Grado en Matemática Computacional

Proyecto Final de Grado

Compresión de datos usando códigos lineales

Autor:
Pablo Jiménez Mateo

Supervisor:
Francesc Josep Alted Abad
Tutor académico:
Fernando Hernando Carrillo

Fecha de lectura: __ de _____ de 20__
Curso académico 2013/2014

Resumen

A lo largo de este trabajo damos una introducción a la teoría algebráica de códigos. Además explicamos cómo usar los códigos lineales para la compresión de datos, en particular usando los códigos de Hamming.

Palabras clave

Códigos, Hamming, Golay, compresión.

Keywords

Codes, Hamming, Golay, compression.

Índice

1. Introducción

Este trabajo está enmarcado dentro de la teoría de la información. No se puede hablar de la teoría de la información sin hablar de C.E. Shannon, un trabajador de Bell Laboratories y al que se le considera como el padre de la teoría de la información a raíz de su revolucionario trabajo titulado «A mathematical theory of communication» que podemos encontrar en [10].

Esta se desarrolló a partir de la segunda mitad del siglo XX motivada por el auge de las comunicaciones. Los códigos correctores de errores se usan siempre que se quiere enviar un mensaje por un canal que contenga ruido, de manera que el mensaje llegue correcto al receptor. El ruido es un fenómeno por el cual el mensaje enviado cambia su significado debido a los distintos errores que pueden haberse producido en el canal. Para mitigar este problema, C.E. Shannon, inventó la teoría de los códigos correctores de errores en 1948. La idea de Shannon es introducir información extra al mensaje de manera que se pueda recuperar la información enviada. Los primeros códigos llamados *códigos de repetición* simplemente repetían cada bit varias veces en función del canal por el que se quisiera transmitir, de manera que la posibilidad de fallo es mucho menor.

En 1950 R.W. Hamming, también trabajador de Bell Laboratories, empezó a trabajar en códigos correctores con una tasa de transmisión mayor que los propuestos por C.E. Shannon. El resultado de su trabajo fueron los *códigos de Hamming* una familia de códigos que permite corregir fácilmente un error en la transmisión.

Entre 1969 y 1973 con el comienzo de la exploración espacial, la teoría de códigos se volvió muy importante. Los NASA Mariners usaban códigos de Reed-Muller capaces de corregir 7 errores en transmisiones de 32 bits. Estos códigos introducían 26 bits de control por cada 6 bits de información. Los códigos correctores se usan mucho en tecnologías actuales, los DVD, discos duros e incluso la comunicación entre dispositivos móviles se benefician de estos conceptos.

Este trabajo está dividido en dos partes.

En la primera parte se explica la estancia en prácticas donde desarrollamos una serie de tutoriales poniendo a prueba los productos de la empresa Continuum Ibérica. Los tutoriales están realizados en el lenguaje de programación Python y se presentan usando ipython-notebooks, una herramienta que permite mostrar la ejecución y los resultados de manera muy visual. Todas las herramientas que se utilizan están destinadas a trabajar con grandes cantidades de datos de una manera cómoda para la memoria. Los tutoriales cubren desde el uso de las herramientas hasta su beneficio en un entorno real, con datos objetivos. Todos los proyectos han sido realizados en inglés y están enlazados en este trabajo.

En la segunda parte explicamos la teoría algebráica de la teoría de códigos. La teoría de códigos ha evolucionado mucho desde entonces, su aplicación más común es la de detectar y corregir errores, pero también se usan para cifrar mensajes y para comprimir información. El objetivo marcado por este trabajo es justamente la compresión de información mediante el uso de códigos lineales, tema que se trata con mayor profundidad en los últimos dos temas.

El trabajo está estructurado en 11 temas, siendo este el primero. En el tema 2 se explica el trabajo que realicé en la empresa Continuum Ibérica, desglosado por los distintos tutoriales que se me pidieron. Los siguientes temas son de la parte teórica de la memoria, en los temas 3, 4 y 5 se describen conceptos básicos necesarios para entender los códigos. En el tema 6 se describen los temas relacionados con los códigos lineales, que son los que se estudian en mayor profundidad en los temas posteriores. En los temas 7 y 8 se describen los códigos de Hamming y los de Golay respectivamente, y en el tema 9 se examinan las distintas cotas para los parámetros de un código lineal. Finalmente los dos últimos temas se centran en el proceso de compresión de datos mediante el uso de códigos lineales, introduciendo la teoría y un ejemplo práctico.

2. Estancia en prácticas

Las prácticas las realicé en la empresa Continuum Ibérica localizada en el CEEI de Castellón de la Plana, Castellón. La empresa está centrada en trabajar con grandes cantidades de datos (o Big Data) de manera cómoda y eficiente. Durante mi estancia se me pidió que probara los distintos productos y realizara una serie de tutoriales demostrando su utilidad.

En esta sección voy a intentar resumir los distintos tutoriales, los podéis encontrar todos con sus resúmenes pinchando aquí.

2.1. Speeding code with Numba

En el primer tutorial se me pidió que probara la herramienta Numba. Esta herramienta desarrollada por Continuum permite compilar funciones de python directamente en C de manera transparente, ganando así mucha velocidad.

Para probar la ganancia de velocidad reduzco una imagen y mido los tiempos reduciéndola sin y con numba. Al final del notabook se ve que utilizar numba es 72 veces mejor que python puro.

El notebook completo lo podéis encontrar pinchando aquí.

2.2. Working with huge images in python, the painless way

En este tutorial sigo trabajando con imágenes pero introduzco una librería diferente, blz, creada también por Continuum. Esta librería permite trabajar de manera transparente con grandes cantidades de datos. Quizá no parezca demasiado impresionante pero voy a intentar explicar las bondades de este formato.

Imaginemos un formato de datos normal (en este caso vamos a quedarnos con los formatos de imagen) como puede ser PNG. PNG es un formato muy bueno de imágenes que comprime la información y da muy buenos resultados pero, ¿qué pasa si la imagen descomprimida es muy grande, mayor que tu memoria RAM? Bien, en ese caso tenemos un problema ya que no podemos abrirla ni trabajar con ella a no ser que utilicemos un software de imágenes especial.

Ahí es donde reside la fortaleza del contenedor barray de la librería blz, un contenedor que actúa como un vector nativo de Python. Para los que estén familiarizados con el lenguaje, es sencillo pensar en este contenedor como una extensión de los vectores de Python que añade distintos compresores con sus niveles de compresión correspondientes. Esto da al usuario control total sobre el contenedor, permitiéndole máxima eficacia. Pero la característica más importante de barray es, sin duda, el acceso por bloques.

El contenedor barray permite acceder y traer a memoria bloques específicos, lo que hace que no tengas que cargar todos los datos de golpe en memoria y puedas trabajar con ellos de manera cómoda. Es justo esto lo que pruebo en este tutorial, cogiendo pedazos de una imagen enorme, partiéndola, haciéndola ese pedazo más pequeño y volviéndolo a almacenar.

El notebook completo lo podéis encontrar pinchando aquí.

2.3. Generating huge Mandelbrot fractals with Python, the quick and memory-safe way

Como no podía ser de otra manera en un trabajo matemático, en este tutorial se trabaja con fractales, en concreto el fractal de Mandelbrot.

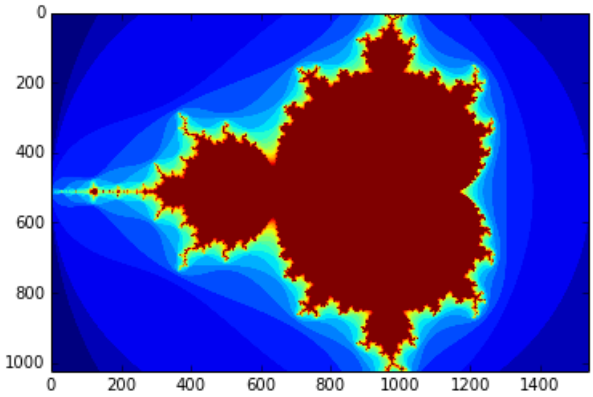

Figura 1: Fractal de Mandelbrot.

Para demostrar que cualquier función python puede trabajar de manera transparente con el contenedor barray de la misma manera con la que trabajaría con una matriz, en este tutorial generamos fractales de distintos tamaños directamente en el contenedor barray.

El tutorial concluye con la generación de un fractal enorme 30000x20000 que ocupa 572 MiB y lo comprime hasta alcanzar un tamaño de 4.2 MiB sin gastar más de 100 MiB de RAM.

El notebook completo lo podéis encontrar pinchando aquí.

2.4. Comparing speedups when using compression with blz barrays

Este tutorial compara los distintos compresores y niveles de compresión usados por el contenedor barray. Para eso se genera un fractal de Mandelbrot de 30000x20000 y se va comprimiendo con cada uno de los compresores y niveles de compresión. Los datos se guardan en CSV para su análisis posterior.

Las gráficas más importantes son las siguientes que demuestran que en algunos casos es más rápido comprimir para traer la información a memoria principal de manera más rápida, descomprimir en memoria principal y realizar las operaciones y por último comprimir el resultado de nuevo para escribir menos en el disco. Esto se debe a que el cuello de botella es el disco duro, es muy lento escribir en él, así que es mejor tomarse un tiempo adicional comprimiendo los datos con tal de escribir menos. La línea roja es el tiempo medio que cuesta en escribir el archivo descomprimido en disco así que todo lo que esté por debajo de ella es más rápido usando compresión.

El notebook completo lo podéis encontrar pinchando aquí.

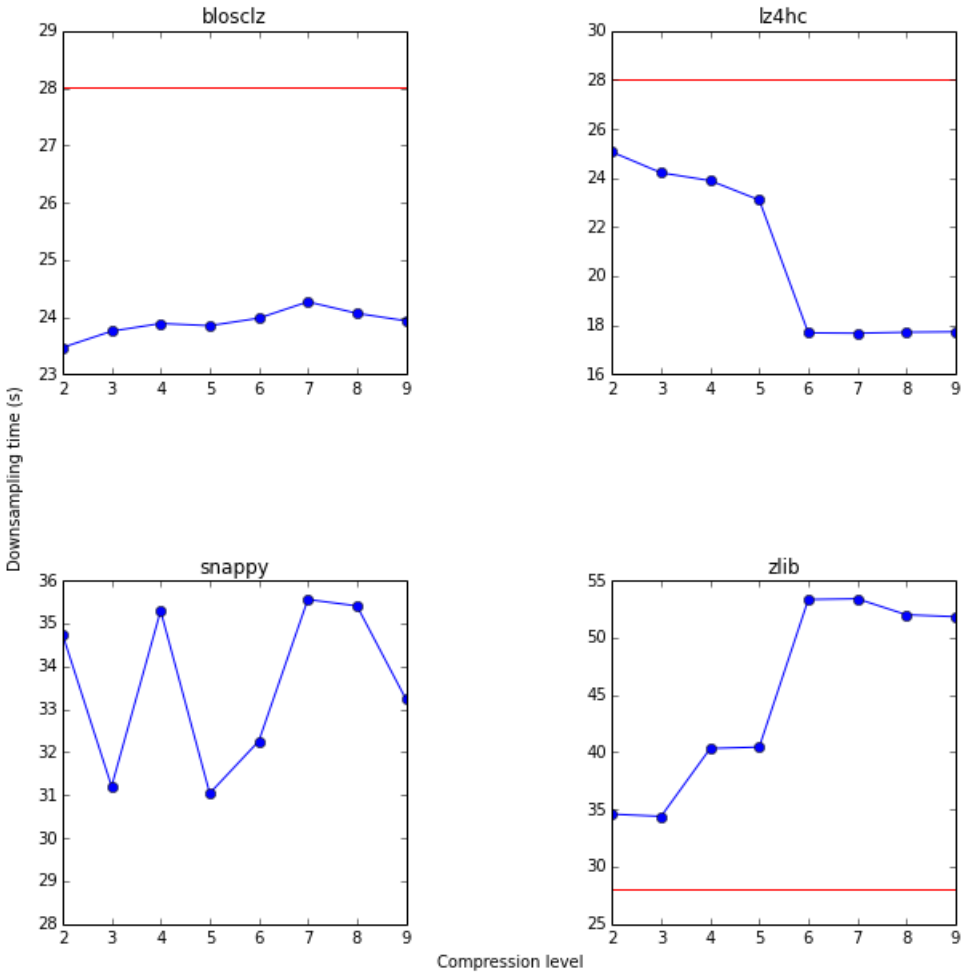

Figura 2: Gráficas de tiempo de computación respecto al nivel de compresión.

