UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio de Posgrados

Teoría de la información, códigos y aplicaciones de la geometría algebraica

Christian Paúl Llano Robayo

Tesis de grado presentada como requisito
para la obtención del título
de Magíster en Matemáticas Aplicadas

Quito, noviembre de 2011

UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio de Posgrados

HOJA DE APROBACIÓN DE TESIS

Teoría de la información, códigos y aplicaciones de la geometría algebraica

Christian Paúl Llano Robayo

Carlos Jiménez Mosquera, Ph.D.
Director de la Maestría
en Matemáticas Aplicadas
y Director de Tesis. _____

Eduardo Alba Cabrera, Ph.D.
Miembro del Comité de Tesis. _____

Fernando Romo, M.Sc.
Decano del Colegio de
Ciencias e Ingeniería. _____

Víctor Viteri Breedy, Ph.D.
Decano del Colegio de Posgrados. _____

Quito, noviembre de 2011

Resumen. Se estudia la codificación en bloques empezando por la descripción de la codificación de Shanon – Fano – Elias (SFE), luego se tiene a la codificación aritmética como caso particular de SFE, los modelos que se ocupan para su implementación y el método de Lempel–Ziv. Se demuestra la eficiencia de la codificación SFE así como también se presentan ejemplos de codificación aritmética. Por último se estudian temas de geometría algebraica relacionados con los códigos lineales y cíclicos, esta relación se presenta de manera sistemática haciendo una relación entre dos teorías que de inicio podrían parecer ajenas una de la otra: la teoría de la información y el álgebra abstracta.

Abstract. Stream codes are studied with a description of Shanon – Fano – Elias code (SFE), as well as arithmetic coding as a particular case of SFE, the models that are used for its implementation and the Lempel - Ziv method. The efficiency of SFE coding is demonstrated and many examples of arithmetic coding are exposed. Finally we study topics on algebraic geometry in connection with linear codes and cyclic codes, this is presented systematically and making a relation between two theories that at a first view may appear to be different: information theory and abstract algebra.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

El proceso de digitalizar la información ha sido de gran utilidad para las ramas de la computación y las telecomunicaciones, de inicio se pensaba en la manera óptima de transmitir los mensajes a través de los canales de comunicación, así mismo se pensaba en los problemas que tienen que ver con la presencia de interferencia o ruido a través de dichos canales y el saber si tal mensaje llega a su destino de manera que el receptor pueda reconocer el mensaje enviado, o por otro lado, que la información enviada no pueda ser entendida por un receptor no autorizado, todos estos aspectos abarcan dos ramas importantes de las matemáticas aplicadas: la primera, el estudio de la manera óptima de comunicación sobre medios con interferencia, toma el nombre de *teoría de la información*, y la segunda, la que trata de la confidencialidad de la información, se llama *criptografía*. En el presente documento se estudian los conceptos que forman la base de la teoría de la información, aunque se puede mencionar que en ciertos aspectos ambas teorías comparten una misma visión: la comunicación de información.

La revolución informática de las últimas décadas ha puesto al alcance de todos el manejo de información y comunicación de la misma, es común en la actualidad usar medios tales como la telefonía celular, la red Internet, medios de comunicación digitales y un largo etcétera, pero: ¿Qué hay detrás de estas tecnologías? ¿Qué procedimientos se usan para poder captar *físicamente* toda la información que nuestros sentidos perciben, luego procesarla, atravesarla a través de un canal de comunicación y receptarla? ¿Cómo es que esta información aunque finita, pero de gran tamaño, puede ser transformada para ocupar un espacio totalmente menor a su original?

Uno de los problemas que se puede presentar en el proceso de comunicación es el siguiente. Imaginemos un satélite artificial que órbita y monitorea un planeta de

nuestro Sistema Solar, éste envía información (desconocida para nosotros) acerca de su superficie en forma de fotografías. Debido a la gran distancia de separación o problemas en los sistemas de comunicación o por radiación emitida por el Sol en el proceso de envío alguna fotografía se podría ver afectada en su contenido, lo cual podría conducir a conclusiones erróneas al momento de recibir tal información. Se precisa entonces una técnica que ayude a corregir los errores ocasionados por factores externos a la comunicación. La primera idea que surge es el reenvío de la información desde su origen, sin embargo esto puede traer consigo un costo muy alto tanto en recursos como en tiempo. Otra solución consistiría en el envío de la información con un componente de redundancia con el fin de tener copias del mismo objeto observado y compararlas al ser recibidas, esto a su vez lleva el riesgo de verse afectada por la interferencia que ocurre en su camino hacia el otro lado de la comunicación. Fue entonces esencial para el desarrollo de las comunicaciones la creación de modelos para que la información sea transportada de manera eficaz y confiable.

Por otra parte, toda la información que podemos transmitir está sometida también a parámetros de capacidad de transmisión, lo cual requiere en determinado momento que los datos a ser transferidos atraviesen un proceso de compresión. Es así que cientos y miles de escritos de todo el conocimiento humano yacen en libros que ocupan edificios enteros en bibliotecas, pero es claro que las actuales bibliotecas virtuales pueden contener tanta o más información que la incluida en una biblioteca común, almacenando datos en dispositivos que caben en la palma de la mano. Junto con la creación de modelos para el tratamiento de la información sobre medios perturbados, fue necesario también desarrollar modelos que permitieran reducir el tamaño de los datos sin que esto afecte al contenido de la información. Por ejemplo, al comprimir un archivo de tipo texto, se puede observar que el tamaño del archivo en ocasiones se reduce hasta la mitad de su original. Surge entonces la motivación para el estudio de los tipos de codificación utilizados en la compresión de la información.

Este trabajo tiene como finalidad la comprensión de algunas de las técnicas fundamentales utilizadas en la teoría de la información, estudiando los métodos conocidos como codificación por bloques, el método de Shanon – Fano – Elias y el método de Lempel – Ziv. Para llegar a este objetivo seguimos de cerca los capítulos de la primera parte de [5], se han utilizado herramientas de la estadística que permiten optimizar los

métodos de compresión, se demuestran resultados que nos llevan hacia el mismo objetivo y además se realiza una implementación en el sistema Python de la codificación aritmética en un caso particular de la codificación de SFE que nos permite entender el método de manera didáctica.

En el capítulo 3 se introduce la codificación de Lempel–Ziv cuyos algoritmos de compresión se utilizaron en programas de compresión de sistemas Unix, para el formato gráfico GIF, y varios métodos de compresión (en su mayoría patentados) usan el método mencionado.

En el capítulo 4 se revisan tópicos relacionados con la teoría de cuerpos y anillos para luego desembocar en el estudio de dos tipos de códigos: lineales y cíclicos. En lo anterior cabe enfatizar la idea a través del siguiente comentario. La teoría de cuerpos y en sí las estructuras algebraicas forman una rama de las matemáticas que organiza los conjuntos de números que tienen asociados a sus elementos operaciones (llamadas suma o multiplicación), a su vez, estudia las propiedades que permiten encontrar inversos en tales operaciones, también analiza cuando cierta ecuación tiene o no una solución en cierto conjunto y varios otros temas que mencionarlos abarcarían largas páginas. Al inicio de la investigación, parecía algo inverosímil relacionar temas de Geometría algebraica con la teoría de codificación. Sin embargo, con ayuda de varios resultados, mencionados en el capítulo 4, ambas teorías parecen ser un todo; lo que aparentemente parecía antagónico culmina por ser una aplicación no menos interesante que elegante de la geometría algebraica en el área aplicativa de la teoría de la información

Este trabajo se ha realizado pensando en concatenar todos los conceptos fundamentales que son base para el desarrollo de la teoría de la información, se muestran sistemáticamente todos los resultados que se toman del álgebra abstracta u otros temas puramente teóricos para luego relacionarlos con la aplicación. Se presentan varios ejemplos en detalle y algoritmos que han sido implementados en el sistema Python. Un par de resultados no han sido demostrados formalmente por tratarse de temas que salen del contexto del objetivo principal del presente documento que es: introducir al lector a la teoría de la información y mostrar la relación que existe entre la geometría algebraica y dos tipos de códigos muy utilizados los códigos lineales y los códigos cíclicos.

1.1. Definiciones preliminares

Antes de concluir con este capítulo introductorio, revisamos algunos conceptos que serán de mucha utilidad en los siguientes capítulos.

Denotamos por $\mathscr{A}^n$ al conjunto todas las n-uplas de elementos del conjunto $\mathscr{A}$. Además $\mathscr{A}^+$ es el conjunto de todas las cadenas de dimensión finita compuestas de elementos del conjunto $\mathscr{A}$, entendiéndose por cadena a la sucesión de elementos de A. Notamos un intervalo semiabierto como $[a, b)$ al conjunto de los $x \in \mathbb{R}$ tales que $a \leq x < b$

Ejemplo 1.1.1 Dado el conjunto $A = \{0, 1\}$ entonces:

$$A^2 = \{00, 01, 10, 11\}$$

$$A^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

$$A^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \dots\}$$

Definición 1.1.1 Se llama un *ensamble* X a $(x, \mathscr{A}_X, \mathscr{P}_X)$, donde x es una variable aleatoria que toma valores del conjunto $\mathscr{A}_X = \{a_1, \dots, a_i, \dots, a_I\}$ con probabilidades $\mathscr{P}_X = \{p_1, \dots, p_i, \dots, p_I\}$ con $P(x = a_i) = p_i,\ p_i \geq 0$ y $\sum_{a_i \in \mathscr{A}_X} P(x = a_i) = 1$

Definición 1.1.2 Se llama un código simbólico binario para el ensamble X a la transformación C:

$$C : \mathscr{A}_X = \{a_1, \dots, a_I\} \longrightarrow \{0, 1\}^+ \tag{1.1}$$

$$x \longmapsto c(x) \tag{1.2}$$

A $c(x)$ llamaremos la palabra código correspondiente a x y $l(x)$ a su longitud, con $l_i = l(a_i)$

Ejemplo 1.1.2 Sea $\mathscr{A}_X = \{l, m, n, o\}$, definimos C a través de $c : \mathscr{A}_X \to \{0, 1\}^+$ y la longitud de las palabras como sigue en la siguiente tabla

C

a_i	$c(a_i)$	$l(a_i)$
l	1110	4
m	010	3
n	00	2
o	0111	4

Entonces tenemos que $C = \{1110, 010, 00, 0111\}$

Definición 1.1.3 Una palabra código k es un prefijo de otra palabra código d si existe un palabra código t tal que la concatenación kt es d

Definición 1.1.4 Un código simbólico se llama código prefijo, si ninguna palabra código es un prefijo de ninguna otra palabra código.

Ejemplo 1.1.3 Observamos varios casos

- $C_1 = \{0, 01\}$, se observa que 0 es prefijo de 01, por lo tanto no es un código prefijo.
- $C_2 = \{0, 11\}$ es un código prefijo ya que ni 0 es prefijo de 11 ni 11 es prefijo de 0
- $C_3 = \{00, 01, 10, 11\}$ es un código prefijo

Sea el alfabeto fuente $\mathscr{A}_X = \{a_1, ..., a_I\}$ y tomamos el símbolo a_I con el significado especial: "fin de la transmisión". Como se verá más adelante, esta fuente emite una sucesión $x_1, x_2, \ldots, x_n, \ldots$. Asumimos que un programa es proporcionado al codificador que asigna una distribución de probabilidad predictiva sobre a_i dado que antes ocurrió una sucesión $x_1, ..., x_{n-1}$, $P(x_n = a_i | x_1, ..., x_{n-1})$. El decodificador tiene un programa idéntico que produce la misma distribución de probabilidad.

Definición 1.1.5 Se llama contenido de información de Shanon a la expresión

$$h(x = a_i) = \log_2 \frac{1}{p_i} \tag{1.3}$$

Se define la entropía como

$$H(X) = \sum_i (p_i) \log_2 \frac{1}{p_i} \tag{1.4}$$

La longitud promedio de un código C para el conjunto X es

$$L(C, X) = \sum_{x \in \mathscr{A}_X} P(x) l(x) \tag{1.5}$$

Ejemplo 1.1.4 Sea como antes el código binario C, definimos en este código probabilidades para cada uno de sus elementos:

C

a_i	$c(a_i)$	$p(a_i)$
l	1110	1/8
m	010	1/2
n	00	1/8
o	0111	1/4

Entonces tenemos que el contenido de información de Shanon para cada letra es:

C

a_i	$c(a_i)$	$p(a_i)$	$h(a_i)$
l	1110	$\frac{1}{8}$	$\log_2(8) = 3$
m	010	$\frac{1}{2}$	$\log_2(2) = 1$
n	00	$\frac{1}{8}$	$\log_2(8) = 3$
o	0111	$\frac{1}{4}$	$\log_2(4) = 2$

La entropía H es igual a $\frac{1}{8} \cdot 2 + \frac{1}{2} \cdot 1 + \frac{1}{8} \cdot 3 + \frac{1}{4} \cdot 2 = \frac{13}{8}$.
La longitud esperada es $\frac{1}{8} \cdot 4 + \frac{1}{2} \cdot 3 + \frac{1}{8} \cdot 2 + \frac{1}{4} \cdot 4 = \frac{26}{8}$.

