            }
       }

        double[][] probap=new double[4][ncolgama];
        for (int k=0; k<ncolgama;k++){

probap[0][k]=Float11.log(Float11.exp(sigma[0][k])+Float11.exp(sigma[4][k])+Float11.exp(sigma[8]
[k])+Float11.exp(sigma[12][k])+Float11.exp(sigma[16][k])+Float11.exp(sigma[20][k])+Float11.exp(
sigma[24][k])+Float11.exp(sigma[28][k])+Float11.exp(sigma[32][k])+Float11.exp(sigma[36][k])+Flo
at11.exp(sigma[40][k])+Float11.exp(sigma[44][k])+Float11.exp(sigma[48][k])+Float11.exp(sigma[52
][k])+Float11.exp(sigma[56][k])+Float11.exp(sigma[60][k]));

probap[1][k]=Float11.log(Float11.exp(sigma[1][k])+Float11.exp(sigma[5][k])+Float11.exp(sigma[9]
[k])+Float11.exp(sigma[13][k])+Float11.exp(sigma[17][k])+Float11.exp(sigma[21][k])+Float11.exp(
sigma[25][k])+Float11.exp(sigma[29][k])+Float11.exp(sigma[33][k])+Float11.exp(sigma[37][k])+Flo
at11.exp(sigma[41][k])+Float11.exp(sigma[45][k])+Float11.exp(sigma[49][k])+Float11.exp(sigma[53
][k])+Float11.exp(sigma[57][k])+Float11.exp(sigma[61][k]));

probap[2][k]=Float11.log(Float11.exp(sigma[2][k])+Float11.exp(sigma[6][k])+Float11.exp(sigma[10
][k])+Float11.exp(sigma[14][k])+Float11.exp(sigma[18][k])+Float11.exp(sigma[22][k])+Float11.exp
(sigma[26][k])+Float11.exp(sigma[30][k])+Float11.exp(sigma[34][k])+Float11.exp(sigma[38][k])+Fl
oat11.exp(sigma[42][k])+Float11.exp(sigma[46][k])+Float11.exp(sigma[50][k])+Float11.exp(sigma[5
4][k])+Float11.exp(sigma[58][k])+Float11.exp(sigma[62][k]));

probap[3][k]=Float11.log(Float11.exp(sigma[3][k])+Float11.exp(sigma[7][k])+Float11.exp(sigma[11
][k])+Float11.exp(sigma[15][k])+Float11.exp(sigma[19][k])+Float11.exp(sigma[23][k])+Float11.exp
(sigma[27][k])+Float11.exp(sigma[31][k])+Float11.exp(sigma[35][k])+Float11.exp(sigma[39][k])+Fl
oat11.exp(sigma[43][k])+Float11.exp(sigma[47][k])+Float11.exp(sigma[51][k])+Float11.exp(sigma[5
5][k])+Float11.exp(sigma[59][k])+Float11.exp(sigma[63][k]));
            /*

probap[0][k]=log_exp(sigma[0][k],sigma[4][k],sigma[8][k],sigma[12][k],sigma[16][k],sigma[20][k]
,sigma[24][k],sigma[28][k],sigma[32][k],sigma[36][k],sigma[40][k],sigma[44][k],sigma[48][k],sig
ma[52][k],sigma[56][k],sigma[60][k]);

probap[1][k]=log_exp(sigma[1][k],sigma[5][k],sigma[9][k],sigma[13][k],sigma[17][k],sigma[21][k]
,sigma[25][k],sigma[29][k],sigma[33][k],sigma[37][k],sigma[41][k],sigma[45][k],sigma[49][k],sig
ma[53][k],sigma[57][k],sigma[61][k]);

probap[2][k]=log_exp(sigma[2][k],sigma[6][k],sigma[10][k],sigma[14][k],sigma[18][k],sigma[22][k
],sigma[26][k],sigma[30][k],sigma[34][k],sigma[38][k],sigma[42][k],sigma[46][k],sigma[50][k],si
gma[54][k],sigma[58][k],sigma[62][k]);

probap[3][k]=log_exp(sigma[3][k],sigma[7][k],sigma[11][k],sigma[15][k],sigma[19][k],sigma[23][k
],sigma[27][k],sigma[31][k],sigma[35][k],sigma[39][k],sigma[43][k],sigma[47][k],sigma[51][k],si
gma[55][k],sigma[59][k],sigma[63][k]);*/
        }
```