2.5. Querying big datasets, the good way

En este tutorial usamos un contenedor diferente de la librería blz. Esta vez se usa un btable, que almacena los datos por columnas y cada columna puede ser de un tipo distinto. Además este es el primer tutorial que usa datos del mundo real, los datos de los movimientos en bolsa de Microsoft.

Aparte de mostrar la versatilidad del formato, el tutorial concluye haciendo un gráfico financiero llamado Open-High-Low-Close.

El notebook lo podéis encontrar pinchando aquí.

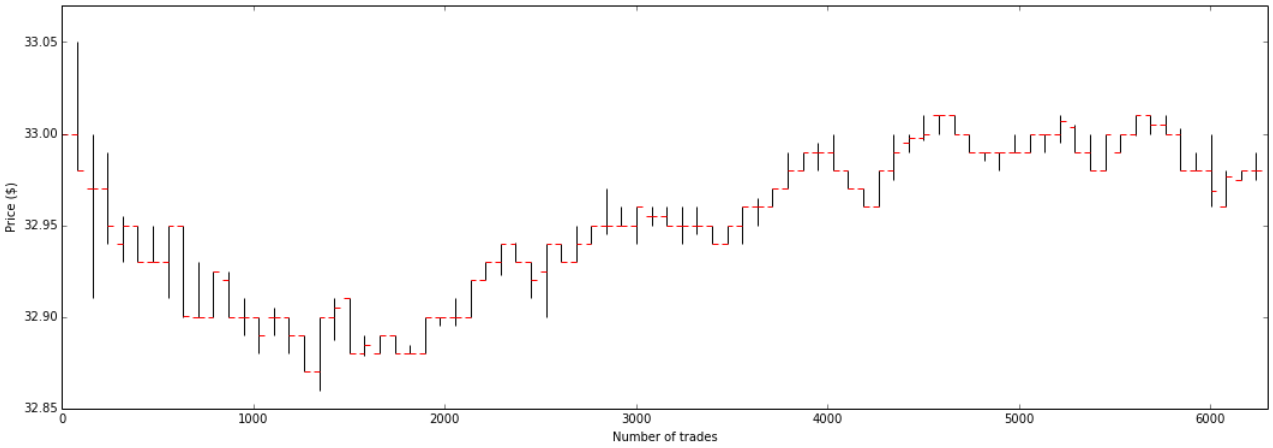

Figura 3: Gráfica Open-High-Low-Close.

2.6. Comparing speedups when using compression with blz btables in disk only

Este tutorial compara los distintos compresores y niveles de compresión para el contenedor btable pero sólo en disco. Esto se consigue borrando la caché cada vez que se graba el archivo de tal manera que todo se hace en disco. Aquí la línea roja vuelve a ser la media de tiempos sin compresión, por lo que cualquier cosa por debajo de ella es más rápido con compresión.

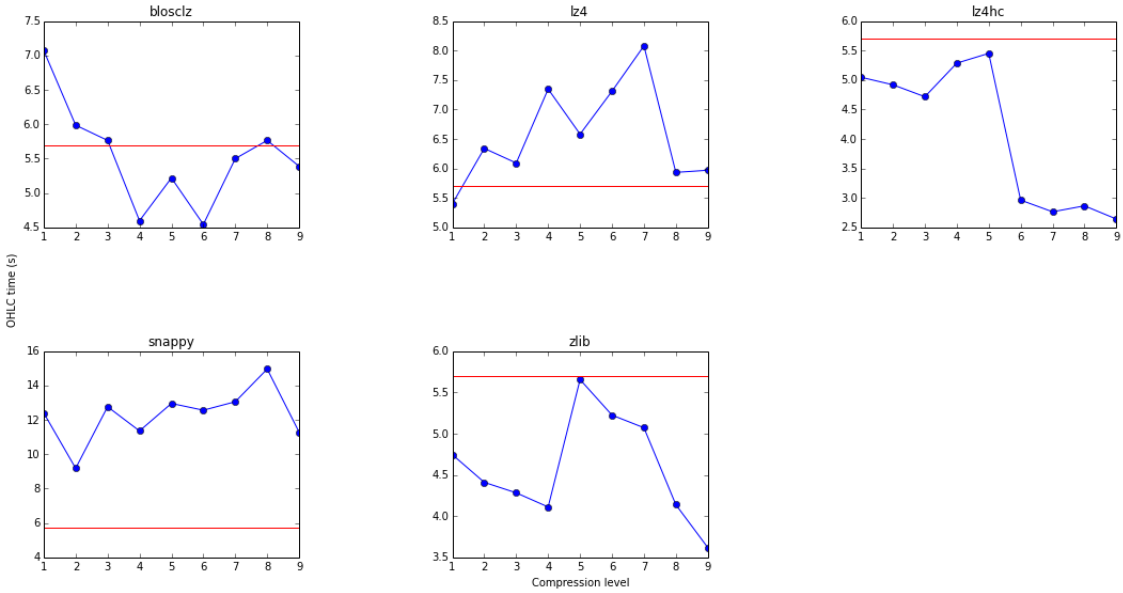

Figura 4: Gráfica mostrando el tiempo de ejecucción respecto al nivel de compresión.

El notebook lo podéis encontrar pinchando aquí.

2.7. Comparing performance of btables in memory and disk

Este tutorial hace lo mismo que el anterior pero además incluye los tiempos de trabajar con memoria caché. Las gráficas son por lo tanto más elaboradas.

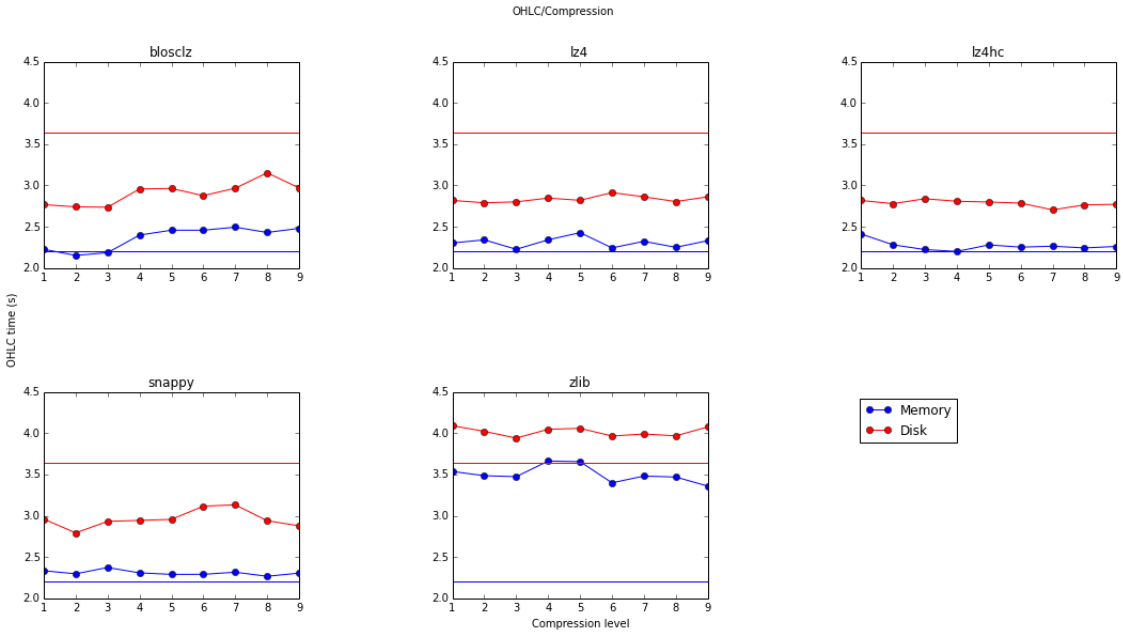

Figura 5: Gráfica mostrando el tiempo de ejecucción en disco y memoria respecto al nivel de compresión.

El notebook lo podéis encontrar pinchando aquí.

3. Conceptos básicos sobre códigos

3.1. ¿Qué es un código?

Un código es una manera de transformar la información. La información se ha transformado siguiendo un conjunto de reglas previamente establecido.

A menudo la información se codifica para que sea más cómoda de manejar a la hora de enviarla. La información original puede ser susceptible a interferencias u ocupar demasiado espacio.

Los objetivos que persigue la codificación son los de *detectar y corregir* los posibles errores que se hayan generado durante la transmisión y garantizar su *privacidad* entre otros. Además, los códigos, también pueden ser utilizados para comprimir la información o para garantizar la privacidad entre otros.

3.2. Codificación

Para realizar el proceso de codificación es necesario definir dos conceptos: *alfabeto* e *información*. Un *alfabeto* es cualquier conjunto finito (piensa en el alfabeto tradicional de la a a la z) e *información* es cualquier sucesión de elementos de ese alfabeto (una frase o palabra tan solo toma elementos del alfabeto).

El alfabeto usado para escribir la información originalmente no tiene por qué coincidir con el alfabeto codificado.

Imaginemos la codificación más sencilla, el código de César. Este código se basa en desplazar cada letra un número de espacios, por ejemplo 3. Así la letra A se convertiría en la D, la B en la E y así con todos. En este ejemplo ambos alfabetos coinciden, pero la información está codificada para darle cierta privacidad.

Como codificación que no usa los mismos alfabetos podemos pensar en la codificación binaria de un número, la transformación sería la siguiente:

Número en decimal	Número en binario
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

En el caso anterior el *alfabeto fuente* $\mathcal{A}$ son los números del 0 al 9 y el *alfabeto de código* $\mathcal{B}$ son simplemente el 0 y el 1.

En este trabajo nos vamos a centrar en la codificación para corregir errores. Para ilustrar una codificación

para corregir errores muy simple vamos a imaginar que el emisor nos envía un 0 ó un 1 y se tiene una probabilidad $p < 0{,}5$ de que haya un error, es decir de que se envíe un 0 y se reciba un 1 ó viceversa.

