Universidad de la República

Proyecto de Grado de Ingeniería en Computación

Codificación de señales multicanal

Autor:
Pablo Cerveñansky

Supervisores:
Álvaro Martín
Gadiel Seroussi

Núcleo de Teoría de la Información
Facultad de Ingeniería

13 de diciembre de 2015

Resumen

En este proyecto estudiamos la codificación de señales multicanal. Consideramos una señal (de audio, una imagen, etc.), compuesta por n canales, sobre los que aplicamos funciones que reducen la cantidad de canales a m ($m < n$). En general existe correlación entre los canales de la señal original, y a su vez las funciones aplicadas al reducir los canales también inducen correlación entre los canales de la señal transformada. Como primer problema de estudio nos planteamos determinar si un algoritmo de compresión en el que el codificador conoce la señal original y las funciones de transformación puede sacar provecho de esta información para obtener mejores niveles de compresión de los que se pueden lograr con otro algoritmo cuyo codificador solamente tiene acceso a la señal transformada. Llegamos a la conclusión de que la información adicional que conoce el codificador no le sirve al primer algoritmo para comprimir la señal transformada con menos bits que el segundo algoritmo. Desde un punto de vista práctico, diseñamos e implementamos dos codificadores, llamados GolombBN y T, para la codificación de variables aleatorias con distribución binomial negativa. Esta distribución surge al sumar dos distribuciones geométricas con igual parámetro. La distribución geométrica es importante ya que aparece naturalmente para modelar variables que se codifican en compresores de, por ejemplo, audio e imágenes. El codificador GolombBN es una versión levemente modificada del codificador de Golomb, y su principal ventaja es que tiene una baja complejidad computacional, lo que le permite codificar con tiempos de ejecución muy cortos. La desventaja es que los largos medios de código obtenidos no están cerca del óptimo. El codificador T, en cambio, es bastante más complejo y requiere mayor tiempo de procesamiento, pero los largos medios de código obtenidos son significativamente menores. Por los datos experimentales obtenidos, pensamos que es o está muy cerca de ser un codificador óptimo para la distribución binomial negativa.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos 20 años hemos sido testigos de un vertiginoso desarrollo de Internet y de las comunicaciones móviles. Ambos factores han contribuido a una expansión acelerada del universo digital. Actualmente se estima que la cantidad de información digital que se crea y se copia anualmente se duplica cada dos años. De acuerdo a un informe [1], en 2013 se crearon y copiaron 4.4 ZB[1] de datos digitales, y se espera que esta cifra se eleve a 44 ZB en 2020. En este escenario, el estudio de la compresión de datos cobra cada vez más relevancia.

La compresión de datos permite reducir el número de bits utilizados para representar determinada información digital. Los algoritmos de compresión se dividen en dos clases: *sin pérdida* y *con pérdida*. Los datos comprimidos utilizando un algoritmo de compresión sin pérdida pueden ser recuperados completamente, pero al utilizar un algoritmo con pérdida generalmente se pierde parte de la información en la reconstrucción. Lógicamente, los algoritmos de compresión con pérdida en general alcanzan mayores niveles de compresión que los algoritmos sin pérdida.

Para cada uno de los distintos tipos de datos digitales (imágenes, audio, video, etc.) existe una gran variedad de algoritmos de compresión con y sin pérdida. La clase de algoritmo utilizado depende del propósito del usuario. Supongamos, por ejemplo, que nos interesa comprimir un CD de audio. Si nuestro objetivo es realizar una copia de seguridad y la capacidad de almacenamiento no es un obstáculo, en general optaremos por comprimirlo utilizando un algoritmo sin pérdida, como por ejemplo FLAC [2], que permite comprimir archivos de audio a 50-60% de su tamaño original. En cambio, si lo que nos interesa es compartir el audio a través de Internet o escucharlo en un reproductor portátil, y no nos molesta perder calidad de sonido respecto al archivo original[2], podemos comprimirlo utilizando un algoritmo con pérdida. Con el algoritmo MP3 [4] se puede comprimir un archivo de audio a 10-20% de su tamaño original. El archivo comprimido se puede subir a Internet a mayor velocidad y es necesario menos espacio de almacenamiento si lo copiamos a un reproductor portátil, lo que es importante ya que en general estos dispositivos tienen capacidad limitada.

Tanto en el área comercial como en diversos campos de la investigación científica, la compresión de datos cumple un rol cada vez más relevante. Vale la pena mencionar algunos ejemplos para ver el alcance:

- Todos los datos que los servicios de streaming de audio (Spotify, Apple Music, etc.) y video (Youtube, Netflix, etc.) envían desde sus servidores hasta el dispositivo del usuario final se comprimen para optimizar el uso del ancho de banda [5, 6]. Algo similar ocurre con la televisión digital, donde las señales se codifican en el emisor y se decodifican en el receptor [7].

[1] Un Zettabyte (ZB) es una unidad de almacenamiento de información digital, equivalente a 10^{12} GB.

[2] Varios estudios sostienen que el oído humano es incapaz de diferenciar entre audio sin compresión y audio comprimido utilizando ciertos algoritmos con pérdida (ver, por ejemplo, [3]).

- Las llamadas de voz y video realizadas utilizando software VoIP (*Voice over IP*), como por ejemplo Skype [8], involucran técnicas de compresión, implementadas en los dispositivos de todos los participantes, sin las cuales no sería posible lograr una comunicación en tiempo real con buena calidad de imagen y sonido.

- Las cámaras de fotos digitales comprimen las fotos para aprovechar mejor el espacio de almacenamiento y hacer más rápida su transferencia. En general el usuario tiene la opción de comprimir las fotos con pérdida (JPEG es el algoritmo más popular) o sin pérdida (TIFF es un algoritmo bastante utilizado, aunque las marcas más grandes tienen algoritmos propios [9, 10]).

- Existe una gran variedad de programas que sirven para comprimir sin pérdida archivos de cualquier tipo. Entre los más populares se encuentran Gzip [11] y WinRAR [12], que además de permitirle al usuario comprimir un archivo, le dejan encriptarlo y partirlo en varios archivos individuales.

- En el área de la medicina, para determinados estudios se requiere que el paciente utilice un dispositivo que capture información de diagnóstico (por ejemplo electroencefalogramas o electrocardiogramas) durante períodos prolongados, transmitiendo los datos de manera inalámbrica a un dispositivo remoto. En estos casos se utilizan algoritmos para comprimir los datos enviados[3], lo que disminuye la cantidad de energía utilizada por el transmisor, ya que debe transmitir una menor cantidad de bits [13].

- En las misiones realizadas por la NASA se debe transmitir información a la Tierra desde lugares muy remotos, lo que implica un alto consumo energético. Tanto en la misión a Marte de la década pasada [14, 15], como en la actual misión a Plutón [16, 17], se utilizaron algoritmos de compresión con el fin de ahorrar recursos tales como energía, espacio y costos.

Como vemos, hay una gran variedad de escenarios en los que los algoritmos de compresión resultan útiles. En particular, la motivación inicial de nuestro proyecto es estudiar el problema que presentamos a continuación. Consideramos una señal, compuesta de n canales, dada como una secuencia de vectores de dimensión n, cuyos componentes toman valores sobre un alfabeto finito. Esta señal podría ser, por ejemplo, audio, un electroencefalograma o una imagen. En determinadas situaciones es necesario reducir la cantidad de canales de la señal, creando m canales ($m < n$), cada uno calculado como función de los canales originales. Por ejemplo, podemos querer convertir una señal de audio grabada en cinco canales a dos canales, para poder reproducirla en un equipo estereofónico. En general existe correlación entre los canales de la señal original, y a su vez las funciones aplicadas al reducir los canales también inducen correlación entre los canales de la señal transformada. Como primer problema de estudio nos planteamos determinar si un algoritmo de compresión en el que el codificador conoce la señal original y las funciones de transformación puede sacar provecho de esta información para obtener mejores niveles de compresión de los que se pueden lograr con otro algoritmo cuyo codificador solamente tiene acceso a la señal transformada. Llegamos a la conclusión de que la información adicional que conoce el codificador no le sirve al primer algoritmo para comprimir la señal transformada con menos bits que el segundo algoritmo.

Continuamos el proyecto estudiando una aplicación práctica del problema teórico presentado en el párrafo anterior. Tomamos $n = 2$ y $m = 1$, y consideramos que las dos componentes de la señal original están gobernadas por una variable aleatoria con distribución geométrica. Como veremos más adelante en este capítulo, la distribución geométrica aparece naturalmente para modelar las variables que se codifican en compresores de, por ejemplo, audio, imágenes y señales médicas.

[3]Se debe utilizar algoritmos de compresión de baja complejidad computacional, que no consuman demasiada energía al comprimir los datos.

Transformamos la señal original sumando sus dos componentes, obteniendo como resultado una señal con distribución binomial negativa, que es la que nos interesa comprimir. Si bien en [18] se demostró que cualquier variable aleatoria con distribución geométrica se puede codificar de forma *óptima*[4], mediante un código de Golomb [19], no encontramos bibliografía específica referida a la compresión de la distribución binomial negativa. Una forma válida de codificar la suma de las dos componentes es codificar ambas componentes por separado utilizando un par de codificadores de Golomb óptimos, pero en ese caso el codificador transmitiría más información de la necesaria (nos interesa solo el resultado de la suma, no el valor de cada sumando por separado). Además, el codificador utiliza información de la señal original, y la conclusión teórica presentada en el párrafo anterior indica que debe existir otro algoritmo de compresión igual de bueno en el que el codificador no disponga de ninguna información adicional. Por lo tanto, nos planteamos el objetivo de estudiar cómo codificar variables con distribución binomial negativa.

En este proyecto proponemos dos codificadores sin pérdida para la distribución binomial negativa: el codificador GolombBN y el codificador T. Implementamos ambos codificadores y realizamos experimentos comparando su rendimiento, los tiempos de codificación y la manera en que varían sus respectivos parámetros. El codificador GolombBN es una versión levemente modificada del codificador de Golomb, y su principal ventaja es que tiene una baja complejidad computacional, lo que le permite codificar con tiempos de ejecución muy cortos. La desventaja es que los largos medios de código obtenidos no están cerca del óptimo. El codificador T, en cambio, es bastante más complejo y requiere mayor tiempo de procesamiento, pero los largos medios de código obtenidos son significativamente menores. De hecho, por los datos experimentales obtenidos, pensamos que es o está muy cerca de ser un codificador óptimo para la distribución binomial negativa. La caracterización teórica de códigos óptimos para la distribución binomial negativa queda fuera del alcance del proyecto, pero es un problema interesante que queremos resolver en el futuro.

Antes de finalizar la introducción nos parece importante plantearnos la siguiente pregunta: ¿En qué casos prácticos puede llegar a ser útil saber cómo codificar una variable con distribución binomial negativa? Para responder esa pregunta, primero debemos presentar los algoritmos de compresión de *modelado predictivo*.

Supongamos que tenemos una foto digital y elegimos un pixel al azar. En general, el color y brillo asociado a ese pixel es similar al de los pixeles cercanos. Los algoritmos de compresión de modelado predictivo para imágenes se basan en esta correlación espacial entre pixeles. El codificador procesa los pixeles de manera secuencial y se utiliza un modelo estadístico para aproximar el valor de cada pixel teniendo en cuenta información relativa a los pixeles que ya fueron procesados. El *predictor* es una función que utiliza la información del modelo para predecir, con mayor o menor exactitud, el valor de cada uno de los pixeles de la imagen. A la diferencia entre el valor calculado por el predictor y el valor real del pixel se le llama *residuo*. El codificador comprime una imagen codificando los residuos de manera secuencial. El decodificador suma los residuos decodificados a sus propias predicciones (que son iguales a las calculadas por el codificador), reconstruyendo de esta manera los valores originales de los pixeles.

Consideremos el siguiente escenario: tenemos dos señales correlacionadas, y para cada una conocemos un algoritmo de compresión de modelado predictivo que utiliza un predictor lineal tal que el residuo se ajusta a una distribución geométrica. Supongamos que nuestro objetivo es comprimir la suma de las dos señales. Como el predictor es lineal, calcular la suma de las predicciones equivale a calcular la predicción aplicada a la suma de las muestras. En este escenario, la suma de los residuos de las respectivas señales tiene una distribución binomial negativa, ya que

[4]Un código es óptimo para una variable aleatoria si la puede codificar utilizando menos bits que cualquier otro código. En 2.2.10 definimos este concepto formalmente.

es el resultado de sumar dos geométricas. Entonces podemos comprimir la suma de las señales utilizando el codificador T para codificar la suma de los residuos.

Para terminar, es importante destacar que existen diferentes algoritmos de compresión sin pérdida con modelado predictivo, por ejemplo LOCO-I [20] para imágenes y Shorten [21] para audio[5], con la siguiente característica: los residuos obtenidos se ajustan muy bien a una distribución geométrica a dos lados centrada en 0. Además, Shorten utiliza un predictor lineal. Por lo tanto, existen en la práctica casos similares al descrito en el párrafo anterior y en el futuro sería interesante estudiar qué modificaciones le podemos hacer al codificador T para comprimir la suma de señales para las que existe un predictor cuyo residuo se ajusta bien a una distribución geométrica a dos lados.

Organización del documento. El documento está organizado en siete capítulos, incluyendo la introducción. En el capítulo 2 exponemos los fundamentos teóricos de la Teoría de la información necesarios para comprender el resto del documento, incluyendo una introducción a los códigos de Golomb. En el capítulo 3 estudiamos el problema de codificación de señales multicanal transformadas a través de un proceso de reducción de canales. En los capítulos 4 y 5 describimos en detalle los dos codificadores implementados en el proyecto, el codificador GolombBN y el codificador T, respectivamente. En el capítulo 6 exponemos los resultados experimentales obtenidos. En el capítulo 7 presentamos las conclusiones del proyecto e identificamos problemas que pensamos sería interesante estudiar en el futuro. Al final del documento incluimos un glosario.

[5]El procedimiento para comprimir audio es análogo al explicado para las imágenes, pero en vez de representar pixeles se representan muestras de formas de onda con correlación temporal en vez de espacial.

Capítulo 2

Fundamentos teóricos

En este capítulo exponemos los fundamentos teóricos que sirven como base para comprender el resto del informe. En la secciones 2.1 y 2.2 definimos algunos conceptos básicos de la Teoría de la Información y la codificación de fuentes. En las secciones 2.3 y 2.4 presentamos los códigos de Huffman y los códigos de Golomb, utilizados por los dos codificadores implementados en el proyecto.

2.1 Teoría de la Información: conceptos básicos

Definición 2.1.1. Utilizamos la notación $\mathbb{Z}$ para los enteros, $\mathbb{Z}_{\geq 0}$ para los enteros no negativos, $\mathbb{Z}_{>0}$ para los enteros positivos, $\mathbb{Z}_{[a,b]}$ para los enteros del intervalo $[a, b]$, $\mathbb{R}_{[0,1]}$ para los reales del intervalo $[0, 1]$ y $\mathbb{R}_{(0,1)}$ para los reales del intervalo $(0, 1)$.

Definición 2.1.2. Un *alfabeto* es un conjunto numerable[1] de *símbolos*. Llamamos *alfabeto D-ario* a un alfabeto $\mathcal{D} = \{0, 1, ..., D-1\}$ compuesto por D símbolos. En algunos casos nos interesa especialmente el *alfabeto binario*, $\mathcal{B} = \{0, 1\}$.

A continuación definimos la entropía, concepto central de la Teoría de la Información. La entropía de una variable aleatoria mide la incertidumbre que existe a priori sobre el resultado de la variable o, equivalentemente, la cantidad de información que aporta en media la observación de la misma.

Definición 2.1.3. Sea X una variable aleatoria que genera símbolos del alfabeto $\mathcal{X}$ siguiendo una función de probabilidad $p(x) = \mathrm{P}(X = x)$, $x \in \mathcal{X}$. Definimos la *entropía* de X como

$$\begin{aligned} H(X) &= E_p\Big[-\log_2 p(X)\Big] \\ &= -\sum_{x\in\mathcal{X}} p(x)\log_2 p(x), \end{aligned} \tag{2.1}$$

donde E_p denota la esperanza con respecto a p. La entropía se expresa en bits y al calcularla utilizamos la convención $0 \log_2 0 = 0$.

Ejemplo 2.1.1. Consideremos una variable aleatoria X que genera símbolos del alfabeto $\mathcal{X} = \{a, b, c, d\}$ con las siguientes probabilidades: $p(a) = 1/8$, $p(b) = 1/4$, $p(c) = 1/8$ y $p(d) = 1/2$. Para calcular la entropía de X utilizamos la fórmula (2.1), obteniendo

[1] Un conjunto S es numerable si existe una función inyectiva $f : S \to \mathbb{Z}_{\geq 0}$.

$$\begin{aligned} H(X) &= -\sum_{x\in\{a,b,c,c\}} p(x)\log_2 p(x) \\ &= -\frac{1}{8}\log_2\frac{1}{8} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{2}\log_2\frac{1}{2} \\ &= \frac{7}{4} \text{ bits.} \end{aligned} \tag{2.2}$$

En el ejemplo 2.1.1 observamos que la entropía no depende del alfabeto de la variable aleatoria, sino únicamente de la probabilidad con la que la variable aleatoria genera cada símbolo. Teniendo esto en cuenta, a continuación definimos la entropía de un vector de probabilidad.

Definición 2.1.4. Sea $\mathbf{p} = (p_1, .., p_n)$ un vector de probabilidad. La *entropía del vector* $\mathbf{p}$ está dada por

$$H(\mathbf{p}) = -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{2.3}$$

En particular, cuando $n = 2$, el vector de probabilidad es de la forma $\mathbf{p} = (p, 1-p)$, $p \in \mathbb{R}_{[0,1]}$. La entropía de $\mathbf{p}$ como función del parámetro escalar p se denomina *entropía binaria* y la denotamos $H(p)$. Aplicando (2.3) obtenemos

$$H(p) = -p\log_2 p - (1-p)\log_2(1-p). \tag{2.4}$$

En la figura 2.1 presentamos la gráfica de la función $H(p)$. Observamos que la entropía binaria vale 0 cuando $p \in \{0, 1\}$. Esto tiene sentido, ya que en ambos casos uno de los elementos del vector de probabilidades vale 1, y por lo tanto no hay incertidumbre. Al contrario, la incertidumbre es máxima cuando $p = 1/2$, ya que los dos elementos del vector tienen igual probabilidad $1/2$. Para ese valor de p la entropía binaria alcanza su máximo valor, 1 bit.

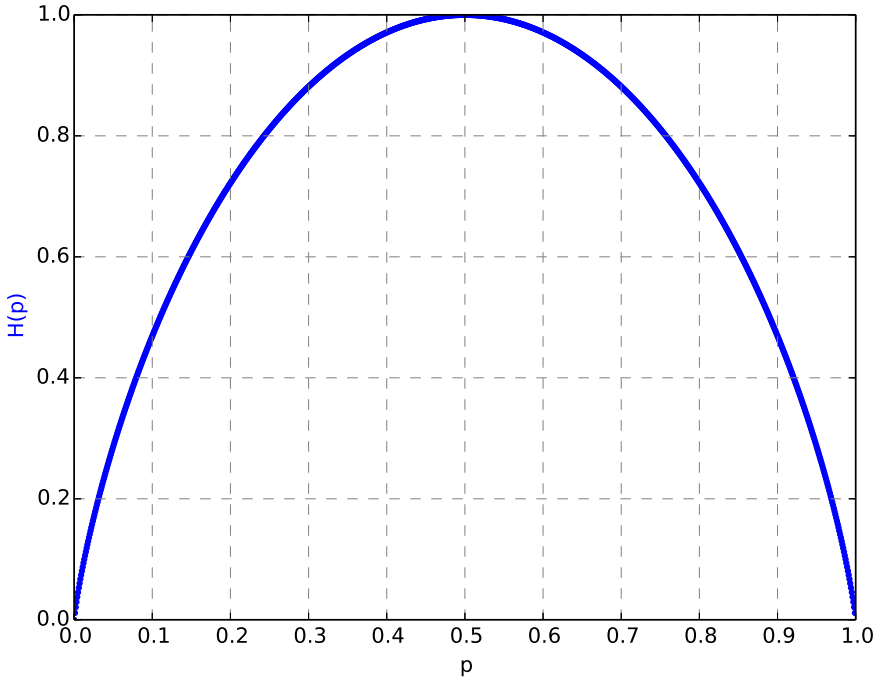

FIGURA 2.1: $H(p)$ para $p \in \mathbb{R}_{[0,1]}$.

2.2 Codificación de fuentes

2.2.1 Conceptos básicos

Definición 2.2.1. Dado un alfabeto $\mathcal{D}$, una *palabra* es una secuencia de símbolos $d_1...d_k$, $d_i \in \mathcal{D}$, $1 \leq i \leq k$. Decimos que k es el *largo* de la palabra y utilizamos la notación d^k para referirnos a una palabra de largo k. A la palabra de largo cero la llamamos *palabra vacía* y la denotamos ϵ.

Definición 2.2.2. Denotamos $\mathcal{D}^*$ al conjunto infinito compuesto por todas las palabras que se pueden formar utilizando símbolos del alfabeto $\mathcal{D}$, incluyendo a la palabra vacía.

Para el alfabeto binario $\mathcal{B} = \{0, 1\}$, $\mathcal{B}^*$ está compuesto por exactamente 2^k palabras para cada largo k posible: la palabra ϵ de largo cero, 2 palabras de largo 1 ("0" y "1"), 4 palabras de largo 2 ("00", "01", "10" y "11"), etc.

Definición 2.2.3. Dado un alfabeto $\mathcal{D}$, la *concatenación* es una operación $\bowtie$: $\mathcal{D}^* \times \mathcal{D}^* \to \mathcal{D}^*$ tal que para cada par palabras $u, v \in \mathcal{D}^*$ se cumple:

1. Si $v = \epsilon$, entonces $u \bowtie v = v \bowtie u = u$.
2. Si $u = u_1...u_n$ y $v = v_1...v_m$, entonces $u \bowtie v = u_1...u_n v_1...v_m$.

Estos son algunos ejemplos que muestran cómo se concatenan palabras en $\mathcal{B}^*$: $01 \bowtie 00 = 0100$, $00 \bowtie \epsilon = 00$, $111 \bowtie 0 = 1110$ y $01 \bowtie 0101 = 010101$. Es sencillo verificar que la concatenación de palabras es una operación asociativa, que tiene a la palabra vacía como el elemento neutro.

Definición 2.2.4. Sea $\mathcal{D}$ un alfabeto y X una variable aleatoria que genera símbolos del alfabeto $\mathcal{X} = \{x_1, ..., x_n\}$. Un *código* C para X es un mapeo $C : \mathcal{X} \to \mathcal{D}^*$. Decimos que el código C *codifica* un símbolo x_i con la *palabra de código* $C(x_i)$. Con $l_C(x_i)$ denotamos el largo de $C(x_i)$.

Ejemplo 2.2.1. Supongamos que X_0 es una variable aleatoria que genera símbolos del alfabeto $\mathcal{X}_0 = \{x_1, x_2, x_3\}$. Consideremos un código $C_0 : \mathcal{X}_0 \to \mathcal{B}^*$, donde $C_0(x_1) = C_0(x_2) = 0$ y $C_0(x_3) = 100$. En este caso, tenemos $l_{C_0}(x_1) = l_{C_0}(x_2) = 1$ y $l_{C_0}(x_3) = 3$.

Definición 2.2.5. Sea X una variable aleatoria que genera símbolos del alfabeto $\mathcal{X}$ siguiendo una función de probabilidad $p(x)$. Definimos el *largo medio* de un código C para X como el valor esperado de $l_C(X)$, es decir

$$L(C) = \sum_{x_i \in \mathcal{X}} p(x_i) l_C(x_i). \tag{2.5}$$

Retomando el ejemplo 2.2.1, supongamos que la variable aleatoria X_0 está dada por $p(x_1) = 0.4$, $p(x_2) = 0.25$ y $p(x_3) = 0.35$. Aplicando (2.5) en este caso tenemos

$$\begin{aligned} L(C_0) = \sum_{x_i \in \mathcal{X}_0} p(x_i) l_{C_0}(x_i) &= p(x_1) l_{C_0}(x_1) + p(x_2) l_{C_0}(x_2) + p(x_3) l_{C_0}(x_3) \\ &= 0.4 \cdot 1 + 0.25 \cdot 1 + 0.35 \cdot 3 = 1.7 \text{ bits.} \end{aligned} \tag{2.6}$$

Definición 2.2.6. Decimos que un código C es *no singular* si codifica a cada símbolo $x_i \in \mathcal{X}$ con una palabra de código diferente. Formalmente, se cumple $x_j \neq x_h \implies C(x_j) \neq C(x_h)$ para todo x_j, $x_h \in \mathcal{X}$.

Ejemplo 2.2.2. El código $C_1 : \mathcal{X}_1 \to \mathcal{B}^*$, con $\mathcal{X}_1 = \{x_1, x_2, x_3\}$ y $C_1(x_1) = 0$, $C_1(x_2) = 1$ y $C_1(x_3) = 101$ es un ejemplo de código no singular. En cambio, el código C_0 del ejemplo 2.2.1 cumple $C_0(x_1) = C_0(x_2)$ y es, por lo tanto, un código singular.

Consideremos un código C que codifica a un símbolo x_i con la palabra de código $C(x_i)$. Si C es un código no singular, siempre podemos obtener (*decodificar*) x_i a partir de $C(x_i)$. En la práctica, sin embargo, no trabajamos con símbolos aislados sino con secuencias de símbolos. Por lo tanto, lo que nos interesa es poder decodificar secuencias de símbolos de $\mathcal{X}$. Con ese propósito en mente, a continuación definimos la extensión de un código.

Definición 2.2.7. La *extensión* de un código C es el código $\mathcal{C} : \mathcal{X}^* \to \mathcal{D}^*$ que cumple

$$\mathcal{C}(x^k) = C(x_1) \bowtie C(x_2) \bowtie ... \bowtie C(x_k) \tag{2.7}$$

para toda palabra $x^k = x_1...x_k \in \mathcal{X}^*$.

Definición 2.2.8. Un código es *unívocamente decodificable (UD)* si y solo si su extensión es no singular.

Ejemplo 2.2.3. Para la extensión $\mathcal{C}_1 : \mathcal{X}_1^* \to \mathcal{B}^*$ del código C_1 del ejemplo 2.2.2, tenemos $\mathcal{C}_1(x_2x_1x_2) = C_1(x_2) \bowtie C_1(x_1) \bowtie C_1(x_2) = 1 \bowtie 0 \bowtie 1 = 101$ y $\mathcal{C}_1(x_3) = C_1(x_3) = 101$. C_1 no es UD porque, como $\mathcal{C}_1(x_2x_1x_2) = \mathcal{C}_1(x_3)$, no es posible decodificar la secuencia 101 de manera unívoca.

Ejemplo 2.2.4. Consideremos un código $C_2 : \mathcal{X}_2 \to \mathcal{B}^*$, con $\mathcal{X}_2 = \{x_1, x_2, x_3, x_4\}$ y $C_2(x_1) = 10$, $C_2(x_2) = 00$, $C_2(x_3) = 11$ y $C_2(x_4) = 110$. El código C_2 es UD. Para probarlo, consideremos cualquier secuencia binaria e intentemos decodificarla. Si sus dos primeros bits son 10 o 00, entonces el primer símbolo decodificado será x_1 o x_2, ya que $C_2(x_1) = 10$ y $C_2(x_2) = 00$. Si, en cambio, los dos primeros bits son 11, debemos considerar los próximos bits. Si el tercer bit es un 1, entonces el primer símbolo decodificado es x_3, ya que $C_2(x_3) = 11$, y además sabemos que la segunda palabra de código de la secuencia comienza con 1. Si el tercer bit es un 0 debemos considerar dos casos: si la cadena de ceros tiene largo impar (por ejemplo 1101... o 110001...) el primer símbolo decodificado es x_4; en cambio, si la cadena de ceros tiene largo par (por ejemplo 11001... o 1100001...) el primer símbolo decodificado es x_3. Repitiendo este argumento podemos ver que el código C_2 es UD.

Definición 2.2.9. Un código es *instantáneo* o *libre de prefijo* si y solo si ninguna palabra de código es prefijo de otra palabra de código.

Los códigos instantáneos presentan la ventaja de que cualquier palabra de código puede ser decodificada de forma inmediata, aún cuando aparece dentro de una secuencia. En particular, un símbolo x_i puede ser decodificado una vez que hemos leído el último bit de la palabra de código asociada $C(x_i)$. Observamos que esto no se cumple con el código C_2 del ejemplo 2.2.4, ya que $C_2(x_3) = 11$ es prefijo de $C_2(x_4) = 110$. Por lo tanto, C_2 no es un código instantáneo.

Ejemplo 2.2.5. El código $C_3 : \mathcal{X}_3 \to \mathcal{B}^*$, con $\mathcal{X}_3 = \{x_1, x_2, x_3, x_4\}$ y $C_3(x_1) = 0$, $C_3(x_2) = 11$, $C_3(x_3) = 101$ y $C_3(x_3) = 100$ es un ejemplo de código instantáneo.

A continuación definimos el código unario, otro ejemplo de código instantáneo. El código unario es utilizado tanto por los códigos de Golomb, presentados en la sección 2.4, como por los dos codificadores implementados en este proyecto, detallados en los capítulos 4 y 5.

2.2.1.1 Código unario El código unario $U : \mathbb{Z}_{\geq 0} \to \mathcal{B}^*$ codifica un entero no negativo i con una palabra de código formada por una cadena de i ceros seguida de un uno. La palabra de código $U(i)$ tiene largo $l_U(i) = i + 1$. A modo de ejemplo, tenemos $U(0) = 1$, $U(1) = 01$, $U(2) = 001$ y $U(3) = 0001$.

En la tabla 2.1 se muestran los códigos presentados en esta sección, los cuales sirven como ejemplo para las clases de código definidas. En el caso del código unario tenemos $x_i = i - 1$ para cada $i \in \mathbb{Z}_{>0}$, y en la tabla solamente figuran los primeros cuatro símbolos (enteros) que podemos codificar utilizando U.

$x_i \in \mathcal{X}$	$C_0(x_i)$	$C_1(x_i)$	$C_2(x_i)$	$C_3(x_i)$	$U(x_i)$
x_1	0	0	10	0	1
x_2	0	1	00	11	01
x_3	100	101	11	101	001
x_4	-	-	110	100	0001

TABLA 2.1: Códigos presentados en los ejemplos de la sección 2.2.1.

En la figura 2.2 se resumen las diferentes clases de código definidas en esta sección, incluyendo los ejemplos presentados para cada clase. Se observa que los códigos no singulares incluyen a los códigos UD, y a su vez estos incluyen a los códigos instantáneos.