```
        return probap;
    }

    public double[][] LDPC_nobin(double[][] pbbapost2){
        int nchk=5;
       int nsimb=10;
        int[][]Ht={
{0,0,0,2,2},{1,3,0,0,0},{2,0,0,0,3},{3,0,3,0,0},{0,3,0,0,1},{0,1,0,2,0},{0,1,0,0,2},{3,0,0,1,0}
,{0,0,2,0,2},{0,0,3,2,0}};
        int[][]H={
{0,1,2,3,0,0,0,3,0,0},{0,3,0,0,3,1,1,0,0,0},{0,0,0,3,0,0,0,0,2,3},{2,0,0,0,0,2,0,1,0,2},{2,0,3,
0,1,0,2,0,2,0}};
        double[][] WHM={{1,1,1,1},{1,-1,1,-1},{1,1,-1,-1},{1,-1,-1,1}};
        double[][] invWHM={{0.25,0.25,0.25,0.25},{0.25,-0.25,0.25,-0.25},{0.25,0.25,-0.25,-
0.25},{0.25,-0.25,-0.25,0.25}};

        //Primero, Chequear si palabra que llega es válida
        int[] bit_deci=decision(pbbapost2);
        //System.out.println("Decision finish");
        int[] resu_deci=Mult_mat_vectgf4(H,bit_deci);
        //System.out.println("1er Chequeo finish");

        int[] no_0_chk=new int[nchk];
        for (int i=0; i<nchk;i++){
            no_0_chk[i]=ind_no_cero(H[i]).length;
        }


        double[][] chk0=new double[no_0_chk[0]][4];
        double[][] chk1=new double[no_0_chk[1]][4];
        double[][] chk2=new double[no_0_chk[2]][4];
        double[][] chk3=new double[no_0_chk[3]][4];
        double[][] chk4=new double[no_0_chk[4]][4];

        //System.out.println("Chequeo creacion matrices de chequeo ok");

        //Crear las matrices de chequeo

            for (int i=0;i<4;i++){
                for (int j=0;j<ind_no_cero(H[0]).length;j++){

chk0[j][i]=pbbapost2[ind_no_cero(H[0])[j]][invmultgf4(i,H[0][ind_no_cero(H[0])[j]])];
                }
            }

            for (int i=0;i<4;i++){
                for (int j=0;j<ind_no_cero(H[1]).length;j++){

chk1[j][i]=pbbapost2[ind_no_cero(H[1])[j]][invmultgf4(i,H[1][ind_no_cero(H[1])[j]])];
                }
            }

            for (int i=0;i<4;i++){
                for (int j=0;j<ind_no_cero(H[2]).length;j++){

chk2[j][i]=pbbapost2[ind_no_cero(H[2])[j]][invmultgf4(i,H[2][ind_no_cero(H[2])[j]])];
                }
            }

            for (int i=0;i<4;i++){
                for (int j=0;j<ind_no_cero(H[3]).length;j++){

chk3[j][i]=pbbapost2[ind_no_cero(H[3])[j]][invmultgf4(i,H[3][ind_no_cero(H[3])[j]])];
```

```
                }
            }

           for (int i=0;i<4;i++){
                for (int j=0;j<ind_no_cero(H[4]).length;j++){

chk4[j][i]=pbbapost2[ind_no_cero(H[4])[j]][invmultgf4(i,H[4][ind_no_cero(H[4])[j]])];
                }
            }
        //Ahora calcular las FFT de estas matrices de Chequeo

        double[][] FFTchk0=Mult_mat(chk0,WHM);
        double[][] FFTchk1=Mult_mat(chk1,WHM);
        double[][] FFTchk2=Mult_mat(chk2,WHM);
        double[][] FFTchk3=Mult_mat(chk3,WHM);
        double[][] FFTchk4=Mult_mat(chk4,WHM);

        //System.out.println("Calculo matrices de chequeo en dominio frecuencia OK");