Si decidiéramos no hacer nada, tendríamos una probabilidad p de que el mensaje fuera erróneo. Con un poco de imaginación podemos pedirle al emisor que envíe cada bit 3 veces, y decidamos su *descodificación* en función del mayor número de bits. Asumiendo que en este canal los errores ocurren de manera independiente, la probabilidad de que haya un error mínimo de dos bits será

$$P = \left(\begin{array}{c} 3 \\ 2 \end{array} \right) p^2(1-p) + \left(\begin{array}{c} 3 \\ 3 \end{array} \right) p^3 = 3p^2 - 2p^3$$

Es fácil ver que esta probabilidad de error es menor que p, así que *codificando* el mensaje original hemos reducido la probabilidad de error.

4. Corrección de errores

4.1. ¿A qué se deben los errores?

A la hora de transmitir información por cualquier medio puede darse el caso de que en el canal exista *ruido* que pueda afectar a cómo la información es recibida por el receptor.

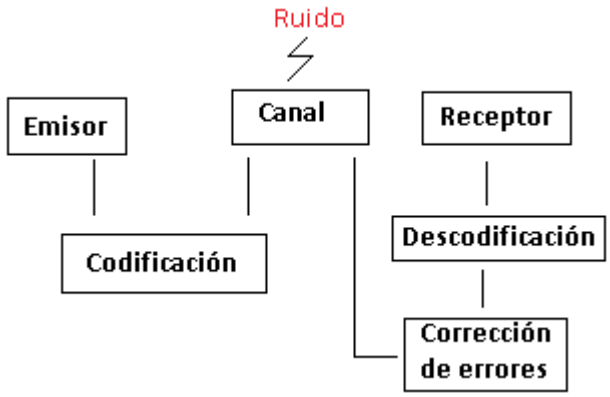

Figura 6: Esquema de transmisión de mensaje.

La cantidad de ruido depende enormemente del tipo de canal utilizado, no es el mismo tipo de error el introducido al leer un Compact Disk que se haya rallado o al recibir un mensaje desde un satélite en órbita.

Para poder mitigar esta pérdida de información se añade redundancia en el mensaje en función de qué se quiere conseguir.

Para esto se usan los *códigos de bloque*[1] y una distancia para poder saber cuán diferente es una palabra de otra.

4.2. Distancia de Hamming

Para que la corrección de errores sea posible dos palabras han de ser lo más diferentes posible, para que sea lo más improbable posible que el ruido del canal cambie una palabra válida a otra palabra válida, haciendo imposible su detección.

Para poder saber cómo de diferente es una palabra de otra definimos la distancia de Hamming:

Sea β^n un alfabeto finito. Si $x,\ y \in \beta^n,\ x = (x_1, ..., x_n),\ y = (y_1, ..., y_n)$ llamaremos distancia de Hamming entre x e y a

$$d(x, y) = \#\{i \ / \ 1 \leq i \leq n,\ x_i \neq y_i\}$$

En lenguaje común, la distancia se define como el número de símbolos que difieren en las dos palabras en la misma posición.

[1] Recordemos que se caracterizan porque todas sus palabras tienen la misma longitud.

Sea β^n el alfabeto, la aplicación d es una distancia en β^n es decir, verifica las propiedades:

(i) d es no negativa y $d(x, y) = 0$ si y sólo si $x = y$

(ii) $d(x, y) = d(y, x)$

(iii) $d(x, y) + d(y, z) \geq d(x, z)$

para todo $x, y, z \in \beta^n$

Definición: Llamaremos distancia *mínima* del código C a

$$d = d(C) = min\{d(x, y) \mid x, y \in C,\ x \neq y\}$$

4.3. Descodificación

Sea C un código. Para descodificar una palabra $c \in C$ recibida se calculará la distancia de esa palabra al resto de palabras de C y la descodificaremos por la más cercana (en caso de que sea única)

La descodificación fallará cuando la palabra más cercana no sea única.

Algoritmo: Para una palabra p el proceso de descodificación será el siguiente:

1. Evaluar la distancia de p a todas las palabras de C

2. Sustituir p por la más cercana (si es única)

Si la palabra más cercana es única, el algoritmo proporcionará una descodificación válida, sea correcta o no. Lo siguiente que habría que mirar es ¿cuándo la descodificación es posible? y ¿cuándo es correcta?

4.4. ¿Cuántos errores puedo corregir?

A la hora de corregir o detectar errores se verifica lo siguiente:

Proposición: Sea C un código de bloque de longitud n y distancia mínima d. El algoritmo anterior aplicado a la n-tupla recibida,

a) Permite detectar t errores siempre que $t < d$

b) Permite corregir t errores siempre que $2t < d$

c) Permite corregir s borrones siempre que $s < d$

d) Permite corregir t errores y s borrones siempre que $2t + s < d$

Un esquema de los apartados a) y b) siendo y la palabra recibida y d distancia mínima, sería:

a) Si ocurren $t < d$ errores y $t > 0$ entonces la palabra recibida no está en C. Basta evaluar si $y \in C$ o no.

b) Si ocurren $t < \frac{d}{2}$ y $t > 0$, como las bolas son disjuntas, y caerá en una única bola y se corregirá por la palabra del centro de la bola.

Para verlo de manera más clara vamos a ver una imagen:

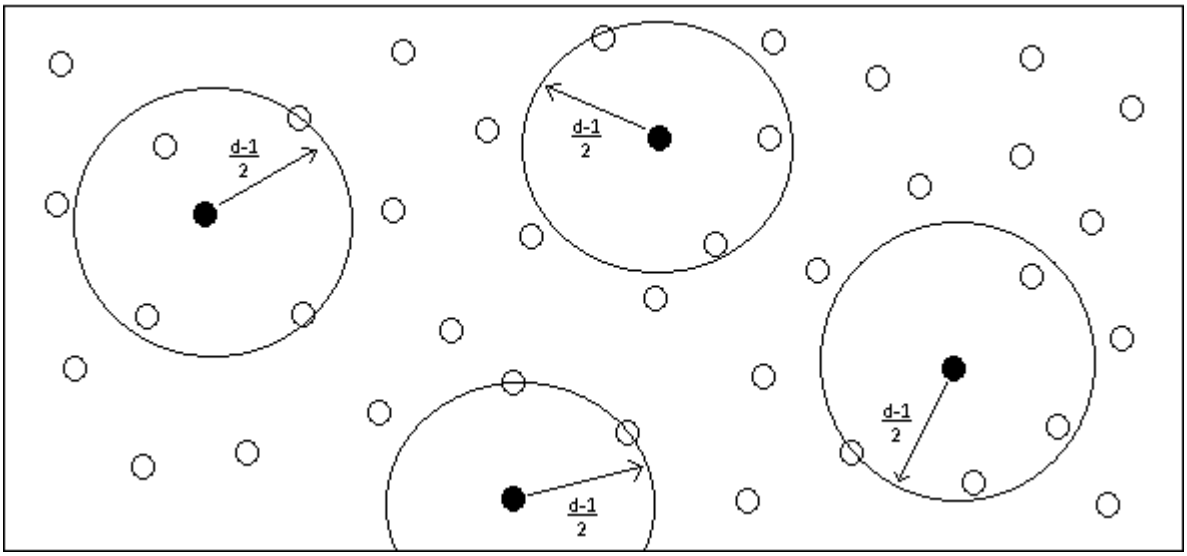

Figura 7: Esquema de bolas.

Las bolas negras representan palabras del código y las bolas blancas representan el resto de palabras de longitud n. Además por definición de distancia mínima, las bolas con centro en las palabras del código y radio $(d-1)/2$ son disjuntas.

5. Cuerpos finitos

5.1. ¿Qué es un cuerpo finito?

En álgebra abstracta, un cuerpo finito es un cuerpo definido sobre un conjunto finito de elementos. Todos los cuerpos finitos tienen un número de elementos $q = p^n$, para algún número primo p y algún entero positivo n. Todos los cuerpos finitos del mismo orden son isomorfos entre si. Para más información sobre cuerpos finitos ver [6].

Para un cuerpo finito K, al tener estructura de cuerpo cumple las siguientes propiedades:

1) K es cerrado para la adición y la multiplicación:

$\forall\ a, b \in K,\ a + b$ y $a \cdot b \in K$

2) Propiedad asociativa:

$\forall\ a, b, c \in K,\ a + (b + c) = (a + b) + c$ y $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

3) Propiedad conmutativa:

$\forall\ a, b \in K, a + b = b + a$ y $a \cdot b = b \cdot a$

4) Existencia del elemento neutro:

$\exists\ 0 \in K$, tal que $\forall a \in K, a + 0 = a$

$\exists\ 1 \in K, \neq 0$, tal que $\forall a \in K, a \cdot 1 = a$

5) Existencia de elemento opuesto y de inverso:

$\forall\ a \in K, \exists\ -a \in K$, tal que $a + (-a) = 0$

$\forall\ a \neq 0 \in K, \exists\ (a - 1) \in K$, tal que $a \cdot (a - 1) = 1$

6) Propiedad distributiva:

$\forall\ a, b, c, \in K, a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

5.2. Existencia y unicidad de los cuerpos finitos

Proposición: Para todo número natural primo p, existe un cuerpo finito con p elementos.

Demostración: Si p es un número primo, el anillo cociente Z/pZ es un cuerpo con p elementos.

El conjunto {0,1,...,p-1} es un sistema completo de representantes de Z/pZ. Esto autoriza a identificar este conjunto con Z/pZ, correspondiendo las operaciones $+, \cdot$, del cuerpo a la adición y multiplicación usuales de los elementos representantes de módulo p. Habitualmente manejaremos de esta forma los elementos de

Z/pZ. Cabe advertir, sin embargo, que la anterior representación, a veces denominada *positiva*, no es la única posible. Es también habitual la representación *simétrica*, consistente (para p primo impar) en el conjunto de representantes {-(p-1)/2,-(p-3)/2,...,(p-3)/2,(p-1)/2}

Ahora que sabemos que los cuerpos finitos existen para un cardinal primo, podemos plantearnos cualquier cardinal.

Proposición: Si q es el cardinal de un cuerpo finito, existen un primo p y un número natural r tales que $q = p^r$

Demostración: Sea K un cuerpo de cardinal q y denotemos por 1_k su elemento unidad. Consideremos la aplicación lineal

$$\varphi : Z \longrightarrow K; \varphi(n) = n \cdot 1_k$$

donde, si n es un número natural, $n \cdot 1_k = 1_k + ... + 1_k$ y si n es negativo, $n \cdot 1_k$ es el opuesto de la anterior suma. El núcleo de φ es no nulo (puesto que Z es infinito) y fácilmente de comprueba que consiste en un ideal de la forma pZ, siendo p un número primo. Por tanto K debe contener un subcuerpo de cardinal p, $k = Im(\varphi) \simeq Z/pZ$. Entonces K tiene estructura de espacio vectorial sobre k con dimensión finita. Si r es tal dimensión, puesto que $K \simeq k^r$, el enunciado es obvio.

De la demostración de la proposición anterior se deduce también el

Corolario: Salvo isomorfismo, Z/pZ es el único cuerpo posible con un número p de elementos.

En adelante denotaremos por $\mathbb{F}_p$ el cuerpo con p elementos. La pregunta que ahora nos planteamos es ¿existe para todo primo p y todo elemento natural r un cuerpo finito con p^r elementos?

