Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Matemáticas

PRODUCTO DE CÓDIGOS REED-SOLOMON

Autor: Pablo Ibarlucea González
Tutor: Diego Ruano Benito
Año 2023

ÍNDICE GENERAL

INTRODUCCIÓN

En este trabajo, nos sumergimos en la teoría de códigos correctores, que son códigos utilizados para tratar la información de una manera que permita corregir posibles errores, esto se consigue añadiendo información adicional derivada de la original. El estudio estará focalizado en los códigos Reed-Solomon, una familia de códigos ampliamente utilizados en aplicaciones cotidianas como DVD y códigos QR, lo que subraya su relevancia práctica [1] [2].

Comenzamos introduciendo los aspectos más básicos de la teoría de códigos correctores en el capítulo 1. Dado que es una asignatura del grado, no se entra en gran detalle, sin embargo se incluye con la intención de crear un texto lo más autocontenido posible. Además debido a que personalmente no cursé esa asignatura, estos contenidos son parte del estudio que se ha realizado para este trabajo.

Seguiremos, en el capítulo 2, presentando los códigos de Reed-Solomon. Como veremos, estos códigos son el resultado de evaluar polinomios en unos valores determinados. Exploraremos sus propiedades esenciales y un algoritmo de decodificación simple. Además, en el capítulo 3, examinamos la clase más común dentro de esta familia: los códigos Reed Solomon cíclicos y cómo están conectados con los códigos BCH.

Luego, nos dedicamos, durante el capítulo 4, a un algoritmo de decodificación más eficiente desde el punto de vista algorítmico: el algoritmo de Berlekamp-Massey. Durante este capítulo, para entender este algoritmo se presenta otro relacionado, el algoritmo de Peterson; además se definen las recurrencias lineales y se obtienen resultados sobre ellas.

Continuamos, en el capítulo 5, analizando algoritmos que buscan una capacidad de corrección superior a la proporcionada por la distancia mínima, específicamente los algoritmos de decodificación con lista. A través de este estudio, también examinamos las circunstancias en las que es factible lograr una decodificación única. Este enfoque ampliado aporta una comprensión más profunda de la estructura inherente a un código corrector. Para los códigos de Reed-Solomon se detallan los algoritmos de Sudan y de Sudan-Guruswami.

Sin embargo, el enfoque final del trabajo, se dirige hacia los códigos producto, que cambian la perspectiva secuencial con la que se suele tratar la información disponiéndola en cambio de manera matricial y aplicando un código a las filas y otro a las columnas. En el capítulo 6, describimos sus propiedades y destacamos dos algoritmos para su decodificación: uno basado en la decodificación con lista y otro a través de un enfoque iterativo. En

este capítulo veremos también los detalles de cómo se utilizan estos códigos en dispositivos de almacenamiento óptico (CD,DVD).

Una de las motivaciones principales para la elección de este tema es su utilidad práctica, lo que ha permitido realizar este trabajo en conjunto con el correspondiente del grado en ingeniería informática. Por ello dedicamos el último capítulo, el capítulo 7, a comentar las implementaciones de los algoritmos presentados que se han utilizado a lo largo del texto para proporcionar ejemplos. Además en este capítulo se incluye también un estudio de la capacidad correctora del algoritmo iterativo de manera experimental. Para estas implementaciones se ha utilizado `python` con la libreria `galois` [3] que simplifica los cálculos en cuerpos finitos.

La referencia básica para este trabajo ha sido la segunda edición del libro *A course in error-correcting codes* de Justesen y Høholdt [4], principalmente en los capítulos 4, *Reed-Solomon Codes and Their Decoding*; 5, *Cyclic Codes*; y 9, *Decoding Reed-Solomon and BCH Codes*. Este libro ha sido esencial para introducirme en el tema y creo que merece una mención especial. Por otra parte en cada sección especifica se mencionan las publicaciones originales y otras fuentes de referencia, que consisten en artículos y tesis doctorales, que han sido necesarias para profundizar en los temas tratados.

Me gustaría acabar la introducción en una nota más personal, exponiendo cómo fue surgiendo la necesidad de incluir cada uno de los capítulos. Después de finalizar este trabajo puedo decir que he pasado de poder dar una definición imprecisa de lo que son los códigos correctores a, creo, entenderlos suficientemente como para poder afrontar el estudio de cualquier tema específico dentro de este área. Para estudiar los códigos producto, he tenido que pasar por conceptos necesarios para ello, y a su vez por conceptos necesarios para estos últimos, etc. En este proceso he aprendido mucho no solo del producto de códigos Reed-Solomon, si no, como digo, lo necesario para encarar con seguridad la literatura sobre códigos correctores en general. Con este texto tengo la intención de que un posible lector me siga en el itinerario que fui descubriendo y pueda llegar al punto en el que estoy yo, ahorrándole algún esfuerzo.

Todo empezó con la idea de replicar una presentación visual de la codificación usada en DVDs, debida a Høholdt y publicada en su web [5], pero que lamentablemente ya no está disponible. En esta página se podía aplicar el algoritmo de decodificación repetidamente sobre una imagen corrupta, mejorando cada vez el resultado. Esta es la idea del algoritmo iterativo, y motiva las preguntas sobre su capacidad correctora.

Para entender mejor los códigos producto, comencé con una tesis que proponía un algoritmo basado en decodificar con lista [6]. Esto justifica la necesidad del capítulo quinto,

donde se presentan estos algoritmos.

Por otra parte, un algoritmo de decodificación que estaba disponible en la librería utilizada para la implementación del algoritmo iterativo es el de Berlekamp-Masey, así que me propuse entenderlo antes de usarlo, lo que dio lugar al cuarto capítulo.

Observé también que estos algoritmos en ocasiones se mencionaban como algoritmos de decodificación de códigos BCH en vez de algoritmos de decodificación de Reed-Solomon. ¿Qué eran los códigos BCH y qué relación tenían con los códigos Reed-Solomon? Esta pregunta fue el germen del tercer capítulo, los códigos Reed-Solomon cíclicos, que concluye relacionándolos con los códigos BCH y estudiando alguna de las propiedades de estos. Para esto fue necesario también estudiar conjuntos ciclotómicos.

1. RESULTADOS GENERALES DE CÓDIGOS CORRECTORES

En este cápitulo presentamos definiciones y resultados básicos sobre códigos correctores con el fin de que el documento sea autocontenido y preciso. Sin embargo no se incluye la prueba de alguno de los resultados enunciados ya que existe una asignatura del grado con estos contenidos (Códigos Correctores). El propósito del capítulo es servir como base y no ser exhaustivo.

1.1. Definiciones básicas

El escenario que nos vamos a plantear recurrentemente es el de una **palabra** de n símbolos que se recibe a través de un canal no fiable, que puede introducir errores. El objectivo del código corrector es permitir la corrección de estos errores. Los símbolos serán elementos del **alfabeto** Σ.

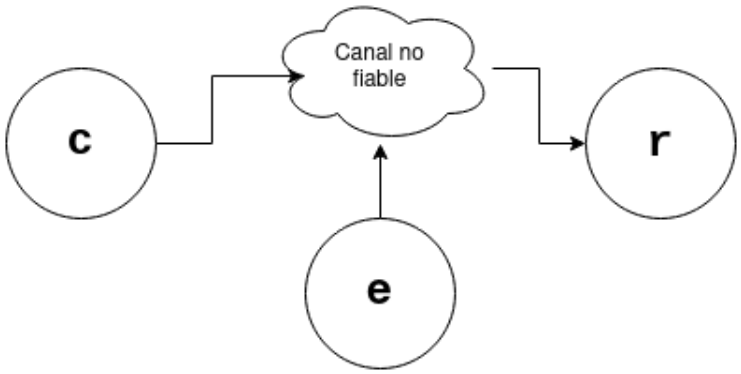


Figura 1.1: Contexto de los códigos correctores. Se envía una palabra c y se producen errores en la comunicación para resultar en la palabra recibida r.

Definición 1. (Código Corrector) Un código corrector C de **longitud** n, sobre un alfabeto Σ es un subconjunto de Σ^n

$$C \subseteq \Sigma^n$$

Un código corrector no es más que un subconjunto de todas las posibles palabras

A la hora de codificar se consideran bloques de k símbolos de **información**, y se necesita una función que envíe vectores de longitud k a palabras del código (de manera inyectiva), una función de este tipo es una **codificación**. Si en los primeros k símbolos se mantienen los símbolos de información decimos que tenemos una **codificación sitemática**. En este

caso los restantes $n - k$ símbolos los llamamos símbolos de **redundancia** o **paridad**.

Esto permite la corrección de errores ya que la palabra enviada es una de las del código y en caso de errores la palabra recibida quedará fuera de este, si el error es razonablemente pequeño la palabra enviada se parecerá a la palabra recibida, es decir se encontrará *cercana* a la recibida y podemos corregir tomando la palabra más cercana.

Para determinar cuál es la más cercana, la distancia más comunmente usada en este contexto es la **distancia de Hamming**.

Definición 2. (Distancia de Hamming) Sean $a, b \in \Sigma^n$. La distancia de Hamming entre a y b, $d(a, b)$, es el número de índices en que difieren.

$$d(a, b) = |\{i \in \{1, 2, \ldots, n\} : a_i \neq b_i\}|$$

Ejemplo 1. Para hacer que un mensaje resista errores podemos añadir redundancia, la manera más directa de hacer esto es repetir varias veces el mensaje. Esto resulta en el tipo más sencillo de código, el que sus palabras son las que tienen el mismo símbolo en todas sus coordenadas.
Consideremos el código $C = \{(0, 0, 0), (1, 1, 1)\} \subset \mathbb{F}_2^3$, si recibimos la palabra $r = (0, 1, 1)$ podemos comprobar que $d(r, (0, 0, 0)) = 2$ y $d(r, (1, 1, 1)) = 1$ luego concluimos que se envió la palabra $(1, 1, 1)$.
Esto es equivalente a comprobar qué símbolo tienen la mayoría de coordenadas.
Usaremos este código para ejemplificar los conceptos expuestos en lo siguiente.

Los códigos más utilizados son aquellos construidos sobre cuerpos finitos, es decir aquellos para los que su alfabeto es un cuerpo finito. En estos casos también se tiene la noción de **peso de Hamming**, que sería la norma asociada a la distancia de Hamming.

Definición 3. (Peso de Hamming) Sean $a \in \mathbb{F}_q^n$. El peso de Hamming de a, $w(a)$, es el número de elementos no nulos de a.

$$w(a) = d(a, 0) = |\{i \in \{1, 2, \ldots, n\} : a_i \neq 0\}|$$

Se suele usar este concepto a la hora de hablar del vector de errores e que es la diferencia entre la palabra recibida r y la palabra enviada c. Vemos por lo tanto en la figura 1.1 la expresión $r = c + e$, que utilizaremos recurrentemente.

Ejemplo 2. En el ejemplo anterior o bien se introdujo 1 error a la palabra enviada, con lo que el vector de errores sería $e = (1, 0, 0)$ que tiene peso de Hamming $w(e) = 1$; o bien se introdujeron 2 errores y se corrigió de manera incorrecta ya que la mayoría de simbolos serían incorrectos, en este caso $e = (0, 1, 1)$ y $w(e) = 2$.

1.2. Códigos lineales

Los códigos más comunes y los más útiles son una subclase de esta definición tan general. Se toma como alfabeto un cuerpo finito $\mathbb{F}_q$ y los códigos correctores se construyen como subespacios vectoriales de $\mathbb{F}_q^n$. En lo siguiente trataremos exclusivamente con esta clase de códigos.

Definición 4. (Código Corrector Lineal). Un código corrector lineal (n, k) sobre un cuerpo $\mathbb{F}_q$ es un subespacio vectorial de dimensión k del espacio vectorial $\mathbb{F}_q^n$. Decimos que k es tambíen la dimensión del código y coincidiendo con la definición general, n es la longitud del código.

Ejemplo 3. C es lineal, es un subespacio de dimensión 1 de $\mathbb{F}_2^3$ generado por $(1, 1, 1)$.

1.3. Distancia mínima

Como ya hemos dicho la idea general detrás de la corrección es buscar la palabra del código que diste menos de la palabra recibida, por ello para tener buenas propiedades de corrección las palabras del código deben destar *separadas* unas de otras. Con esta idea definimos la siguiente propiedad:

Definición 5. (Distancia Mínima) La distancia mínima d, de un código C es la mínima distancia (de Hamming) entre dos palabras cualesquiera.

$$d = \text{mín}\,\{d(a, b) : a, b \in C\}$$

Esto significa que dos palabras del código cualesquiera diferirán en al menos d posiciones.

Usamos la notación $C[n, k, d]$ para referirnos a un código corrector con longitud n, dimensión k y distancia mínima d, resumiendo así sus parámetros.

> **Ejemplo 4.** La distancia mínima del código de los ejemplos anteriores es 3, en general el código resultante de repetir n veces un símbolo es un código $[n, 1, n]$.

La importancia de la distancia mínima reside en que de ella depende el número de errores que puede corregir.

Teorema 1. *(Capacidad correctora) Un código con distancia mínima d puede detectar hasta* $d - 1$ *errores y corregir* $t = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Esto último se debe a que dos bolas centradas en palabras del código de radio t tienen que ser disjuntas por virtud de la distancia mínima, por lo tanto si se producen t errores la palabra recibida pertencerá a una única bola de este radio, la centrada en la palabra enviada.

Decimos que t es la **capacidad correctora** y que el código es **t-corrector**.

Las propiedades de linealidad de un código nos permiten calcular de una manera más sencilla el valor de d:

Teorema 2. *Si C es un código lineal y d su distancia minima, entonces*

$$d = \text{mín}\{w(c) = d(c, 0) : c \in C \setminus \{0\}\}.$$

Es decir la distancia mínima es el menor de los pesos (de Hamming) de todas las palabras no nulas del código.

La distancia mínima depende a su vez de la longitud y dimensión del código. Podemos dar una cota superior en función de n y k, la cota de Singleton:

Teorema 3. *(La cota de Singleton) Sea* $C[n, k, d]$ *un código, se cumple*

$$d \leq n - k + 1$$

Vamos a probar esta cota, sea $C \subset \mathbb{F}_q^n$, consideremos la siguiente aplicación lineal entre dos espacios vectoriales:

$$\begin{aligned} f : C &\to \mathbb{F}_q^{n-d+1} \\ (c_1, c_2, \ldots, c_n) &\mapsto (c_1, c_2, \ldots, c_{n-d+1}). \end{aligned}$$

Es decir la aplicación *trunca* las palabras del código, elimininando los últimos $d-1$ símbolos. Como todas las palabras se diferencia en al menos d símbolos entonces la aplicación es inyectiva. Esto significa que la dimensión del espacio de salida tiene que ser menor o igual que la dimensión del espacio de llegada, es decir, $k \leq n - d + 1 \implies d \leq n - k + 1$.

Ejemplo 5. Para C la cota de Singleton es una igualdad, en este caso decimos que un código es MDS, *Maximum distance separable*.

1.4. Matriz generadora y matriz de control

Una manera de obtener una codificación es interpretando los k símbolos como las coordenadas en una base del código (que recordamos es espacio vectorial). Este enfoque se ve en las matrices generadoras:

Definición 6. Matriz generadora. Una matriz generadora G de un código $C[n, k]$, es una matriz (k, n) cuyas filas forman una base del código (son linealmente indeprendientes).

Obtenemos una codificación a partir de la matriz generadora de la siguiente manera:

$$\begin{aligned} \mathbb{F}_q^n &\to C \\ u &\mapsto uG. \end{aligned}$$

Una matriz generadora que resulte en una codificación sistemática será de la forma

$$G = (I, A)$$

siendo I la matriz identidad (k, k) y A una matriz $(k, n - k)$. Todo código lineal, salvo permutación, tiene una matriz generadora sistemática.

Ejemplo 6. La matriz generadora de C es simplemente

$$G = (1, 1, 1)$$

Y la codificación dada (que es sistemática) es la que repite el símbolo de mensaje 3 veces, es decir $a \in \mathbb{F}_2 \mapsto aG = (a, a, a)$.

Ahora nos centramos en el problema de comprobar si una palabra pertenece al código. El subespacio de dimensión k en el espacio de dimensión n se puede definir mediante un conjunto de $n-k$ ecuaciones implícitas. Estas ecuaciones se pueden poner en forma matricial:

Definición 7. Matriz de control. Una matriz de control H para un código $C[n,k]$ es una matriz $(n-k,n)$ tal que

$$c \in C \iff Hc^T = 0$$

Es decir, las filas de la matriz de control forman una base del espacio ortogonal al código.

Ejemplo 7. Una palabra (x_1, x_2, x_3) de C cumple

$$x_1 = x_2$$

$$x_1 = x_3$$

Luego una matriz de control sería

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Nos interesa también conocer el valor Hr^T cuando no es 0, en este caso se ha introducido un error, $r = c + e$, y entonces

$$H(c+e)^T = Hc^T + He^T = He^T.$$

Es esta la propiedad interesante, este valor no depende de la palabra original, solo del error y se puede utitlizar para caracterizarlo en los casos en que sea corregible. Por ese motivo es clave en muchos métodos de decodificación. Este valor es el síndrome:

Definición 8. Sindrome. Sea H una matriz de control para un código $C[n,k]$ y $r \in \mathbb{F}_q^n$. El síndrome de r viene dado por

$$s = Hr^T$$

Ejemplo 8. Veamos que los síndromes caracterizan los errores corregibles por C.
Los tres errores corregibles (C es 1-corrector) son $e_1 = (1,0,0)$, $e_2 = (0,1,0)$, $e_3 = (0,0,1)$ y sus síndromes son

$$He_1^T = (1,1)^T$$

$$He_2^T = (1,0)^T$$

$$He_3^T = (0,1)^T$$

Usando el síndrome se puede corregir. Si recibimos la palabra $r = (0,1,1)$ vemos que su síndrome es $Hr^T = (1,1)$ y concluimos que el error que se ha producido ha de ser e_1, es decir $c = r - e_1 = (1,1,1)$.

1.5. Códigos de Hamming

Introducimos este tipo de códigos para ilustrar con un ejemplo no trivial, pero sencillo, los conceptos definidos en el capítulo.

Los códigos de Hamming deben su nombre a R.W. Hamming que en los describió en 1950, en la publicación *Error Detecting and Error Correcting Codes* para *Nokia Bell Labs* donde trabajaba. [7]. En esta publicación también se define originalmente la distancia de Hamming.

Estos códigos son binarios, construidos sobre $\mathbb{F}_2$. La idea es añadir símbolos (bits) que sean la suma de símbolos en otras posiciones, a esto se le llaman bits de paridad. Se eligen las posiciones que cubre cada uno de los bits de paridad de manera que el error pueda ser caracterizado. Trataremos con el caso concreto $[7, 4]$, dando una codificación sistemática.

En este caso se tienen 3 bits de paridad p_1, p_2, p_3 para 4 bits de información a_1, a_2, a_3, a_4. Se calculan los bits de paridad de la siguiente manera

$$p_1 = a_1 + a_2 + a_4$$
$$p_2 = a_1 + a_3 + a_4$$
$$p_3 = a_2 + a_3 + a_4$$

Esto permite la correción ya que si solo uno de los bits de paridad es incorrecto en la palabra recibida, ese bit contiene el error; en otro caso si más de un bit de paridad es incorrecto se puede determinar qué bit de información contiene el error.

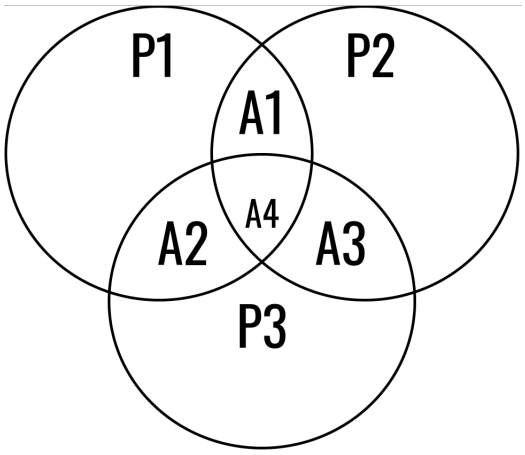


Figura 1.2: Qué bit contiene el error según resultados incorrectos para más de un bit de paridad.

Ejemplo 9. Se recibe la palabra $(a_1, a_2, a_3, a_4, p_1, p_2, p_3) = (1, 1, 1, 1, 0, 0, 1)$. Se calculan los bits de paridad a partir de las 4 primeras posiciones de información y se comparan con los valores recibidos en las 3 últimas.

$$\begin{aligned} p_1 = a_1 + a_2 + a_4 &= 1 + 1 + 1 = 1 \\ p_2 = a_1 + a_3 + a_4 &= 1 + 1 + 1 = 1 \\ p_3 = a_2 + a_3 + a_4 &= 1 + 1 + 1 = 1 \end{aligned} \tag{1.1}$$

con lo que hay discrepancia en los bits p_1, p_2 y de esto podemos concluir que se ha producido un error en el bit de información a_1. La palabra corregida es $(0, 1, 1, 1, 0, 0, 1)$.

El código es lineal ya que los bits de paridad de la suma de dos palabras será la suma de los bits de paridad de cada una.

La distancia mínima es 3 ya que cambiar un bit de información hace que cambien al menos 2 bits de paridad, $(d \geq 3)$; y existen palabras de peso 3 como $(1, 0, 0, 0, 1, 1, 0)$, $(d \leq 3)$. Entonces el código es 1-corrector, como hemos podido comprobar.

Vamos a calcular la matriz generadora. La codificación de los vectores de información $(1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1)$ resultará en una base del código, en las filas de una matriz generadora:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Y podemos obtener una matriz de control simplemente expresando las ecuaciones 1.1 de manera matricial:

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

De esta manera el síndrome indicará qué bits de paridad son incorrectos, que como ya hemos visto caracteriza el error.

Ejemplo 10. Se recibe la palabra $r = (1, 1, 1, 1, 0, 0, 1)$. Calculemos su síndrome.

$$Hr^T = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

Esto nos dice que hay un error en las ecuaciones para el primer y segundo bit de paridad y podemos determinar el error. Podemos comprobar que este síndrome coincide con el síndrome del error:

$$H(1, 0, 0, 0, 0, 0, 0)^T = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

Y por tanto de decodifica r por $(0, 1, 1, 1, 0, 0, 1)$.

2. CÓDIGOS REED SOLOMON

A lo largo este capítulo seguiremos [4] para presentar los códigos de Reed-Solomon. Estos códigos fueron introducidos por Irving S. Reed y Gustave Solomon en 1960 en el artículo *Polynomial codes over certain finite fields* [8], su uso es muy extendido y se han estudiado ampliamente desde esta definición original.

2.1. Definición y parámetros del código

Definición 9. (Código Reed-Solomon) Sean $x_1, \ldots, x_n$ elementos distintos de un cuerpo finito $\mathbb{F}_q$. Sea $\mathbb{P}_k$ el espacio de polinomios con coeficientes en $\mathbb{F}_q$ de grado menor que $k \leq n$. Entonces el código Reed-Solomon $[n, k]$, que denotaremos como $RS[n, k]$, consiste en las palabras

$$\{(f(x_1)), \ldots, f(x_n)) : f \in \mathbb{P}_k\}$$

Es decir, los mensajes son los polinomios de $\mathbb{P}_k$ que se codifican mediante la aplicación de evaluacion ev. Si tenemos un mensaje consistente en k símbolos de $\mathbb{F}_q^k$, $(a_0, \ldots, a_{k-1})$, podemos asociarlo al polinomio $a_0 + a_1 x + \ldots a_{k-1} x^{k-1}$ y codificar este.

$$\begin{aligned} ev\colon \mathbb{P}_k &\to \mathbb{F}_q^n \\ f &\mapsto (f(x_1)), \ldots, f(x_n)). \end{aligned}$$

Esta aplicación es claramente lineal, por lo tanto el código, que se correponde con su imagen también lo es.