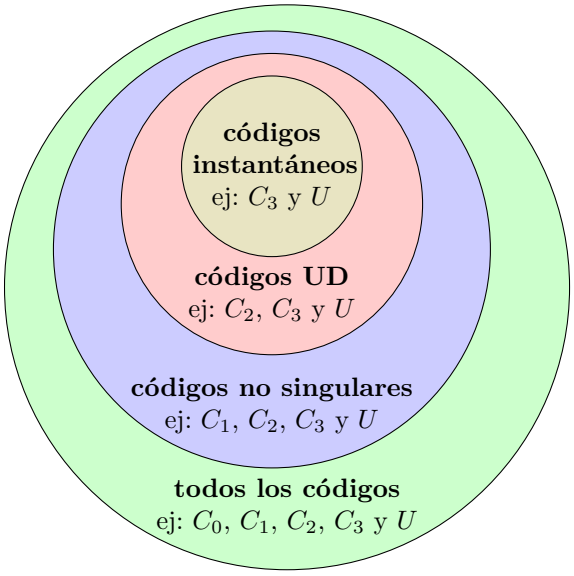

FIGURA 2.2: Clases de código definidas en la sección 2.2.1, con sus respectivos ejemplos.

2.2.2 Esquema de codificación de fuente

En la figura 2.3 presentamos un esquema que resume el proceso de codificación de fuente. Tenemos una fuente gobernada por una variable aleatoria X que genera símbolos de un alfabeto $\mathcal{X} = \{x_1, x_2, ..., x_n\}$. Nuestro objetivo es transmitir las cadenas de símbolos generadas por la fuente a un destino, a través de un canal, cumpliendo con dos requisitos:

- Las cadenas de símbolos que llegan al destino deben ser exactamente las mismas que fueron generadas por la fuente.
- Debemos transmitir a través del canal la menor cantidad de bits posible. Dicho de otra manera, nos interesa hacer un uso *óptimo* del canal.

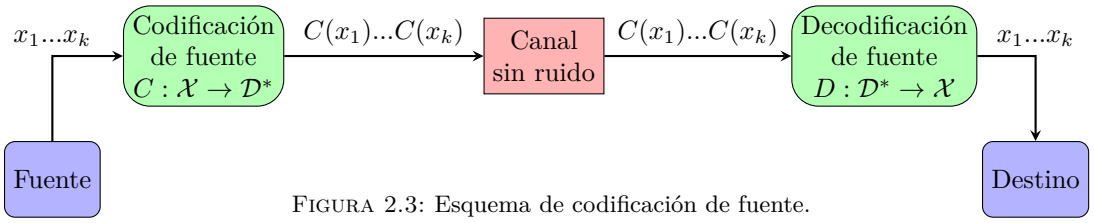

FIGURA 2.3: Esquema de codificación de fuente.

Considerado desde un punto de vista *espacial*, un canal nos permite enviar información entre dos puntos físicos distantes. El ejemplo típico es Internet. Supongamos que queremos enviar un archivo desde una computadora a otra a través de Internet. En ese caso, ya sea por razones de tiempo (menor tiempo de envío) o por razones económicas (menor gasto en servicio de Internet), en general conviene enviar el archivo utilizando la menor cantidad posible de bits. Para lograr esto, podemos comprimir el archivo en el origen y luego descomprimirlo en el destino. En la figura 2.3, los procesos de compresión y descompresión del archivo están representados por las etapas de codificación y decodificación de fuente, respectivamente.

El canal también puede ser considerado desde un punto de vista *temporal*. Por ejemplo, si tenemos un disco duro en el que almacenamos información a la que queremos acceder en el futuro. El disco duro tiene un espacio limitado, por lo que en general conviene comprimir los archivos que queremos almacenar.

Comentamos que no solo nos interesa enviar información de la fuente al destino, sino que queremos hacer un uso óptimo del canal. Para lograr esto son necesarias las etapas de codificación y decodificación de fuente. En la etapa de codificación, los símbolos generados por la fuente se codifican utilizando un código instantáneo $C : \mathcal{X} \to \mathcal{D}^*$. Esto implica que si el codificador recibe una cadena de símbolos $x_1...x_k$, la salida será la cadena de símbolos mapeados $C(x_1)...C(x_k)$.

Como se observa en la figura 2.3, el canal del esquema no tiene ruido, lo que nos asegura que los bits que se envían a través del canal no son alterados. Esto implica que el decodificador recibe la misma secuencia de bits que sale del codificador. En general esto no ocurre en la práctica, pero es un modelo razonable si agregamos una etapa de codificación/decodificación de códigos para corrección de errores como se explica más adelante.

En la etapa de decodificación de fuente se debe aplicar una función inversa a la aplicada en la etapa de codificación. El decodificador D realiza entonces un mapeo de la forma $D : \mathcal{D}^* \to \mathcal{X}$. Si D funciona correctamente, esperamos que al recibir una cadena $C(x_1)...C(x_k)$, la salida sea la cadena $D(C(x_1))...D(C(x_k)) = x_1...x_k$. Observar que como C es un código instantáneo, D es capaz de decodificar cualquier palabra de código de forma inmediata.

Para optimizar el uso del canal debemos utilizar un código C tal que el largo medio $L(C)$ sea el mínimo posible para la variable aleatoria X que gobierna la fuente. A un código con esa característica le llamamos código óptimo. A continuación lo definimos formalmente.

Definición 2.2.10. Decimos que C es un *código óptimo* para una fuente si el largo medio obtenido al codificarla con el código C es menor o igual al largo medio obtenido al codificarla con cualquier otro código.

A continuación presentamos una extensión del Primer Teorema de Shannon [22], también conocido como Teorema de codificación de fuente. Ese teorema nos da una cota inferior, y la extensión una cota superior, para el largo medio de un código óptimo.

Teorema 2.2.1. Sea C un código binario instantáneo óptimo para una variable aleatoria X sobre un alfabeto $\mathcal{X}$. Entonces su largo medio $L(C)$ cumple

$$H(X) \leq L(C) < H(X) + 1, \tag{2.8}$$

con $L(C) = H(X)$ si y solo si se tiene $p(x_i) = 2^{-l_C(x_i)}$ para todo $x_i \in \mathcal{X}$.

El Primer Teorema de Shannon es muy importante porque nos da un cota inferior para el largo medio de código que podemos obtener al codificar una fuente. Sabemos que no existe ningún código que nos permita comprimir la fuente más allá de su entropía. Además, la extensión del teorema nos garantiza que cualquier código óptimo para una variable aleatoria X tiene un largo medio que está a menos de un bit de la entropía de X.

Es importante destacar que para cualquier variable aleatoria siempre existe más de un código óptimo. Por ejemplo, dado un código óptimo, alcanza con intercambiar los unos y los ceros de cada palabra de código para obtener un código óptimo distinto. Partiendo de un código óptimo, también podemos obtener otros códigos óptimos haciendo permutaciones entre las palabras de código de igual largo. Incluso en algunos casos, dos códigos óptimos distintos pueden codificar el mismo símbolo con palabras de diferente largo.

Ejemplo 2.2.6. Supongamos que la fuente de la figura 2.3 genera símbolos de un alfabeto $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ con probabilidades $p(x) = (1/2, 1/4, 1/8, 1/8)$. Queremos construir un código C binario instantáneo que sea óptimo para X. Tenemos $p(x_1) = 2^{-1}$, $p(x_2) = 2^{-2}$ y $p(x_3) = p(x_4) = 2^{-3}$. Por lo tanto, por el teorema 2.2.1, sabemos que si el código cumple $l_C(x_1) = 1$, $l_C(x_2) = 2$ y $l_C(x_3) = l_C(x_4) = 3$, tendremos $L(C) = H(X)$. En efecto, aplicando las definiciones de entropía y largo medio de código, observamos que

$$\begin{aligned} H(X) &\overset{(2.1)}{=} -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8} \\ &= \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 3 + \frac{1}{8}\cdot 3 \overset{(2.5)}{=} L(C). \end{aligned} \tag{2.9}$$

En resumen, sabemos que utilizando un código C que codifique los símbolos con palabras de esos largos, minimizaremos la esperanza de la cantidad de bits que son enviados a través del canal. Como vimos, la única restricción que deben cumplir las palabras de código para que C sea instantáneo, es que ninguna palabra sea prefijo de otra. Entonces, por ejemplo, podemos definir el código C de esta forma: $C(x_1) = 1$, $C(x_2) = 01$, $C(x_3) = 000$ y $C(x_4) = 001$. Observamos que, lógicamente, los símbolos con mayor probabilidad se codifican con palabras de menor largo que los símbolos de menor probabilidad.

Una técnica utilizada para construir códigos óptimos para distribuciones de probabilidad conocidas es el algoritmo de Huffman, presentado en la sección 2.3. En el ejemplo 2.3.1 utilizamos el algoritmo de Huffman para obtener otro código óptimo para el vector probabilidades $p(x)$ del ejemplo 2.2.6.

Antes de finalizar esta sección, nos parece importante hacer algunas aclaraciones sobre el esquema de codificación de fuente presentado:

- El esquema es un modelo simplificado de la realidad. En la práctica puede ocurrir que el canal tenga ruido, en cuyo caso los bits que salen del canal no son necesariamente los mismos que ingresaron. En ese caso, para poder contrarrestar el ruido del canal debemos agregar un par de etapas al esquema: una etapa de *codificación de canal* y otra de *decodificación de canal*, inmediatamente antes y después del canal. El objetivo es agregarle redundancia a la información enviada a través del canal para poder recuperar los bits originales en caso de que el ruido en el canal los modifique. Para ello se pueden utilizar una variedad de mecanismos. El estudio de la codificación de canal es una rama importante de la Teoría de la Información, pero en este proyecto nos enfocamos en la codificación de fuente y no le dedicamos más que este párrafo al tema.

- En el esquema presentado se codifica y decodifica un símbolo de la fuente por vez. En la práctica, en ciertos casos podemos hacer un mejor uso del canal si trabajamos con *bloques* de símbolos. En esos casos, en la etapa de codificación se mapean los diferentes bloques de

símbolos a diferentes palabras de código. Cuando codificamos de a bloque tampoco debe haber palabras de código que sean prefijos de otras palabras.

- En la figura 2.2 observamos que la clase de los códigos UD incluye estrictamente a la clase de los códigos instantáneos. Sin embargo, en nuestro esquema hemos considerado un código instantáneo. Cabe preguntarse entonces si es posible hacer un mejor uso del canal utilizando algún código UD, en lugar de restringirnos al uso de los códigos instantáneos. La respuesta es que no, y es una consecuencia directa del Teorema de McMillan [23]. Al no obtener ninguna ventaja al considerar un código UD, siempre convendrá utilizar un código instantáneo, ya que eso nos permite decodificar las palabras de código de forma inmediata.

2.3 Códigos de Huffman

Utilizando el algoritmo de Huffman [24] es posible construir un código instantáneo óptimo para una distribución de probabilidad conocida sobre un alfabeto finito. Al código construido se le llama *código de Huffman*. Utilizamos la notación $CH_{\mathbf{p}}$ para designar al código obtenido al aplicar el algoritmo sobre una distribución de probabilidad dada por un vector $\mathbf{p} = (p_1, .., p_n)$. Además, con la notación CH_X designamos al código de Huffman construido para una variable aleatoria X cuyas probabilidades son conocidas.

El algoritmo utilizado para construir un código de Huffman binario es recursivo y consta de $n-1$ pasos para un vector $\mathbf{p}$ de tamaño n. En cada paso se agrupan los dos símbolos menos probables, formando un nuevo símbolo al que se le asigna una probabilidad igual a la suma de las probabilidades de los símbolos agrupados. En el último paso siempre se agrupan dos símbolos cuyas probabilidades suman 1, ya que el resultado de dicha suma equivale a la sumatoria de las probabilidades de los n símbolos originales[2].

Ejemplo 2.3.1. A continuación calcularemos el código de Huffman asociado a la variable aleatoria definida en el ejemplo 2.2.6. En ese caso tenemos $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ y $\mathbf{p} = (1/2, 1/4, 1/8, 1/8)$. En la tabla 2.2 se muestra la palabra de código obtenida para cada símbolo, junto con su largo y probabilidad.

$x_i \in \mathcal{X}$	$CH_{\mathbf{p}}(x_i)$	$l_{CH_{\mathbf{p}}}(x_i)$	$p(x_i)$
x_1	0	1	1/2
x_2	10	2	1/4
x_3	110	3	1/8
x_4	111	3	1/8

Tabla 2.2: Código de Huffman $CH_{\mathbf{p}}$ para el ejemplo 2.3.1.

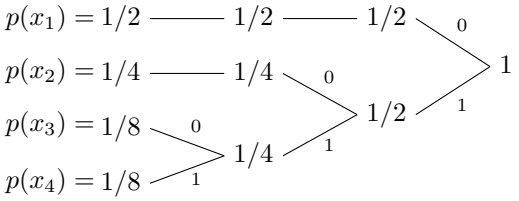

Figura 2.4: Algoritmo de Huffman para el ejemplo 2.3.1.

En la figura 2.4 se resumen de izquierda a derecha los 3 pasos llevados a cabo por el algoritmo. En el primer paso sumamos las probabilidades de los dos símbolos menos probables ($p(x_3)+p(x_4) = 1/8 + 1/8 = 1/4$). En el segundo paso, consideramos tres probabilidades: las de los símbolos x_1 y x_2 y la obtenida en el primer paso. Nuevamente sumamos las dos menores probabilidades, obteniendo $p(x_2) + 1/4 = 1/4 + 1/4 = 1/2$. En el tercer y último paso, solamente nos quedan dos probabilidades, que al ser sumadas dan 1.

[2]En este proyecto solamente vamos a trabajar con códigos de Huffman binarios, pero el algoritmo utilizado para construir códigos con alfabetos de tamaño $D \geq 3$ es análogo. La única diferencia es que en determinados casos se deben agregar símbolos dummy con probabilidad nula para garantizar que en cada paso de la recursión se puedan combinar D símbolos.

Para la suma realizada en cada paso, etiquetamos a uno de los sumandos con 0 y al otro con 1. Partiendo desde el 1 obtenido en el último paso, existe un camino único a cada símbolo x_i. El código de Huffman para un símbolo x_i se obtiene concatenando las etiquetas que aparecen en el camino desde el 1 hasta el símbolo x_i.

Para codificar un símbolo $x_i \in \mathcal{X}$ con el código $CH_{\mathbf{p}}$, simplemente debemos encontrar en la tabla 2.2 la palabra de código asociada $CH_{\mathbf{p}}(x_i)$. El proceso de decodificación no es tan sencillo y para llevarlo a cabo representamos al código con un árbol binario llamado árbol de Huffman. En la figura 2.5 presentamos el árbol de Huffman asociado al código $CH_{\mathbf{p}}$. Para decodificar un símbolo de una cadena de bits, se recorre el árbol desde la raíz hasta una hoja, siguiendo un camino determinado por los bits leídos. Cada símbolo $x_i \in \mathcal{X}$ está asociado a una hoja del árbol, y el símbolo decodificado es el que está asociado a la hoja en la que termina el recorrido.

A modo de ejemplo, supongamos que luego de codificar una cadena de símbolos $x_i \in \mathcal{X}$ con el código $CH_{\mathbf{p}}$, obtenemos la cadena de bits 11010011110. Para decodificarla vamos leyendo los bits en orden y bajando por el árbol desde la raíz. Leer los primeros tres bits de la cadena (110) equivale a tomar el camino A-B-C-x_3, por lo que el primer símbolo decodificado es x_3. Luego leemos los siguientes dos bits (10), tomamos el camino A-B-x_2 y decodificamos el símbolo x_2. Al leer el bit 0 tomamos el camino A-x_1 y decodificamos x_1. Aplicando un razonamiento análogo con los bits restantes, decodificamos los símbolos x_4 (camino A-B-C-x_4) y $x2$ (camino A-B-x_2). En resumen, al decodificar toda la cadena de bits obtenemos la cadena de símbolos $x_3x_2x_1x_4x_2$.

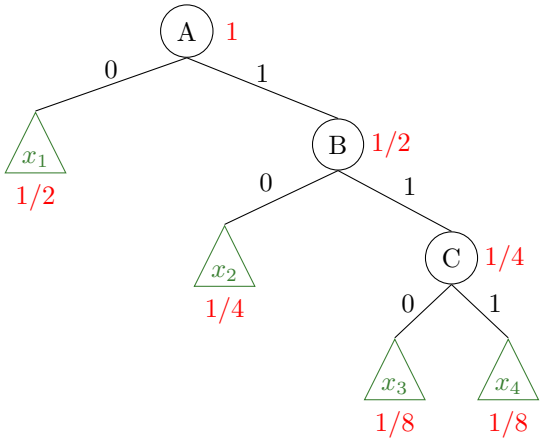

FIGURA 2.5: Árbol de Huffman asociado al código $CH_{\mathbf{p}}$ del ejemplo 2.3.1.

Ejemplo 2.3.2. Sea Y una variable aleatoria que toma valores en el conjunto $\mathcal{Y} = \{y_1, y_2, y_3, y_4, y_5, y_6\}$ con las probabilidades dadas por el vector $\mathbf{p}' = (0.25, 0.25, 0.20, 0.15, 0.10, 0.05)$. Aplicando el algoritmo de Huffman obtenemos el código presentado en la tabla 2.3, cuyo árbol asociado se muestra en la figura 2.6.

$y_i \in \mathcal{Y}$	$CH_{\mathbf{p}'}(y_i)$	$l_{CH_{\mathbf{p}'}}(y_i)$	$p(y_i)$
y_1	0	1	0.25
y_2	100	3	0.25
y_3	101	3	0.20
y_4	110	3	0.15
y_5	1110	5	0.10
y_6	1111	5	0.05

TABLA 2.3: Código de Huffman $CH_{\mathbf{p}'}$ para el ejemplo 2.3.2.

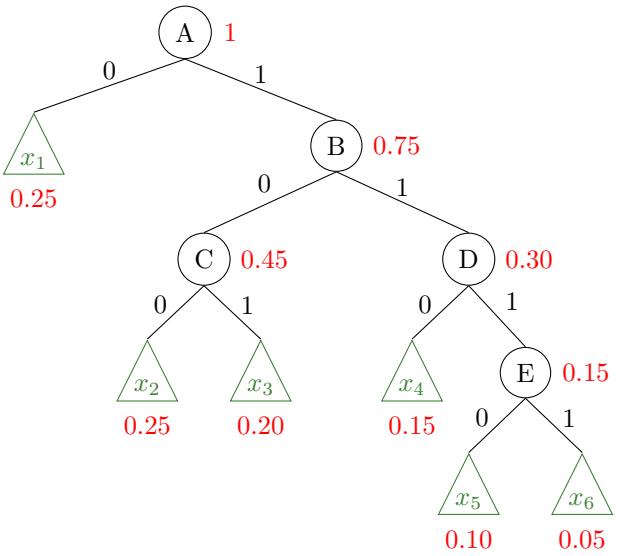

FIGURA 2.6: Árbol de Huffman asociado al código $CH_{\mathbf{p}'}$ del ejemplo 2.3.2.

En general existen varios códigos óptimos para una distribución conocida y el algoritmo de Huffman nos permite obtener uno de ellos. Considerando el esquema de codificación de la figura 2.3, lo anterior implica que si conocemos la distribución de probabilidad asociada a la variable aleatoria que gobierna la fuente, somos capaces de enviar información desde la fuente al destino haciendo un uso óptimo del canal. Codificando la fuente con un código de Huffman logramos minimizar la cantidad esperada de bits que circulan por el canal.

2.4 Códigos de Golomb

2.4.1 Descripción

Definición 2.4.1. Consideramos una secuencia de ensayos de Bernoulli independientes con idéntica probabilidad de éxito $\theta \in \mathbb{R}_{(0,1)}$. Definimos la variable aleatoria $X \in \mathbb{Z}_{\geq 0}$ como el número de ensayos exitosos realizados antes de que ocurra el primer fallo. Decimos que la variable aleatoria X sigue una *distribución geométrica* con parámetro θ, y nos referimos a ella utilizando la notación $X \sim Geo(\theta)$. La distribución de probabilidad de X está dada por

$$\mathrm{P}(X = i) = (1 - \theta)\theta^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{2.10}$$

Los códigos de Golomb [19] son una familia de códigos instantáneos utilizados para codificar sin pérdida fuentes con distribución geométrica. Dentro de la familia, un código de Golomb particular queda definido por un parámetro entero $l \in \mathbb{Z}_{>0}$. Utilizamos la notación G_l para el código de Golomb con parámetro l.

En [18] se demuestra que para cualquier variable aleatoria $X \sim Geo(\theta)$, existe un parámetro l tal que se cumple que el código G_l es óptimo para X. El parámetro l se calcula a partir de θ y es el único entero positivo que satisface la ecuación

$$\theta^l + \theta^{l+1} \leq 1 < \theta^l + \theta^{l-1}. \tag{2.11}$$

Un código de Golomb G_l codifica a un $i \in \mathbb{Z}_{\geq 0}$ de la siguiente manera

$$G_l(i) = U(i \text{ div } l) \bowtie B_l(i \bmod l), \tag{2.12}$$

donde:

- i div l y $i \bmod l$ representan, respectivamente, al cociente entero y resto de la división i/l,
- $U(j)$ es la palabra de código con la que el código unario codifica al entero j,
- $B_l(j)$ es la representación binaria de j utilizando $c_l = \lfloor \log_2 l \rfloor$ bits si $j < d_l = 2^{\lfloor \log_2 l + 1 \rfloor} - l$ o la representación binaria de $j + d_l$ utilizando $c_l + 1$ bits en otro caso.

Observar que se cumple $G_1(i) = U(i)$ para todo $i \in \mathbb{Z}_{\geq 0}$. Es decir que el código de Golomb con parámetro $l = 1$ es igual al código unario.

2.4.1.1 Códigos GPO2 Los códigos de Golomb cuyo parámetro l es potencia de 2 son llamados códigos *GPO2* (*Golomb-power-of-2*). Utilizamos la notación $\mathcal{G}_k$ para el código GPO2 con parámetro $l = 2^k$, con $k \in \mathbb{Z}_{\geq 0}$. Un código $\mathcal{G}_k$ codifica a un $i \in \mathbb{Z}_{\geq 0}$ con una palabra de código de largo $l_{\mathcal{G}_k}(i) = i \text{ div } 2^k + k + 1$, donde la representación binaria $B_l(i \bmod l)$ tiene k bits para todo $i \in \mathbb{Z}_{\geq 0}$.

Como en los códigos GPO2 la representación binaria tiene la misma cantidad de bits para cualquier entero, los procesos de codificación y decodificación son más simples que para el resto de códigos de Golomb. Un código GPO2 está formado por exactamente l palabras de código para cada largo mayor a k (l palabras de largo $k+1$, l palabras de largo $k+2$, etc). Un código G_l que no es GPO2 está formado por d_l palabras de código de largo c_l+1 y exactamente l palabras de código para cada largo mayor a c_l+1.

2.4.2 Ejemplos

Ejemplo 2.4.1. Supongamos que $l=5$ y que queremos calcular el código de Golomb asociado G_5. Para todo entero i en el rango $[0,4]$ se cumple $U(i \text{ div } 5) = U(0) = 1$ y $B_5(i \bmod 5) = B_5(i)$. Para todo entero j menor que $d_5 = 2^{\lfloor \log_2 5+1 \rfloor} - 5 = 2^3 - 5 = 3$, $B_5(j)$ es la representación binaria de j utilizando $c_5 = \lfloor \log_2 5 \rfloor = 2$ bits. Entonces tenemos $G_5(0) = 1 \bowtie 00$, $G_5(1) = 1 \bowtie 01$ y $G_5(2) = 1 \bowtie 10$. En cambio, para un $h \in \{3,4\}$, $B_5(h)$ es la representación binaria de $h+d_5 = h+3$ utilizando $c_5+1 = 3$ bits, que es igual a $6 = 110_2$ cuando $h=3$ y a $7 = 111_2$ cuando $h=4$. Entonces tenemos $G_5(3) = 1 \bowtie 110$ y $G_5(4) = 1 \bowtie 111$. Aplicando un razonamiento análogo es posible codificar cualquier otro entero no negativo con el código G_5.

En la tabla 2.4 presentamos el código G_5 para cada $i \in \mathbb{Z}_{[0,15]}$. El valor de la última columna, $l_{G_5}(i)$, representa el largo de la palabra de código para el entero i. Se puede verificar que el código está formado por $d_5 = 3$ palabras de largo $c_5+1 = 3$ (las que están asociadas a los primeros tres enteros) y exactamente $l=5$ palabras para cada largo mayor a c_5+1.

i	i div 5	i mod 5	$U(i \text{ div } 5)$	$B_5(i \bmod 5)$	$G_5(i)$	$l_{G_5}(i)$
0	0	0	1	00	100	3
1	0	1	1	01	101	3
2	0	2	1	10	110	3
3	0	3	1	110	1110	4
4	0	4	1	111	1111	4
5	1	0	01	00	0100	4
6	1	1	01	01	0101	4
7	1	2	01	10	0110	4
8	1	3	01	110	01110	5
9	1	4	01	111	01111	5
10	2	0	001	00	00100	5
11	2	1	001	01	00101	5
12	2	2	001	10	00110	5
13	2	3	001	110	001110	6
14	2	4	001	111	001111	6
15	3	0	0001	00	000100	6

TABLA 2.4: Código de Golomb G_5 para $i \in \mathbb{Z}_{[0,15]}$.

En la figura 2.7 presentamos el árbol binario asociado al código G_5. A diferencia de los árboles de Huffman, presentados en la sección anterior, este árbol es infinito, ya que los códigos de Golomb se definen para el conjunto (infinito) de los enteros no negativos. Observar que los nodos de color rojo están asociados a subárboles con la misma estructura. El camino desde la raíz del árbol a esos nodos representa la parte unaria del código.

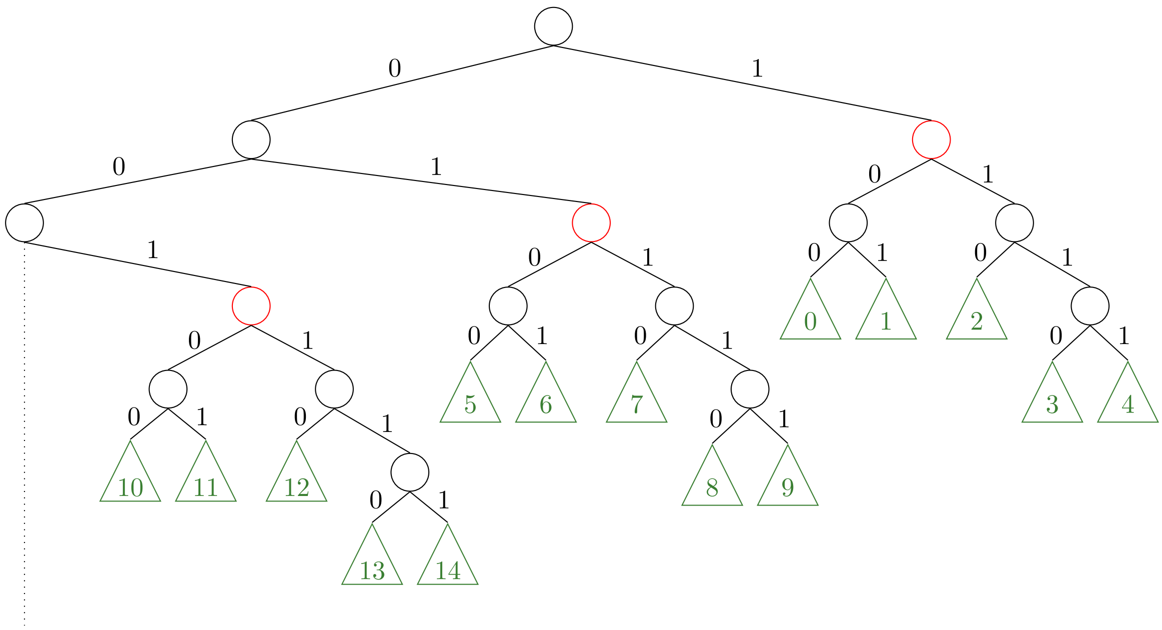

FIGURA 2.7: Árbol binario asociado al código de Golomb G_5.

Ejemplo 2.4.2. En este ejemplo consideramos un código GPO2. Supongamos que $l = 2^3$ y que queremos calcular el código $\mathcal{G}_3$, que es igual a G_8. Para todo entero i en el rango $[0, 7]$ se cumple $U(i \text{ div } 8) = U(0) = 1$ y $B_8(i \text{ mod } 8) = B_8(i)$. Todos los enteros del rango son menores a $d_8 = 2^{\lfloor \log_2 8+1 \rfloor} - 8 = 2^4 - 8 = 8$, por lo que $B_8(i)$ equivale a la representación binaria de i utilizando $c_8 = \lfloor \log_2 8 \rfloor = 3$ bits. Es posible codificar cualquier otro entero no negativo con el código $\mathcal{G}_3$ aplicando un razonamiento análogo.

En la tabla 2.5 presentamos el código $\mathcal{G}_3$ para cada $i \in \mathbb{Z}_{[0,15]}$. $l_{\mathcal{G}_3}(i)$ representa el largo de la palabra de código para el entero i, que en este caso cumple $l_{\mathcal{G}_3}(i) = i \text{ div } 2^3 + 3 + 1 = i \text{ div } 8 + 4$. Se puede verificar que el código está formado por exactamente $l = 8$ palabras para cada largo mayor a $k = 3$.

i	i div 8	i mod 8	$U(i$ div 8)	$B_8(i$ mod 8)	$\mathcal{G}_3(i)$	$l_{\mathcal{G}_3}(i)$
0	0	0	1	000	1000	4
1	0	1	1	001	1001	4
2	0	2	1	010	1010	4
3	0	3	1	011	1011	4
4	0	4	1	100	1100	4
5	0	5	1	101	1101	4
6	0	6	1	110	1110	4
7	0	7	1	111	1111	4
8	1	0	01	000	01000	5
9	1	1	01	001	01001	5
10	1	2	01	010	01010	5
11	1	3	01	011	01011	5
12	1	4	01	100	01100	5
13	1	5	01	101	01101	5
14	1	6	01	110	01110	5
15	1	7	01	111	01111	5

TABLA 2.5: Código GPO2 $\mathcal{G}_3$ para $i \in \mathbb{Z}_{[0,15]}$.

2.4.3 Importancia práctica

El hecho de que para cualquier variable aleatoria con distribución geométrica exista un código de Golomb que es óptimo, vuelve a los códigos de Golomb muy útiles para resolver ciertos tipos de problemas prácticos. A continuación explicaremos la utilidad de estos códigos en el proceso de codificación de distintos tipos de señales, como por ejemplo audio e imágenes.

Supongamos que tenemos una foto digital y elegimos un pixel al azar. En general, el color y brillo asociado a ese pixel es similar al de los pixeles que lo rodean. En este razonamiento intuitivo se basan los algoritmos de compresión de *modelado predictivo*. En el caso de las imágenes, se utiliza un modelo estadístico para aproximar el valor de un pixel teniendo en cuenta información relativa a los pixeles ya procesados[3]. Le llamamos *predictor* a la función que predice el valor de un pixel utilizando información del modelo. El predictor estima el valor de cada pixel de la imagen que se codifica y, dependiendo de las características particulares de la imagen y de qué tan bueno es el predictor, las estimaciones tienen más o menos diferencia con el valor real de los respectivos pixeles. A la diferencia entre el valor predicho y el valor real de un pixel se le llama *residuo*.