Nota 1.1.1 La conversión de un número fraccionario binario a decimal se realiza iniciando por el lado izquierdo del número, cada número se multiplica por dos y se eleva a la potencia consecutiva (empezando por -1), luego estos resultados se suman. Así 0.01101_2 corresponde en decimal a $0*2^{-1}+1*2^{-2}+1*2^{-3}+0*2^{-4}+1*2^{-5} = 0.40625$.

Capítulo 2

Codificación por bloques

Dentro de los métodos de codificación, en el método de Huffman ([5] 5.5) cada símbolo del mensaje codificado tiene una cantidad entera de bits, si tenemos un mensaje de texto, cuyos símbolos se repiten varias veces o en el que se pueden predecir los siguientes símbolos sea por la semántica del texto o por reglas gramaticales, podemos asignar una codificación basándonos en la probabilidad de ocurrencia de cada símbolo, y de esta manera ocupar menos cantidad de bits. Podemos entonces establecer un modelo matemático para conjuntos de textos mediante una sucesión de letras (en algún alfabeto), e introduciendo en esta sucesión una medida de probabilidad.

En contraste al método de Huffman, la codificación aritmética permite representar un mensaje por medio de un intervalo de números reales entre 0 y 1. Cuanto más largo sea el mensaje más corto será el intervalo que lo representa, y el número de bits necesarios para especificar el intervalo crece. Revisamos a continuación un caso general de codificación aritmética.

2.1. Codificación de Shanon-Fano-Elias (SFE)

Supongamos tenemos una fuente aleatoria que toma n valores y de la cual se conoce su función de probabilidad acumulada definida como

$$F(x) = \sum_{a \in X} P(a) \tag{2.1}$$

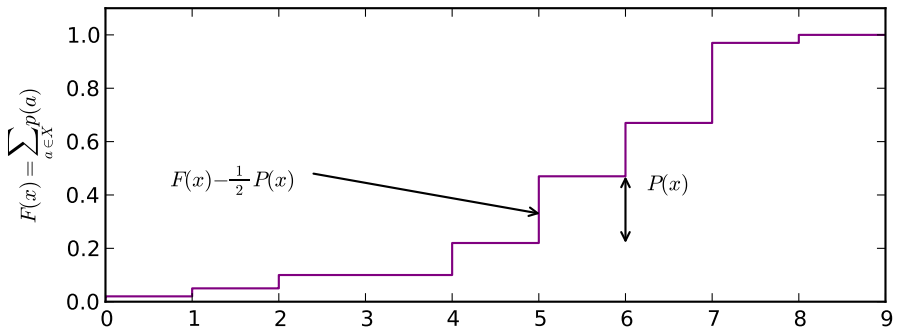

Figura 2.1: Función de probabilidad acumulada (Cpf), $n = 9$

Para codificar una palabra x tomamos el valor $F(x)$ y además $F(x-1)$ que no es más que $F(x) - P(x)$, obtenemos el valor promedio entre estos dos valores que es $F(x) - \frac{1}{2}P(x)$. Luego calculamos el valor

$$l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1 \tag{2.2}$$

que corresponderá a la longitud del código de x. Ya que $F(x) - \frac{1}{2}P(x)$ está en $[0, 1]$, expresamos esta cantidad como un número decimal binario:

$$\left(F(x) - \frac{1}{2}P(x) \right)_2 \tag{2.3}$$

es decir una sucesión de ceros y unos luego de la coma decimal, y tomamos de esta sucesión los primeros $l(x)$ bits, es decir, hemos truncado al número decimal binario en $l(x)$ bits, notamos este número por

$$\left[F(x) - \frac{1}{2}P(x) \right]_{l(x)} \tag{2.4}$$

El valor correspondiente a (2.4) es menor que (2.3) debido justamente a que (2.4) es un valor binario truncado, sin embargo, no es menor que $F(x-1)$. En efecto, cuando tomamos el número $l(x)$ de bits estamos dividiendo el intervalo $[0, 1]$ en $2^{l(x)}$ partes (existen $2^{l(x)}$ combinaciones de $l(x)$ bits en $[0, 1]$). Ya que (2.4) se encuentra en una de estas divisiones, entonces tenemos:

$$\begin{aligned}\left(F(x)-\frac{1}{2}P(x)\right)_2-\left[F(x)-\frac{1}{2}P(x)\right]_{l(x)} &\leq 2^{-l(x)}\\ &= 2^{-\left\lceil\log\frac{1}{P(x)}\right\rceil-1}\\ &\leq 2^{-\log\frac{1}{P(x)}-1}=\frac{P(x)}{2}\end{aligned}$$

y de esta manera

$$\left(F(x)-\frac{1}{2}P(x)\right)_2-\left[F(x)-\frac{1}{2}P(x)\right]_{l(x)}\leq\frac{P(x)}{2} \tag{2.5}$$

lo cual indica que el valor de (2.4) no puede ser menor que $F(x-1)$.

Para cada valor de x realizamos el mismo procedimiento de cálculo y obtenemos así sus palabras código. Probamos que este método de codificación se decodifica de manera única e incluso es un código prefijo.

Proposición 2.1.1 *El código de Shannon-Fano-Elias es un código prefijo.*

Demostración.

Tomamos (2.4) y sumamos 1 a su último bit, obteniendo un nuevo número binario al cual llamaremos $\tilde{X}$. Al sumar 1 al último bit de (2.4) hemos sumado $2^{-l(x)}$ a (2.4), es decir $\tilde{X}=\left[F(x)-\frac{1}{2}P(x)\right]_{l(x)}+2^{-l(x)}$, por (2.5) sabemos que hemos añadido a (2.4) una cantidad menor a $\frac{P(x)}{2}$, luego $\tilde{X}\leq F(x)$. Entonces, todos los números para los cuales (2.4) sea prefijo se encuentran dentro del intervalo $\left[\left[F(x)-\frac{1}{2}P(x)\right]_{l(x)},\tilde{X}\right]$. Para cada valor de x, podemos obtener un intervalo de este tipo, y así observamos que la palabra código que obtenemos para el valor de x no es un prefijo para ningún valor menor o mayor que x, ya que en tal caso estos intervalos se interceptarían, pero hemos dicho que $\tilde{X}\leq F(x)$, lo cual garantiza que estos intervalos no se intercepten. De esta manera el código de SFE es un código prefijo y entonces es un código con decodificación única. □

Por otra parte, nos interesa también en un código simbólico qué tanto podemos comprimir usando esta codificación, es decir, cuál es la longitud media obtenida a través de esta codificación.

Proposición 2.1.2 *La longitud media del código de Shanon-Fano-Elias, no sobrepasa en más de dos bits a su entropía. [5] Ej.6.1*

Demostración.

Calculamos la longitud promedio del código S-F-E

$$\begin{aligned} L &= \sum_i P(x_i) l_i = \sum_i P(x_i) \left(\left\lceil \log \frac{1}{P(x_i)} \right\rceil + 1 \right) \\ &= \sum_i P(x_i) \left\lceil \log \frac{1}{P(x_i)} \right\rceil + \sum_i P(x_i) \\ &= \sum_i P(x_i) \left\lceil \log \frac{1}{P(x_i)} \right\rceil + 1 \\ &\leq \sum_i P(x_i) \left(\log \frac{1}{P(x_i)} + 1 \right) + 1 = \sum_i P(x_i) \log \frac{1}{P(x_i)} + 2 = H(X) + 2 \ \square \end{aligned}$$

Se dice que un código es *óptimo* cuando su longitud media coincide con su entropía, lo anterior muestra entonces que el código SFE es subóptimo (sobrepasa a la entropía en dos bits). Sin embargo, notamos que si juntamos n símbolos en bloques y realizamos la codificación SFE de cada bloque obtenemos como longitud promedio L_n y la longitud promedio L del código sería

$$\begin{aligned} L &= \frac{L_n}{n} \\ &\leq \frac{nH(X) + 2}{n} = H(X) + \frac{2}{n} \overset{n \to \infty}{\longrightarrow} H(x) \end{aligned}$$

Es decir, mientras nuestros bloques de símbolos sean más grandes, (la longitud del código sea más grande) el código SFE se convierte en óptimo, o lo que es lo mismo, es asintóticamente óptimo.

Con lo anterior, estamos listos para introducir la codificación aritmética que es un caso particular de la codificación SFE pero aplicada a bloques de símbolos, que como vimos anteriormente nos permiten (cuando la longitud del código es grande) obtener un código óptimo.

2.2. Codificación aritmética

La codificación aritmética es una forma alternativa de la codificación de SFE, la cual codifica largos bloques de símbolos de manera iterativa, es decir vamos a realizar SFE para un bloque de símbolos y codificarlos secuencialmente . Esto permite reducir el retraso en el envío de información, además de que la longitud de los bloques puede incrementarse sin necesidad de incrementar el retraso en la codificación o decodificación.

Supongamos que tenemos n símbolos y queremos realizar SFE, en primera instancia, como en la sección anterior, necesitamos saber cuál es la función de probabilidad acumulada (Cpf) para éstos n símbolos. Antes de calcular la Cpf debemos disponer todos los bloques de n símbolos en un orden secuencial. Usaremos en adelante el orden lexicográfico para aprovechar sus buenas propiedades como se muestra a continuación.

Definición 2.2.1 Dado el alfabeto $\mathscr{A}_X$, el bloque de n símbolos $a_{i_1}, a_{i_2}, \ldots, a_{i_n}$ es lexicográficamente menor que (simbolizamos $\prec$) $a_{j_1}, a_{j_2}, \ldots, a_{j_n}$, es decir:

$$a_{i_1}, a_{i_2}, \ldots, a_{i_n} \prec a_{j_1}, a_{j_2}, \ldots, a_{j_n} \text{ ssi } i_k = j_k \text{ para } k < l \text{ y } i_l \leq j_l$$

Ejemplo 2.2.1 Sea el alfabeto $\{a, b, c, \ldots, z\}$, tomemos como bloque de letras a *bcv*, *bzb*, *cab*. Se tiene que $n = 3$ consideremos el par *bcv* y *bzb* tenemos que $a_{i_1} = a_{j_1} = b$ además $i_2 \leq j_2$ por tanto $bcv \prec bzb$, por otro lado $bzb \prec cab$ ya que $i_1 \leq j_1$. Como se puede observar, el orden es el que aparece en los diccionarios.

Sea $\mathbf{x}^{(i)} \stackrel{def}{=} (x_1, x_2, \ldots, x_i)$. Definimos la Cpf como

$$F(\mathbf{x}) = \sum_{\mathbf{y}^{(i)} \prec \mathbf{x}^{(i)}} P\left(\mathbf{y}^{(i)}\right) \tag{2.6}$$

Es decir, definimos la Cpf como la suma de todas las i-uplas (*palabras*) que vienen antes de $\mathbf{x}^{(i)}$. Ahora extendemos la sumatoria de la definición anterior y obtenemos

$$
\begin{aligned}
F(\mathbf{x}) &= \sum_{\mathbf{y}^{(i)} \prec \mathbf{x}^{(i)}} P\left(\mathbf{y}^{(i)}\right) \\
&= \sum_{\mathbf{y}^{(i-1)} \prec \mathbf{x}^{(i-1)}} P\left(\mathbf{y}^{(i-1)}\right) + \sum_{y_i \prec x_i} P\left(\mathbf{x}^{(i-1)}, y_i\right) \\
&= F\left(\mathbf{x}^{(i-1)} - 1\right) + \sum_{y_i \prec x_i} P\left(\mathbf{x}^{(i-1)}\right) P\left(y_i | \mathbf{x}^{(i-1)}\right) \\
&= F\left(\mathbf{x}^{(i-1)} - 1\right) + P\left(\mathbf{x}^{(i-1)}\right) \sum_{y_i \prec x_i} P\left(y_i | \mathbf{x}^{(i-1)}\right) \\
&= F\left(\mathbf{x}^{(i-1)} - 1\right) + P\left(\mathbf{x}^{(i-1)}\right) F\left(x_i | \mathbf{x}^{(i-1)}\right) \qquad (2.7)
\end{aligned}
$$

Observemos detenidamente lo anterior. Supongamos que hemos calculado el valor de la Cpf definida para un bloque de $i-1$ símbolos que hemos recibido, mantenemos este cálculo (primer sumando de (2.7)). Luego de recibir el símbolo i encontramos la Cpf de la secuencia hasta i usando el cálculo previo: el primer sumando está calculado, $P\left(\mathbf{x}^{(i-1)}\right)$ de (2.7) es conocido al recibir el símbolo $i-1$ y únicamente lo que debemos calcular es $F\left(x_i|\mathbf{x}^{(i-1)}\right)$ de (2.7), que corresponde a la función de probabilidad acumulada, condicionada a $\mathbf{x}^{(i-1)}$. De esta manera, observamos que en el caso de la codificación aritmética no es necesario calcular todas las probabilidades de la función de probabilidad acumulada, lo cual da una ventaja de esta codificación frente, por ejemplo a la codificación de Huffman, en la cual requerimos calcular todas las probabilidades del conjunto de símbolos recibidos.