        //Ahora se deben calcular los Rij
        double[][] Rfrec0=new double[FFTchk0.length][4];
        double[][] Rfrec1=new double[FFTchk1.length][4];
        double[][] Rfrec2=new double[FFTchk2.length][4];
        double[][] Rfrec3=new double[FFTchk3.length][4];
        double[][] Rfrec4=new double[FFTchk4.length][4];

        for (int j=0;j<4;j++){
            double prodaux=1;
            for (int i=0;i<FFTchk0.length;i++){
            prodaux=prodaux*FFTchk0[i][j];
            }
            for (int k=0;k<FFTchk0.length;k++){
            Rfrec0[k][j]=prodaux/FFTchk0[k][j];
            }
        }

        for (int j=0;j<4;j++){
            double prodaux=1;
            for (int i=0;i<FFTchk1.length;i++){
            prodaux=prodaux*FFTchk1[i][j];
            }
            for (int k=0;k<FFTchk1.length;k++){
            Rfrec1[k][j]=prodaux/FFTchk1[k][j];
            }
        }

        for (int j=0;j<4;j++){
            double prodaux=1;
            for (int i=0;i<FFTchk2.length;i++){
            prodaux=prodaux*FFTchk2[i][j];
            }
            for (int k=0;k<FFTchk2.length;k++){
            Rfrec2[k][j]=prodaux/FFTchk2[k][j];
            }
        }

        for (int j=0;j<4;j++){
            double prodaux=1;
            for (int i=0;i<FFTchk3.length;i++){
            prodaux=prodaux*FFTchk3[i][j];
            }
            for (int k=0;k<FFTchk3.length;k++){
```

```
            Rfrec3[k][j]=prodaux/FFTchk3[k][j];
            }
        }

        for (int j=0;j<4;j++){
            double prodaux=1;
            for (int i=0;i<FFTchk4.length;i++){
            prodaux=prodaux*FFTchk4[i][j];
            }
            for (int k=0;k<FFTchk4.length;k++){
            Rfrec4[k][j]=prodaux/FFTchk4[k][j];
            }
        }
        //Aplicar IFFT para tener Rij en el dominio del tiempo

        //System.out.println("Calculo convolucion OK");

       double Rchk[][][]=new double[5][][];
        Rchk[0]=Mult_mat(Rfrec0,invWHM);
        Rchk[1]=Mult_mat(Rfrec1,invWHM);
        Rchk[2]=Mult_mat(Rfrec2,invWHM);
        Rchk[3]=Mult_mat(Rfrec3,invWHM);
        Rchk[4]=Mult_mat(Rfrec4,invWHM);

        //Ahora se debe reordenar
        //System.out.println("Regreso al dominio del tiempo OK");
        double Rreord[][][]=new double[nchk][][];
        for (int i=0;i<nchk;i++){
            Rreord[i]=new double[Rchk[i].length][4];
        }



        for (int i=0;i<nchk;i++){
            for (int j=0;j<Rchk[i].length;j++){
                for (int k=0;k<4;k++){
                    Rreord[i][j][k]=Rchk[i][j][multgf4(H[i][ind_no_cero(H[i])[j]],k)];
                }
            }
        }


        //Finalmente se calculan las Probabilidades a posteriori
        int[] flagsimb=new int[nchk];
        int[][] ind_no_cero_simb=new int[10][];

        for (int i=0;i<10;i++){
            ind_no_cero_simb[i]=ind_no_cero(Ht[i]);
        }

        double psimb[][]=new double[10][4];
        double psimbnorm[][]=new double[10][4];

        //System.out.println("Inicializacion vector psimb OK");
        for (int i=0;i<10;i++){
            for (int j=0;j<4;j++){

psimb[i][j]=pbbapost2[i][j]*Rreord[ind_no_cero_simb[i][0]][flagsimb[ind_no_cero_simb[i][0]]][j]
*Rreord[ind_no_cero_simb[i][1]][flagsimb[ind_no_cero_simb[i][1]]][j];
            }
            psimbnorm[i]=normalizar(psimb[i]);
            flagsimb[ind_no_cero_simb[i][0]]++;
            flagsimb[ind_no_cero_simb[i][1]]++;
```

