2. CODIGOS LDPC

2.1 Introducción a los LDPC

Los códigos LDPC (Low Density Parity Check codes) son uno de los temas más de moda en la actualidad de la teoría de la codificación. Fueron inventados en 1960 por Robert Gallager (por lo que son también conocidos como códigos Gallager) pero a pesar de que actualmente son considerados los mejores códigos del mundo, fueron olvidados por mucho tiempo. Se descartaron a pesar de teóricamente prometían ecepcionales resultados debido a que las investigaciones de la época en sistemas concatenados los ensombrecieron y a que el hardware no permitía implementaciones efectivas para sus decodificadores. Los LDPC permanecieron entonces estancados durante 30 años hasta que recientemente fueron relanzados, comenzando con los trabajos de David Mackay.

Los LDPC han demostrado ser capaces de aproximarse a la capacidad del canal. De hecho, Mackay ha probado, usando códigos LDPC aleatorios, que son capaces de aproximarse al límite de Shannon de forma exponencial con la longitud del código.

Son códigos lineales que pueden ser descritos a partir de matrices o de grafos bipartitos dispersos. Decimos que una matriz es dispersa cuando más de la mitad de sus elementos valen 0 y un grafo bipartito es aquel que se compone de dos tipos de nodos que solo pueden conectarse con nodos del grupo contrario. La representación mediante grafos es análoga a la representación matricial. Vamos a estudiar ambas representaciones empezando por la representación mediante grafos.

Supongamos un grafo con n nodos a la izquierda (nodos de bit) y m nodos a la derecha (nodos de chequeo). Este grafo representa un código lineal de longitud n y dimensión (como mínimo) $n - m$ de la siguiente manera: La coordenada n de las palabras de código se asocian a los n nodos de bit y los nodos de chequeo cumplen que la suma de todas las posiciones de la palabra de código en los nodos de bit vecinas al nodo de chequeo en cuestión es cero. El grafo tendrá una arista entre el n-ésimo nodo de bit y el m-ésimo nodo de chequeo si y solo si el bit n-ésimo esta involucrado en el chequeo m-ésimo. Se explica más fácilmente en la figura 2.1. A estos grafos se les conoce como Grafos de Tanner.

A partir del grafo de Tanner podemos construir la matriz de chequeo de paridad. Si tenemos una matriz binaria H de tamaño $m \times n$, el elemento (i, j) vale 1 si y solo si el

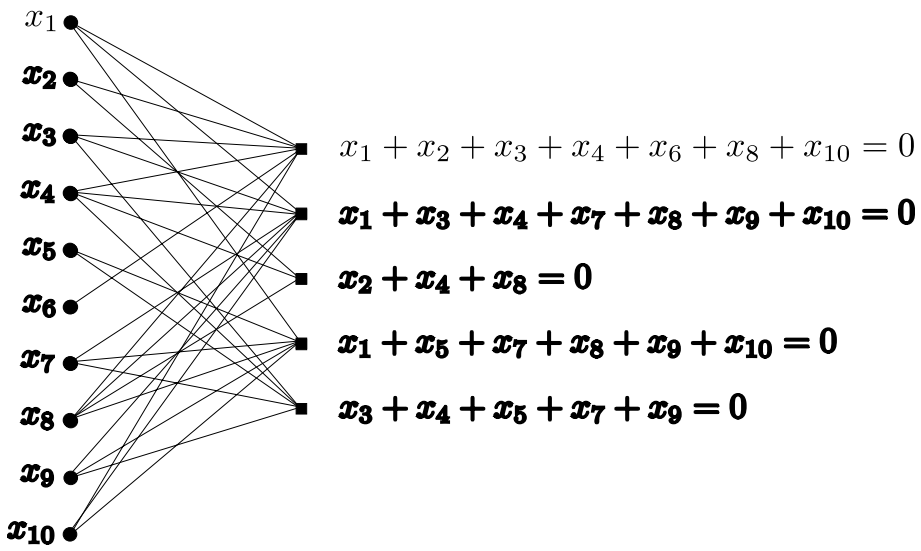

Figura 2.1: Grafo de Tanner

nodo de chequeo i-ésimo está conectado al nodo de bit j-ésimo en el grafo. Por lo tanto, el código LDPC definido por el grafo es el conjunto de vectores $c = (\ c_1 \quad c_2 \quad \cdots \quad c_n\ )$ tal que $H \cdot c^T = 0$. La matriz H construida será la matriz de chequeo de paridad del código.

De manera inversa, partiendo de una matriz binaria $m \times n$ podemos construir el nodo bipartito entre con n nodos de bit y m nodos de chequeo.

Dicho todo esto, comprobamos que cualquier código linear tiene una representación en grafo bipartito (el grafo no está unívocamente definido por el código) aunque no todos los códigos lineales poseen una representación en grafo bipartito disperso. Si el grafo es disperso, entonces el código será LDPC. Es precisamente esta característica de dispersión la clave para la eficiencia algorítmica de los códigos LDPC.

Además de ser dispersa, impondremos que cualesquiera dos de sus columnas no tengan más de una fila cuyo elemento sea uno en ambas (esto equivale a decir que no haya bucles en el Grafo de Tanner).

Además, los códigos LDPC pueden ser regulares o irregulares. Una matriz generadora de LDPC será regular si todos los pesos (el número de elementos distintos de cero que posee un vector) de sus columnas son iguales (columna a columna) y todos los pesos de sus filas lo sean también (fila a fila), y dará lugar a un código LDPC regular. De igual modo, para el caso opuesto, tendremos un código LDPC irregular.

Los códigos regulares tienen la ventaja de que pueden ser generados fácilmente de forma aleatoria fijando una serie de parámetros que marcará la configuración deseada para la matriz. Si seleccionamos un peso de columna w_c (normalmente un valor entero pequeño,

p.e. $w_c = 3$) y los valores para N (la longitud del código) y M (la redundancia). Dado el tamaño de la matriz y el peso de las columnos, el peso de las filas, w_r, nos viene impuesto por la relación $w_c N = w_r M$. La tasa para este tipo de códigos puede calcularse como: $R = 1 - w_c/w_r$

Gallager demostró que la distancia mínima de un LDPC crece linealmente con N, siempre que $w_c \geq 3$ pero como decíamos, la matriz de chequeo de paridad no tiene por que ser regular. Los códigos con peso de columnas variable son en general superiores a los códigos regulares. La tabla 2.1 muestra umbrales de distintos códigos LDPC a varias tasas, así como la capacidad del canal a esa tasa. Los umbrales se muestran en términos de E_b/N_0 y en términos de la desviación estándar del canal σ_τ, donde

$$\frac{2RE_b}{N_0} = \frac{1}{\sigma_\tau^2}$$

La tabla muestra una mejora del código (menor umbral) con el decrecimiento de la tasa, pero además, para una misma tasa, los valores para w_c y w_r tienen repercusión sobre el umbral. Valores de $w_c > 3$ aumentan el umbral. Esta influencia sugiere que ubicando los pesos inteligentemente tenemos una útil herramienta de diseño. De hecho, los mejores códigos LDPC son irregulares, ganando hasta 0.5dB frente a los regulares. Para entender esta mejoría, podemos usar el razonamiento empleado en [19].