Teorema 5.2.1. $\forall q = p^r, \exists$ *un cuerpo finito con* q *elementos.*

Demostración: Al igual que en el caso primo, construiremos explícitamente tal cuerpo. Sea $\mathbb{F}_p[X]$ el anillo de polinomios sobre el cuerpo primo con p elementos y sea $f(X)$ un polinomio irreducible de grado r. El anillo cociente $A = \mathbb{F}_p[X]/\langle f(X)\rangle$ es un cuerpo. Utilizando la división euclídea es fácil probar que todo polinomio de $\mathbb{F}_p[X]$ posee un único representante en A de grado $< r$. Por tanto, conjuntistamente, A puede identificarse con el conjunto de polinomios de $\mathbb{F}_p[X]$ de grado $< r$, luego su cardinal es p^r.

Más adelante probaremos que, existe básicamente un solo cuerpo finito de cardinal $q = p^r$, que denotaremos con $\mathbb{F}_q$.

La unicidad la podemos ver por el siguiente lema

Lema: Sea $\mathbb{F}_q$ un cuerpo finito contenido, como subcuerpo, en otro cuerpo finito $\mathbb{F}_s$. Sea α un elemento no nulo de $\mathbb{F}_s$. Entonces el subconjunto de $\mathbb{F}_q[X]$ de polinomios que tienen a α por raíz, no se reduce al polinomio cero y coincide con el de múltiplos de un polinomio mónico irreducible.

Demostración: $\mathbb{F}_s$ es un espacio vectorial de dimensión finita sobre $\mathbb{F}_q$. Si r es tal dimensión, entonces el conjunto $\{1,\alpha,...,\alpha^r\}$ es ligado sobre $\mathbb{F}_q$. Una relación de dependencia no trivial proporciona un polinomio no nulo que tiene α por raíz. Sea $f(x)$ el polinomio mónico no nulo de grado mínimo, $f(X)$ es irreducible. En efecto, si $f(X) = f_1(X)f_2(X)$, entonces α será raíz bien de $f_1(X)$ ó $f_2(X)$, lo que contradice el grado de $f(X)$ sea mínimo. Para terminar, sea $g(X)$ un polinomio que tiene α como raíz. Realizando la división

euclidea, podemos escribir $g(X) = f(X)c(X) + h(X)$, siendo $h(X)$ un polinomio de grado menor que el de $f(X)$. Evaluando en α obtenemos que $0 = g(\alpha) = h(\alpha)$, luego $h(X) = 0$, y $g(X)$ es múltiplo de $f(X)$.

5.3. Estructura de los cuerpos finitos

Un cuerpo finito $\mathbb{F}_q$ contiene dos grupos abelianos, $(\mathbb{F}_q, +)$ y $(\mathbb{F}_q^*, \cdot)$. Las estructuras de estos grupos son particularmente simples.

Teorema 5.3.1. ***Estructura aditiva:*** *Si $q = p^r$, el grupo aditivo $(\mathbb{F}_q, +)$ es un producto directo de r grupos cíclicos de orden p:*

$$(\mathbb{F}_q, +) \simeq Z/pZ \times ... \times Z/pZ$$

Para estudiar la estructura multiplicativa, recordemos que dado un grupo abeliano finito $(G, \cdot)$, llamamos orden de $x \in G$ al orden del subgrupo engendrado por x, es decir $\text{ord}(x) = min\{n \mid x^n = 1\}$.

Definición: Sea $(G, \cdot)$ un grupo abeliano finito. Llamaremos *exponente* de G a $\exp(G) = \text{mcm}\{\text{ord}(x) \text{ — } x \in G\}$

Lema: Si $(G, \cdot)$ es un grupo abeliano finito de exponente n, entonces existe un elemento $x \in G$ de orden n.

Teorema 5.3.2. ***Estructura multiplicativa:*** *El grupo multiplicativo $(\mathbb{F}_q^*, \cdot)$ es cíclico de orden $q - 1$.*

Demostración: Sea n el exponente de $\mathbb{F}_q^*$. En virtud del lema anterior debe existir un elemento de orden n. Por tanto $n \leq q - 1 = \#(\mathbb{F}_q^*)$. Por otra parte, por ser n múltiplo del orden de todo elemento, los $q - 1$ elementos de $\mathbb{F}_q^*$ satisfacen la ecuación $X^n - 1 = 0$, con lo que $q - 1 \leq n$ y finalmente $n = q - 1$. Dado que existe un elemento de orden $q - 1$ el grupo es cíclico.

Definición: Llamaremos *elemento primitivo* de $\mathbb{F}_q$ a un generador del grupo cíclico $(\mathbb{F}_q^*, \cdot)$.

Teorema 5.3.3. *Si $q = p^r$ y α es un elemento primitivo de $\mathbb{F}_q$, entonces $\mathbb{F}_q = \mathbb{F}_q[\alpha]$.*

Ejemplo: Dado $\mathbb{F}_2 = \{0, 1\}$, calcular $\mathbb{F}_{2^2}$.

Primero debemos calcular el polinomio irreducible de $\mathbb{F}_2$, es fácil ver que en este caso sólo existe uno que es

$$f(x) = 1 + x + x^2$$

Ahora debemos elegir un $\alpha / f(\alpha) = 0$, y calculando $\beta_0, \beta_1 \in \mathbb{F}_2 / \beta_0\alpha^0 + \beta_1\alpha^1 = 0$, llegamos a que $\mathbb{F}_4 = \{0, 1, \alpha, 1 + \alpha\}$.

Ahora podemos construir la tabla de la adición:

$+$	0	1	α	α^2
0	0	1	α	α^2
1	1	0	α^2	α
α	α	α^2	0	1
α^2	α^2	α	1	0

6. Códigos lineales

6.1. ¿Qué es un código lineal?

Debido al coste de almacenar todas las palabras de un código cualquiera C, los códigos de bloque son computacionalmente costosos y exigen una gran cantidad de memoria (habría que almacenar todas las palabras del código). Esto llevó a buscar códigos con una estructura algebraica añadida. En este caso todas las palabras de los códigos lineales son combinación lineal de otras palabras de C.

Sea $\mathbb{F}_q$ un cuerpo finito con q elementos.

Definición: Un código lineal de longitud n sobre $\mathbb{F}_q$ es un subespacio vectorial de $\mathbb{F}_q^n$.

Todo código lineal C de longitud n, es también un código en bloque de longitud n. Además, como espacio vectorial, C posee una *dimensión*, k, luego su cardinal es siempre una potencia de q, q^k. Estos números, n, k y la distancia mínima d, son llamados *parámetros fundamentales* de C. Para abreviar, de un código cuyos parámetros sean n, k y d, se dice que es de tipo $[n,\ k]$ o $[n,\ k,\ d]$. La redundancia de un código $[n,\ k]$ es $r = n - k$.

La tasa de transmisión de información viene dada por

$$R(C) = \frac{k}{n}$$

Esto nos indicará la cantidad de información promedio por bit.

6.2. Matriz generatriz

Como todo subespacio $\mathbb{F}_q^n$ de dimensión k puede ser interpretado como imagen de una aplicación lineal inyectiva $f : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$, podemos entender que f es la aplicación de codificación y, por tanto, $\mathbb{F}_q^k$ es la fuente de información que estamos codificando con C (siendo C en este caso $Im(f)$). Esto nos lleva a la siguiente definición.

Definición: Llamaremos *matriz generatriz* de un código lineal C a la matriz de una aplicación lineal inyectiva $f : \mathbb{F}_q^k \longrightarrow C \subset \mathbb{F}_q^n$, es decir, una matriz $k \times n$ cuyas filas son la base de C.

Del mismo modo en que existen varias bases de C, existen varias matrices generatrices. Pero todas las matrices generatrices son semejantes de tamaño $k \times n$ y rango k.

Una matriz generatriz G proporciona además la codificación. Por lo que un mensaje $a \in \mathbb{F}_q^k$ se codifica por $aG \in \mathbb{F}_q^n$. De esta forma en lugar de almacenar todas las palabras del código como en un código de bloque normal, sólo hace falta almacenar la matriz generatriz.

Para ver más claramente esto veamos el siguiente ejemplo.

Ejemplo: Consideremos el código binario $\mathbb{F}_2$, C, con matriz generatriz

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

Como podemos ver C tiene longitud 4 y dimensión 3, luego hay $2^3 = 8$ palabras código. Estas son 0011, 1100, 1010, 1111, 1001, 0110, 0101, 0000. Podemos ver que la distancia mínima es 2. La fuente de mensajes es $\mathbb{F}_2^k = \mathbb{F}_2^3$. El mensaje 110 se codifica

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

De ahora en adelante todos los códigos que vamos a tratar serán códigos lineales.

6.3. Matriz de control

Un subespacio vectorial $\mathbb{F}_q^n$ además de poder escribirse como un sistema de generadores, puede también expresarse mediante unas ecuaciones implícitas.

Definición: Diremos que una matriz H es una *Matriz de control* del código C si para todo vector $x \in \mathbb{F}_q^n$ se verifica que $x \in C$ si y sólo si $Hx^t = 0$.

Si C está definido sobre $\mathbb{F}_q$ y es de tipo $[n,\ k]$, entonces también H está definida sobre $\mathbb{F}_q$, es de tamaño $(n-k) \times n$ y rango $n-k$.

Nota: Si G es una matriz generatriz de C dada en forma estándar, $G = (I_k, \widehat{G})$ siendo la primera parte de G la matriz identidad I_k y la segunda la matriz $\widehat{G}$, entonces es fácil ver que la matriz $H = (-\widehat{G}^t, I_{n-k})$ tiene tamaño $(n-k) \times n$, rango $n-k$ y verifica $GH^t = 0$, luego es una matriz de control para C. Diremos que una matriz de control está en forma estándar si es de la forma $(B,\ I_{n-k})$.

Definición: Sea $x = (x_1, ..., x_n) \in \mathbb{F}_q^n$. Llamaremos *soporte* de x al conjunto

$$sop(x) = \{i \mid 1 \leq i \leq n, x_i \neq 0\}$$

Llamaremos peso de Hammming de x a

$$w(x) = \#sop(x) = d(x, O)$$

siendo O el vector $O = (0, 0, ..., 0)$

Definición: El *peso mínimo* de un código C se define como

$$w(C) = min\{w(c) \mid c \in C,\ c \neq 0\}$$

6.4. Código dual

Sea C un código lineal y H una matriz de control de C. Como tiene rango máximo, a su vez H puede ser interpretada como matriz generatriz de otro código sobre $\mathbb{F}_q$. Tal código es llamado código *dual* de C, y denotado por $C^{\perp}$. Obviamente, si C tiene dimensión k, entonces $C^{\perp}$ tiene dimensión $n-k$. Además, si G es una matriz generatriz de C, como la igualdad $GH^t = 0$ implica que $HG^t = 0$, G es una matriz de control para $C^{\perp}$.

Proposición: Si C es un código lineal, entonces su dual $C^{\perp}$ es el ortogonal de C.

Demostración: Sean G y H matrices generatriz y de control de C respectivamente. El resultado enunciado es una consecuencia inmediata de la igualdad $GH^t = 0$, ya que rango(G) + rango$(H) = n$.

Nota: Como la forma bilineal $\langle , \rangle$ es simétrica y no degenerada, se verifica que $(C^{\perp})^{\perp} = C$, es decir, el dual del dual de un código es el propio código.

Nótese que, es posible que $C \bigcap C^{\perp} \neq \{0\}$. El caso extremo se presenta cuando $C = C^{\perp}$. Por supuesto, una condición necesaria para que esto ocurra es que la longitud de C sea par (puesto que debe verificarse $k = n - k$).

Definición: Diremos que un código lineal es *autodual* cuando coincide con su código dual.

A diferencia de lo que ocurre con la misma dimensión, no es posible, en general, determinar la distancia mínima de $C^{\perp}$ únicamente en términos de la distancia mínima de C. De hecho se precisa mucha más información, en términos de polinomios de pesos, cuyo estudio abordamos a continuación.

6.5. Polinomio de pesos

Una información considerable acerca de un código C la proporciona el conocimiento de los pesos de todas las palabras de C. Así, para $1 \leq i \leq n$ (siendo n la longitud de C), consideramos los enteros

$$a_i = a_i(C) = \#\{c \in C | w(c) = i\}$$

Evidentemente $a_0 = 1$, $a_i = 0$ para todo $0 < i < d$ y $\sum_{i=0}^{n} a_i = p^k$, siendo d la distancia mínima y $w(c)$ el peso mínimo de C.

En matemáticas es usual reunir una cantidad de datos que se refieren a la misma entidad en un solo objeto. Siguiendo esta filosofía se define el *polinomio de pesos* (o *enumerador de pesos, o espectro*) de C, como

$$W(X) = \sum_{i=0}^{n} a_i X^i = \sum_{c \in C} X^{w(C)}$$

El polinomio de pesos es fundamental para el estudio de la probabilidad de error en la descodificación. Sin embargo, su cálculo mediante el cómputo directo de los pesos de todas las palabras de un código puede ser muy laborioso; piénsese por ejemplo en el caso de n grande y k próximo a n. En estos casos $n-k$ es pequeño y es mucho más sencillo calcular el polinomio de pesos del código dual.

Ejemplo: Sea $C = \mathcal{H}_2(3)$ el código de Hamming binario de redundancia 3 que ya definimos en el ejemplo anterior. Este código posee dimensión 4, luego $2^4 = 16$ palabras. Puede comprobarse que su polinomio de pesos es

$$W(X) = 1 + 7X^3 + 7X^4 + X^7$$

Su código dual $C^\perp$ tiene dimensión 3, luego 8 palabras. Su polinomio de pesos es

$$W^\perp(X) = 1 + 7X^4$$

El polinomio de pesos de un código determina unívocamente el polinomio de pesos de su dual, mediante la conocida como *identidad de MacWilliams*. Esta identidad es la fórmula fundamental que relaciona un código con su dual, y su enunciado preciso es el siguiente:

Teorema 6.5.1. ***Identidad de MacWilliams:*** *Sean C un código lineal $[n, k]$ sobre $\mathbb{F}_q$ y $C^\perp$ su dual. Si $W(X, Y)$ y $W^\perp(X, Y)$ son los polinomios de pesos de C y $C^\perp$ respectivamente, entonces*

$$W^\perp(X, Y) = q^{-k} W(Y - X, Y + (q-1)X)$$

Propiedad: Si C es un código lineal entonces la distancia mínima es igual al peso mínimo.

6.6. Descodificación de códigos lineales y síndrome

Sea C un código lineal $[n, k, d]$ sobre $\mathbb{F}_q$. Sabemos que C corrige $t = \frac{d-q}{2}$ errores. En esta sección explicaremos de una manera general cómo funciona la descodificación.

Si se envía la palabra $c \in C$ y se recibe el vector $y \in \mathbb{F}_q^n$. El error cometido durante la transmisión ha sido $e = y - c$. Para descodificar la palabra recibida calcularemos la distancia a todas las palabras de C y la descodificaremos por la más cercana. Si durante la transmisión se han cometido más de t errores (es decir $w(e) \leq t$), entonces $d(c, y) = w(e) \leq t$ y c es la única palabra del código con tal propiedad; la descodificación será correcta. Si $t < w(e) < d$, podemos detectar que se han producido errores (puesto que $y \notin C$), pero no corregirlos en general. Si $w(e) \geq d$ la descodificación fallará.

Para corregir los errores cometidos, hay que hablar del *síndrome*. Supongamos que se envía c y se recibe $y = c + e$. Sea H una matriz de control de C.

Definición: Llamaremos *síndrome* de y al vector

$$s(y) = Hy^t \in \mathbb{F}_q^{n-k}$$

Notemos que $y \in C$ si y sólo si $s(y) = 0$. Por tanto, al ser el síndrome una aplicación lineal, $s(y) = s(c+e) = s(c) + s(e) = s(e)$. Así conocer el síndrome del error es inmediato.

Proposición: El síndrome del vector recibido y es una combinación lineal de las columnas de H correspondientes a las posiciones en las que han ocurrido los errores.

Comprobemos la proposición anterior con un caso simple. Supongamos que C corrige al menos un error y que durante la transmisión ha ocurrido un único error. $w(e)$ valdrá pues 1 siendo $e = (0, ..., 0, e_i, 0, ..., 0)$. Por ser $d \geq 3$, cualesquiera dos columnas de H son linealmente independientes. Así el síndrome del vector recibido será múltiplo de una y sólo una columna de H. Según la proposición anterior, la posición de esa columna es precisamente la posición en que se ha cometido el error, es decir, si h_i es la i-ésima columna de H, $s(y) = e_i h_i$, de donde pueden deducirse inmediatamente e y el mensaje enviado $c = y - e$.

Esto se ve de manera mucho más clara con un ejemplo.

Ejemplo: Utilizando el código de Hamming binario $\mathcal{H}_2(3)$ de matriz de control

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

se envía el mensaje $c = 0010110$. Durante el envío se produce un error y se recibe $y = 0011110$. El síndrome del vector recibido es

$$H(0011110)^t = (100)^t$$

Como 100 es la representación binaria de 4, el error ha ocurrido en la cuarta posición, luego y se descodifica por 0010110 que era precisamente el mensaje enviado.

Ahora necesitamos definir el líder de una clase. Consideremos en $\mathbb{F}_q^n$ la relación de equivalencia: $u \sim v$ si y sólo si $u - v \in C$. El espacio vectorial cociente obtenido, módulo tal relación, se denota por $\mathbb{F}_q^n/C$ son clases de equivalencia $u + C = \{u + x | x \in C\}$. Como cada clase posee $\#C = q^k$ elementos, el cardinal de $\mathbb{F}_q^n/C$ es q^{n-k} y su dimensión es $n - k$.

Notemos que u, v están en la misma clase si y sólo si $u - v \in C$, es decir si y sólo si $s(u) = s(v)$. Así, recibido y, al conocer $s(y)$ conocemos la clase a la que pertenece el error cometido.

Definición: Si en una clase existe un único elemento de peso mínimo, este recibe el nombre de líder de la clase.

Algunos autores exigen, demás, que el peso de tal elemento sea $\leq t$ (siendo t la capacidad de corrección del código) para darle el nombre de líder. En cualquier caso, en general no toda clase tendrá líder ya que el elemento de peso mínimo no será, en general, único. Sin embargo, si una clase contiene un elemento de peso $\leq t$, éste es el líder de la clase, como asegura la siguiente proposición.

Proposición: Cada clase de $\mathbb{F}_q^n/C$ posee a lo sumo un elemento de peso $\leq t$.

Demostración: Si existen u, v en la misma clase, ambos de peso $\leq t$ entonces $u - v \in C$ y $w(u - v) \leq w(u) + w(v) \leq 2t < d(C)$, lo cual implica que $u - v = 0$ y $u = v$.

Paso ahora a explicar la descodificación por líder.

Recibido un vector y, descodificar y significa encontrar la palabra de C más próxima a y, y la descodificación será posible si y sólo si esta palabra existe (es decir, si es única). Todos los vectores $y - x, x \in C$, están en la misma clase de $\mathbb{F}_q^n/C$, que es el líder de la clase. Por tanto la descodificación es posible si y sólo si la clase de y, el mínimo de $d(x, y) = w(y - x)$ se obtiene cuando $y - x$ es el líder de la clase. Por tanto la descodificación es posible si y sólo si la clase del vector recibido posee líder, y el error es asumido como el líder de la clase. La proposición anterior garantiza que si el número de errores no supera la capacidad correctora del código, entonces la descodificación es correcta.

Para llevar a cabo este proceso, construiremos una tabla con dos columnas y tantas filas como clases hay en $\mathbb{F}_q^n/C$ (es decir, q^{n-k} filas). En la primera columna escribimos el síndrome de un elemento cualquiera de cada una de las clases; en la segunda el líder de la clase correspondiente (si existe). Esta tabla se construye una vez por todas y sirve para la descodificación de cualquier vector.

Algoritmo: Recibido una palabra$y$,

1.- Calcular $s(y)$ y buscar en la columna de síndromes

2.- Si la clase correspondiente no posee líder, la descodificación falla. Fin.

3.- Si la clase posee líder, e, se dice que e es el error cometido. La palabra descodificada es $y - e$. Fin.

Ejemplo: El código binario de matriz generatriz

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

tiene distancia mínima 5, luego corrige 2 errores. La tabla de síndromes y líderes asociada a este código es la siguiente:

síndrome	líder	síndrome	líder
000000	00000000	100000	00100000
000001	00000001	100001	00100001
000010	00000010	100010	00100010
000011	00000011	100011	11000000
000100	00000100	100100	00100100
000101	00000101	100101	00100101
000110	00000110	100110	00100110
000111		100111	11000100
001000	00001000	101000	00101000
001001	00001001	101001	00101001
001010	00001010	101010	00101010
001011		101011	11001000
001100	00001100	101100	10010000
001101		101101	10010001
001110		101110	10010010
001111	01010000	101111	01110000
010000	00010000	110000	00110000
010001	00010001	110001	00110001
010010	00010010	110010	00110010
010011		110011	11010000
010100	00010100	110100	10001000
010101		110101	10001001
010110		110110	10001010
010111	01001000	110111	01101000
011000	00011000	111000	10000100
011001		111001	10000101
011010		111010	10000110
011011	01000100	111011	01100100
011100	10100000	111100	10000000
011101	01000010	111101	10000001
011110	01000001	111110	10000010
011111	01000000	111111	01100000

Figura 8: Tabla de síndromes.

Supongamos recibido el mensaje 11011011. A su síndrome, 111000^t, según la tabla le corresponde el líder 10000100. El mensaje descodificado es, por tanto, 01011111 = 11011011 + 10000100. Como el líder tiene peso 2, podemos estar seguros de que la descodificación es correcta si han ocurrido a lo más 2 errores.

Supongamos recibido el mensaje 01110010. Su síndrome es 101101^t y el líder de esta clase es 10010001. Aunque el líder tiene peso mayor que la capacidad teórica de corrección del código, es posible descodificar el mensaje recibido por 11100011. En cualquier caso sabemos que han ocurrido al menos tres errores durante la transmisión, y que nuestra descodificación es correcta si y sólo si han ocurrido exactamente tres errores.

Supongamos recibido 01011000. Su síndrome es 000111^t. Como esta clase no tiene líder, no es posible la descodificación. Sabemos además que han ocurrido al menos tres errores.

7. Códigos de Hamming

Los códigos de Hamming son unos excelentes ejemplos de códigos perfectos, así que dedicaremos esta sección a su estudio. La definición de código perfecto se puede encontrar en la sección 9.3.

7.1. Códigos de Hamming binarios

Se desea construir un código en $\mathbb{F}_2$ que sea capaz de corregir un error y que contenga el mayor número de palabras posibles en relación a su longitud. Como estamos en el caso binario y sabemos que, para corregir un error, la distancia mínima ha de ser 3, con lo que necesitamos que dos columnas de la matriz de control sean linealmente independientes.

Definición: Sea r un entero positivo $r \geq 2$. Llamaremos *código de Hamming* binario de redundancia r, $\mathcal{H}_2(r)$, al código cuya matriz de control tiene por columnas a los vectores no nulos de $\mathbb{F}_2^r$.

El código de Hamming $\mathcal{H}_2(r)$ tendrá pues una longitud de $n = \#\mathbb{F}_2^r - 1 = 2^r - 1$ y una dimensión $k = n - r = 2^r - r - 1$. Su redundancia será r.

Ejemplo: $\mathcal{H}_2(2)$ es un código con parámetros [3,1] y con matriz de control

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

Su matriz generatriz es $G = (111)$.

$\mathcal{H}_2(3)$ es un código con parámetros [7,4,3] y con matriz de control

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

7.2. Descodificación de los códigos de Hamming

Para poder descodificar una palabra recibida, nos basaremos de nuevo en el síndrome de la palabra.

Se C un código de Hamming. Como ya hemos visto, la distancia mínima de $\mathcal{H}_2(r)$ es 3, luego C va a poder corregir un error. Supongamos que se ha enviado la palabra c y hemos recibido el vector y. Si $y \in C$ la palabra es correcta y no es necesario descodificarla. Si $y \notin C$, entonces su síndrome coincide con una y sólo una de las columnas de la matriz de control H. Por lo tanto con sólo pasar el síndrome de base 2 a base 10 obtendremos la columna y bastará entonces con modificar el bit para obtener la descodificación correcta.

7.3. Códigos de Hamming no binarios

En este apartado generalizaremos la construcción de un código de Hamming para cualquier cuerpo $\mathbb{F}_q$. En este caso también debemos generar una matriz de control donde cualquier par de columnas sean linealmente independientes. Como cada vector $\mathbb{F}_q^r$ posee $q-1$ múltiplos no nulos, podemos hacer una partición de los q^r-1 vectores no nulos de $\mathbb{F}_q^r$ en $(q^r-1)/(q-1)$ clases y tomar como columnas un vector de cada clase. El código de Hamming de parámetro r es $[n, 2^r-r-1, 3]$ recibe el nombre de *código de Hamming q-ario* de redundancia r, $\mathcal{H}_q(r)$.

Esto se puede ver de manera más clara con un ejemplo.

Ejemplo: $\mathcal{H}_3(3)$ es un código [13, 10] cuya matriz de control es

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \end{pmatrix}$$

Imaginemos ahora que se ha enviado la palabra $c = 0120120120000$ y se ha recibido el vector $x = 0120120122000$ que contiene un error. Calculamos el síndrome de la palabra $S(x) = (2,2,1)^t = 2(1,1,2)^t$. Se puede observar que $(1,1,2)^t$ es justamente la décima columna de la matriz de control, lo que nos indica que el error se ha cometido en la décima posición del mensaje y además que su valor es dos. Por lo tanto, x se descodifica por $c = 0120120122000$ que era la palabra enviada.

8. Códigos de Golay

Los códigos de Golay están formados por una familia de cuatro códigos, en este apartado haremos la presentación mediante el uso de matrices generatrices.

8.1. Códigos autoduales

Sea A una matriz cuadrada $k \times k$ sobre el cuerpo $\mathbb{F}_q$. Se dice que A es una matriz de *conferencia* si $AA^t = (q-1)I_k$. La relación con el código es la siguiente: si A es una matriz de conferencia, consideremos el código C de matriz generatriz $G = (I_k, A)$. C es un código $[2k, k]$ e inmediatamente se prueba que $GG^t = 0$ luego, dada la igualdad de las dimensiones de C y $C^\perp$, concluimos que C es autodual. Recíprocamente, todo código autodual proporciona una matriz de conferencia sin más que escribir su matriz generatriz en forma estándar.

Sea A una matriz de conferencia y C el código (autodual) de matriz generatriz $G = (I_k, A)$.

Proposición: Si A es simétrica, entonces para cada $a, b \in \mathbb{F}_q^k$ tales que $(a, b) \in C$, se verifica que $(-b, a) \in C$.

Vamos a considerar una nueva operación entre vectores de $\mathbb{F}_q^n$: dados $u, v \in \mathbb{F}_q^n$, definimos $u*v$ como el vector cuya i-ésima componente es el producto de las componentes i-ésimas de u y v, es decir, $u*v = (u_1v_1, ..., u_nv_n)$.

En esta sección nos dedicaremos a estudiar únicamente los códigos autoduales definidos sobre $\mathbb{F}_2$ y $\mathbb{F}_3$.

Proposición: Si C es un código autodual definido sobre $\mathbb{F}_2$, entonces todas sus palabras tienen peso múltiplo de 2. En el caso de $\mathbb{F}_3$ todas sus palabras serían múltiplo de 3.

8.2. Códigos de Golay binarios y ternarios

Consideremos la matriz 12×24 sobre $\mathbb{F}_2$

$$G_{24} = \left(\begin{array}{c|cccccccccccc} & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ I_{12} & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array} \right)$$

Si A_{12} es la submatriz derecha de G_{24} y A_{11} la matriz obtenida a partir de ella borrando su primera fila y columna, observemos que A_{11} es construida del modo siguiente: cada fila, a partir de la primera, se obtiene de la anterior desplazando sus elementos una unidad a la izquierda. una matriz de este tipo se llama *circulante* y es siempre simétrica.

Definición: Llamaremos *código de Golay binario* $\mathcal{G}_{24}$, al código binario que tiene a G_{24} por matriz generatriz.

Una primera inspección de G_{24} muestra que $\mathcal{G}_{24}$ es un código [24,12] autodual, ya que A_{12} es una matriz de conferencia. Para evaluar su distancia mínima vamos a estudiar algunas de sus propiedades. Como ya hemos visto anteriormente, todas sus palabras tienen peso múltiplo de 2. Y vimos que esto sólo pasa si y sólo si todas las filas de la matriz generatriz tienen peso múltiplo de 2. Las filas de G_{24} tienen peso múltiplo de 4. Ahora veremos que siendo $\mathcal{G}_{24}$ autodual definido sobre $\mathbb{F}_2$, esto implica que todas sus palabras tienen peso múltiplo de 4. Para ello precisamos de un lema previo.

Lema: Para cada par de vectores $u, v \in \mathbb{F}_2^n$, se verifica que $w(u+v) = w(u) + w(v) - 2w(u * v)$.

Lema: Sean $f_1, ..., f_{12}$ los vectores fila de G_{24}. Para cada par de índices $i, j, 1 \leq i, j \leq 12$, se verifica que

a) Si $i, j \neq 1, i \neq j$, entonces $w(f_i * f_j) = 4$

b) Si $j \neq 1$, entonces $w(f_j * f_j) = 8$ y $w(f_1 * f_j) = 6$

c) $w(f_1 * f_1) = 12$

Nota: Como hemos visto en los resultados anteriores, toda palabra de $\mathcal{G}_{24}$ tiene peso 8, 12, 16, 20 ó 24. Como es habitual, denotemos por a_i el número de palabras de peso i. $a_{24} = 1$, ya que la suma de todas las filas de G_{24} es el vector $1 = (1,...,1)$. Si x tiene peso i, $1 + x$ tiene peso $24 - i$, luego $a_{20} = 0$ y $a_8 = a_{16}$. Por lo que es suficiente conocer uno de estos valores para determinar los restantes. Así, por ejemplo, si $a_8 = \alpha$, entonces $a_{16} = \alpha$ y $a_{12} = 2^{12} - 2 - 2\alpha$.

La distancia mínima de $\mathcal{G}_{24}$ es 8, luego este código corrige 3 errores. También corregiría 3 errores si su distancia fuera 7, lo que da lugar a la definición siguiente.

Definición: Llamaremos *código de Golay binario* $\mathcal{G}_{23}$, al código binario que tiene por matriz generatriz la obtenida borrando una cualquiera de las columnas de G_{24}.

Como curiosidad, vamos a tratar los códigos de Golay ternarios de manera muy breve.

Llamaremos *código de Golay ternario* $\mathcal{G}_{12}$, al código ternario (sobre $\mathbb{F}_3$ de matriz generatriz

$$G_{12} = \left(\begin{array}{c|cccccc} & 0 & 1 & 1 & 1 & 1 & 1 \\ & 1 & 0 & 2 & 1 & 1 & 2 \\ & 1 & 2 & 0 & 2 & 1 & 1 \\ I_6 & 1 & 1 & 1 & 0 & 2 & 1 \\ & 1 & 1 & 1 & 2 & 0 & 2 \\ & 1 & 2 & 1 & 1 & 2 & 0 \end{array} \right)$$

$\mathcal{G}_{12}$ es un código autodual de parámetros [12,6,6], que sólo contiene palabras de peso múltiplo de 3. Suprimiendo una cualquiera de las columnas de G_{12} obtenemos un código [11,6,5], llamado *código de Golay ternario*, $\mathcal{G}_{11}$

8.3. Descodificación de los códigos de Golay

A continuación describiremos el modo de descodificación del código de Golay $\mathcal{G}_{24}$. Para simplificar la notación escribiremos G, H, I, A en lugar de $G_{24}, H_{24}, I_{12}, A_{12}$.

Como sabemos, $\mathcal{G}_{24}$ corrige 3 errores. Supongamos enviada la palabra $c \in \mathcal{G}_{24}$ y recibido el vector $y = c + e$, con $w(e) \leq 3$. Escribamos $e = (e_I, e_D)$ con $e_I, e_D \in \mathbb{F}_2^{12}$. La idea del proceso de descodificación consiste en aprovechar la estructura autodual de $\mathcal{G}_{24}$, debido a la cual G es también una matriz de control del código, y podemos considerar síndromes de y respecto a H y respecto a G. Si denotamos estos síndromes por s_H, s_G,

$s_H(y) = s_H = Hy^t = He^t = (A, I)(e_I, e_D)^t = Ae_I^t + e_D^t = (e_I A + e_D)^t$

$s_G(y) = s_G = Gy^t = Ge^t = (I, A)(e_I, e_D)^t = e_I^t + Ae_D^t = (e_I + e_D A)^t$

Proposición: Si el error cometido $e = (e_I, e_D)$ tiene peso $w(e) \leq 3$ entonces

$$w(s_G) = \begin{cases} \leq 3 & \text{si} \quad e_D = 0 \\ \geq 5 & \text{si} \quad e_D \neq 0 \end{cases} \quad w(s_H) = \begin{cases} \leq 3 & \text{si} \quad e_I = 0 \\ \geq 5 & \text{si} \quad e_I \neq 0 \end{cases}$$

Demostración: Como ambas demostraciones son iguales, realizaremos únicamente la correspondiente a s_G. Como sabemos, $s_G^t = e_I + e_D A$. Si $e_D = 0$ entonces $s_G^t = e_I$, luego $w(s_G) = w(e_I) \leq w(e) \leq 3$. Si $e_D \neq 0$, entonces

$$w(s_G) = w(e_I + e_D A) = w(e_I) + w(e_D A) - 2w(e_I * e_D A)$$

Para acotar esta suma observemos que dados dos vectores cualesquiera a, b, de la misma longitud, $w(a * b) \leq min\{w(a), w(b)\} \leq w(a)$. Por tanto

$$w(s_G) \geq w(e_I) + w(e_D A) - 2w(e_I) = w(e_D A) - w(e_I)$$

y sólo resta estimar el valor de $w(e_D A)$. Notemos que $0 \neq (e_D, e_D A) = e_D\ G \in \mathcal{G}_{24}$ luego $e(e_D) + w(e_D A) = w((e_D, e_D A)) \geq 8$. Trasladando esta igualdad a la fórmula anterior

$$w(s_G) \geq 8 - w(e_I) - w(e_D) = 8 - w(e) \geq 5$$

y el resultado queda probado.

Por tanto, si $w(s_G) \leq 3$ ó $w(s_H) \leq 3$, la decodificación es trivial. En efecto, si $w(s_G) \leq 3$ entonces $e_D = 0$, con lo que $e_I = s_G^t$ y $e = (s_G^t, 0)$. Y del mismo modo, si $w(s_H) \leq 3$ entonces $e_I = 0$, con lo que $e_D = s_H^t$ y $e = (0, s_H^t)$.

Resta estudiar el caso en que $e_I \neq 0$ y $e_D \neq 0$ es decir, el caso en que $w(s_G) \geq 5$ y $w(s_H) \geq 5$. En este caso, y como $w(e) \leq 3$, forzosamente $w(e_I) = 1$ ó $w(e_D) = 1$.

Sean $u_1, ..., u_{12}$ los vectores de la base canónica de $\mathbb{F}_2^{12}$. Si $w(e_I) = 1$ entonces existe un único índice i tal que $e_I = u_i$. Por lo tanto $e_I + u_i = 0$ y el vector $y + (u_i, o)$ no posee ningún error en las 12 primeras coordenadas, con lo que según la proposición anterior,

$$w(s_H(y + (u_j, 0))) = \begin{cases} \leq 3 & \text{si} \quad e_I = u_j \\ \geq 5 & \text{si} \quad e_I \neq u_j \end{cases}$$

y para determinar e_I, es suficiente calcular los 12 síndromes $s_H(y + (u_1, 0)), ..., s_H(y + (u_{12}, 0))$. Entre ellos habrá uno y sólo uno de peso menor o igual a 3; si tal síndrome es el i-ésimo, entonces $e = (u_i, s_H(y + (u_i, 0))^t)$.

Si ninguno de los 12 síndromes anteriores tiene peso menor o igual a 3, es porque $w(e_I) \neq 1$. Realizamos entonces el proceso análogo con los síndromes de s_G. Si alguno de ellos tiene peso menor o igual que 3, entonces $e = (s_G(y + (0, u_i))^t, u_i)$. Si todos los síndromes tienen peso al menos 5, entonces se han cometido más de 3 errores.

En definitiva, el proceso de descodificar $\mathcal{G}_{24}$ requiere el cálculo de, en el peor caso, $2 + 12 + 12 = 26$ síndromes.

9. Cotas en los parámetros de un código

Para saber si los parámetros de los códigos de Hamming y Golay son buenos, es necesario tener un conocimiento preciso de qué valores de k y d son posibles para un n.

9.1. Cotas principales

Tanto la cota de Singleton como la cota de Plotkin son cotas superiores.

Teorema 9.1.1. ***Cota de Singleton:*** *Si C es un código de tipo [n,k,d] sobre $\mathbb{F}_q$, entonces $k + d \leq n + 1$.*

Definición: Los códigos para los que se alcanza la igualdad $k+d = n+1$ son llamados de *máxima distancia de separación* o MDS.

Teorema 9.1.2. ***Cota de Plotkin:*** *Sea C un código de tipo [n,k,d] sobre $\mathbb{F}_q$. Entonces*

$$d \leq \frac{nq^{k-1}(q-1)}{q^k - 1}$$

La cota de Plotkin expresa que la distancia mínima de un código es menor o igual al peso promedio de todas sus palabras no nulas. Los códigos para los que se tiene la igualdad son aquellos en que todas las palabras tienen el mismo peso, d y se denominan *códigos equidistantes*. Un ejemplo de código equidistante sería cualquier dual de los códigos binarios de Hamming.

La siguiente cota se verifica para cualquier código en bloque, sea o no lineal. Para cada $r = 1, ..., n$ existen $\begin{pmatrix} n \\ r \end{pmatrix} (q-1)^r$ vectores en $\mathbb{F}_q^n$ de peso exáctamente r. Por tanto, una bola de radio $t \in N$ centrada en 0 contiene

$$V_q(n,t) = 1 + \begin{pmatrix} n \\ 1 \end{pmatrix} (q-1) + ... + \begin{pmatrix} n \\ t \end{pmatrix} (q-1)^t$$

elementos de $\mathbb{F}_q^n$. Como la distancia de Hamming es invariante por translación, una bola del mismo radio centrada en cualquier vector de $\mathbb{F}_q^n$ posee el mismo número de elementos, $V_q(n,t)$.

Teorema 9.1.3. ***Cota de Hamming:*** *Si C es un código de longitud n sobre $\mathbb{F}_q$ que corrige t errores, entonces*

$$mV_q(n,t) \leq q^n$$

siendo m el número de palabras de C.

Los códigos C para los que se alcanza la igualdad en la cota anterior, son llamados *perfectos*.

9.2. Códigos MDS

Un código C de tipo [n,k,d] es MDS si $d = n - k + 1$ (como hemos visto anteriormente).

Proposición: Si C es MDS entonces también $C^\perp$ es MDS.

Para cualquier entero n existen siempre códigos MDS de longitud n y dimensiones 1, $n-1$ y n; los llamaremos MDS *triviales*. Sobre $\mathbb{F}_2$ estos son los únicos que existen. Sobre otros cuerpos, existen códigos MDS tanto triviales como no triviales.

Teorema 9.2.1. *Sea C un código [n,k,d] MDS sobre $\mathbb{F}_q$. El número de palabras de peso $n-k+r$ en C es*

$$a_{n-k+r} = \begin{pmatrix} n \\ k-r \end{pmatrix} \sum_{i=0}^{r-1} (-1)^i \begin{pmatrix} n-k+r \\ i \end{pmatrix} (q^{r-i} - 1)$$

9.3. Códigos perfectos

Un código lineal C con m palabras y capacidad de corrección t, es *perfecto*, si

$$mV_q(n,t) = q^n$$

es decir, si las bolas de radio t centradas en las palabras del código recubren todo $\mathbb{F}_q^n$. En consecuencia, si C es perfecto, todo vector $y \in \mathbb{F}_q^n$ recibido estará a distancia $\leq t$ de una y sólo una palabra de C (ya que las bolas son disjuntas), y siempre es posible su descodificación de forma única (aunque puede no ser correcta). Dicho de otro modo

Proposición: Si C es perfecto con capacidad de corrección t, entonces todas las clases de $\mathbb{F}_q^n/C$ poseen un líder de peso $\leq t$.

Ahora debemos plantearnos si existen los códigos perfectos. Es fácil comprobar que todos los códigos binarios de repetición impar son perfectos. También es perfecto $\mathbb{F}_q^n$. Estos códigos son llamados *triviales*. En cuanto a los códigos perfectos no triviales, las familias más importantes son las que ya conocemos.

Proposición: Los códigos de Hamming son perfectos. Los códigos de Golay $\mathcal{G}_{23}$ y $\mathcal{G}_{11}$ son perfectos.

Demostración: Es una simple comprobación, teniendo en cuenta que $\mathcal{H}_q(r)$ tiene parámetros $[\frac{q^r-1}{q-1}, \frac{q^r-1}{q-1} - r, 3]$, $\mathcal{G}_{23}$ tiene parámetros [23,12,7] y $\mathcal{G}_{11}$ tiene parámetros [11,6,5].

Esencialmente no existen más códigos perfectos que los anteriores:

Teorema 9.3.1. *Todo código perfecto C no trivial sobre $\mathbb{F}_q$ tiene los mismos parámetros que un código de Hamming o que un código de Golay. Si, además, C es lineal, entonces es equivalente a un código de Hamming o a un código de Golay.*

10. Compresión

En cuanto a los métodos de compresión usando códigos, vamos a centrarnos en la compresión usando códigos de Hamming.

10.1. Compresión sin pérdida

Sea C el código de Hamming binario de parámetros $[2^r - 1, 2^r - r - 1, 3]$. Sean H matriz de control y G matriz generatriz de C en manera sistemática. Supongamos que recibimos una palabra $y = (y_1, ..., y_n)$, el proceso de compresión consistirá en:

1.- Comprobamos si $y \in C$, para ello calculamos Hy y en caso de ser 0, pertenecerá al código.

2.1.- Si $y \in C$ almacenaremos $(1, y_1, ..., y_k)$, el 1 lo almacenamos de manera adicional para saber si la palabra ha sido comprimida o no.

2.2.- Si $y \notin C$ almacenamos $(0, y_1, ..., y_n)$, el 0 nos indicará que la palabra no ha sido comprimida.

Un vector aleatorio está en el código con probabilidad $\frac{q^k}{q^n}$.

El algoritmo de descompresión comprobará si el primer bit es un 1 ó un 0. En caso de ser un 1, haremos

$$(y_1, ..., y_k)G = y$$

Y obtendremos $y = (y_1, ..., y_n)$ la palabra descomprimida. En caso de ser 0 no haremos nada. Para más información sobre la compresión sin pérdida véase [1].

10.2. Compresión con pérdida

Este método es muy similar al anterior, pero en lugar de comprimir sólo las palabras que pertenecen al código se comprimen todas las palabras. Esto supone almacenar k bits por palabra en lugar de n para cada palabra, lo que es un factor de compresión mayor que el anterior.

El proceso completo de compresión, para una palabra y, es el siguiente:

1.- Calculamos el síndrome $S(y) = Hy$

2.1.- En caso de que $S(y) = 0$ almacenamos $\hat{y} = (y_1, \ldots, y_k)$.

2.2.- En caso de que $S(y) \neq 0$, buscamos con qué columna de H se corresponde $S(y)$.

Suponiendo que se cumple 2.2:

3.- Sea i el índice de dicha columna, se hace $y_i = y_i + 1$.

4.- Almacenamos $\widehat{y} = (y_1, ..., y_k)$

Para realizar la descompresión simplemente cogemos $\widehat{y} = (y_1, ..., y_k)$ y realizamos la siguiente operación para cada palabra

$$\widetilde{y} = \widehat{y}G$$

Nótese que $\widetilde{y}$ es de longitud n. Para más información sobre compresión con pérdida véase [2].

Una prueba de concepto de este método se puede ver en mayor profundidad en el tema siguiente.

11. Compresión de imágenes usando códigos de Hamming

En este tema explicaré paso por paso y con los códigos correspondientes cómo comprimir una imagen con códigos de Hamming de distintos parámetros.

11.1. Funciones en python

Necesitamos en primer lugar una función que convierta una imagen RGB a un vector de 1s y 0s. La siguiente función coge cada uno de los bytes, la pasa a binario asegurándose de que todos los números tengan longitud 8 y los añade al vector.

```
def imagen_a_array(img):
        '''
        Esta funcion coge una imagen RGB cualquiera y la pasa a un array de 0s y 1s,
        siendo el tamanyo de palabra 8 bits
        '''

        imagen_array = []

        #La paso a array
        for x in xrange(img.shape[0]):
                for y in xrange(img.shape[1]):
                        for z in xrange(3): #RGB

                                #Cogemos el valor RGB
                                palabra = img[x][y][z]
                                #Pasamos a binario de tamanyo fijo 8
                                palabra = ( bin(palabra)[2:] ).zfill(8)
                                #Pasamos a lista
                                lst = [int(i) for i in str(palabra)]
                                #Anyadimos al array final
                                imagen_array.extend(lst)

        return imagen_array
```

Necesitaremos además una función que nos pase un vector a $\mathbb{F}_2$.

```
def a_base_2(v):
    for i in xrange(len(v)):
        v[i] = v[i] % 2
```

Esta función calcula el síndrome de la palabra $S(v) = Hv$ y el resultado lo traslada a $\mathbb{F}_2$. Después comprueba en qué columna de H se halla, lo que nos da la posición del error. Para finalizar corrige el error y nos devuelve los primeros k bits.

```
def comprimir_palabra(v, H, k):

        '''
        Esta funcion calcula el sindrome de v, decodifica la palabra usando el sindrome
        y devuelve los primeros        k bits.

        '''

        sindrome = np.dot(H,v)
        a_base_2(sindrome)
        ret = v

        #Buscamos a que columna corresponde el sindrome y descodificamos
        for ncol in xrange(len(H[0,:])):

                if (H[:, ncol] == sindrome).all():
                        ret[ncol] = (ret[ncol] + 1) % 2
            break

        return ret[:k]
```

Ahora ya podemos definir las funciones principales, esta se encarga de coger la imagen, pasarla a un array de 0s y 1s, comprimir cada palabra y devolver la imagen comprimida.

```
def comprimir_imagen(src, H, n, k):
    '''
    Esta funcion recibe la ruta de una imagen y los datos del codigo
    y devuelve la imagen comprimida.
    '''

    img = misc.imread(src)
    array = imagen_a_array(img)
    imagen_comprimida = []

    #Parte entera
```

```
    palabras_completas = len(array)/n

    #Comprimimos las palabras
    for i in xrange(palabras_completas-1):

        palabra_comprimida = comprimir_palabra(array[i * n : i * n + n], H, k)
        imagen_comprimida.extend(palabra_comprimida)

    #Anyadimos el final de la imagen, los bits que sobran
    if ((len(array)/float(n)) % 1 != 0):
        start = (palabras_completas-1) * n + n
        imagen_comprimida.extend(array[(palabras_completas-1)*n:])

    #Debemos ademas guardar el shape para poder descomprimirla
    imagen_ret = { "array": imagen_comprimida, "shape": img.shape,
                            "u_size": getsizeof(array), "c_size": getsizeof(imagen_comprimida) }

    return imagen_ret
```

Y por último la función que descomprime. En primer lugar coge todas las palabras del archivo y las descomprime $\widehat{y_i} = y_i G$, la pasa a $\mathbb{F}_2$ y añade los bits sobrantes. Esto nos deja la matriz en binario, así que lo único que nos queda es coger grupos de 8 1s y 0s y pasarlos a decimal. Luego sólo queda pasar el vector a la misma forma que la imagen.

```
def descomprimir_imagen(img_c, G, k):

    img = []
    ret = []

    palabras_completas = len(img_c['array'])/k

    #Decodificamos las palabras
    for i in xrange(palabras_completas-1):
        palabra = img_c['array'][i * k : i * k + k]
        original = np.dot(palabra, G)
        a_base_2(original)
        img.extend(original)

    #Anyadimos el final de la imagen
    if ((len(img_c['array'])/float(k)) % 1 != 0.):
        start = (palabras_completas-1) * k + k
        img.extend(img_c['array'][start:])

    #Pasamos de binario a decimal
    for i in xrange(len(img)/8):
        num = img[i * 8 : i * 8 + 8]
        ret.append(255 - int(''.join(map(str,num)),2))

    #Anyadimos los bits que faltan debido a errores de redondeo
```

```
    while( len(ret) != (img_c['shape'][0] * img_c['shape'][1] * img_c['shape'][2])):
        ret.append(0)

    img = np.array(ret).reshape(img_c['shape'])

    return img
```

11.2. Resultados

Primero vamos a medir tiempos y el ratio de compresión para los distintos parámetros. Para ello usaremos la siguiente pieza de código cambiando los parámetros para cada código. En este caso usaremos los parámetros r=4, 5 y 6.

```
t1 = time()
comprimida = comprimir_imagen("lena512color.tiff", H, n, k)
t2 = time()
img4 = descomprimir_imagen(comprimida, G, k)
t3 = time()

print "Comprimir: ", t2 - t1, " Descomprimir: ", t3 - t2
print "Tamanyo comprimida: ", comprimida['c_size'], " Tamanyo descomprimida: ",\
                comprimida['u_size'], " Ratio: ",\
                float(comprimida['c_size'])/float(comprimida['u_size'])
```

Esto nos da las siguientes salidas:

```
Para r=4
Comprimir:  141.998831987  Descomprimir:  63.5667247772
Tamaño comprimida:  37216888  Tamaño descomprimida:  54872616  Ratio:  0.67824154766

Para r=5
Comprimir:  134.544667006  Descomprimir:  63.943832159
Tamaño comprimida:  43391208  Tamaño descomprimida:  54872616  Ratio:  0.790762518047

Para r=6
Comprimir:  134.1144979  Descomprimir:  62.8220951557
Tamaño comprimida:  46922176  Tamaño descomprimida:  54872616  Ratio:  0.855110971928
```

Pero quizá lo más interesante es ver cómo han quedado las imágenes después de descomprimirlas:

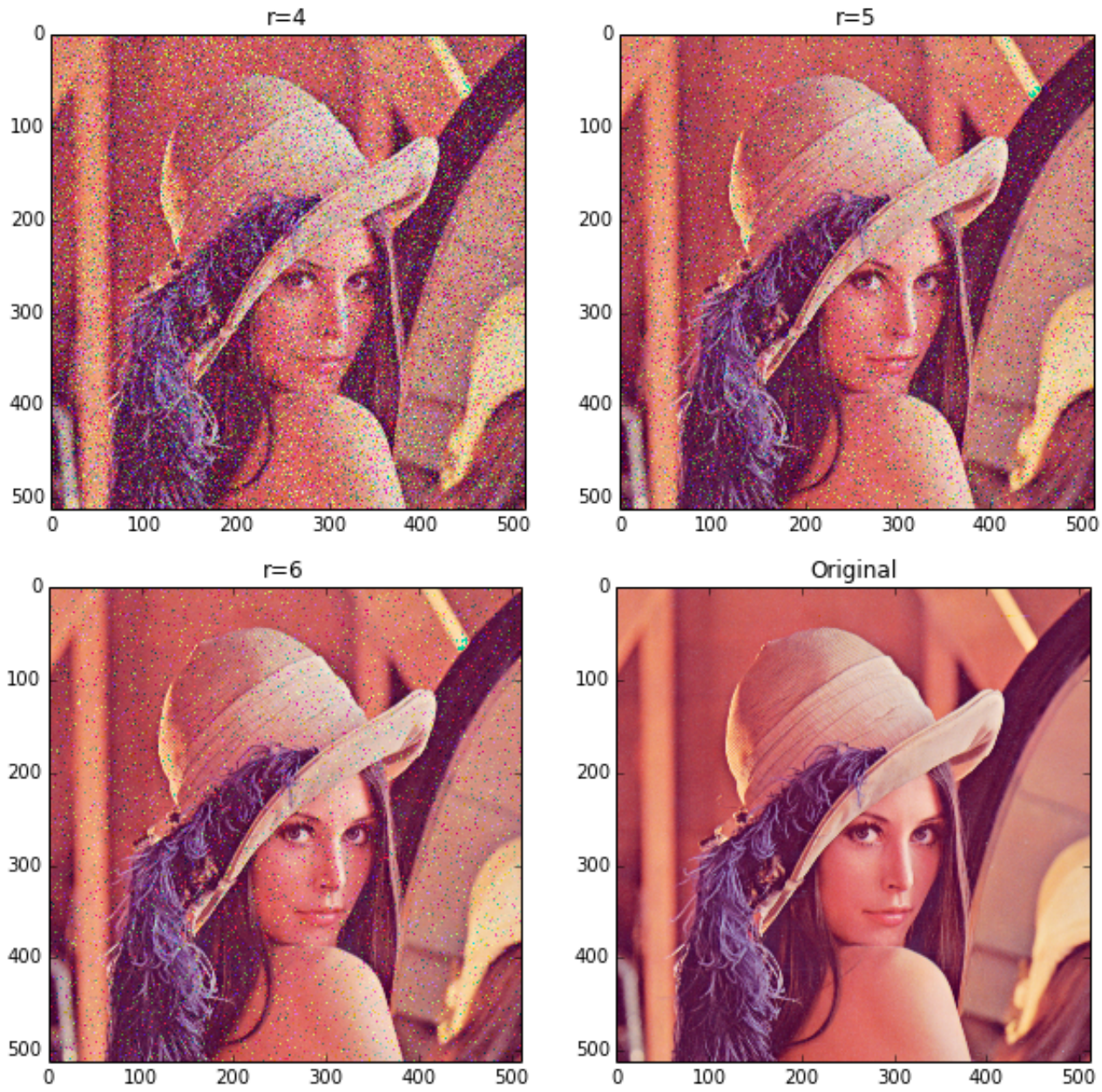

Figura 9: Comparación del error para distintos r.

Se puede ver como ya sabíamos que a medida que r se va haciendo más grande conseguimos menos errores y también menos ratio de compresión.

El código completo lo podéis encontrar pinchando aquí.

Referencias

[1] Al-Bahadili, Hussein A novel lossless data compression scheme based on the error correcting Hamming codes. Comput. Math. Appl. 56 (2008), no. 1, 143–150.

[2] Ancheta, Teofilo C., Jr. Syndrome-source-coding and its universal generalization. IEEE Trans. Information Theory IT-22, 1976.

[3] Blahut, Richard E. Theory and practice of error control codes. Addison-Wesley Publishing Company, Advanced Book Program, Reading, MA, 1983.

[4] J. L. Massey. The codeword and syndrome methods for data compression with error-correcting codes. In J. K. Skwirzynski, editor, New Directions in Signal Processing in Communication and Control, NATO Advanced Study Institutes Series E12, pages 3-13, Leyden, The Netherlands, 1975. Noordhoff.

[5] Justesen, Jørn; Høholdt, Tom. A course in error-correcting codes. EMS Textbooks in Mathematics. European Mathematical Society (EMS), Zürich, 2004.

[6] Lidl, Rudolf; Niederreiter, Harald. Introduction to finite fields and their applications. Revision of the 1986 first edition. Cambridge University Press, Cambridge, 1994.

[7] MacWilliams, F. J.; Sloane, N. J. A. The theory of error-correcting codes. I. North-Holland Mathematical Library, Vol. 16. North-Holland Publishing Co., Amsterdam-New York-Oxford, 1977.

[8] Munuera, Carlos; Tena, Juan. Codificación de la información. Universidad de Valladolid; Edición: 1. Valladolid, 1997.

[9] Roman, Steven. Coding and information theory. Graduate Texts in Mathematics, 134. Springer-Verlag, New York, 1992.

[10] Shannon, Claude Elwood (1948). «A mathematical theory of communication». Bell System Technical Journal 27 (379-423 and 623-656)

[11] Stichtenoth, Henning. Algebraic function fields and codes. Second edition. Graduate Texts in Mathematics, 254. Springer-Verlag, Berlin, 2009.

[12] van Lint, J. H. Introduction to coding theory. Third edition. Graduate Texts in Mathematics, 86. Springer-Verlag, Berlin, 1999.