```
        }

        return psimbnorm;
    }

    public double[][] ecu_ldpc_nobin(double[] senal_rec,double[][] Le){

            double[] trellis=creatrellis();
            //System.out.println("TRELLIS CALCULADO");
            double[][] gama=calc_gama(trellis, senal_rec,Le);
            //System.out.println("GAMA CALCULADO");
            //System.out.println("# filas GAMA = "+gama.length);
            //System.out.println("# columnas GAMA = "+gama[0].length);
            double[][] alfalfa=calcalfa(gama);
           //System.out.println("ALFA CALCULADO");
            //System.out.println("# filas ALFA = "+alfalfa.length);
            //System.out.println("# columnas ALFA = "+alfalfa[0].length);
            double[][] betabeta=calcbeta(gama);
            //System.out.println("BETA CALCULADO");
            //System.out.println("# filas BETA = "+betabeta.length);
            //System.out.println("# columnas BETA = "+betabeta[0].length);
            double[][] proba_apos=pbbapos(alfalfa,betabeta,gama);
            //System.out.println("PBB aposteriori de log MAP calculada");
            //System.out.println("# filas PBB aposteriori log MAP = "+proba_apos.length);
            //System.out.println("# columnas PBB aposteriori log MAP = "+proba_apos[0].length);
            //Aplicar deinterleaver
            for (int i=0;i<proba_apos.length;i++){
                proba_apos[i]=deinterleaver(proba_apos[i]);
            }
            //System.out.println("Deinterleaver OK");
            int[][]H={
{0,1,2,3,0,0,0,3,0,0},{0,3,0,0,3,1,1,0,0,0},{0,0,0,3,0,0,0,0,2,3},{2,0,0,0,0,2,0,1,0,2},{2,0,3,
0,1,0,2,0,2,0}};
            //Pasar al dominio exponencial
            for (int i=0;i<proba_apos.length;i++){
                for (int j=0;j<proba_apos[0].length;j++){
                    proba_apos[i][j]=prueba.Float11.exp(proba_apos[i][j]);
                }
            }
            //Trasponer y normalizar
            proba_apos=trasponer(proba_apos);
            //System.out.println("# filas PBB apost log MAP TRASPUESTA= "+proba_apos.length);
            //System.out.println("# columnas PBB apost log MAP TRASPUESTA=
"+proba_apos[0].length);
            for (int i=0;i<proba_apos.length;i++){
                 proba_apos[i]=normalizar(proba_apos[i]);
            }
            //Finalmente se manda al LDPC no binario

            double[][] pbb_postldpc=LDPC_nobin(proba_apos);

        return pbb_postldpc;
    }

        static double log_exp(double x1, double x2){
        double log_n;
        log_n=Math.max(x1,x2) + buscatabla(Math.abs(x1-x2));
       return log_n;
        }

        static double log_exp(double a, double b, double c, double d){
            double log_exp4;
            log_exp4=log_exp(log_exp(log_exp(a,b),c),d);
```

```
            return log_exp4;
        }

        static double log_exp(double a,double b,double c,double d,double e,double f,double
g,double h,double i,double j,double k,double l,double m, double n,double o,double p){
            double log_exp16;

log_exp16=log_exp(log_exp(log_exp(log_exp(log_exp(a,b,c,d),e,f,g),h,i,j),k,l,m),n,o,p);
            return log_exp16;
        }

        static double buscatabla(double num){
            if (num > 3.7) return 0;
            else if (3.7 >= num && num > 2.25) return 0.05;
            else if (2.25 >= num && num > 1.5) return 0.15;
            else if (1.5 >= num && num > 1.05) return 0.25;
            else if (1.05 >= num && num > 0.7) return 0.35;
            else if (0.7 >= num && num > 0.43) return 0.45;
            else if (0.43 >= num && num > 0.2) return 0.55;
            else return 0.65;
    }

        static int minimo(int valor[]){
            int resultado,largo,tmp;
            largo=valor.length;
            resultado=valor[0];
            for(int i=1;i<largo;i++){
                tmp=valor[i];
                if(resultado>=tmp)
                    resultado=tmp;
            }
            return resultado;
        }

        static int dist_hamming(int ent1 , int ent2){

            int salida;
            salida = Math.abs((ent1%2)-(ent2%2)) + Math.abs(((int)(ent1/2))%2 -
((int)(ent2/2))%2);
            return salida;
        }

        static int decim(String cad){
            int tmp,sali,lon,ex;
            String tempo;
            double num,expo,sal;
            lon=cad.length();
            int uno[]=new int[lon];
            sal=0;
            for(int i=0;i<lon;i++)
                uno[i]=Integer.parseInt(cad.substring(i,i+1));
            for(int i=0;i<lon;i++){
                ex=lon-1-i;
      expo=(double)ex;
      num=(double)uno[i];
      sal=sal+num*Float11.pow(2,expo);
            }
            sali=(int)sal;
            return sali;
      }
        static byte[] convert_byte(String palab){
            int longit=palab.length();
            int largo=longit/7;
```