Calculamos $F(\mathbf{x}^{(i)})$ de forma iterativa. Conocemos hasta este momento tres valores: $F(\mathbf{x}^{(i-1)})$, $P(\mathbf{x}^{(i-1)})$ y $F(\mathbf{x}^{(i-1)}-1)$, que forman el intervalo $[F(\mathbf{x}^{(i-1)}-1), F(\mathbf{x}^{(i-1)})]$ cuando recibimos el i–ésimo símbolo calculamos $F\left(x_i|\mathbf{x}^{(i-1)}\right)$, $P\left(x_i|\mathbf{x}^{(i-1)}\right)$ y $F\left(x_i - 1|\mathbf{x}^{(i-1)}\right)$ que forman el intervalo $\left[F\left(x_i - 1|\mathbf{x}^{(i-1)}\right), F\left(x_i|\mathbf{x}^{(i-1)}\right)\right]$ y por último debemos comprimir este último intervalo dentro del intervalo anterior, reescalando los valores de tal manera que los extremos del anterior intervalo corresponda a los extremos del intervalo $[0, 1]$. Entendemos por reescalar el intervalo en multiplicar sus valores por el factor $P(\mathbf{x}^{(i-1)})$ y realizamos este proceso para todos los símbolos que recibimos.

La codificación final será representada por un subintervalo $[\alpha, \beta)$ dentro del intervalo $[F(x^{(i)} - 1), F(x^{(i)})]$ y la codificación, corresponderá a la representación binaria de α.

Si $F(\mathbf{x}^{(i)} - 1)$ y $F(\mathbf{x}^{(i)})$ tienen la misma representación binaria para algunos bits

iniciales, entonces la codificación del bloque, que es un número binario que se encuentra dentro de este intervalo, va a tener como prefijo justamente los bits comunes para los dos números, es decir de antemano podemos conocer los primeros bits de la codificación sin necesidad de esperar toda la información enviada. Mientras i se incrementa el intervalo $[F(\mathbf{x}^{(i)}-1), F(\mathbf{x}^{(i)})]$ es cada vez más pequeño, y habrá más bits comunes entre $F(\mathbf{x}^{(i)}-1)$ y $F(\mathbf{x}^{(i)})$.

Ahora, necesitamos saber cómo calcular el factor $P(x_i|\mathbf{x}^{(i-1)})$, que es la clave para calcular iterativamente la Cpf. Hay casos especiales, donde el cálculo de esta probabilidad es simple, por ejemplo, cuando tenemos una fuente que emite símbolos independientes idénticamente distribuidos donde sabemos que $P(x_i|\mathbf{x}^{(i-1)}) = P(x_i)$. Otro caso en el cual el cálculo es simple corresponde a las fuentes con procesos Markovianos, donde $P(x_i|\mathbf{x}^{(i-1)}) = P(x_i|x_{i-1})$. Sin embargo puede darse el caso que no sabemos de antemano las estadísticas de la fuente, en este caso, la idea que podemos aplicar es que mientras recibamos símbolos podemos estimar la distribución de probabilidad de la fuente a la vez, y usar esa distribución para hacer la codificación. Un modelo para realizar esta estimación es el modelo de Laplace, en el cual tenemos

$$P(X_i = \alpha|\mathbf{x}^{(i-1)}) = \frac{F_\alpha + 1}{\sum_{\lambda\in\mathscr{A}_X} F_\lambda + 1} \tag{2.8}$$

donde F_λ es el número de símbolos λ que aparecen en $\mathbf{x}^{(i-1)}$. Es decir, estamos calculando un tipo de frecuencia de las letras en el bloque de símbolos.

Otro modelo para estimar es el modelo de Dirichlet:

$$P(X_i = \alpha|\mathbf{x}^{(i-1)}) = \frac{F_\alpha + \epsilon}{\sum_{\lambda\in\mathscr{A}_X} F_\lambda + \epsilon} \tag{2.9}$$

vemos que en el modelo de Laplace se tiene $\epsilon = 1$. Si $\epsilon < 1$ entonces se tiende a aprender el estadístico rápidamente, sin embargo tendremos un mayor error en la estimación.

2.3. Modelo básico de codificación aritmética

Para comprender la idea de la codificación aritmética, comencemos trabajando con números decimales (aun no binarios) y para lo cual codificamos el mensaje 'MI EJEMPLO' y hacemos corresponder a cada símbolo su probabilidad de ocurrencia en el mensaje, como se indica en el siguiente Cuadro 2.1:

Símbolo	Probabilidad
Espacio	1/10
E	2/10
I	1/10
J	1/10
L	1/10
M	2/10
O	1/10
P	1/10

Cuadro 2.1: Probabilidades para "MI EJEMPLO"

Luego damos a cada símbolo un intervalo de probabilidad entre 0 y 1. La longitud del intervalo para cada símbolo es igual a su probabilidad de aparición en el mensaje. La posición del intervalo de probabilidad no tiene significado, lo que si es importante es que tanto el codificador como el decodificador, hayan distribuido los símbolos mediante reglas idénticas. Los intervalos se muestran en la Cuadro 2.2:

Símbolo	Probabilidad	Intevalo
Espacio	1/10	[0,0.1)
E	2/10	[0.1,0.3)
I	1/10	[0.3,0.4)
J	1/10	[0.4,0.5)
L	1/10	[0.5,0.6)
M	2/10	[0.6,0.8)
O	1/10	[0.8,0.9)
P	1/10	[0.9,1)

Cuadro 2.2: Intervalos correspondientes a "MI EJEMPLO"

En términos generales el algoritmo para la codificación aritmética puede ser escrito:

Algoritmo 1

```
Inf= 0.0
Sup= 1.0
While((Símbolo)!= Fin){
        Long=Sup - Inf
        Sup=Inf+Long*SupPara(Símbolo)
        Inf=Inf+Long*InfPara(Símbolo)
}
```

```
Return (Inf)
```

Para el ejemplo tenemos los resultado en el Cuadro 2.3

Símbolo	Inf	Sup
	0.0	1
M	0.6	0.8
I	0.66	0.68
Espacio	0.660	0.662
E	0.6602	0.6606
J	0.66036	0.6604
E	0.660364	0.660372
M	0.6603688	0.6603704
P	0.66037024	0.6603704
L	0.66037032	0.660370336
O	0.6603703328	0.6603703344

Cuadro 2.3: Codificación de "MI EJEMPLO"

De esta manera tenemos la codificación del mensaje. El algoritmo para la decodificación del mensaje es:

Algoritmo 2

```
Símbolo= Encontrar(n)
#función Encontrar,
#encuentra símbolo en el intervalo donde está el número "n"
While((Símbolo)!=Fin){
        Emision(Símbolo)Interv=SupPara(Simb)-InfPara(Simb)
        n=n-InfPara(Simb)
        n=n/Interv
        }
```

Para nuestro ejemplo los resultados del algoritmo anterior serán:

n	Símbolo	Inf	Sup	Intervalo
0.6603703328	M	0.6	0.8	0.2
0.301851664	I	0.3	0.4	0.1
0.01851664	Espacio	0.0	0.1	0.1
0.1851664	E	0.1	0.3	0.2
0.425832	J	0.4	0.5	0.1
0.25832	E	0.1	0.3	0.2
0.7916	M	0.6	0.8	0.2
0.958	P	0.9	1	0.1
0.58	L	0.5	0.6	0.1
0.8	O	0.8	0.9	0.1

Cuadro 2.4: Decodificación

Procedemos a implementar los algoritmos anteriores. Primero realizamos un programa (frequencies) para calcular las frecuencias de las letras en un texto dado, retorna un vector que contiene las letras y sus frecuencias, luego tenemos un programa para codificar el cual retorna el número real entre 0 y 1 que codifica el texto "MI EJEMPLO" y otro para decodificar, el cual retorna en cada paso las letras decodificadas.

```
from mpmath import *
mp.dps = 30; mp.pretty = False
def frequencies(s):

        count=[] #lista vacia
        substring=''
        k=0.0
        k=len(s)
        z=0.0
        inf=0.0

        for j in range(len(s)): #subcadena para cada elemento
            substring=substring+s[j]
        substring=substring.upper()  #cambiamos mayusculas
        for ch in ' ABCDEFGHIJKLMNOPQRSTUVWXYZÁÉÍÓÚÑ ':
        #frecuencias de cada letra
            if substring.count(ch)>0:
```

```
                z=fdiv(substring.count(ch),k)
                count.append([ch,z,inf,fadd(inf,z)])
                #añada a la lista y calcula
                #la frecuencia de la letra
                inf=fadd(inf,z)

        #print count
        return count

def encode(cod):
    inf=0.0
    sup=1.0
    x=[]
    x=frequencies(cod)
    i=0
    while(i < len(cod)):
        lng=fsub(sup ,inf)
        for j in range(len(cod)):
            if(cod[i] == x[j][0]):
                sup=fadd(inf,fmul(lng,x[j][3]))
                #print sup
                inf=fadd(inf,fmul(lng,x[j][2]))
                print inf
                break
        i=i+1
    return inf

def decode(n,cod):
    x=frequencies(cod)
    itrv=0.0
    i=0
```

```
    while(i<len(cod)):

        for j in range(len(cod)):
            if(x[j][2]<=n and n<x[j][3]):
                print x[j][0]
                print x[j][3]
                print x[j][2]
                itrv=fsub(x[j][3],x[j][2])
                print itrv
                #print n
                n=fsub(n,x[j][2])
                n=fdiv(n,itrv)
                print n
                print j
                break
        i=i+1

def main():
    code='MI EJEMPLO'
    code=code.replace('\n','')  #quita los fines de renglón
    x=[]
    x=frequencies(code)
    print x
    print "Número correspondiente a codificación de cadena: "
    y=encode(code)
    print y
    #print x
    print "Decodificación correspondiente: "
    decode(0.6603703328,code)
main()
```

2.4. Modelo probabilístico de codificación aritmética

Una transmisión binaria define un intervalo en la recta real desde 0 hasta 1. Por ejemplo la cadena **01** es interpretada como un número real binario 0.01... que corresponde a un intervalo $[0.01, 0.10)$ en binario, y que corresponde a su vez a $[0.25, 0.50)$ en base 10. A la cadena más larga **01101** le corresponderá un intervalo más pequeño $[0.01101, 0.01110)$, además, ya que la cadena **01101** tiene como prefijo a **01**, notamos que el intervalo de **01101** es un subintervalo del correspondiente a **01**. Así 0.01101_2 corresponde en decimal a 0.40625 y 0.01110_2 corresponde a 0.4375. Entonces el intervalo correspondiente sería $[0.40625, 0.4375)$ que es subintervalo de $[0.25, 0.50)$.

Podemos dividir el intervalo $[0, 1)$ en I subintervalos de longitud igual a las probabilidades $P(x_1 = a_i)$ como se muestra a continuación en la Figura 2.2:

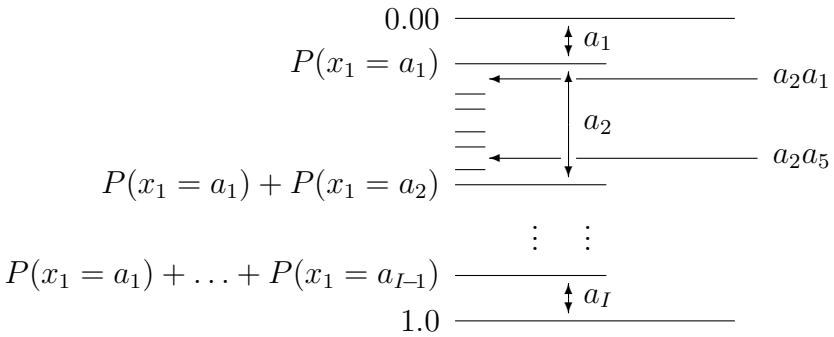

Figura 2.2: Modelo probabilístico

Además cada intervalo a_i se lo puede subdividir en intervalos $a_ia_1, a_ia_2, ..., a_ia_I$ tal que la longitud de los a_ia_j sea proporcional a $P(x_2 = a_j|x_1 = a_i)$, de hecho el intervalo de a_ia_j es justamente la probabilidad conjunta $P(x_1 = a_i, x_2 = a_j) = P(x_1 = a_i)P(x_2 = a_j|x_1 = a_i)$.