El algoritmo de compresión de imágenes sin pérdida LOCO-I [20] utiliza un predictor con la siguiente característica: los residuos obtenidos se ajustan muy bien a una distribución TSGD (*Two-Sided Geometric Distribution - distribución geométrica a dos lados*) centrada en 0. De acuerdo a esta distribución, la probabilidad de que un residuo entero valga ϵ es proporcional a $\theta^{|\epsilon|}$ con $\theta \in \mathbb{R}_{(0,1)}$. Vimos que para cualquier distribución geométrica existe un código de Golomb óptimo, por lo que es esperable que en el proceso de codificación de una TSGD convenga utilizar alguna variante de los códigos de Golomb. El algoritmo LOCO-I codifica los residuos utilizando un código de Golomb GPO2 con mapeo de Rice [25]. Trabajar con códigos GPO2 permite implementar los procedimientos de codificación y decodificación en una computadora de forma relativamente sencilla y con baja complejidad computacional.

El algoritmo Shorten [21] se utiliza para comprimir señales de audio sin pérdida. Típicamente, una forma de onda está representada por enteros de 16 bits y en general existe correlación entre muestras cercanas en el tiempo. Esta característica permite codificar la señal utilizando un algoritmo de modelado predictivo y codificación de residuos análogo a LOCO-I. Con el predictor utilizado en Shorten los residuos obtenidos también se ajustan a una distribución TSGD y nuevamente se utiliza un código de Golomb GPO2 con mapeo de Rice para comprimir (los residuos de) la señal de audio.

Nuestra descripción de los algoritmos LOCO-I y Shorten es superficial, con el único objetivo de transmitir cómo nos pueden servir los códigos de Golomb en la implementación de codificadores de distintos tipos de señales. Recomendamos leer los respectivos artículos por más detalles.

[3] Nota: los pixeles se procesan y codifican de manera secuencial, en general de izquierda a derecha y de arriba hacia abajo.

Capítulo 3

Codificación de señales multicanal

En este capítulo estudiamos el problema de la codificación de señales multicanal transformadas a través de un proceso de reducción de canales. En la sección 3.1 describimos el sistema de codificación del problema estudiado, presentando los resultados teóricos obtenidos en diferentes casos. En la sección 3.2 explicamos por qué en los siguientes capítulos nos enfocamos en codificar la distribución binomial negativa como consecuencia del problema teórico planteado originalmente.

3.1 Presentación del problema teórico

Consideremos el sistema de codificación presentado en la figura 3.1. Tenemos una fuente que genera una señal X, compuesta por n canales. Esta señal podría ser, por ejemplo, audio, un electroencefalograma o un registro de temperatura en diferentes locaciones. La señal viene dada como una secuencia de vectores de largo n, cuyos componentes toman valores sobre un alfabeto finito $\mathcal{X}$ compuesto de $|\mathcal{X}|$ símbolos distintos.

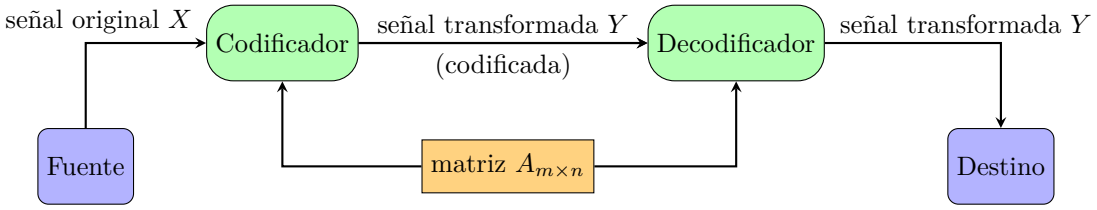

FIGURA 3.1: Sistema de codificación del problema estudiado.

Queremos reducir la cantidad de canales de la señal, creando m canales ($m < n$), cada uno calculado como una combinación lineal de los n canales de la señal original. A modo de ejemplo, si la señal original fuera de audio generada en cinco canales, podríamos querer reducirla a dos canales, de manera que pudiera ser reproducida en un equipo estereofónico.

Suponemos que la transformación lineal está dada por una matriz $A_{m\times n}$, tal que si $x = (x_1, ..., x_n)^T$ es una muestra de la señal original X, la muestra $y = (y_1, ..., y_m)^T$, correspondiente a la señal transformada Y, se obtiene calculando $y = Ax$. En la figura 3.2 presentamos un esquema del proceso de cálculo de la señal transformada a partir de la señal original.

En general existe correlación entre los canales de la señal original X. Además, la proyección sobre un espacio de menor dimensión que resulta de multiplicar por $A_{m\times n}$, también induce correlación

entre los canales de la señal transformada Y. Uno de los objetivos de este proyecto es estudiar si el conocimiento de la señal original X y la matriz $A_{m\times n}$ por parte del codificador puede ser aprovechada para comprimir la señal transformada Y mejor de lo que se comprimiría sin ese conocimiento.

Como se observa en la figura 3.1, asumimos que el decodificador también conoce la matriz $A_{m\times n}$. Este hecho no modifica el comportamiento asintótico del sistema de codificación, ya que antes de comenzar a codificar la señal, el codificador puede describirle la matriz al decodificador utilizando una cantidad finita de bits[1].

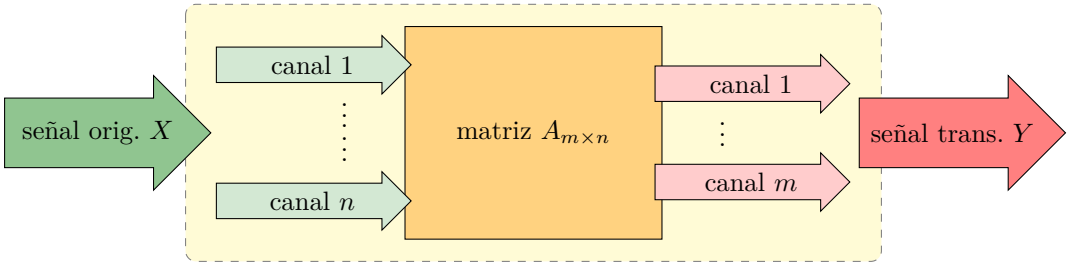

FIGURA 3.2: Esquema del cálculo de la señal transformada Y a partir de la señal original X.

Comenzamos considerando un caso particular sencillo que nos permitiera familiarizarnos con el problema estudiado. Luego, en la sección 3.1.2, avanzamos hacia el caso más general y de mayor complejidad.

3.1.1 Primer escenario: distribución de probabilidad conocida

En este primer escenario vamos a comparar los dos sistemas de codificación presentados en la figura 3.3. Nos interesa comprimir la señal Y que obtenemos al aplicarle la función $f : \mathcal{X}^n \to \mathcal{Y}^m$ a la señal X. Observar que el escenario estudiado incluye el caso en el que Y es una transformación lineal de X.

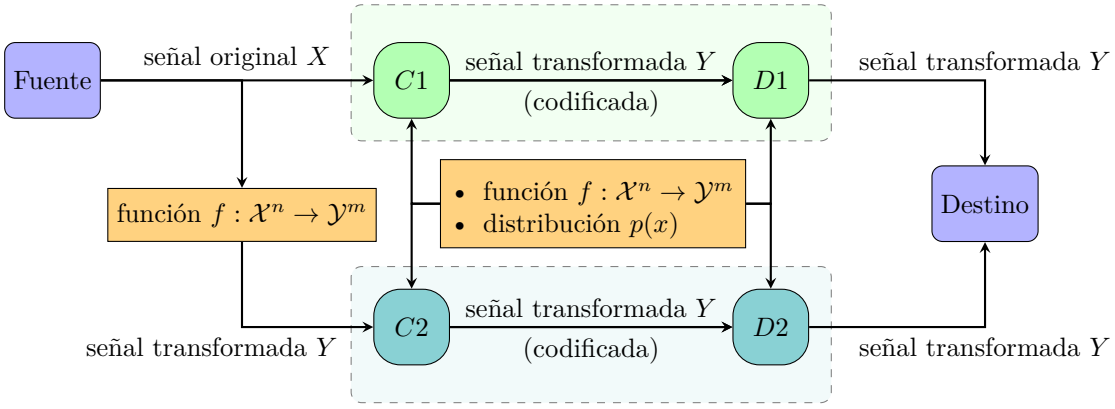

FIGURA 3.3: Sistemas de codificación comparados. El primer sistema está dentro del rectángulo verde punteado y el segundo dentro del rectángulo azul.

Tanto el codificador y decodificador del primer sistema de codificación ($C1$ y $D1$), como el codificador y decodificador del segundo sistema ($C2$ y $D2$), conocen la función f y la distribución

[1]Esto ocurre siempre que estemos bajo la hipótesis de que las entradas de la matriz $A_{m\times n}$ se representan con precisión finita.

de probabilidad $p(x)$ de la señal original X. La única diferencia entre ambos sistemas es que el codificador $C1$ conoce la señal original y la transformada[2], mientras que el codificador $C2$ solo conoce la señal transformada.

Vamos a probar que utilizando el segundo sistema de codificación es posible comprimir la señal transformada tanto como la podemos comprimir utilizando el primer sistema. Dicho de otra manera, en el escenario planteado conocer la señal original no le aporta a un codificador elementos que le permitan reducir la esperanza del largo medio de código al comprimir la señal transformada. Formalizamos este planteo en la proposición 3.1.1.

Proposición 3.1.1. Sea X una señal de dimensión n cuyos componentes toman valores sobre el alfabeto $\mathcal{X}$ siguiendo una distribución de probabilidad $p(x)$. Sea $f : \mathcal{X}^n \to \mathcal{Y}^m$ una función tal que $Y = f(X)$ es una señal de dimensión m cuyos componentes toman valores sobre el alfabeto $\mathcal{Y}$. Se considera un sistema de codificación para la señal Y donde tanto el codificador como el decodificador conocen la función f y la distribución de probabilidad $p(x)$. En este sistema, la esperanza de largo de código óptimo cuando el codificador conoce la señal original X es la misma que cuando no la conoce.

Demostración. La señal Y es función de la señal X, con $f(X) = Y$. Por lo tanto, conociendo f y la distribución de probabilidad $p(x)$ de la señal original X, podemos calcular la distribución de probabilidad $p(y)$ de la señal transformada Y. Esto implica que el codificador y el decodificador conocen $p(y)$. Conociendo la distribución de probabilidad de la señal que se quiere comprimir, es posible crear un código óptimo para la misma, utilizando por ejemplo el algoritmo de Huffman presentado en la sección 2.3. Utilizar un código óptimo para codificar Y nos garantiza que la esperanza de largo de código obtenida es mínima. □

3.1.2 Caso general

Investigando el problema planteado encontramos varias publicaciones que estudian escenarios similares al nuestro. Slepian y Wolf [26] analizaron distintos sistemas de codificación para fuentes correlacionadas, similares al que presentamos en la figura 3.4.

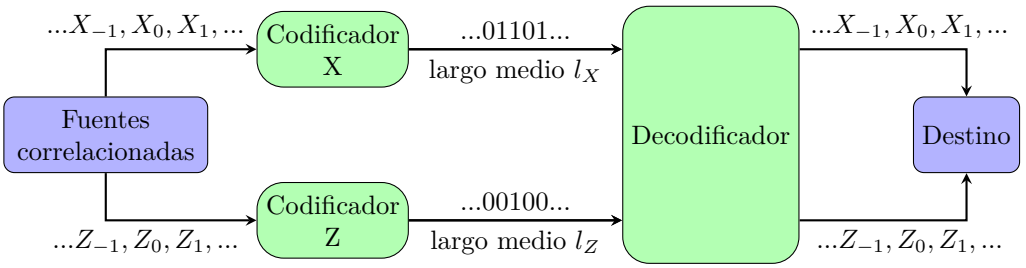

FIGURA 3.4: Uno de los sistemas de codificación para fuentes correlacionadas estudiado por Slepian y Wolf en [26].

En esos sistemas se generan secuencias de símbolos correlacionadas $...X_{-1}, X_0, X_1, ...$ y $...Z_{-1}, Z_0, Z_1, ...$, asociadas a un par de variables aleatorias discretas X y Z con distribución conjunta $p(x, z)$. En el ejemplo de la figura 3.4 el codificador para cada fuente opera sin conocimiento de la otra fuente, pero el decodificador recibe las cadenas de bits que salen de ambos codificadores. En [26] se determinan los largos medios de código mínimos l_X y l_Z necesarios para codificar las secuencias, para este y otros sistemas de codificación similares.

[2] $C1$ conoce la señal transformada porque la puede calcular aplicando la función f a la señal original.

Los resultados de Slepian y Wolf no aplican específicamente al problema que nosotros queremos estudiar. En [26] interesa codificar a las dos variables aleatorias correlacionadas. En nuestro caso tenemos dos señales correlacionadas, X e Y, pero solamente nos interesa codificar Y.

Wyner y Ziv [27] estudiaron la codificación de fuente en un escenario en el que el decodificador cuenta con *información adicional* (ellos le llaman *side information*) sobre la fuente que se desea codificar. Esta información adicional podría ser, por ejemplo, una variable aleatoria dependiente de la variable asociada a la fuente que interesa codificar.

Wyner y Ziv llegan a la conclusión de que, cuando el decodificador tiene acceso a la información adicional, el conocimiento de esta misma información por parte del codificador permite lograr una menor esperanza de largo medio de código. Pero en nuestro caso de estudio, solamente el codificador tiene acceso a la información adicional, por lo que no podemos aplicar el mismo razonamiento de Wyner y Ziv.

En [28] Gallager estudia la codificación de fuente con información adicional en el marco de la codificación universal. Si bien esta publicación solamente tiene una relación tangencial con nuestro caso de estudio, en un párrafo se explica por qué no se analizan situaciones donde solo el codificador puede acceder a la información adicional. Utilizando argumentos similares, pudimos llegar a una conclusión respecto al problema estudiado, presentada en la siguiente proposición.

Proposición 3.1.2. Sea X una señal de dimensión n cuyos componentes toman valores sobre el alfabeto $\mathcal{X}$. Sea Y la señal de dimensión m cuyos componentes toman valores sobre el alfabeto $\mathcal{Y}$, obtenida al aplicar la transformación lineal dada por la matriz $A_{m\times n}$ sobre X. Se considera un sistema de codificación para la señal Y donde tanto el codificador como el decodificador conocen la matriz $A_{m\times n}$. En este sistema, la esperanza de largo de código óptimo cuando el codificador conoce la señal original X es la misma que cuando no la conoce.

Demostración. En el sistema de codificación estudiado, representado en la figura 3.1, el decodificador define un mapeo entre palabras de código y secuencias de símbolos de Y. Esto se cumple siempre, sin importar la forma del código utilizado. El mapeo está fijo, definido a priori, ya que el decodificador no tiene acceso a la señal X. Entonces, el codificador, aun conociendo la señal X, lo mejor que puede hacer es codificar mapeando una secuencia de símbolos de Y a una palabra de código que le permita al decodificador reconstruir la misma secuencia. Como el decodificador no tiene acceso a la señal X, al codificador no le sirve de nada la información aportada por la señal X. Es decir que la esperanza de largo de código no cambia si el codificador tiene acceso a la señal X. □

La proposición 3.1.2 también es válida para los casos en los que la señal transformada Y se obtiene al aplicarle una función arbitraria a la señal original X, no necesariamente una transformación lineal.

3.2 Aplicación práctica del problema

En el final de la sección anterior llegamos a la conclusión de que conocer la señal original X en el codificador en general no sirve para codificar la señal transformada Y con una menor esperanza de largo medio de código. Pero en la práctica puede ocurrir que tengamos herramientas para codificar la señal original pero no sepamos cómo codificar la señal transformada.

Consideremos, por ejemplo, una señal X con dos componentes X_1 y X_2 independientes que siguen una distribución geométrica $X_1, X_2 \sim Geo(p)$ con $p \in \mathbb{R}_{(0,1)}$. Supongamos que obtenemos la señal Y aplicando la transformación lineal $Y = X_1 + X_2$. En el apéndice A demostramos que en este caso se cumple $Y \sim BN(2, p)$, es decir que Y sigue una distribución binomial negativa con parámetros 2 y p. A continuación, definimos formalmente esa distribución.

Definición 3.2.1. Consideramos una secuencia de ensayos de Bernoulli independientes con idéntica probabilidad de éxito $p \in \mathbb{R}_{(0,1)}$. Definimos la variable aleatoria $Y \in \mathbb{Z}_{\geq 0}$ como el número de ensayos exitosos realizados antes de que ocurran los primeros dos fallos. Decimos que la variable aleatoria Y sigue una *distribución binomial negativa* con parámetros 2 y p, y nos referimos a ella utilizando la notación $Y \sim BN(2, p)$. La distribución de probabilidad de Y está dada por

$$\mathrm{P}(Y = i) = (1-p)^2(i+1)p^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{3.1}$$

En lo que resta del informe, cuando nos referimos a la distribución binomial negativa estamos hablando de la distribución binomial negativa con parámetros 2 y p.

Sigamos con el ejemplo. Nuestro objetivo es comprimir la señal transformada $Y \sim BN(2, p)$, pero no conocemos un código óptimo para la distribución binomial negativa. En cambio, sabemos que los códigos de Golomb, presentados en 2.4, codifican de manera óptima las distribuciones geométricas. Por lo tanto, en este caso conocemos un código óptimo para las dos componentes de la señal X pero no sabemos cómo codificar la señal Y. Entonces podríamos codificar las componentes $X1$ y $X2$ por separado, y el decodificador, que conoce la matriz asociada a la transformación lineal, podría decodificar X_1 y X_2 y luego aplicar la transformación lineal para obtener la señal Y.

El algoritmo descripto en el párrafo anterior permite codificar la señal transformada Y, pero su codificador transmite más información de la necesaria (nos interesa solo el resultado de la suma, no el valor de cada sumando por separado). Además, el codificador utiliza información de la señal original, y por la proposición 3.1.2 sabemos que debe existir otro algoritmo de compresión igual de bueno en el que el codificador no disponga de ninguna información adicional. Por lo tanto, nos planteamos como problema práctico del proyecto implementar un codificador para la distribución binomial negativa que esté lo más cerca posible de ser óptimo. Implementamos dos codificadores, los cuales presentamos en los capítulos 4 y 5.

En la práctica, conocer el código óptimo de una distribución binomial negativa puede ser útil para implementar algoritmos de compresión de modelado predictivo, como los que presentamos en 2.4.3. Por ejemplo, supongamos que tenemos una señal X' con dos componentes independientes X_1' y X_2'. Además, supongamos que tenemos un predictor lineal para las componentes, tal que los respectivos residuos X_1 y X_2 se modelan bien con la misma distribución geométrica, $X_1, X_2 \sim Geo(p)$. Para codificar la señal $Y' = X_1' + X_2'$, podríamos utilizar el código óptimo de la binomial negativa para codificar la suma de los residuos, dada por $Y = X_1 + X_2$, que arriba vimos cumple $Y \sim BN(2, p)$.

Es importante aclarar que no siempre es posible codificar la suma de señales utilizando una técnica que consista en codificar la suma de los residuos. Para que esa técnica funcione, el decodificador tiene que ser capaz de reconstruir la suma de las predicciones a partir de la suma de las señales. Esto ocurre, por ejemplo, cuando se usa el mismo predictor lineal fijo para $X1$ y $X2$. En ese caso, la suma de las predicciones da lo mismo que la predicción aplicada a la suma de las muestras, que es lo que el decodificador es capaz de calcular. Debemos destacar que la utilización de predictores lineales es bastante común, utilizándose por ejemplo en algoritmos de compresión de audio sin pérdida, tales como Shorten [21] y FLAC [2].

No encontramos bibliografía específica referida a la compresión de la distribución binomial negativa, pero sí encontramos varios artículos que estudian técnicas de compresión para variantes de la distribución geométrica y la Laplaciana, su versión continua. En [29], por ejemplo, se obtuvieron los códigos óptimos para pares de variables aleatorias con distribución geométrica. En la sección 2.4.3 presentamos los algoritmos LOCO-I [20] y Shorten [21], los cuales codifican distribuciones TSGD con variantes de los códigos de Golomb.

Capítulo 4

Codificador GolombBN

En este capítulo desarrollamos el codificador GolombBN, el cual tiene un algoritmo de implementación relativamente sencilla y sirve como alternativa al codificador T, presentado en el capítulo 5. En la sección 4.1 presentamos al codificador GolombBN, sus parámetros y funcionamiento, incluyendo pseudocódigos de los algoritmos de codificación y decodificación. En las secciones 4.2 y 4.3 explicamos cómo obtener los parámetros del codificador a partir del parámetro p de la distribución binomial negativa que gobierna la fuente a codificar. Por último, en la sección 4.4 desarrollamos los cálculos que permiten obtener el largo medio de código resultante al codificar una variable aleatoria con distribución binomial negativa.

4.1 Descripción

El codificador GolombBN es utilizado para comprimir una fuente que genera enteros no negativos de acuerdo a una variable aleatoria $Y \sim BN(2, p)$ con $p \in \mathbb{R}_{(0,1)}$. En la definición 3.2.1 vimos que la probabilidad de que esta fuente genere cierto entero i está dada por

$$f_Y(i) = \mathrm{P}(Y = i) = (1-p)^2(i+1)p^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{4.1}$$

El codificador GolombBN es una versión modificada del codificador de Golomb. Como vimos en la sección 2.4, para cualquier variable aleatoria $X \sim Geo(\theta)$, con $\theta \in \mathbb{R}_{(0,1)}$, existe un código de Golomb que es óptimo. En la definición 2.4.1 vimos que la función de probabilidad para la variable aleatoria X es

$$f_X(j) = \mathrm{P}(X = j) = (1-\theta)\theta^j\,, \quad j \in \mathbb{Z}_{\geq 0}. \tag{4.2}$$

Las modificaciones realizadas al codificador de Golomb surgen al considerar las particularidades que tiene la distribución binomial negativa respecto a la geométrica. Ambas distribuciones son analizadas y comparadas en el apéndice B.

Definimos un parámetro $\lambda \in \mathbb{Z}_{>0}$ y una función biyectiva $Perm : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$, con $Perm(j) = j$ para todo $j \in \mathbb{Z}_{\geq \lambda}$, la cual establece una permutación de los primeros λ valores enteros que puede asumir Y. Utilizamos la función $Perm$ para reordenar los enteros en orden decreciente de probabilidad, lo que permite replicar una característica de la distribución geométrica en la binomial negativa. En la sección 4.2 desarrollamos el cálculo del parámetro λ y especificamos la función $Perm$. Ambos dependen únicamente del parámetro p de la distribución.

El codificador GolombBN utiliza códigos GPO2. En la sección 2.4.1.1 vimos que esto simplifica la implementación de los procesos de codificación y decodificación. El parámetro $l \in \mathbb{Z}_{>0}$ del codificador GolombBN cumple un rol análogo al del parámetro homónimo en el código de Golomb. En la sección 4.3 explicamos cómo calcular el parámetro l a partir del parámetro p de la distribución.

4.1.1 Algoritmo de codificación

Definimos GBN(i) como la palabra de código con la que el codificador GolombBN codifica un entero $i \in \mathbb{Z}_{\geq 0}$. En la figura 4.1 presentamos el algoritmo de codificación del código GolombBN. Recibe cuatro entradas: i es el entero generado por la fuente que queremos codificar, l y λ son los parámetros del codificador y *perm* es un vector de λ enteros, tal que $perm[x]$ contiene al resultado de calcular $Perm(x)$ para $x \in \mathbb{Z}_{[0,\lambda-1]}$.

entradas: $i \in \mathbb{Z}_{\geq 0}$: entero a codificar
$l \in \mathbb{Z}_{>0}$ y $\lambda \in \mathbb{Z}_{>0}$: parámetros del codificador
perm: vector de largo λ con $perm[x] = Perm(x)$, $x \in \mathbb{Z}_{[0,\lambda-1]}$
salida : GBN(i): palabra de código con la que se codifica i

```
1 if (i < λ) then
2 |  return codificar_Golomb_GPO2(l, perm[i])
3 else
4 |  return codificar_Golomb_GPO2(l, i)
5 end
```

FIGURA 4.1: Algoritmo de codificación del código GolombBN.

Distinguimos dos casos dependiendo del valor de i. Si i es menor que λ (líneas 1 y 2), entonces GBN(i) es igual a $G_l(Perm(i))$, que es la palabra con la que el código de Golomb con parámetro l codifica al entero resultado de calcular $Perm(i)$. Para todos los demás valores de i (líneas 3 y 4), el algoritmo devuelve $G_l(i)$.

La similitud entre el codificador de Golomb y el codificador GolombBN es evidente. Observar que ambos codifican con la misma palabra de código una cantidad infinita de enteros (todos los que son mayores o iguales a λ). Además, el conjunto de palabras de código con que se codifican los enteros en el rango $[0, \lambda - 1]$ es igual para ambos códigos; la única diferencia entre los códigos es que las palabras de ese conjunto están permutadas, de acuerdo a la función *Perm*.

4.1.1.1 Codificación de cadenas Para codificar una cadena de enteros generados por la variable aleatoria Y debemos seguir dos pasos:

- En primer lugar, utilizamos el algoritmo definido en la sección 4.2 para, a partir del parámetro p, obtener el parámetro λ y la función *Perm*. Luego, también a partir de p, calculamos el parámetro l, siguiendo el algoritmo de la sección 4.3. Realizamos este paso una única vez.
- Luego ejecutamos el algoritmo de la figura 4.1, una vez por cada entero de la cadena que nos interesa codificar. Excepto el entero i a codificar, todas las variables de entrada (l, λ y *perm*) son iguales en todas las ejecuciones.

4.1.2 Algoritmo de decodificación

En la figura 4.2 presentamos el algoritmo de decodificación del código GolombBN. Tiene cuatro entradas: *bit_stream* es una cadena de bits de donde se decodifica el entero i que se de devuelve; las otras entradas son los dos parámetros del codificador GolombBN y el vector *perm_inversa*. Este último es un vector de λ enteros, tal que $perm\_inversa[x]$ contiene al entero resultado de calcular $Perm^{-1}(x)$[1] para $x \in \mathbb{Z}_{[0,\lambda-1]}$.

En primer lugar, decodificamos un entero (*i_mapeado*) de la cadena de bits de entrada utilizando el algoritmo de decodificación del código de Golomb con parámetro l. Luego distinguimos dos casos dependiendo del valor de *i_mapeado*. Si es menor que λ (líneas 2 y 3), el algoritmo de decodificación devuelve al entero resultado de calcular $Perm^{-1}(i\_mapeado)$. En cualquier otro caso (líneas 4 y 5) devuelve *i_mapeado*.

entradas: *bit_stream*: cadena de bits a decodificar
$l \in \mathbb{Z}_{>0}$ y $\lambda \in \mathbb{Z}_{>0}$: parámetros del codificador
perm_inversa: vector de largo λ con $perm\_inversa[x] = Perm^{-1}(x)$, $x \in \mathbb{Z}_{[0,\lambda-1]}$
salida : $i \in \mathbb{Z}_{\geq 0}$: entero decodificado

```
1 i_mapeado = decodificar_Golomb_GPO2(l, bit_stream)
2 if (i_mapeado < λ) then
3 |  i = perm_inversa[i_mapeado]
4 else
5 |  i = i_mapeado
6 end
7 return i
```

FIGURA 4.2: Algoritmo de decodificación del código GolombBN.

4.1.2.1 Decodificación de cadenas Para decodificar una cadena de enteros generados por la misma fuente, debemos ejecutar el algoritmo de la figura 4.2 dentro de un ciclo cuya condición de parada es que ya han sido leídos todos los bits de la cadena de entrada.

4.2 Cálculo del parámetro λ y la función *Perm*

En esta sección explicamos cómo calcular el parámetro λ y la función *Perm* asociados al codificador GolombBN. Con este fin, definimos la relación binaria de orden total $\preceq$ sobre los elementos del dominio de Y de la siguiente manera

$$i \preceq j \iff (\ f_Y(i) > f_Y(j) \ ) \quad \vee \quad (\ f_Y(i) = f_Y(j) \quad \wedge \quad i \leq j \ ), \tag{4.3}$$

donde f_Y es la función de probabilidad definida en (4.1). En palabras, $\preceq$ ordena los enteros en orden decreciente de probabilidad, desempatando en caso de igualdad por orden natural de los enteros. Queremos calcular el orden relativo de todos los enteros del dominio de Y de acuerdo a la relación $\preceq$. Más específicamente, para cada $i \in \mathbb{Z}_{\geq 0}$ queremos calcular la función $Perm : Z_{\geq 0} \to Z_{\geq 0}$ definida a continuación

$$Perm(i) = \left| \ k \in Z_{\geq 0} : k \prec i \ \right|. \tag{4.4}$$

$Perm$ es una función biyectiva, porque $\preceq$ es una relación de orden total en el domino de Y, que coincide con el domino de $Perm$. Para poder calcular $Perm$ debemos tener en cuenta las

[1] La función $Perm : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ es invertible porque es biyectiva.

características de la función f_Y, estudiada en el apéndice B. Allí observamos que para todo $p \in \mathbb{R}_{(0,1)}$ se cumple lo siguiente:

- $f_Y(0) = (1-p)^2$
- $f_Y(i) > 0 \ \ \forall i \in \mathbb{Z}_{\geq 0}$
- $\lim_{i\to\infty} f_Y(i) = 0^+$

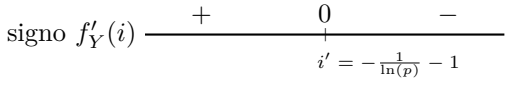

FIGURA 4.3: Signo de $f'_Y(i)$.

- La derivada de f_Y se anula en el punto $i' = -\frac{1}{\ln(p)} - 1$, y f_Y crece cuando $i < i'$ y decrece cuando $i > i'$. En la figura 4.3 presentamos el signo de $f'_Y(i)$.

A continuación planteamos el cálculo realizado para obtener el parámetro $\lambda \in \mathbb{Z}_{>0}$ y la permutación $Perm$. Definimos

$$\begin{aligned}
\lambda &= \min\left\{ \ i \in Z_{>0} : \ 0 \preceq i \ \right\} \\
&\overset{(4.3)}{=} \min\left\{ \ i \in Z_{>0} : \ \ (\ f_Y(0) > f_Y(i)\ )\ \vee\ (\ f_Y(0) = f_Y(i) \quad \wedge \quad 0 \leq i\ ) \ \right\} \\
&= \min\left\{ \ i \in Z_{>0} : \ \ f_Y(0) \geq f_Y(i) \ \right\}. \qquad (4.5)
\end{aligned}$$

Como $f_Y(0) \geq f_Y(\lambda)$, con $\lambda > 0$ por definición, entonces f_Y es decreciente en algún segmento del intervalo $[0, \lambda]$. Por lo tanto, observando el signo de f'_Y sabemos que $i' < \lambda$.