```
        byte []salida=new byte[largo];
        String tempo;
        for(int i=0;i<largo;i++){
            tempo=palab.substring(7*i,7*(i+1));
            salida[i]=(byte)decim(tempo);

        }
        return salida;

    }
    static String byte_to_String(byte entrada[]){
        int largo=entrada.length;
        char []tempo=new char[largo];
        String salida;
        for(int i=0;i<largo;i++){
            tempo[i]=(char)entrada[i];
        }
        salida=new String(tempo);
        return salida;
    }

  static double[] normalizar(double[] vect){
      double suma=0;
      int larvect=vect.length;
     for (int i=0;i<larvect;i++){
         suma=suma+vect[i];
      }
      for (int k=0;k<larvect;k++){
          vect[k]=vect[k]/suma;
      }
      return vect;
  }

  static double [][] Mult_mat(double [][] a, double [][] b) {

  if (a [0].length != b.length) {
      System.err.println ("# columnas de Matriz 1 no corresponde a # Filas de Matriz 2");
      return null;
  }

  double [][] result = new double[a.length][];
  for (int i = 0; i < result.length; i++){
       result [i] = new double [b [0].length];
  }

  for (int i = 0; i < a.length; i++){
       for (int j = 0; j < b [0].length; j++){
            for(int k = 0; k < a [0].length; k++){
                result [i][j] += a [i][k] * b [k][j];
            }
       }
  }
  return result;
}


  static int [][] Mult_matgf4(int [][] a, int [][] b) {

  if (a [0].length != b.length) { //nfilas de b
      System.err.println ("# columnas de Matriz 1 no corresponde a # Filas de Matriz 2");
      return null;
  }
```

```
      int [][] result = new int[a.length][];
      for (int i = 0; i < result.length; i++){
           result [i] = new int [b [0].length]; //ncols de b
      }

      for (int i = 0; i < a.length; i++){
           for (int j = 0; j < b [0].length; j++){
                for(int k = 0; k < a [0].length; k++){
                    result [i][j] = sumagf4(result[i][j],multgf4(a [i][k],b [k][j]));
                }
           }
      }
      return result;
    }


      static int [] Mult_mat_vectgf4(int [][] a, int[] b) {

      int [] result = new int[a.length];
      for (int i = 0; i < a.length; i++){
                for(int k = 0; k < a [0].length; k++){
                    result [i]= sumagf4(result[i],multgf4(a [i][k],b [k]));
                }
      }
      return result;
    }


     static int[] truncagf4(double[] recib){
          int[] recib2=new int[recib.length];
          for(int i=0;i<recib.length;i++){
              if(recib[i]<0.5) recib2[i]=0;
              if(recib[i]>=0.5 && recib[i]<1.5) recib2[i]=1;
              if(recib[i]>=1.5 && recib[i]<2.5) recib2[i]=2;
              else recib2[i]=3;
          }
      return recib2;
      }

      static int[] decision(double[][] recib){
          int[] p=new int[10];
          double max_todos;
          for (int i=0;i<10;i++){

max_todos=Math.max(Math.max(Math.max(recib[i][0],recib[i][1]),recib[i][2]),recib[i][3]);
                if (max_todos==recib[i][0]) p[i]=0;
                 if (max_todos==recib[i][1]) p[i]=1;
                 if (max_todos==recib[i][2]) p[i]=2;
                if (max_todos==recib[i][3]) p[i]=3;
            }
          return p;
      }

      static int[] ind_no_cero(int[] Hk){
       int sizeHk=Hk.length;
        int contador=0;
        for (int i=0;i<sizeHk;i++){
            if(Hk[i]!=0)contador++;
        }
        int[] no_cero=new int[contador];
        int k=0;
        for (int j=0;j<sizeHk;j++){
            if(Hk[j]!=0){no_cero[k]=j;k++;}
```

```
    }
    return no_cero;
  }

    static int sumagf4(int p1, int p2){
  int rsum;
  int suma=p1+p2;
  if(p1==0 || p2==0) rsum=p1+p2;
  else if(suma==2 && p1==p2) rsum=0;
  else if(suma==3) rsum=3;
  else if(suma==4 && p1!=p2) rsum=2;
  else if(suma==4 && p1==p2) rsum=0;
  else if(suma==5) rsum=1;
  else if(suma==6) rsum=0;
  else rsum=5;
  return rsum;
 }

  static int multgf4(int p1, int p2){
  int rmult=p1*p2;
  int multip=p1*p2;
  if (p1<2 || p2<2) rmult=p1*p2;
  else if(multip==4) rmult=3;
  else if(multip==6) rmult=1;
  else if(multip==9) rmult=2;
  return rmult;
  }

        static int invmultgf4(int p2, int p1){
            int resu=0;
            if (p1==0) resu=0;
            if (p1==1) resu=p2;
            if (p1==2 && p2==0) resu=0;
            if (p1==2 && p2==1) resu=3;
            if (p1==2 && p2==2) resu=1;
            if (p1==2 && p2==3) resu=2;
            if (p1==3 && p2==0) resu=0;
            if (p1==3 && p2==1) resu=2;
            if (p1==3 && p2==2) resu=3;
            if (p1==3 && p2==3) resu=1;
            return resu;
        }
    static double[][] trasponer(double[][] vecto)   {
        double[][] resulta=new double[vecto[0].length][vecto.length];
        for (int i=0;i<vecto.length;i++){
            for (int j=0;j<vecto[0].length;j++){
                resulta[j][i]=vecto[i][j];
            }
        }
        return resulta;
    }

    public double[] canalfad(int[] recib, int SNRNodB){
        int lreci=recib.length;
        double[] salida=new double[lreci];
        int[] estado=new int[lreci+2];
        System.arraycopy(recib,0,estado,2,lreci);
        estado[0]=0; estado[1]=0;

        //Generacion de ruido
        Random r1=new Random();
        double S,X,Y,Z,sigma;
        double U1=0;
```

```
            double U2=0;
            double[] ruido=new double[lreci];
            sigma = 1/(2*Float11.pow(10,0.1*SNRNodB)) ;
            for (int i=0;i<lreci;i++){
               do {
                   U1=r1.nextDouble();
                   U2=r1.nextDouble();
                   X=2*U1-1;
                   Y=2*U2-1;
                   S=X*X+Y*Y;
               }while (S>1);
               Z= (Math.sqrt(-2*Float11.log(S)/S))*X;
               ruido[i]=Z*sigma;
            }
            for (int i =0; i<recib.length;i++){
                salida[i]=0.407*estado[i+2]+0.815*estado[i+1]+0.204*estado[i]+ruido[i];
            }
            return salida;
        }

         public String nobin2bin(int[][] snobin){
            int nfilas=snobin.length;
            int ncols=snobin[0].length/2;
            String sbin="";
            for (int i=0;i<nfilas;i++){
                for (int j=0;j<ncols;j++){
                       if (snobin[i][j]==0){ sbin = sbin + "00";}
                       if (snobin[i][j]==1){ sbin = sbin + "01";}
                       if (snobin[i][j]==2){ sbin = sbin + "10";}
                       if (snobin[i][j]==3){ sbin = sbin + "11";}
                }
            }
            System.out.println("PALABRA ANTES DEL RECORTE"+sbin);
            int largosbin=sbin.length();
            largosbin=(int)(largosbin/7);
            sbin=sbin.substring(0,7*largosbin);
            return sbin;
        }


        static double[] deinterleaver(double[] bit_inter){
    int largo=bit_inter.length;
    double[] bit_inter2= new double[largo];

     bit_inter2[0]=bit_inter[4];
     bit_inter2[1]=bit_inter[5];
     bit_inter2[2]=bit_inter[1];
     bit_inter2[3]=bit_inter[8];
     bit_inter2[4]=bit_inter[0];
     bit_inter2[5]=bit_inter[9];
     bit_inter2[6]=bit_inter[7];
     bit_inter2[7]=bit_inter[3];
     bit_inter2[8]=bit_inter[6];
               bit_inter2[9]=bit_inter[2];

  return bit_inter2;
    }

static double[] interleaver(double[] bit_inter){
    int largo=bit_inter.length;
    double[] bit_inter2= new double[largo];

     bit_inter2[0]=bit_inter[4];
```

```
      bit_inter2[1]=bit_inter[2];
      bit_inter2[2]=bit_inter[9];
      bit_inter2[3]=bit_inter[7];
      bit_inter2[4]=bit_inter[0];
      bit_inter2[5]=bit_inter[1];
      bit_inter2[6]=bit_inter[8];
      bit_inter2[7]=bit_inter[6];
      bit_inter2[8]=bit_inter[3];
               bit_inter2[9]=bit_inter[5];

   return bit_inter2;
     }

}
```