Podemos pensar en el proceso como un juego, con los nodos de mensaje y de bit como los jugadores. Cada jugador intenta elegir el número de aristas correcto. Los nodos de mensaje y los de bit deben estar de acuerdo en el número de aristas. Desde el punto de vista de los nodos de mensaje, es mejor tener un grado alto (muchas aristas) ya que cuanta más información obtengan de los nodos de chequeo, con más precisión podrán juzgar cual es el valor correcto. En contraposición, desde el punto de vista de los nodos de chequeo, es mejor tener un grado bajo, ya que cuanto mayor sea el grado de un nodo de chequeo, más probable es transmitir suposiciones incorrectas a los nodos de mensaje. Estos dos requerimientos enfrentados deben balancearse.

Si permitimos grafos irregulares tendremos más flexibilidad a la hora de balancear los requerimientos, permitiendonos optimizar los resultados. La idea es que los nodos de mensaje de alto grado tiendan a su valor correcto con rapidez y proporcionen buena información a los nodos de chequeo, que proporcionaran mejor información a los nodos de mensaje con grado menor. Por lo tanto, los grafos irregulares conducen a un efecto ola en el que los nodos de mensaje con alto grado tienden a corregirse primero, después lo hacen los nodos de mensaje con grado ligeramente menor, y así hasta el final.

Como puede apreciarse del razonamiento anterior, la desventaja del uso de códigos LDPC irregulares es la dificultad implícita en su diseño.De cualquier modo, son este tipo

w_c	w_r	Tasa	Umbral σ_τ	Umbral E_b/N_0(dB)	Capacidad (dB)	Distancia (dB)
3	12	0.75	0.6297	2.2564	1.6264	0.63
3	9	2/3	0.7051	1.7856	1.0595	0.7261
4	10	0.6	0.7440	1.7767	0.6787	1.098
3	6	0.5	0.8747	1.1628	0.1871	0.9757
4	8	0.5	0.8323	1.5944	0.1871	1.4073
5	10	0.5	0.7910	2.0365	0.1871	1.8494
3	5	0.4	1.0003	0.9665	-0.2383	1.2048
4	6	1/3	1.0035	1.7306	-0.4954	2.226
3	4	0.25	1.2517	1.0603	-0.7941	1.8544

Tabla 2.1: Umbrales de distintos códigos LDPC en un canal AWGN

de matrices las que se usan en la vida real, constituyendo un buen ejemplo la matriz de DVB-S2, que es la que tendremos en cuenta en nuestras simulaciones.

2.2 Decodificación de códigos LDPC

Los algoritmos de decodificación de los LDPC se engloban en los llamados algoritmos de paso de mensajes. Son algoritmos iterativos y su nombre se debe a que el algoritmo de decodificación se basa en la observación de la estructura del grafo y en como en cada iteración, se pasan mensajes de los nodos de bit a los nodos de chequeo y viceversa. Los mensajes de los nodos de bit a los nodos de chequeo se computan en base al valor del nodo de bit y de los mensajes recibidos de sus nodos de chequeo vecinos en la iteración anterior. Hay que tener en cuenta que el mensaje enviado por el nodo de bit v al nodo de chequeo c, no debe tener en cuenta el mensaje de c a v en la iteración anterior. Lo mismo se cumple para los mensajes de c a v. Una importante subclase de los algoritmos de paso de mensajes son los llamados "Algoritmos Belief-Propagation" (BP) que fueron presentados por Gallager en [13].

Los mensajes enviados por las aristas en este algoritmo son probabilidades o "creencias" ("beliefs"). De una forma más precisa, el mensaje enviado por el nodo de bit v al nodo de chequeo c, es la probabilidad de que v tenga un cierto valor dado el valor observado por dicho nodo y los valores recibidos de sus nodos vecinos exceptuando a c. Por otro lado, el mensaje enviado de c a v es la probabilidad de que v tenga un cierto valor dados los valores pasados a c desde sus nodos de bit vecinos, exceptuando a v.

Podemos interpretar los nodos del grafo como pequeñas unidades de procesamiento de señal, que reciben unas entradas y generan unas salidas. El funcionamiento de este tipo de algoritmos se ve mas claramente en la figura 2.2.

Supondremos que queremos arreglar un bit c_n erróneo. Desarrollaremos entonces el grafo en forma de arbol con c_n como raíz. En la parte superior del arbol, las ojas, tenemos los nodos de bit que no dependen de ningún nodo de chequeo, sino de la información obtenida directamente del canal. Esto se corresponde con la inicialización (o iteracción 0) del algoritmo.

La primera iteracción comienza cuando, los nodos de chequeo realizan su cálculo usando la información de las "hojas" y lo transmiten a sus nodos de bit, que computan sus valores. Esto concluiría la iteración número uno.

En la segunda iteración se observaría un comportamiento similar, los nodos de chequeo generarián sus salidas a partir de los valores producidos por los nodos de bit en la iteración anterior y mandarían su cómputo a sus nodos de bit, que realizarían sus cálculos, chequeando su resultado para comprobar si la palabra se ha decodificado correctamente y si no, se continuaría siguiendo la misma ejecución, hasta que se consumiese el número de iteraciones establecido o se obtuviese la palabra correcta.

En la figura 2.2 hemos supuesto un caso ideal, en el que los nodos de bit que influyen en un nodo de chequeo son independientes (ausencia de bucles) pero esto no es así, ya que los bits en la fila 1 y en la 2 no tendrían porque ser distintos, por lo que se formarían bucles, como puede verse en la figura 2.3. Este efecto no deseado hace que los algoritmos no se comporten como idealmente se espera, pero aún así obtienen espectaculares resultados. Si trabajamos con códigos suficientemente grandes, la probabilidad de que ocurran estos bucles es pequeña (por lo menos en el primer nivel), por lo que asumiremos que la estructura será la mostrada en la figura 2.2 y no en la figura 2.3. Esta es la llamada "Suposición de independencia".

Además de la ausencia de bucles, hemos realizado la decoficación tomando el bit c_n como la raíz del arbol, pero en realidad no tenemos una raíz, sino que cada uno de los nodos de bit constituye la raíz de un arbol. El caso real puede verse en la figura 2.4, donde se ha marcado uno de los bucles existentes que deterioran la eficacia del algoritmo.

La representación gráfica de los códigos LDPC es atractiva porque además de ayudar a entender la estructura del código proporciona un poderoso enfoque de decodificación. La clave de los pasos de la decodificación es la aplicación directa de las reglas de Bayes a cada nodo sobre los mensajes intercambiados.

Es fácil derivar fórmulas para estas probabilidades bajo la "Suposición de independencia".

Vamos definir la nomenclatura para describir el funcionamiento del algoritmo. Para facilitar la explicación emplearemos una matriz (2.1) de ejemplo sobre la que iremos denotando

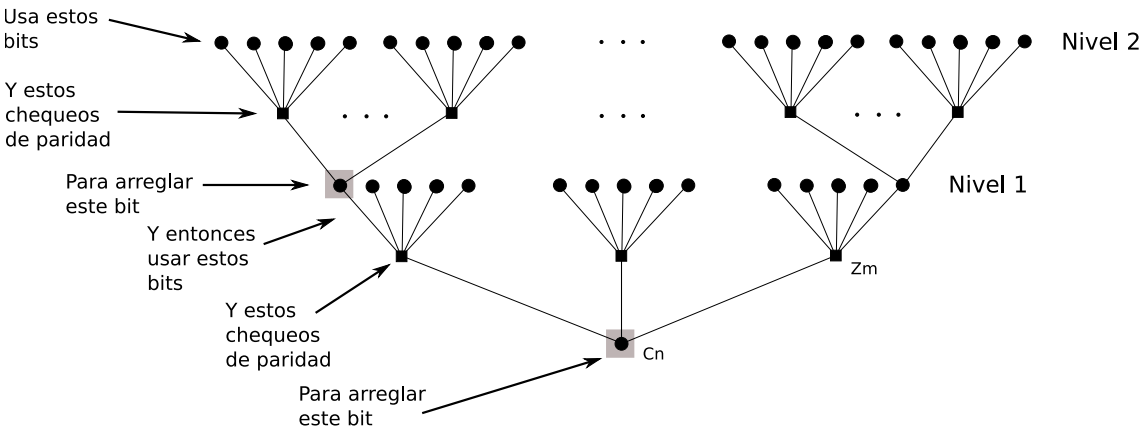

Figura 2.2: Funcionamiento de los algoritmos de decodificación

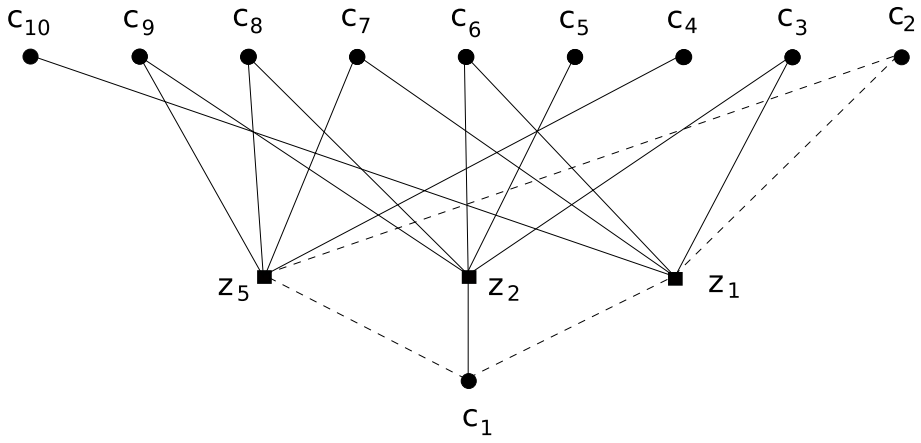

Figura 2.3: Grafo de Tanner con bucles

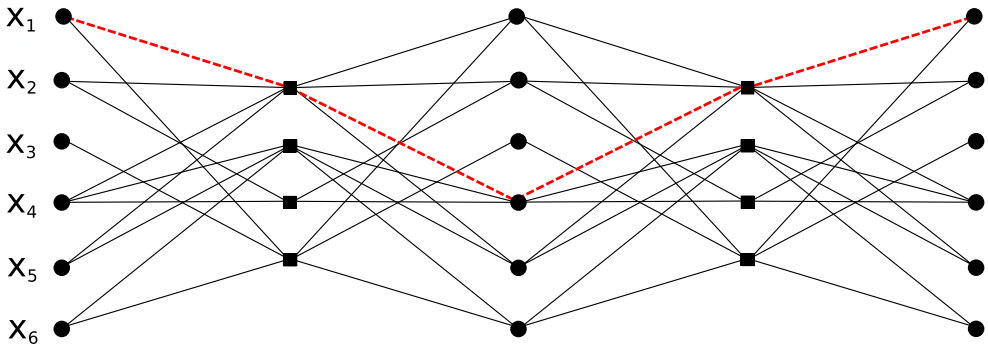

Figura 2.4: Grafo real con bucles

los conceptos introducidos.

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \tag{2.1}$$

Si denotamos los bits por $c_{n'}$ y los chequeos de paridad por $z_{m'}$, el conjunto de bits que interviene en el chequeo z_m (los elementos distintos de cero de la m-ésima fila de A) se denota como

$$\mathcal{N}_m = \{n : A_{mn} = 1\}$$

Podemos escribir entonces el chequeo m-ésimo como

$$z_m = \sum_{n \in \mathcal{N}_m} c_n$$

El conjunto de bits involucrado en el chequeo z_m excluido el bit n se denotará

$$\mathcal{N}_{m,n} = \mathcal{N}_m \backslash n$$

La notación $|\mathcal{N}_m|$ indica el número de elementos en $\mathcal{N}_m$. Estos conjuntos pueden considerarse listas ordenadas, con el i-ésimo elemento de $\mathcal{N}_m$ denotado por $\mathcal{N}_m(i)$. Análogamente, el conjunto de chequeos en los que interviene el bit n será:

$$\mathcal{M}_n = \{m : A_{mn} = 1\}$$

Para un código LDPC regular, $|\mathcal{M}_n| = w_c$. El conjunto de chequeos en los que interviene el bit c_n menos el m será

$$\mathcal{M}_{n,m} = \mathcal{M}_n \backslash m$$

Podemos ahora aplicar todas estas definiciones sobre nuestra matriz de ejemplo 2.1 para afianzarlas:

$$\mathcal{N}_1 = \{1, 2, 3, 6, 7, 10\},\ \mathcal{N}_2 = \{1, 3, 5, 6, 8, 9\},\ \text{etc.}$$
$$\mathcal{M}_1 = \{1, 2, 5\},\ \mathcal{M}_2 = \{1, 4, 5\},\ \text{etc.}$$
$$\mathcal{N}_{2,1} = \{3, 5, 6, 8, 9\},\ \mathcal{N}_{2,3} = \{1, 5, 6, 8, 9\},\ \text{etc.}$$
$$\mathcal{M}_{2,1} = \{4, 5\},\ \mathcal{M}_{2,4} = \{1, 5\},\ \text{etc.}$$

Aunque el decodificador óptimo sería aquel que buscase la palabra de código con menor distancia Hamming a la recibida la complejidad de decodificación del código crece exponencialmente con la longitud del mismo lo que hace esta opción inviable para códigos reales (de gran longitud, 64800bits para DVB-S2). Por lo tanto, en lugar de buscar la palabra $\hat{c}$ que maximice $P(c|r, Ac = 0)$, se busca aquella que maximice

$$P(c_n|r, \text{todos los nodos de chequeo que involucran } c_n \text{ se satisfacen})$$

Es decir, la probabilidad de que para un único bit dado, se satisfagan los sus nodos de chequeo. Con esta simplificación conseguimos que la complejidad de la nueva tarea sea lineal con la longitud.