Iterando este proceso podemos dividir el intervalo en una sucesión de intervalos correspondientes a todas las cadenas finitas posibles, de tal manera que la longitud de un intervalo es igual a la probabilidad de la cadena dada por el modelo.

En el ejemplo que describimos a continuación, vamos a comprimir los lanzamientos de una moneda sesgada, los dos resultados posibles del lanzamiento lo simbolizamos mediante **a** y **b**. Una tercera posibilidad es que se deje de lanzar la moneda y esto lo representamos mediante □. Ya que la moneda es sesgada, las probabilidades de **a** y **b**

no serán iguales, además de que no conocemos qué valor es más probable de antemano.

Supongamos que necesitamos codificar la cadena **bbba**□. Enviamos uno a uno los símbolos de la cadena y usamos un modelo, para calcular la distribución de probabilidad del siguiente símbolo dados los símbolos que hayamos recibido. Así el modelo que ocuparemos será

1. $P(\square|\mathbf{x}^{(i-1)}) = 0.15$
2. $P(x_i = a|\mathbf{x}^{(i-1)}) = 0.85 \cdot \frac{F_a+1}{F_a+F_b+2}$
3. $P(x_i = b|\mathbf{x}^{(i-1)}) = 0.85 \cdot \frac{F_b+1}{F_a+F_b+2}$

   Donde F_a es la cantidad de **a** que existen en $\mathbf{x}^{(i-1)}$ y F_b es la cantidad de **b** que existen en $\mathbf{x}^{(i-1)}$

Mediante este modelo podemos calcular las probabilidades condicionadas:

Contexto $\mathbf{x}^{(i-1)}$	Probabilidades Condicionadas		
	$P(x_i = \mathbf{a}\ \|\mathbf{x}^{(i-1)})$	$P(x_i = \mathbf{b}\|\mathbf{x}^{(i-1)})$	$P(x_i = \square\|\mathbf{x}^{(i-1)})$
	$P(\mathbf{a}) = 0.425$	$P(\mathbf{b}) = 0.425$	$P(\square) = 0.425$
b	$P(\mathbf{a}\|\mathbf{b}) = 0.28$	$P(\mathbf{b}\|\mathbf{b}) = 0.57$	$P(\square\|\mathbf{b}) = 0.15$
bb	$P(\mathbf{a}\|\mathbf{bb}) = 0.21$	$P(\mathbf{b}\|\mathbf{bb}) = 0.64$	$P(\square\|\mathbf{bb}) = 0.15$
bbb	$P(\mathbf{a}\|\mathbf{bbb}) = 0.17$	$P(\mathbf{b}\|\mathbf{bbb}) = 0.68$	$P(\square\|\mathbf{bbb}) = 0.15$
bbba	$P(\mathbf{a}\|\mathbf{bbba}) = 0.28$	$P(\mathbf{b}\|\mathbf{bbba}) = 0.57$	$P(\square\|\mathbf{bbba}) = 0.15$

Cuadro 2.5: Cálculo de probabilidades

La Figura 2.3 muestra los intervalos correspondientes, por ejemplo el intervalo **b** es 0.425 del centro del intervalo $[0, 1)$

2.4.1. Codificación

Cuando el primer símbolo **b** es recibido, el codificador sabe que la codificación iniciará con '01', '10' o '11' pero no sabe exactamente cuál de ellos, es decir, no podemos codificar aún, debemos esperar el siguiente símbolo que es **b**. El intervalo **bb** se encuentra completamente dentro del intervalo '1', luego el codificador puede escribir el primer bit '1'. El tercer símbolo **b** está dentro del intervalo pero no lo suficiente para estar dentro del intervalo '10'. Únicamente cuando se lee el siguiente símbolo **a** podemos transmitir más bits, el intervalo **bbba** está completamente dentro del intervalo '1001', luego el codificado añade '001' al ya codificado '1'. Finalmente cuando □ es

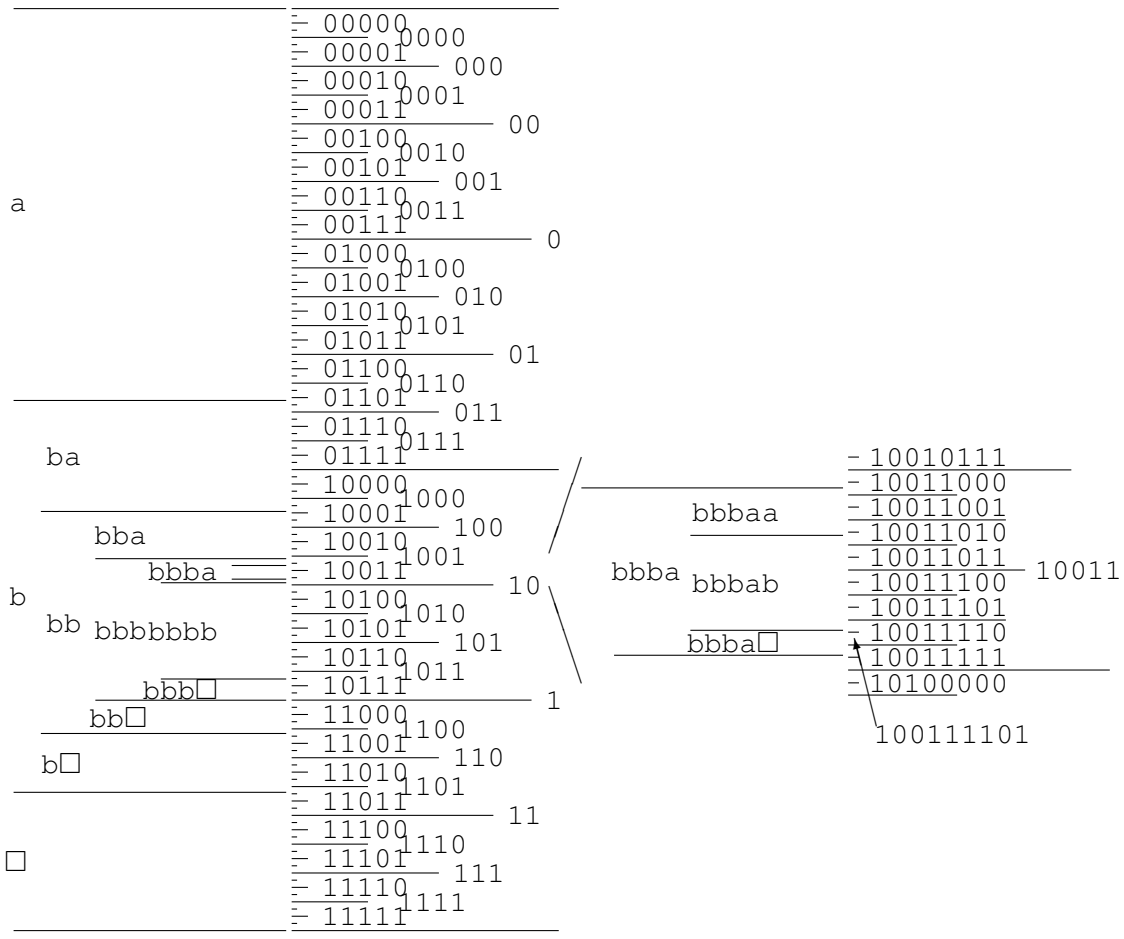

Figura 2.3: Codificación aritmética de **bbba□**

leído. Como se observa en la Figura 2.3 en la parte derecha, se ha hecho un *zoom* el intervalo **bbba**□ y el intervalo señalado '100111101' está completamente contenido en **bbba**□ y añadimos '11101' a lo ya codificado.

2.4.2. Decodificación

El decodificador recibe la cadena '100111101'. Las probabilidades de **a**, **b** y □ se calculan de manera idéntica ya que tanto codificador como decodificador tienen el mismo modelo y pueden reproducir todas las probabilidades calculadas. Cuando vemos el primer bit '1', observando la Figura 2.3 vemos que puede ser el primer símbolo o bien **b** o □. Tomamos el segundo bit y tenemos '10' podemos escribir como primer símbolo decodificado **b** ya que esta dentro del intervalo cuyo prefijo es '10', sin embargo, no podemos escribir aún el segundo símbolo decodificado ya que puede ser **ba** o **bb**, con los siguientes dos símbolos '1001', podemos asegurar que la decodificación es **bb**, continuamos el proceso hasta que decodificamos □ que indica el fin del mensaje.

Capítulo 3

Codificación de Lempel – Ziv

El método de codificación de Lempel-Ziv (LZ) difiere del método de codificación arit-mética esencialmente en que no hay una separación entre el modelo y la codificación. Como pudimos comprobar en el capítulo anterior, la codificación aritmética tiene asociada un modelo estadístico. En el método LZ no hablamos de un modelo para elaborar la codificación, sino que vamos creando un código a partir de los mismos mensajes a ser comprimidos, se crea un diccionario de palabras código las cuales pueden ser reutilizadas en futuro únicamente ocupando punteros a su posición.

3.1. Algoritmo L – Z

El proceso de codificación que ocupa LZ se resume mediante los siguientes pasos:

1. Conservar una lista de subcadenas que ha sido recibida previamente.
2. Guardar en el buffer las cadenas que obtenemos de la fuente.
3. Comprobamos si la cadena actual en el buffer está en la lista:

   a. Si está, esperamos por un bit más que llegue al buffer y regresamos al paso 3.

   b. Si no está:

      i) Encontrar la subcadena en la lista (cadena del buffer excluyendo el último símbolo), y transmitir su índice.

      ii) Transmitir el último símbolo.

      iii) Vaciar el buffer.

Así, el método de compresión consiste en remplazar una subcadena con un puntero hacia una ocurrencia anterior de la misma subcadena. Por ejemplo si la cadena es $1011010100010\ldots$, la convertimos en un diccionario ordenado de subcadenas que no han aparecido antes, es decir: λ, 1, 0, 11, 01, 010, 00, 10, ... λ representa una subcadena vacía y será la primera subcadena de nuestro diccionario, ordenamos las subcadenas en el orden que aparecen desde la fuente. Luego de cada coma leemos a lo largo de la sucesión hasta que leamos una subcadena que no haya aparecido antes. De lo anterior podemos concluir que esta subcadena es más larga en un bit que alguna subcadena que haya ocurrido antes en el diccionario. Esto significa que podemos codificar cada subcadena mediante un puntero hacia la ocurrencia anterior de ese prefijo y luego enviar un bit extra por el cual la nueva subcadena en el diccionario, difiere de la subcadena anterior. Si en el n-ésimo bit hemos enumerado $s(n)$ subcadenas, entonces podemos dar el valor del puntero en $\lceil \log s(n) \rceil$ bits. En la tabla que sigue mostramos la codificación para la sucesión anterior

Subcadenas	λ	1	0	11	01	010	00	10
$s(n)$	0	1	2	3	4	5	6	7
$s(n)_2$	000	001	010	011	100	101	110	111
(puntero,bit)		$(,1)$	$(0,0)$	$(01,1)$	$(10,1)$	$(100,0)$	$(010,0)$	$(001,0)$

Cuadro 3.1: Subcadenas para el ejemplo

Proposición 3.1.1 *Si un código únicamente decodificable desde* $\{0,1\}^+$ *hacia* $\{0,1\}^+$ *hace que algunas cadenas sean mas cortas, entonces otras cadenas se haran más largas*

Demostración.

Ya que tenemos un código únicamente decodificable, entonces se cumple la desigualdad de Kraft $\sum_{i=1}^{I} 2^{-l_i} \leq 1$, donde $I = |\{0,1\}^+|$. Si existe alguna palabra cuya codificación tenga menos bits que élla misma, entonces su aporte en la parte izquierda de la desigualdad incrementará el valor en la sumatoria, para evitar que la sumatoria exceda a 1, necesariamente algunos de los sumandos deben disminuir en valor, lo cual implica que la longitud de algunas codificaciones debe aumentar en longitud (así 2^{-l_i} disminuye)

3.2. Aplicación del algoritmo L – Z

Codificamos la cadena 000000000000100000000000. Creamos primero el diccionario, hacemos la numeración y por último codificamos

Subcadenas	λ	0	00	000	0000	001	00000	000000
$s(n)$	0	1	2	3	4	5	6	7
$s(n)_2$	000	001	010	011	100	101	110	111
(puntero,bit)		$(,0)$	$(1,0)$	$(10,0)$	$(11,0)$	$(010,1)$	$(100,0)$	$(110,0)$

Cuadro 3.2: Diccionario para cadena 000000000000100000000000

Codificación: 010100110001110001100

Tomando otra palabra, decodificamos 00101011101100100100011010101000011.

Para ello, transformemos el código separando en comas como anteriormente:
$(puntero, bit)|$ $(,0)$ $(0,1)$ $(01,0)$ $(11,1)$ $(011,0)$ $(010,0)$ $(100,0)$ $(110,1)$ $(0101,0)$ $(0001,0)$
Así, yendo por cada uno de los pares ordenados, vamos descubriendo las palabras que resultan en el diccionario de la siguiente tabla.