Además $Ker(ev) = \{0\}$ ya que por el teorema fundamental del álgebra sabemos que un polinomio no nulo de grado menor que k tiene a lo sumo $k - 1$ raíces, luego su imagen tiene a lo sumo $k - 1$ ceros y en particular es no nula. Esto nos permite concluir que la aplicación es inyectiva. Se tiene que $RS[n, k] = Im(ev) \cong \mathbb{P}_k$, y que la dimensión del código es efectivamente k, ya que $\mathbb{P}_k$ es un espacio vectorial de dimensión k (tomando por ejemplo la base $\{1, x, x^2, \ldots, x^{k-1}\}$), y la aplicación de evaluación es inyectiva.

Siendo $RS[n, k]$ lineal, la distancia mínima d es el menor de los pesos (de Hamming) de todas las palabras no nulas del código (Ver teorema 2). Cómo hemos visto, una palabra no nula tiene a lo sumo $k - 1$ ceros, por lo tanto su peso es como mínimo $n - (k - 1)$, esto es $d \geq n - (k - 1)$. Pero por la cota de Singleton concluimos que la distancia mínima es exactamente esa. $d = n - (k - 1)$. Los códigos Reed-Solomon alcanzan esta cota y son

óptimos en este sentido (MDS, *Maximum distance separable*).

Ejemplo 11. Consideremos el código $RS[3,2]$ ccon elementos en $\mathbb{F}_4$ y α un elemento primitivo con $\alpha^2 + \alpha + 1 = 0$. Para cada polinomio con grado menor que 2, evaluamos en $1, \alpha, \alpha + 1$:

Polinomio	1	α	$1+\alpha$	Palabra
0	0	0	0	$(0,0,0)$
1	1	1	1	$(1,1,1)$
α	α	α	α	(α,α,α)
$1+\alpha$	$1+\alpha$	$1+\alpha$	$1+\alpha$	$(1+\alpha,1+\alpha,1+\alpha)$
x	1	α	$1+\alpha$	$(1,\alpha,\alpha+1)$
$1+x$	0	$1+\alpha$	α	$(0,1+\alpha,\alpha)$
$\alpha+x$	$1+\alpha$	0	1	$(\alpha+1,0,1)$
$1+\alpha+x$	α	1	0	$(\alpha,1,0)$
αx	α	$1+\alpha$	1	$(\alpha,\alpha+1,1)$
$1+\alpha x$	$1+\alpha$	α	0	$(1+\alpha,\alpha,0)$
$\alpha+\alpha x$	0	1	$1+\alpha$	$(0,1,1+\alpha)$
$1+\alpha+\alpha x$	1	0	α	$(1,0,\alpha)$
$(1+\alpha)x$	$1+\alpha$	1	α	$(1+\alpha,1,\alpha)$
$1+(1+\alpha)x$	α	0	$1+\alpha$	$(\alpha,0,1+\alpha)$
$\alpha+(1+\alpha)x$	1	$1+\alpha$	0	$(1,1+\alpha,0)$
$1+\alpha+(1+\alpha)x$	0	α	1	$(0,\alpha,1)$

2.2. Matriz generadora

Las filas de una matriz generadora forman una base del código. La codificación de cada uno de los elementos de una base de $\mathbb{P}_k$ como es $\{1, x, x^2, \ldots, x^{k-1}\}$ resulta en una base del código y por tanto una matriz generadora de un código Reed-Solomon con puntos de evaluación $x_1, x_2, \ldots, x_n$ es

$$G = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \ldots & x_n^{k-1} \end{pmatrix}$$

De esta manera la codificación dada por la matriz equivale a evaluar el polinomio asociado

a la palabra del mensaje. Si $u = (u_0, u_1, \ldots, u_{k-1})$ entonces considerando el polinomio asociado $u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1}$

$$uG = (u(x_1), u(x_2), \ldots, u(x_n))$$

Ejemplo 12. La matiz generadora del código especificado en el ejemplo 11, cuyas filas son la evaluación de 1 y x, es la siguiente

$$G = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & 1+\alpha \end{pmatrix}$$

2.3. Puntos de evaluación

La manera más habitual de seleccionar los puntos de evaluación, es tomando todas las potencias de un elemento no nulo β del cuerpo, y en particular las de un elemento primitivo α. Esta elección proporciona varias ventajas, principalmente que el código resultante en estos casos es un código cíclico, lo que proporciona propiedades adicionales que hacen que sea más sencillo trabajar con el código. En el siguiente capítulo veremos esto con detalle.

Además una limitación de los códigos Reed-Solomon es tener que elegir n elementos del cuerpo distintos en los que evaluar un polinomio, con lo que $n \leq q$. Esto obliga a trabajar con cuerpos más grandes. En el caso de que los puntos sean todas las potencias de un elemento primitivo tendríamos el código cíclico con mayor n posible en ese cuerpo. En ese caso $n = q - 1$ y los puntos de evaluación son todos los de $\mathbb{F}_q \setminus \{0\} = \{\alpha^i : 0 \leq i \leq q-1\}$ en un orden determinado por el elemento primitivo particular; la elección de otro elemento primitivo resultaría simplemente en una permutación del código.

Ejemplo 13. El código especificado en el ejemplo 11 es de este tipo

$$x_1 = \alpha^0 = 1$$

$$x_2 = \alpha^1 = \alpha$$

$$x_3 = \alpha^2 = 1 + \alpha$$

2.4. Decodificación

Hay distintos algoritmos para decodificar códigos Reed-Solomon en particular. En esta sección vamos a ver un algoritmo que consiste en construir un polinomio en dos variables que interpole la palabra recibida, y veremos que la palabra original (la más cercana) se va a poder calcular encontrando un factor lineal de este polinomio, siempre que se hayan producido como mucho $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ errores.

En el contexto de un código $RS[n, k]$, consideramos una palabra del código c que será el resultado de evaluar un cierto polinomio $f(x)$ con $\deg f(x) < k$ en los puntos adecuados, es decir $c = (f(x_1), \ldots, f(x_n))$. A esta palabra se le introducen a lo sumo t errores y resulta en la palabra recibida r.

El algoritmo se basa en construir un polinomio con coeficientes en el cuerpo $\mathbb{F}_q$ a partir de esta palabra recibida que sea de la forma

$$Q(x, y) = Q_0(x) + yQ_1(x)$$

Y cumpliendo

- $Q(x_i, r_i) = 0, \quad i = 1, \ldots, n.$
- $\deg(Q_0) \leq n - 1 - t = l_0.$
- $\deg(Q_1) \leq n - 1 - t - (k - 1) = l_1.$

Si tenemos un polinomio de estas características y $d(c, r) < t$ entonces $Q(x_i, f(x_i)) = 0$ en al menos $n - t$ puntos ya que $f(x_i) = r_i$ en todas las posiciones sin error. A partir de esto podemos concluir que $Q(x, f(x)) = 0$, ya que si f tiene grado a lo sumo $k - 1$, entonces $Q(x, f(x)) = Q_0(x) + f(x)Q_1(x)$ es un polinomio en $\mathbb{F}_q[x]$ de grado $n - t - 1$ a lo sumo y solo puede tener $n - t$ ceros si es el polinomio nulo.

Esto nos dice que Q tiene como factor $(y - f(x))$ por consiguiente, de $Q_0(x) + f(x)Q_1(x) = 0$ podemos despejar $f = -\frac{Q_0(x)}{Q_1(x)}$.

Veamos que existe un polinomio no nulo que cumple las condiciones anteriores. La primera condición nos da n ecuaciones en los coeficientes del polinomio mientras que los límites en los grados de los polinomios constituyentes nos dicen de cuántas incógnitas disponemos. Se tiene que Q_0 tiene $n - t$ coeficientes y que Q_1 tiene $n - t - (k - 1)$ coeficientes, en total

$$(n - t) + (n - t - (k - 1)) = 2n - 2t - (k - 1) \geq 2n - (n - k) - (k - 1) = n + 1 > n.$$

Luego el sistema tiene soluciones no nulas ya que el número de ecuaciones es estrictamente mayor al número de incógnitas

El sistema en forma matricial resultante de estas ecuaciones es el siguiente

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{l_0} & r_1 & r_1x_1 & r_1x_1^2 & \dots & r_1x_1^{l_1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{l_0} & r_2 & r_2x_2 & r_2x_2^2 & \dots & r_2x_2^{l_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{l_0} & r_n & r_nx_n & r_nx_n^2 & \dots & r_nx_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,l_0} \\ Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.1}$$

El algoritmo se reduce entonces a encontrar una solución no nula del sistema planteado de esta manera. Si la palabra recibida está lo suficientemente cerca como para ser corregida entonces el resultado será la palabra generada por $f = -\frac{Q_0(x)}{Q_1(x)}$, en otro caso Q_0 no será divisible por Q_1 y no se podrá llevar a cabo la decodificación o se decodificará incorrectamente.

Algoritmo 1.

Encontrar Q_0, Q_1 una solución no nula de la ecuación 2.1, a partir de la palabra recibida r.

si $Q_1 \mid Q_0$ **entonces**
  $f(x) \leftarrow -Q_0(x)/Q_1(x)$
  $c = (f(x_1), \dots, f(x_n))$
  si $d(c, r) \leq t$ **entonces**
    devolver c
  en otro caso
    devolver *error*
  fin
en otro caso
  devolver *error*
fin

El sistema a resolver es de n ecuaciones y por tanto la complejidad de este algoritmo será $O(n^3)$ (usando un algoritmo estándar para resolver un sistema lineal). Más adelante veremos un algoritmo más eficiente.

Ejemplo 14. Supongamos que se transmite la palabra (α, α, α) de un código $RS[3, 1]$, con α un elemento primitivo de $\mathbb{F}_4$ cumpliendo $\alpha^2 = \alpha + 1$. Para este código los puntos de evaluación son $x_1 = 1, x_2 = \alpha, x_3 = \alpha + 1$.
Los códigos de Reed Solomon con $k = 1$ contienen las palabras con todas las coordenadas iguales ya que derivan de evaluar los polinomios de grado 0, constantes. En este tipo de códigos se puede corregir basándose en una mayoría de coordenadas pero vamos a usar este ejemplo sencillo para aplicar el algoritmo suponiendo que durante la transmisión se introduce el error $(0, 0, 1)$, resultando en la palabra recibida $(\alpha, \alpha, \alpha + 1)$.

Los parámetros del código son $n = 3$, $k = 1$ y $t = 1$ luego $l_0 = n - 1 - t = 1$ y $l_1 = n - 1 - t - (k - 1) = 1$.

$$\begin{bmatrix} 1 & x_1 & r_1 & r_1x_1 \\ 1 & x_2 & r_2 & r_2x_2 \\ 1 & x_3 & r_3 & r_3x_3 \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{1,0} \\ Q_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \alpha & \alpha \\ 1 & \alpha & \alpha & \alpha + 1 \\ 1 & \alpha + 1 & \alpha + 1 & \alpha \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{1,0} \\ Q_{1,1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Suponiendo $Q_{1,1} = 1$, encontramos la solución $Q_0(x) = \alpha x$ y $Q_1(x) = x$ y en efecto $f(x) = -Q_0(x)/Q_1(x) = \alpha x/x = \alpha$, que genera la palabra original.

3. RS Y CÓDIGOS CÍCLICOS

Al igual que la propiedad de linealidad permite deducir propiedades y algoritmos útiles que no se tendrían en un caso más general, se puede considerar una clase más restringida de códigos que facilitará el estudio. Además veremos que los códigos Reed-Solomon más utilizados son de este tipo. En este capítulo seguimos teniendo a [4] como referencia principal.

3.1. Conceptos generales de códigos cíclicos

Definición 10. (Código Cíclico) Un código lineal C es cíclico si para cualquier palabra $c = (c_0, \ldots, c_{n-1}) \in C$ se tiene que

$$\hat{c} = (c_{n-1}, c_0, \ldots, c_{n-2}) \in C$$

3.1.1. Códigos vistos en espacios de polinomios

El estudio de códigos cíclicos es más sencillo si consideramos las palabras del código como polinomios. Asociando $(a_0, \ldots, a_{n-1})$ a $a(x) = a_0 + a_1 x + \ldots a_{n-1}x^{n-1}$. De hecho esta es una manera alternativa de ver cualquier código lineal.

Considerando las palabras como polinomios podemos escribir la palabra permutada de la siguiente manera

$$\hat{a}(x) = a_{n-1} + a_0 x + a_1 x^2 \ldots a_{n-2}x^{n-1} = xa(x) - a_{n-1}(x^n - 1) \tag{3.1}$$

De esta ecuación podemos deducir otra manera útil de ver un código cíclico: en el anillo $\mathbb{F}_q[x]/(x^n - 1)$, ya que permutar equivale en este anillo a multiplicar la palabra por x.

$$x^n = 1 \implies x(a_0 + a_1 x + \ldots a_{n-1}x^{n-1}) = a_0 x + a_1 x^2 + \ldots a_{n-1}x^n = a_{n-1} + a_0 x + a_1 x^2 + \ldots a_{n-2}x^{n-1}$$

La aplicación implícita de asociar una palabra del código a un polinomio, bien en $\mathbb{P}_n$ o bien en $\mathbb{F}_q[x]/(x^n - 1)$, es inyectiva, lineal y con llegada a espacios de la misma dimensión que $\mathbb{F}_q^n$ y son por lo tanto isomorfismos. Lo que justifica que podamos pasar de un enfoque a otro.

3.1.2. Polinomio generador

Desde esta perspectiva polinómica, al igual que hablamos de la matriz generadora de un código tenemos un polinomio generador.

Definición 11. (Código generado por un polinomio) El código $C[n,k]$ generado por un polinomio $g(x) \in \mathbb{F}_q[x]$ de grado $n-k$ es

$$\{u(x) \in \mathbb{P}_n : u(x) = 0 \pmod{g(x)}\} = \{a(x)g(x) : a(x) \in \mathbb{P}_k\}$$

La dimensión del código generado por un polinomio de grado $n-k$ es k ya que $\mathbb{P}_k$ isomorfo al código considerando la siguiente aplicación lineal, $f(a(x)) = a(x)g(x) \in C$ para $a(x) \in \mathbb{P}_k$ y su inversa $f^{-1}(c(x)) = c(x)/g(x) \in \mathbb{P}_k$ para $c(x) \in C$.

> **Ejemplo 15.** Con coeficientes en el cuerpo $\mathbb{F}_q$ consideramos el código $C[4,2]$ generado por el polinomio $g(x) = (x^2+1)$.
>
> $$\{(a_0 + a_1x)(x^2+1) : a_0, a_1 \in \mathbb{F}_q\} = \{a_1x^3 + a_0x^2 + a_1x + a_0\}$$
>
> Este es un código que repite los 2 símbolos del mensaje y permite dectectar un error pero no corregirlo, puesto que tiene distancia mínima 2 (x^2+1 es una palabra del código con peso 2).

Utilizando esta definición de polinomio generador podemos caracterizar los códigos cíclicos de la siguiente manera

Teorema 4. *Un código $C[n,k]$ es cíclico sí y solo sí es generado por un polinomio $g(x)$ que es divisor de $x^n - 1$.*

Operando en $\mathbb{F}_q[x]$, supongamos primero que $C[n,k]$ es generado por $g(x)$, divisor de $x^n - 1$. Sea $a(x) \in C$, entonces $g(x)$ divide a $a(x)$. Veamos que la palabra permutada también está en el código.

Refiriéndonos a la ecuación 3.1 para expresar $\hat{a}(x)$, observamos que $g(x)$ divide a $xa(x)$ por dividir a $a(x)$ y también divide a $(x^n - 1)$ por hipótesis, luego $g(x)$ divide a la palabra permutada $\hat{a}(x)$, es decir $\hat{a}(x)$ está en el código.

Por otra parte si un código es cíclico veamos que es generado por un polinomio que sea divisor de $x^n - 1$. Tomemos $g(x)$ la polinomio mónico de menor grado en C. Esto es siempre posible de manera única ya que si existiesen 2 polinomios de estas características, su resta también pertenecería a C y sería de menor grado.

Pasamos ahora a operar en $\mathbb{F}_q[x]/(x^n-1)$, tenemos que $xg(x) \in C$ por ser C cíclico, por lo tanto $x^s g(x) \in C \ \forall s > 0$. De esto y la linealidad deducimos que

$$a(x)g(x) \in C \quad \forall a(x) \quad \deg a(x) < n - \deg g(x)$$

Por otra parte, cualquier palabra del código puede expresarse de esta manera. Si $a(x)$ es una palabra del código, dividiendo tenemos

$$a(x) = b(x)g(x) + r(x) \quad \deg r(x) < \deg g(x)$$

Hemos visto que $b(x)g(x) \in C$, entonces por linealidad

$$r(x) = a(x) - b(x)g(x) \in C$$

que contradice el hecho de que $g(x)$ sea la palabra de menor grado del código.

Esto se corresponde con que el código sea un ideal de $\mathbb{F}_q[x]/(x^n-1)$.

$$C = (g(x)) = \{a(x)g(x) : a(x) \in \mathbb{F}_q[x]/(x^n-1)\}$$

Ejemplo 16. El código del ejemplo anterior es cíclico ya que su generador (x^2+1) es un factor de (x^n-1) :

$$(x^4-1) = (x^2+1)(x^2-1)$$

En efecto el resultado de permutar una palabra también está en el código:

$$a(x) = (a_0 + a_1 x)g(x) = a_0 + a_1 x + a_0 x^2 + a_1 x^3$$

$$\hat{a}(x) = (a_1 + a_0 x)g(x) = a_1 + a_0 x + a_1 x^2 + a_0 x^3$$

A partir del polinomio generador se puede encontrar una matriz generadora.

Teorema 5. *Sea $C[n,k]$ un código generado por $g(x) = g_0 + g_1 x + \ldots g_{n-k}x^{n-k}$, entonces una matriz generadora es*

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & & & & \\ & g_0 & g_1 & \cdots & g_{n-k} & & & \\ & & g_0 & g_1 & \cdots & g_{n-k} & & \\ & & & \ddots & \ddots & \ddots & \ddots & \\ & & & & g_0 & g_1 & \cdots & g_{n-k} \end{pmatrix}$$

Como ya hemos visto en otras ocasiones a partir de una codificación se puede obtener una matriz generadora codificandodo una base del espacio de los mensajes. En este caso la codificación viene dada por

$$\begin{aligned} \mathbb{P}_k &\to C \\ a(x) &\mapsto a(x)g(x) \end{aligned}$$

y por tanto codificando la base usual $\{1, x, x^2, \ldots, x^k\}$ tenemos la base del código representada en las filas de G por sus coeficientes.

Ejemplo 17. Para el código de los ejemplos anteriores

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Y tenemos la codificación $(a_0, a_1)G = (a_0, a_1, a_0, a_1)$.

3.1.3. Polinomio de control

También de manera análoga a las matrices de control, podemos hablar de polinomio de control que caracteriza las palabras del código y da un método para comprobar la pertenencia o no al código.

Definición 12. (Polinomio de control) Sea C un código lineal, y $g(x)$ su polinomio generador. El polinomio de control es

$$h(x) = (x^n - 1)/g(x)$$

Es decir

$$(x^n - 1) = h(x)g(x)$$

La propiedad clave de este polinomio es que caracteriza a las palabras del código, visto en $\mathbb{F}_q[x]/(x^n - 1)$ tenemos

$$a(x) \in C \iff a(x)h(x) = 0$$

Esto se debe a que por ser C generado por $g(x)$ entonces $a(x) = b(x)g(x)$ y $a(x)h(x) = b(x)g(x)h(x) = b(x)(x^n - 1) = 0 \pmod{x^n - 1}$

3.2. Códigos Reed-Solomon cíclicos

Teorema 6. *Los códigos RS son cíclicos si se toman como puntos de evaluación todas las potencias de un elemento no nulo del cuerpo.*

Sea $\beta \in \mathbb{F}_q$ tal que $ord(\beta) = n$, el código Reed-Solomon correspondiente a estos puntos de evaluación es

$$C[n,k] = \{(f(1), f(\beta), f(\beta^2) \ldots, f(\beta^{n-1})) : f(x) \in \mathbb{P}_k\}$$

Para comprobar que este código es cíclico, tomamos la palabra resultante de evaluar un polinomio $f(x)$ y observamos que se puede construir un polinomio que resulte en la palabra permutada haciendo uso de que $\beta^n = 1$

$$f'(x) = f(\beta^{n-1}x)$$

En este caso la palabra resultante es

$$(f'(1), f'(\beta), f'(\beta^2) \ldots, f'(\beta^{n-1})) = (f(\beta^{n-1}), f(1), f(\beta) \ldots, f(\beta^{n-2}))$$

Como ya se mencionó en el capítulo anterior los códigos RS más utilizados son de esta forma, normalmente escogiendo un elemento primitivo y en tal caso $n = q - 1$. De esta manera tenemos el código RS cíclico más grande posible en un cuerpo dado ya que el conjunto de los puntos de evaluación será en este caso el grupo multiplicativo más grande del cuerpo.

En lo siguiente consideramos un código $RS[n,k]$ cíclico con puntos de evaluación las potencias de un elemento β de orden n.

3.2.1. Matriz generadora y de control

Una matriz generadora de un código de este tipo siguiendo la forma general dada en el capítulo anterior es la siguiente

$$G = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \beta & \beta^2 & \ldots & \beta^{n-1} \\ 1 & \beta^2 & \beta^4 & \ldots & \beta^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \beta^{k-1} & \beta^{2(k-1)} & \ldots & \beta^{(k-1)(n-1)} \end{pmatrix}$$

Además gracias a la estructura adicional de los códigos cíclicos, podemos dar una matriz de control

$$H = \begin{pmatrix} 1 & \beta & \beta^2 & \dots & \beta^{n-1} \\ 1 & \beta^2 & \beta^4 & \dots & \beta^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \beta^{(n-k)} & \beta^{2(n-k)} & \dots & \beta^{(n-1)(n-k)} \end{pmatrix}$$

Veamos por qué esta es la matriz de control.

Sabemos que las palabras del código son combinación lineal de las filas de la matriz generadora, entonces podemos dar una caracterización las matrices de control dada una matriz generadora G. En concreto una matriz H de dimensiones $(n-k, n)$ es de control si $GH^T = 0$.

Sea

$$M = GH^T = (m_{ir})_{\substack{1 \le i \le k \\ 1 \le r \le n-k}}$$

Vamos a comprobar que esta matriz es nula viendo que se anulan todos sus elementos m_{ir} que es el resultado de multiplicar la fila i-ésima de G por la fila r-ésima de H, esto es

$$m_{ir} = \sum_{j=1}^{n} \beta^{(i-1)(j-1)} \beta^{r(j-1)} = \sum_{j=1}^{n} (\beta^{i-1+r})^{(j-1)}$$

Observamos que es la suma de los primeros términos de una sucesión geométrica de razón β^{i-1+r}, para la cual tenemos que es igual a

$$\frac{(\beta^{i-1+r})^n - 1}{\beta^{i-1+r} - 1}$$

ya que la razón es distinta de 1:

$$i - 1 + r \le k - 1 + n - k = n - 1 < n$$

Como el exponente al que está elevado β es estrictamente menor que su orden, $\beta^{i-1+r} \neq 1$.

Por otra parte como el orden de β es n, $(\beta^{i-1+r})^n = 1$ y el numerador se anula, con lo que tenemos que m_{ir} es 0 para todo i, r y que efectivamente la matriz H dada es una matriz de control.

Ejemplo 18. Consideremos un código $RS[7,4]$ en $\mathbb{F}_8$ con puntos de evaluación las potencias de un elemento primitivo α que cumple $\alpha^3 = \alpha^2 + 1$. En este caso la matriz de generadora y de control son:

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^2+1 & \alpha^2+\alpha+1 & \alpha+1 & \alpha^2+\alpha \\ 1 & \alpha^2 & \alpha^2+\alpha+1 & \alpha^2+\alpha & \alpha & \alpha^2+1 & \alpha+1 \\ 1 & \alpha^2+1 & \alpha^2+\alpha & \alpha^2 & \alpha+1 & \alpha & \alpha^2+\alpha+1 \end{pmatrix}$$

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^2+1 & \alpha^2+\alpha+1 & \alpha+1 & \alpha^2+\alpha \\ 1 & \alpha^2 & \alpha^2+\alpha+1 & \alpha^2+\alpha & \alpha & \alpha^2+1 & \alpha+1 \\ 1 & \alpha^2+1 & \alpha^2+\alpha & \alpha^2 & \alpha+1 & \alpha & \alpha^2+\alpha+1 \end{pmatrix}$$

Se observa que H es una submatriz de G en el caso en que $n - k \leq k - 1$.

3.2.2. Polinomio generador

Veamos que el polinomio generador de un código $RS[n,k]$ es el siguiente:

$$g(x) = (x - \beta)(x - \beta^2) \dots (x - \beta^{(n-k)})$$

Primero, observemos que $g(x)$ divide a otro polinomio $a(x)$ si y solo si $a(\beta^i) = 0$, $\forall i = 1, \dots, (n-k)$.

Por otra parte multiplicar por H resulta en evaluar el polinomio asociado al vector en las raíces de $g(x)$. De esto podemos concluir que la palabra está en el código si y solo si es un múltiplo de $g(x)$, es decir $g(x)$ es el polinomio generador.

$$a \in RS[n.k] \iff Ha^T = (a(\beta), a(\beta^2), \dots, a(\beta^{n-k}))^T = 0 \iff a(x) = b(x)g(x)$$

Ejemplo 19. Para el código del ejemplo anterior consideremos la palabra resultante de evaluar el polinomio $\alpha^2 x^3 + x + \alpha$

$$a = (\alpha^2 + \alpha + 1, \alpha + 1, \alpha^2, 0, \alpha^2, \alpha^2, \alpha)$$

Ahora evaluamos el polinomio asociado $a(x)$ en las primeras $n - k = 3$ potencias

$$a(\alpha) = a(\alpha^2) = a(\alpha^3) = 0$$

Y sin embargo $a(\alpha^4) = \alpha^2 \neq 0$. Hemos comprobado con esto que efectivamente $g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)$ divide a $a(x)$, de hecho $a(x) = (\alpha x^3+\alpha x^2+(\alpha+1))g(x)$.

3.3. BCH

En el caso $q = 2^r$, un código BCH (Bose–Chaudhuri–Hocquenghem) es el subcódigo compuesto por las palabras binarias de un código RS, es decir, los polinomios de grado menor o igual que n con coeficientes no en $\mathbb{F}_q$ si no en $\mathbb{F}_2$. A este tipo de construcción, de manera general se le llama código subcuerpo.

Su polinomio generador tiene los mismos ceros que el polinomio generador del código original, pero tiene que ser binario. El polinomio generador es entonces el mínimo común múltiplo de los polinomios mínimos (tomando $\mathbb{F}_2$ como cuerpo base), de los ceros del código original $\beta, \beta^2, \ldots, \beta^{n-k}$.

De las propiedades del código RS, hereda obviamente la linealidad, que es cíclico y que la distancia mínima es al menos $n - k + 1$. Pero k aquí es la dimensión del código original, el subcódigo siendo un subespacio tendrá una dimensión menor que denotamos por s.

Una propiedad importante que facilita la decodificación es que para códigos binarios, localizar los errores es equivalente a corregirlos.

Ejemplo 20. Calculemos el polinomio generador del subcódigo BCH para el código $RS[7,4]$ que veníamos considerando. Tenemos que hallar los polinomios mínimos de $\alpha, \alpha^2, \alpha^3$.
Al ser α un elemento primitivo que cumple $\alpha^3+\alpha^2+1=0$, entonces su polinomio mínimo será x^3+x^2+1 que es irreducible en $\mathbb{F}_2$, además este será también el polinomio mínimo de α^2 ya que en los cuerpos de característica 2 si un elemento es raíz de un polinomio también lo es su cuadrado.
Por otra parte el polinomio mínimo de α^3 es x^3+x+1, que es el otro polinomio irreducible en $\mathbb{F}_2$ de grado 3.
Entonces el polinomio de menor grado con coeficientes en $\mathbb{F}_2$ que tiene como raíces las del polinomio generador del código RS es

$$(x^3+x^2+1)(x^3+x+1) = x^6+x^5+x^4+x^3+x^2+x+1$$

El código resultante tiene únicamente dos palabras, las únicas palabras binarias del código original : $(0,0,0,0,0,0,0)$ y $(1,1,1,1,1,1,1)$. Es un código $[7,1]$. Veremos como acotar la dimensión de un código BCH para poder asegurar que no sea tan pequeña.

3.3.1. Generalización

Aunque los más extendidos sean los códigos BCH binarios, se puede generalizar este proceso para tomar subcódigos en otras situaciones. Para cualquier código RS en un cuerpo K, si seleccionamos un subcuerpo $K' < K$ que tomamos como cuerpo base, buscamos los polinomios mínimos de las raíces del código RS en este cuerpo y construimos el polinomio generador como producto de estos, entonces obtendremos el subcódigo correspondiente a quedarnos solo con las palabras del código original con coeficientes en K'.

3.3.2. Dimensión. Conjuntos Ciclotómicos.

Se puede derivar la expresión exacta de la dimensión del código en ciertos casos, las pruebas de estos resultados involucran conjuntos ciclotómicos. Recogemos aquí el resultado en el caso $n = p^r - 1$, cuando el código se construye a partir de un elemento primitivo del cuerpo $\mathbb{F}_{p^r}$:

Teorema 7. *La dimensión de un código BCH derivado de un código RS* $[n,k]$ *sobre* $\mathbb{F}_{q^r}$ *con* $n = p^r - 1$ *con* $1 \leq (n-k) \leq p^{\lceil r/2 \rceil} - 1$ *es*

$$s = n - r\lceil (n-k)(1-1/p) \rceil$$

Vamos a dar una prueba de este resultado basada en [9], donde se da una versión más general además de otros resultados aplicables en otros códigos BCH.

Empezamos definiendo qué es un conjunto ciclotómico en este contexto particular para el cuál lo usaremos.

Definición 13. (Conjunto Ciclotómico) El conjunto ciclotómico p-ario modulo n de x es

$$C_s = \{sp^j \mod n : 0 \leq j < r\} \subset \frac{\mathbb{Z}}{(n)}$$

Donde r es el orden de p módulo n, es decir $p^r = 1 \mod n$.

Ejemplo 21. Calculemos los conjuntos ciclotómicos binarios módulo 7:

$$C_1 = C_2 = C_4 = \{1, 2, 4\}$$

$$C_3 = C_5 = C_6 = \{3, 6, 5\}$$

Lo usaremos en el contexto de un cuerpo finito $\mathbb{F}_p^r$ con $n = p^r - 1$. Pasamos ahora a dar un lema útil para trabajar con estos conjuntos.

Lema 1. *Sean* x *e* y *elementos de un cuerpo de característica* p, $\mathbb{F}_p^r$. *Entonces*

$$(x+y)^{p^m} = x^{p^m} + y^{p^m}$$

Esto se puede ver por inducción ya que para $m = 1$

$$(x+y)^p = x^p + \sum_{i=1}^{p-1} \binom{p}{i} x^{p-i} y^i + y^p$$

En esta expresión todos los otros sumandos tiene el factor p luego se anulan, por otra parte suponemos que la igualdad es cierta para $m-1$ entonces

$$(x+y)^{p^m} = ((x+y)^{p^{m-1}})^p = (x^{p^{m-1}} + y^{p^{m-1}})^p = x^{p^m} + y^{p^m}$$

Donde en la última igualdad se ha aplicado el caso $m = 1$.

Este resultado nos permite dar el siguiente teorema, con el objetivo de poder razonar más tarde sobre polinomios minimos usando conjuntos ciclotómicos

Teorema 8. *Sea $\beta \in \mathbb{F}_{p^r}$, sea $f \in \mathbb{F}_p[x]$. Si $f(\beta) = 0$ entonces $f(\beta^p) = 0$.*

Si $f(\beta) = 0$, y $f(x) = \sum_{i=0}^{\deg f(x)} f_i x^i$ entonces

$$f(\beta^p) = \sum_{i=0}^{\deg f(x)} f_i \beta^{pi} = (\sum_{i=0}^{\deg f(x)} f_i \beta^i)^p = f(\beta)^p = 0$$

De este teorema podemos concluir que todas los elementos β^{pj} tienen el mismo polinomio minimo. De hecho son todas las raíces de ese polinomio mínimo y pueden usarse para expresarlo.

Teorema 9. *Sea α un elemento primitivo de $\mathbb{F}_{p^r}$, y sea $m_s(x)$ el polinomio mínimo de α^s entonces*

$$m_s(x) = \prod_{k \in C_s} (x - \alpha^k)$$

Ya hemos visto que para que un polinomio tenga a α^s como raíz, tiene que tener como raíz a todos los elementos α^{sp^j}, luego basta comprobar que los coeficientes estan en el cuerpo base $\mathbb{F}_p$.

Sea $t \leq r$ el mínimo número tal que $\alpha^{sp^t} = \alpha^s$, es sencillo ver entonces que

$$C_s = \{sp^j \mod n : 0 \leq j < t\}$$

Usando esto podemos expresar el producto anterior como

$$\prod_{k \in C_s} (x - \alpha^k) = \prod_{j=0}^{t-1} (x - \alpha^{sp^j}) = x^{t-1} - \left(\sum_{0 \leq k_1 \leq t-1} \alpha^{sp^{k_1}} \right) x^{t-2} + \left(\sum_{0 \leq k_1 < k_2 \leq t-1} \alpha^{sp^{k_1}} \alpha^{sp^{k_2}} \right) x^{t-3} - \dots$$

En general la expresión para el coeficiente de x^{t-1-j} con $j > 0$ es

$$(-1)^j \sum_{0 \leq k_1 < k_2 < \dots < k_j \leq t-1} \alpha^{sp^{k_1}} \alpha^{sp^{k_2}} \dots \alpha^{sp^{k_j}} = (-1)^j c_j$$

Tenemos que comprobar que $c_j \in \mathbb{F}_p$, para ello observamos que gracias al lema 1 entonces

$$c_j^p = \left(\sum_{0 \leq k_1 < k_2 < \dots < k_j \leq t-1} \alpha^{sp^{k_1}} \alpha^{sp^{k_2}} \cdots \alpha^{sp^{k_j}} \right)^p = \sum_{0 \leq k_1 < k_2 < \dots k_j \leq t-1} \alpha^{sp^{k_1+1}} \alpha^{sp^{k_2+1}} \cdots \alpha^{sp^{k_j+1}}$$
$$= \sum_{1 \leq k_1 < k_2 < \dots k_j \leq t} \alpha^{sp^{k_1}} \alpha^{sp^{k_2}} \cdots \alpha^{sp^{k_j}}$$

Finalmente, notando de nuevo que $\alpha^{sp^t} = \alpha^s$, concluimos que $c_j^p = c_j$, o alternativamente $c_j(c_j^{p-1} - 1) = 0$, que es exactamente la ecuación que caracteriza los elementos de F_p en una extensión F_{p^r} : o bien es 0 o un elemento de orden $p-1$. Con esto hemos comprobado que efectivamente, $\prod_{k \in C_s}(x - \alpha^k)$ es el polinomio mínimo de α^s.

Ejemplo 22. Siguiendo con el ejemplo 20. El polinomio mínimo de α es

$$m_1(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4) = 1 + x^2 + x^3$$

Y el polinomio mínimo de α^3

$$m_3(x) = (x - \alpha^3)(x - \alpha^5)(x - \alpha^6) = 1 + x + x^3$$

De este teorema se deduce inmediatamente una expresión del polinomio generador de un código BCH construido de un código $RS[n, k]$ con $n = p^r - 1$ sobre $\mathbb{F}_{p^r}$.

$$g(x) = \prod_{z \in Z} (x - \alpha^z) \quad Z = C_1 \cup C_2 \cup \cdots \cup C_{n-k} \tag{3.2}$$

Pasamos a dar ahora 2 resultados de conjuntos ciclotómicos auxiliares que nos permitirán determinar el grado de este polinomio y por tanto la dimensión del código.

Lema 2. *El conjunto ciclotómico C_s tiene cardinal r para todo s con $1 \leq s \leq p^{\lceil r/2 \rceil}$.*

Para $r = 1$ obviamente $|C_s| = 1$. Consideramos entonces $r > 1$ y, buscando una contradicción, suponemos $|C_s| < r$. Si se da esto entonces existen $j' > i'$ con $sp^{j'} = sp^{i'} \mod n$, entonces para $j = j' - i'$ se tiene $sp^j = 0 \mod n$ (se puede dividir $p \mod n$ al darse $gcd(p, n) = 1$), o lo que es lo mismo $s(p^j - 1) = 0 \mod n$. Por otra parte $p^r = 1 \mod n$ y entonces $j|r$.
Al ser j divisor de r se tiene que $1 \leq j \leq r/2$, pero entonces

$$s(p^j - 1) < p^{r/2}(p^{r/2} - 1) < n$$

Lo cuál contradice $s(p^j - 1) = 0 \mod n$.

Lema 3. *Sean x e y enteros distintos con $1 \leq x, y \leq p^{\lceil r/2 \rceil} - 1$ y $x, y \neq 0 \mod p$. Entonces C_x, C_y son disjuntos.*

Primero observamos que si $C_x \neq C_y$ entonces tienen que ser disjuntos ya que si $z \in C_s$ entonces $pz \mod n \in C_s$, es decir $C_s = C_z$ para todo $z \in C_s$. Por esto si $z \in C_x \cap C_y$, entonces $C_x = C_y$.

El caso para $r = 1$ es trivial ya que $C_x = \{x\}$ y $C_y = \{y\}$. Si $r > 1$ consideramos ahora el conjunto

$$S = \{xp^j \mod n : 0 \leq j \leq \lfloor r/2 \rfloor\} \cup \{yp^j \mod n : 0 \leq j \leq \lfloor r/2 \rfloor\}$$

Como $x, y \leq p^{\lceil r/2 \rceil} - 1$ entonces $xp^j, yp^j \leq p^{\lfloor r/2 \rfloor}(p^{\lceil r/2 \rceil} - 1) = n + 1 - p^{\lfloor r/2 \rfloor} < n$ si $r > 1$, entonces dado $x, y \neq 0 \mod p$ ningún elemento puede coincidir con ningún otro y $|S| = 2(\lfloor r/2 \rfloor + 1) \geq r + 1$. Si suponemos $C_x = C_y$ entonces $|C_x| = |C_y| \geq |S| \geq r + 1$ lo cual contradice el lema anterior en que $|C_x| = r$.

Ejemplo 23. Vamos a ilustrar estos dos lemas anteriores con el ejemplo de los conjuntos ciclotómicos binarios módulo 15, que son los siguientes:

$$C_1 = \{1, 2, 4, 8\}$$

$$C_3 = \{3, 6, 12, 9\}$$

$$C_5 = \{5, 10\}$$

$$C_7 = \{7, 14, 13, 11\}$$

Como se puede ver son disjuntos y C_1, C_3 que cumplen la condición $s \leq p^{\lceil r/2 \rceil} = 4$ tienen cardinal $r = 4$. Por otra parte observamos que C_5 no tiene este tamaño luego la condición no se puede relajar.

Vamos a probar finalmente la expresión de la dimensión de un código BCH. Vamos a ver que el grado del polinomio generador es $r\lceil (n-k)(1-1/p) \rceil$, viendo que este es el valor de $|Z|$ en la ecuación en que damos un expresión del polinomio 3.2. Comenzamos volviendo a observar que si s es un multiplo de p módulo n entonces $C_{x/q} = C_x$, de manera que el numero de conjuntos en la unión es $(n-k) - \lfloor (n-k)/p \rfloor$. Se dan las hipótesis de los lemas anteriores y por tanto todos son disjuntos y con r elementos, y tenemos el resultado.

Ejemplo 24. En efecto para el ejemplo 20 el polinomio generador resulta en

$$g(x) = m_1(x)m_3(x) = \prod_{k \in Z}(x - \alpha^k) \quad Z = C_1 \cup C_2 \cup C_3 = C_1 \cup C_3$$

Y la dimensión aplicando el resultado es

$$s = 7 - 3\left\lceil \frac{3}{2} \right\rceil = 7 - 6 = 1$$

3.3.3. Decodificación

Concluimos esta sección remarcando que los códigos BCH son una posibilidad para aplicar los códigos Reed Solomon a la manera más habitual de representar información, el

alfabeto binario, sobrepasando de esta manera la limitación que tienen los códigos RS en la longitud de los mensajes en función del tamaño del cuerpo ($n < q$). Sin embargo no son más que subcódigos de un código RS, de manera que todas los resultados que se verán más adelante de estos pueden ser particularizadas para códigos BCH.

En concreto los algortimos de decodificación de códigos RS sirven para BCH.

Ejemplo 25. Sea α un elemento primitivo de $\mathbb{F}_9$ tal que $\alpha^2 = \alpha + 1$. Construimos un código BCH $[8, 4]$ tomando como polinomio generador

$$g(x) = (x - \alpha)(x - \alpha^3)(x - \alpha^2)(x - \alpha^6) = x^4 + 2x^3 + 2x + 2$$

A partir de un código $RS\,[8, 6]$ de la manera detallada en la prueba de la dimensión.

Si tenemos la palabra dada por $(x^3 + 1)g(x) = 2 + 2x + x^3 + 2x^6 + x^7$ que es

$$(2, 2, 0, 1, 0, 0, 2, 1)$$

y que corromperemos con un error en la tercera posición

$$(2, 2, 2, 1, 0, 0, 2, 1)$$

Podemos hallar un polinomio Q como en la sección 2.4 siguiendo el código $RS\,[8, 6]$ original.

$$Q(x, y) = \left(\alpha^5 x^4 + \alpha^3 x^3 + \alpha^7 x^2 + \alpha^6 x + \alpha^2\right) + y(x + \alpha^6)$$

Entonces

$$f = -Q_0(x)/Q_1(x) = \alpha x^3 + \alpha x + 1$$

Que, se puede comprobar, da la palabra original.

4. BERLEKAMP-MASSEY

Dedicamos este capítulo a presentar el algoritmo de Berlekamp-Massey. Este algoritmo se debe originalmente a Berlekamp en el libro *Algebraic Coding Theory* [10] donde lo aplica a la decodificación de códigos BCH, más tarde Massey ([11]) simplificó el algoritmo y reconoció su aplicación para la síntesis de recurrencias lineales, más concretamente, para encontrar la mínima recurrencia lineal que puede generar una secuencia dada. Veremos como esto va a ser equivalente a poder encontrar el patrón del error en una palabra recibida a partir de su síndrome.

La referencia seguida en este capítulo es [12], donde se presentan las recurrencias lineales además del algoritmo.

4.1. Recurrencia lineal

Las recurrencias lineales, qué se explican aquí para aplicarlas al algoritmo de decodificación, tienen utilidad en otros campos como por ejemplo para generar numeros pseudoaleatorios y además son fácilmente implementables en hardware [13].

Definición 14. (Recurrencia lineal). Una recurrencia lineal es una ecuación que relaciona el k-ésimo elemento de una secuencia $(V_1, V_2, \dots)$ con los L elementos inmediatamente anteriores mediante una función lineal. Denotaremos $(\mathbf{\Lambda}, L)$ a una recurrencia de la siguiente forma

$$V_k = -\sum_{j=1}^{L} \Lambda_j V_{k-j} \tag{4.1}$$

Decimos que L es la **longitud de la recurrencia**, y $\mathbf{\Lambda} = (\Lambda_1, \Lambda_2, \dots, \Lambda_L)$ es el **vector de pesos**.
Diremos que la recurrencia **genera** una secuencia $\mathbf{V} = (V_1, V_2, \dots)$ si se cumple la ecuación 4.1 para $k > L$ debido a que a partir de los primeros L elementos se puede calcular la secuencia entera.
Donde la longitud no sea relevante nos referiremos a una recurrencia solo por su vector de pesos.

A modo de caso especial, diremos que la recurrencia vacía de longitud 0 genera la secuencia nula $(0, 0, \dots)$.

Ejemplo 26. Sea $L = 2$, $\mathbf{\Lambda} = (-1, -1)$ y los valores iniciales $V_1 = 0, V_2 = 1$. El siguiente elemento generado por la recurrencia es

$$V_3 = -\Lambda_1 V_2 - \Lambda_2 V_1 = 0 + 1 = 1$$

Los primeros terminos de la secuencia son $(0, 1, 1, 2, 3, 5, 8, 13, 21 \dots)$, la sucesión de Fibonacci es un ejemplo de recurrencia lineal.

A lo largo de esta sección trataremos indistintamente vector y polinomio de pesos. La diferencia respecto a secciones anteriores en las que hemos usado esta perspectiva polinómica es que consideraremos el termino independiente siempre 1. Esto es, para un vector $\mathbf{\Lambda} = (\Lambda_1, \dots, \Lambda_L)$ denotaremos por $\Lambda(x)$ al polinomio $1 + \Lambda_1 + \Lambda_2 x^2 + \cdots + \Lambda_L x^L$.

Este polinomio asociado tiene grado no superior a la longitud de la recurrencia, además extendiendo de esta forma $\Lambda_0 = 1$ podemos reescribir la ecuación 4.1, sin pérdida de generalidad, como

$$\sum_{j=0}^{L} \Lambda_j V_{k-j} = 0 \quad \text{para } k > L$$

(Esta ecuación justifica el caso especial que mencionamos antes de la recurrencia vacía, si $L = 0$ se convierte en $V_k = 0$ para $k = 1, 2, \dots$ esto es $\mathbf{V} = (0, 0, \dots)$)

Por otra parte, puede parecer arbitrario indexar los valores anteriores de manera descendente (Con índice $k - j$), la justificación es la siguiente: si a la secuencia generada $\mathbf{V} = (V_1, V_2, \dots)$ le hacemos corresponder la serie de potencias $V(x) = \sum_{j=1}^{\infty} V_j x^j$ entonces la expresión en la ecuación anterior se puede interpretar como el k-ésimo coeficiente del producto $\Lambda(x)V(x)$.

Entonces una secuencia generada por la recurrencia cumplirá

$$V(x)\Lambda(x) = \Gamma(x) \quad \deg \Gamma(x) \leq L \tag{4.2}$$

Usaremos este producto para caracterizar si una recurrencia lineal genera un vector finito. En este caso en vez de serie de potencias asociamos al vector un polinomio de la misma manera. Al vector $\mathbf{V} = (V_1, \dots, V_r)$ le hacemos corresponder el polinomio $V(x) = \sum_{j=1}^{r} V_j x^j$.

Un vector es producido por la secuencia $(\Lambda(x), L)$ sí y solo los coeficientes de los monomios de grado $k > L$, del producto $V(x)\Lambda(x)$ son nulos. Esto es

$$V(x)\Lambda(x) = \Gamma(x) + x^{r+1}\Theta(x), \quad \deg \Gamma(x) \leq L \tag{4.3}$$

Ejemplo 27. Continuando con el ejemplo anterior veamos que tomando $r = 5$, el producto en 4.3 no tiene coeficientes de grado $3, 4, 5$.

$$V(x) = x^2 + x^3 + 2x^4 + 3x^5$$

$$\Lambda(x) = 1 - x - x^2$$

$$V(x)\Lambda(x) = x^2 - 5x^6 - 3x^7 \implies \Gamma(x) = x^2$$

Como curiosidad tangente al tema, esto significa que si los coeficientes de una serie de potencias son generados por una recurrencia lineal entonces podemos encontrar una expresión racional de la serie dada por $\Gamma(x)/\Lambda(x)$, con $\deg \Gamma(x) = \deg \Lambda(x) = L$, donde solo el numerador depende de los L valores iniciales. (Siendo éstas expresiones formales, sin entrar en la convergencia o no de la serie).

Ejemplo 28. Si $V(x)$ es la serie de potencias que tiene como coeficientes los términos de la sucesión de Fibonacci, en el ejemplo anterior vimos que para esta secuencia $\Gamma(x) = x^2$. Entonces $V(x)\Lambda(x) = \Gamma(x)$ y podemos hacer corresponder a la serie de potencias la expresión racional $f(x) = \frac{x^2}{1-x-x^2}$. Podemos comprobar que la serie de Taylor de $f(x)$ en $x = 0$ es $V(x)$.

Definición 15. (Complejidad lineal). La complejidad lineal de un vector $(V_1, \ldots, V_n)$ es la menor longitud posible que tiene una recurrencia lineal que lo genere.

Lema 4. *Si* $\mathbf{S} = (S_1, \ldots, S_r)$ *tiene complejidad lineal* L' *y* $\mathbf{T} = (T_1, \ldots, T_r)$ *tiene complejidad* L''. *entonces la recurrencia* $\mathbf{T} - \mathbf{S} = (T_1 - S_1, \ldots, T_r - S_r)$ *tiene una complejidad no mayor a* $L' + L''$.

Sea (Λ', L') una recurrencia lineal de longitud mínima que genera $\mathbf{S}$, sea (Λ'', L'') una recurrencia lineal de longitud mínima que genera $\mathbf{T}$. Entonces según 4.3

$$\Lambda'(x)S(x) = \Gamma'(x) + x^{r+1}\Theta'(x) \quad \deg \Gamma'(x) \leq L'$$
$$\Lambda''(x)T(x) = \Gamma''(x) + x^{r+1}\Theta''(x) \quad \deg \Gamma'(x) \leq L''$$

Sea L la longitud mínima de una recurrencia que genere $\mathbf{T} - \mathbf{S}$. Queremos ver que $L \leq L' + L''$.

Separemos dos casos, si $L' + L'' > r$ entonces como claramente $L \leq r$, se tiene que $L < L' + L''$.

De otro modo supongamos ahora que $L' + L'' \leq r$, entonces

$$\Lambda'(x)\Lambda''(x)[T(x) - S(x)] = [\Lambda'(x)\Gamma''(x) - \Lambda''(x)\Gamma'(x)] + x^{r+1}[\Lambda''(x)\Theta''(x) - \Lambda''(x)\Theta'(x)]$$

Esta ecuación cumple con la forma de la ecuación 4.3 que determinaría que $(\Lambda'(x)\Lambda''(x), L'+L'')$ produce la secuencia $\mathbf{T}-\mathbf{S}$. (Esta recurrencia solo tiene sentido considerarla en el caso en que la longitud $(L' + L'')$ es menor que el tamaño del vector r)

Hemos encontrado una recurrencia de longitud $L' + L''$ que produce $\mathbf{T} - \mathbf{S}$, luego $L \leq L' + L''$.

4.2. Algortimo de Peterson

En esta sección presentamos el algoritmo de Peterson, que fue el primero históricamente usado para la decodificación de códigos Reed Solomon. El algoritmo calcula las localizaciones de los errores a partir del síndrome de la palabra recibida, resolviendo un sistema lineal de ecuaciones. Viendo este sistema, será clara la relación con encontrar la mínima secuencia lineal que genera una secuencia dada. (Para este algoritmo retornamos a la referencia básica habitual [4].)

Teorema 10. *Sea* $l_1 = n-t-1-(k-1)$. *Sea* $(S_1, S_2, \ldots, S_{n-k}) = (r(x_1), r(x_2), \ldots, r(x_{n-k}))$ *el síndrome de la palabra recibida. Sea* $Q_1 = \sum_{i=0}^{l_1} Q_{1,i}x^i$ *un polinomio localizador de errores. Entonces se verifica el siguiente sistema de ecuaciones lineales:*

$$\begin{bmatrix} S_1 & S_2 & \ldots & S_{l_1+1} \\ S_2 & S_3 & \ldots & S_{l_1+2} \\ \vdots & \vdots & \vdots & \vdots \\ S_{l_1} & S_{l_1+1} & \ldots & S_{2l_1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.4}$$

Refiriéndonos a 2.4, habíamos determinado que Q_1 era un polinomio localizador de errores y que se cumplían las ecuaciones

$$0 = Q(x_i, r_i) = Q_0(x_i) + r_i Q_1(x_i) \quad \forall i$$

Escribiendo todas estas de forma matricial (siendo $l_0 = n - t - 1$ el grado del polinomio Q_0):

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{l_0} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{l_0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{l_0} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,l_0} \end{bmatrix} + \begin{bmatrix} r_1 & 0 & \ldots & 0 \\ 0 & r_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & r_n \end{bmatrix} \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{l_1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{l_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Se puede comprobar que multiplicando esta ecuación matricial por la izquierda por la siguiente matriz

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{l_1} & x_2^{l_1} & \dots & x_n^{l_1} \end{bmatrix}$$

El primer termino se anula ya que sus columnas son palabras del código y la matriz por la que estamos multiplicando tiene las l_1 primeras filas de la matriz de control habitual y el segundo pasa a ser la ecuación 4,4.

La idea clave que desarrollaremos más adelante con el algoritmo de Berlekamp-Massey es que el sistema de la ecuación 4.4 representa las condiciones que debe cumplir una recurrencia lineal **S** de longitud l_1 y vector de pesos

$$\mathbf{\Lambda} = (Q_{1,l_1-1}/Q_{1,l_1}, Q_{1,l_1-2}/Q_{1,l_1} \dots, Q_{1,0}/Q_{1,l_1}).$$

Por lo tanto si encontramos la recurrencia lineal de longitud mínima que genere la secuencia de síndromes tendremos el polinomio localizador de errores.

Para finalizar la decodificación necesitamos calcular la magnitud del error a partir de los sindromes también aprovechando su propiedad clave, que dependen exclusivamente del error: $S_j = e(x_j)$. Sabiendo las posiciones de los errores $i_1, \dots, i_t$ entonces

$$\begin{bmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_t} \\ x_{i_1}^2 & x_{i_2}^2 & \dots & x_{i_t}^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_{i_1}^t & x_{i_2}^t & \dots & x_{i_t}^t \end{bmatrix} \begin{bmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_t} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_t \end{bmatrix} \tag{4.5}$$

El algoritmo de Peterson, de forma esquemática, es el siguiente:

Algoritmo 2. Algoritmo de Peterson

- Calcular los síndromes a partir de la palabra recibida.
- Resolver el sistema lineal 4.4 para calcular el polinomio localizador de errores.
- Factorizar el polinomio para encontrar las posiciones de los errores. (Es razonable en muchos casos evaluar en las n posibilidades para encontrar las raíces.)
- Resolver el sistema lineal 4.5 para calcular la magnitud del error.

Ejemplo 29. Consideremos el código $RS[7,3]$ con elementos en $\mathbf{F}_8$ y α un elemento primitivo con $\alpha^3 + \alpha + 1 = 0$. El código tiene polinomio generador $g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)$. Supongamos que recibimos la palabra $r(x) = x^5 + \alpha^6 x^3 + x^2 + \alpha^2 x + \alpha^6$:

- Calculamos los síndromes $\mathbf{S} = (r(\alpha), r(\alpha^2), r(\alpha^3), r(\alpha^4)) = (1, 1, \alpha^5, 1)$.
- $l_1 = n - t - 1 - (k-1) = 2$. Resolvemos el sistema

$$\begin{bmatrix} 1 & 1 & \alpha^5 \\ 1 & \alpha^5 & 1 \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

  Podemos buscarlo mónico con $Q_{1,2} = 1$,

$$Q_{1,0} + Q_{1,1} + \alpha^5 = 0$$

$$Q_{1,0} + \alpha^5 Q_{1,1} + 1 = 0$$

  Con lo que $Q_{1,1} = 1$ y $Q_{1,0} = (\alpha^5 + 1)$ y $Q_1(x) = x^2 + x + (\alpha^5 + 1)$.
- Las raíces de este polinomio son α y α^3 con lo que los errores están en los coeficientes de grado 1 y 3.
- Finalizar encontrando la magnitud del error.

$$\begin{bmatrix} \alpha & \alpha^3 \\ \alpha^2 & \alpha^6 \end{bmatrix} \begin{bmatrix} e_1 \\ e_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

  Luego $e_1 = e_3 = 1$, hay que restar 1 de los coeficientes de grados 1 y 3.

La palabra corregida es $x^5 + \alpha^2 x^3 + x^2 + \alpha^6 x + \alpha^6$.

4.3. Berlekamp-Massey

Como ya hemos visto el algoritmo de Peterson encuentra el polinomio localizador de errores resolviendo un sistema lineal. (Complejidad algoritmica $O(n^3)$). Sin embargo podemos aprovechar la estructura de la matriz que caracteriza un polinomio localizador de errores según el teorema 10 y resolver ese sistema de una manera conceptualmente más compleja pero algorítmicamente más rápida: tratándolo como una recurrencia lineal.

Contamos con $2t$ síndromes ($n-k \leq 2t$), el problema queda reducido a encontrar entonces la recurrencia lineal de menor grado que genera la secuencia $S_1, S_2, \ldots, S_{2t}$.

$$S_k = -\sum_{j=1}^{\nu} \Lambda_j S_{k-j}, \quad k = \nu+1, \ldots, 2t$$

Si se determina que el grado de la recurrencia ν es mayor que la capacidad correctora t o que $\Lambda_\nu = 0$ entonces el polinomio $\Lambda(x)$ no sería un posible polinomio localizador de errores, en este caso sabríamos que la secuencia de síndromes recibida no podría ser resultado de un patrón de errores corregible (se habrían producido más de t errores) y por tanto la decodificación fallaría.

4.3.1. Concepto general del algoritmo

El proceso es inductivo; para cada $r = 1 \ldots, 2t$ se encontrará la recurrencia de menor longitud que produce la secuencia truncada $S_1, \ldots, S_r$; $(\Lambda^r(x), L_r)$. En el siguiente paso se comprueba si la recurrencia hallada en el paso anterior sigue sirviendo, es decir se calcula el siguiente término $\hat{S}_r$ y se comprueba si coincide con S_r. En caso afirmativo $(\Lambda^{r+1}(x), L_{r+1}) = (\Lambda^r(x), L_r)$. En otro caso podemos *actualizar* la recurrencia para acomodar al nuevo síndrome.

La discrepancia r-ésima será un valor que caracterizará si la recurrencia en el paso $r-1$ sigue sirviendo para el siguiente paso de la iteración. Se define como la resta del síndrome r-ésimo real y el producido por la recurrencia $(\Lambda^{r-1}(x), L_{r-1})$, denotado $\hat{S}_r$. Por tanto solo habrá necesidad de actualizar la recurrencia si $\Delta_r \neq 0$.

$$\Delta_r = S_r - \hat{S}_r = S_r + \sum_{j=1}^{L_{r-1}} \Lambda_j^{r-1} S_{r-j} = \sum_{j=0}^{L_{r-1}} \Lambda_j^{r-1} S_{r-j}$$

Veamos en qué consiste el proceso de *actualizar* la recurrencia en el caso en que $\Delta_r \neq 0$. Primero, escojemos la iteración más reciente en que aumentó la longitud de la recurrencia, $(\Lambda^m(x), L_m)$, $L_m > L_{m-1}$, esto implicará en particular que $\Delta_m \neq 0$. Pongamos también $l = r - m$.

La nueva recurrencia va a venir dada, (de manera polinómica por):

$$\Lambda^r(x) = \Lambda^{r-1}(x) - \frac{\Delta_r}{\Delta_m} x^l \Lambda^{(m-1)}(x)$$

y la longitud será por lo tanto, observando el posible grado de este polinomio, igual a

$$L_r = \text{máx}\{L_{r-1}, l + L_{m-1}\}$$

Veamos que efectivamente la discrepancia es nula para el índice r, pero también para el resto de índices $L_r < j \leq r$ Esto es genera un término más, además de los que ya generaba la recurrencia anterior.

Es claro que si para $\Lambda(x)$ la discrepancia r-ésima es Δ_r y para $\Lambda'(x)$ es Δ'_r entonces para la recurrencia dada por $A\Lambda(x)+B\Lambda'(x)$ es $A\Delta_r+B\Delta'_r(x)$ (*Linealidad de la discrepancia*).

Podemos estudiar las discrepancias de la nueva recurrencia estudiando las discrepancias para $x^l\Lambda^{m-1}(x)$, ya que ya conocemos las de $\Lambda^{r-1}(x)$. Para $x^l\Lambda^{m-1}(x)$ los primeros l coeficientes son 0, entonces la discrepancia r-ésima será

$$\sum_{j=0}^{L_{m-1}} \Lambda_j^{m-1} S_{r-j-l} = \sum_{j=0}^{L_{m-1}} \Lambda_j^{m-1} S_{m-j} = \Delta_m$$

Entonces la nueva discrepancia es

$$\Delta'_r = \Delta_r - \frac{\Delta_r}{\Delta_m}\Delta_m = 0$$

Para otro índice $j > L_r = \text{máx}\{L_{r-1}, l + L_{m-1}\} > l$ la recurrencia sigue produciendo los elementos deseados ya que el término introducido, gracias a que la elección de l coincide con la discrepancia de $(\Lambda^{m-1}(x), L_{m-1})$ en $j - l < m$, es decir se anula.

Con esto tenemos que la recurrencia se irá actualizando para generar todos los síndromes. Falta comprobar que el algoritmo en efecto produce la recurrencia de menor longitud.

4.3.2. Prueba de recurrencia de longitud mínima

Lema 5. *Si* $(\Lambda^{r-1}(x), L_{r-1})$ *es una recurrencia lineal de longitud mínima que produce* $S_1, \ldots, S_{r-1}$ *y* $(\Lambda^r(x), L_r)$ *es una recurrencia lineal de longitud mínima que produce* $S_1, \ldots, S_r$ *con* $\Lambda^r \neq \Lambda^{r-1}$*, entonces*

$$L_r \geq \text{máx}\{L_{r-1}, r - L_{r-1}\}$$

Se tiene que L_{r-1} es la mínima longitud posible para una recurrencia que produzca los $r-1$ primeros valores, pero si $(\Lambda^r(x), L_r)$ también los produce entonces

$$L_r \geq L_{r-1}$$

Falta ver $L_r \geq r - L_{r-1}$. Sea $\mathbf{\Delta}$ la secuencia de las discrepancias de la recurrencia $(\Lambda^{r-1}(x), L_{r-1})$, $(0, 0, \ldots, 0, \Delta_r)$ donde $\Delta_r \neq 0$ ya que $\Lambda^r \neq \Lambda^{r-1}$. $\mathbf{\Delta}$ solo puede ser producida por una secuencia de longitud r. Se tiene que $(\Lambda^{r-1}(x), L_{r-1})$, produce $\mathbf{S} + \mathbf{\Delta}$, y $(\Lambda^r(x), L_r)$ produce $\mathbf{S}$ entonces podemos aplicar 4 a estas dos recurrencias $\mathbf{\Delta} = \mathbf{S} + \mathbf{\Delta} - \mathbf{S}$. De esto tenemos $r \leq L_{r-1} + L_r$ luego $L_r \geq r - L_{r-1}$ y se tiene el resultado.

Este lema nos permite concluir que si encontramos una recurrencia lineal que alcance la igualdad en esta desigualdad, entonces tendrá que ser la de mínima longitud.

Volviendo al algoritmo, vamos a ver que este construye iterativamente recurrencias que cumplen esta igualdad. Partimos de $r-1$ recurrencias ya construidas $(\Lambda^i(x), L_i)$ que producen cada una los i primeros síndromes. Por inducción, nuestra hipótesis va a ser

$$L_i = \text{máx}\{L_{i-1}, i - L_{i-1}\} \quad si \quad \Lambda^i \neq \Lambda^{i-1}$$

Esto es cierto para el primer i tal que S_i es no nulo porque entonces $L_{i-1} = 0$ y $L_i = i$, esto se debe a que una secuencia inicial de ceros solo puede generar una secuencia de ceros.

Siguiendo la notación que habíamos usado para la presentación del algoritmo, sea m el índice más reciente en que cambió la longitud de la recurrencia. Esto es $L_{r-1} = L_m > L_{m-1}$, por otra parte por inducción $L_m = \text{máx}\{L_{m-1}, m - L_{m-1}\}$ y como $L_m > L_{m-1}$ entonces $L_{r-1} = L_m = m - L_{m-1}$. Nos interesa el caso en que $\Delta_r \neq 0$, cuando hay que definir una nueva recurrencia que genere hasta r síndromes, se hace de la siguiente manera:

$$\Lambda^r(x) = \Lambda^{r-1}(x) - \Delta_r \Delta_m^{-1} x^{r-m} \Lambda^{m-1}(x)$$

Como el grado de este polinomio es la longitud de la recurrencia tenemos que

$$L_r \leq \text{máx}\{L_{r-1}, r - m + L_{m-1}\} \leq \text{máx}\{L_{r-1}, r - L_{r-1}\} \implies L_r = \text{máx}\{L_{r-1}, r - L_{r-1}\}$$

Debiéndose la igualdad a que por el lema 5 ya teníamos la desigualdad contraria.

Entonces como hemos visto esta recurrencia es de longitud mínima, en particular el producto final del algoritmo también es la recurrencia de longitud mínima que genera la secuencia de todos los $2t$ síndromes.

4.3.3. Algortimo en pseudocódigo

Ahora que se ha presentado y justificado el método vamos a dar una descripción detallada del algoritmo:

Algoritmo 3. Berlekamp-Massey
Input: Vector de síndromes $(S_1, S_2, \ldots, S_{2t})$
Output: Polinomio de pesos $\Lambda(x)$

```
(Inicializaciones)
Λ(x) ← 1
B(x) ← 1 (El término que se utiliza para actualizar
 Δ_m^{-1} x^{r-m} Λ_m(x))
l ← 0
r ← 0
mientras r ≠ 2t hacer
    r ← r + 1
    Δ ← Σ_{j=0}^{l} Λ_j S_{r-j}
    si Δ ≠ 0 entonces
        (La recursión tiene que actualizarse)
        Λ'(x) ← Λ(x) − ΔxB(x)
        (En cada paso vimos que L_r = máx{L_{r-1}, r − L_{r-1}} con lo
         que para que la longitud cambie en una iteración
         L_{r-1} < r − L_{r-1}, en términos del algoritmo 2l < r)
        si 2l < r entonces
            (Cambia l, cambia Λ_m(x))
            l ← r − l
            B(x) ← Δ^{-1}Λ(x)
        en otro caso
            B(x) ← xB(x)
        fin
        Λ(x) ← Λ'(x)
    en otro caso
        B(x) ← xB(x)
    fin
fin
devolver Λ(x)
```

La terminación del algoritmo está garantizada ya que se realizan siempre $2t$ iteraciones. Por otra parte, la complejidad del algoritmo será $O(t^2)$, ya que para cada iteración tanto el calculo de la discrepancia como la actualización de los polinomios $\Lambda(x)$, $B(x)$ requieren a la sumo $O(t)$ operaciones, y se realizan $2t$ iteraciones. Esto es una mejoría notable en comparación con la complejidad cúbica de resolver el sistema lineal del algoritmo de Peterson.

Ejemplo 30. Consideremos en el cuerpo $\mathbb{F}_8$ un elemento primitivo α con $\alpha^3+\alpha+1=0$ y la secuencia $\mathbf{S}=(1,1,\alpha^5,1)$. Apliquemos el algoritmo. En la siguiente tabla se muestra el valor las variables durante cada iteración.

r	Δ	$\Lambda(x)$	$B(x)$
1	1	$1+x$	1
2	0	$1+x$	x
3	α^4	$1+x+\alpha^4x^2$	$\alpha^3+\alpha^3x$
4	0	$1+x+\alpha^4x^2$	$\alpha^3x+\alpha^3x^2$

Se puede comprobar que efectivamente la recurrencia dada por $\Lambda(x)=1+x+\alpha^4x^2$ genera la secuencia.

4.3.4. Decodificación usando Berlekamp-Massey

Como ya hemos visto al compararlo con el algoritmo de Peterson, para aplicar Berlekamp-Massey a la decodificación de códigos Reed-Solomon se debe hallar la recurrencia lineal que genera el vector de los $2t$ síndromes. Del teorema 10 y la definición con la que estamos trabajando de recurrencia lineal, teníamos que $\mathbf{\Lambda}=(Q_{1,l_1-1}/Q_{1,l_1}, Q_{1,l_1-2}/Q_{1,l_1}\ldots, Q_{1,0}/Q_{1,l_1})$, que en forma polinómica resulta

$$\Lambda(x)=1+Q_{1,l_1-1}/Q_{1,l_1}x+Q_{1,l_1-2}/Q_{1,l_1}x^2+\cdots+Q_{1,0}/Q_{1,l_1}x^{l_1}$$

El polinomio $Q_{1,l_1}\Lambda(x)$ tiene justamente los coeficientes en orden inverso al producido por el algoritmo de Peterson, debido a la manera en que se han definido las recurrencias lineales. Para deshacer este cambio podemos aplicar la siguiente transformación $Q_1(x)=Q_{1,l_1}x^{l_1}\Lambda(x^{-1})$, con lo que concluimos que si β es una raíz de $Q_1(x)$ entonces β^{-1} lo es de $\Lambda(x)$ y podemos hallar las posiciones de los errores igualmente factorizando este polinomio.

Finalmente al igual que para el algoritmo de Peterson, se deberá aplicar un método para hallar la magnitud de los errores una vez conocidas sus posiciones. (Por ejemplo 4.5)

Ejemplo 31. Descodificamos lo mismo que en el ejemplo 29 (algoritmo de Peterson) haciendo uso de Berlekamp-Massey.

- Calculamos los síndromes $\mathbf{S} = (r(\alpha), r(\alpha^2), r(\alpha^3), r(\alpha^4)) = (1, 1, \alpha^5, 1)$.
- Aplicamos Berlekamp-Massey y obtenemos $\Lambda(x) = 1 + x + \alpha^4 x^2$. Ver el ejemplo 30.
- Factorizamos $\Lambda(x)$, y obtenemos las raices α^{-1}, α^{-3}. Con lo que los errores están en los coeficientes de grado 1 y 3.
- Finalizar encontrando la magnitud del error. (Igual que en el ejemplo del algoritmo de Peterson 29).

5. DECODIFICAR CON LISTA

En este capítulo planteamos la posibilidad, de corregir más allá del límite en la capacidad correctora dada por la distancia mínima $t = \left\lfloor \frac{d-1}{2} \right\rfloor$. Además de la referencia habitual ([4]) se ha consultado [14], que es la referencia original.

5.1. Concepto y motivación

El concepto de decodificación con lista fue introducido independiente por Elias y Wozencraft a finales de los años 50 en las publicaciones [15] [16], al permitir más errores que los dados por la capacidad correctora existe la posibilidad de que se encuentre más de una palabra candidata. De manera más formal:

Definición 16. (Algoritmo de decodificación con lista) Un algoritmo de decodificación con lista toma como argumentos la palabra recibida r, además de una capacidad correctora τ; y resulta en una lista de a lo sumo l palabras del código que difieren de r en como mucho τ posiciones. (Distan de r menos de τ en el sentido de Hamming.)

Si la lista de palabras posibles es suficientemente pequeña, es un intento de recuperación razonable: podría usarse otra información contextual u otro código para realizar la decisión, o en el peor de los casos elegir una de ellas al azar. De hecho veremos que, si τ tiene un valor moderado, el caso más común es en el que una lista de este tipo contiene una única palabra y ninguna consideración adicional es necesaria, pero de cualquier forma obtener una lista de candidatos es mejor que un fallo en la decodificación.

Corregir t errores es una propiedad garantizada del código, siempre se podrá hacer, pero es sencillo modificar $t + 1$ símbolos de una palabra del código de manera que el decodificador se equivoque. Lo normal, sin embargo, en el contexto de una aplicación real de los códigos correctores (por ejemplo superar interferencias en una transmisión inalámbrica, o permitir que un DVD funcione con un rayón), es que el error introducido sea ruido aleatorio o en ráfaga (errores seguidos) no que alguien modifique la palabra de la manera exacta para acercarse más a otra. Este es el peor de los casos (modelo adversarial de introducción de errores). Podemos intentar mejorar la capacidad correctora en la *mayoría* de los casos.

5.2. Distribución de pesos

En esta sección vamos a presentar la importancia de la distribución de pesos en un código más allá del peso mínimo que da la distancia mínima, en el contexto de la decodificación con lista y también de manera general.

Para comenzar observemos que una manera equivalente de caracterizar la distancia mínima es como el máximo valor para el cuál las bolas de radio d con centro en cualquier palabra, no contienen ninguna otra palabra.

$$d = \text{máx}\{\delta \in \mathbb{N} : B(c, \delta) \cap C = \{c\}, \quad \forall c \in C\}$$

Siguiendo esta línea, como vamos a considerar todas las palabras que disten menos de τ de una palabra recibida, convendrá estudiar cuántas palabras hay en una bola de radio $2\tau + 1$ con centro en otra palabra. Ya que la existencia de una palabra en esta bola implica la existencia de un error de peso a lo sumo τ que puede impedir la corrección única:

Sea $c_1 \in C$, si existe otra palabra $c_2 \neq c_1$ en

$$B(c_1, 2\tau + 1) \cap C$$

entonces existira un vector (error) e con peso a lo sumo τ que coincide en suficientes posiciones con el vector $c_2 - c_1$ de manera que

$$d(c_2, c_1 + e) \leq d(c_1, c_1 + e)$$

Nos interesa por lo tanto que estas bolas tengan el menor número posible de palabras del código. De manera análoga al estudio de la distancia mínima gracias a la linealidad, esto va a ser equivalente a estudiar la distribución de pesos de las palabras del código (bolas centradas en el origen). Definimos entonces la siguiente cantidad

$$v_{2\tau+1} = |B(0, 2\tau + 1) \cap C|$$

Que será un indicador de la probabilidad de decodificación única para una capacidad correctora. Tenemos v_d=1, que significa que la la decodificación es única para t.

Estudiemos el caso concreto en que queremos aumentar la capacidad correctora en 1, esto es $\tau = t + 1$. Suponemos que se introduce un error de peso τ de manera totalmente aleatoria, es decir todas las posiciones tienen la misma probabilidad de contener un error, y la magnitud del error también está uniformemente distribuida para cada posición.

En este caso en la bola $B(0, 2\tau + 1)$ existen dos tipos de palabras no nulas del código, las que tienen peso d y las que tienen peso $d + 1$. De la misma manera, existirán 2 tipos

de errores con peso τ que puedan hacer que la palabra recibida sea incorregible: los que coinciden en $\tau - 1$ posiciones no nulas con una palabra de peso d y los que coinciden en τ posiciones no nulas con una palabra de peso $d + 1$.

- En el primer caso, sea $\delta \in C$ con $w(\delta) = d$, $e \in \mathbb{F}_q^n$ con $w(e) = \tau$ de tal manera que coincida en al menos $\tau - 1$ símbolos con δ. Entonces si $c \in C$ es la palabra enviada

$$d(c + \delta, c + e) \leq d - (\tau - 1) = t + 1 = \tau = d(c, c + e)$$

- En el segundo caso, sea $\delta \in C$ con $w(\delta) = d + 1$, $e \in \mathbb{F}_q^n$ con $w(e) = \tau$ y $d(\delta, e) = d + 1 - \tau$. De esta manera si $c \in C$ es la palabra enviada y se recibe $c + e$, entonces $d(c + \delta, c + e) = d + 1 - \tau = t + 1 = \tau = d(c, c + e)$. Tendríamos 2 opciones equidistantes de la palabra recibida.

Ejemplo 32. Consideramos el código $RS\,[15, 3, 13]$ sobre el cuerpo $\mathbb{F}_{16}$ y elemento primitivo α que cumple $\alpha^4 + \alpha + 1$. Sea c la palabra generada por el polinomio $\alpha^{11}x^2 + \alpha x + \alpha^9$.

$$c = (\alpha^5, \alpha^4, \alpha^4, \alpha^{13}, \alpha^{12}, \alpha^9, \alpha^2, \alpha^3, \alpha^{12}, \alpha^2, \alpha^5, \alpha^{13}, 1, \alpha^3, 1)$$

Consideramos ahora una palabra $\delta \in RS\,[15, 3, 13]$ con $w(\delta) = d = 13$

$$\delta = (\alpha^8, \alpha^4, \alpha^5, \alpha^{11}, \alpha^8, \alpha^5, \alpha^7, \alpha^3, 0, \alpha^{11}, 0, \alpha^7, \alpha^{13}, \alpha^{13}, \alpha^3)$$

Y el error

$$e = (\alpha^8, \alpha^4, 0, 0, 0, \alpha^5, 0, \alpha^3, 0, 0, 0, \alpha^7, \alpha^{13}, \alpha, \alpha^3)$$

que observamos coincide en $t = 6$ símbolos con δ e introduce un error adicional (en rojo). Este es un error del primer tipo.
En este caso un algoritmo decodificador con lista con $\tau = 7$ que intentara descodificar $r = c + e$, devolvería tanto c como $c + \delta$ ya que ambas van a distar 7 de la palabra recibida.

Observamos también que por virtud de la distancia mínima estos errores no se solapan, es decir los dados por una palabra concreta δ no serán dados por otra palabra.

Sea $A_i = |c \in C : w(c) = i|$, el número de palabras de un código $C[n, k]$ que tienen peso i, entonces el número de errores del primer tipo es

$$A_d \binom{d}{\tau - 1} (n - (\tau - 1))(q - 1)$$

Y del segundo tipo

$$A_{d+1}\binom{d+1}{\tau}$$

Finalmente el número de errores posibles viene dado por

$$\binom{n}{\tau}(q-1)^{\tau}$$

Con lo que la probabilidad de que un algoritmo de decodificacion con lista produzca un resultado con más de 1 palabra es

$$\frac{A_d\binom{d}{\tau-1}(n-(\tau-1))(q-1)+A_{d+1}\binom{d+1}{\tau}}{\binom{n}{\tau}(q-1)^{\tau}}$$

Ejemplo 33. Tomemos como ejemplo un código $RS\,[15,3]$.

Una palabra de peso $d = 13$ será la correspondiente a un polinomio con dos raíces distintas de cero que tendrá la forma

$$c(x-\beta)(x-\gamma) \quad c,\gamma,\beta \in \mathbb{F}_{16} \setminus \{0\}$$

Tenemos entonces la elección de dos raices y un coeficiente no nulos.

$$A_d = 15\binom{15}{2} = 1575$$

Por otra parte una palabra de peso $d+1 = 14$ será la correspondiente a un polinomio con una raíz distinta de cero. Estos son los polinomios lineales de la forma

$$c(x-\beta) \quad c,\beta \in \mathbb{F}_{16} \setminus \{0\}$$

Pero también pueden ser estos mismos multiplicados por x, o por su respectivo $(x-\beta)$ si tiene la raíz con multiplicidad 2. Tenemos entonces la elección de una raíz y un coeficiente no nulos. Y la elección entre multiplicar por x, $(x-\beta)$ o 1.

$$A_{d+1} = 15 \times 15 \times 3 = 675$$

Realizando los cálculos concluimos que la probabilidad de que un error que haga que un algoritmo decodificador con lista devuelva más de una palabra, bajo la hipótesis de error aleatorio (Se producen exáctamente 7 errores, de magnitud y posición aleatoria) es **0.000333959**, que es extremadamente baja. De media solo 1 de cada 3000 palabras aproximadamente no es corregible de manera única.

Esta ha sido una primera aproximación sencilla, con cálculos propios, al problema, que nos permite ver la importancia de la distribución de pesos de un código y concluir que el caso de decodificación no única es extraño para un código y parámetros concretos. Para dar un resultado más general nos referimos a Nielsen y Høholdt en [17], vamos a ver el esquema allí propuesto (sin entrar en detalles) para calcular la probabilidad de decodificación no única para códigos $RS[n,k]$ sobre un cuerpo $\mathbb{F}_{11}$.

En esta publicación, la hipótesis sobre los errores es que todos los de peso menor o igual que τ son equiprobables mientras que en nuestro caso habíamos planteado que todos tuvieran peso exactamente τ.

El problema radica en encontrar la fracción de palabras de $B(0,\tau)$ (centrada en 0 por linealidad) que también estén a distancia al menos τ de otra palabra. Primero se da una cota de este valor por

$$M(\tau) = \sum_{c \in RS[n,k] \setminus \{0\}} \left| \bar{B}(0,\tau) \cap \bar{B}(c,\tau) \right|$$

Después se dan expresiones explícitas mediante argumentos combinatorios de las siguientes cantidades

- El número de palabras en la intersección de esferas, que se usará para calcular el número de palabras en la intersección de bolas.

  $$|S(0,a) \cap S(0,i)|$$

  Donde $S(c,r)$ es la esfera de centro c y radio r.

- Número de palabras de peso i, la distribución de pesos.

  $$A_i = \binom{n}{i} \sum_{j=0}^{i-d} (-1)^j \binom{i}{j} q^{(i-d+1-j)} - 1$$

Y finalmente se puede calcular $M(\tau)$ por

$$M(\tau) = \sum_{u=d}^{\min 2\tau, n} A_u \sum_{a=u-\tau}^{\tau} \sum_{i=u-a}^{\tau} |S(0,a) \cap S(0,i)|$$

para dar una cota superior de la probabilidad de decodificación no única

$$\frac{M(\tau)}{|B(0,\tau)|}$$

La expresión de la cota es compleja y dificil de estimar pero Nielsen y Høholdt concluyen que normalmente es muy pequeña. Por ejemplo realizando el cálculo exacto para $n = 63$, para distintas dimensiones del código, muestran que la cota es como mucho del orden de $10^{-}5$.

5.3. Algoritmo de Sudan

Este algoritmo, presentado originalmente por Madhu Sudan en [18], expande la idea presentada en la sección 2.4 generalizándola para un algoritmo que devuelva una lista con las palabras más cercanas. Se sigue el mismo esquema general:

- **Interpolación.** Se construye un polinomio en dos variables que interpola la palabra recibida.
- **Factorización.** Se buscan factores lineales $(y - f(x))$.

5.3.1. Idea y validez

Al igual que para el algoritmo de decodificación única vamos a considerar $r = c + e$ con r la palabra recibida, c la palabra original y e el error. En este caso consideraremos dos parámetros adicionales para el algoritmo, τ que será su capacidad correctora (posiblemente mayor que la dada por la distancia mínima t) y l que será la longitud máxima de la lista resultante.

Con esto se determina un polinomio no nulo de la forma

$$Q(x, y) = Q_0(x) + Q_1(x)y + Q_2(x)y^2 + \cdots + Q_l(x)y^l$$

Cumpliendo

- $Q(x_i, r_i) = 0, i = 1, 2, \ldots, n.$
- $\deg Q_j(x) \leq n - \tau - 1 - j(k-1) = l_j, j = 0, 1, \ldots, l.$

Teorema 11. *Sea $f(x)$ tal que $\deg f(x) \leq k$ y tal que $d(f(x_1), \ldots, f(x_n), r) < \tau$, entonces*

$$(y - f(x))|Q(x, y).$$

Esto significa que podemos encontrar todas las palabras que disten menos de τ de la recibida factorizando el polinomio Q.

La prueba de este resultado sigue un argumento similar al visto en la sección 2.4, por la elección de los grados de los polinomios constituyentes Q_j tenemos que $\deg Q(x, f(x)) \leq n - \tau - 1$ y que $Q(x_i, f(x_i)) = 0$ en al menos $n - \tau$ posiciones. Entonces $Q(x, f(x))$ es el polinomio nulo, lo que equivale a que $(y - f(x))$ sea un factor.

5.3.2. Condiciones sobre los parámetros del código

Ahora queda la siguiente cuestión ¿Para qué elecciones de parámetros existe este polinomio Q? De manera análoga de nuevo al caso particular de decodificación única, estudiamos el sistema lineal, número de ecuaciones y número de incógnitas, dado por las condiciones impuestas en los coeficientes.

Consideramos

$$Q_j(x) = \sum_{r=0}^{l_j} Q_{j,r} x^r$$

Entonces el sistema lineal a resolver para poder computar el polinomio $Q(x, y)$ es

$$\sum_{j=0}^{l} \begin{pmatrix} r_1^j & \dots & 0 & 0 \\ 0 & r_2^j & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & r_n^j \end{pmatrix} \begin{pmatrix} 1 & x_1 & \dots & x_1^{l_j} \\ 1 & x_2 & \dots & x_2^{l_j} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^{l_j} \end{pmatrix} \begin{pmatrix} Q_{j,0} \\ Q_{j,1} \\ \vdots \\ Q_{j,l_j} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{5.1}$$

El número de ecuaciones es n, para que el sistema tenga una solución no nula necesitamos un número estrictamente mayor de incógnitas.

El número de incógnitas (coeficientes) es

$$\sum_{j=0}^{l} l_j + 1 = \sum_{j=0}^{l} n - \tau - j(k-1) = (l+1)(n-\tau) - (k-1)\sum_{j=0}^{l} j =$$

$$(l+1)(n-\tau) - \frac{1}{2}(k-1)l(l+1)$$

Esto impone las siguiente condición sobre τ.

$$\tau < n\frac{l}{l+1} - \frac{l}{2}(k-1) \tag{5.2}$$

Podemos observar que si $l = 1$ entonces $\tau < \frac{n-k+1}{2} = \frac{d}{2}$ de manera que τ no podría ser mayor que t.

Además, para asegurar que $Q_l \neq 0$, (de la definición del grado de Q_l)

$$(n-\tau) - l(k-1) > 0 \tag{5.3}$$

Esto es $\tau < n - l(k-1)$, por otra parte queremos que $\tau > t = \lfloor (n-k)/2 \rfloor$. Entonces $n - l(k-1) > (n-k)/2$ de aquí podemos obtener una cota sobre la tasa del código. La tasa del código es la proporción de la palabra que es información útil k/n.

$$k/n < 1/(2l-1) + 2l/n \tag{5.4}$$

Consideramos códigos con tasa baja entonces, de hecho solo a partir de $k/n < 1/3$ podemos tomar $l \geq 2$, luego solo entonces tiene sentido usar este algoritmo.

Ahora partiendo vamos a buscar el tamaño de lista que maximizaría la cota dada en 5.2, obviando la integralidad por el momento podemos tomar la derivada en l e igualarla a 0 obteniendo así el valor

$$l = \sqrt{\frac{2n}{(k-1)}} - 1$$

Y sustituyendo este valor en 5.3, tenemos

$$\tau < n - \sqrt{2n(k-1)} - (k-1)$$

Que de manera asintótica, con n tendiendo a infinito y manteniendo una tasa de código $R = k/n$ constante nos da la expresión

$$\tau/n < 1 - \sqrt{2R} - R/2$$

Ejemplo 34. Considerando $l = 2$, $k/n < 1/3 + 2l/n$ es decir no más de un tercio de los simbolos contiene información y el resto son redundancia. Para poder usar este algoritmo necesitamos usar un código con tasa de información baja.
Para el código $RS\,[15, 3]$ comprobamos cual es el mayor τ para el cuál la ecuación 5.2.

$$\tau < 15\frac{2}{3} - 2 = 8$$

Entonces podemos elegir $\tau = 7$ que es superior a la capacidad correctora del código en una unidad.
También se cumple la ecuación 5.3. Podemos utilizar el algoritmo para un código $[15, 3]$ con $l = 2$ y $\tau = 7$.

5.3.3. Algoritmo en pseudocódigo

Presentamos el algoritmo en pseudocódigo.

Algoritmo 4. Sudan.
Suponiendo un código que cumple con los condiciones sobre los parámetros anteriores, encontrar $Q(x, y)$ una solución no nula de la ecuación 5.1, a partir de la palabra recibida r.

(Buscar factores $(y - f(x))$)
$fs \leftarrow \{f(x) \in \mathbb{P}_k : (y - f(x)) \mid Q(x, y)\}$
$lista \leftarrow$
para cada $f \in fs$ **hacer**
  $c \leftarrow (f(x_1), f(x_2), \ldots, f(x_n))$
  si $d(c, r) \leq \tau$ **entonces**
    $lista \leftarrow lista \cup \{c\}$
  fin
fin
devolver $lista$

El teorema 11 garantiza que los polinomios buscados si existen se pueden encontrar como factores pero puede haber otros factores de este tipo que no sean los que buscamos, por eso se filtra comprobando la distancia a la palabra recibida.

La lista de palabras devuelta tiene a lo sumo l elementos, ya que ese es el grado en y del polinomio $Q(x, y)$. Puede ser vacía en cuyo caso tenemos un error de decodificación.

Ejemplo 35. Aplicamos el algoritmo de Sudan al ejemplo 32. En este caso $\tau = 7$ y $l = 2$.
Tenemos

$$r = (\alpha^4, 0, \alpha^4, \alpha^{13}, \alpha^{12}, \alpha^6, \alpha^2, 0, \alpha^{12}, \alpha^2, \alpha^5, \alpha^5, 1, \alpha^9, \alpha^{14})$$

Una solución del sistema dado por las ecuaciones 5.1 es

$$Q_0(x) = \alpha^2 + \alpha^{12}x + \alpha^3x^2 + \alpha^5x^4 + \alpha^9x^5 + \alpha^{14}x^6 + \alpha^2x^7$$

$$Q_1(x) = \alpha^{13} + \alpha^{11}x + \alpha^{10}x^2 + \alpha^4x^3 + \alpha^2x^4 + \alpha x^5$$

$$Q_2(x) = \alpha^9 + x^3$$

Seguidamente, se comprueba que $Q(x, y) = Q_0(x) + yQ_1(x) + y^2Q_2(x)$ se puede factorizar de la siguiente manera

$$Q(x, y) = (\alpha^9 + x^3)(y - (\alpha^9 + \alpha x + \alpha^{11}x^2))(y - (\alpha^{14} + \alpha^5 x + \alpha^6 x^2))$$

Con lo que los polinomios que producen las palabras más cercanas a las recibidas son

$$f_1 = \alpha^9 + \alpha x + \alpha^{11}x^2$$

$$f_2 = \alpha^{14} + \alpha^5 x + \alpha^6 x^2$$

Y las palabras son c_1, producida por f_1

$$c_1 = (\alpha^5, \alpha^4, \alpha^4, \alpha^{13}, \alpha^{12}, \alpha^9, \alpha^2, \alpha^3, \alpha^{12}, \alpha^2, \alpha^5, \alpha^{13}, 1, \alpha^3, 1)$$

Y c_2 producida por f_2

$$c_2 = (\alpha^4, 0, \alpha^8, \alpha^4, \alpha^9, \alpha^6, \alpha^{12}, 0, \alpha^{12}, \alpha^9, \alpha^5, \alpha^5, \alpha^6, \alpha^8, \alpha^{14})$$

Que coinciden cada una en 8 posiciones con la palabra recibida (marcadas en rojo), con lo que distan $7 = \tau$ de la palabra recibida y el resultado del algoritmo es

$$\{c_1, c_2\}$$

5.3.4. Factorización

La complejidad oculta en este proceso está en el paso de factorización. Investiguemos una posibilidad para encontrar factores de la forma requerida $(y - f(x))$.

Consideramos la ecuación

$$Q(x, u(x)) = 0$$

Para empezar podemos considerar está ecuación módulo x

$$Q(0, u(0)) = Q(0, u_0) = 0$$

Esto es, el término independiente de u, u_0, tiene que ser raíz del polinomio $Q(0, y)$.

Ahora, disponiendo del termino anterior tomando una de estas raíces, para calcular el siguiente coeficiente podemos considerar la ecuación módulo x^2:

$$Q(x, u_1 x + u_0) = 0$$

No conocemos todavía u_1 así que lo representamos con la incógnita y para considerar $Q'(x, y) = Q(x, xy + u_0)$. sabemos sin embargo que $Q'(0, y) = 0$ para todo y, por lo que no podemos hallar una solución directamente. Para evitar esto dividiremos repetidamente por x hasta que no sea un factor.

Podemos considerar m tal que $x^m | Q'(x, y)$, y sin embargo $x^{m+1} \nmid Q'(x, y)$. En este caso ponemos $Q''(x, y) = x^{-m} Q'(x, y)$ y tenemos que u_1 va a ser una raíz de $Q''(0, y)$.

Es fácil observar que podemos aplicar iterativamente este método, planteando la ecuación original módulo potencias cada vez mayores de x, para obtener todos los coeficientes de una posible solución como raíces de estos polinomios.

Una iteración para hallar un coeficiente sigue estos pasos:

1. Hallar u_i como raíz de $M_i(0, y)$
2. M'_i=$M_i(x, xy + u_i)$
3. Hallar m_i tal que $x^m | M'_i(x, y)$ pero $x^{m+1} \nmid M'_i(x, y)$
4. $M_{i+1} = x^{-m_i} M'_i(x, y)$

En cada paso escogemos una raíz, pero puede haber varias. Esto permite organizar las soluciones con estructura de árbol, donde en cada nodo la elección de una raíz particular lleva a distintas soluciones. El proceso completo es el siguiente:

Algoritmo 5. Algoritmo para encontrar factores lineales en y $(y - f(x))$ de $Q(x, y)$.
Primero definimos dos operaciones auxiliares sobre polinomios

- $divX(M(x, y))$ que divide repetidamente el polinomio por x hasta que no sea un factor.
- $raices(M(0, y))$ que devuelve la lista de raíces del polinomio $M(0, y)$.

Y además una función recursiva $r(M(x, y), sol, k, m)$ para recorrer el árbol:

Función r($M(x, y)$, sol,k,m):
  (sol es un vector con los coeficientes de una solución u_i calculados hasta la iteración actual m.)
  si $m = k$ **entonces**
    (Se han calculado ya suficientes coeficientes, se devuelve una solución. Caso base.)
    $f(x) \leftarrow \sum_{i=0}^{k-1} sol[i]x^i$
    $sols \leftarrow \{f(x)\}$
  en otro caso
    (Se recogen las soluciones de las ramas con llamadas recursivas.)
    $sols \leftarrow \{\}$
    para cada $u \in raices(M(0, y))$ **hacer**
      $sols \leftarrow sols \cup r(divX(M(x, xy + u)), sol + u, k, m + 1)$
      (sol + u indica concatenar el elemento u al vector)
    fin
  fin
  devolver $sols$
fin

Finalmente el algoritmo se puede expresar en términos de una sola llamada a esta función:

Input: $Q(x, y)$ y k según lo visto previamente.

Output: Lista de polinomios de grado k, $f(x)$ tales que $(y - f(x))$ es factor $Q(x, y)$.

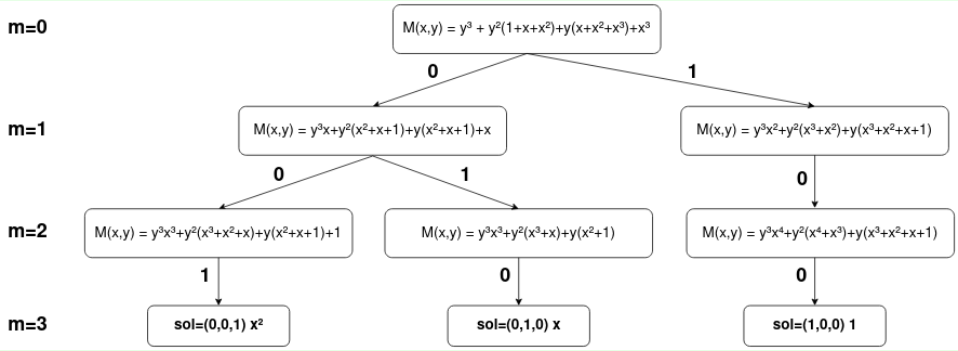
devolver $r(divX(Q(x, y)), (), k, 0)$

La terminación del algoritmo viene de que solo puede haber a lo sumo l factores de este tipo.

Ejemplo 36. Vamos a aplicar el algoritmo para factorizar el polinomio con coeficientes en $\mathbb{F}_2$

$$Q(x,y) = (y-1)(y-x)(y-x^2) = y^3 + y^2(1+x+x^2) + y(x+x^2+x^3) + x^3.$$

La siguiente figura es una representación de la traza del algoritmo, cada nodo tiene el polinomio $M(x,y)$ con el que se está trabajando y se bifurca según las raíces de $M(0,y)$.



5.4. Sudan-Guruswami

En esta sección se introduce una mejora del algoritmo de Sudan, debido al propio Sudan junto con su entonces estudiante de doctorado Guruswami publicado en [19]. Supone una mejora en cuanto a las condiciones sobre los parámetros resultando en mejores capacidades correctoras para códigos con la misma tasa (k/n) en comparación con el algoritmo de Sudan.

La base sigue siendo la misma, interpolación y factorización, pero esta vez las raices $Q(x_i, r(x_i))$ serán de multiplicidad $r > 1$. Necesitamos definir con precisión esta noción de multiplicidad.

5.4.1. Conceptos previos

Definición 17. (Multiplicidad de una raíz). Decimos que $Q(x,y)$ tiene una raíz de multiplicidad r en (x_0, y_0) si los coeficientes de los monomios de grado $< r$ del polinomio trasladado $Q(x+x_0, y+y_0)$ son todos 0.

Observamos que los coeficientes de un polinomio trasladado $Q(x+x_0, y+y_0)$ son una función lineal de los coeficientes del polinomio original. Esto es útil para definir el sistema

lineal que habrá que resolver para obtener un polinomio interpolador.

Ejemplo 37. El polinomio $Q(x, y) = (x + y - 2)(x - y)$ tiene una raíz de multiplicidad 2 en $(1, 1)$.
El polinomio traladado es $Q(x + 1, y + 1) = (x + y)(x - y) = x^2 - y^2$ que no tiene monomios de grado < 2.

Necesitamos definir también el grado pesado de un polinomio con 2 variables.

Definición 18. (Grado (w_1, w_2)). El grado (w_1, w_2) de un monomio $x^n y^m$ es

$$\deg_{(w_1,w_2)} x^n y^m = w_1 n + w_2 m$$

El grado (w_1, w_2) de un polinomio es el máximo grado de entre los de sus monomios.

5.4.2. Idea y validez

Dada una palabra recibida $r(x)$, una multiplicidad r y un tamaño de lista l, las condiciones sobre el polinomio interpolador $Q(x, y)$ son las siguientes

- $Q(x, y) \neq 0$
- $(x_i, r(x_i))$ es una raíz de multiplicidad r de $Q(x, y)$.
- $\deg_{(1,k-1)} Q(x, y) \leq l$

Comprobamos ahora que este polinomio, tiene los factores que necesitamos para la corrección. Si τ es la capacidad correctora que asignamos al algoritmo entonces sea $t = n - \tau$ el número de posiciones en que coincidirán, al menos, la palabra recibida y la palabra enviada.

Teorema 12. *Si $p(x)$ es un polinomio con* $\deg p(x) < k$ *tal que*

$$|\{i \in \{1, 2, \ldots, n\} : r(x_i) = p(x_i)\}| \geq t$$

y $rt > l$, entonces $(y - p(x))|Q(x, y)$.

Para empezar veamos que si $p(x)$ es un polinomio tal que $r(x_i) = p(x_i)$ entonces $(x - x_i)^r \mid Q(x, p(x))$.

Sea $p'(x) = p(x + x_i) - r(x_i)$ de manera que $p'(0) = 0$. Entonces $x \mid p'(x)$ con lo que podemos poner $p''(x) = p'(x)/x$. Sea $g(x) = Q(x + x_i, p'(x) + r(x_i))$, entonces

$$Q(x, p(x)) = Q(x, p'(x - x_i) + r(x_i)) = g(x - x_i)$$

Sabemos que $Q(x + x_i, y + r(x_i))$ no tiene monomios de grado $< r$, de esta manera sustituyendo $y = p'(x) = xp''(x)$ tenemos $g'(x)$ y podemos ver que $x^r \mid g(x)$ luego $(x - x_i)^r \mid g(x - x_i) = Q(x, p(x))$.

Observamos que $\deg_{(1,k)} Q(x, y) \leq l$ implica que el grado de $Q(x, p(x))$ es a lo sumo l. Por otra parte sabemos ahora que $(x - x_i)^r \mid Q(x, p(x))$, luego si

$$S = \{i \in \{1, 2, \ldots, n\} : r(x_i) = p(x_i)\},$$

entonces

$$\prod_{i \in S} (x - x_i)^r \mid Q(x, p(x)).$$

Ahora bajo las hipótesis del teorema, dado que $|S| \geq t$ entonces el grado de $\prod_{i \in S} (x - x_i)^r$ será al menos rt que divide al polinomio $Q(x, p(x))$ de grado a lo sumo $l < r\tau$. Esto solo se puede dar si $Q(x, p(x)) = 0$, es decir $(y - p(x)) \mid Q(x, y)$.

Pasamos ahora a estudiar la existencia de este polinomio $Q(x, y)$. Como en otras ocasiones, estudiamos el número de ecuaciones y el número de incógnitas del sistema lineal que se ha de resolver para computar los coeficientes de $Q(x, y)$.

En cuanto a las ecuaciones, para cada uno de los n polinomios trasladados a los puntos $(x_i, r(x_i))$, tenemos que todos los monomios de grado $< r$ tienen que ser nulos. Los monomios $x^j y^m$ con grado $< r$ cumplen

$$j, m \geq 0 \quad j + m < r$$

Y existen $\frac{r(r+1)}{2}$ monomios que cumplen con esto, por cada uno de estos polinomios trasladados dando un total para las ecuaciones de

$$n\left(\frac{r(r+1)}{2}\right)$$

En cuanto a incógnitas, tenemos que contar el número de monomios que cumple con la restricción del grado (con peso) de $Q(x, y)$, es decir

$$j, m \geq 0 \quad j + km < l$$

que podemos acotar por la expresión

$$\left(\frac{l}{k}\right)\left(\frac{l+2}{2}\right)$$

La restricción sobre los parámetros que garantiza la existencia de $Q(x, y)$ es entonces

$$n\left(\frac{r(r+1)}{2}\right) < \frac{l(l+2)}{2k}$$

5.4.3. Condiciones sobre los parámetros

Veamos ahora como elegir más concretamente estos parámetros.

Para cumplir con $rt > l$ elegimos $l = rt - 1$. Ahora para satisfacer la ecuación anterior observamos que es equivalente a

$$r^2t^2 - 1 > kn(r^2) + r$$

Y a su vez

$$r^2(t^2 - kn) - knr - 1 > 0$$

Para cumplir esta ecuación hacemos notar que se tiene que dar ($t^2 - kn > 0$) o lo que es lo mismo $t > \sqrt{kn}$. Esto es importante por que limita la capacidad correctora a $n - \sqrt{kn}$, que de manera asintótica en función de la tasa $R = k/n$

$$\tau/n < 1 - \sqrt{R}$$

Que es mejor que la cota equivalente para el algoritmo de Sudan, $1 - \sqrt{2R} - (R/2)$, siempre que $R < 2(\sqrt{2} - 1)$, lo que se garantiza en el caso de Sudan por las restricciones que impone en la tasa del código (5.4).

Finalmente se puede escoger r como un entero mayor que la mayor de las raices de la ecuación cuadrática anterior, lo que garantiza que se cumpla la condición. La raíz más grande de $r^2(t^2 - kn) - knr - 1 = 0$ es

$$\frac{kn + \sqrt{k^2n^2 + 4(t^2 - kn)}}{2(t^2 - kn)}$$

y por tanto escogemos r un entero inmediatamente superior.

$$r = 1 + \left\lfloor \frac{kn + \sqrt{k^2n^2 + 4(t^2 - kn)}}{2(t^2 - kn)} \right\rfloor$$

Ejemplo 38. Si queremos aplicar el algoritmo a un código $RS[7, 2]$ y queremos que pueda corregir un error más que el dado por la distancia mínima esto es $t = 3$, entonces $\tau = 4$.
Con la expresión dada para r, tenemos $r = 8$ y $l = rt - 1 = 31$. A pesar de que el algoritmo de Sudan-Guruswami mejora la capacidad correctora asintóticamente con respecto al algoritmo de Sudan en la práctica resulta muy costoso ya que los parámetros del algoritmo (multiplicidad, tamaño de lista) son grandes incluso para un código pequeño para el cual queremos obtener una mejora de un error corregible más.

6. CÓDIGOS PRODUCTO

La idea de un código producto es sencilla, una palabra de un código producto se puede tratar como una matriz en la que las filas son palabras de un código consitituyente y las columnas son palabras de otro. El concepto fue introducido por primera vez por Elias en 1954 en [20].

El interés de estos códigos está en la manera en que se puede aprovechar esta estructura para corregir más allá de la distancia mínima. Por ejemplo si el código de las filas corrige hasta t errores, entonces si los errores están aislados en como mucho t columnas no importa cuántos errores hay en estas; corregiría bien este tipo de errores contiguos (error de ráfaga). Esto también es testimonio de que hay patrones de errores que aún teniendo un peso grande son corregibles y de que los códigos producto tienen buenas cualidades para corregir más allá de la distancia mínima. La referencia principal para este capítulo es [21] además de [6].

Comenzamos dando la definición formal de código producto con la que trabajaremos.

Definición 19. (Código producto) Sea $\mathcal{A}$, $\mathcal{B}$ dos códigos con parámetros respectivamente $[n_A, k_A, d_A]$ y $[n_B, k_B, d_B]$, tales que $\mathcal{A} \subset \mathbb{F}_q^{n_A}$ y $\mathcal{B} \subset \mathbb{F}_q^{n_B}$. El código producto $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ es el conjunto de matrices (n_B, n_A) tales que sus filas son palabras del código $\mathcal{A}$ y sus columnas son palabras del código $\mathcal{B}$.

6.1. Codificación

En esta sección trabajamos con $\mathcal{C} = \mathcal{A}[n_A, k_A, d_A] \times \mathcal{B}[n_B, k_B, d_B]$

Si tenemos una codificación sistemática para cada código, entonces podemos codificar una matriz de (k_B, k_A) símbolos de mensaje, codificando primero las filas usando $\mathcal{A}$ y después las columnas (tanto las originales como las resultantes de los símbolos de paridad de las filas codificadas) usando $\mathcal{B}$. Esta codificación es sistemática para el nuevo código y la matriz codificada resultante tiene la siguiente estructura:

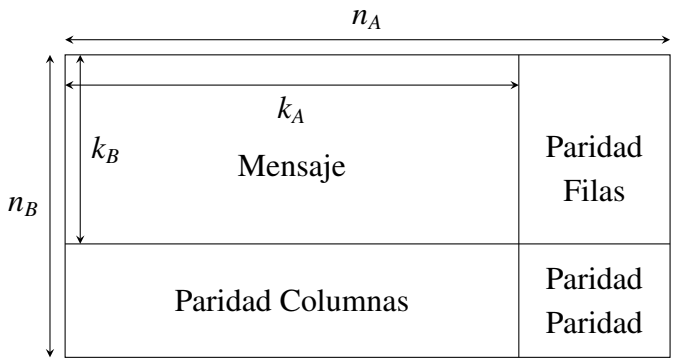


Figura 6.1: Estructura de una codificación sistemática de un código producto. Un diagrama similar se encuentra en la publicación original de Elias [20]

Ejemplo 39. Consideramos $\mathcal{A} = \mathcal{B} = RS[7,4]$ y $C = \mathcal{A} \times \mathcal{B}$. Trabajamos sobre el cuerpo $\mathbb{F}_8$ tomando un elemento α que cumple $\alpha^3 = \alpha + 1$. Vamos a codificar de forma sistemática la matriz (palabra)

$$a = \begin{pmatrix} \alpha^6 & \alpha^2 & \alpha^5 & 1 \\ \alpha^4 & 0 & 1 & \alpha^6 \\ \alpha^3 & \alpha^6 & \alpha^5 & \alpha^6 \\ \alpha^4 & \alpha^6 & \alpha^3 & \alpha^2 \end{pmatrix}$$

Codificamos primero sistemáticamente cada una de las filas resultando en

$$a' = \begin{pmatrix} \alpha^6 & \alpha^2 & \alpha^5 & 1 & \alpha^6 & \alpha^3 & \alpha^6 \\ \alpha^4 & 0 & 1 & \alpha^6 & \alpha^6 & \alpha^3 & \alpha^6 \\ \alpha^3 & \alpha^6 & \alpha^5 & \alpha^6 & \alpha^3 & 0 & \alpha^6 \\ \alpha^4 & \alpha^6 & \alpha^3 & \alpha^2 & \alpha^6 & 1 & \alpha^3 \end{pmatrix}$$

Y por último las columnas

$$c = \begin{pmatrix} \alpha^6 & \alpha^2 & \alpha^5 & 1 & \alpha^6 & \alpha^3 & \alpha^6 \\ \alpha^4 & 0 & 1 & \alpha^6 & \alpha^6 & \alpha^3 & \alpha^6 \\ \alpha^3 & \alpha^6 & \alpha^5 & \alpha^6 & \alpha^3 & 0 & \alpha^6 \\ \alpha^4 & \alpha^6 & \alpha^3 & \alpha^2 & \alpha^6 & 1 & \alpha^3 \\ \alpha^5 & \alpha^3 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^4 & \alpha^4 \\ \alpha^5 & \alpha^5 & \alpha^4 & \alpha^3 & 0 & 0 & \alpha \\ \alpha^5 & \alpha^5 & 1 & \alpha^4 & 1 & 0 & \alpha^4 \end{pmatrix}$$

$$c \in C$$

De manera general, dadas las codificaciones por dos matrices generadoras de $\mathcal{A}$, G_A y de

$\mathcal{B}$, G_B de los códigos constituyentes entonces una codificación de C es

$$\begin{aligned} \mathcal{M}(k_B, k_A) &\to \mathcal{M}(n_B, n_A) \\ u &\mapsto G_B^T u G_A \end{aligned} \tag{6.1}$$

La dimensión de G_B^T es (n_B, k_B) luego $G_B^T u$ tiene dimensión (n_B, k_A). El producto final resulta entonces de aplicar G_A a cada una de las filas de longitud k_A de $G_B^T u$ y por lo tanto sus filas son palabras del código $\mathcal{A}$. Análogamente multiplicar $G_B^T(uG_A)$ es aplicar G_B a las columnas de uG_A y por lo tanto sus columnas son palabras del código $\mathcal{B}$.

La inyectividad de esta aplicación viene dada de la inyecyividad de $u \mapsto uG_A$ y $u \mapsto uG_B$. Esta inyectividad está en principio dada para cuando se aplican las generadoras a un vector, pero se extiende fácilmente a aplicarlas a una matriz.

Si tenemos las matrices generadoras en forma $G_A = (I_{k_A}|A)$, $G_B = (I_{k_B}|B)$, de manera que representan una codificación sistemática de los códigos constituyentes entonces la codificación dada por 6.1 es la misma que la presentada al comienzo de la sección:

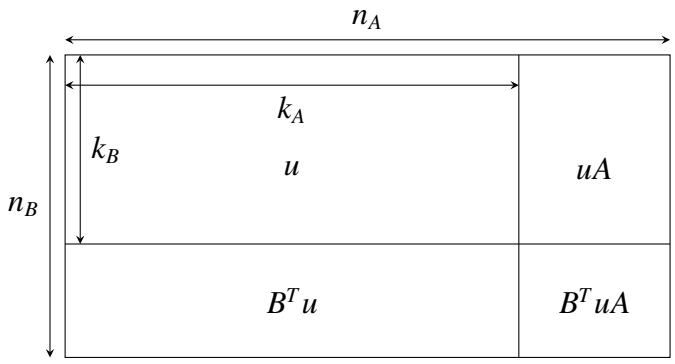


Figura 6.2: Resultado del producto $G_A u G_B$ en le caso sistemático

Ejemplo 40. Vamos a codificar una palabra con el producto de dos códigos $RS[3,2]$, trabajando sobre $\mathbb{F}_4$ tomando como elemento primitivo α que cumple $\alpha^2 = \alpha + 1$.

Una matriz generadora de los códigos constituyentes es

$$G_A = G_B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix}$$

Vamos a codificar la palabra

$$u = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Entonces

$$uG_A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix} = \begin{pmatrix} 0 & \alpha^2 & \alpha \\ 0 & \alpha^2 & \alpha \end{pmatrix}$$

$$G_B^T(uG_A) = \begin{pmatrix} 1 & 1 \\ 1 & \alpha \\ 1 & \alpha^2 \end{pmatrix}\begin{pmatrix} 0 & \alpha^2 & \alpha \\ 0 & \alpha^2 & \alpha \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \alpha & 1 \\ 0 & 1 & \alpha^2 \end{pmatrix}$$

6.2. Parámetros del código

Veremos ahora que los parámetros del código son el producto de los parametros de los códigos constituyentes como cabría esperar.

Teorema 13. *El producto de códigos* $\mathcal{A}[n_A, k_A, d_A] \times \mathcal{B}[n_B, k_B, d_B]$ *es un código* C *con parámetros* $[n_A n_B, k_A k_B, d_A d_B]$.

Como hemos visto a partir de sendas codificaciones de los códigos constituyentes se puede obtener una codificación del código producto

$$\mathcal{M}(k_B, k_A) \to C \subset \mathcal{M}(n_B, n_A)$$

y de aquí es obvio que la dimensión del código es $k_A k_B$ y la longitud $n_A n_B$.

En cuanto a la distancia minima veamos primero que mayor o igual que $d_A d_B$. Sea $c \in C$ no nula, entonces una de sus columnas será no nula y al ser una palabra del código $\mathcal{B}$ tiene d_B elementos no nulos. Las filas correspondientes a esas posiciones no nulas son palabras del código $\mathcal{A}$ y tienen d_A elementos no nulos. Por lo tanto las palabras no nulas del código

tienen peso como mínimo $d_A d_B$.

Buscamos ahora una palabra con exactamente ese peso. Sea $a \in \mathcal{A}$ con $w(a) = d_A$ y $b \in \mathcal{B}$ con $w(b) = d_B$. Vamos a justificar que podemos suponer que las posiciones no nulas son las iniciales en ambas palabras. Si esto no fuera así podemos considerar los códigos resultantes de permutar las palabras de tal manera que se satisfaga esa condición, es obvio que el producto de estos códigos tendrá la misma distancia mínima, la misma distribución de pesos, que el producto de los códigos originales. De esta manera pasamos a considerar la siguiente palabra

$$c = \begin{pmatrix} b_1a_1 & b_1a_2 & \dots & b_1a_{d_A} & 0 & \dots & 0 \\ b_2a_1 & b_2a_2 & \dots & b_2a_{d_A} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{d_B}a_1 & b_{d_B}a_2 & \dots & b_{d_B}a_{d_A} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}$$

Es claro que c es una palabra de C, ya que las filas son múltiplos de a y las columnas múltiplos de b. Por lo tanto dado que $w(c) = d_A d_b$ podemos concluir que la distancia mínima de C es $d_A d_B$.

Ejemplo 41. Continuando con el ejemplo anterior, los parámetros del código producto serán entonces

$$[9, 4, 4]$$

Observamos que el producto de dos códigos MDS ($n - k + 1 = d$) no es MDS.

Vamos a construir una palabra de peso 4 como en el ejemplo, usando la palabra $(\alpha, 1, 0) \in RS[3, 2]$.

$$c = \begin{pmatrix} \alpha^2 & \alpha & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

6.3. Matriz generadora

Para hablar de la matriz generadora de un código producto tenemos que *aplanar* las palabras pasando de matrices a vectores consistentes de las filas, una detrás de otra. Entonces la matriz generatiz de C consiste en

$$G_C = G_B \otimes G_A$$

El producto matricial usado se define de la siguiente manera. Siendo L y M matrices de dimensiones (a, b) y (c, d) respectivamente. El resultado será una matriz de dimensiones (ad, bc).

$$L \otimes M = \begin{bmatrix} L_{11}M & L_{12}M & \dots & L_{1b}M \\ L_{21}M & L_{22}M & \dots & L_{2b}M \\ \vdots & \vdots & \vdots & \vdots \\ L_{a1}M & L_{a2}M & \dots & L_{ab}M \end{bmatrix}$$

Donde cada entrada de la matriz en la definición es una submatriz (c, d).

Veamos que genera palabras del código, sea $u = (u_1, u_2, \dots, u_{k_A})$ con u_i de longitud k_B cada uno, que representan las filas del mensaje a codificar.

$$uG_C = uG_B \otimes G_A = (\sum_{i=1}^{k_B} G_{B,(i,1)} u_i G_A, \sum_{i=1}^{k_B} G_{B,(i,2)} u_i G_A, \dots, \sum_{i=1}^{k_B} G_{B,(i,n_B)} u_i G_A)$$

Donde de nuevo cada una de las componentes de este vector representa una de las n_A filas del resultado. Observemos que cada una de ellas es combinación lineal de las palabras del código $\mathcal{B}$ $u_i G_B$ y por tanto son palabras de $\mathcal{B}$.

Falta ver que las columnas son palabras del código $\mathcal{B}$. Por ejemplo la primera columna resulta de considerar el primer elemento de cada fila. Sea a_i el primer elemento de $u_i G_A$ entonces la primera columna

$$\begin{bmatrix} (\sum_{i=1}^{k_B} G_{B,(i,1)} a_i \\ (\sum_{i=1}^{k_B} G_{B,(i,2)} a_i \\ \vdots \\ (\sum_{i=1}^{k_B} G_{B,(i,n_B)} a_i \end{bmatrix} = \begin{bmatrix} G_{B,(1,1)} \\ G_{B,(1,2)} \\ \vdots \\ G_{B,(1,n_B)} \end{bmatrix} a_1 + \begin{bmatrix} G_{B,(2,1)} \\ G_{B,(2,2)} \\ \vdots \\ G_{B,(2,n_B)} \end{bmatrix} a_2 + \cdots + \begin{bmatrix} G_{B,(k_B,1)} \\ G_{B,(k_B,2)} \\ \vdots \\ G_{B,(k_B,n_B)} \end{bmatrix} a_{k_B}$$

La columna es combinación lineal de columnas de la matriz generadora G_B, luego es una palabra del código. Se puede razonar de manera análoga para todo el resto de columnas.

La codificación mediante las matrices generadoras de los códigos constituyentes resulta más sencilla utilizando la expresión ya vista

$$G_B^T u G_A$$

que es equivalente a usar la matriz generadora sobre la matriz u hecha vector.

Ejemplo 42. De nuevo continuando el ejemplo anterior, vamos a calcular su matriz generadora.

Recordamos que una matriz generadora del código constituyente es

$$G_A = G_B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix}$$

Entonces la matriz generadora del producto es

$$G_A \otimes G_B = \begin{pmatrix} 1\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix} & 1\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix} & 1\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix} \\ 1\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix} & \alpha\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix} & \alpha^2\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{pmatrix} \end{pmatrix}$$

$$G_A \otimes G_B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & 1 & \alpha & \alpha^2 & 1 & \alpha & \alpha^2 \\ 1 & 1 & 1 & \alpha & \alpha & \alpha & \alpha^2 & \alpha^2 & \alpha^2 \\ 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & 1 & \alpha^2 & 1 & \alpha \end{pmatrix}$$

Vamos a codificar $u = (1, 1, 1, 1)$ usando esta matriz

$$u(G_A \otimes G_B) = (0, 0, 0, 0, \alpha, 1, 0, 1, \alpha^2)$$

Que de forma matricial es

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & \alpha & 1 \\ 0 & 1 & \alpha^2 \end{pmatrix}$$

Vemos que es equivalente a usar la codificación dada por $G_B^T u G_A$.

6.4. Códigos concatenados

En esta sección vamos a seguir la sección 5.5 de [22].

Se introducen ahora los códigos concatenados que son una manera más general que los códigos producto de combinar códigos para obtener uno de longitud superior. Veremos de que manera un código producto se puede ver como un código concatenado.

Esta sección se incluye para demostrar, bajo esta perspectiva que lo hace más sencillo, la capacidad de los códigos producto para recuperarse de errores de ráfaga.

Definición 20. (Código concatenado) La concatenación de dos códigos $\mathcal{A}[n_A, k_A, d_A]$ sobre $\mathbb{F}_q$ y $\mathcal{B}[n_B, k_B, d_B]$ sobre $\mathbb{F}_{q^{k_A}}$ mediante una biyección lineal en $\mathbb{F}_q$

$$\psi : \mathbb{F}_{q^{k_A}} \to \mathcal{A}$$

es

$$C = \{(\psi(b1), \psi(b2), \ldots, \psi(b_{n_b})) : (b_1, b_2, \ldots, b_{n_b}) \in \mathcal{B}\}$$

Decimos que $\mathcal{A}$ es el código interior y $\mathcal{B}$ es el código exterior.

En esencia, un código concatenado es sustituir cada uno de los símbolos de una palabra del código exterior (elementos de $\mathbb{F}_{q^{k_A}}$) por una palabra del código interior.

Veamos cuales son los parámetros de estos códigos. Recordamos que una extensión se puede ver como un espacio vectorial sobre el cuerpo base ($(\mathbb{F}_q)^{k_A} \cong \mathbb{F}_{q^{k_A}}$), vamos a razonar mediante aplicaciones lineales biyectivas entre espacios vectoriales de $\mathbb{F}_q$.

$$\begin{aligned} ((\mathbb{F}_q)^{k_A})^{k_B} \xrightarrow{\quad f \quad} \mathcal{B} \subset ((\mathbb{F}_q)^{k_A})^{n_B} \xrightarrow{\quad \boldsymbol{\psi} \quad} (\mathcal{A})^{n_B} \subset ((\mathbb{F}_q)^{n_A})^{n_B} \\ (b_1, b_2, \ldots, b_n) \longmapsto (\psi(b_1), \psi(b_2), \ldots, \psi(b_n)) \end{aligned}$$

Donde f es una codificación de $\mathcal{B}$. Se puede observar que la dimensión del código resultante es $k_A k_B$ y la longitud $n_A n_B$.

Veamos ahora que esta construcción puede usarse para obtener un código producto. La clave para esto es de nuevo la estructura de espacio vectorial de una extensión sobre un cuerpo base.

$$(\mathbb{F}_q)^{k_A} \cong \mathbb{F}_{q^{k_A}}$$

Luego podemos interpretar una palabra de $\mathcal{B}$ como una matriz (n_B, k_A) de elementos de $\mathbb{F}_q$ si reemplazamos cada símbolo de $\mathbb{F}_{q^{k_A}}$ con una fila de k_A símbolos de $\mathbb{F}_q$.

Un código producto $\mathcal{A}[n_A, k_a] \times \mathcal{B}'[n_B, k_B]$ (ambos sobre $\mathbb{F}_q$) es equivalente a la concatenación de $\mathcal{A}$ y $\mathcal{B} = (\mathcal{B}')^{k_A}$. De esta manera B es el código de matrices (n_B, k_A) con elementos en $\mathbb{F}_q$ cuyas columnas son palabras de $\mathcal{B}'$. Ahora si fijamos una base para $\mathbb{F}_{q^{k_A}}$ como $\mathbb{F}_q$ espacio vectorial podemos interpretar $\mathcal{B}$ como un código $[n_B, k_B]$ sobre la extensión.

Por otra parte la aplicación ψ no es más que una codificación de $\mathcal{A}$ interpretando un elemento de la extensión como una palabra en $(\mathbb{F}_q)^{k_A}$.

Ejemplo 43. Continuando con nuestro ejemplo de código producto (ejemplo 40), veamos como se reinterpretaría en terminos de códigos concatenados. En este caso tenemos que el código exterior va a ser el de las matrices $(n_b, k_a) = (3, 2)$ cuyas columnas son palabras del código $RS\,[3, 2]$, pero lo vamos a ver como un código sobre la extensión $\mathbb{F}_{q^{k_A}} = \mathbb{F}_8 = \mathbb{F}_4[x]/(x^2 + x + 1)$. Representamos cada elemento de la extensión como un polinomio de grado < 2 con coeficientes en $\mathbb{F}_4$ y de esta manera podemos interpretar un par de elementos de $\mathbb{F}_4$ como un solo elemento de $\mathbb{F}_8$ usando la base $\{1, x\}$.

Por ejemplo la matriz

$$\begin{pmatrix} 0 & 0 \\ \alpha^2 & \alpha^2 \\ \alpha & \alpha \end{pmatrix}$$

Se interpreta como la palabra de $(0, \alpha^2 x + \alpha^2, \alpha x + \alpha)$ de un código $[3, 2]$ sobre $\mathbb{F}_8$. Ahora la aplicación ψ consiste en enviar cada uno de estos elementos a una palabra del código de las filas, que será el código interior; primero interpretando cada uno mediante la base como 2 elementos de $\mathbb{F}_4$ y después codificando estos. Por ejemplo $\alpha x + \alpha$ es (α, α) y se codifica por $(0, \alpha, 1)$.

Sin embargo un código concatenado no es siempre expresable como código producto como se demuestra en el ejemplo 44.

Ejemplo 44. Sea $\mathcal{B} = RS[3,2]$ sobre $\mathbb{F}_4$ y α un elemento primitivo cumpliendo $\alpha^2 = \alpha + 1$. Y sea ψ dada por

$$\begin{aligned}\psi :&\mathbb{F}_4 \to \mathcal{A} \subset \mathbb{F}_2^3\\ &1 \mapsto (1,0,1)\\ &\alpha \mapsto (0,1,1)\end{aligned}$$

$\mathcal{A}$ que es la imagen de esta aplicación es un código $[3,2]$.

Entonces la palabra correpondiente a $(0,\alpha^2,\alpha) \in \mathcal{B}$ en el código concatenado será entonces

$$(0,0,0 \quad 1,1,0 \quad 0,1,1)$$

De manera matricial $(0,\alpha^2,\alpha)$ sustituyendo cada elemento por su representación vectorial en $(\mathbb{F}_2)^2$ tomando la base $e_1 = 1$ y $e_2 = \alpha$

$$\begin{pmatrix}0 & 0\\ 1 & 1\\ 0 & 1\end{pmatrix}$$

Obtener la palabra a partir de esta forma no es más que codificar las filas

$$\begin{pmatrix}0 & 0 & 0\\ 1 & 1 & 0\\ 0 & 1 & 1\end{pmatrix}$$

Este código no es equivalente a un código producto ya que podemos encontrar una palabra que contenga cualquier columna es decir el código de las columnas es $[3,3]$ y por tanto el producto con el código de las filas $\mathcal{A}[3,2]$ tendría dimensión $[9,6]$ que no coincide con la dimensión del código concatenado.

Los códigos concatenados mejoran la capacidad correctora cuando se dan errores de ráfaga de la siguiente manera. Si se transmite el mensaje como una cadena de $n_A n_B$ símbolos de $\mathbb{F}_q$, una ráfaga de t errores sobre esta cadena se correspondrá a una ráfaga de a lo sumo $\lceil t/n_A \rceil + 1$ errores para el código exterior. Es decir el código interior agrupa los errores para el código exterior, que lo recibe como uno único independientemente de cuántos se hayan producido al nivel de $\mathbb{F}_q$.

Además de esta propiedad que acorta las ráfagas, el código interior también puede marcar las posiciones como error que éste no haya podido corregir para facilitar la corrección al código exterior.

Ejemplo 45. Continuando con el ejemplo anterior, supongamos que se produce un error en ráfaga en la palabra $(0,0,0 \quad 1,1,0 \quad 0,1,1)$, limitado a las 3 primeras posiciones resultando en

$$(1,1,1 \quad 1,1,0 \quad 0,1,1)$$

Durante la decodificación primero el código interior determina que las 6 últimas posiciones son correctas pero las 3 primeras contienen un error y pasa $(-,\alpha^2,\alpha)$ al código exterior, que gracias a conocer la posición del error puede corregir correctamente.

6.5. Algoritmo de decodificación simple. Iterativo

Presentamos un algoritmo de decodificación que aprovecha la estructura del código producto para obtener una gran capacidad correctora. El algoritmo consiste en decodificar alternativamente filas y columnas hasta que no se puedan corregir más errores. Decodificar una sola vez filas y columnas no es suficiente, pues el código columna puede corregir suficientes errores en una fila como para que una palabra que no se pudo corregir a priori ahora se pueda corregir.

Algoritmo 6. Decodificación iterativa de códigos producto.

Se supone una función de decodificación que cuente el número de errores que ha corregido, $decod_A$ para el código de las filas, $decod_B$ para el código de las columnas. Se supone que el número de errores corregidos en caso de error de decodificación es 0.
Input: u Matriz (n_A, n_B), palabra a corregir
Output: Matriz (n_A, n_B), palabra con la mayor cantidad posible de errores corregidos

```
iter ← 0
errores_filas = 1
errores_columnas = 1
mientras errores_filas ≠ 0 o errores_columnas ≠ 0 hacer
    si iter mód 2 = 0 entonces
        (Se corrige por filas)
        para cada r fila de u hacer
            corregida, e ← decod_A(r)
            r ← corregida
            errores_filas ← errores_filas + e
        fin
    en otro caso
        (Se corrige por columnas)
        para cada c columna de u hacer
            corregida, e ← decod_B(c)
            c ← corregida
            errores_columnas ← errores_columnas + e
        fin
    fin
    iter ← iter + 1
fin
devolver u
```

El algoritmo termina ya que eventualmente, o bien no quedan errores por corregir o no se pueden corregir más, sin embargo esto puede llevar unas cuantas iteraciones. En la siguiente sección estudiamos el número de iteraciones y errores corregibles.

6.5.1. Rendimiento con errores uniformemente distribuidos

Esta sección está basada en una similar que se presenta en [4].

Vamos a realizar un estudio del rendimiento de este algoritmo suponiendo que se introducen W errores de manera aleatoria en una palabra perteneciente a un código $[n,k,d] \times [n,k,d]$, siendo t la capacidad correctora del código constituyente. Suponemos ambos códigos iguales con el fin de facilitar la notación pero el argumento presentado es fácilmente generalizable a un código producto de códigos distintos.

Interesa considerar aquellos casos en que una fila (o columna) puede acumular más de t fallos. Es en estos casos que no se podrá corregir en la primera iteración.

Al principio hay n posibles filas donde puede ir a parar un error, todas ellas equiprobables.

El numero de errores en una fila concreta seguirá entonces una distribución binomial de parámetros $(W, \frac{1}{n})$.

Entonces la probabilidad de que una fila concreta contenga más de t errores es

$$p(t, n, W) = 1 - \sum_{i=0}^{t} \binom{W}{i} \left(\frac{1}{n}\right)^{i} \left(1 - \frac{1}{n}\right)^{W-i}$$

Vamos a calcular la media de errores corregidos. Por cada fila posible consideramos la variable aleatoria $Z_i = X_i Y_i$, donde Y_1 es el numero de errores en la fila correspondiente, que sabemos sigue $binom(W, \frac{1}{n})$; y $X_i = 1$ si la fila se puede corregir $(Y_i > t)$ y 0 en otro caso. De esta manera la media de errores corregidos será

$$m(t, n, W) = E\left(\sum Z_i\right) = nE(X_i Y_i) = n\left(\sum_{i=0}^{t} t\binom{W}{i} \left(\frac{1}{n}\right)^{i} \left(1 - \frac{1}{n}\right)^{W-i}\right)$$

Despúes de este paso repetiremos el mismo argumento para las columnas, sin embargo ahora tenemos que considerar que la iteración sobre las columnas tendrá que intentar corregir menos errores. De media $W - m(t, n, W)$.

A la hora de considerar sucesivas iteraciones, tenemos que tener encuenta que existen filas/columnas que ya han sido corregidas y que los errores se distribuyen entre el resto. Suponemos, y esto afectará a la capacidad predictiva del modelo en casos extremos, que los errores restantes vuelven a estar uniformemente distribuidos.

El proceso para calcular cuántas filas/columnas y errores se corrigen de media en cada iteración es el siguiente (algorítmicamente), usando las funciones $p(t, n, W), m(t, n, W)$ presentadas:

Algoritmo 7. Modelo de corrección de errores

Input: W, errores introducidos; t, capacidad correctora; n longitud de los códigos constituyentes.

$it \leftarrow 0$
$n\_filas \leftarrow n$;
$n\_columnas =\leftarrow n$;
mientras $W > 0$ **hacer**
  si $iter \mod 2 = 0$ **entonces**
    $W \leftarrow W - m(t, n\_filas, W)$
    $n\_filas \leftarrow \lfloor n\_filas * (1 - p(t, n\_filas, W)) \rfloor$
  en otro caso
    $W \leftarrow W - m(t, n\_columnas, W)$
    $n\_columnas \leftarrow \lfloor n\_columnas * (1 - p(t, n\_columnas, W)) \rfloor$
  fin
  $it \leftarrow it + 1$
fin

Podemos usar este modelo para estimar cuantas iteraciones se necesitarán.

Ejemplo 46. Tomemos el producto $RS[255, 223] \times RS[255, 223]$. La capacidad correctora de los códigos constituyentes es $t = 16$, supongamos además que se introduce una densidad de errores ρ, es decir $W = \rho n^2$.

Vamos a comparar las predicciones del modelo con una ejecución particular del algoritmo sobre un error generado con la densidad de error especificada. Vamos a comparar las predicciones del modelo con una ejecución particular del algoritmo sobre un error generado con la densidad de error especificada. En la tabla las entradas son de la forma [errores corregidos] | [bloques sin corregir].

$\rho = 0{,}07$	$it = 0(F)$	$it = 1(C)$	$it = 2(F)$
Modelo	1359 | 156	2571 | 33	622 | 0
Real	1320 | 159	2639 | 31	576 | 0

$\rho = 0{,}08$	$it = 0(F)$	$it = 1(C)$	$it = 2(F)$	$it = 3(C)$	$it = 4(F)$
Modelo	710 | 206	1426 | 151	1780 | 67	1270 | 1	17 | 1
Real	847 | 197	1525 | 142	1935 | 48	882 | 0	

En este rango de valores para la densidad el modelo se ajusta adecuadamente, sin embargo para $\rho = 0{,}09$ el ritmo al que se corrigen los errores en la realidad es distinto, pero el modelo sigue siendo util para predecir las iteraciones que necesitaremos.

Finalmente para una densidad de errores mayor no se consigue eliminar la totalidad de los errores en la realidad aunque el modelo predice que se conseguiría en muchas iteraciones. Esto se debe a la simplificación que hicimos considerando que los errores estaban de nuevo uniformemente distribuidos.

6.6. Algoritmo a partir de decodificar con lista

Este algoritmo se presenta en la referencia [6], que es la base para esta sección.

6.6.1. Representación alternativa

Sean $\mathcal{A}'$, $\mathcal{B}'$ los códigos constituyentes de parámetros (n_A, k_A, d_A) y (n_B, k_B, d_B) respectivamente. Construimos el código $\mathcal{A}$ como el conjunto de matrices (n_B, n_A) tales que cada fila es un elemento del código $\mathcal{A}'$, expresado de otra manera $\mathcal{A}$ es el producto cartesiano de $\mathcal{A}'$ consigo mismo n_B veces, $\mathcal{A} = (\mathcal{A}')^{n_B}$. De la misma manera $\mathcal{B}$ será el código con-

sistente de las matrices (n_B, n_A) tales que cada columna es un elemento de $\mathcal{B}'$. Podemos entonces describir el código producto como intersección de dos códigos,

$$C = \mathcal{A}' \times \mathcal{B}' = \mathcal{A} \cap \mathcal{B}$$

6.6.2. Algoritmo

Veamos como la anterior perspectiva puede ser útil a la hora de descodificar. Sea r una palabra recibida.

Para estimar la palabra enviada c, tenemos que buscar aquella del código que diste menos de r. Sea A la lista de palabras del código $\mathcal{A}$ que menos distan de la palabra recibida r ordenadas ascendentemente según la distancia a r. La palabra del código producto que menos diste de r será obviamente el primer miembro de esta lista que esté también en $\mathcal{B}$, ya que si existiera una palabra más cercana de C también estaría en $\mathcal{A}$ y aparecería primero en A.

Mediante un decodificador con lista para el código $\mathcal{A}$ podemos usar el otro código para decidir cual de las palabras de la lista es la adecuada.

Consideremos entonces el problema de obtener un decodificador con lista de longitud l para el código $\mathcal{A}$ a partir de un decodificador con lista de longitud l del código constituyente $\mathcal{A}'$.

Si decodificamos cada fila por separado tendremos l candidatos para cada fila, lo que hace un total de l^{n_B} palabras de $\mathcal{A}$. Existe entonces el problema de ordenar estos resultados de una manera computacionalmente eficiente, calcular todas las posibilidades y sus distancias a r no es una opción.

Se combinarán los resultados de las dos primeras filas, de manera que resulte en una lista de l posibilidades en total para ambos **ordenada** por distancia a las filas correspondientes de r, y a este resultado se irá agregando iterativamente de una en una los resultados del resto de filas.

La distancia de una palabra a r es la suma de la distancia de cada fila a la fila correspondiente de r, por lo que a la hora de combinar lo que hay que tener en cuenta es la suma de las distancias de ambos resultados.

El siguiente algoritmo toma dos listas ordenadas u, v de tamaño l y devuelve una lista ordenada de tamaño l de combinaciones.

Algoritmo 8. Combinación de listas de distancias

INPUT: Listas ordenadas de enteros de longitud l, u y v.
OUTPUT: Lista de tuplas (i, j, n) de longitud l, ordenada por $n = u_i + v_j$.

(Lista ordenada por tercer componente)
$g = \{(i, 0, u_i) : i \in \{1, 2, \ldots, l\}\}$
$f = \{\}$
$i = 1$
mientras $i \leq l$ **hacer**
  $z \leftarrow$ primer elemento de g
  $g \leftarrow g \setminus \{z\}$
  si $z_2 = 0$ **entonces**
    (El resultado debe ser combinación de un elemento de cada lista)
    $z_2 \leftarrow < -1$
    $z_3 \leftarrow u_{z_1} + v_{z_2}$
    $g \leftarrow g \cup \{z\}$ (Ordenadamente)
  en otro caso
    $f \leftarrow f \cup \{z\}$
    $i \leftarrow i + 1$
    $z_2 \leftarrow z_2 + 1$
    si $z_2 \leq l$ **entonces**
      $z_3 = u_{z_1} + v_{z_2}$
      $g \leftarrow g \cup \{z\}$ (Ordenadamente)
    fin
  fin
fin
devolver f

Ejemplo 47. Supongamos que un decodificador con lista para el código de las filas devuelve listas de longitud 4, y obtiene unas posibilidades para la primera y segunda filas respectivamente, que distan lo siguiente de las filas correspondientes de la palabra recibida

$$(0, 2, 3, 5)$$

$$(0, 1, 1, 3)$$

Entonces el algoritmo resultará en

$$(0, 0, 0), (0, 1, 1), (0, 2, 1), (1, 0, 2)$$

Falta adaptar el algoritmo para que combine una lista de matrices (i, n_A) con una de filas $(1, n_A)$ para devolver una lista de matrices $(i+1, n_A)$. Esto no es más que ordenar las listas por distancia a la parte correspondiente de la palabra recibida y pasar estas listas de distancias como input al algoritmo anterior, después se crean las combinaciones según los índices devueltos por este. Llamemos a esta adaptación *combinar*.

Ya tenemos todos los ingredientes para dar una descripción general del algoritmo, un decodificardor en lista del código de las filas $\mathcal{A}'$, $decod_A$, una manera de comprobar la pertenencia al código de las columnas $\mathcal{B}'$ y el algoritmo que combinando los resultados de decodificar con lista las filas puede obtener una decodificación con lista de $\mathcal{A}$.

Algoritmo 9. Decodificar Producto

INPUT: M matriz (n_B, n_A) a decodificar
OUTPUT: $C \in \mathcal{C}$ más cercana a M, o error

(Decodificar con lista $\mathcal{A}$)
$lista \leftarrow decod_A$(fila 1 de M)
$i \leftarrow 1$
mientras $i \leq n_B$ **hacer**
  $lista \leftarrow combinar(parcial, decod_A$(fila i de M))
  $i \leftarrow i + 1$
fin
para cada $e \in lista$ **hacer**
  si $e \in \mathcal{B}'$ **entonces**
    devolver e
  fin
fin
devolver *Error*

Dado que todas las listas son ordenadas de la misma longitud l que el output de $decod_A$, el algoritmo termina y para un l suficientemente grande encontrará la palabra más cercana. En otro caso se da un error de decodificación. La complejidad del algoritmo está dominada por la complejidad de decodificar con lista el código de las filas, que dependerá del tamaño de este código y del tamaño de lista que fijemos; y el número de filas. Al combinar las posibilidades manteniendo siempre l en consideración y al utilizar el código de las columnas simplemente como una comprobación, evitamos que la complejidad escale.

Ejemplo 48. En el contexto del código producto $RS[7,4] \times RS[7,6]$ sobre el cuerpo $\mathbb{F}_8$ con elemento primitivo α cumpliendo $\alpha^3 = \alpha + 1$. Como el código de las filas es pequeño se ha construido un decodificador con lista general que devuelve todas las palabras del código ordenadas por distancia a otra dada. Esto nos permite poner un tamaño de lista l tan grande como queramos.

A continuación se da el resultado de una aplicar este algoritmo a la matriz con $l = 30$.

$$M = \begin{pmatrix} \alpha^3 & \alpha & \alpha^3 & 1 & \alpha & 1 & 0 \\ \alpha^4 & \alpha^4 & \alpha & 1 & \alpha & \alpha^4 & \alpha \\ \alpha^5 & \alpha^2 & \alpha^4 & \alpha^4 & 1 & \alpha & \alpha^5 \\ 0 & \alpha^5 & \alpha^6 & \alpha^6 & \alpha^3 & \alpha^6 & \alpha^3 \\ \alpha^3 & \alpha & \alpha^3 & \alpha & 0 & \alpha^3 & 0 \\ \alpha & 1 & \alpha^6 & \alpha^3 & \alpha^6 & 0 & \alpha^3 \\ \alpha^5 & \alpha^5 & \alpha^4 & 1 & \alpha^3 & \alpha^6 & \alpha^4 \end{pmatrix}$$

Después de obtener la lista correspondiente al código $\mathcal{A} = (RS[7,4])^7$, usamos el código de las columnas para decidirnos por una de las opciones. El resultado es

$$C = \begin{pmatrix} 1 & \alpha & \alpha^4 & 1 & \alpha & 1 & 0 \\ \alpha^4 & \alpha^4 & \alpha & 1 & \alpha & \alpha^6 & 1 \\ \alpha & \alpha^2 & \alpha^4 & \alpha & 1 & \alpha^4 & \alpha^5 \\ 0 & 1 & \alpha^6 & \alpha^6 & \alpha^3 & \alpha^6 & \alpha^4 \\ \alpha^3 & \alpha & \alpha & \alpha & 0 & \alpha^3 & \alpha^4 \\ \alpha & 1 & \alpha^6 & \alpha^3 & \alpha^2 & 0 & \alpha^4 \\ \alpha & \alpha^4 & \alpha^4 & 1 & \alpha^3 & \alpha^5 & \alpha^4 \end{pmatrix}$$

que dista 16 de la palabra recibida.

6.7. Codificación en CDs y DVDs

En esta sección presentaremos cómo se utilizan los códigos Reed-Solomon en CDs y DVDs. Para CDs nos referimos a la sección 5.6 de [22]. Para la codificación de DVDs nos referimos a [23].

A la hora de codificar los datos de un disco, tenemos que considerar que el tipo de error más común es de ráfaga (rayones que afectan datos contiguos). Cómo es común en otras aplicaciones, se trabajará sobre el cuerpo $\mathbb{F}_{2^8}$ que representará un byte.

Vamos a ver un concepto útil para la corrección de errores en ráfaga: intercalación (*inter-*

leaving).

6.7.1. Intercalación

Esta es otra herramienta utilizada para limitar el impacto de los errores en ráfaga. La idea es sencilla, si tenemos un número t de palabras de un código $C[n,k]$ y queremos evitar los errores de ráfaga. Para conseguir esto mezclamos los símbolos de distintos bloques para que un posible error de ráfaga se divida, al reconstruir las palabras originales, entre las t palabras. Una manera habitual de mezclarlos es la siguiente

$$\begin{aligned} \mathbb{F}_q^{tn} \supseteq (C)^t &\xrightarrow{\qquad f \qquad} I(C,t) \subset \mathbb{F}_q^{tn} \\ (c_1, c_2, \ldots, c_t) &\longmapsto (c_{11}, c_{21}, \ldots, c_{t1}, c_{12}, c_{22}, \ldots, c_{t2}, \ldots, c_{1n}, c_{2n}, \ldots c_{tn}) \end{aligned}$$

Donde c_{ij} es la componente j-ésima de la palabra c_i y $I(C,t) = Im(f)$, es el código intercalado con profundidad t de C. Esta transformación se puede ver también como colocar las palabras en una matriz (t,n), y posicionar sus columnas una detrás de otra

$$\begin{pmatrix} c_{11} & c_{12} & \ldots & c_{1n} \\ c_{21} & c_{22} & \ldots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{t1} & c_{t2} & \ldots & c_{tn} \end{pmatrix}$$

Si C puede corregir cualquier error de ráfaga de longitud b, entonces $I(C,t)$ puede corregir cualquier error de ráfaga de longitud tb ya que un error de este tipo resultará en ráfagas de a lo sumo longitud b en las palabras originales.

Ejemplo 49. Supongamos un código con $n = 5$ binario que puede corregir ráfagas de hasta 2 errores. El código intercalado de profundidad $t = 3$, puede corregir ráfagas de hasta 6 errores consecutivos. Por ejemplo si el error es

000001111110000

Al reorganizar la palabra el error tendrá el siguiente aspecto

001100011001100

Donde las ráfagas son corregibles de longitud 2.

6.7.2. Códigos recortados

Otro concepto utilizado para la codificación en CDs y DVDs es el de códigos recortados. Dado un código $C[n,k]$ un código recortado una cantidad m es un subcódigo de C que definimos eligiendo un conjunto de m coordenadas $0 < i_1 < i_2 < \ldots < i_m \leq n$ a las que se impondrá la condición de ser nulas. Es decir el código resultante es

$$\{(c_1, c_2, \ldots, c_n) \in C : c_{i_j} = 0 \quad \forall 1 \leq j \leq m\}$$

Que tiene obviamente dimensión $k - m$. Además se puede evitar transmitir los símbolos que se saben son nulos, reincorporándolos antes de la decodificación el receptor, de esta manera la longitud del código es $n - m$.

La manera más habitual de utilizar esto en la práctica es, dados $k - m$ símbolos de información, rellenar a la derecha con m ceros. Codificar esta palabra de longitud k mediante una codificación sistemática y enviar únicamente los $k - m$ símbolos de información junto con los $n - k$ símbolos de redundancia obtenidos.

Ejemplo 50. Consideremos un código $RS[7,4]$ sobre $\mathbb{F}_8$ y un elemento primitivo $\alpha^3 = \alpha + 1$. Veamos como codificar una palabra del código recortado $RS[5,2]$.

Tenemos los siguientes símbolos de información

$$(1, \alpha)$$

Añadimos ahora los ceros necesarios al final de la palabra

$$(1, \alpha, 0, 0)$$

Codificamos usando $RS[7,4]$ para obtener

$$(1, \alpha, 0, 0, \alpha^2, 0, \alpha^4)$$

Y finalmente enviamos

$$(1, \alpha, \alpha^2, 0, \alpha^4)$$

El receptor, que sabe que codigo estamos utilizando, sabría donde añadir los ceros para realizar la decodificación.

6.7.3. Codificación CD

Ya tenemos todas las herramientas necesarias para presentar el proceso de codificación en un CD. Los códigos utilizados para la corrección de errores son C_1 y C_2 código RS

recortados con parámetros $[28, 24, 5]$ y $[32, 28, 5]$ ambos sobre el cuerpo $\mathbb{F}_{256}$ (bytes). La información a codificar es una secuencia de tramas de la forma

$$L_1 R_1 L_2 R_2 \dots L_6 R_6$$

Donde L_i, R_i representan 2 bytes correspondientes a los canales izquierdo y derecho de una pista de audio. Cada trama tiene por lo tanto 24 bytes. Primero se mezclan los indices impares de una trama con los pares de 2 tramas más adelante. El resultado de esto se codifica mediante C_1 de manera sistemática pero introduciendo la redundancia en mitad de la palabra con el objetivo de separar lo más posible la información de las tramas distintas y que no sean afectadas por la misma ráfaga. Siendo $P_1 P_2$ esta redundandia introducida entonces

$$\begin{array}{l} T_i = L_1 R_1 L_2 R_2 \dots L_6 R_6 \\ \qquad\qquad \searrow \\ \qquad\qquad\quad L_1 L_3 L_5 R_1 R_3 R_5 \tilde{L}_2 \tilde{L}_4 \tilde{L}_6 \tilde{R}_2 \tilde{R}_4 \tilde{R}_6 \xrightarrow{C_1} L_1 L_3 L_5 R_1 R_3 R_5 P_1 P_2 \tilde{L}_2 \tilde{L}_4 \tilde{L}_6 \tilde{R}_2 \tilde{R}_4 \tilde{R}_6 \\ \qquad\qquad \nearrow \\ T_{i+2} = \tilde{L}_1 \tilde{R}_1 \tilde{L}_2 \tilde{R}_2 \dots \tilde{L}_6 \tilde{R}_6 \end{array}$$

Después de este proceso hemos transformado la secuencia de tramas en una secuencia de palabras de C_1: $c_1, c_2, c_3, \dots$

A continuación se hace un intercalado un poco distinto a la definición presentada, se colocan las palabras del código C_1 en una matriz de la siguiente manera

$$\begin{array}{cccccccccccccc} c_{1,1} & c_{2,1} & c_{3,1} & c_{4,1} & c_{5,1} & c_{6,1} & c_{7,1} & c_{8,1} & c_{9,1} & c_{10,1} & c_{11,1} & c_{12,1} & c_{13,1} & \dots \\ 0 & 0 & 0 & 0 & c_{1,2} & c_{2,2} & c_{3,2} & c_{4,2} & c_{5,2} & c_{6,2} & c_{7,2} & c_{8,2} & c_{9,2} & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{1,3} & c_{2,3} & c_{3,3} & c_{4,3} & c_{5,3} & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{1,4} & \dots \\ & & & & & & \vdots & & & & & & & \end{array}$$

De manera que una palabra original se encuentra siguiendo una pendiente de $-1/4$ en la matriz. Tenemos una matriz de 28 filas con tantas columnas como fuesen necesarias para acomodar todas las palabras de esta manera, rellenando con ceros.

Finalmente cada una de estas columnas se codifica entonces mediante C_2, completando el proceso de codificación. El proceso de decodificación no presenta ningún problema dado un método para decodificar códigos RS ya que lo único necesario es deshacer el intercalado, lo cual es sencillo.

Esta construcción utilizada por los CDs se llama CIRC (*Cross interleaved Reed-Solomon Code*), que trata los datos secuencialmente. A continuación veremos como, trantando los datos como una matriz, se utiliza el producto de códigos de Reed-Solomon para la codificación en un DVD.

6.7.4. Codificación en DVD

La tecnología del DVD fue introducida en 1993 por Toshiba, compañía que fue clave en el desarrollo de un formato unificado y de muchas de las tecnologías clave para el funcionamiento del disco. En comparación con la capacidad correctora del CD, que podía llegar a corregir errores de hasta 500 bits sucesivos, un DVD puede corregir errores suscesivos de alrededor de 2800 bits. Un salto importante en este aspecto.

Como ya hemos mencionado al contrario que CIRC, se tratan en este caso los datos como bloques, matrices. Estos datos están organizados en sectores de 2064 bytes que contienen una cabecera de 12 bytes, 2048 bytes de datos útiles y 4 bytes de corrección de errores. De nuevo se trata la información por bytes de manera que los código utilizados están construidos sobre el cuerpo $\mathbb{F}_{256}$.

Tanto para la codificación como la decodificación se combinan 16 sectores en un bloque. Estos bloques son el nivel al que trabaja el código corrector. La correspondencia entre bloques y sectores se detalla en la figura 6.3. El código es un producto de códigos Reed-Solomon, que en el contexto del DVD se suele abreviar RSPC (Reed-Solomon Product Code). En concreto es el producto de dos códigos RS recortados $[182, 172, 10] \times [208, 192, 17]$.

Las palabras se almacenan en el disco por filas, luego los errores en ráfagas están siempre en la direeción de las filas y el código de las columnas puede corregirlos facilmente.

En cuanto a la decodificación se realiza primero por filas y después por columnas utilizando el algoritmo de Berlekammp-Massey con la particularidad de que también se marcan en el primer paso los errores de corrección como borrones.

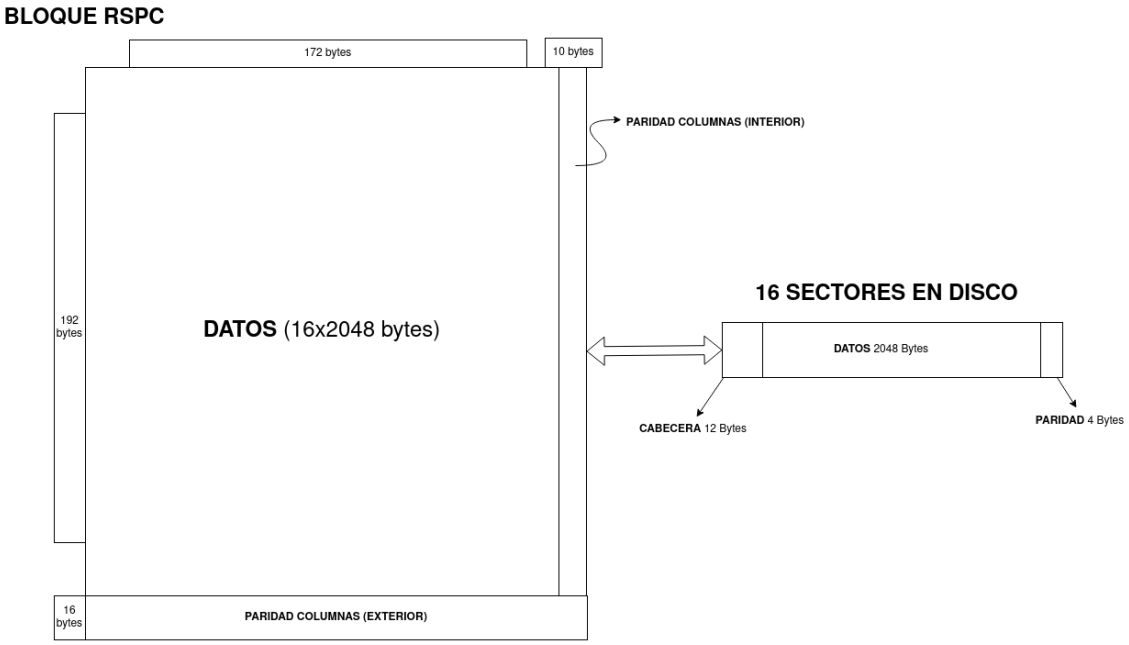


Figura 6.3: Correspondencia entre bloques y sectores en un DVD

7. IMPLEMENTACIONES Y CONCLUSIÓN

Este trabajo se ha realizado en conjunto con el trabajo final del grado de ingeniería informática [24], como parte de este último se han implementado la mayoría de los algoritmos presentados teóricamente aquí. Estas implementaciones han sido utilizadas en muchos casos para realizar los cálculos de los ejemplos que se han dado a lo largo de todo el texto.

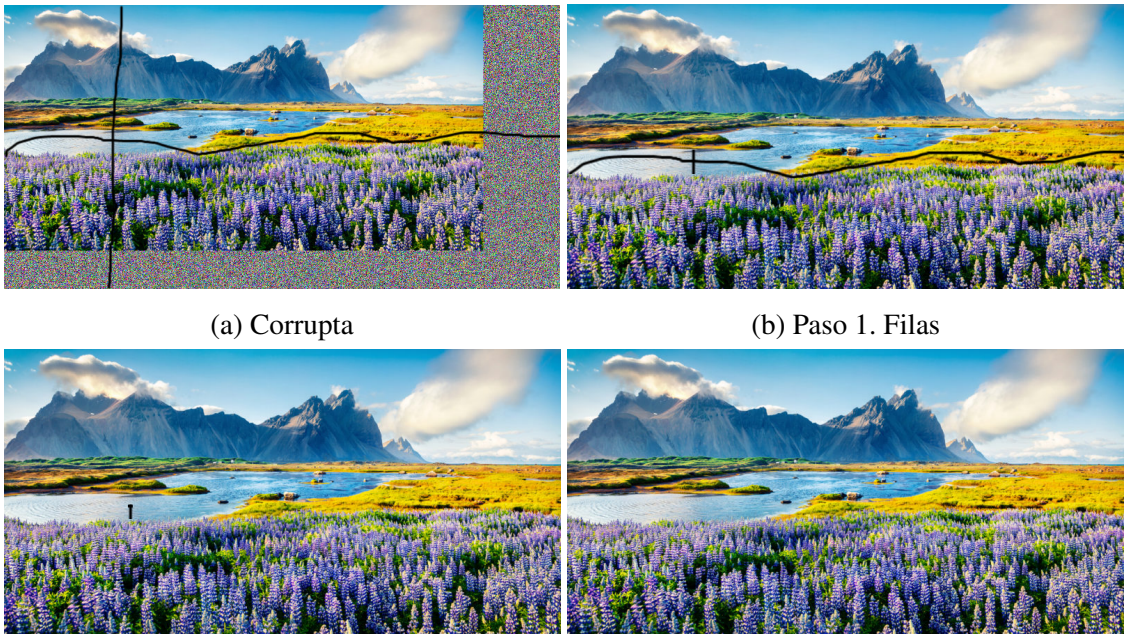
La principal herramienta utilizada ha sido la libreria `galois` [3], de `python`. Esta libreria abstrae los cálculos en cuerpos finitos de una manera computacionalmente eficiente y compatible con la librería `numpy` de uso muy extendido. Además implementa códigos Reed Solomon con su codificación y decodificación utilizando el algoritmo de Berlekamp-Massey, sin embargo también han sido implementados a parte para poder dar los resultados intermedios en los ejemplos.

La idea inicial del trabajo surge de intentar replicar una página web que presentaba visualmente el uso de códigos producto aplicados en DVD, debida Høholdt, que ya no está disponible [5]. En su web, permitía la corrupción de una imagen, bien dibujando sobre ella con el ratón o bien introduciendo ruido aleatorio y demostraba la corrección primero por filas y luego por columnas. En la siguiente imagen se muestra mi versión de esto.

Figura 7.1: Interfaz que permite corromper una imagen codificada.

El programa divide la imagen en bloques del tamaño adecuado para un código elegido, y añade la información de redundancia en el borde de la imagen (codificación sistemática).

Una linea dibujada en la imagen replica un rayón de un disco CD o DVD y podemos ver cómo el código producto maneja bien este tipo de errores



(a) Corrupta (b) Paso 1. Filas

(c) Paso 2. Columnas (d) Paso 3. Filas

Figura 7.2: Proceso de corrección de rayones. La corrección por filas es efectiva para rayones más verticales, mientras que la corección por columnas lo es para rayones horizontales.

La limitación de la implementación es que usa la longitud de código 255, ya que es muy útil trabajar con el cuerpo $\mathbb{F}_{256}$ donde cada elemento puede representar un byte de información. Por otra parte la dimensión del código si se ha dejado variable. Aprovechamos este capitulo de implementación para dar unos resultados empíricos sobre la capacidad correctora del algoritmo en función de la dimensión. Para varios valores de los parámetros k y ρ estimamos mediante 40 observaciones la media de iteraciones realizadas y el porcentaje de correcciones completas. Se incluyen los reultados que obtuvieron un tasa de corrección mayor que 0,9 pero menor que 1, para estimar de esta manera el valor limite de la densidad de errores corregible. Estos datos se proporcionan en la tabla 7.1.

k	ρ	Tasa de corrección	Iteraciones
140	0.275	0.975	7.725
150	0.255	0.95	8.925
155	0.246	0.95	8.575
160	0.233	0.975	8.925
165	0.225	0.9	9.075
175	0.202	0.9	8.525
180	0.189	0.9	9.55
185	0.18	0.9	9.55
190	0.165	0.975	9.325
195	0.155	0.975	8.6
200	0.143	0.9	11.225
205	0.133	0.975	10.825
210	0.119	0.95	11.425
215	0.109	0.975	10.725
220	0.095	0.95	12.25

Cuadro 7.1: Resultados de ejecutar el algoritmo iterativo con un error aleatorio introducido de densidad ρ, utilizando un código producto $RS[255, k] \times RS[255, k]$. Media de iteraciones y tasa de corrección sobre 40 ejecuaciones.

En el gráfico 7.3 se compara la densidad que hemos obtenido de estas mediciones (la capacidad correctora limite para el algoritmo iterativo), con la capacidad correctora dada por la distancia mínima.

$$\frac{t}{n^2} = \frac{d-1}{2n^2} = \frac{(255-k+1)^2-1}{2(255^2)}$$

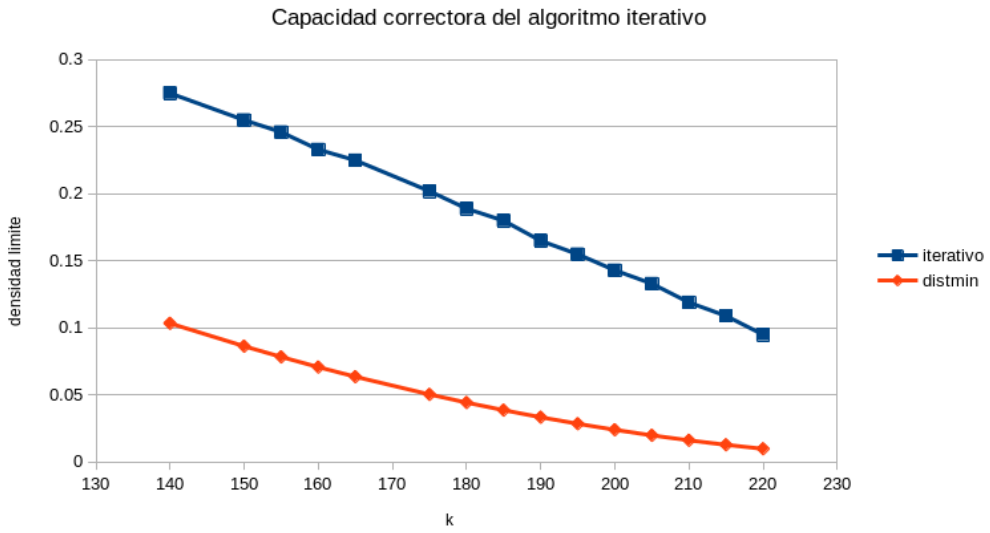


Figura 7.3: Gráfico comparativo de la capacidad correctora posible mediante el algoritmo iterativo con respecto de la dada por la distancia minima, para distintas elecciones de dimensión del código constituyente.

El resto de algoritmos programados (Berlekamp-Massey, Sudan, combinar decodificación con lista) no se han aplicado a la correción de imágenes por bloques. Se aplican directamente sobre palabras de los códigos correspondientes y de manera interactiva gracias a cuadernos `jupyter` .

Para otros detalles de la implementación de estos algoritmos nos referimos a la memoria correspondiente al trabajo de informática.

BIBLIOGRAFÍA

[1] Wikipedia contributors. "QR code." (2023), dirección: https://en.wikipedia.org/wiki/QR_code (visitado 04-05-2023).

[2] Wikipedia contributors. "Reed–Solomon error correction." (2023), dirección: https://en.wikipedia.org/wiki/Reed-Solomon_error_correction (visitado 04-05-2023).

[3] M. Hostetter, *Galois: A performant NumPy extension for Galois fields*, nov. de 2020. dirección: https://github.com/mhostetter/galois.

[4] J. Justesen y T. Høholdt, *A Course In Error-Correcting Codes*. European Mathematical Society, 2017.

[5] T. Høholdt. "RS en-/decoding in DVD players." (), dirección: http://www2.mat.dtu.dk/people/T.Hoeholdt/DVD/index.html (visitado 22-07-2023).

[6] O. Al-Askary, "Iterative decoding of product codes," Tesis doct., KunglTekniska Högskolan (KTH) Royal Institute of technology, 2003.

[7] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, n.º 2, págs. 147-160, 1950.

[8] I. S. Reed y G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, n.º 2, págs. 300-304, 1960.

[9] S. A. Aly, A. Klappenecker y P. K. Sarvepalli, "On quantum and classical BCH codes," *IEEE Transactions on Information Theory*, vol. 53, n.º 3, págs. 1183-1188, 2007.

[10] E. R. Berlekamp, *Algebraic coding theory (revised edition)*. World Scientific, 2015.

[11] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE transactions on Information Theory*, vol. 15, n.º 1, págs. 122-127, 1969.

[12] R. E. Blahut, *Algebraic codes for data transmission*. Cambridge university press, 2003.

[13] Wikipedia contributors. "Linear-feedback shift register." (2023), dirección: https://en.wikipedia.org/wiki/Linear-feedback_shift_register (visitado 04-06-2023).

[14] M. Sudan, "List decoding: Algorithms and applications," *ACM SIGACT News*, vol. 31, n.º 1, págs. 16-27, 2000.

[15] P. Elias, "List decoding for noisy channels," *Tech report*, 1957.

[16] J. M. Wozencraft, "List decoding," *Quarterly Progress Report*, vol. 48, págs. 90-95, 1958.

[17] R. R. Nielsen y T. Høholdt, "Decoding Reed-Solomon codes beyond half the minimum distance," en *Coding Theory, Cryptography and Related Areas: Proceedings of an International Conference on Coding Theory, Cryptography and Related Areas, held in Guanajuato, Mexico, in April 1998*, Springer, 2000, págs. 221-236.

[18] M. Sudan, "Decoding of Reed Solomon codes beyond the error-correction bound," *Journal of complexity*, vol. 13, n.º 1, págs. 180-193, 1997.

[19] V. Guruswami y M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," en *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, IEEE, 1998, págs. 28-37.

[20] P. Elias, "Error-free coding," *Tech report*, 1954.

[21] Á. I. Barbero Díez, "Códigos Producto Correctores de Errores," Tesis doct., Universidad de Valladolid, Valladolid, España, 1994.

[22] W. C. Huffman y V. Pless, *Fundamentals of error-correcting codes*. Cambridge university press, 2010.

[23] J. P. Nguyen, "Applications of Reed-Solomon codes on optical media storage," Tesis doct., San Diego State University, 2011.

[24] P. Ibarlucea, "Corrección de errores en imágenes mediante producto de códigos de Reed-Solomon, trabajo final del grado de informática," Universidad de Valladolid, 2023.