Como $i' < \lambda$ y f_Y es decreciente para $i > i'$, entonces f_Y es decreciente para $i \geq \lambda$. Entonces, considerando el signo de f'_Y y la definición de λ deducimos que, para cualquier par de enteros $i \in [0, \lambda - 1]$ y $j \geq \lambda$, se cumple $f_Y(i) \geq f_Y(j)$. Por lo tanto, tenemos que $Perm(j) = j$ para todo $j \geq \lambda$.

Entonces solamente resta calcular el valor de $Perm$ para los λ enteros del intervalo $[0, \lambda - 1]$. Para ello seguimos el siguiente procedimiento:

- En primer lugar calculamos $\lfloor i' \rfloor$ y $\lceil i' \rceil$. Como $f'_Y(i') = 0$, conociendo el signo de f'_Y sabemos que se debe cumplir lo siguiente: $Perm(\lceil i' \rceil) = 0$ si $f_Y(\lfloor i' \rfloor) < f_Y(\lceil i' \rceil)$ y $Perm(\lfloor i' \rfloor) = 0$ en otro caso.
- Una vez obtenido el i que anula la función $Perm$, consideramos los dos enteros más próximos, $i-1$ e $i+1$. Procedemos dependiendo del valor obtenido al evaluar f_Y en ambos puntos: si $f_Y(i-1) < f_Y(i+1)$ entonces $Perm(i+1) = 1$ y repetimos el procedimiento con $i-1$ e $i+2$; si en cambio $f_Y(i-1) \geq f_Y(i+1)$, entonces $Perm(i-1) = 1$ y repetimos el procedimiento con $i-2$ e $i+1$. Debemos continuar aplicando este algoritmo hasta haber calculado $Perm$ para todos los enteros del intervalo $[0, \lambda - 1]$, donde λ está dado por (4.5).

Como ejemplo, en la figura 4.4 presentamos las gráficas de $f_Y(i)$ y $Perm(i)$ tomando $p = 0.90$, para $i \in \mathbb{Z}_{[0,60]}$. Para ese valor del parámetro p, aplicando el cálculo definido en (4.5) obtenemos $\lambda = 34$. Observar que para cualquier $i \in \mathbb{Z}_{[34,60]}$ se cumple $Perm(i) = i$, y lo mismo ocurre para todos los enteros del dominio de f_Y que no aparecen en la gráfica. Para los enteros menores que λ, observamos que $Perm(i)$ toma valores más bajos para los enteros cercanos a i' y valores más altos para los enteros cercanos a 0 y λ. Esto es coherente con el procedimiento descrito en el párrafo anterior.

En la figura 4.5 presentamos las gráficas de $f_X(i)$, que recordamos está definida en (4.2), tomando $\theta = 0.90$ y $f_Y(Perm^{-1}(i))$ tomando $p = 0.90$, para $i \in \mathbb{Z}_{[0,60]}$. También está graficada nuevamente $f_Y(i)$, pero únicamente para $i \in \mathbb{Z}_{[0,33]}$, ya que se cumple $f_Y(i) = f_Y(Perm^{-1}(i))$ para todo $i \geq \lambda = 34$.

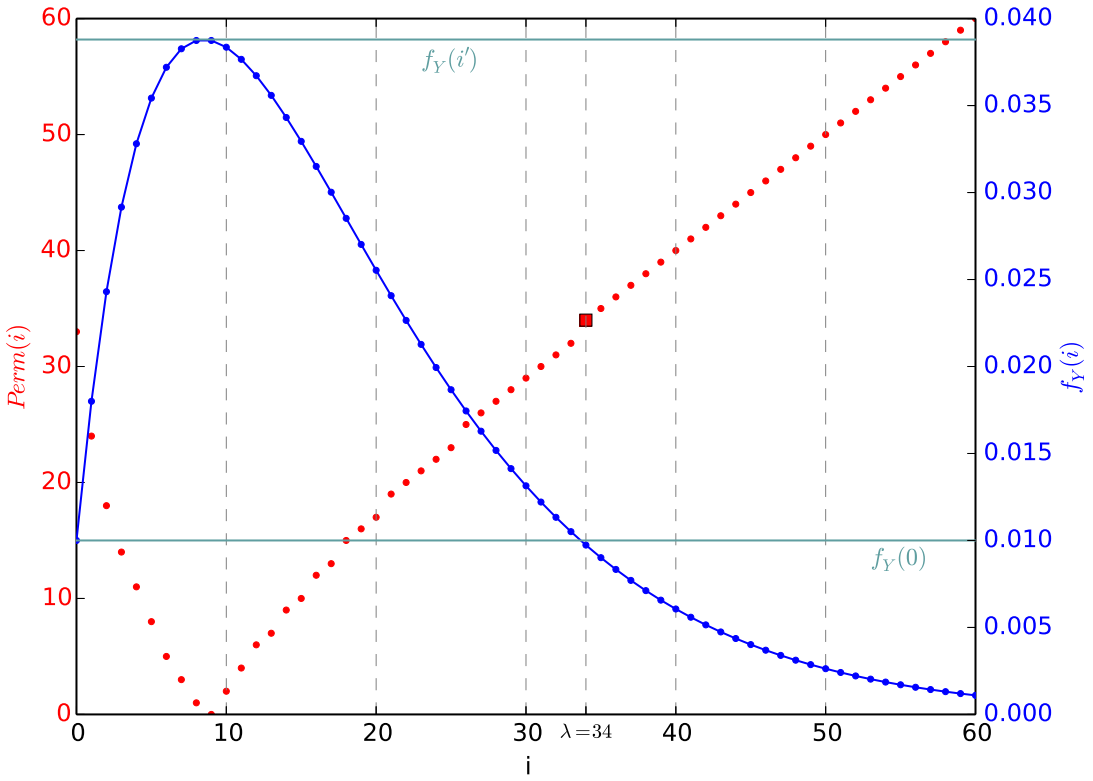

FIGURA 4.4: Gráficas de las funciones $f_Y(i)$, en azul con la escala a la derecha, y $Perm(i)$, en rojo con la escala a la izquierda, tomando $p = 0.90$ para $i \in \mathbb{Z}_{[0,60]}$. Marcamos $f_Y(i')$ y $f_Y(0)$ con líneas celestes y $Perm(\lambda)$ con un cuadrado rojo.

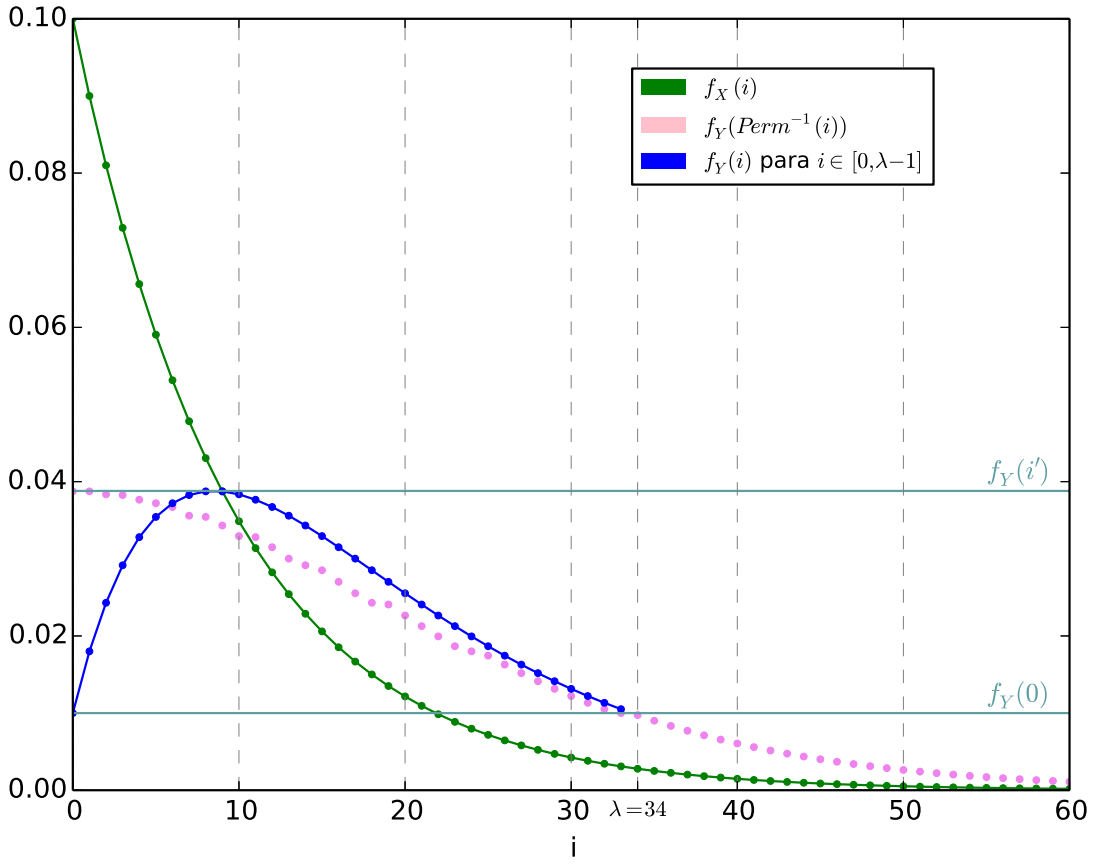

FIGURA 4.5: Gráficas de las funciones $f_X(i)$ tomando $\theta = 0.90$ y $f_Y(Perm^{-1}(i))$ tomando $p = 0.90$, para $i \in \mathbb{Z}_{[0,60]}$, y $f_Y(i)$ tomando $p = 0.90$ para $i \in \mathbb{Z}_{[0,33]}$. Marcamos $f_Y(i')$ y $f_Y(0)$ con líneas celestes.

Observando la figura 4.5 queda más claro el razonamiento que hay detrás del algoritmo de codificación del código GolombBN, presentado en la figura 4.1. Al aplicar el mapeo definido por la función $Perm$ logramos que los enteros a los que f_Y les asigna mayor probabilidad sean codificados con una menor cantidad de bits, de forma análoga a lo que ocurre cuando codificamos una variable aleatoria con distribución geométrica con un código de Golomb. De esta forma, al codificar la variable aleatoria Y con el código GolombBN, logramos obtener un largo de código menor que si la codificáramos directamente con el código de Golomb (es decir, sin aplicar el mapeo). En la figura 4.5 se observa que f_X y $f_Y(Perm^{-1})$ son decrecientes en los enteros no negativos, por lo que tanto al codificar X con un código de Golomb, como Y con un código GolombBN, se utilizan palabras de código más largas para codificar enteros con menor probabilidad.

En la tabla 4.1 mostramos los valores de λ obtenidos para un conjunto representativo de valores del parámetro p. También se muestran los valores de i' (punto máximo de f_Y, en donde se anula su derivada) y de $f_Y(0)$, para agregar información sobre la función de distribución de probabilidad. Observar que el parámetro λ se incrementa a medida que p se acerca a 1. Esto tiene consecuencias en la implementación del codificador GolombBN, ya que mientras más cerca esté p de 1, más memoria será necesaria para almacenar a los vectores *perm* y *perm_inversa*.

p	λ	i'	$f_Y(0)$
0.50	1	0.44	0.25
0.60	3	0.96	0.16
0.70	6	1.80	0.09
0.80	12	3.48	0.04
0.90	34	8.49	0.01
0.95	88	18.50	0.0025
0.96	117	23.50	0.0016
0.97	169	31.83	0.0009
0.98	279	48.50	0.0004
0.99	644	98.50	0.0001

Tabla 4.1: λ, i' y $f_Y(0)$ obtenidos para un conjunto representativo de valores de p. Marcamos la fila asociada al p tomado en el ejemplo de las figuras 4.4 y 4.5.

4.3 Cálculo del parámetro l

En esta sección explicamos cómo calcular el parámetro l del codificador GolombBN. Como vimos, para codificar cualquier entero i devuelto por Y, el codificador GolombBN calcula el código de Golomb de parámetro l del entero $Perm(i)$.

Uno de los objetivos del proyecto es crear un codificador para la variable aleatoria Y con el menor largo de código posible. El resultado obtenido es el codificador T, presentado en el capítulo 5. Con el codificador GolombBN, en cambio, nuestro objetivo es tener un codificador de implementación relativamente sencilla y de bajo costo de cómputo. Por lo tanto, es importante aclarar que no buscamos calcular el parámetro l que minimice exactamente el largo de código, sino una aproximación que sea fácil de calcular con herramientas conocidas.

Debido a la similitud que hay entre el codificador de Golomb y el codificador GolombBN, decidimos utilizar una técnica conocida (ver, por ejemplo, [30]), con la cual se estima el parámetro del código de Golomb a partir de parámetros de la distribución geométrica de la variable a codificar. El resultado establece que para una fuente geométrica con media μ, el parámetro del codificador

GPO2 óptimo es $l = 2^{k^*}$, con

$$k^* = \max\left\{ 0, 1 + \left\lfloor \log_2\left(\frac{\log(\Phi - 1)}{\log(\frac{\mu}{\mu+1})}\right)\right\rfloor \right\}, \tag{4.6}$$

donde $\Phi = (1 + \sqrt{5})/2$ es la razón áurea.

En nuestro caso, en vez de θ tomamos el parámetro p de la distribución binomial negativa. Por lo tanto, como p está dado, solamente nos resta calcular μ para poder obtener k^* a través de (4.6).

Para obtener la media de los primeros λ enteros debemos aplicar la siguiente fórmula

$$\sum_{i=0}^{\lambda-1} i \cdot \mathrm{P}(Y = Perm^{-1}(i)) \overset{(4.1)}{=} (1-p)^2 \sum_{i=0}^{\lambda-1} i \cdot (Perm^{-1}(i) + 1) p^{Perm^{-1}(i)}. \tag{4.7}$$

Para obtener la media de los enteros restantes aplicamos la fórmula

$$\sum_{i=\lambda}^{\infty} i \cdot \mathrm{P}(Y = i) = \frac{p^\lambda}{1-p}\Big(\lambda(\lambda-1)p^2 - 2(\lambda^2-1)p + \lambda(\lambda+1)\Big), \tag{4.8}$$

desarrollada en el apéndice D.

Finalmente, obtenemos μ al sumar los resultados de las fórmulas (4.7) y (4.8).

4.4 Largo medio de código

En esta sección explicamos cómo calcular el largo medio de código obtenido al codificar una fuente con variable aleatoria $Y \sim BN(2, p)$ con $p \in \mathbb{R}_{(0,1)}$, con el código GolombBN. Como se observa en el algoritmo de la figura 4.1, cuando el entero i a codificar es menor que el parámetro λ, i se codifica con el código de Golomb de parámetro l asociado al entero $Perm(i)$. Por lo tanto, la contribución al largo medio de los enteros menores que λ está dada por la ecuación

$$\begin{aligned}\sum_{i=0}^{\lambda-1} \mathrm{P}(Y = i) \cdot l_{GBN}(i) &\overset{(4.1)}{=} (1-p)^2 \sum_{i=0}^{\lambda-1} (i+1) p^i \cdot l_{G_l}(Perm(i)) \\ &= (1-p)^2 \sum_{i=0}^{\lambda-1} (i+1) p^i \cdot (Perm(i) \text{ div } 2^k + k + 1),\end{aligned} \tag{4.9}$$

donde $l_{G_l}(Perm(i))$ es el largo de la palabra de código con la que el codificador de Golomb de parámetro l codifica el entero $Perm(i)$. En la sección 4.3 vimos que el parámetro l es de la forma $l = 2^k$ para cierto $k \in \mathbb{Z}_{\geq 0}$, lo que, como vimos en 2.3.1.1, implica que $l_{G_l}(i) = l_{\mathcal{G}_k}(i) = i \text{ div } 2^k + k + 1$. Aplicamos esta fórmula en la segunda igualdad de la ecuación 4.9.

En el algoritmo de la figura 4.1 vimos que los enteros mayores o iguales que λ se codifican con su respectivo código de Golomb de parámetro l. Para calcular la contribución al largo medio de esos enteros, partimos el espacio en l clases de congruencia (una asociada a cada entero j, con $\lambda \leq j < \lambda + l$). A continuación planteamos la ecuación que permite calcular la contribución al largo medio de todos los enteros i, $i \geq \lambda$, tales que $i \equiv j \pmod{l}$, que son todos los enteros que están en la misma clase de congruencia módulo l que j

$$\sum_{\substack{i=\lambda \\ i\equiv j \pmod{\beta}}}^{\infty} \mathrm{P}(Y=i)\cdot l_{GBN}(i) = \sum_{k=0}^{\infty} \mathrm{P}(Y=j+lk)\cdot l_{GBN}(j+lk). \tag{4.10}$$

En el apéndice E desarrollamos los cálculos de la ecuación (4.10) y luego de agrupar términos obtenemos una función cuyo valor depende únicamente de p, l y j, la cual llamamos $LMCC_{GBN}$ (por "largo medio de la clase de congruencia") y que presentamos a continuación

$$\begin{aligned} LMCC_{GBN}(p,l,j) = \frac{(1-p)^2 p^j}{(1-p^l)^3}\Big(& (l_{G_l}(j)-1)(j-l+1)p^{2l} \\ & + (j+l+1+l_{G_l}(j)\cdot(-2j+l-2))p^l \\ & + l_{G_l}(j)(j+1)\Big). \end{aligned} \tag{4.11}$$

Si calculamos la función (4.11) para cada uno de los l enteros del intervalo $[\lambda, \lambda+l-1]$, y luego sumamos los resultados, obtenemos el largo medio de código para todos los enteros mayores o iguales que λ. Observar que nuevamente tenemos $l_{G_l}(j) = l_{\mathcal{G}_k}(j) = j \text{ div } 2^k + k + 1$.

En el algoritmo de la figura 4.6 resumimos los pasos necesarios para calcular el largo medio del código GolombBN para la variable aleatoria Y. Este algoritmo tiene las mismas entradas que el algoritmo de codificación: el parámetro p de la distribución Y, los dos parámetros del codificador GolombBN y el vector *perm*, que permite calcular la función $Perm$ sobre los primeros λ enteros.

entradas: $p \in \mathbb{R}_{(0,1)}$: parámetro de la variable aleatoria Y
$l \in \mathbb{Z}_{>0}$ y $\lambda \in \mathbb{Z}_{>0}$: parámetros del codificador
perm: vector de largo λ con $perm[x] = Perm(x)$, $x \in \mathbb{Z}_{[0,\lambda-1]}$
salida : *largo_medio_código*

```
   // ecuación (4.9) - líneas 1-4
 1 largo_medio_cabeza = 0
 2 foreach i ∈ {0, ..., λ − 1} do
 3 |  largo_medio_cabeza  += probY(p, i) · largo_código_Golomb_GPO2(l, perm[i])
 4 end
 5 largo_medio_cola = 0
 6 foreach j ∈ {λ, ..., λ + l − 1} do
   |  // ecuación (4.11) - línea 7
 7 |  largo_medio_cola  += LMCC_GBN(p, l, j)
 8 end
 9 return  (largo_medio_cabeza + largo_medio_cola)
```

FIGURA 4.6: Algoritmo para calcular el largo medio del código GolombBN.

En las primeras cuatro líneas del algoritmo obtenemos la contribución al largo medio para los primeros λ enteros, de la forma planteada en (4.9). A continuación, obtenemos la contribución al largo largo medio correspondiente al resto de enteros (líneas 6-8). Para ello, mediante (4.11) calculamos $LMCC_{GBN}$ para cada uno de los l enteros en el intervalo $[\lambda, \lambda+l-1]$. Finalmente, el algoritmo devuelve la suma de los dos valores calculados previamente (línea 9).

Capítulo 5

Codificador T

En este capítulo describimos en detalle el codificador T implementado en el proyecto. En la sección 5.1 presentamos el codificador, explicamos cómo calcular sus parámetros y detallamos los algoritmos de codificación y decodificación, incluyendo sus pseudocódigos. En la sección 5.2 exponemos un ejemplo que muestra cómo codificar (y luego decodificar) enteros de una fuente en la práctica. Por último, en la sección 5.3 calculamos el largo medio de código para una variable aleatoria con la distribución que nos interesa codificar, la binomial negativa.

5.1 Descripción

El objetivo del codificador T es comprimir de la mejor forma posible una fuente que genera enteros no negativos de acuerdo a una variable aleatoria $Y \sim BN(2, p)$ con $p \in \mathbb{R}_{(0,1)}$. Al definir la distribución binomial negativa en 3.2.1, vimos que en este caso la probabilidad de que la fuente genere cierto entero i está dada por

$$\mathrm{P}(Y = i) = (1-p)^2(i+1)p^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{5.1}$$

En la sección 2.4 presentamos los códigos de Golomb y comentamos que en [18] se demuestra que para cualquier distribución geométrica existe un código de Golomb que es óptimo. En esa publicación se explica que el algoritmo de Huffman puede ser utilizado de manera indirecta para probar la optimalidad de un código para un alfabeto infinito si uno primero intuye la forma que debe tener el código. En ese concepto se basaron Gallager y Van Voorhis para probar la optimalidad de los códigos de Golomb. Nosotros seguimos un razonamiento similar para implementar el codificador T.

Nos interesa saber qué forma tiene el código óptimo para la binomial negativa. Para ello calculamos el código de Huffman para un vector $\mathbf{p}$ de largo $n+1$, donde n es un parámetro a definir, con las siguientes probabilidades

$$p_i = \begin{cases} \mathrm{P}(Y = i), & \text{si } 0 \leq i < n, \\ \sum\limits_{j=n}^{\infty} \mathrm{P}(Y = j), & \text{si } i = n. \end{cases} \tag{5.2}$$

Estas probabilidades representan una distribución binomial negativa realizando un truncamiento en el último entero. El entero n acumula las probabilidades de todos los enteros mayores o iguales que n.

La intención es tomar un n lo suficientemente grande, manteniendo el error computacional que ocurre al calcular las probabilidades dentro de cierto límite. En nuestra implementación tomamos n de la siguiente manera

$$n = \min\left\{ i \in Z_{>0} : \left| \frac{p^i}{p^{i-1}} - p \right| > 1 \times 10^{-10} \right\}. \tag{5.3}$$

Consideramos un conjunto representativo de valores del parámetro p e implementamos un programa que para cada p lleva a cabo los siguientes pasos:

1. Calcula n con la fórmula (5.3).
2. Para el n obtenido en el paso 1, calcula el vector $\mathbf{p}$ cuyas probabilidades definimos en (5.2).
3. Calcula el código de Huffman para el vector $\mathbf{p}$ obtenido en el paso 2.

Luego nos dedicamos a estudiar las características de los códigos de Huffman calculados. Analizamos el largo de las palabras de código obtenidas en cada caso, buscando patrones de repeticiones de series de palabras del mismo largo. A continuación explicamos el proceso con un ejemplo.

rango	largo rango	#series	#palabras c/serie	largo palabras
0	1	1	1	9
1 - 3	3	1	3	8
4 - 10	7	1	7	7
11 - 44	34	1	34	6
45 - 73	29	1	29	7
74 - 199	46	2	23	8[74,96] – 9[97,119]
120 - 159	40	2	20	10[120,139] – 11[140,159]
160 - 180	21	1	21	12
181 - 256	76	4	19	13[181,199] – 16[238,256]
257 - 276	20	1	20	17
277 - 870	594	33	18	18[277,294] – 50[853,870]
$\boldsymbol{\alpha = 871}$ - 17598	16728	984	$\boldsymbol{\beta = 17}$	51[871,887] – 1034[17582,17598]
17599 - 17616	18	1	18	1035
17617 - 17633	17	1	17	1036
17634 - 17649	16	1	16	1037
17650 - 17666	17	1	17	1038
17667 - 17684	18	1	18	1039
17685 - 17692	8	1	8	1040
17693 - 17693	1	1	1	50

Tabla 5.1: Patrones de series obtenidas para el código de Huffman del vector $\mathbf{p}$, tomando $p = 0.96$ y $n = 17693$. Marcamos el patrón más largo del código, del cual obtenemos los parámetros $\alpha = 871$ y $\beta = 17$.

En la tabla 5.1 detallamos la información relativa al largo de las palabras del código de Huffman del vector $\mathbf{p}$, tomando $p = 0.96$. Para ese valor de p, al aplicar el cálculo de la fórmula (5.3) obtuvimos $n = 17693$.

Para comprender el ejemplo, antes debemos definir algunos términos. Definimos una *serie* como un conjunto de palabras de código de igual largo asociadas a enteros sucesivos. Definimos un *patrón* como un conjunto de series sucesivas[1] que cumple dos condiciones:

- Todas las series del patrón tienen la misma cantidad de palabras de código. Por lo tanto, el largo del rango del patrón es igual a la cantidad de palabras de cada serie multiplicado por la cantidad de series del patrón.
- Dos series sucesivas cualesquiera cumplen lo siguiente: el largo de las palabras de la segunda serie es igual al largo de las palabras de la primera incrementado en una unidad.

Cada fila de la tabla 5.1 tiene la información relativa a un patrón. En la tabla está coloreada la fila asociada al patrón más representativo del código de Huffman, que es el patrón cuyo rango (enteros entre 871 y 17598 inclusive) es el de mayor largo (tiene 16728 palabras de código, un 95% del total). Este patrón tiene 984 series y cada una de esas series tiene 17 palabras. En la última columna de la tabla se muestra el largo de las palabras: las 17 palabras de código de la primera serie (asociadas a los enteros entre 871 y 887 inclusive) tienen largo 51; las 17 palabras de código de la última serie (asociadas a los enteros entre 17582 y 17598 inclusive) tienen largo 1034.

Para cada uno de los valores de p con los que repetimos el experimento, el código de Huffman obtenido siempre tiene la característica observada en el ejemplo: a partir de cierto punto comienza un patrón que se mantiene casi hasta al final, y que contiene a la gran mayoría de palabras del código. Además, el largo de las palabras de código decrece al principio y luego crece, observándose fluctuaciones cerca del final, que se deben al truncamiento realizado en el último entero.

Los experimentos realizados nos permitieron intuir la forma aproximada que debe tener un código óptimo. Pensamos que en el código óptimo, el patrón más representativo que obtuvimos con el código de Huffman se repite hasta el infinito. Esto es algo similar a lo que ocurre con los códigos de Golomb, con la diferencia de que en los códigos de Golomb el patrón comienza en el cero, el primer entero.

Para replicar el comportamiento descripto, nuestro código T debe tener dos parámetros, a los que decidimos llamar α y β. Para cada p, a estos parámetros les asignamos valores asociados al patrón de mayor largo obtenido al calcular el código de Huffman del vector con probabilidades (5.2). En particular, α es igual al índice asociado a la primera palabra de código del patrón y β es igual a la cantidad de palabras que hay en cada serie del patrón. Por lo tanto, en el ejemplo de la tabla 5.1 tenemos $\alpha = 871$ y $\beta = 17$.

Lo que buscamos con el código T es replicar el comportamiento del código de Huffman obtenido pero para un rango infinito de enteros. Por eso es que los parámetros del código T surgen al analizar el patrón más representativo del código de Huffman: consideramos que este patrón representaría el comportamiento del código de Huffman si el mismo pudiera codificar una cantidad infinita de enteros.

En la tabla 5.2 presentamos los resultados obtenidos para un conjunto representativo de valores de p. Mostramos el valor de n obtenido y detallamos la información del patrón más representativo en cada caso. En las últimas dos columnas están los parámetros obtenidos para el codificador T. En la sección 6.2 estudiamos cómo cambian los parámetros α y β conforme p se acerca a 1.

[1] Consideramos que dos series A y B son sucesivas si al último entero de la serie A le sigue el primer entero de la serie B.

p	n	largo rango (% del total)	rango	#palabras serie	α	β
0.50	1075	1015 (94%)	58 - 1072	1	58	1
0.60	1417	1384 (98%)	32 - 1415	1	32	1
0.70	2026	1914 (94%)	107 - 2020	2	107	2
0.80	3242	3060 (94%)	173 - 3232	3	173	3
0.90	6855	6734 (98%)	76 - 6809	7	76	7
0.95	14073	13272 (94%)	678 - 13949	14	678	14
0.96	17693	16728 (95%)	871 - 17598	17	871	17
0.97	23700	22379 (94%)	1158 - 23536	23	1158	23
0.98	35733	26486 (74%)	9060 - 35545	34	9060	34
0.99	71826	58305 (81%)	12975 - 71279	69	12975	69

TABLA 5.2: Obtención de parámetros del código T para un conjunto representativo de valores de p. La tercera, cuarta y quinta columnas muestran información del patrón más largo obtenido al calcular el código de Huffman del vector con probabilidades (5.2). Marcamos la fila asociada al ejemplo de la tabla 5.1.

Una vez que hemos calculado los parámetros α y β, definimos una fuente reducida que genera enteros en el rango $[0, \alpha + \beta - 1]$ de acuerdo a una variable aleatoria Y' con las siguientes probabilidades

$$\mathrm{P}(Y' = j) = \begin{cases} \mathrm{P}(Y = j), & \text{si } 0 \leq j < \alpha, \\ \sum_{k=0}^{\infty} \mathrm{P}(Y = j + \beta k), & \text{si } \alpha \leq j < \alpha + \beta. \end{cases} \tag{5.4}$$

Observamos que la probabilidad de que se genere alguno de los primeros α enteros es la misma en la fuente reducida que en la original. Los restantes β enteros actúan como *supersímbolos*, ya que tienen una probabilidad equivalente a la sumatoria de probabilidades asociadas a enteros pertenecientes a conjuntos disjuntos, donde cada uno de estos conjuntos está formado por un número infinito de enteros de la fuente original. En particular, la probabilidad de que la fuente reducida genere el supersímbolo s equivale a la sumatoria de las probabilidades de todos los enteros $i \geq \alpha$ generados por la fuente original tales que $i \equiv s \pmod{\beta}$, es decir que están en la misma clase de congruencia módulo β que s.

Nos interesa calcular el código de Huffman binario para la fuente reducida definida en (5.4). Para ello debemos aplicar el algoritmo presentado en la sección 2.3, por lo que necesitamos conocer de manera explícita las probabilidades con que la fuente reducida genera cada uno de los $\alpha + \beta$ enteros. La probabilidad de cada uno de los primeros α enteros es función de p y del respectivo entero, y está dada por la ecuación (5.1). Para obtener la probabilidad de los restantes β enteros recurrimos a la siguiente fórmula

$$\mathrm{P}(Y' = j) = (1-p)^2 p^j \frac{(-j + \beta - 1)p^\beta + j + 1}{(1-p^\beta)^2}, \tag{5.5}$$

que derivamos en el apéndice G desarrollando la sumatoria infinita definida en (5.4).

Observar que en el caso particular en que $\alpha = 0$ y $\beta = 1$, la fuente reducida consta de un único símbolo, el entero 0, que tiene probabilidad 1. En consecuencia, el valor de Y' es constante y no es necesario utilizar ningún bit para su codificación. En este caso, el código de Huffman de Y' asigna una palabra de código vacía al entero 0.

5.1.1 Algoritmo de codificación

Definimos T(i) como la palabra de código con la que el codificador T codifica un entero $i \in \mathbb{Z}_{\geq 0}$. En la figura 5.1 presentamos el algoritmo de codificación del código T. Recibe cuatro entradas: i es el entero generado por la fuente que nos interesa codificar, α y β son los parámetros del codificador y *vector_huff* es un vector de largo $\alpha + \beta$, tal que *vector_huff*[i] contiene la palabra del código de Huffman $CH_{Y'}$ asociada al entero i.

```
entradas: i ∈ Z≥0: entero a codificar
          α ∈ Z≥0 y β ∈ Z>0: parámetros del codificador
          vector_huff: vector de palabras de código asociado al código de Huffman CH_Y'
salida  : T(i): palabra de código con la que se codifica i
 1 if (i < α) then
 2 |  return vector_huff[i]
 3 else
 4 |  cociente = (i − α) div β
 5 |  resto = (i − α) mod β
 6 |  return concatenar_palabras( vector_huff[α + resto], codificar_unario(cociente) )
 7 end
```

FIGURA 5.1: Algoritmo de codificación del código T.

Dependiendo del valor de i distinguimos dos casos. Si i es menor que α (líneas 1 y 2), T(i) es la palabra del código de Huffman $CH_{Y'}$ para el entero i. En cualquier otro caso (líneas 3-6), primero calculamos el cociente y resto de la división $(i-\alpha)/\beta$. Luego obtenemos dos palabras de código: la palabra del código $CH_{Y'}$ para el entero $\alpha + resto$, y la codificación unaria del cociente; T(i) es la palabra de código que resulta al concatenar ambas palabras.

5.1.1.1 Codificación de cadenas Para codificar una cadena de enteros generados por la variable aleatoria Y seguimos los siguientes pasos:

- Primero calculamos α y β a partir del parámetro p, de la manera explicada en la sección 5.1, y luego calculamos el código de Huffman para la fuente reducida Y', $CH_{Y'}$. Este paso se realiza una única vez.
- Luego ejecutamos el algoritmo de la figura 5.1, una vez por cada entero de la cadena que queremos codificar. El único valor de entrada que no es constante en todas las ejecuciones del algoritmo es el entero i a codificar. Los dos parámetros del codificador T, así como el vector que representa al código de Huffman calculado, son siempre los mismos.

5.1.2 Algoritmo de decodificación

En la figura 5.2 presentamos el algoritmo de decodificación del código T. Tiene cuatro entradas: *bit_stream* es una cadena de bits de donde se va a decodificar el entero i que se devuelve; las otras entradas son los dos parámetros del codificador T y el código de Huffman para Y', representado como un árbol binario llamado *arbol_huff*.

Para decodificar un entero, en primer lugar debemos recorrer el árbol de Huffman desde la raíz hasta una hoja. En la primera línea del algoritmo obtenemos el nodo raíz del árbol. Luego, mientras no se haya llegado a un nodo hoja (línea 2), continuamos bajando por el árbol siguiendo un camino que depende de los bits leídos de la cadena (líneas 3-8). Cuando llegamos a una hoja del árbol, obtenemos el entero x asociado a la misma (línea 10).

```
entradas: bit_stream: cadena de bits a decodificar
          α ∈ Z≥0 y β ∈ Z>0: parámetros del codificador
          arbol_huff: árbol binario asociado al código de Huffman C_H_Y'
salida  : i ∈ Z≥0: entero decodificado
 1 nodo_actual = obtener_raíz_árbol(arbol_huff)
 2 while not nodo_es_hoja(nodo_actual) do
 3 |   bit_actual = leer_próximo_bit(bit_stream)
 4 |   if (bit_actual = 0) then
 5 |   |   nodo_actual = obtener_hijo_cero_nodo(nodo_actual)
 6 |   else
 7 |   |   nodo_actual = obtener_hijo_uno_nodo(nodo_actual)
 8 |   end
 9 end
10 x = obtener_entero_hoja(nodo_actual)
11 if (x < α) then
12 |   return x
13 else
14 |   cociente = decodificar_unario(bit_stream)
15 |   return (x + β * cociente)
16 end
```

FIGURA 5.2: Algoritmo de decodificación del código T.

Dependiendo del valor de x pueden darse dos casos. Si es menor que α, el algoritmo de decodificación devuelve al mismo entero x (líneas 11 y 12). En cualquier otro caso (líneas 13-15), se decodifica la próxima palabra de código unario de la cadena de bits, almacenándose su valor en la variable *cociente*, y el algoritmo devuelve al entero resultado de calcular $x + \beta * cociente$. Notar que el valor de x obtenido en el proceso de decodificación de un entero i, coincide con el valor $\alpha + resto$ en el proceso de codificación de ese mismo entero i. La variable *cociente* vale lo mismo en ambos casos.

Ya mencionamos que, en el caso particular en que $\alpha = 0$ y $\beta = 1$, el código de Huffman de Y' asigna una palabra de código vacía al único entero generado por la fuente reducida, el 0. Para esa combinación de parámetros, el árbol binario *arbol_huff* utilizado en el algoritmo consta de un único nodo, que es una hoja asociada al entero 0. En ese caso, en el algoritmo de decodificación x siempre vale 0 y, como $\beta = 1$, siempre se devuelve *cociente*.

5.1.2.1 Decodificación de cadenas Si queremos decodificar una cadena de enteros generados por la misma fuente, debemos ejecutar el algoritmo de la figura 5.2 dentro de un ciclo cuya condición de parada es que ya han sido leídos todos los bits de la cadena de entrada.

5.2 Ejemplo

En esta sección utilizamos el codificador T para codificar y luego decodificar una cadena de enteros. En el ejemplo presentado tomamos $p = 0.998$ y, para simplificar las cuentas, asumiremos que para ese p se obtienen $\alpha = 5$ y $\beta = 3$, aunque los valores de α y β son mucho más grandes en la práctica[2].

[2]En la sección 6.2 graficamos los parámetros del codificador T obtenidos para un conjunto representativo de valores de p.

5.2.1 Codificación

Vamos a aplicar los pasos detallados en la sección 5.1.1.1. Para $\alpha = 5$ y $\beta = 3$ la variable Y' queda definida como

$$\mathrm{P}(Y' = j) = \begin{cases} (1-p)^2(j+1)p^j, & \text{si } 0 \leq j < 5, \\ (1-p)^2 p^j \frac{(2-j)p^3+j+1}{(1-p^3)^2}, & \text{si } 5 \leq j < 8, \end{cases} \tag{5.6}$$

donde hemos utilizado la fórmula (5.5) para evaluar las probabilidades de los supersímbolos.

En la tabla 5.3 detallamos todos los valores del vector de probabilidades, así como el código obtenido al aplicarle el algoritmo de Huffman. Llamamos $CH_{Y'}(j)$ a la palabra de código asociada al entero j y $l_{CH_{Y'}}(j)$ a su largo.

j	$\mathrm{P}(Y' = j)$	$l_{CH_{Y'}}(j)$	$CH_{Y'}(j)$
0	0.0000040000	5	00000
1	0.0000079840	5	00001
2	0.0000119520	4	0001
3	0.0000159042	4	0010
4	0.0000198405	4	0011
5	0.3333204906	2	10
6	0.3333138754	2	11
7	0.3333059533	2	01

Tabla 5.3: Vector de probabilidades para la variable aleatoria Y' definida en (5.6) y código de Huffman $CH_{Y'}$ asociado.

Supongamos que la fuente que nos interesa codificar genera la siguiente cadena de enteros[3]: 4, 6, 22, 2902, 12, 0 y 105. Como vimos en la sección 5.1.1.1, esto implica ejecutar el algoritmo de codificación de la figura 5.1 una vez para cada uno de estos enteros.

Tanto el 4 como el 0 son enteros menores que α, por lo que ambos se codifican directamente con las respectivas palabras del código de Huffman $CH_{Y'}$ (líneas 1 y 2 del algoritmo). Entonces, de acuerdo a la tabla 5.3, el 4 se codifica con *0011* y el 0 con *00000*.

El resto de los enteros que debemos codificar son mayores que α, por lo que sus palabras de código se definen al ejecutar las lineas 3-6 del algoritmo. Para $i = 6$ tenemos $i - \alpha = 1$, y calculamos $\mathit{cociente} = 1 \text{ div } \beta = 0$ y $\mathit{resto} = 1 \bmod \beta = 1$. Luego obtenemos la palabra del código $CH_{Y'}$ para el entero $\alpha + \mathit{resto} = 6$ (es *11* de acuerdo a la tabla 5.3), y le concatenamos el código unario asociado a $\mathit{cociente} = 0$ (es decir *1*). Entonces, el entero 6 se codifica con la palabra de código *111*.

Para $i = 22$ obtenemos $\mathit{cociente} = 17 \text{ div } \beta = 5$ y $\mathit{resto} = 17 \bmod \beta = 2$. La palabra del código $CH_{Y'}$ para $\alpha + \mathit{resto} = 7$ es *01* y el código unario de $\mathit{cociente} = 5$ es *000001*. Por lo tanto, el entero 22 se codifica con la palabra de código *01000001*.

Para codificar enteros más grandes seguimos el mismo razonamiento. Por ejemplo, para $i = 2902$ tenemos $\mathit{cociente} = 2897 \text{ div } \beta = 965$ y $\mathit{resto} = 2897 \bmod \beta = 2$. El resto tiene el mismo valor que en el caso anterior, por lo que la palabra del código $CH_{Y'}$ es nuevamente *01*. En este caso, le debemos concatenar el código unario del 965, que es una tira de 965 ceros seguida por un uno.

En la tabla 5.4 presentamos un resumen del proceso de codificación de la cadena de enteros. Para cada entero codificado, indicamos cuáles fueron las líneas del algoritmo de codificación

[3]Estos enteros no fueron generados de acuerdo a ninguna distribución, simplemente elegimos enteros que permitieran mostrar cómo funciona el codificador T en varios casos diferentes.

ejecutadas. Además, en los casos en que fue necesario calcularlos, mostramos los valores obtenidos para el cociente y el resto. En la última columna especificamos la palabra de código con la que el codificador T codifica cada entero, utilizando diferentes colores para distinguir los bits obtenidos del código de Huffman $CH_{Y'}$, de los bits que dependen de la codificación unaria.

entero i	líneas ejecutadas	resto	cociente	palabra de código T(i)
4	1-2	-	-	$CH_{Y'}(4) = 0011$
6	3-6	1	0	$CH_{Y'}(6) \bowtie \mathrm{U}(0) = 111$
22	3-6	2	5	$CH_{Y'}(7) \bowtie \mathrm{U}(5) = 01000001$
2902	3-6	2	965	$CH_{Y'}(7) \bowtie \mathrm{U}(965) = 010.....01$
12	3-6	1	2	$CH_{Y'}(6) \bowtie \mathrm{U}(2) = 11001$
0	1-2	-	-	$CH_{Y'}(0) = 00000$
105	3-6	0	20	$CH_{Y'}(5) \bowtie \mathrm{U}(20) = 100.....01$

Tabla 5.4: Resumen del proceso de codificación de la cadena de enteros 4-6-22-2902-12-0-105, presentado en la sección 5.2.1.

5.2.2 Decodificación

Consideramos la cadena de bits obtenida al concatenar (en orden) las respectivas palabras de código con que fueron codificados los enteros de la cadena definida en la sección 5.2.1. A esa cadena de bits, que mostramos en (5.7), la vamos a decodificar en esta sección.

$$bit\_stream = 00111110100000101\underbrace{0.....0}_{965\ \text{ceros}}1110010000010\underbrace{0.....0}_{20\ \text{ceros}}1 \tag{5.7}$$

Como fue explicado en 5.1.2.1, para decodificar una cadena de enteros debemos ejecutar el algoritmo de decodificación de la figura 5.2 en un ciclo que finalice una vez que hayan sido leídos todos los bits de la cadena de bits de entrada[4]. Cada vez que se ejecuta, el algoritmo decodifica un entero de la cadena. Además de la cadena de bits, el algoritmo recibe los valores de α y β utilizados al codificar ($\alpha = 5$ y $\beta = 3$ en este caso). El algoritmo también recibe un árbol binario llamado *arbol_huff*, que es el árbol asociado al código de Huffman $CH_{Y'}$ calculado al codificar la cadena de enteros. Este árbol está representado en la figura 5.3.

La primera vez que ejecutamos el algoritmo de decodificación, como aún no hemos leído ningún bit, la cadena de bits de entrada es la que aparece representada en (5.7). En la primera línea del algoritmo, se iguala la variable *nodo_actual* al nodo raíz del árbol (nodo A). Luego, entre las líneas 2 y 9 hay un ciclo en el que se va actualizando *nodo_actual* a partir del bit leído de la cadena. Después de leer los primeros 4 bits ("0011") se recorrió el camino A-B-D-F-4 y *nodo_actual* es una hoja, por lo que se sale del ciclo. A continuación, al ejecutar la línea 10, la variable x pasa a valer 4, que es el entero asociado a la hoja de acuerdo al código $CH_{Y'}$. Luego, como $x < \alpha$, la condición de la línea 11 es verdadera y el algoritmo devuelve $x = 4$. Por lo tanto, el primer entero de la cadena es decodificado correctamente.

En la segunda ejecución del algoritmo ingresa la cadena de bits "111010..." (cadena de la ecuación (5.7) sin los cuatro bits que ya fueron decodificados). En este caso, después de leer los dos primeros bits ("11"), llegamos a la hoja 6 por el camino A-C-6. Pero ahora, como x vale 6, la condición de la línea 11 es falsa y se deben ejecutar las líneas 14 y 15 del algoritmo. En la línea

[4]Nota: al trabajar con archivos es necesario utilizar alguna técnica de parada, ya que un archivo almacena una cantidad de bits que debe ser múltiplo de 8.

14, se iguala la variable *cociente* al resultado de decodificar el código unario de la cadena de bits. Como el primer bit leído es un 1, *cociente* es igual a 0. En la línea 15, el algoritmo devuelve un entero igual a $x + \beta * cociente = 6 + 3 * 0 = 6$. Es decir que el segundo entero de la cadena también es decodificado correctamente.

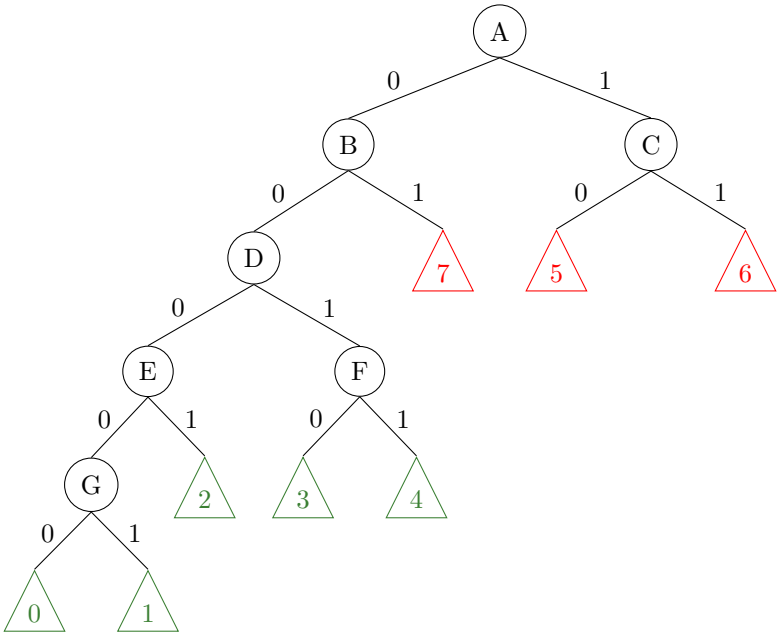

FIGURA 5.3: Árbol de Huffman para el código $CH_{Y'}$ presentado en la tabla 5.3. Las hojas de color rojo están asociadas a supersímbolos. Los nodos tienen letras asociadas que utilizamos como referencia en la descripción del ejemplo de la sección 5.2.2.

En la tercera ejecución ingresa la cadena de bits "0100000101...". Luego de leer "01" se llega a la hoja 7. Al igual que en el caso anterior, la hoja está asociada a un supersímbolo, por lo que se deben ejecutar las líneas 14 y 15 del algoritmo. En este caso, como "000001" es la codificación unaria del 5, *cociente* vale 5 y entonces se devuelve el entero igual a $x+\beta*cociente = 7+3*5 = 22$, por lo que nuevamente la decodificación es exitosa.

Con los tres enteros que hemos decodificado hasta aquí, ya hemos cubierto todo el código del algoritmo de decodificación. Aplicando razonamientos análogos es posible decodificar los restantes enteros de la cadena de bits que aún no ha sido procesada.

5.2.2.1 Árbol binario extendido En la figura 5.4 presentamos una versión extendida del árbol binario de Huffman de la figura 5.3. A las tres hojas del árbol original que están asociadas a supersímbolos se les concatena un árbol infinito, donde el vértice etiquetado con 1 siempre corresponde a una hoja. La hoja del nivel superior queda asociada al mismo entero que el supersímbolo del que se cuelga el árbol infinito, y el resto de las hojas están asociadas a un entero que se va incrementando $\beta = 3$ unidades por nivel. Por lo tanto, estos árboles infinitos representan al código unario que se concatena a la palabra del código de Huffman en el algoritmo de codificación (línea 6 de la figura 5.1).

Obviamente, en la práctica no es posible representar un árbol infinito en memoria[5], por lo que cuando en el algoritmo de la figura 5.2 se llega a uno de los nodos asociado a un supersímbolo, siempre se debe decodificar una palabra de código unario y realizar el cálculo especificado en la línea 15. Sin embargo, nos parece importante presentar el árbol binario extendido, ya que es

[5]No es posible representarlo en un sentido estricto, considerando que cada uno de sus nodos ocupe memoria.

una buena manera de ver el código T de forma gráfica y ayuda a comprender los procesos de codificación y decodificación implementados.

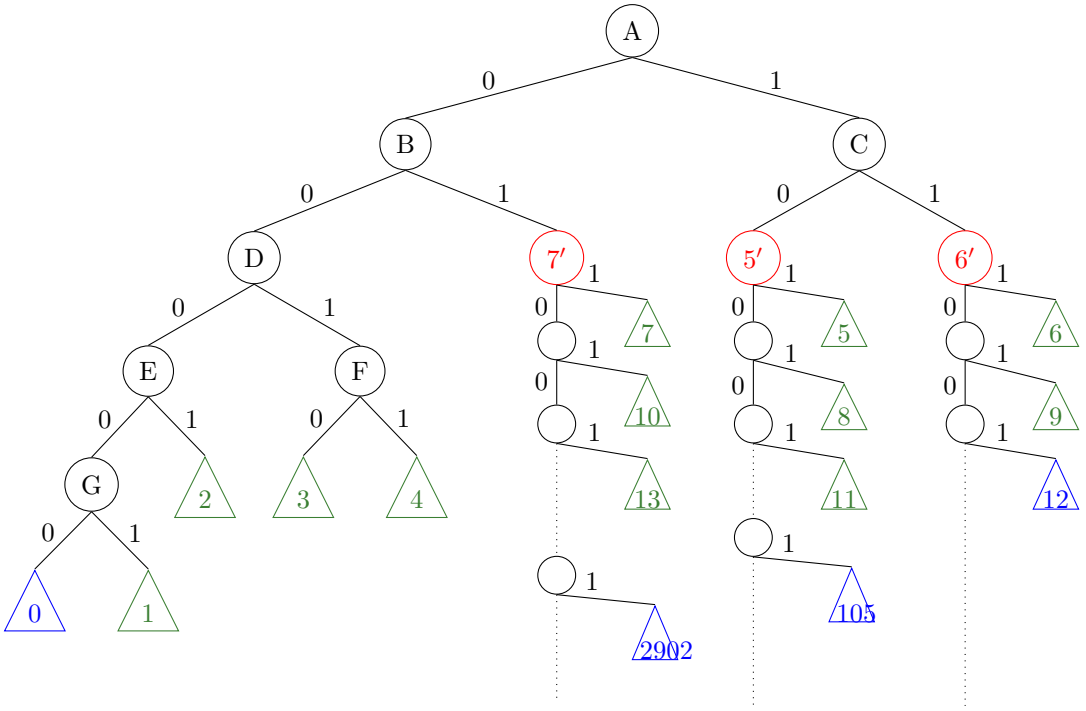

FIGURA 5.4: Árbol binario extendido asociado al código T del ejemplo de la sección 5.2. Los nodos de color rojo son las hojas asociadas a supersímbolos en el árbol de la figura 5.3. Las hojas de color azul están asociadas a los últimos 4 enteros decodificados de la cadena de bits (2902, 12, 0 y 105).

Retomemos el proceso de decodificación de la cadena de enteros, pero ahora decodificando con el árbol extendido. Luego de haber decodificado los primeros tres enteros, la cadena leída es "01 $\underbrace{0.....0}_{965 \text{ ceros}}$ 111001...". Al leer "01" se llega al nodo 7' del árbol extendido. Luego se desciende 965 niveles (un nivel por cada 0 leído) y al leer el 1 se llega a la hoja asociada al entero 2902. Por lo tanto, en este caso se decodifica el cuarto entero de la cadena de forma correcta.

Repitiendo el mismo razonamiento, bajando desde la raíz hasta alguna de las hojas del árbol extendido, es posible decodificar el resto de los enteros de la cadena. En la figura 5.4 las hojas de color azul están asociadas a los últimos cuatro enteros decodificados (2902, 12, 0 y 105).

5.3 Largo medio de código

En esta sección explicamos cómo calcular el largo medio de código obtenido al codificar una fuente con variable aleatoria $Y \sim BN(2,p)$ con $p \in \mathbb{R}_{(0,1)}$, con el código T. Como se observa en el algoritmo de la figura 5.1, cuando el entero i es menor que el parámetro α, i se codifica utilizando el código de Huffman de Y', $CH_{Y'}$. Por lo tanto, la contribución al largo medio de los enteros menores que α está dada por la ecuación

$$\sum_{i=0}^{\alpha-1} \mathrm{P}(Y=i) \cdot l_{\mathrm{T}}(i) \overset{(5.1)}{=} (1-p)^2 \sum_{i=0}^{\alpha-1} (i+1)p^i \cdot l_{CH_{Y'}}(i), \tag{5.8}$$

donde $l_{CH_{Y'}}(i)$ es el largo de la palabra de código con la que se codifica al entero i utilizando el código de Huffman para Y'.

Para calcular la contribución al largo medio de los enteros mayores o iguales que α, partimos el espacio en β clases de congruencia (una para cada supersímbolo j, con $\alpha \leq j < \alpha + \beta$). A continuación planteamos el cálculo del largo medio de todos los enteros $i \geq \alpha$ tales que $i \equiv j \pmod{\beta}$, es decir de todos los i que están en la misma clase de congruencia módulo β que el supersímbolo j

$$\sum_{\substack{i=\alpha \\ i \equiv j \pmod{\beta}}}^{\infty} \mathrm{P}(Y = i) \cdot l_{\mathrm{T}}(i) \overset{(5.4)}{=} \sum_{k=0}^{\infty} \mathrm{P}(Y = j + \beta k) \cdot l_{\mathrm{T}}(j + \beta k). \tag{5.9}$$

En el apéndice H desarrollamos los cálculos de la ecuación (5.9) y luego de agrupar términos obtenemos una función cuyo valor depende únicamente de p, β, j y del código $CH_{Y'}$[6]. A esta función la llamamos $LMCC_T$ (por "largo medio de la clase de congruencia") y la presentamos a continuación

$$\begin{aligned} LMCC_T(p, \beta, j, CH_{Y'}) = \frac{(1-p)^2 p^j}{(1-p^\beta)^3} \Big(& l_{CH_{Y'}}(j) \cdot (j - \beta + 1) p^{2\beta} \\ & + (-j + 2\beta - 1 + l_{CH_{Y'}}(j) \cdot (-2j + \beta - 2)) p^\beta \\ & + (l_{CH_{Y'}}(j) + 1)(j + 1) \Big). \end{aligned} \tag{5.10}$$

Si realizamos este cálculo para cada uno de los β supersímbolos y luego sumamos los resultados, obtenemos la contribución al largo medio de código de todos de los enteros mayores o iguales que α.

En el algoritmo de la figura 5.5 detallamos los pasos seguidos para calcular el largo medio del código obtenido al codificar Y con el codificador T.

entradas: $p \in \mathbb{R}_{(0,1)}$: parámetro de la variable aleatoria Y
$\alpha \in \mathbb{Z}_{\geq 0}$ y $\beta \in \mathbb{Z}_{>0}$: parámetros del codificador
vector_huff: vector de palabras de código asociado al código de Huffman $CH_{Y'}$
salida : *largo_medio_código*

```
   // ecuación (5.8) - líneas 1-4
 1 largo_medio_cabeza = 0
 2 foreach i ∈ {0, ..., α − 1} do
 3 |  largo_medio_cabeza += probY(p, i) · largo_palabra(vector_huff[i])
 4 end
 5 largo_medio_cola = 0
 6 foreach j ∈ {α, ..., α + β − 1} do
   |  // ecuación (5.10) - línea 7
 7 |  largo_medio_cola += LMCC_T(p, β, j, vector_huff[j])
 8 end
 9 return (largo_medio_cabeza + largo_medio_cola)
```

FIGURA 5.5: Algoritmo para calcular el largo medio del código T.

Este algoritmo tiene las mismas entradas que el algoritmo de codificación: el parámetro p de la distribución Y, los dos parámetros del codificador T y el vector *vector_huff*, que tiene las $\alpha + \beta$ palabras del código de Huffman $CH_{Y'}$.

En las primeras cuatro líneas obtenemos el largo medio para los primeros α enteros, de la forma planteada en la ecuación (5.8). Luego obtenemos el largo medio para el resto de enteros (líneas 6-8), ejecutando la ecuación (5.10) para cada uno de los β supersímbolos. Finalmente, el algoritmo devuelve la suma de los dos valores calculados (línea 9).

[6]Más específicamente, de los largos de las palabras del código $CH_{Y'}$.

Capítulo 6

Resultados experimentales

En este capítulo presentamos los resultados experimentales del proyecto. En la sección 6.1 describimos el entorno en donde ejecutamos los programas con los que obtuvimos los resultados. En la sección 6.2 estudiamos la forma en que varían los parámetros del código T. En la sección 6.3 comparamos la entropía de la fuente con los respectivos largos medios de código obtenidos al codificar la fuente con los dos codificadores implementados en el proyecto, para un conjunto representativo de fuentes. Por último, en la sección 6.4 comparamos los tiempos de ejecución necesarios para construir cada código.

6.1 Entorno

Tanto el codificador T como el codificador GolombBN fueron implementados en el lenguaje *C++* compilado con *GNU-GCC*. Obtuvimos todos los resultados experimentales presentados en este capítulo en una computadora con las siguientes características:

- Sistema operativo: ubuntu 14.04 LTS 64-bits
- Memoria: 7,7 GiB
- Procesador: Intel Core i7-3630QM CPU @ 2.40GHz x 8

Para todos los resultados experimentales presentados, consideramos valores de p en el intervalo $[0.5, 0.9997]$. Para valores de p superiores a $p = 0.9997$ el tiempo de construcción del código T se vuelve prohibitivo, como veremos en la sección 6.4. En el intervalo $[0.5, 0.9]$ evaluamos valores de p equiespaciados cada 0.001 unidades, es decir $p \in \{0.5, 0.501, ..., 0.899, 0.9\}$ (401 puntos en total). En cambio, en el intervalo $[0.9, 0.9997]$ tomamos valores de p cada 0.0001 unidades, es decir $p \in \{0.9, 0.9001, ..., 0.9996, 0.9997\}$ (998 puntos en total).

Tomando los intervalos descriptos logramos obtener resultados más representativos para los valores de p más cercanos a 1. Estos casos son los que más nos interesa estudiar, ya que, como vimos en la definición 3.2.1, el parámetro p está asociado a la probabilidad de éxito, que en general es alta en la práctica.

Nos parece importante explicar por qué no consideramos valores de p menores a 0.5 en ninguno de nuestros experimentos. Supongamos que nos interesa comprimir una fuente que genera enteros siguiendo una variable aleatoria $Y \sim BN(2, p)$, con $p \in \mathbb{R}_{(0,0.5)}$. En ese caso obtendremos un menor largo medio de código si tomamos $p' = 1 - p$ y codificamos la variable aleatoria equivalente

$Z \sim BN(2, p')$, con $p' \in \mathbb{R}_{(0.5,1)}$. Observar que los experimentos considerados en este capítulo se pueden aplicar a Z, y por lo tanto no aporta nada considerar el caso equivalente de la variable aleatoria Y.

6.2 Variación de los parámetros del código T

En las gráficas de las figuras 6.2 y 6.3 se muestra cómo varían los dos parámetros del código T al cambiar p. Observamos que tanto α como β tienden a aumentar a medida que p se acerca a 1. Para entender este comportamiento debemos considerar la función de probabilidad f_Y de la distribución binomial negativa, estudiada en el apéndice B.

A medida que el parámetro p se acerca a 1, se incrementa el punto máximo de la función f_Y, que es el punto en el que se anula su derivada. Al aumentar el punto máximo, es esperable que también se incremente el valor del parámetro α. Además, sabemos que el parámetro l del código de Golomb, que es óptimo para codificar f_X (función de probabilidad de la distribución geométrica), aumenta al acercarse p a 1. Por lo tanto, es esperable que el parámetro β, que cumple un rol análogo en nuestro código T, también aumente con p.

En las gráficas de las figuras 6.2 y 6.3 notamos que el parámetro α cambia de una forma particular cuando aumenta el valor de p. Para cada valor de β observamos que el parámetro α tiene dos patrones de variación distintos. La figura 6.1 es una ampliación de una zona de la gráfica de la figura 6.3 en la que podemos observar claramente estos dos patrones. A los conjuntos disjuntos de valores de α pertenecientes a cada patrón para un β dado les llamamos $P1_\beta$ y $P2_\beta$. En la figura 6.1, para cada β los valores de α del patrón 1 están dentro de un rectángulo celeste, y el resto de valores de α pertenecen al patrón 2. Observamos que, para cada uno de los tres valores de β, la forma de los patrones es muy similar. Este comportamiento se repite para casi todos los valores graficados en las figuras 6.2 y 6.3. Para valores de p mayores que 0.96 el parámetro β se mantiene constante para muy pocos valores de p, por lo que es más difícil observar los patrones descriptos.

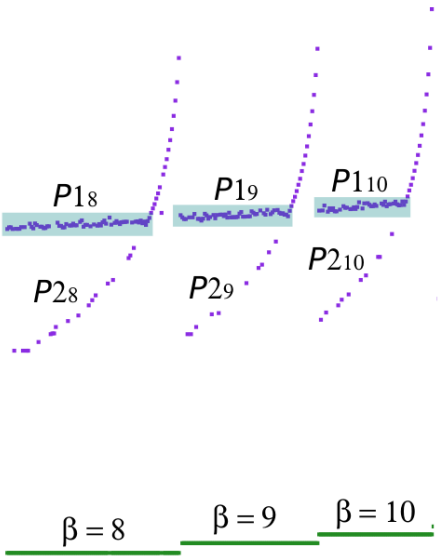

FIGURA 6.1: Patrones de variación del parámetro α del código T para tres valores de β distintos.

Notamos que para cada β, la diferencia entre el máximo y el mínimo valor es siempre mayor para los $\alpha \in P2_\beta$ que para los $\alpha \in P1_\beta$. Para un β dado, los $\alpha \in P1_\beta$ aumentan con p de forma lineal, mientras que los $\alpha \in P2_\beta$ aumentan con p de forma exponencial. Además, al aumentar el valor del parámetro β también aumentan las respectivas pendientes para cada uno de los patrones.

Al momento de escribir el informe, no estamos seguros de a qué responden estos dos patrones del parámetro α. No descartamos que pueda estar asociado a límites de precisión numérica y pensamos estudiar más a fondo este comportamiento en el futuro.

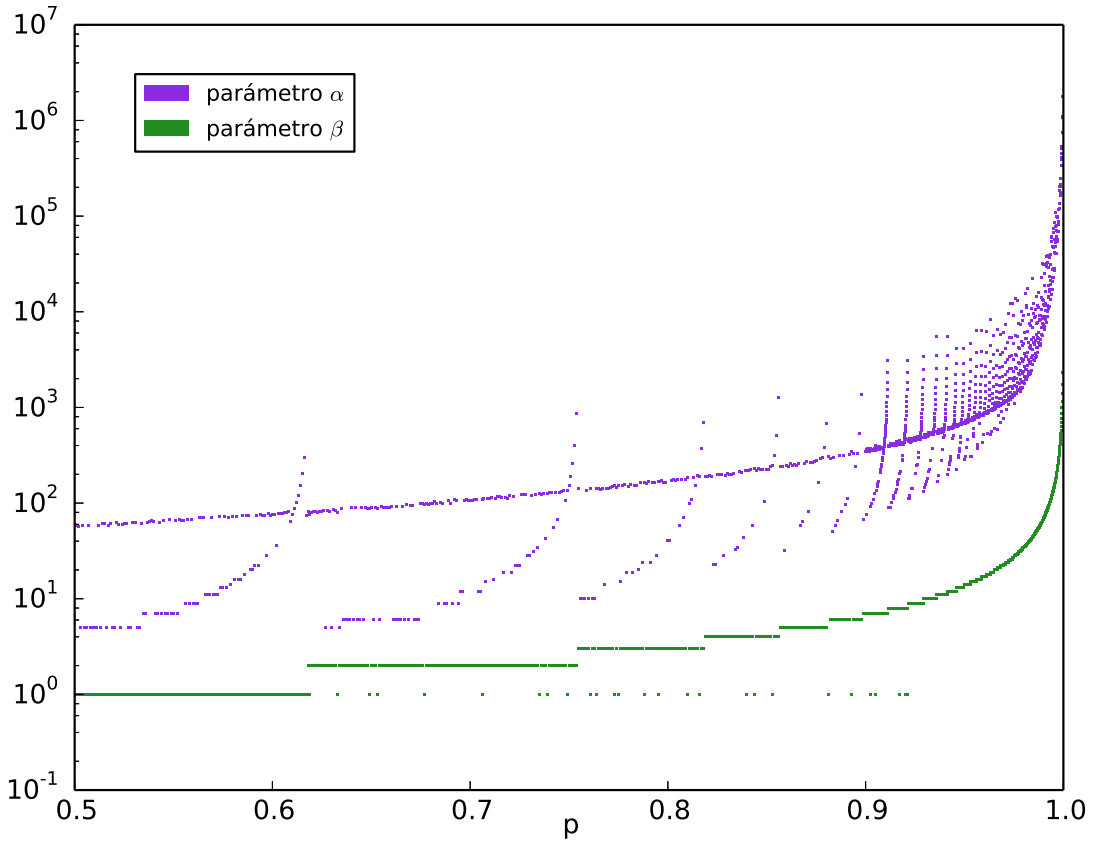

FIGURA 6.2: Parámetros del código T, para $p \in [0.5, 0.9997]$.

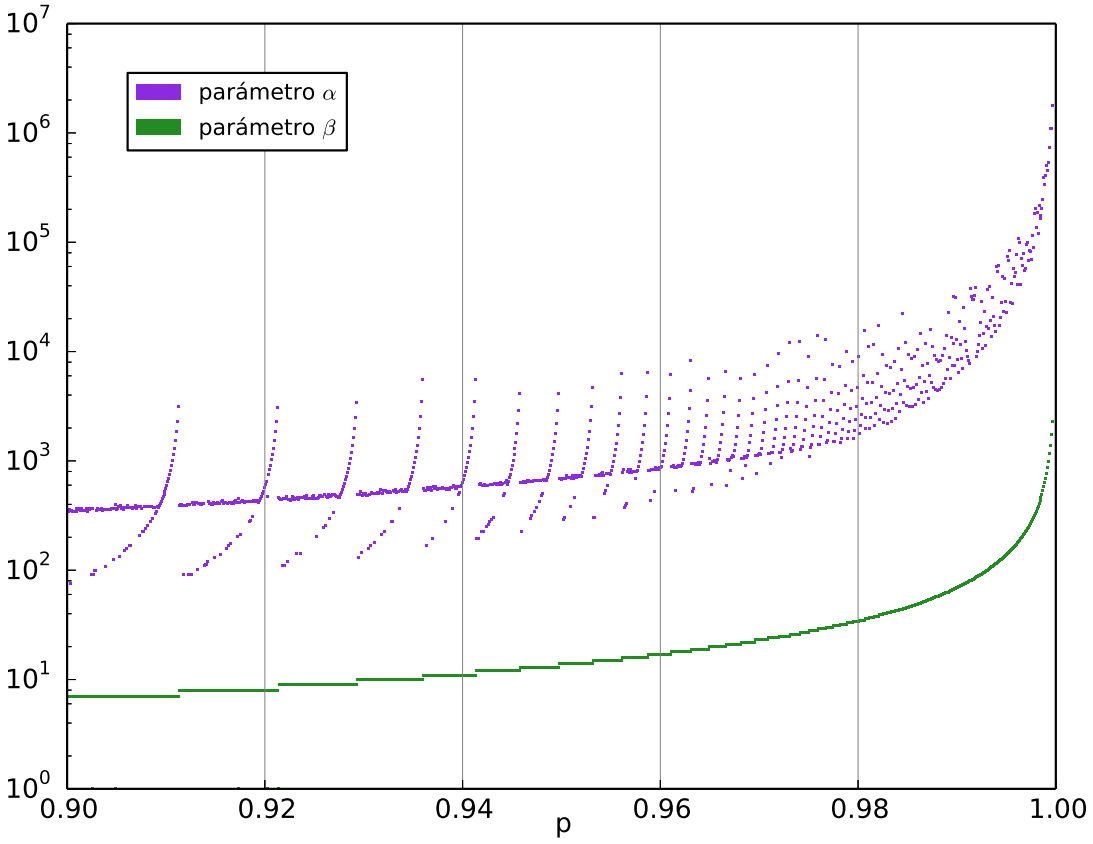

FIGURA 6.3: Parámetros del código T, para $p \in [0.9, 0.9997]$.

6.2.1 Comparación del parámetro β del código T y el parámetro l del código de Golomb

Al analizar el parámetro β notamos que para todo el rango de valores de p considerado, β toma valores muy similares al parámetro l del código de Golomb. En las figuras 6.4 y 6.5 graficamos la diferencia entre el parámetro β del código T y el parámetro l del código de Golomb, para el rango de valores de p estudiado.

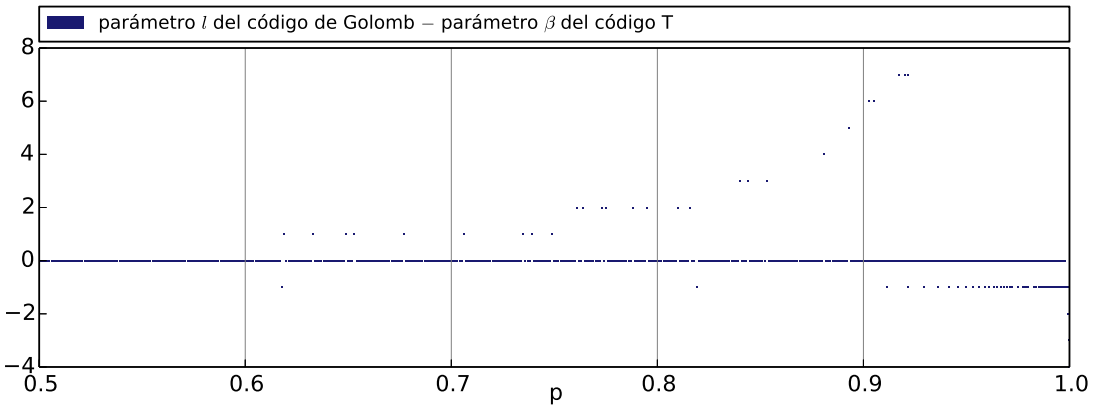

FIGURA 6.4: Diferencia entre el parámetro β del código T y el parámetro l del código de Golomb, para $p \in [0.5, 0.9997]$.

Observamos que para los 400 valores de p menores a 0.90 solamente hay 2 casos (0.5%) en los que β es mayor que l, 22 casos (5.5%) en los que l es mayor que β y en la amplia mayoría de casos (94%) son iguales. Para los valores de p en los que l es mayor que β, la diferencia tiende a aumentar a medida que p crece, hasta aproximadamente $p = 0.92$, donde la diferencia se maximiza y l vale 7 unidades más.

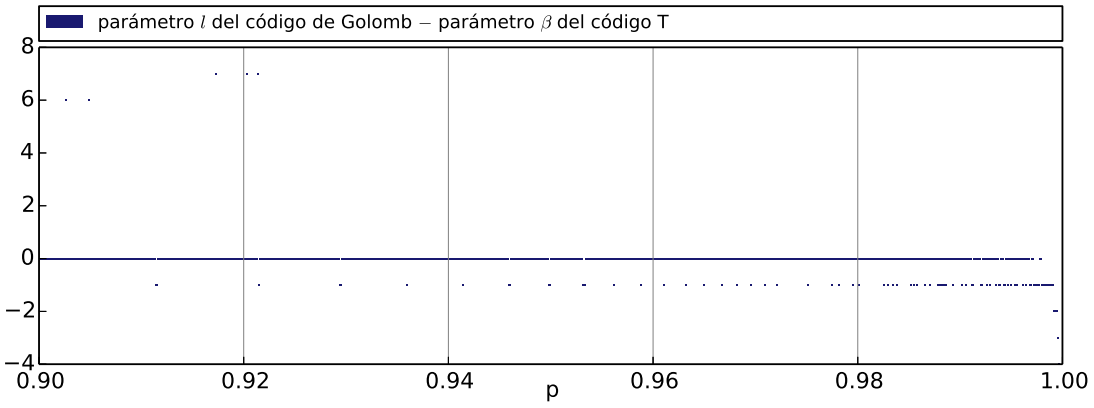

FIGURA 6.5: Diferencia entre el parámetro β del código T y el parámetro l del código de Golomb, para $p \in [0.9, 0.9997]$.

A medida que p se acerca a 1 se incrementa el porcentaje de puntos en los que β es mayor que l. Para los valores de p más cercanos a 1 se alcanza la máxima diferencia a favor de β, cuando β vale 3 unidades más que l. Sin embargo, para p cercanos a 1 los valores que toma β se disparan, por lo que de hecho la diferencia relativa con l tiende a disminuir. Para los 998 valores de p mayores o iguales a 0.90 obtenemos los siguientes resultados: hay 88 casos (9%) en los que β es mayor que l, solamente 5 casos (0.5%) en los que l es mayor que β y 905 casos (90.5%) en los que son iguales.

Como vimos en la sección 2.4, el código de Golomb codifica la distribución geométrica de forma óptima. Por otro lado, construimos el código T con el objetivo de codificar de la mejor manera posible la suma de dos geométricas. Por lo tanto, es esperable que los valores de los parámetros que cumplen una función parecida en ambos códigos sean similares. Sin embargo, destacamos que los valores son prácticamente iguales para todos los valores de p analizados.

Esta observación sugiere que podríamos tomar el parámetro de Golomb despejado de la fórmula (2.11) en vez de calcular β. Sin embargo, el cálculo de α de todos modos implica la ejecución del algoritmo, por lo que a fin de cuentas esta aproximación no redundaría en un ahorro sustancial de recursos de cómputo para la construcción del código T.

6.2.2 Comparación del parámetro β del código T y el parámetro l del código GolombBN

En las gráficas de las figuras 6.6 y 6.7 comparamos la manera en que varían el parámetro β del código T y el parámetro l del código GolombBN, para el rango de valores de p estudiado.

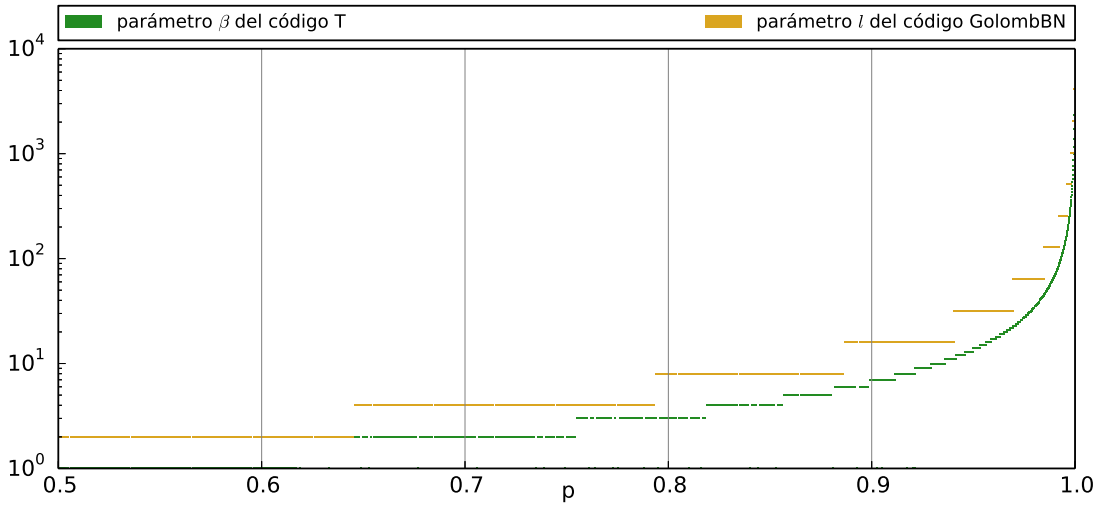

FIGURA 6.6: Parámetro β del código T y parámetro l del código GolombBN, para $p \in [0.5, 0.9997]$.

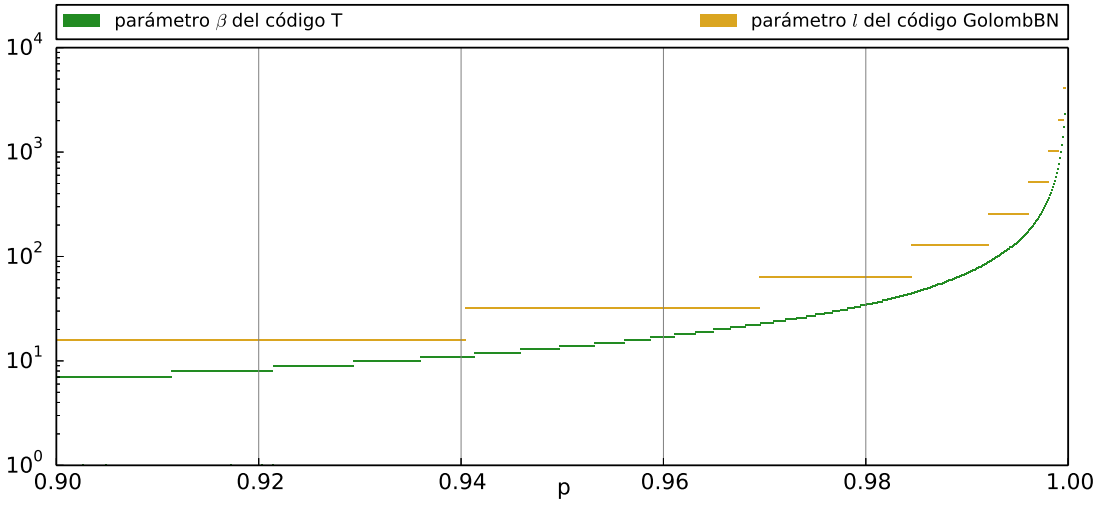

FIGURA 6.7: Parámetro β del código T y parámetro l del código GolombBN, para $p \in [0.9, 0.9997]$.

Se observa que para los valores de p en el rango $[0.5, 0.65]$ se cumple $l = \beta = 1$, mientras que para el resto de valores de p el parámetro l es casi siempre mayor que el parámetro β. En la sección 4.3 explicamos la manera en que se calcula el parámetro l del código GolombBN y vimos que siempre es potencia de dos. Por lo tanto, es lógico que el parámetro l se incremente de forma más abrupta que el parámetro β, que cambia de a una unidad por vez.

En la sección 6.3 estudiamos, entre otras cosas, como influye el cambio de los parámetros β y l en los respectivos largos medios de los códigos T y GolombBN.

6.3 Largos medios de código

Al contrario de lo que ocurre con la distribución geométrica, no existe una fórmula analítica que nos permita calcular la entropía de la distribución binomial negativa. Por lo tanto, en esta sección consideramos una cota inferior de la entropía, la cual calculamos aplicando la fórmula (2.3) sobre el vector finito definido en (5.2). Recordemos que este vector es utilizado para calcular los parámetros del código T y está compuesto por probabilidades asociadas a una variable aleatoria $Y \sim BN(2, p)$ con $p \in \mathbb{R}_{(0,1)}$, realizando un truncamiento en el último símbolo. Este truncamiento nos garantiza que la entropía de este vector sea una cota inferior de la verdadera entropía de la fuente. En lo que resta del presente capítulo, cuando decimos entropía nos estamos refiriendo a esta cota inferior de la entropía.

En las figuras 6.8 y 6.9 presentamos las gráficas de la entropía de la fuente, el largo medio del código GolombBN y el largo medio del código T, para el rango de valores de p estudiado. Calculamos el largo medio del código GolombBN utilizando el procedimiento detallado en la sección 4.4. Para calcular el largo medio del código T seguimos los pasos descriptos en la sección 5.3. El largo medio del código de Huffman que calculamos para obtener los parámetros del código T, de la manera explicada en la sección 5.1, es igual al largo medio del código T hasta las primeras 10 cifras después de la coma.

En primer lugar, observamos que tanto la entropía como los largos de ambos códigos son crecientes con p. Esto es una consecuencia lógica de la forma que tiene la función de probabilidad de la distribución binomial negativa, estudiada en el apéndice B. Al igual que ocurre con la distribución geométrica, la incertidumbre sobre el resultado de la variable aleatoria se dispara cuando p se acerca a 1.

En las figuras 6.10 y 6.11 graficamos la diferencia entre los largos de código de cada uno de los códigos implementados y la entropía de la fuente codificada, para el rango de valores de p estudiado. Estas gráficas son importantes ya que permiten una visualización comparativa del rendimiento del codificador T y el codificador GolombBN.

Observando las primeras cuatro gráficas presentadas en esta sección, lo primero que debemos destacar es que para todos los valores de p estudiados se cumplen dos cosas:

- la entropía es menor al largo medio del código T
- el largo medio del código T es menor al largo medio del código GolombBN[1].

[1] En la figura 6.10, para algunos valores de p cercanos a 0.57, da la impresión de que las gráficas llegan a tocarse, pero analizando los datos graficados verificamos que esto no ocurre.

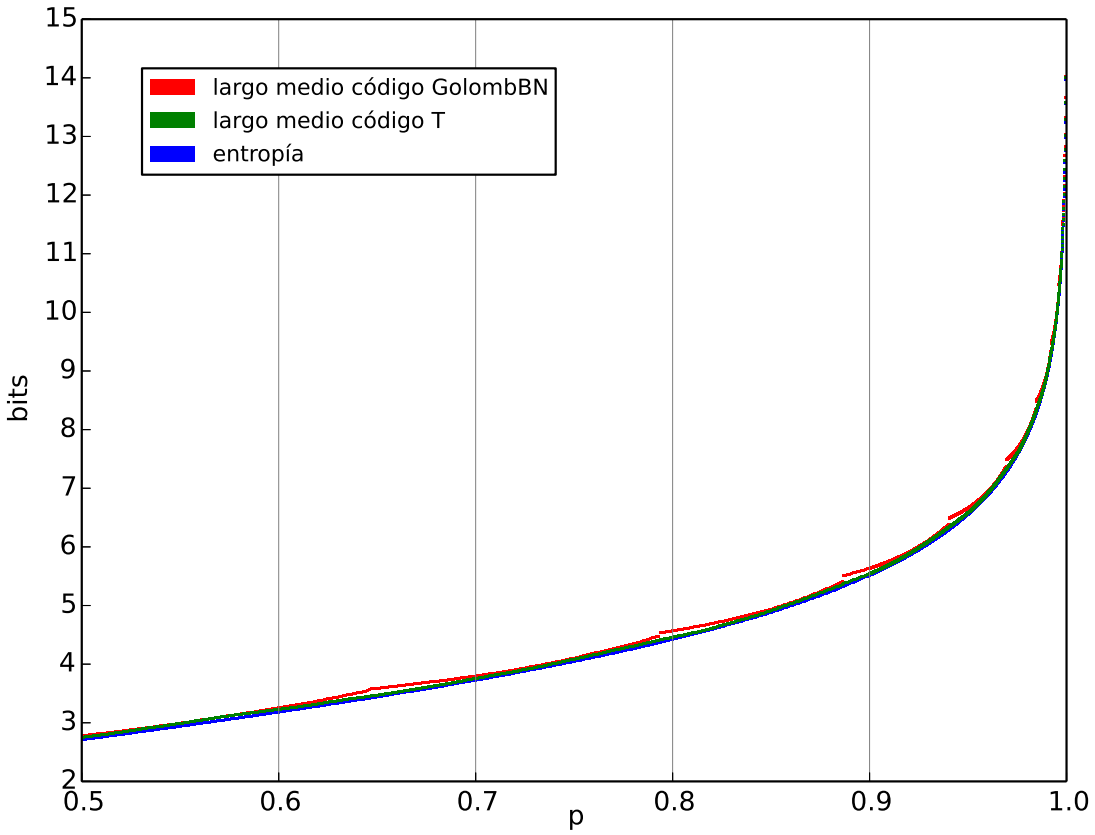

FIGURA 6.8: Entropía y largo medio de los códigos GolombBN y T, para $p \in [0.5, 0.9997]$.

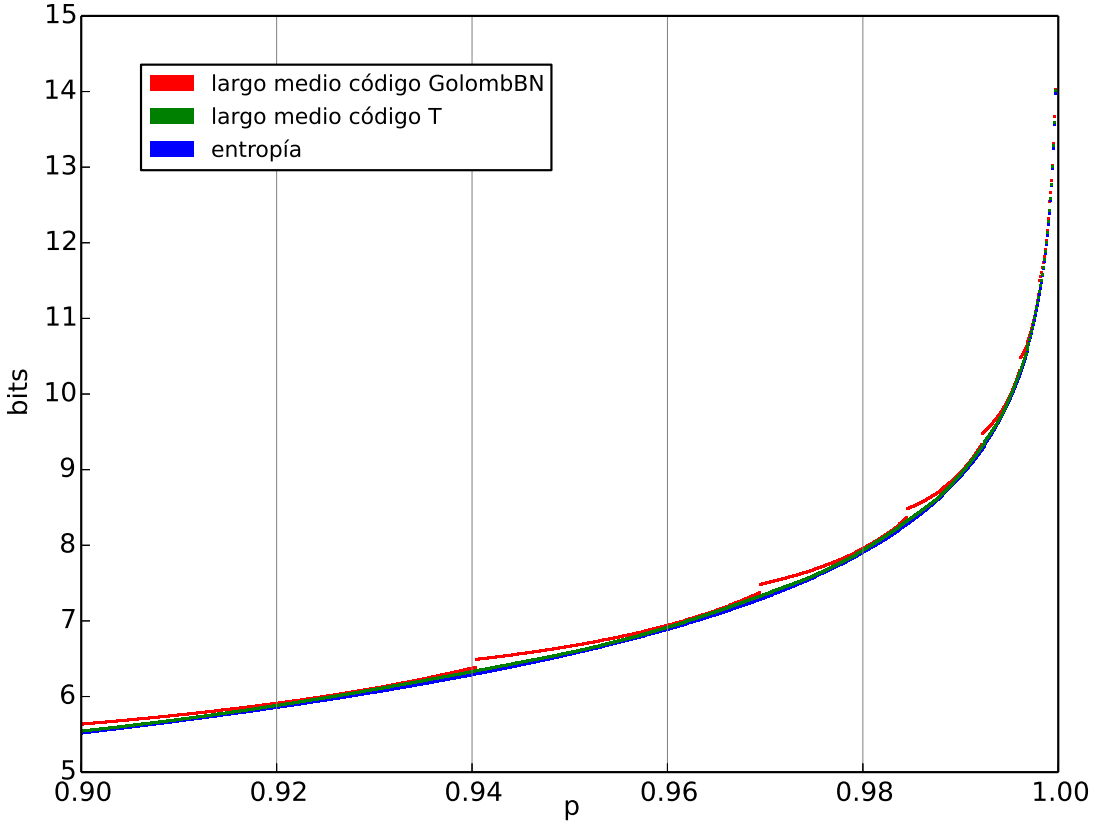

FIGURA 6.9: Entropía y largo medio de los códigos GolombBN y T, para $p \in [0.9, 0.9997]$.

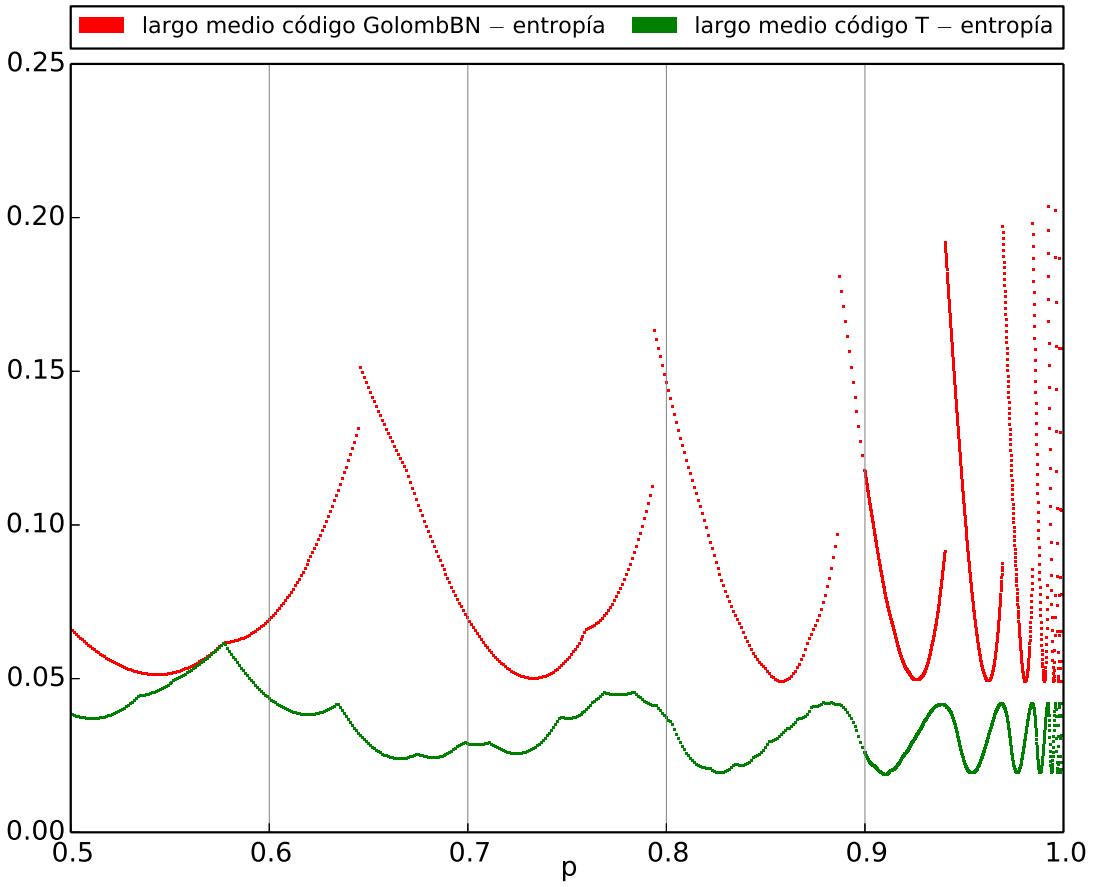

FIGURA 6.10: Diferencia entre el largo medio de código y la entropía para los códigos GolombBN y T, para $p \in [0.5, 0.9997]$.

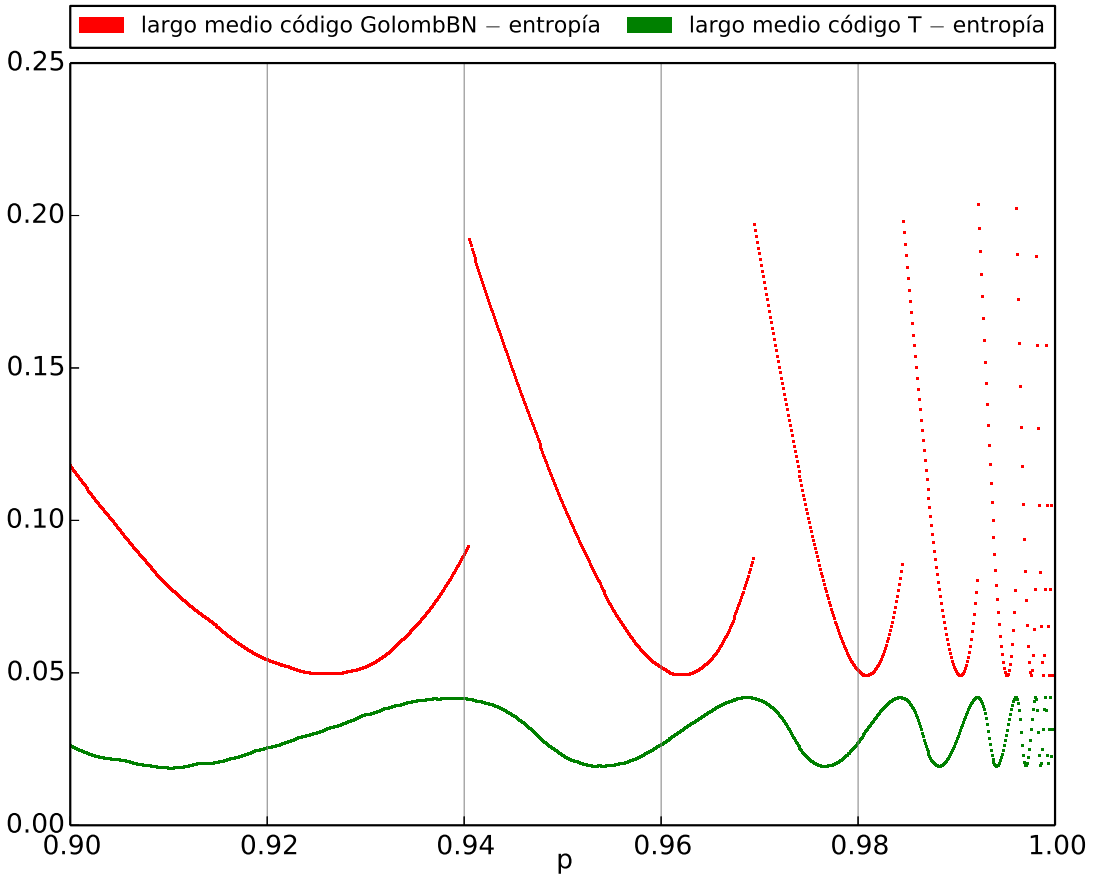

FIGURA 6.11: Diferencia entre el largo medio de código y la entropía para los códigos GolombBN y T, para $p \in [0.9, 0.9997]$.

Es lógico que la entropía sea menor al largo medio de ambos códigos. Esto es una consecuencia directa del Primer Teorema de Shannon, presentado en 2.2.1, que nos dice que ningún código puede comprimir la fuente más allá de su entropía. Además, que el largo medio del código T sea menor que el largo medio del código GolombBN también es esperable. Recordemos que el objetivo con que diseñamos el codificador T es minimizar el largo medio de código, mientras que GolombBN es un codificador poco complejo utilizado para economizar recursos de cómputo.

Otra cosa que debemos destacar es que los incrementos repentinos en el largo del código GolombBN, que se ven claramente en las figuras 6.8 y 6.9, ocurren exactamente para los valores de p en los que aumenta el parámetro l, como se observa en las gráficas de las figuras figuras 6.6 y 6.7. En particular, esto se cumple para valores de p cercanos a los siguientes puntos: 0.65, 0.79, 0.88, 0.94, 0.97, 0.985, 0.992 y 0.995. Si bien en la figura 6.11 podemos distinguir que el parámetro l se incrementa para un par de valores de p más cercanos al 1, en la figura 6.9 los puntos de esa zona están muy desperdigados como para ver con claridad los incrementos repentinos del largo medio.

Sin embargo, no observamos el comportamiento anterior en el caso del código T. No notamos ninguna relación entre el largo medio de código y la variación del parámetro l. Pensamos que esto tiene lógica, teniendo en cuenta la forma en que calculamos el parámetro. Esperamos que el valor de β calculado para cada p sea el parámetro óptimo, en el sentido de que tomando cualquier otro β el largo medio del código T para ese p sea mayor. Por lo tanto cuando cambia β los saltos en el valor del largo medio son imperceptibles en las gráficas de las figuras 6.8 y 6.9.

En las figuras 6.12 y 6.13 graficamos la redundancia relativa[2] de cada uno de los códigos implementados, para el rango de valores de p estudiado. Observamos que la redundancia relativa de ambos códigos tiende a disminuir al acercarse p a 1. Esto es importante, ya que como comentamos en la sección 6.1, estos valores de p son los que más interesan en la práctica.

La redundancia relativa resulta útil para comparar de forma cuantitativa el rendimiento de los dos códigos implementados. En la tabla 6.1 presentamos la redundancia relativa promedio de ambos códigos en tres intervalos disjuntos. Observamos que en todos los intervalos la redundancia relativa promedio del código T es menor que la del código GolombBN. Para los valores de $p \in [0.5, 0.9]$ el código GolombBN tiene una redundancia relativa promedio aproximadamente 2 veces mayor que el código T, mientras que para los valores de $p \in [0.90, 0.9997]$ la redundancia relativa es casi 3 veces mayor.

rango p	# puntos	(A) redundancia relativa promedio código T	(B) redundancia relativa promedio código GolombBN	(B)/(A)
$[0.5, 0.9]$	401	0.0100307627094668	0.0212900813849203	2.12247881856753
$[0.9, 0.95]$	501	0.00489466451896755	0.0141414488844169	2.88915590223123
$[0.95, 0.9997]$	498	0.00377216873570429	0.0107658518773017	2.85402181917285

Tabla 6.1: Redundancia relativa promedio de los códigos T y GolombBN en tres intervalos.

[2]La redundancia relativa se calcula dividiendo la redundancia del código (la diferencia entre el largo medio de código y la entropía de la fuente) por la entropía de la fuente.

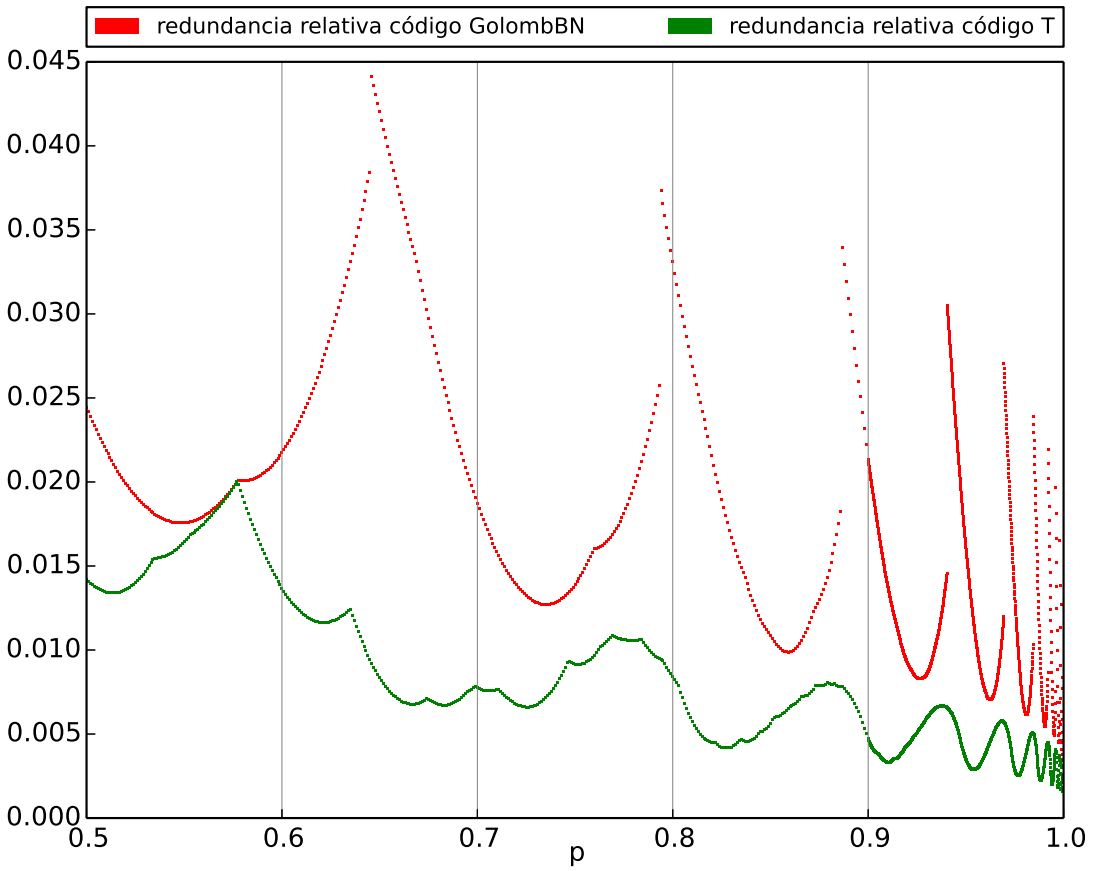

FIGURA 6.12: Redundancia relativa (diferencia entre el largo medio de código y la entropía, dividida entre la entropía) de los códigos GolombBN y T, para $p \in [0.5, 0.9997]$.

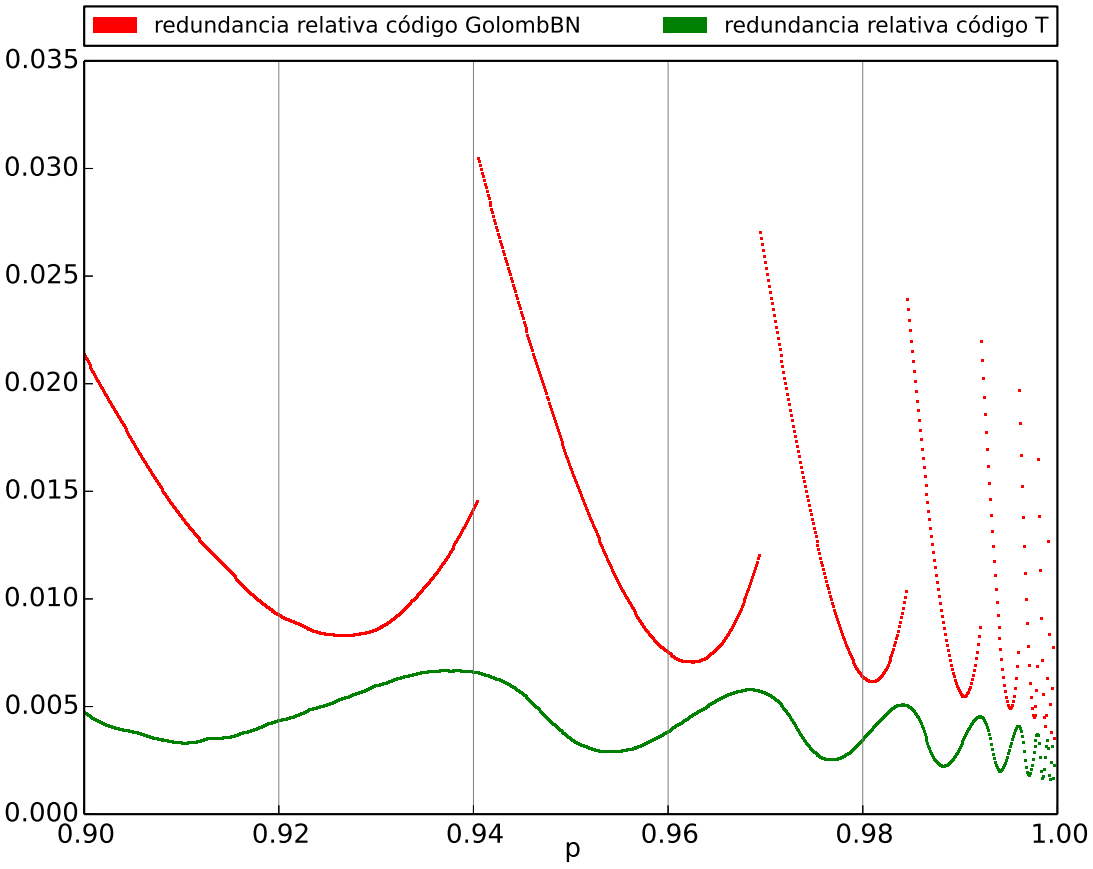

FIGURA 6.13: Redundancia relativa (diferencia entre el largo medio de código y la entropía, dividida entre la entropía) de los códigos GolombBN y T, para $p \in [0.9, 0.9997]$.

En lo que resta de esta sección nos interesa estudiar qué tan bueno es el código T. Dicho de otra manera, queremos saber si es o está cerca de ser un código óptimo. En este sentido, ya vimos que el largo medio obtenido con el código T coincide con el obtenido por el código de Huffman de la fuente truncada en el último símbolo. Además, sabemos que el código de Huffman truncado, por el hecho de estar truncado, debe tener un largo de código medio menor al obtenido con un código óptimo para el alfabeto completo. Concluimos entonces que el código T o bien es óptimo o bien alcanza largos medios muy similares a los que obtendríamos con un código óptimo.

Si comparamos el largo medio del código T con la entropía, en la figura 6.11 vemos que la diferencia entre ambos oscila entre 0.02 y 0.04 bits al acercarse p a 1. Este comportamiento oscilatorio nos recuerda al observado en [18] para el código de Golomb y la distribución geométrica. En ese caso se demostró que la diferencia entre la entropía de la fuente y el largo medio de código no tiene límite cuando p tiende a 1. Además, la diferencia se aproxima en [18] con una fórmula matemática. En el caso que estudiamos en el proyecto, no tenemos fórmula cerrada ni para la entropía ni para el largo medio del código T, por lo que obtener una aproximación de la diferencia, en caso de ser posible, es un problema muy complejo que queda fuera del alcance del proyecto.

6.4 Tiempos de ejecución

En las gráficas de las figuras 6.14 y 6.15 comparamos los tiempos de ejecución necesarios para construir los dos códigos implementados, para el rango de valores de p estudiado. En el caso del código GolombBN, cuyo tiempo de ejecución es muy corto, realizamos 10 ejecuciones para cada p y graficamos el promedio. De esta manera intentamos evitar posibles sesgos causados por variables del sistema en el que fueron ejecutados los experimentos.

Lo primero que notamos es que la diferencia entre los tiempos de construcción de ambos códigos es muy importante. Como vimos en el capítulo 5, la construcción del código T implica crear un árbol de Huffman de gran tamaño, que se utiliza para calcular los parámetros α y β. Además, utilizamos un vector para codificar, y otro árbol de Huffman, de menor tamaño que el primero, para decodificar. En los experimentos consideramos la construcción de ambas estructuras. En cuanto al código GolombBN, vimos en el capítulo 4 que para construirlo solo debemos calcular la función $Perm$ y aplicar una fórmula matemática sencilla para calcular el parámetro l. Como el codificador GolombBN utiliza códigos GPO2, no es necesario crear ninguna estructura para llevar a cabo los procesos de codificación y decodificación.

Por lo anterior, es esperable que el tiempo de construcción del código T sea significativamente mayor que el de GolombBN. La mayoría de ese tiempo se consume armando el árbol de Huffman a partir del cual se calculan los parámetros del código.

Observamos que a medida que p se acerca a 1, los tiempos de construcción de ambos códigos se disparan. En el caso del código T, las dimensiones de las estructuras construidas (dos árboles y un vector) aumentan, por lo que también se incrementa el tiempo requerido. En el caso del código GolombBN, los algoritmo utilizados para calcular la función $Perm$ y el parámetro l también se vuelven más complejos (deben realizar más iteraciones), y requieren más tiempo computacional.

Si bien observamos pequeños picos en las gráficas del tiempo de construcción para los dos códigos, estos no parecen estar relacionados a cambios en los valores de sus respectivos parámetros. En el caso del código GolombBN notamos un descenso brusco del tiempo de construcción para un valor de p cercano a 0.58, pero ignoramos la causa.

Antes de terminar, es importante destacar que pese a las grandes diferencias en los tiempos de construcción, una vez construidos los códigos el tiempo requerido para codificar y decodificar es del mismo orden en ambos casos.

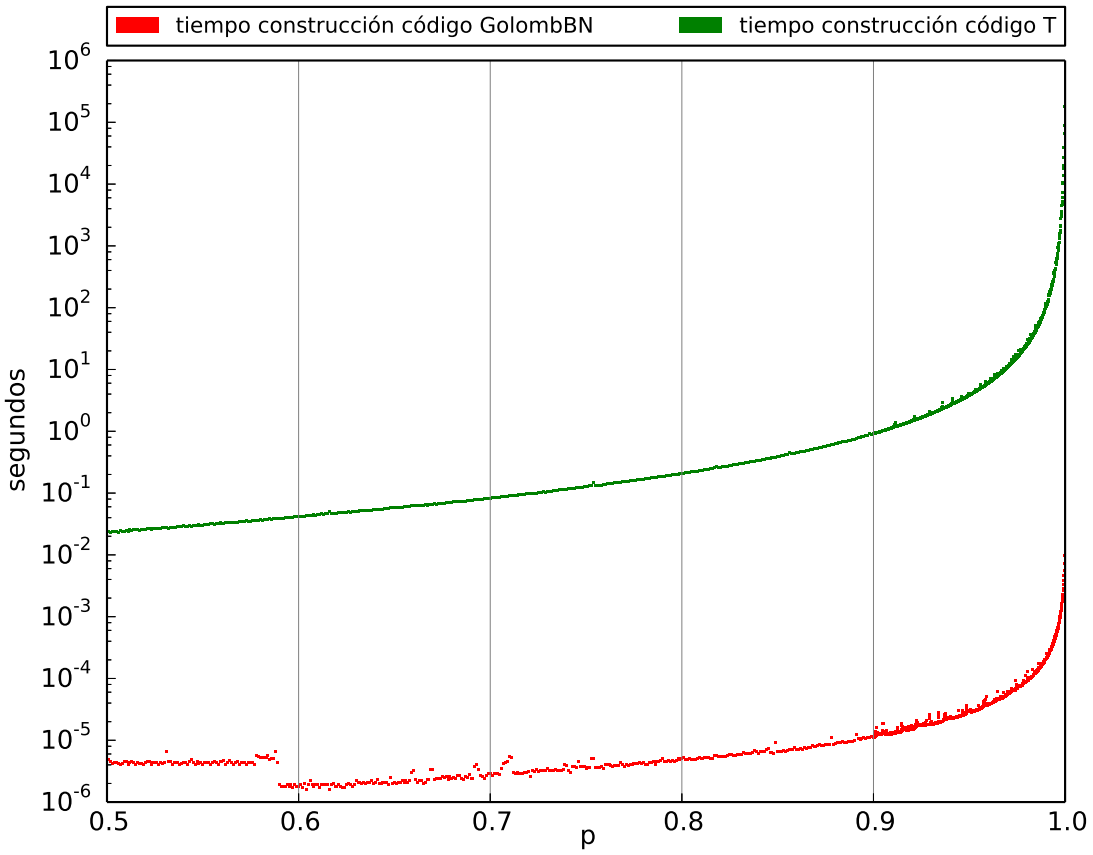

FIGURA 6.14: Tiempo de construcción de los códigos GolombBN y T, para $p \in [0.5, 0.9997]$.

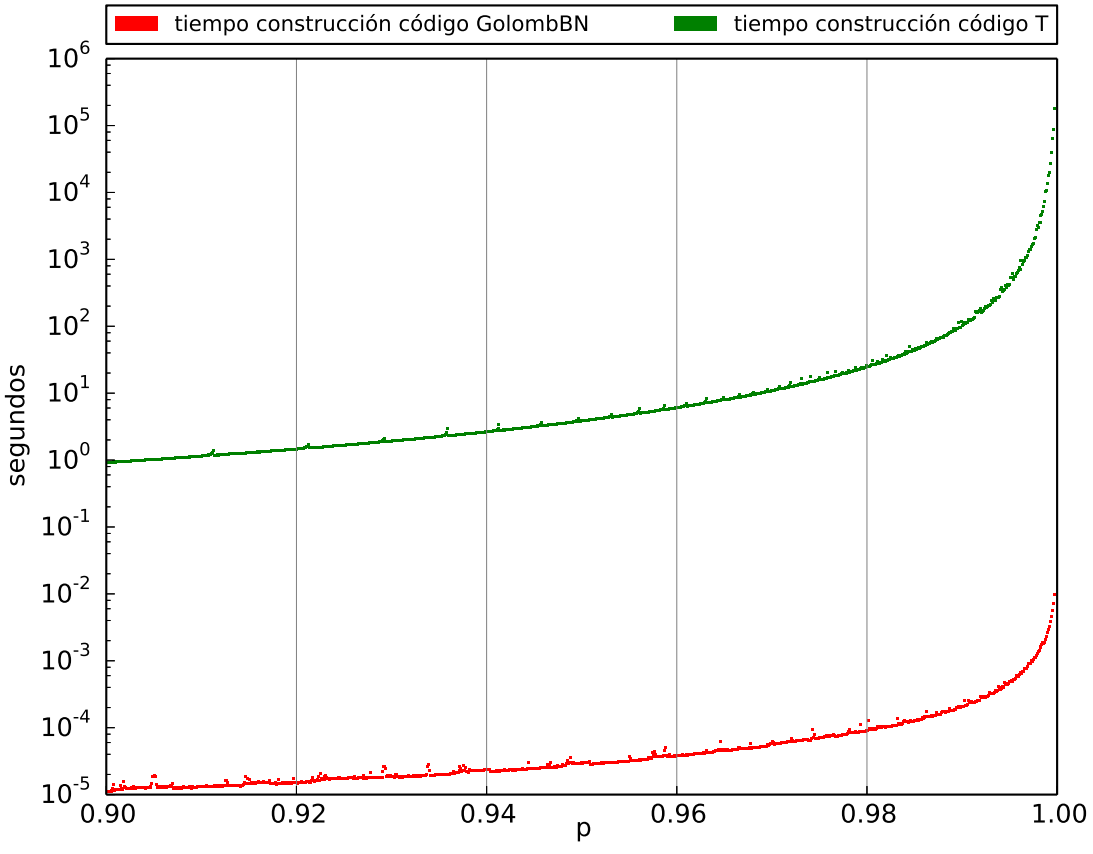

FIGURA 6.15: Tiempo de construcción de los códigos GolombBN y T, para $p \in [0.9, 0.9997]$.

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo repasamos los resultados obtenidos en el proyecto e identificamos algunos problemas que consideramos sería interesante estudiar en el futuro para profundizar el conocimiento adquirido.

7.1 Conclusiones

En una primera etapa del proyecto nos dedicamos a estudiar el problema teórico relativo a la codificación de señales multicanal. Consideramos un sistema de codificación donde nos interesa codificar una señal Y que es una transformación lineal de una señal multicanal X. En este punto llegamos a la conclusión de que la esperanza del largo medio de código obtenido no disminuye cuando el codificador tiene acceso a la señal X. Dicho de otra manera, si lo que interesa es codificar la señal transformada, no aporta nada el conocimiento de la señal original, en cuanto al largo medio de código que podemos obtener.

Continuamos el proyecto estudiando una aplicación práctica del problema de la codificación de señales multicanal. Consideramos un sistema de codificación en el que tenemos una señal multicanal X, que tiene dos componentes independientes $X_1, X_2 \sim Geo(p)$, con $p \in \mathbb{R}_{(0,1)}$, donde nos interesa codificar la señal $Y = X_1 + X_2$ obtenida al sumar las componentes de X. Probamos que en este caso se cumple $Y \sim BN(2, p)$. Entonces nos dedicamos a implementar un codificador para la distribución binomial negativa con parámetros 2 y $p \in \mathbb{R}_{(0,1)}$.

En primer lugar implementamos el codificador GolombBN, que es una versión levemente modificada del codificador de Golomb. Como cabía esperar, el código obtenido en este caso no es óptimo para la distribución binomial negativa, pero de todas maneras fue útil para comparar su rendimiento con el otro codificador implementado, el codificador T. Este último tiene una implementación bastante más compleja que la de GolombBN, pero nos permite obtener mejores resultados en cuanto al largo medio de código.

Estas conclusiones están respaldadas por una serie de experimentos que realizamos durante el proyecto. Comparamos el largo medio obtenido con ambos códigos al codificar variables con distribución binomial negativa para un rango representativo de valores de p y en todos los casos el largo medio obtenido con el codificador T es menor que el obtenido con el codificador GolombBN. Aunque no lo hemos demostrado, pensamos que el código T está muy cerca del código óptimo para la distribución binomial negativa. También estudiamos los tiempos de ejecución necesarios para construir los dos códigos implementados. Para todos los valores de p considerados, el tiempo de ejecución necesario para construir el código T es considerablemente mayor que el necesario

para construir el código GolombBN. Esta diferencia es esperable, ya que obtener los parámetros del código T implica construir un árbol de Huffman de importantes dimensiones. Un mayor tiempo de construcción es el precio que debemos pagar para obtener un código más cercano al óptimo.

Nos parece importante destacar que, si bien hay diferencias notorias entre ambos códigos en cuanto al tiempo de construcción, una vez que los códigos están construidos, el tiempo requerido para codificar y decodificar un símbolo tiene el mismo orden en ambos casos. Además, comentamos que la operación que más tiempo consume en el caso del código T es el cálculo de sus parámetros. Por lo tanto, para utilizar el codificador T haciendo un uso más eficiente del tiempo, conviene tener una tabla precomputada con los respectivos parámetros obtenidos para un rango representativo de valores de p.

El codificador T tiene dos parámetros, α y β, y pensamos que sería interesante estudiar la manera en que varían para los valores de p considerados. En nuestros experimentos observamos que para cada valor de β, el parámetro α sigue dos patrones de variación diferentes. En las gráficas presentadas notamos que los respectivos patrones son similares para casi todos los valores de β, pero no podemos afirmar que esto ocurra para los p cercanos a 1, ya que en esos casos el mismo valor del parámetro β se mantiene para pocos p.

En cuanto al parámetro β, nos llamó la atención la gran similitud que tiene con el parámetro l de los códigos de Golomb. Para la mayoría de valores de p considerados, el β del código T es igual al l de los códigos de Golomb. A medida que p se acerca a 1, se incrementa el porcentaje de casos en los que β es mayor a l, y en casi todos estos casos la diferencia es solamente una unidad.

7.2 Trabajo futuro

El código T, sea o no óptimo, es el código que conocemos que nos permite codificar una distribución binomial negativa con la menor esperanza de largo medio de código. Pero su construcción requiere mucho tiempo computacional, tiempo invertido principalmente en el cálculo de sus dos parámetros, α y β

Para solucionar este problema debemos investigar si existe alguna manera de calcular los parámetros del código T de manera más eficiente. Sería interesante estudiar los resultados experimentales para ver si podemos encontrar alguna relación matemática entre el parámetro de la distribución binomial negativa y los valores de α y β obtenidos.

Como comentamos en la sección anterior, pensamos que el código T está cerca del código óptimo para una distribución binomial negativa. En [18] se utilizó el algoritmo de Huffman de manera indirecta para probar que los códigos de Golomb son óptimos para la distribución geométrica. Nosotros utilizamos herramientas similares para implementar el código T. También nos parece que sería importante investigar cómo aplicar un razonamiento análogo para demostrar formalmente que el código T (o uno muy similar) es óptimo para la distribución binomial negativa. Pensamos que deberíamos encarar este problema después de estudiar si existe alguna relación matemática entre el parámetro de la distribución binomial negativa y los α y β obtenidos, ya que ello nos puede brindar más herramientas para encarar el problema de encontrar el código óptimo.

Creemos que el conocimiento adquirido al implementar el codificador T puede servirnos para estudiar problemas similares en el futuro. Sería interesante analizar cómo se codifica la suma de dos distribuciones geométricas con distinto parámetro. Otro problema que podríamos estudiar

es la codificación de más de dos geométricas con igual o distinto parámetro. Vimos que en la práctica, existen algoritmos para codificar imágenes [20] y audio [21] sin pérdida, que utilizan predictores cuyos residuos se ajustan a una TSGD (distribución geométrica a dos lados). Por lo tanto, también podría ser importante estudiar cómo codificar la suma de dos TSGD con igual o distinto parámetro.

Apéndice A

Suma de distribuciones geométricas

Proposición A.0.1. Sean X_1 y X_2 dos variables aleatorias independientes tales que $X_1, X_2 \sim Geo(p)$ con $p \in \mathbb{R}_{(0,1)}$. Sea Y una variable aleatoria tal que $Y = X_1 + X_2$. Entonces se cumple $Y \sim BN(2, p)$.

Demostración. Como $X_1, X_2 \sim Geo(p)$, por definición sabemos que

$$\mathrm{P}(X_1 = i) = \mathrm{P}(X_2 = i) = (1-p)p^i\,, \quad i \in \mathbb{Z}_{\geq 0}, \tag{A.1}$$

y queremos probar que $Y \sim BN(2, p)$, es decir que

$$\mathrm{P}(Y = i) = (1-p)^2(i+1)p^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{A.2}$$

Como X_1 y X_2 son independientes, para cualquier entero $i \in \mathbb{Z}_{\geq 0}$ se cumple

$$\begin{aligned} \mathrm{P}(Y = i) = \mathrm{P}(X_1 + X_2 = i) &= \sum_{k=0}^{k=i} \mathrm{P}(X_1 = k) \cdot \mathrm{P}(X_2 = i-k) \\ &\overset{(A.1)}{=} \sum_{k=0}^{k=i} (1-p)p^k \cdot (1-p)p^{i-k} \\ &= (1-p)^2 p^i \sum_{k=0}^{k=i} 1 \\ &= (1-p)^2(i+1)p^i. \end{aligned} \tag{A.3}$$

Por lo tanto, se cumple la ecuación (A.2), y entonces $Y \sim BN(2, p)$. □

Apéndice B

Distribución geométrica y distribución binomial negativa

En este apéndice realizamos un análisis comparativo de las funciones de probabilidad f_X para la variable aleatoria $X \sim Geo(p)$ y f_Y para la variable aleatoria $Y \sim BN(2,p)$. En ambos casos consideramos $p \in \mathbb{R}_{(0,1)}$.

Al definir las variables aleatorias, en 2.4.1 y 3.2.1 respectivamente, vimos que

$$f_X(i) = \mathrm{P}(X = i) = (1-p)p^i\,, \quad i \in \mathbb{Z}_{\geq 0} \tag{B.1}$$

y

$$f_Y(i) = \mathrm{P}(Y = i) = (1-p)^2(i+1)p^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{B.2}$$

En las figuras B.2, B.3, B.4 y B.5 presentamos algunos ejemplos de gráficas de $f_X(i)$ y $f_Y(i)$. En cada caso tomamos un valor de p en el conjunto $\{0.60, 0.90, 0.95, 0.99\}$ y graficamos $f_X(i)$ y $f_Y(i)$ para un rango de enteros i elegido para tener una buena visualización en cada caso.

Para cualquier valor del parámetro p se cumple que $f_X(i) > 0$ y $f_Y(i) > 0$ para todo $i \in \mathbb{Z}_{\geq 0}$. Además, para cualquier p se cumple $\lim_{i\to\infty} f_X(i) = \lim_{i\to\infty} f_Y(i) = 0^+$.

Calculamos las derivadas de las funciones, obteniendo

$$f'_X(i) = (1-p)p^i \ln(p)\,, \quad i \in \mathbb{Z}_{\geq 0} \tag{B.3}$$

y

$$f'_Y(i) = (1-p)^2 p^i (1 + (i+1)\ln(p))\,, \quad i \in \mathbb{Z}_{\geq 0}, \tag{B.4}$$

donde $\ln(p)$ denota al logaritmo neperiano de p.

Para cualquier $p \in \mathbb{R}_{(0,1)}$ se cumple $\ln(p) < 0$, y por lo tanto $f'_X(i) < 0$ para todo $i \geq 0$. Por otro lado, $f'_Y(i)$ se anula en el punto $i' = -1/\ln(p) - 1$, con $f'_Y(i) > 0$ cuando $i < i'$ y $f'_Y(i) < 0$ cuando $i > i'$. En la figura B.1 detallamos el signo de las respectivas derivadas.

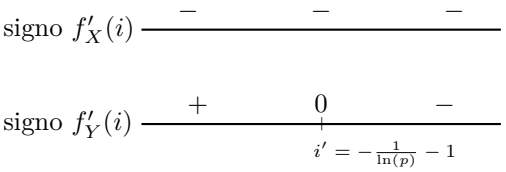

FIGURA B.1: Signos de $f'_X(i)$ y $f'_Y(i)$.

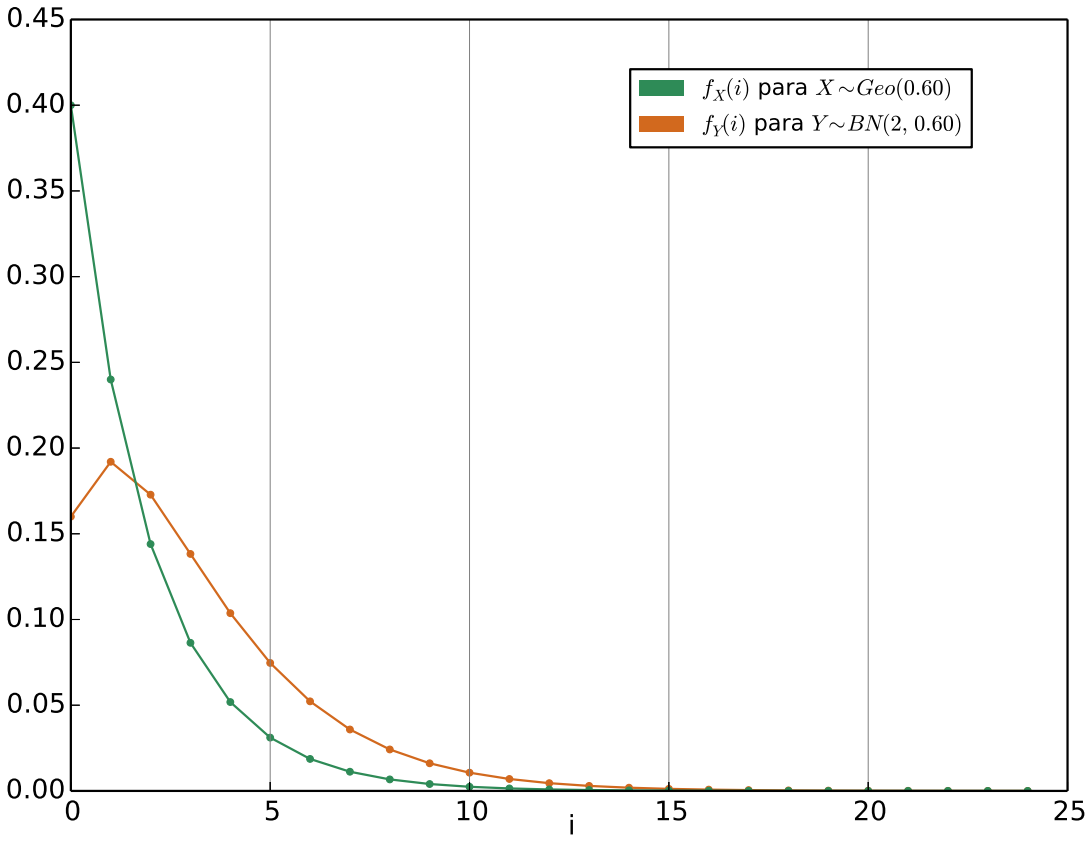

FIGURA B.2: $f_X(i)$ con $X \sim Geo(0.60)$ y $f_Y(i)$ con $Y \sim BN(2,\ 0.60)$, para $i \in \mathbb{Z}_{[0,25]}$. Trazamos rectas entre los puntos adyacentes para mejorar la visualización de la gráfica.

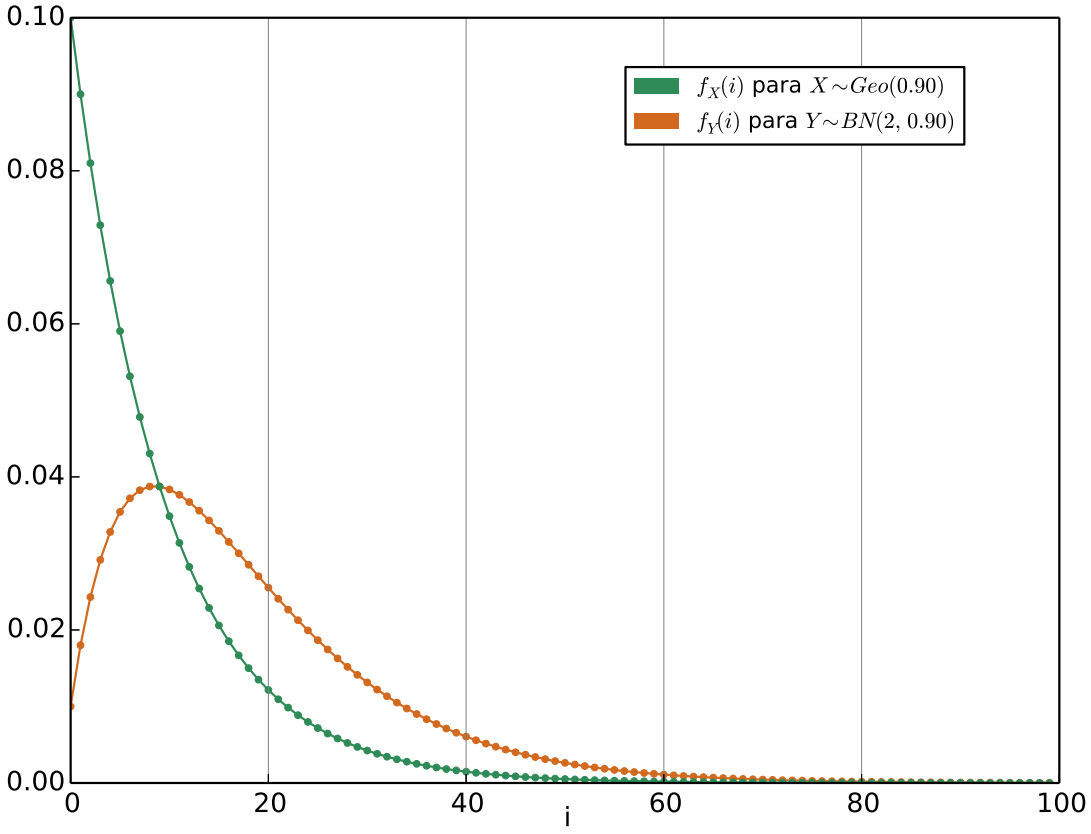

FIGURA B.3: $f_X(i)$ con $X \sim Geo(0.90)$ y $f_Y(i)$ con $Y \sim BN(2,\ 0.90)$, para $i \in \mathbb{Z}_{[0,100]}$. Trazamos rectas entre los puntos adyacentes para mejorar la visualización de la gráfica.

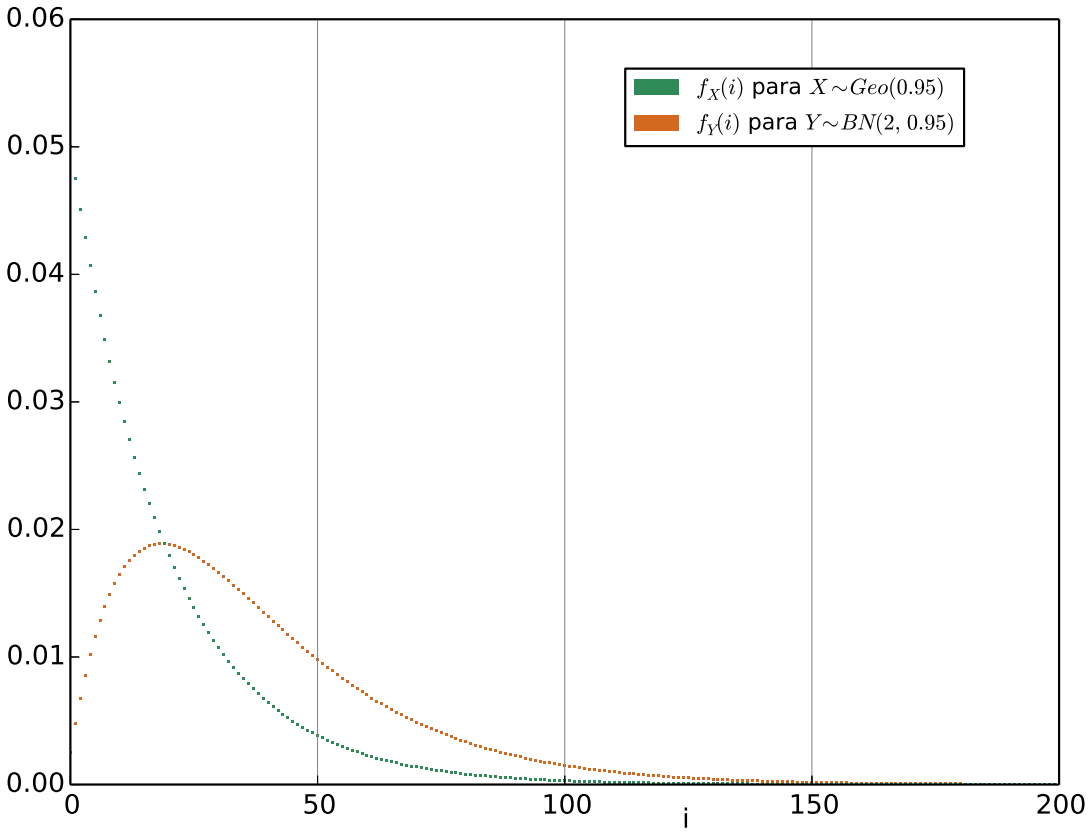

FIGURA B.4: $f_X(i)$ con $X \sim Geo(0.95)$ y $f_Y(i)$ con $Y \sim BN(2,\ 0.95)$, para $i \in \mathbb{Z}_{[0,200]}$.

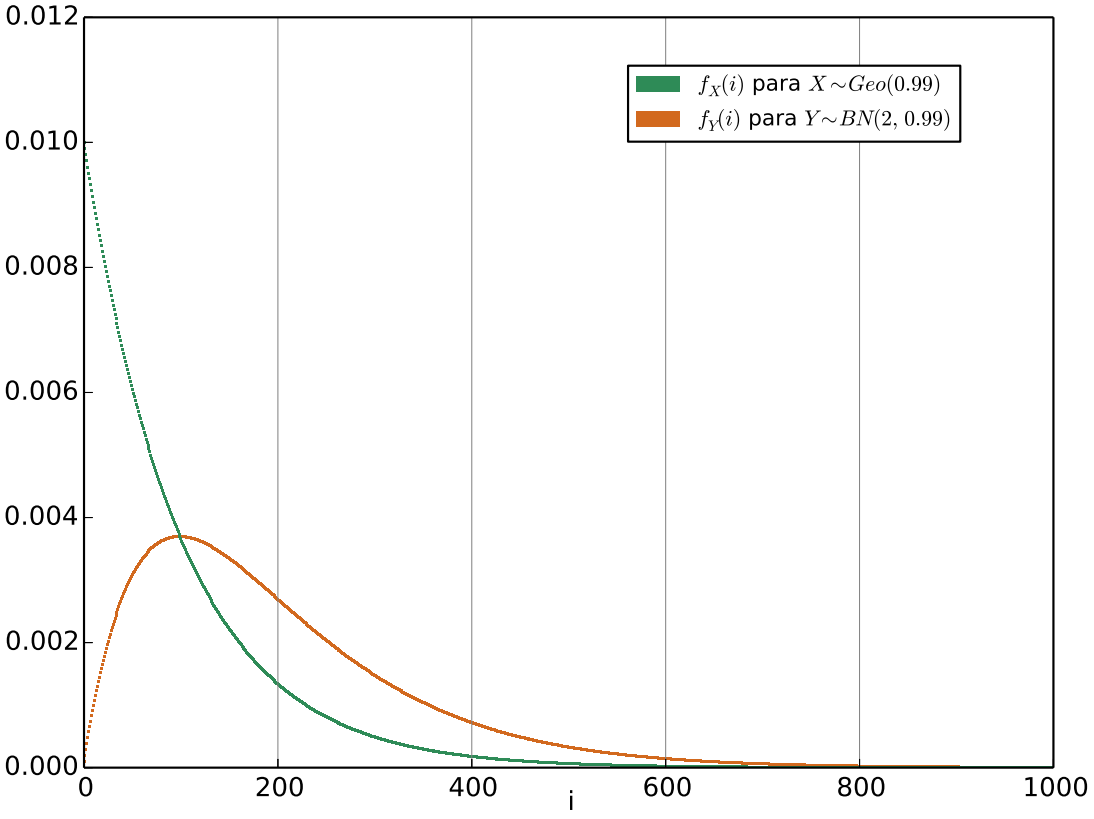

FIGURA B.5: $f_X(i)$ con $X \sim Geo(0.99)$ y $f_Y(i)$ con $Y \sim BN(2,\ 0.99)$, para $i \in \mathbb{Z}_{[0,1000]}$.

Para comparar f_X y f_Y, a continuación definimos la función g

$$\begin{aligned} g(i) &= f_X(i) - f_Y(i) \\ &\overset{(B.1,B.2)}{=} (1-p)p^i(1-(1-p)(i+1)) \\ &= (1-p)p^i(pi+p-i)\,, \quad i \in \mathbb{Z}_{\geq 0}. \end{aligned} \tag{B.5}$$

$g(i)$ se anula en el punto $i_1 = p/(1-p)$, con $g(i) > 0$ para $i < i_1$ y $g(i) < 0$ para $i > i_1$. Además, para cualquier valor del parámetro p tenemos $\lim_{i\to\infty} g(i) = 0^-$.

Apéndice C

Series matemáticas

En este apéndice presentamos las fórmulas de las series matemáticas utilizadas en los cálculos realizados durante el desarrollo del proyecto.

(C.1), (C.2) y (C.3) son fórmulas de **sumatorias finitas** para $p \in \mathbb{R}_{(0,1)}$.

$$\sum_{k=0}^{n} p^k = \frac{1-p^{n+1}}{1-p} \tag{C.1}$$

$$\sum_{k=0}^{n} kp^k = \frac{p(1-p^n)}{(1-p)^2} - \frac{np^{n+1}}{1-p} \tag{C.2}$$

$$\sum_{k=0}^{n} k^2p^k = \frac{p(1+p-(n+1)^2p^n+(2n^2+2n-1)p^{n+1}-n^2p^{n+2})}{(1-p)^3} \tag{C.3}$$

(C.4), (C.5), (C.6) y (C.7) son fórmulas de **sumatorias infinitas** para $p \in \mathbb{R}_{(0,1)}$, considerando el parámetro $\sigma \in \mathbb{Z}_{>0}$.

$$\sum_{k=0}^{\infty} p^k = \frac{1}{1-p} \tag{C.4}$$

$$\sum_{k=0}^{\infty} p^{\sigma k} = \sum_{k=0}^{\infty} (p^\sigma)^k \overset{(C.4)}{=} \frac{1}{1-p^\sigma} \tag{C.5}$$

$$\sum_{k=0}^{\infty} \sigma k p^{\sigma k} = p\sum_{k=0}^{\infty} \sigma k p^{\sigma k-1} = p\Big[\frac{d}{dp}\Big(\sum_{k=0}^{\infty} p^{\sigma k}\Big)\Big] \overset{(C.5)}{=} p\Big[\frac{d}{dp}\Big(\frac{1}{1-p^\sigma}\Big)\Big] = \frac{\sigma p^\sigma}{(1-p^\sigma)^2} \tag{C.6}$$

$$\sum_{k=0}^{\infty} \sigma k^2 p^{\sigma k} = p\sum_{k=0}^{\infty} \sigma k^2 p^{\sigma k-1} = p\Big[\frac{d}{dp}\Big(\sum_{k=0}^{\infty} kp^{\sigma k}\Big)\Big] \overset{(C.6)}{=} p\Big[\frac{d}{dp}\Big(\frac{p^\sigma}{(1-p^\sigma)^2}\Big)\Big] = \frac{\sigma p^\sigma(1+p^\sigma)}{(1-p^\sigma)^3} \tag{C.7}$$

Apéndice D

Codificador GolombBN: cálculo del parámetro l

En la sección 4.3 explicamos que el parámetro l del codificador GolombBN se calcula utilizando la fórmula (4.6). Para calcular la contribución a la media μ de los enteros $i \in \mathbb{Z}_{\geq\lambda}$ desarrollamos la siguiente ecuación

$$\begin{aligned}\sum_{i=\lambda}^{\infty} i \cdot \mathrm{P}(Y=i) &\overset{(4.1)}{=} (1-p)^2 \sum_{i=\lambda}^{\infty} i(i+1)p^i \\ &= (1-p)^2 \Big(\sum_{i=0}^{\infty} i(i+1)p^i - \sum_{i=0}^{\lambda-1} i(i+1)p^i \Big) \\ &= (1-p)^2 \Big(\sum_{i=0}^{\infty} i^2 p^i + \sum_{i=0}^{\infty} i p^i - \sum_{i=0}^{\lambda-1} i^2 p^i - \sum_{i=0}^{\lambda-1} i p^i \Big).\end{aligned} \tag{D.1}$$

Sustituyendo las cuatro sumatorias por las respectivas fórmulas (C.7), (C.6), (C.3) y (C.2), tenemos

$$\begin{aligned}\sum_{i=\lambda}^{\infty} i \cdot \mathrm{P}(Y=i) = (1-p)^2 \Big(& \frac{p(1+p)}{(1-p)^3} + \frac{p}{(1-p)^2} \\ & - \frac{p(1+p-\lambda^2 p^{\lambda-1} + (2(\lambda-1)^2 + 2(\lambda-1) - 1)p^\lambda - (\lambda-1)^2 p^{\lambda+1})}{(1-p)^3} \\ & - \frac{p(1-p^{\lambda-1})}{(1-p)^2} + \frac{(\lambda-1)p^\lambda}{1-p} \Big) \\ = \frac{p^\lambda}{1-p} & \Big(\lambda(\lambda-1)p^2 - 2(\lambda^2-1)p + \lambda(\lambda+1) \Big).\end{aligned} \tag{D.2}$$

Apéndice E

Codificador GolombBN: cálculo del largo medio de código

En este apéndice desarrollamos la ecuación (4.10), utilizada para calcular el largo medio del código GolombBN en la sección 4.4.

$$\begin{aligned}
\sum_{\substack{i=\lambda \\ i\equiv j \pmod{\beta}}}^{\infty} \mathrm{P}(Y=i)\cdot l_{GBN}(i) &= \sum_{k=0}^{\infty} \mathrm{P}(Y=j+lk)\cdot l_{GBN}(j+lk) \\
&\overset{(4.1)}{=} \sum_{k=0}^{\infty}(1-p)^2(j+lk+1)p^{j+lk}\cdot l_{G_l}(j+lk) \\
&= (1-p)^2p^j\sum_{k=0}^{\infty}(j+lk+1)p^{lk}\cdot(l_{G_l}(j)+k) \\
&= (1-p)^2p^j\Big(\sum_{k=0}^{\infty}(j+lk+1)kp^{lk}+l_{G_l}(j)\sum_{k=0}^{\infty}(j+lk+1)p^{lk}\Big) \\
&\overset{(E.2),(G.1)}{=} (1-p)^2p^j\Big(\frac{(-j+l-1)p^{2l}+(j+l+1)p^l}{(1-p^l)^3} \\
&\qquad + l_{G_l}(j)\frac{(-j+l-1)p^l+j+1}{(1-p^l)^2}\Big) \\
&= \frac{(1-p)^2p^j}{(1-p^l)^3}\Big((l_{G_l}(j)-1)(j-l+1)p^{2l} \\
&\qquad + (j+l+1+l_{G_l}(j)\cdot(-2j+l-2))p^l \\
&\qquad + l_{G_l}(j)(j+1)\Big)
\end{aligned} \tag{E.1}$$

Para resolver la ecuación (E.1) utilizamos la siguiente fórmula

$$\begin{aligned}
\sum_{k=0}^{\infty}(j+lk+1)kp^{lk} &= (j+1)\sum_{k=0}^{\infty}kp^{lk}+\sum_{k=0}^{\infty}lk^2p^{lk} \\
&\overset{(C.6),(C.7)}{=} \frac{(j+1)p^l}{(1-p^l)^2}+\frac{lp^l(1+p^l)}{(1-p^l)^3} \\
&= \frac{(-j+l-1)p^{2l}+(j+l+1)p^l}{(1-p^l)^3}.
\end{aligned} \tag{E.2}$$

Apéndice F

Codificador T: fuente reducida

En la sección 5.1 definimos una fuente reducida con $\alpha + \beta$ símbolos. En la ecuación (F.1) demostramos que la variable aleatoria Y', asociada a dicha fuente, cumple que la suma de las probabilidades de todos sus símbolos es 1.

$$\begin{aligned}
\sum_{i=0}^{\alpha+\beta-1} \mathrm{P}(Y'=j) &\overset{(5.4)}{=} \sum_{i=0}^{\alpha-1} \mathrm{P}(Y=j) + \sum_{j=\alpha}^{\alpha+\beta-1} \left\{ \sum_{k=0}^{\infty} \mathrm{P}(Y=j+\beta k) \right\} \\
&\overset{(5.1),(G.1)}{=} \sum_{i=0}^{\alpha-1} (1-p)^2 (i+1) p^i + \sum_{j=\alpha}^{\alpha+\beta-1} (1-p)^2 p^j \frac{(-j+\beta-1)p^\beta + j + 1}{(1-p^\beta)^2} \\
&= (1-p)^2 \left\{ \sum_{i=0}^{\alpha-1} (i+1) p^i + \frac{1}{(1-p^\beta)^2} \sum_{j=\alpha}^{\alpha+\beta-1} p^j (-j+\beta-1)p^\beta + j + 1 \right\} \\
&\overset{(F.2),(F.3)}{=} (1-p) \left\{ \frac{p(1-p^{\alpha-1})}{1-p} + 1 - \alpha p^\alpha + p^\alpha \left[\frac{1}{1-p} + \alpha \right] \right\} \\
&= (1-p) \left\{ \frac{p(1-p^{\alpha-1}) + p^\alpha}{1-p} + 1 - \alpha p^\alpha + \alpha p^\alpha \right\} \\
&= (1-p) \left\{ \frac{p}{1-p} + 1 \right\} \\
&= 1 \quad \square
\end{aligned} \tag{F.1}$$

Las ecuaciones (F.2) y (F.3) tienen cálculos auxiliares utilizados para resolver la ecuación (F.1).

$$\begin{aligned}
\sum_{i=0}^{\alpha-1} (i+1) p^i &= \sum_{i=0}^{\alpha-1} i p^i + \sum_{i=0}^{\alpha-1} p^i \\
&\overset{(C.1),(C.2)}{=} \frac{p(1-p^{\alpha-1})}{(1-p)^2} - \frac{(\alpha-1)p^\alpha}{1-p} + \frac{1-p^\alpha}{1-p} \\
&= \frac{p(1-p^{\alpha-1})}{(1-p)^2} + \frac{1-\alpha p^\alpha}{1-p} \\
&= \frac{1}{1-p} \left[\frac{p(1-p^{\alpha-1})}{1-p} + 1 - \alpha p^\alpha \right]
\end{aligned} \tag{F.2}$$

$$\begin{aligned}\sum_{j=\alpha}^{\alpha+\beta-1} p^j(-j+\beta-1)p^\beta + j + 1 &= ((\beta-1)p^\beta+1)\sum_{i=\alpha}^{\alpha+\beta-1} p^j + (1-p^\beta)\sum_{i=\alpha}^{\alpha+\beta-1} jp^j \\ &\overset{(F.4),(F.5)}{=} \frac{p^\alpha(1-p^\beta)}{1-p}\Bigg[(\beta-1)p^\beta+1 \\ &\qquad + \frac{1-p^\beta}{1-p} + \alpha - 1 - (\alpha+\beta-1)p^\beta\Bigg] \\ &= \frac{p^\alpha(1-p^\beta)}{1-p}\left[\frac{1-p^\beta}{1-p} + \alpha - \alpha p^\beta\right] \\ &= \frac{p^\alpha(1-p^\beta)^2}{1-p}\left[\frac{1}{1-p} + \alpha\right]\end{aligned} \tag{F.3}$$

Las ecuaciones (F.4) y (F.5) tienen cálculos auxiliares utilizados para resolver la ecuación (F.3).

$$\begin{aligned}\sum_{i=\alpha}^{\alpha+\beta-1} p^j &= \sum_{i=0}^{\alpha+\beta-1} p^j - \sum_{i=0}^{\alpha-1} p^j \\ &\overset{(C.1)}{=} \frac{1-p^{\alpha+\beta}}{1-p} - \frac{1-p^\alpha}{1-p} \\ &= \frac{p^\alpha - p^{\alpha+\beta}}{1-p} \\ &= \frac{p^\alpha}{1-p}(1-p^\beta)\end{aligned} \tag{F.4}$$

$$\begin{aligned}\sum_{i=\alpha}^{\alpha+\beta-1} jp^j &= \sum_{i=0}^{\alpha+\beta-1} jp^j - \sum_{i=0}^{\alpha-1} jp^j \\ &\overset{(C.2)}{=} \frac{p(1-p^{\alpha+\beta-1})}{(1-p)^2} - \frac{(\alpha+\beta-1)p^{\alpha+\beta}}{1-p} - \frac{p(1-p^{\alpha-1})}{(1-p)^2} + \frac{(\alpha-1)p^\alpha}{1-p} \\ &= \frac{p^\alpha - p^{\alpha+\beta}}{(1-p)^2} + \frac{(\alpha-1)p^\alpha - (\alpha+\beta-1)p^{\alpha+\beta}}{1-p} \\ &= \frac{p^\alpha}{1-p}\left[\frac{1-p^\beta}{1-p} + \alpha - 1 - (\alpha+\beta-1)p^\beta\right]\end{aligned} \tag{F.5}$$

Apéndice G

Codificador T: cálculo de la probabilidad de un supersímbolo

En este apéndice presentamos los cálculos realizados para obtener la probabilidad de un supersímbolo del código T. Consideramos los parámetros $p \in \mathbb{R}_{(0,1)}$, $\alpha \in \mathbb{Z}_{\geq 0}$ y $\beta \in \mathbb{Z}_{>0}$. Nos interesa obtener una expresión analítica de la sumatoria infinita planteada en la ecuación (5.4), que corresponde a la probabilidad de un supersímbolo j, tal que $\alpha \leq j < \alpha + \beta$. Esta expresión analítica es desarrollada en la siguiente ecuación

$$\begin{aligned}
\mathrm{P}(Y' = j) &\overset{(5.4)}{=} \sum_{k=0}^{\infty} \mathrm{P}(Y = j + \beta k) \\
&\overset{(5.1)}{=} \sum_{k=0}^{\infty} (1-p)^2 (j + \beta k + 1) p^{j+\beta k} \\
&= (1-p)^2 p^j \sum_{k=0}^{\infty} (j + \beta k + 1) p^{\beta k} \\
&= (1-p)^2 p^j \Big((j+1) \sum_{k=0}^{\infty} p^{\beta k} + \sum_{k=0}^{\infty} \beta k p^{\beta k} \Big) \\
&\overset{(C.5),(C.6)}{=} (1-p)^2 p^j \Big(\frac{j+1}{1-p^\beta} + \frac{\beta p^\beta}{(1-p^\beta)^2} \Big) \\
&= (1-p)^2 p^j \, \frac{(-j + \beta - 1) p^\beta + j + 1}{(1-p^\beta)^2},
\end{aligned} \tag{G.1}$$

en donde utilizamos las fórmulas auxiliares (C.5) y (C.6).

Apéndice H

Codificador T: cálculo del largo medio de código

En este apéndice desarrollamos la ecuación (5.9), utilizada para calcular el largo medio del código T en la sección 5.3.

$$\begin{aligned}\sum_{\substack{i=\alpha \\ i\equiv j \pmod{\beta}}}^{\infty} \mathrm{P}(Y=i)\cdot l_{\mathrm{T}}(i) &\overset{(5.4)}{=} \sum_{k=0}^{\infty}\mathrm{P}(Y=j+\beta k)\cdot l_{\mathrm{T}}(j+\beta k)\\ &\overset{(5.1)}{=} \sum_{k=0}^{\infty}(1-p)^2(j+\beta k+1)p^{j+\beta k}\cdot(l_{CH_{Y'}}(j)+k+1)\\ &= (1-p)^2p^j\sum_{k=0}^{\infty}(j+\beta k+1)p^{\beta k}\cdot(l_{CH_{Y'}}(j)+k+1)\\ &= (1-p)^2p^j\Big(\sum_{k=0}^{\infty}(j+\beta k+1)kp^{\beta k}+(l_{CH_{Y'}}(j)+1)\sum_{k=0}^{\infty}(j+\beta k+1)p^{\beta k}\Big)\end{aligned} \tag{H.1}$$

Utilizando las ecuaciones (G.1) y (E.2) la última igualdad de la ecuación (H.1) nos queda

$$(1-p)^2p^j\Big(\frac{(-j+\beta-1)p^{2\beta}+(j+\beta+1)p^{\beta}}{(1-p^{\beta})^3}+(l_{CH_{Y'}}(j)+1)\frac{(-j+\beta-1)p^{\beta}+j+1}{(1-p^{\beta})^2}\Big), \tag{H.2}$$

y agrupando términos obtenemos

$$\begin{aligned}\frac{(1-p)^2p^j}{(1-p^{\beta})^3}\Big(&l_{CH_{Y'}}(j)\cdot(j-\beta+1)p^{2\beta}+(-j+2\beta-1+l_{CH_{Y'}}(j)\cdot(-2j+\beta-2))p^{\beta}\\ &+(l_{CH_{Y'}}(j)+1)(j+1)\Big).\end{aligned} \tag{H.3}$$

Bibliografía

[1] Vernon Turner, David Reinsel, John F. Gantz, and Stephen Minton. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. http://idcdocserv.com/1678, Abril 2014. [Dirección web. Accedida el 13-12-2015].

[2] Josh Coalson. FLAC format. https://xiph.org/flac/format.html, 2014. [Dirección web. Accedida el 13-12-2015].

[3] Amandine Pras, Rachel Zimmerman, Daniel Letvin, and Catherine Guastavino. Subjective evaluation of MP3 compression for different musical genres. In *Audio Engineering Society, 127th Convention 2009*, pages 1–7, 2009.

[4] Scot Hacker. *MP3: The Definitive Guide*. O'Reilly Media, 1st edition, 2000.

[5] