Temas Avanzados en Códigos para Corrección de Errores

Tomás Gonzalez - CI: 5.164.667-3
Marcos Rapetti - CI: 5.273.409-5

16 de Febrero de 2025

Índice

1. Desarrollo teórico

Cantidad de bits de $u(l)$

Sea 0^m un bloque de m bits con todos sus valores nulos, y $u(l) = [0^m \cdots 0^m\, 1^m\, 0^m \cdots 0^m]$ formado por l bloques nulos, uno de unos y $j-l-1$ bloques nulos. De esta forma, $u(l)$ estará formado por j bloques de m bits, por lo que se cumple:

$$\#bits = j \times m = r$$

Cálculo de $u(l)H$

Notamos que el conjunto $\{u(l) \,/\, 0 \leq l \leq j-1\}$ es LI. Se puede ver que dado $0 \leq l_i \leq j-1$, $u(l_i)$ es el único con elementos no nulos en el bloque l_i. Por lo que cualquiera sea la combinación lineal de los otros $u(l)$, siempre el bloque l_i será el nulo y es imposible obtener a $u(l_i)$.

Sea $H \in M_{r\times n}$ matriz de chequeo de paridad. En sus columnas, H está formada por j bloques de m bits, donde cada bloque contiene solamente un 1 y son 0 el resto de sus elementos. Al multiplicar $u(l)H$ solamente participará el bloque l de todas las columnas de H. Quedando así:

$$\begin{aligned} u(l) \times H_i &= 1 \quad \forall \text{ columna } i \\ u(l) \times H &= [1\ 1 \cdots 1] \\ &= 1^n \end{aligned}$$

Vemos que el resultado obtenido es independiente de el valor de l. De esta forma se podrá calcular:

$$\begin{aligned} [u(0) + u(l)] \times H &= u(0) \times H + u(l) \times H \\ &= 1^n + 1^n \\ &= 0^n \quad \text{en GF(2)} \end{aligned}$$

Entonces concluimos que $[u(l_1) + u(l_2)] \in \text{Ker}(H)$. En particular, si fijamos $l_1 = 0$, y con un razonamiento análogo al que hicimos anteriormente, vemos que el conjunto $\{u(0) + u(l) / \ 0 < l \leq j-1\}$ de $j-1$ elementos es LI.

Determinación de rank(H)

Como $H \in M_{r\times n}$, y por propiedades del Álgebra Lineal, sabemos que se cumple:

$$\dim\big(\text{Ker}(H)\big) + \text{rank}(H) = r \tag{1}$$

De la parte anterior, tenemos una cota inferior laxa para la dimensión del núcleo de H:

$$j-1 \leq \dim\big(\text{Ker}(H)\big) \tag{2}$$

A partir de las ecuaciones 1 y 2 obtenemos:

$$\begin{aligned} j-1 + \text{rank}(H) &\leq r \\ \text{rank}(H) &\leq r-j+1 \end{aligned} \tag{3}$$

2. Análisis de resultados

```
ldpc_Utils/bbchannel -s 314 -p 0.5 -Z 32000000 testdata.dat
./ldpc_decode -i testdata.dat -o testdata_corr.dat -n 20
ldpc_Utils/diffblock -m 8 -z -n 1600 testdata_corr.dat
```

Listing 1: Ejemplo de comandos para generar el mensaje ruidoso y decodificarlo con `ldpc_decode`.

```
./test_all.sh
```

Listing 2: Comando para correr experimentos para todas las probabilidades y numero de iter maximo.

```
https://github.com/tomas-gr/ldpc_iterative_decoding
```

Listing 3: Link a un repositorio de github que contiene el decodificador.

Lo esperado es que para bajas cantidades de iteraciones máximas, las tasas de error aumenten. Podemos ver en la figura 1 que esto se cumple para valores de probabilidad de error de bit en canal entre 0.05 y 0.09. Para probabilidad 0.04 vemos que no afecta el número máximo de iteraciones. Esto podría darse debido a que los errores producidos por este canal se corrigen en menos de 10 iteraciones para la mayoría de los casos. Para probabilidades de error mayores a 0.1, vemos que no es posible corregir ningun bloque.

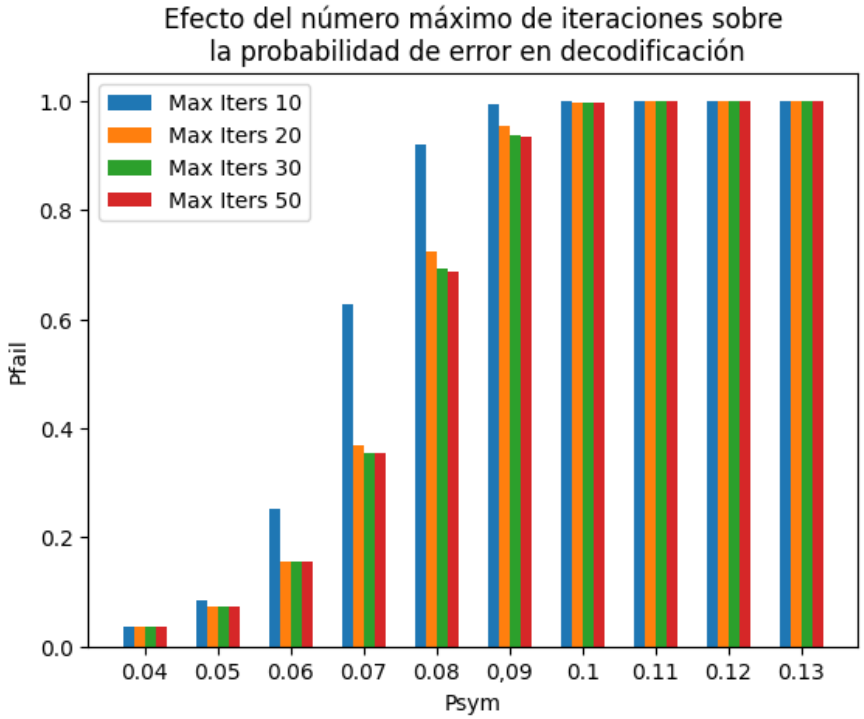

Figura 1: Comparación de probabilidad de error por bloque para 20 iteraciones máximas (azul) y 50 (naranja).

Tenemos que la capacidad del canal BSC está dada por:

$$C = 1 - H_2(p)$$

donde la entropía binaria $H_2(p)$ se define como:

$$H_2(p) = -p \log_2 p - (1-p) \log_2 (1-p)$$

Por lo que para probabilidad de error $p_{sym} = 0{,}1$

$$H_2(0{,}1) = -0{,}1 \log_2 0{,}1 - (0{,}9) \log_2 (0{,}9) C = 1 - H_2(0{,}1) = 0{,}5310044064$$

Luego podemos calcular la tasa del código LDPC como $\frac{k-n}{n} = 0{,}25$

Por lo que con este código nos estamos pudiendo aproximar hasta la mitad de la capacidad de canal aproximadamente.

3. Comparación con BCH

Se trabajará con un código BCH [1600, 401, 219], con redundancia $r_{BCH} = n - k = 1199$ bytes. Vemos que para el código LDPC tenemos la cota 3 que nos indica que $r_{LDPC} \leq 1198$. Es decir, que la redundancia en el código BCH es estrictamente mayor a la del código LDPC.

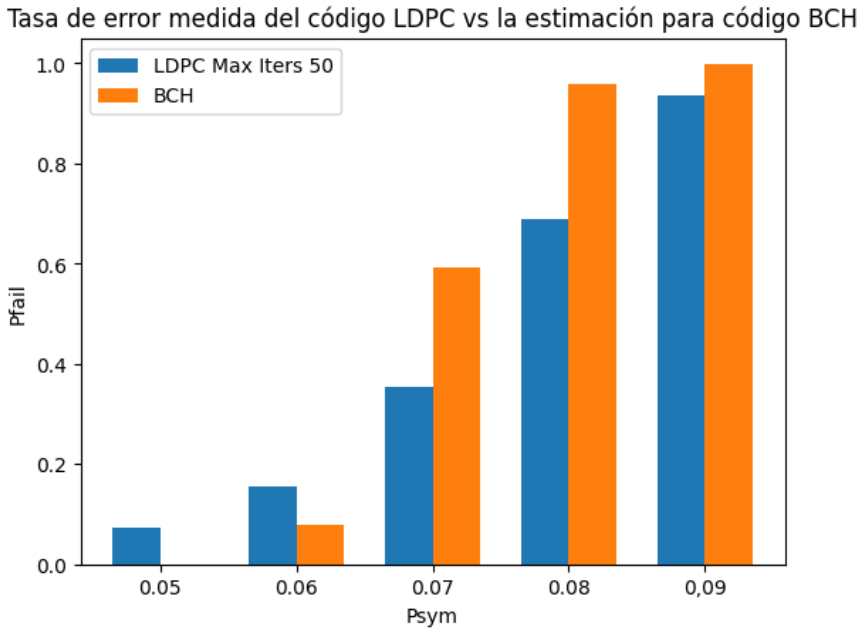

Figura 2: Comparación de probabilidad de error por bloque para decodificación BCH (azul) y LDPC (naranja) con 50 iteraciones máximas.

Como se puede ver en la Figura 2 el crecimiento de la probabilidad de error en bloque para los códigos BCH y LDPC son muy distintos. En el primero, se tiene una probabilidad de error cercana a 0 para $p = 0,05$, pero que rápidamente se incrementa, llegando a probabilidades cercanas a 1. Sin embargo, en el código LDPC tenemos que la probabilidad de error es mayor para P_{sym} bajas $(0{,}05, 0{,}06)$. Sin embargo a medida que aumenta P_{sym} la probabilidad del código LDPC aumenta más lento que para BCH. Diremos entonces que será más conveniente el código BCH para tasas de error pequeñas, mientras que el código LDPC lo será para tasas mayores. En este caso para probabilidades de error del canal BSC mayores a 0.1, ambos códigos dejan de poder recuperar los mensajes.

4. Documentación de funciones

convertirADispersa Input: H. Output: H_cols, H_rows. Sintetiza la información de H en dos matrices H_cols y H_rows, devueltas por referencia. Estas indican la posición de los 1 en las columnas y filas de H respectivamente.

chequeoParidad Input: H_rows, palabra decodificada. Output: Booleano. Verifica que el síndrome de la palabra decodificada es nulo.

ldpc_decode_unanimity Input: H_rows, H_cols, iteraciones máximas, mensaje recibido. Output: Mensaje decodificado y booleano de éxito. Si hay un error en el mensaje (síndrome no nulo), itera hasta decodificar correctamente la palabra. Devuelve una tupla conteniendo

el mensaje decodificado y un booleano indicando si el proceso fue exitoso o no.

corregir_archivo Input: Archivo de entrada, H_cols, H_rows, iteraciones máximas. Output: Archivo de salida. Corrige los errores de canal del archivo de entrada y guarda los resultados en el archivo de salida. Imprime en la consola estadísticas de la decodificación al procesar todo el archivo.

5. Referencias

Notas del curso.
Gallager, R. Lower-Density Parity-Check Codes (1993).