$s(n)$	palabra
1	0
2	1
3	00
4	001
5	000
6	10
7	0010
8	101
9	0000
10	01

Decodificación: 010000100010001010101000001.

A diferencia de los códigos de bloques, el código de Huffman y la codificación aritmética, la codificación L-Z se define sin necesidad de ningún modelo probabilístico para la fuente, por esta razón se lo conoce como *universal*.

3.3. Aplicaciones de los códigos por bloques.

A continuación se presentan varios ejemplos explicativos y de aplicación de los métodos de codificación por bloques, estos ejemplos que siguen a continuación son tomados de [5] 6. 7.

Ejemplo 3.3.1 Una fuente binaria X emite símbolos iid con distribución de probabilidad $\{f_0, f_1\}$, donde $f_1 = 0.01$. Encontrar un código simbólico óptimo únicamente decodificable para una cadena $\mathbf{x} = x_1x_2x_3$ de tres muestras sucesivas de la fuente. Estimar (a un decimal) el factor por el cual la longitud esperada de este código óptimo es mayor que la entropía de la cadena de tres bits $\mathbf{x}$. Un código aritmético es usado para comprimir una cadena de 1000 muestras de la fuente X. Estimar la media y la desviación estándar de la longitud del archivo comprimido.

Solución. Para obtener un código simbólico óptimo y únicamente decodificable realizamos el procedimiento para el método Huffman y las posibles combinaciones en tres posiciones, el esquema se presenta a continuación:

palabra	p_i
000	0.970299
001	0.009801
010	0.009801
011	0.000099
100	0.009801
101	0.000099
110	0.000099
111	0.000001

Para la realización del método de Huffman, primero ordenamos las palabras en orden descendente de probabilidad.

palabra	p_i
000	0.970299
001	0.009801
010	0.009801
100	0.009801
011	0.000099
101	0.000099
110	0.000099
111	0.000001

Realizamos la codificación de Huffman sumando las probabilidades menores y asignando 0 a la mayor probabilidad y 1 a la menor, reordenamos y realizamos el mismo proceso añadiendo el bit que corresponde a la izquierda. Así obtenemos la codificación siguiente:

palabra	$c(a_i)$	l_i
000	0	1
001	100	3
010	101	3
100	110	3
011	11100	5
101	11101	5
110	11110	5
111	11111	5

La longitud promedio es 1.059998, la entropía es 0.2423794077. Luego el factor por el cual la longitud esperada de este código óptimo es mayor que la entropía de la cadena de tres bits $\mathbf{x}$ es $4,166666667$.

Calculando la entropía de la codificación aritmética $H_2(0.01) \approx 0.08$, obtenemos la longitud media con $N \cdot H_2 \approx 80$.

Ejemplo 3.3.2 Use una modificación del algoritmo Lempel – Ziv, en el cual el diccionario de prefijos es reducido escribiendo nuevos prefijos en el espacio ocupado por los prefijos que no se usarán otra vez. Tales prefijos pueden ser identificados cuando sus dos hijos han sido añadidos al diccionario de prefijos. (Se puede omitir la emisión de terminación de codificación) Use este algoritmo para codificar la cadena 0100001000100010101000001. Subraye los bits que siguen a un prefijo en la segunda ocasión que ese prefijo es usado. (Como se discutió anteriormente estos bits pueden omitirse).

Solución. Codificando la cadena con el método Lempel Ziv tenemos

Subc	λ	0	1	00	001	000	10	0010	101	0000	01
$s(n)$	0	1	2	3	4	5	6	7	8	9	10
$s(n)_2$	000	001	010	011	100	101	110	111	1000	1001	1010
(pt,bit)		$(,0)$	$(0,\mathbf{1})$	$(1,0)$	$(11,1)$	$(011,\mathbf{0})$	$(010,0)$	$(100,0)$	$(110,1)$	$(0101,0)$	$(0001,\mathbf{1})$

La codificación sería 00**1**10111011**0**0100100011010101000001**1**

Ejemplo 3.3.3 Mostrar que el algoritmo Lempel – Ziv modificado, no es completo, esto es, hay códigos binarios que no son codificación de ninguna cadena.

Solución. Al codificar una cadena con el algoritmo, luego de 5 prefijos el puntero puede ser cualquiera de las cadenas 000, 001, 010,011,100, pero no puede ser ninguna 101,110,111. Así existen algunas cadenas binarias que no pueden ser producidas como codificaciones.

Ejemplo 3.3.4 Considerar el texto plano del libro el Quijote de la Mancha. En primera instancia, en texto plano el libro posee 2050 Kb, luego de realizar la compilación del

algoritmo Lempel – Ziv usado en el programa de compresión de gzip obtenemos un archivo comprimido equivalente a 775 Kb. Esto indica una compresión del 62 %.

Ejemplo 3.3.5 Consideremos una distribución Gaussiana N dimensional

$$P(x) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{\sum_n x_n^2}{2\sigma^2}\right)$$

Definimos el radio de un punto $\mathbf{x}$ como $r = (\sum_n x_n^2)^{1/2}$. Estimar la media y la varianza del cuadrado del radio r^2. Asumiendo que N es grande, muestre que cerca de toda la probabilidad de una Gaussiana está contenida en una delgada cáscara de radio $\sqrt{N}\sigma$. Encontrar el ancho de la cáscara. Evaluar la densidad de probabilidad anterior en un punto en la cáscara y en el origen $\mathbf{x} = 0$, compare. Use el caso $N = 1000$ como ejemplo. Note que casi toda la probabilidad de masa esta localizada en una parte diferente del espacio a la región de alta probabilidad.

Solución.

Tomando en cuenta que la varianza para el caso $N = 1$ de x es σ^2, y ya que tenemos N variables aleatorias independientes, entonces tenemos que $E[r^2] = N\sigma^2$. Bajo el mismo argumento, la varianza de r^2 es N veces la varianza de x^2, es decir:

$$var(x^2) = E[x^4] - E^2[x^2] = \int x^2 \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{\sum_n x_n^2}{2\sigma^2}\right) dx - \sigma^4$$

tomando en cuenta que

$$\int x^2 \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{\sum_n x_n^2}{2\sigma^2}\right) dx = 3\sigma^4$$

entonces tenemos $var(x^2) = 3\sigma^4 - \sigma^4$ y por último $var(r^2) = 2N\sigma^2$.

Por el teorema del límite central, tenemos que para N grande, r^2 tiene una distribución con media $N\sigma^2$ y varianza $\sqrt{2N}\sigma^2$.

La anchura de la cáscara la encontramos tornando la desviación estándar de r^2 en la desviación estándar de r, para pequeños $\delta r/r$.

Tenemos que $\delta \log r = \delta r/r = \frac{1}{2}\delta \log r^2 = \frac{1}{2}\frac{\delta(r^2)}{r^2}$, tomando como $\delta(r^2) = \sqrt{2N}\sigma^2$ entonces $\delta r = \frac{1}{2}\frac{r\delta(r^2)}{r^2} = \frac{1}{2}\frac{\sqrt{2N}\sigma^2}{\sqrt{N}\sigma} = \frac{\sqrt{2}\sigma}{2}$

La densidad de probabilidad en un punto de la cáscara $\mathbf{x}_{csc}$ con $r = \sqrt{N}\sigma$ es

$$P(\mathbf{x}_{csc}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N\sigma^2}{2\sigma^2}\right) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N}{2}\right)$$

La densidad en el origen es $P(\mathbf{x} = 0) = \frac{1}{(2\pi\sigma^2)^{N/2}}$, y de esta manera la relación $P(\mathbf{x}_{csc})/P(\mathbf{x} = 0) = \exp(-N/2)$

Ejemplo 3.3.6 Explique qué se entiende por un código simbólico binario óptimo. Encontrar un código binario óptimo para el ensamble

$$\mathcal{A} = \{a, b, c, d, e, f, g, h, i, j\}$$

$$P = \{\frac{1}{100}, \frac{2}{100}\ \frac{4}{100}, \frac{5}{100}, \frac{6}{100}, \frac{8}{100}, \frac{9}{100}, \frac{10}{100}, \frac{25}{100}, \frac{30}{100}\}$$

y calcule la longitud media del código.

Solución. Un código simbólico es óptimo si no existe una codificación con menor longitud media. Para el ejemplo construimos la tabla

palabra	$c(a_i)$	l_i
j	00	2
i	10	2
h	110	3
g	111	3
f	0100	4
e	0110	4
d	0111	4
c	01010	5
b	010110	6
a	010111	6

La longitud media del código es 2.81

Ejemplo 3.3.7 Una cadena $\mathbf{y} = x_1x_2$ la conforman dos muestras independientes de ensamble:

$$\mathbf{X} : \mathcal{A}_\mathbf{X} = \{a, b, c\};\ \mathcal{P}_\mathbf{X} = \left\{\frac{1}{10}, \frac{3}{10}, \frac{6}{10}\right\}$$

Cuál es la entropía de $\mathbf{y}$. Construya un código simbólico binario óptimo para las cadenas $\mathbf{y}$ y encuentre su longitud media

Solución.

palabra	$c(a_i)$	l_i	p_i
cc	1	1	36/100
bc	000	3	18/100
cb	001	3	18/100
bb	0100	4	9/100
ac	0110	4	6/100
ca	0111	4	6/100
ab	01011	5	3/100
ba	010100	6	3/100
aa	010101	6	1/100

Longitud media = 2.67. Entropía = 2.59

Ejemplo 3.3.8 Cadenas de N muestras independientes de un ensamble con $P = \{0.1, 0.9\}$ son comprimidas usando codificación aritmética. Estimar la media y la desviación estándar de las longitudes de las cadenas comprimidas para el caso $N = 1000$

Solución. Sabiendo que la entropía para nuestro caso es $H(0.1) \approx 0.47$ luego la longitud media de la cadena comprimida sería $1000 * H(0.1) \approx 470$

Capítulo 4

Temas de geometría algebraica y su relación con la teoría de la codificación

En este capítulo se contempla una de las más importantes aplicaciones de la geometría algebraica (G.A.) para la resolución de problemas de la codificación. De inicio se presentan resultados de la aritmética de cuerpos y luego se tiene la terminología y definiciones para describir los códigos de corrección de errores. En especial se estudiarán dos tipos especiales: los códigos lineales y los códigos cíclicos.

4.1. Conceptos de la geometría algebraica

Definición 4.1.1

1. Un monomio en una colección de variables $x_1, x_2, \ldots, x_n$ es el producto
$$x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$$
donde α_i es un entero no negativo. Aveces se representa por x^α donde $\alpha = (\alpha_1, \alpha_2, \ldots \alpha_n)$

2. El grado de un monomio x^α es la suma de los exponentes $\alpha_1 + \cdots + \alpha_n$. Notamos el grado del monomio x^α por $|\alpha|$ o $\deg(x)$

3. Sea k un campo. Se llaman polinomios en $x_1, x_2, \ldots, x_n$ al resultado de las combinaciones lineales de monomios en k. Se llama *término* al producto de un elemento no nulo de k y un monomio que aparece en un polinomio. Notamos por $k[x_1, \ldots, x_n]$ al conjunto de todos los polinomios en $x_1, \ldots, x_n$ con coeficientes en k

4. Un polinomio g es homogéneo si todos sus monomios tienen el mismo grado.

Ejemplo 4.1.1 Dado el monomio $x_1^3 x_2^2 x_5$ tenemos que $\alpha = (3, 2, 0, 0, 1)$ y $|\alpha| = 6$.

Sea el campo $\mathbb{Q}$, el polinomio $g = x^3 + 3xy^2 + 3x^2y + y^3$ es un polinomio homogéneo, en cambio $h = x^2 + 2y$ no lo es.

Definición 4.1.2 Sean $f_1, f_2, \ldots, f_s \in k[x_1, x_2, \ldots, x_n]$ definimos

$$\langle f_1, f_2, \ldots, f_s\rangle = \{p_1 f_1 + \cdots + p_s f_s : p_i \in k[x_1, x_2, \ldots, x_n],\ \forall i = 1, \ldots, s\}$$

Ejemplo 4.1.2 Sea $g = x + 1$ y $h = x - 1$ entonces, $f = x^2 - 1 \in \langle g, h\rangle$

Definición 4.1.3 Se dice que $g \in k[x]$ $\deg(g) \geq 1$ es un polinomio irreducible en $k[x]$ si de la descomposición $f(x) = \phi(x)\psi(x)$ con $\phi(x), \psi(x) \in k[x]$ se sigue que: o bien $\deg(\phi) = 0$, $\deg(f) = \deg(\psi)$ o bien $\deg(\psi) = 0$, $\deg(f) = \deg(\phi)$.

Ejemplo 4.1.3 Sea $g = x^2 + 1$. g es irreducible en $\mathbb{R}$, sin embargo es reducible en $\mathbb{C}$

Definición 4.1.4 Sea $I \subset k[x_1, \ldots, x_n]$ un subconjunto no vacío en $k[x_1, x_2, \ldots, x_n]$. I es un ideal si y solo si

1. Si f, $g \in I$ entonces $f + g \in I$.
2. Si $f \in I$ y $p \in k[x_1, x_2, \ldots, x_n]$ entonces $pf \in I$

Proposición 4.1.1 *El conjunto $\langle f_1, f_2, \ldots, f_s\rangle$ es un ideal en el anillo $k[x_1, \ldots, x_n]$*

Demostración.

1. Sean f y g dos elementos de $\langle f_1, f_2, \ldots, f_s\rangle$ entonces $f = p_1 f_1 + \cdots + p_s f_s$ y $g = p'_1 f_1 + \cdots + p'_s f_s$, luego tenemos que:

$$\begin{aligned} f + g &= p_1 f_1 + \cdots + p_s f_s + p'_1 f_1 + \cdots + p'_s f_s \\ &\quad (p_1 + p'_1) f_1 + \cdots + (p_s + p'_s) f_s \in \langle f_1, f_2, \ldots, f_s\rangle \end{aligned}$$

2. Sea $p \in k[x_1, \ldots, x_n]$ y $f \in \langle f_1, f_2, \ldots, f_s\rangle$ de aquí se tiene que $f = p_1 f_1 + \cdots + p_s f_s$ y $pf = (pp_1) f_1 + \cdots + (pp_s) f_s$. Así $pf \in \langle f_1, f_2, \ldots, f_s\rangle$ $\square$

Al ideal $\langle f_1, \ldots, f_s\rangle$ se lo llama el ideal *generado* por $f_1, \ldots, f_s$ debido a la siguiente propiedad.

Proposición 4.1.2 *El ideal $\langle f_1, \ldots, f_s\rangle$ es el menor de los ideales que contiene a $f_1, \ldots, f_s$.*

Demostración.

Mostramos que si J es un ideal que contiene a $f_1, \ldots, f_s$ entonces $\langle f_1, \ldots, f_s\rangle \subset J$. En efecto, ya que J es un ideal tenemos que dados $p_i \in k[x]$ se tiene que $p_i f_i \in J$ para $i = 1, \ldots, s$; por definición de ideal además se tiene que $p_1 f_1 + \cdots + p_s f_s \in J$. Por otro lado $p_1 f_1 + \cdots + p_s f_s \in \langle f_1, \ldots, f_s\rangle$. Concluimos que $\langle f_1, \ldots, f_s\rangle \subset J$.$\square$

Teorema 4.1.1 (Teorema de bases de Hilbert) *Cada ideal I en $k[x_1, \ldots, x_n]$ tiene un conjunto generador finito, es decir existe una colección finita de polinomios $\{f_1, \ldots, f_s\} \subset k[x_1, \ldots, x_n]$ tal que $I = \langle f_1, \ldots, f_s \rangle$.*

Demostración.

Realizamos la demostración para el caso de una variable, esto es, el ideal $I \subset k[x]$ es generado por un polinomio, para el caso general tenemos el resultado de [2] 2 §5 . Sea entonces $I = \langle g \rangle$ para algún $g \in k[x]$. Asumimos que $I \neq \{0\}$ ya que si fuera el caso, no hay nada que probar. Sea g un elemento no nulo de I de grado mínimo. Por el algoritmo de Euclides de la división se tiene que para cualquier $f \in I$, $\deg(f) \geq \deg(g)$ existen $p \in k[x]$ y $r \in k[x]$ tales que $f = pg + r$ y o bien $r = 0$ o $\deg(r) < \deg(g)$. Si $r = 0$ se tiene que $f = pg$ lo cual indica que $f \in I$ y el resultado está mostrado. Si $\deg(r) < \deg(g)$ y además considerando que al despejar $r = f - pg \Rightarrow r \in I$ (debido a que $f \in I$, $pg \in I$ y además I es un ideal) concluimos que $r = 0$ ya que g es de grado mínimo en I. Si esto no fuera así g no sería minimal y contradecimos a lo asumido sobre g. Luego para este caso también se tiene que $f \in I$. Hemos mostrado que cualquier elemento $f \in I$ es divisible para g lo cual muestra a su vez que I es generado por un único elemento g.□

Definición 4.1.5 Sea I un ideal y $f \in k[x]$ se define la clase $[f]$ como:

$$[f] = f + I = \{f + h : h \in I\}$$

con la propiedad de clases:

$$[f] = [g] \iff f - g \in I$$

Se denomina anillo cociente a $k[x]/I = \{[f] : f \in k[x]\}$

Proposición 4.1.3 *Sea k un campo y sea $g \in k[x]$ un polinomio irreducible en k. El ideal $\langle g \rangle \subset k[x]$ es un ideal maximal. Además es $k[x]/\langle g \rangle$ es un campo si g es irreducible.*

Demostración.

1. Mostramos que el ideal $I = \langle g \rangle$ es maximal, esto es, mostramos que no existe otro ideal J que satisface $I \subset J \subset k[x]$ excepto $J = I$ y $J = k[x]$. Supongamos que existe un ideal J de $k[x]$ tal que $I \subset J$. Luego por el teorema 4.1.1 tenemos que $J = \langle f \rangle$ para algún $f \in k[x]$. Ya que $g \in J$ tenemos que $g = fh$ para algún $h \in k[x]$. Debido a que g es irreducible tenemos que o bien f tiene grado cero o bien h tiene grado cero.

Si f tiene grado cero entonces f es la unidad en $k[x]$ lo cual indica que $J = k[x]$. Si h tiene grado cero entonces $h \in k \setminus \{0\}$ entonces $g = \alpha f$ para algún $\alpha \in k \setminus \{0\}$ y $f = \alpha^{-1} g \in I$, luego $J = I$.

2. Del punto anterior tenemos que $\langle g \rangle$ es maximal. Tomando el teorema del álgebra abstracta que menciona: sea K un anillo conmutativo con identidad y J un ideal maximal de K entonces el anillo cociente K/J es un campo, tenemos que todas las hipótesis son válidas para nuestro caso. Concluimos que $k[x]/\langle g \rangle$ es un campo. □

4.2. Relaciones entre la G.A. y la teoría de la codificación

Se estudia un tipo específico de códigos en el cual la información que va a ser codificada está formada por *palabras* de una longitud fija k usando símbolos de un alfabeto fijo. Los mensajes codificados son divididos en cadenas llamadas *palabras código* con una longitud fija de bloque n y usando símbolos del mismo alfabeto. Para detectar o corregir errores en el envío de información es necesaria cierta *redundancia* lo cual indica que las palabras código en el proceso de codificación serán más largas que las palabras, es decir $n > k$.

Se usará como alfabeto fijo al conjunto $\{0, 1\}$ y lo identificaremos con el campo finito $\mathbb{F}_2$. En general como alfabeto usamos $\mathbb{F}_q$, donde q es el orden del campo finito. Denotamos también $\mathbb{F}_q^n$ el conjunto de las n – úplas de elementos de $\mathbb{F}_q$.

Formalmente, el proceso de codificación de un mensaje es una función inyectiva:

$$E : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$$

El código entonces es $C = E(\mathbb{F}_q^k) \subset \mathbb{F}_q^n$. La operación de decodificación puede verse como una función:

$$D : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^k$$

tal que $D \circ E$ es la identidad en $\mathbb{F}_q^k$.

Así pues, un código C sobre $\mathbb{F}_q$ es un subconjunto no vacío de $\mathbb{F}_q^n$. El número $n = n(C)$ se llama la longitud del código C. Cualquier elemento de C se llama una

palabra código de C. Adicionalmente se tiene que el radio de información de un código C se define como:

$$R(C) = \frac{\log_q |C|}{n(C)} \tag{4.1}$$

Uno de los aspectos importantes de la teoría de la codificación es medir cuán eficiente es un código: a mayor radio de información la redundancia será menor.

Ejemplo 4.2.1 Sea $C = \{(0,0,0),(1,1,1)\} \subseteq \mathbb{F}_2^3$. Codificamos una cadena de bits por medio de la función $0 \longmapsto (0,0,0)$ y $1 \longmapsto (1,1,1)$. El radio de información de C es $\frac{\log_2(2)}{3} = \frac{1}{3}$, que indica que cada bit de cada palabra código de C lleva $\frac{1}{3}$ de información actual. El código C permite corregir un error en un bloque de 3 bits, es decir, si queremos enviar el mensaje 1 entonces lo codificamos como $(1,1,1)$ antes de la transmisión, digamos que al fin del canal de comunicación se recibe $(1,0,1)$ y si asumimos que máximo un error es cometido durante la transmisión, entonces la palabra código no puede ser $(0,0,0)$. Así el único posible resultado es $(1,1,1)$ que se decodifica como 1.

Definición 4.2.1 Sea $\mathbf{x} \in \mathbb{F}_q^n$ el peso $\omega(\mathbf{x})$ es el número de coordenadas no nulas de $\mathbf{x}$.

Definición 4.2.2 Sea $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ se define la distancia de Hamming como

$$d(\mathbf{x}, \mathbf{y}) = \omega(\mathbf{x} - \mathbf{y}) \tag{4.2}$$

Es decir la distancia (de Hamming) entre dos elementos de $\mathbb{F}_q^n$ es el número de coordenadas en que difieren. Se verifica que esta es una distancia en el sentido del análisis, y tenemos como definición de bolas:

$$B_r(x) = \{y \in \mathbb{F}_q^n : d(x,y) \leq r\} \tag{4.3}$$

En otras palabras, $B_r(x)$ es el conjunto de elementos y que difieren de x en a lo más r componentes.

Definición 4.2.3 La distancia de un código C se define por

$$d = \text{mín}\{d(\mathbf{x}, \mathbf{y}) : \mathbf{x} \neq \mathbf{y} \, \forall \mathbf{x}, \mathbf{y} \in C\} \tag{4.4}$$

La distancia de Hamming nos da una manera útil para medir la capacidad de corregir errores de los códigos. Supongamos que cada par de palabras código distintas $x \in C$, $y \in C$, $C \subset \mathbb{F}_q^n$ satisface $d(x,y) \geq d$ para algún $d \geq 1$. Si al transmitir x se producen errores, podemos ver la palabra recibida como $w = x + e$ para algún vector de error e. Si $\omega(e) = d(x,w) \leq d-1$ entonces w no es otra palabra código. De esta manera cualquier vector de error con peso a lo más $d-1$ puede ser detectado. Más

aún si $d \geq 2t+1$ para algún $t \geq 1$ entonces para cualquier $w \in \mathbb{F}_q^n$, por la desigualdad triangular $d(x,w)+d(w,y) \geq d(x,y) \geq 2t+1$. De aquí se sigue que, o bien $d(x,w) > t$ o bien $d(y,w) > t$. Luego $B_t(x) \cap B_t(y) = \emptyset$. Así la única palabra código en $B_t(x)$ es x. Resumiendo, si un vector de error con un peso a lo sumo t se introduce en la transmisión de una palabra código, éste puede ser corregido mediante la función decodificación del vecino más cercano:

$$D(x) = E^{-1}(c) \text{ donde } c \in C \text{ minimiza } d(x,c)$$

Así hemos mostrado el siguiente resultado.

Proposición 4.2.1 *Sea C un código con distancia d. Todos los errores de peso menor o igual que $d-1$ pueden ser detectados. Más aún si $d \geq 2t+1$, entonces todo los vectores de error posibles de peso menor que t pueden ser corregidos por la decodificación del vecino más cercano.*

4.3. Códigos lineales

Una clase especial de códigos, que trataremos a continuación es útil debido a que su estructura es muy conveniente tanto para la codificación como para la decodificación, estos son los códigos lineales.

Definición 4.3.1 Se llaman códigos lineales a aquellos en los que el conjunto de palabras código $C \subset \mathbb{F}_q^n$ forma un subespacio vectorial de $\mathbb{F}_q^n$ de dimensión k.

Tenemos un resultado muy importante de los códigos lineales.

Proposición 4.3.1 *La distancia de los códigos lineales es el mínimo peso de cualquier palabra código no nula.*

Demostración.

Sean $\mathbf{x}$ y $\mathbf{y}$ dos elementos de C distintos. Ya que C es un s.e.v. tenemos que $\mathbf{x}-\mathbf{y} \neq \mathbf{0}$, además, sea $\mathbf{z} = \mathbf{x}-\mathbf{y}$ entonces por el mismo argumento $\mathbf{z} \in C$ y $\mathbf{z} \neq \mathbf{0}$. Por definición $d(\mathbf{x},\mathbf{y}) = \omega(\mathbf{x}-\mathbf{y}) = \omega(\mathbf{z})$. Así tenemos que $d = \text{mín}\{\mathbf{z} \in C : \mathbf{z} \neq \mathbf{0}\}$ □

En el caso de los códigos lineales, como transformación $E : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ usamos una transformación lineal cuya imagen es un subespacio C de $\mathbb{F}_q^n$. La matriz de E con respecto a las bases canónicas en el dominio y la imagen es llamada la matriz *generadora* correspondiente a E.

Sea C un código lineal de longitud n y dimensión k. Si G es una matriz generadora para C y u es una palabra de longitud k (como vector fila) entonces $v = uG$ es una palabra código en C ya que v es una combinación lineal de las filas de G las cuales forman una base en C. Adicionalmente, s i $u_1G = u_2G$ entonces se sigue que $u_1 = u_2$, ya que cada palabra en C tiene una única combinación lineal en un base. Así, ninguna palabra código $v = uG$ se obtiene de más de un único u en $\mathbb{F}_2^k$.

Lo anterior muestra que los mensajes que pueden ser codificados por un código lineal (simbolizado (n, k, d) código lineal de longitud n dimensión k y distancia d) son todos los mensajes $u \in \mathbb{F}_2^k$. El mensaje u es codificado mediante $v = uG$, luego en cualquier palabra código solo k bits son usados para transportar el mensaje. Entonces tenemos que el radio de información para (n, k, d) es $\frac{\log_2(2^k)}{n} = \frac{k}{n}$.

Describimos otra matriz asociada con los códigos lineales.

Definición 4.3.2 Una matriz H se llama matriz de comprobación de paridad para el código C si las columnas de H forman una base para el espacio dual $C^{\perp}$.

Si C tiene longitud n y dimensión k debido a que la suma de las dimensiones de los espacios C y $C^{\perp}$ es n entonces cualquier matriz de comprobación de paridad tiene n filas, $n - k$ columnas y rango $n - k$. Se tiene además el siguiente resultado el cual lo mencionamos sin demostración.

Proposición 4.3.2 *Si H es una matriz de comprobación de paridad para un código lineal de longitud n, entonces C esta formado por todas las palabras $v \in \mathbb{F}_2^n$ tal que $vH = 0$.*

Definición 4.3.3 Sea C un código con longitud n y u cualquier palabra de longitud n definimos la clase de C con respecto a u al conjunto $C + u = \{v + u : v \in C\}$.

Las clases junto con la matriz de comprobación de paridad son de mucha utilidad al momento de la decodificación de los códigos lineales. En efecto, sea C un código lineal, asumimos que la palabra código $v \in C$ es transmitida y que es recibido w, obtenemos como resultado que el patrón de error es $e = v + w$ o bien $w + e = v \in C$, luego el patrón de error e y la palabra recibida están en la misma clase de C. Debido a que los patrones de error de menor peso son los más probables ([5] 1.2), el proceso para la decodificación de códigos lineales sería: Luego de recibir la palabra w escogemos la palabra u de menor peso en la clase $C + w$ (la cual contiene a w) y concluimos que $v = w + u$.

Podemos usar la matriz de comprobación de paridad para realizar el último procedimiento. Sea H la matriz de comprobación de paridad para C. Para cualquier $w \in \mathbb{F}_2^n$ definimos el *síndrome* de w como la palabra $wH \in \mathbb{F}_2^{n-k}$.

Proposición 4.3.3 *Sea C un código lineal de longitud n y H una matriz de comprobación de paridad de C. Sean $w, u \in \mathbb{F}_2^n$:*

1. *$wH = 0$ ssi $w \in C$*
2. *$wH = uH$ ssi w y u están en la misma clase de C*
3. *Si u es un patrón de error en una palabra w entonces uH es la suma de las filas de H que corresponden a las posiciones en donde ocurrieron errores en la transmisión.*

Demostración.

1. Se sigue del hecho que $C = (C^\perp)^\perp$ y de que H es la matriz del espacio dual $C^\perp$
2. $w + C = v + C \iff w - v \in C \overset{1.}{\iff} (w - v)H = 0 \iff wH = vH$
3. Ya que u y w se encuentran en la misma clase de C, y u es de menor peso en la clase, entonces posee los valores que se cambiaron en la transmisión, concluimos que uH es la suma de los lugares donde hubo errores.

Proposición 4.3.4 *Sea $G = (I_k \,|\, P)$ una matriz generadora de un código lineal C, donde I_k es la matriz identidad $k \times k$ y P una matriz $k \times n - k$. Entonces $H = \binom{-P}{I_{n-k}}$ es una matriz de comprobación de paridad de C.*

Demostración

Si tomamos la multiplicación $GH = (I_k \, P)\binom{-P}{I_{n-k}} = \mathbf{0}_{k \times n-k}$ se tiene el resultado. □

4.4. Códigos cíclicos

Definición 4.4.1 Un código lineal $C \in \mathbb{F}_q^n$ es cíclico si es cerrado con respecto a una permutación cíclica, esto es si el desplazamiento cíclico de una palabra código es un palabra código.

Usando el isomorfismo que existe entre $\mathbb{F}_q^n$ y el espacio vectorial de los polinomios de grado $n - 1$ con coeficientes en $\mathbb{F}_q$:

$$\phi : (a_0, a_1, \ldots, a_{n-1}) \longleftrightarrow a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

podemos identificar a un código cíclico con el conjunto de polinomios de grado $n-1$. La permutación hacia la derecha envía a $(a_0, a_1, \ldots, a_{n-1})$ hacia $(a_{n-1}, a_0, \ldots, a_{n-2})$ es lo mismo que multiplicar el polinomio $p(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ por x y luego tomar el residuo de la división para $x^n - 1$. En efecto, al realizar $\frac{xp(x)}{x^n-1}$ obtenemos que

$$\begin{aligned} xp(x) &= a_0x + a_1x^2 + \cdots + a_{n-1}x^n \\ &= a_{n-1}(x^n - 1) + a_{n-1} + a_0x + a_1x^2 + \cdots + a_{n-3}x^{n-2} + a_{n-2}x^{n-1} \end{aligned}$$

y vemos que la relación que existe entre polinomios y códigos es:

$$a_{n-1} + a_0x + a_1x^2 + \cdots + a_{n-3}x^{n-2} + a_{n-2}x^{n-1} \longleftrightarrow (a_{n-1}, a_0, a_1, \ldots, a_{n-3}, a_{n-2})$$

Esto indica que para trabajar con códigos cíclicos consideraremos los polinomios de grado $n-1$ como los elementos del anillo cociente $R = \mathbb{F}_q^n / \langle x^n - 1 \rangle$. Más aun, mostramos lo siguiente.

Proposición 4.4.1 *Sea $C \subset \mathbb{F}_q^n$ un código lineal. C es cíclico si y solo si $\phi(C)$ es un ideal en $F_q[x]/\langle x^n - 1\rangle$.*

Demostración.

$\phi(C)$ es un ideal en $\mathbb{F}_q[x]/\langle x^n - 1\rangle$ siempre que $p(x) \in \mathbb{F}_q[x]/\langle x^n - 1\rangle$ y $c \in C$ entonces $c(x)p(x) \in \phi(C)$ donde $c(x) = \phi(c)$. Ya que ϕ es un isomorfismo lo anterior es equivalente a que $xc(x) \in \phi(C)$. Pero $xc(x) = c_{n-1} + c_0x + \cdots + c_{n-2}x^{n-1} \mod (x^n - 1)$. □

Teorema 4.4.1 *Sea C un ideal de $\mathbb{F}_q[x]/\langle x^n - 1\rangle$. Entonces*

1. *Existe un único polinomio mónico $g(x)$ de grado mínimo r en C y $g(x)|x^n - 1$.*
2. *$C = \langle g(x)\rangle = \{r(x)g(x) : \deg(r(x)) < n - r\}$*
3. *$\dim C = n - r$ además $B = \{g(x), xg(x), \ldots, x^{n-r-1}g(x)\}$ es una base de C*
4. *Si $g(x) = g_0 + g_1x + \cdots + g_rx^r$ entonces $g_0 \neq 0$ y C tiene como matriz generadora:*

$$G = \begin{pmatrix} g_0 & g_1 & g_2 & \ldots & \ldots & g_r & 0 & 0 & \ldots & 0 \\ 0 & g_0 & g_1 & g_2 & \ldots & \ldots & g_r & 0 & \ldots & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \ldots & \ldots & g_r & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \ldots & 0 & g_0 & g_1 & g_2 & \ldots & \ldots & g_r \end{pmatrix} \in \mathbb{F}_q^{(n-r)\times n}$$

Demostración.

1. Supongamos que existen dos polinomios mónicos $g_1 \neq g_2$ de C que tienen grado mínimo r. Entonces $g_1 - g_2$ es un polinomio no nulo con grado menor que r lo cual es una contradicción, por tanto existe un único polinomio mónico g de grado mínimo en C. Mostramos que g es tal que $g|x^n - 1$. Por el algoritmo de Euclides tenemos que existen q y r tales que $x^n - 1 = qg + r$ con $\deg(r) < \deg(g)$ ó $r = 0$. Si r =0, no hay nada que probar. Si $\deg(r) < \deg(g)$, tenemos que debido a que $x^n - 1 \mod x^n - 1 = 0$ entonces tenemos que $x^n - 1$ es 0 en $\mathbb{F}_q[x]/\langle x^n - 1\rangle$ entonces $r = -qg \in C$ luego $r = 0$, caso contrario contradice minimalidad de grado de g.

2. La primera igualdad viene del teorema de bases de Hilbert (Teorema 4.1.1), mostramos la segunda igualdad. Por el punto anterior tenemos que $x^n - 1 = qg$ para algún q tal que $\deg(q) = n - r$. Sea f un polinomio entonces dividiendo para q tenemos que $f = hq + r$ donde o bien $r = 0$ o bien $\deg(r) < \deg(q)$. Entonces se tiene:

$$fg = hqg + rg = h(x^n - 1) + rg$$

   Pero $h(x^n - 1) = 0$ en $\mathbb{F}_q[x]/\langle x^n - 1\rangle$ luego tenemos $fg = rg$. Por último $\deg f < \deg q = n - r$.

3. Por el punto anterior, se tiene que $B = \{g(x), xg(x), \ldots, x^{n-r-1}g(x)\}$ genera a C y debido a que es un conjunto linealmente independiente se tiene el resultado $\dim C = n - r$.

4. Supongamos que $g_0 = 0$ entonces $g(x) = xp_1(x)$ donde $\deg p_1(x) < r$. Luego tendríamos que $p_1(x) = 1 \cdot p_1(x) = x^n \cdot p_1(x)$ (lo último debido a que $x^n \mod (x^n - 1) = 1$). Así $p_1(x) = x^{n-1}g(x) \in C$ lo cual no puede ser, debido a que $p_1(x) \neq 0$ y tiene menor grado que $g(x)$ minimal. La matriz G es generadora de C debido a que por el punto anterior se tiene que B es una base de C. □

Al polinomio del teorema anterior se lo llama *polinomio generador* de C. Sin embargo C puede ser generado por otro polinomio, por ello escribimos $\langle\langle p(x)\rangle\rangle$ para notar que C es un ideal generado por $p(x)$ y además que $p(x)$ es el polinomio generador de C.

Proposición 4.4.2 *Un polinomio mónico $p(x) \in \mathbb{F}_q^n / \langle x^n - 1 \rangle$ es el polinomio generador de un código cíclico ssi $p(x)|x^n - 1$.*

Demostración.

Supongamos que $p(x)|x^n - 1$ y que $g(x)$ es el polinomio generador de $C = \langle p(x) \rangle$ además que $p(x) \neq g(x)$. Debido que aque $g(x)$ es mónico y por el teorema anterior tenemos que $\deg g(x) < \deg p(x)$. Tenemos que $x^n - 1 = p(x)f(x)$ para algún $f(x) \neq 0$ además $g(x) \in C$ esto indica que existe $q(x) \in \mathbb{F}_q^n / \langle x^n - 1 \rangle$ tal que $g(x) = q(x)p(x)$. Por tanto $g(x)f(x) = q(x)p(x)f(x) = g(x)(x^n - 1) = 0$. Pero $\deg g(x)f(x) < \deg p(x)f(x) = n$ lo cual indica que $g(x)f(x) = 0$ que es una contradicción ($f(x), g(x) \neq 0$). Conclusión: $p(x) = g(x)$. $\square$

Definición 4.4.2 Definimos los conjuntos:

$$D_n = \{g(x) \in \mathbb{F}_q[x] : g(x)|(x^n - 1),\ g(x) \text{ mónico}\}. \tag{4.5}$$

$$I_n = \{I \subset \mathbb{F}_q^n / \langle x^n - 1 \rangle : \text{I es ideal}\} = \{C \subset \mathbb{F}_q^n :\ \text{C es un código cíclico}\} \tag{4.6}$$

El teorema 4.4.1 y la proposición 4.4.2 muestran que la aplicación:

$$\Psi : D_n \longrightarrow I_n,\ g(x) \longmapsto \langle\langle g(x) \rangle\rangle \tag{4.7}$$

es una correspondencia uno a uno entre D_n y I_n.

Ejemplo 4.4.1 Calculamos todos los códigos cíclicos de $\mathbb{F}_2^7$. Para ello factorizamos $x^7 - 1$ en $\mathbb{F}_2$. $x^7 - 1 = (x - 1)(x^6 + x^5 + x^4 + x^3 + x^2 + x + 1)$. Además tenemos que

$$\begin{aligned} (x^3 + x^2 + 1)(x^3 + x + 1) &= x^6 + x^4 + x^3 + x^5 + x^3 + x^2 + x^3 + x + 1 \\ &= x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \end{aligned}$$

Luego $x^7 - 1 = (x - 1)(x^3 + x^2 + 1)(x^3 + x + 1)$ y tenemos un producto de polinomios irreducibles en $\mathbb{F}_2$. De aquí podemos decir que como existen 3 factores de $x^7 - 1$ existen $2^3 = 8$ polinomios generadores, y por tanto tenemos 8 códigos cíclicos distintos correspondientes a los siguientes polinomios:

Ideal	Polinomio generador
$C_1 = \langle\langle g_0(x)\rangle\rangle$	1
$C_2 = \langle\langle g_1(x)\rangle\rangle$	$x-1$
$C_3 = \langle\langle g_2(x)\rangle\rangle$	x^3+x+1
$C_4 = \langle\langle g_3(x)\rangle\rangle$	x^3+x^2+1
$C_5 = \langle\langle g_{1,2}(x)\rangle\rangle$	$x^4+x^3+x^2+1$
$C_6 = \langle\langle g_{1,3}(x)\rangle\rangle$	x^4+x^2+x+1
$C_7 = \langle\langle g_{2,3}(x)\rangle\rangle$	$x^6+x^5+x^4+x^3+x^2+x+1$
$C_8 = \langle\langle g_{1,2,3}(x)\rangle\rangle$	x^7-1

Cuadro 4.1: Códigos cíclicos para $\mathbb{F}_2^7$

Tomemos el caso $C_1 = \langle\langle g_0(x)\rangle\rangle$ ya que $g_0 = 1 \longleftrightarrow (1,0,0,0,0,0,0)$ se tiene que la matriz generadora $G = \mathrm{Id}_{7\times 7}$ es la matriz identidad de orden 7.

Veamos el caso de $C_4 = \langle\langle g_3(x)\rangle\rangle$. Para ello debemos calcular todos los elemento que corresponden a las clases de $\mathbb{F}_2^7/\langle g_3(x)\rangle$. En la primera columna de la tabla 4.2 tenemos todos los polinomios $p(x)$ tales que $\deg p(x) \leq 3$ en la segunda columna sus respectivos productos por $g_3(x)$.

$p(x)$	$p(x)(x^3+x^2+1)$	Código
0	0	0000000
1	x^3+x^2+1	1011000
x	x^4+x^3+x	0101100
x^2	$x^5+x^4+x^2$	0010110
x^3	$x^6+x^5+x^3$	0001011
$x+1$	x^4+x^2+x+1	1110100
x^2+1	$x^5+x^4+x^3+1$	1001110
x^3+1	$x^6+x^5+x^2+1$	1010011
x^2+x	$x^5+x^3+x^2+x$	0111010
x^3+x	$x^6+x^5+x^4+x$	0100111
x^3+x^2	$x^6+x^4+x^3+x^2$	0011101
x^2+x+1	x^5+x+1	1100010
x^3+x+1	$x^6+x^5+x^4+x^3+x^2+x+1$	1111111
x^3+x^2+1	x^6+x^4+1	1000101
x^3+x^2+x	x^6+x^2+x	0110001
x^3+x^2+x+1	x^6+x^3+x+1	1101001

Cuadro 4.2: Códigos C_4

4.4.1. Codificación y decodificación

Sea C un código cíclico en $\mathbb{F}_n^q/\langle g(x)\rangle$ con $\deg g(x) = r$. Dado el mensaje $m_0m_1\ldots m_{k-1} \in \mathbb{F}_q^k$ el polinomio mensaje es $m(x) = m_0 + \cdots + m_{k-1}x^{k-1}$, procedemos a codificar $m(x)$.

Sea $m^*(x) = x^r m(x) = m_0 x^r + m_1 x^{r+1} \cdots + m_{k-1} x^{n-1}$ ya que $r + k = n$, claramente $\deg m^*(x) \leq n - 1$ y los r primeros términos son nulos. Dividiendo $m^*(x)/g(x)$ obtenemos $m^*(x) = p(x)g(x) + r(x)$ con $\deg r(x) < \deg g(x) = r$ ó $r(x) = 0$. Definimos por $c(x)$ a $c(x) = m^*(x) - r(x)$ y observamos que debido a que $p(x)g(x) \in C$ entonces $c(x) \in C$. Codificamos de la siguiente manera

$$m(x) \longrightarrow c(x) = x^r m(x) - [x^r m(x)] \quad \text{mód } g(x)$$

Debido que $m^*(x)$ tiene términos de grado mayores o iguales que r y que $r(x)$ tiene términos de grado menores que r vemos que se obtiene

$$c = (-r_0, -r_1, \ldots, -r_{r-1}, m_0, m_1, \ldots, m_{k-1})$$

y el mensaje inicial aparece en los últimos k lugares (bits).

Ejemplo 4.4.2 Tomamos el código cíclico C_4, generado por $g(x) = x^3 + x^2 + 1$ codificamos el mensaje $m = 0110 \leftrightarrow x + x^2$. Para nuestro caso $n = 7$, $r = 3$. Entonces $m^*(x) = x^3(x+x^2) = x^5+x^4$, dividiendo para $g(x)$ tenemos $m^*(x) = x^2(x^3+x^2+1)+x^2$ y tenemos $c(x) = x^2(x^3 + x^2 + 1) = x^5 + x^4 + x^2$. Teniendo entonces:

$$0110 \longmapsto 001\mathbf{0110}$$

Para el proceso de decodificación, procedemos de manera similar que en los códigos lineales (todo código cíclico es lineal) pero en forma polinómica. Sea $c \in C$ y $c(x)$ su correspondiente polinomio, enviamos $c(x)$. Sea $u(x)$ es el polinomio recibido entonces el polinomio de error es $e(x) = u(x) - c(x)$. Definimos el síndrome para este caso.

Definición 4.4.3 Sea C un código cíclico generado por $g(x)$ con distancia d. El síndrome del polinomio $u(x)$ denotado $\text{syn}(u(x))$ es el resto de la división de $u(x)$ para el $g(x)$. Luego $u(x) = p(x)g(x) + \text{syn}(u(x))$ con $\deg(\text{syn}(u(x))) < \deg g(x)$ ó $\text{syn}(u(x)) = 0$.

Es claro que si $u(x) \in C$ entonces $\text{syn}(u(x)) = 0$ y que $\text{syn}(u(x)) = \text{syn}(v(x)))$ ssi tanto $u(x)$ como $v(x)$ están en la misma clase de C. Supongamos que $u(x) = c(x)+e(x)$ y que el polinomio error tiene un peso $\omega(e(x)) \leq t = \left\lfloor \frac{d-1}{2} \right\rfloor$. Entonces la decodificación sería $c(x) = u(x) - e(x)$.

Para encontrar el error, vemos que $e(x)$ y $c(x)$ están en la misma clase de C. Es decir $\text{syn}(u(x)) = \text{syn}(e(x))$. Calculamos entonces el síndrome de $u(x)$ y tomamos como $e(x)$ el polinomio de menor peso de la clase para $u(x)$ de C. A este polinomio se lo

llama el *líder* de la clase. De esta manera corregimos los errores de peso menor o igual que $\left\lfloor \frac{d-1}{2} \right\rfloor$

Ejemplo 4.4.3 Tomando el código C_4 tenemos que su distancia es $d = 3$ (el menor peso posible en todas sus palabras código), ayudándonos de lo anterior y la proposición 7.4 tenemos que C_4 corrige todos los errores de peso 1. Los líderes de clase son:

$e(x)$	$\text{syn}(e(x))$
0	0
1	1
x	x
x^2	x^2
x^3	x^2+1
x^4	x^2+x+1
x^5	$x+1$
x^6	x^2+x

Cuadro 4.3: Líderes de clase C_4

Supongamos que recibimos $u(x) = x^6 + x + 1$ entonces dividiendo para $x^3 + x^2 + 1$:

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

Entonces $\text{syn}(u(x)) = x^2 + x + 1$ que corresponde a $e(x) = x^4$. Decodificando tenemos $x^6 + x + 1 - x^4 = x^6 + x^4 + x + 1$. Es decir, recibimos $1100001 \notin C$ con al menos un error y decodificamos $1100101 \in C$.

Conclusiones

En los temas tratados en el presente documento se ha pretendido incursionar en los fundamentos de los procesos de compresión de información y codificación óptima, además de ello de manera sistemática hemos llegado a evidenciar la relación que existe entre la teoría y la práctica (geometría algebraica y teoría de la información), es así que se mencionan a continuación las siguientes conclusiones de la presente investigación.

- Los resultados que se han obtenido del presente trabajo están orientados al comprensión de los métodos que se utilizan en la compresión de la información, así como también se ha logrado identificar cuan amplia es la relación que existe entre los resultados del álgebra abstracta y la codificación de información.

- La codificación por bloques nos permite convertir la información en una cadena binaria representada por un número entre 0 y 1 el cual, tiene la propiedad de ser un código prefijo y por tanto únicamente decodificable.

- Existe una ventaja significativa en la codificación SFE (y su caso particular: la codificación aritmética) con respecto a la codificación de Huffman, esta es que en la primera se pueden plantear modelos matemáticos (Modelo de Laplace o de Dirichlet) para el cálculo iterativo de las probabilidades de la fuente de emisión de información, en la segunda exigidamente se requiere calcular *todas* las probabilidades de cada símbolo para realizar la codificación.

- Sin embargo, la codificación SFE es una codificación subóptima, en efecto, su longitud media sobrepasa en dos bits la entropía (Proposición 2.1.2)

- Si incrementamos la longitud de los bloques, la longitud promedio tiende a la entropía del código, por ello, hemos llamado a la codificación SFE, asintóticamente óptima.

- La codificación de Lempel Ziv, no requiere de cálculo de probabilidades como en el caso de SFE o Huffman, en este tipo de codificación se construyen listas a partir de la información que llega al receptor, por ello es que se llama *universal* a este tipo de codificación. Al igual que SFE esta codificación es asintóticamente óptima.

- El algoritmo implementado en la sección 2, corresponde a una aplicación práctica del problema de la codificación aritmética, sin embargo esta sujeta a restricciones de aritmética computacional al momento de calcular las probabilidades, existen implementaciones alternativas con el objeto de extender la codificación a datos mucho más largos en [10].

- Las implementaciones de la codificación de Lempel Ziv en su mayoría son patentadas, sin embargo existen varios programas que permiten comprimir información usando este algoritmo.

- Varios conceptos y resultados de la geometría algebraica son utilizados en la teoría de la información para la representación de los códigos lineales y cíclicos.

- A partir de la definición 4.4.2, podemos concluir que existe una relación fundamental entre el álgebra y la codificación, se interpreta a los coeficientes de los polinomios en campos finitos como las palabras codificadas mediante codificación lineal. Además, los códigos cíclicos al ser un caso particular de códigos lineales, han aparecido también como una interpretación del concepto algebraico de permutación.

Bibliografía

[1] Cox D., Little J., O'Shea D. Using Algebraic Geometry. Springer, 2005.

[2] Cox D., Little J., O'Shea D. Ideals, Varieties, and Algorithms. Springer, 2007.

[3] Fionov A.N. An efficient method for the randomization of messages based on arithmetic coding.

[4] W. C. Huffman, V. Pless, Fundamentals of Error-Correcting Codes. Cambridge University Press, (2003).

[5] MacKay D. Information Theory, Inference, and Learning Algorithms. Cambridge University Press.Version 7.0 (third printing) August 25, 2004.

[6] Milne J.S. Algebraic Geometry. September. www.jmilne.org/math/ 2009.

[7] Niederreiter H, Xinga C. Algebraic Geometry in Coding Theory and Cryptography. Princenton University Press, 2009.

[8] Potapov B.N. Information Theory. Coding discrete probabilistic sources. Novosibirsk. Izd. centr. NGU,1999.

[9] Ryabko, B. Ya.; Fionov, A. N. An efficient method for adaptive arithmetic coding of sources with large alphabets. Problems of Information Transmission,v. 35 (1999).

[10] Shanon C.E. A Mathematical Theory of Communication. The Bell System Technical Journal,Vol. 27, pp. 379-423, 623-656, July, October, 1948.

[11] Witten I.H., Neal R. M., Cleary J.G. Arithmetic coding for data compression. Commun. ACM. 1987