Vamos a definir los dos conjuntos de probabilidades con los que trabajaremos (para los nodos de chequeo y para los nodos de bit). Por un lado los relacionados con el criterio de decodificación, denotado por

$$q_n(x) = P(c_n = x|r, \{z_m = 0, m \in \mathcal{M}_n\}), \quad x \in \{0, 1\} \tag{2.2}$$

Esta es la probabilidad que se usa para tomar las decisiones acerca de los bits codificados. Si eliminamos la influencia del nodo z_m la fórmula resultante tiene la siguiente forma:

$$q_{mn}(x) = P(c_n = x|r, \{z_{m'} = 0, m' \in \mathcal{M}_{n,m}\}), \quad x \in \{0, 1\} \tag{2.3}$$

El segundo juego de probabilidades se refiere a las probabilidades de los chequeos dados los bits, denotada por

$$r_{mn}(x) = P(z_m = 0|c_n = x, \{z_{m'} = 0, r) \tag{2.4}$$

El cómputo de estas cantidades solo se realiza para los elementos $A_{mn} \neq 0$. Como explicabamos antes el algoritmo funciona de la siguiente manera: Usamos los datos medidos para computar las probabilidades de los nodos de chequeo r_{mn}, emplearemos esta información para obtener información sobre los bits, q_{mn}, que a su vez se usará para calcular una nueva r_{mn}... Se sigue iterativamente hasta que todos los chequeos se cumplan o hasta que se supere un numero límite de iteraciones.

Bajo la "Suposición de independencia", para un c_n dado podemos expresar 2.2 como

$$\begin{aligned} q_n(x) &= P(c_n = 0|r, \{z_m = 0, m \in \mathcal{M}_n\}) = \frac{P(c_n = 0, \{z_m = 0, m \in \mathcal{M}_n\}|r)}{P(\{z_m = 0, m \in \mathcal{M}_n\}|r)} \\ &= \frac{1}{P(\{z_m = 0, m \in \mathcal{M}_n\}|r)} P(c_n = x|r) P(\{z_m = 0, m \in \mathcal{M}_n\}|c_n = x, r) \\ &= \underbrace{\frac{1}{P(\{z_m = 0, m \in \mathcal{M}_n\}|r)}}_{constante\ \alpha_n} P(c_n = x|r_n) \prod_{m \in \mathcal{M}_n} P(z_m = 0|c_n = x, r) \\ &= \underbrace{\alpha P(c_n = x|r_n)}_{\text{prob. intrínseca}} \underbrace{\prod_{m \in \mathcal{M}_n} P(z_m = 0|c_n = x, r)}_{\text{prob. extrínseca}} = \alpha_n P(c_n = x|r_n) \prod_{m \in \mathcal{M}_n} r_{mn}(x) \end{aligned} \quad (2.5)$$

De 2.5 podemos obtener:

$$q_{mn}(x) = \alpha_{mn} P(x) \prod_{m' \in \mathcal{M}_{n,m}} r_{m'n}(x) \quad (2.6)$$

Donde α_{mn} y α_n son factores de normalización que se toman tal que $q_{mn}(0) + q_{mn}(1) = 1$ y $q_n(0) + q_n(1) = 1$.

Como el producto 2.6 se calcula hacia abajo por las columnas de la matriz A (a lo largo de los chequeos), el cálculo de q_{mn} se denomina paso vertical del algoritmo. El proceso puede describirse como: Para cada posición (m, n) distinta de 0 de la matriz A computa el producto de los $r_{m',n}(x)$ hacia abajo por la columna n-ésima de A, excluyendo la de la fila m, entonces multiplica por la probabilidad a posteriori del canal.

De un modo similar, podemos expresar 2.4 como

$$\begin{aligned} r_{mn}(x) = &\sum_{x_{n'}: n' \in \mathcal{N}_{m,n}} [P(z_m = 0|c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}, r) \\ &\times P(\{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}|r)] \\ = &\sum_{x_{n'}: n' \in \mathcal{N}_{m,n}} [P(z_m = 0|c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}) \\ &\times \prod_{n' \in \mathcal{N}_{m,n}} P(c_{n'} = x_{n'}|r)] \\ = &\sum_{x_{n'}: n' \in \mathcal{N}_{m,n}} [P(z_m = 0|c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}) \times \prod_{n' \in \mathcal{N}_{m,n}} q_{mn'}(x_{n'}) \end{aligned} \quad (2.7)$$

Las probabilidades condicionadas en las sumas valen o cero o uno dependiendo de que el z_m observado coincida con los valores supuestos para x_n y $\{x_{n'}\}$. Las probabilidades para z_m dados sus valores observados pueden ser encontradas de forma eficiente usando el

algoritmo "forward-backward"[7][6].

Una implementación particularmente conveniente de este método usa pasadas "forward-backward" en las que se computan los productos de las diferencias $\delta q_{mn} \equiv q_{mn}(0) - q_{mn}(1)$. Obtenemos $\delta r_{mn} \equiv r_{mn}(0) - r_{mn}(1)$ y de ahí la identidad

$$\delta r_{mn} = \prod_{n' \in \mathcal{N}_{m,n}} \delta q_{mn'} \tag{2.8}$$

Esta identidad se deriva por la iteración de la siguiente observación: Si $\zeta = x_\mu + x_\nu$ en módulo 2 y x_μ y x_ν tienen probabilidades $q_\mu(0), q_\nu(0)$ y $q_\mu(1), q_\nu(1)$ de ser 0 o 1, entonces $P(\zeta = 1) = q_\mu(1)q_\nu(0) + q_\mu(0)q_\nu(1)$ y $P(\zeta = 0) = q_\mu(0)q_\nu(0) + q_\mu(1)q_\nu(1)$. Por lo tanto

$$P(\zeta = 0) - P(\zeta = 1) = (q_\mu(0) - q_\mu(1))(q_\nu(0) - q_\nu(1))$$

Finalmente, vemos que $r_{mn}(0) + r_{mn}(1) = 1$ y por lo tanto $r_{mn}(0) = (1 + \delta r_{mn})/2$ y $r_{mn}(1) = (1 - \delta r_{mn})/2$

Lo que quiere decir 2.8 expresado con palabras, es: Para cada elemento (m, n) de A, calcula el producto de los $\delta q_{mn'}$ a lo largo de la fila m-ésima, excepto el valor correspondiente a la columna n. Por eso, este paso se denomina paso horizontal.

Este algoritmo recibe el nombre de "Sum-Product" ya que todas las operaciones empleadas para el cálculo de los mensajes son sumas y productos.

En el Algoritmo 1 puede verse la implementación en pseudo código del algoritmo Sum-Product.

2.3 Representaciones numéricas precisas del Algoritmo BP

En la práctica, es preferible trabajar en el dominio logarítmico. Esto nos permite usar LLR (loglikelihood ratios o relación de verosimilitud) como mensajes y aprovecharnos de las ventajas de implementación sobre el uso de probabilidades o verosimilitudes. Las multiplicaciones son sustituidas por sumas y el paso de normalización es eliminado. En esta sección vamos a estudiar los enfoques más populares para después pasar a analizar alternativas de decodificación que son más fáciles de implementar y pueden reducir el retardo de decodificación.

Antes de nada, vamos a introducir una serie de definiciones matemáticas que usaremos

Algoritmo 1 Algoritmo "Belief Propagation (BP)"

Entrada: La matriz del código (A), las probabilidades a posteriori del canal $p_n(x) = P(c_n = x|r_n)$ y el número máximo de iteraciones L.

Inicialización: Hacer $q_{m,n}(x) = p_n(x)$ para todo (m, n) con $A(m, n) = 1$

1: **mientras** $l < L$ y $A\hat{c} \neq 0$ **hacer**

2: **Paso horizontal:** Para cada (m, n) con $A(m, n) = 1$, Calcular:

$$\delta q_{ml} = q_{ml}(0) - q_{ml}(1)$$

$$\delta r_{mn} = \prod_{n' \in \mathcal{N}_{m,n}} \delta q_{mn'} \tag{2.9}$$

$$r_{mn}(1) = (1 - \delta r_{mn})/2 \qquad \text{y} \qquad r_{mn}(0) = (1 + \delta r_{mn})/2$$

3: **Paso vertical:** Para cada (m, n) con $A(m, n) = 1$, Calcular:

$$q_{mn}(0) = \alpha_{mn} p_n(0) \prod_{\{m' \in \mathcal{M}_{n,m}\}} r_{m'n}(0) \qquad \text{y} \qquad q_{mn}(1) = \alpha_{mn} p_n(1) \prod_{\{m' \in \mathcal{M}_{n,m}\}} r_{m'n}(1) \tag{2.10}$$

Donde α_{mn} se ha elegido para tal que $q_{mn}(0) + q_{mn}(1) = 1$

Calcular también las "pseudoprobabilidades" a posteriori

$$q_n(0) = \alpha_n p_n(0) \prod_{\{m' \in \mathcal{M}_n\}} r_{m'n}(0) \qquad \text{y} \qquad q_n(1) = \alpha_n p_n(1) \prod_{\{m' \in \mathcal{M}_n\}} r_{m'n}(1)$$

Donde α_n se ha elegido para tal que $q_n(0) + q_n(1) = 1$

Tomar una decisión: Hacer $\hat{c}_n = 1$ si $q_n(1) > 0{,}5$ y si no, hacer $\hat{c}_n = 0$.

4: **fin mientras**

5: **si** $A\hat{c} = 0$ **entonces**

6: Palabra decodificada correctamente

7: **si no**

8: Se han producido error de decodificación

9: **fin si**

como herramientas en la deducción de las ecuaciones.

El LLR de una variable aleatoria binaria U es definido como $L(U) \stackrel{\text{def}}{=} log(P_U(u = 0)/P_U(u = 1))$ donde $P_U(u)$ denota la probabilidad de que U tome el valor u. Podemos expresar entonces $P_U(u = 0) = e^{L(U)}/(1 + e^{L(U)})$ y $P_U(u = 1) = 1/(1 + e^{L(U)})$, lo que nos lleva a $P_U(u = 0) - P_U(u = 1) = \tanh(L(U)/2)$. Para dos dos variables binarias estadísticamente independientes U y V, la llamada "regla de la tanh" viene dada por

$$L(U \oplus V) = 2\tanh^{-1}\left(\tanh\left(\frac{L(U)}{2}\right)\tanh\left(\frac{L(V)}{2}\right)\right) \tag{2.11}$$

Una práctica simplificación se deduce del hecho de que las funciones tanh y $\tanh^{-1}$ son monótonamente crecientes y tienen simetría impar, lo que implica que $\tanh(x) = sign(x)\tanh(|x|)$ y $\tanh^{-1}(x) = sign(x)\tanh^{-1}(|x|)$, por lo tanto, en la práctica, el signo y la magnitud de $L(U \oplus V)$ son separables. Equivalentemente, la regla de la tanh es $L(U \oplus V) = sign(L(U))sign(L(V)) \cdot 2\tanh^{-1}(\tanh(|L(U)|/2)\tanh(|L(V)|/2))$

2.3.1 **LLR-BP basado en la regla de la tanh**

El algoritmo que vamos a estudiar se basa en el LLR previamente definido. Vamos a profundizar un poco más en las reglas del álgebra de verosimilitud logarítmica y a definir los LLR de las probabilidades de los nodos. Sea

$$\lambda(c_n|r) = \log\frac{P(c_n = 1|r)}{P(c_n = 0|r)} = \log\frac{P(c_n = 1|r_n, \{r_i, i \neq n\})}{P(c_n = 0|r_n, \{r_i, i \neq n\})} \tag{2.12}$$

el LLR, que por aplicación de la regla de Bayes y teniendo en cuenta que para un c_n dado, r_n es independiente, puede ser reescrita como

$$\lambda(c_n|r) = \log\frac{p(r_n|c_n = 1)}{p(r_n|c_n = 0)} + \log\frac{P(c_n = 1|\{r_i, i \neq n\})}{P(c_n = 0|\{r_i, i \neq n\})} \tag{2.13}$$

Si estamos trabajando con canales Gaussianos

$$\log\frac{p(r_n|c_n = 1)}{p(r_n|c_n = 0)} = L_c r_n,$$

donde $L_c = 2\sqrt{E_c/\sigma^2}$ es la fiabilidad del canal. Identificando términos en la suma obtenemos

$$\lambda(c_n|r) = \underbrace{L_c r_n}_{\text{intrínseco}} + \underbrace{\log \frac{P(c_n = 1|\{r_i, i \neq n\})}{P(c_n = 0|\{r_i, i \neq n\})}}_{\text{extrínseco}} \tag{2.14}$$

El término intrínseco está determinado por la medida explícita del r_n proveniente del canal y el extrínseco está determinado por la información de las demás observaciones y la estructura del código.

Podemos ahora expresar las probabilidades del término extrínseco en función de los chequeos de paridad. Si $z_{m,n}$ denota el chequeo de paridad m-ésimo asociado a c_n despreciando la influencia del propio c_n:

$$z_{m,n} = \sum_{i \in \mathcal{N}_{m,n}} c_i \tag{2.15}$$

Si $c_n = 1$, para que $z_m = z_{mn} + c_n = 0$, $z_{m,n} = 1$ para todos los chequeos $m \in \mathcal{M}_n$ en los que c_n participa. Del mismo modo, si $c_n = 0$, $z_{m,n} = 0$ para todos los $m \in \mathcal{M}_n$. Podemos reescribir entonces 2.14 como

$$\lambda(c_n|r) = L_c r_n + \log \frac{P(z_{m,n} = 1 \,\text{para todos los}\, m \in \mathcal{M}_n|\{r_i, i \neq n\})}{P(z_{m,n} = 0 \,\text{para todos los}\, m \in \mathcal{M}_n|\{r_i, i \neq n\})}$$

Asumiendo que no existen bucles en el grafo, el conjunto de bits asociado con $z_{m,n}$ será independiente del conjunto de $z_{m',n}$ para $m' \neq m$, por lo que es posible expresar en forma de producto la probabilidad conjunta del numerador y el denominador

$$\begin{aligned} \lambda(c_n|r) &= L_c r_n + \log \frac{\prod_{m \in \mathcal{M}_n} P(z_{m,n} = 1\{r_i, i \neq n\})}{\prod_{m \in \mathcal{M}_n} P(z_{m,n} = 0\{r_i, i \neq n\})} \\ &= L_c r_n + \sum_{m \in \mathcal{M}_n} \log \frac{P(z_{m,n} = 1\{r_i, i \neq n\})}{P(z_{m,n} = 0\{r_i, i \neq n\})} \end{aligned} \tag{2.16}$$

Definimos ahora el "log likelihood ratio" como

$$\lambda(z_{m,n}|\{r_i, i \neq n\}) = \log \frac{P(z_{m,n} = 1\{r_i, i \neq n\})}{P(z_{m,n} = 0\{r_i, i \neq n\})}$$

Entonces podemos expresar 2.16 como

$$\lambda(c_n|r) = L_c r_n + \sum_{m\in\mathcal{M}_n} \lambda(z_{m,n}|\{r_i, i\neq n\}) = L_c r_n + \sum_{m\in\mathcal{M}_n} \lambda\left(\sum_{j\in\mathcal{N}_{m,n}} c_j|\{r_i, i\neq n\}\right) \tag{2.17}$$

$$= L_c r_n - 2\sum_{m\in\mathcal{M}_n} \tanh^{-1}\left(\prod_{j\in\mathcal{N}_{m,n}} \tanh\left(-\frac{\lambda(c_j|\{r_i, i\neq n\})}{2}\right)\right) \tag{2.18}$$

Donde hemos usado la relación 2.15 y el siguiente lema

Lema 1. *Si $\tilde{x}_1$ y $\tilde{x}_2$ son estadísticamente independientes, entonces, usando las relaciones*

$$P(\tilde{x}_1 \oplus \tilde{x}_2 = +1) = P(\tilde{x}_1 = +1)\cdot P(\tilde{x}_2 = +1) + (1 - P(\tilde{x}_1 = +1))\cdot(1 - P(\tilde{x}_2 = +1))$$

con

$$P(\tilde{x} = +1) = \frac{e^{\lambda(\tilde{x})}}{1 + e^{\lambda(\tilde{x})}}$$

Podemos expresar

$$\lambda(\tilde{x}_1 \oplus \tilde{x}_2) = \left(\frac{e^{\lambda(\tilde{x}_1)}}{1+e^{\lambda(\tilde{x}_1)}}\right)\cdot\left(\frac{e^{\lambda(\tilde{x}_2)}}{1+e^{\lambda(\tilde{x}_2)}}\right) + \left(1 - \left(\frac{e^{\lambda(\tilde{x}_1)}}{1+e^{\lambda(\tilde{x}_1)}}\right)\right)\cdot\left(1 - \left(\frac{e^{\lambda(\tilde{x}_2)}}{1+e^{\lambda(\tilde{x}_2)}}\right)\right)$$

$$\lambda(\tilde{x}_1 \oplus \tilde{x}_2) = \log\frac{1 + e^{\lambda(\tilde{x}_1)}e^{\lambda(\tilde{x}_2)}}{e^{\lambda(\tilde{x}_1)} + e^{\lambda(\tilde{x}_2)}}$$

Si expandimos esta expresión y agrupamos las exponenciales formando tanh *obtenemos*

$$\lambda(\tilde{x}_1 \oplus \tilde{x}_2) = \log\frac{(e^{\lambda(\tilde{x}_1)}+1)(e^{\lambda(\tilde{x}_2)}+1) + (e^{\lambda(\tilde{x}_1)}-1)(e^{\lambda(\tilde{x}_2)}-1)}{(e^{\lambda(\tilde{x}_1)}+1)(e^{\lambda(\tilde{x}_2)}+1) - (e^{\lambda(\tilde{x}_1)}-1)(e^{\lambda(\tilde{x}_2)}-1)}$$

$$= \log\frac{1 + \tanh(\lambda(\tilde{x}_1/2))\tanh(\lambda(\tilde{x}_2/2))}{1 - \tanh(\lambda(\tilde{x}_1/2))\tanh(\lambda(\tilde{x}_2/2))}$$

$$= 2\tanh^{-1}(\tanh(\lambda(\tilde{x}_1/2))\tanh(\lambda(\tilde{x}_2/2)))$$

El segundo término de 2.18 corresponde con el sumatorio de los mensajes enviados desde

los nodos de chequeo a los nodos de paridad. Podemos definir entonces el mensaje del nodo de chequeo m al nodo de bit n como

$$\eta_{m,n} = -2\tanh^{-1}\left(\prod_{j\in\mathcal{N}_{m,n}}\tanh\left(-\frac{\lambda(c_j|\{r_i, i\neq n\})}{2}\right)\right) \tag{2.19}$$

Permitiéndonos reescribir 2.18 de una forma más cómoda y compacta

$$\lambda(c_n|r) = L_c r_n + \sum_{m\in\mathcal{M}_n}\eta_{m,n} \tag{2.20}$$

La ecuación 2.20 puede verse como el mensaje que el nodo de bit n pasa a sus nodos de chequeo. Sin embargo, si iteramos usando estas dos ecuaciones en el algoritmo de decodificación, en 2.19 estaremos teniendo en cuenta en el nodo de chequeo m, la información que este mandó al bit n en la iteración anterior, con lo que debemos sacar esa información de la ecuación. La ecuación final de $\eta_{m,n}$ (donde el superíndice indica la iteración a la que nos referimos) quedará

$$\eta_{m,n}^{[l]} = -2\tanh^{-1}\left(\prod_{j\in\mathcal{N}_{m,n}}\tanh\left(-\frac{\lambda_j^{[l-1]} - \eta_{m,j}^{[l-1]}}{2}\right)\right) \tag{2.21}$$

Si volvemos a describir el funcionamiento del algoritmo teniendo en cuenta estas últimas consideraciones podremos entender con mayor facilidad las similitides entre la fórmula y el paso de mensajes a lo largo del grafo.

Supongamos que tenemos un nodo de bit c_n, que espera los mensajes de un único nodo de chequeo z_m. El nodo de chequeo, toma todos los mensajes de us nodos de bit en la iteracción anterior salvo el generado por c_n ($\prod_{j\in\mathcal{N}_{m,n}}$), evitando así contaminar la nueva información (evitando la dependencia de c_n consigo mismo, gevitando un bucle), y genera su mensaje. A este mensaje le elimina también su propia influencia, transmitida en la iteracción anterior a los nodos de bit ($-\frac{\lambda_j^{[l-1]}-\eta_{m,j}^{[l-1]}}{2}$), y finalmente lo manda a c_n ($\eta_{m,n}^{[l]}$), donde se computa junto con el resto de us nodos de chequeo ($\sum_{m\in\mathcal{M}_n}\eta_{m,n}$) y se chequea el resultado.

El algoritmo 2 muestra la implementación del algoritmo basado en la regla de la tanh.

Aunque el uso de LLR elimina las multiplicaciones y la normalización, nos topamos con otros problemas. Ahora se requieren la evaluación de tangentes hiperbólicas, por lo que sigue siendo un algoritmo poco práctico para su implementación hardware. Además, su implementación en software se ve afectada por el efecto de la precisión finita.

Algoritmo 2 Algoritmo "LLR-BP" basado en la regla de la tanh

Entrada: La matriz del código (A), el vector recibido **r**, el numero de iteraciones L y la fiabilidad del canal L_c

Inicialización: Hacer $\eta_{m,n}^{[0]} = 0$ para todo (m,n) con $A(m,n) = 1$
Hacer $\lambda_n^{[0]} = L_c r_n$
Hacer el contador de iteraciones $l = 1$

1: **mientras** $l < L$ y $A\hat{c} \neq 0$ **hacer**
2: **Actualizar los nodos de chequeo:** Para cada (m,n) con $A(m,n) = 1$, Calculamos:

$$\eta_{m,n}^{[l]} = -2\tanh^{-1}\left(\prod_{j\in\mathcal{N}_{m,n}} \tanh\left(-\frac{\lambda_j^{[l-1]} - \eta_{m,j}^{[l-1]}}{2}\right)\right)$$

3: **Actualizar los nodos de bit:** Para $n = 1, 2 \ldots N$, Calculamos:

$$\lambda_n^{[l]} = L_c r_n + \sum_{m\in\mathcal{M}_n} \eta_{m,n}^{[l]}$$

$\hat{c}_n = 1$ **si** $\lambda_n[l] > 0$ **o** $\hat{c}_n = 0$ **en otro caso.**
4: **fin mientras**
5: **si** $A\hat{c} = 0$ **entonces**
6: Palabra decodificada correctamente
7: **si no**
8: Se han producido error de decodificación
9: **fin si**

En que consiste exactamente y a qué se debe este problema de precisión finita? Si expresamos la tangente hiperbólica de 2.21 como

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Se ve claramente que para $x \to \pm\infty$, $\tanh(x) \to \pm 1$. Esto produce un fuerte impacto en la $\tanh^{-1}(x)$, que también interviene en 2.21, ya que cuando $x \to \pm 1$, $\tanh^{-1}(x) \to \pm\infty$.

Este efecto es frecuente cuando los decodificadores de LDPC actúan en la región de alta SNR. En esta región, los valores de la fiabilidad del canal de 2.14 (el término intrínseco) pueden superar en módulo a sus correspondientes valores extrínsecos que son producidos en las primeras iteraciones del algoritmo. Por lo tanto, los mensajes del nodo de bit con un valor elevado, hacen que en la actualización de los nodos de chequeo la $\tanh^{-1}$ se aproxime a infinito, con lo que se produce un desbordamiento en el decodificador.

Este fenómeno ha sido estudiado y verificado en [24], donde en la evaluación de los resultados aparece un suelo de error para BER bajas trabajando con la fórmula de 2.21.

En la realización de las simulaciones ya nos topamos con este problema, que puede ser fácilmente resuelto definiendo una modificación de la tanh mediante el truncamiento de

su argumento de la siguiente manera:

$$\tanh_{modif}(x) = \begin{cases} \tanh(x) & \text{si } |x| < x_0 \\ sign(x)\tanh(x_0) & \text{si } |x| \geq x_0 \end{cases} \tag{2.22}$$

Esto asegura que $-1 < \tanh_{modif}(x) < 1$, y por lo tanto, $-\infty \ll \tanh^{-1}(x) \ll +\infty$. El valor de x_0 será relativamente pequeño y positivo. En [24] se estudia el efecto de los diferente valores que puede tomar x_0, y por ensayo y error, se encuentra que los mejores resultados se encuentran para los valores $x_0 = 7$ y $x_0 = 3$. Nosotros tomaremos $x_0 = 7$, proporcionando $\tanh_{modif}(x_0) = 0{,}999998$. Esto añade complejidad computacional al algoritmo.

En la figura 2.5 podemos ver el nuevo valor que toma el argumento de la tanh.

Otra posible modificación introducida para atajar el problema de aproximación a infinito de la $\tanh^{-1}(x)$ similar a la anterior sería redefinirla como

$$\tanh^{-1}_{modif}(x) = \begin{cases} \tanh^{-1}(x) & \text{si } |x| < x_0 \\ sign(x)\tanh^{-1}(x_0) & \text{si } |x| \geq x_0 \end{cases} \tag{2.23}$$

Aseguramos así que $-\infty \ll \tanh^{-1}_{modif}(x) \ll \infty$. Ahora el valor de x_0 sería menor que 1 y positivo y por ensayo puede obtenerse su resultado. Por simetría con la anterior solución se toma $\tanh^{-1}(x_0) = 7$, que corresponde a $x_0 = 0{,}999998$. El aumento computacional es similar al de la solución anterior.

En la figura 2.6 se observa el efecto del suelo de error y la mejora al introducir la modificación de la tanh. Los mejores resultados se obtienen para este algoritmo por lo que en adelante referiremos a ellos todas las simulaciones de los demás algoritmos.

Hay que recalcar que estos resultados no quieren decir que el uso de una tangente no ideal nos permite obtener resultados superiores a los del algoritmo original, sino que ya que trabajamos con sistemas reales, debemos corregir sus imperfecciones (en esta caso la precisión finita) adaptando la ecuación.

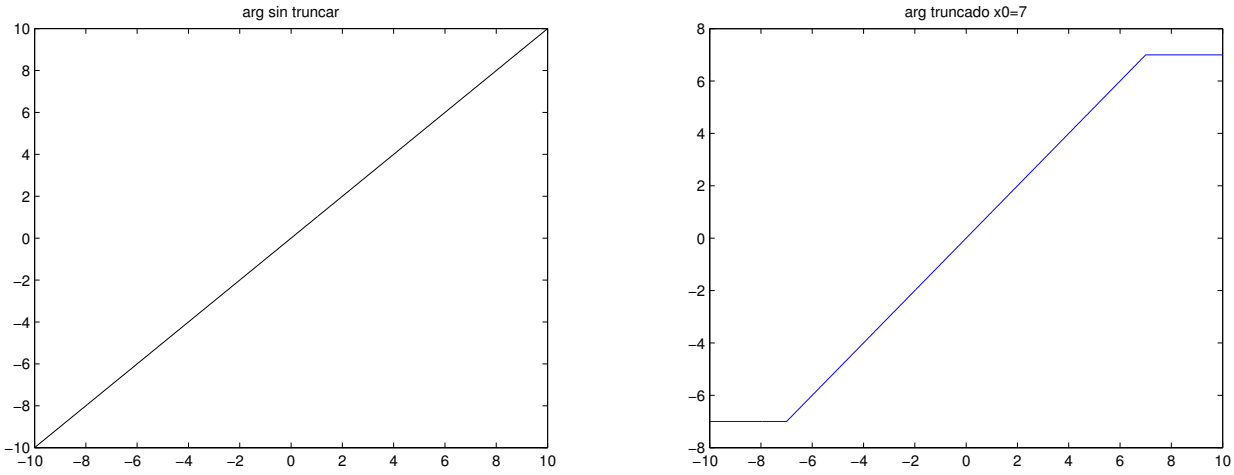

Figura 2.5: Argumento modificado de la tanh

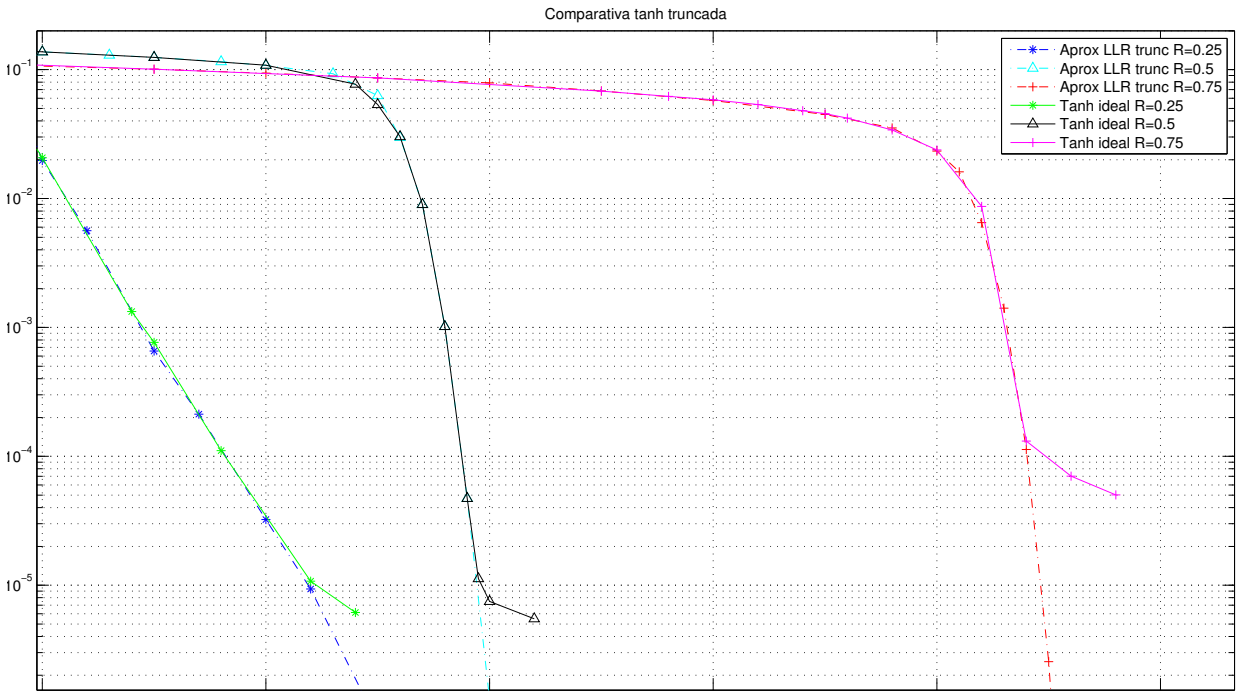

Figura 2.6: Comparativa LLR con tanh y tanh modificada N=64800bits, Nit=50

2.3.2 LLR-BP basado en el enfoque de Gallager

Este enfoque se basa en el uso de una representación alternativa de la regla de la tanh. A esta nueva ecuación se la conoce como "La Regla de Gallager" [13]:

$$L(U \oplus V) = sign(L(U))sign(L(V)) \times f(f(|L(U)|) + f(|L(V)|)) \tag{2.24}$$

donde la función $f(x) = \log((e^x + 1)/(e^x - 1))$, la ecuación de Gallager, es una transformada involutiva, es decir, una de sus propiedades es que $f(f(x)) = x$.

La única influencia de este cambio sobre las ecuaciones de los nodos se aprecia en los nodos de chequeo, quedando la nueva expresión como:

$$L_{m\to n}(x_n) = \left(\prod_{n' \in \mathcal{N}_{mn}} sign(Z_{n'\to m}(x_{n'})) \right) \times f \left(\sum_{n' \in \mathcal{N}_{mn}} f(|Z_{n'\to m}(x_{n'})|) \right) \tag{2.25}$$

El algoritmo puede ser implementado en software donde la los problemas de representación con precisión tienen un menor efecto. La transformación es primero realizada sobre todos los mensajes entrantes. Entonces todos los términos se suman y términos individuales se sustraen para obtener los pseudo mensajes de los que se obtienen los mensajes salientes por medio de la transformación involutiva. La implementación es simple, fácilmente paralelizable y por lo tanto recomendable para aplicaciones de muy alta velocidad. Sin embargo, la implementación hardware es muy problemática, ya que son sensibles a la cuantización y el redondeo.

2.3.3 LLR-BP basado en el enfoque Jacobiano

Como hemos visto, la regla de la tanh puede representarse como $L(U \oplus V) = log((1 + e^{L(U)+L(V)})/(e^{L(U)} + e^{L(V)}))$, que puede expresarse usando el logaritmo Jacobiano [11] dos veces como

$$\begin{aligned} L(U \oplus V) = {} & sign(L(U))sign(L(V)) \min(|L(U)|, |L(V)|) \\ & + \log\left(1 + e^{-|L(U)+L(V)|}\right) - \log\left(1 + e^{-|L(U)-L(V)|}\right) \end{aligned} \tag{2.26}$$

Considerando el nodo de chequeo m con d_c aristas desde los nodos de bit de $\mathcal{N}_m = (n_1, n_2, \ldots, n_{d_c})$. Los mensajes entrantes son $Z_{n_1 \to m}(x_{n_1}), Z_{n_2 \to m}(x_{n_2}), \ldots, Z_{n_{d_c} \to m}(x_{n_{d_c}})$. Definimos dos sets de variables binarias aleatorias $f_1 = x_{n_1}, f_2 = f_1 \oplus x_{n_2}, f_3 = f_2 \oplus x_{n_3}, \ldots, f_{d_c} = f_{d_c-1} \oplus x_{n_{d_c}}$ y $b_{d_c} = x_{n_{d_c}}, b_{d_c-1} = b_{d_c} \oplus x_{n_{d_c-1}}, \ldots, b_1 = b_2 \oplus x_{n_1}$. Aplicando 2.26 repetidamente obtenemos $L(f_1), L(f_2), \ldots, L(f_{d_c})$ y $L(b_1), L(b_2), \ldots, L(b_{d_c})$ de forma recursiva desde los mensajes entrantes. Usando que $(x_{n_1} \oplus x_{n_2} \oplus \cdots \oplus x_{n_{d_c}}) = 0$, obtenemos los $x_{n_i} = (f_{i-1} \oplus b_{i+1})$ para todo $i \in 2, 3, \ldots, d_c - 1$. Por lo tanto, el mensaje saliente desde el nodo de chequeo m será

$$L_{m \to n_i}(x_{n_i}) = \begin{cases} L(b_2) & \text{si } i = 1 \\ L(f_{i-1} \oplus b_{i+1}) & \text{si } i = 2, 3, \ldots, d_c - 1 \\ L(f_{d_c-1}) & \text{si } i = 2, 3, \ldots, d_c - 1 \end{cases} \tag{2.27}$$

Esto es esencialmente el algoritmo "forward-backward" aplicado a la trellis de un SPC (single parity-check code), requiriendo $3(d_c - 2)$ computaciones de la operación $L(U \oplus V)$ en 2.26 por cada actualización de los nodos de chequeo. La función $g(x) = \log(1 + e^{-|x|})$ puede implementarse usando una LUT o una aproximación lineal a trozos. Por lo tanto, la operación $L(U \oplus V)$ puede realizarse con 4 sumas, una comparación y 2 correcciones (mirando la LUT o con una evaluación lineal).

En este proyecto no estudiaremos este último enfoque pero lo proponemos como una posible ampliación.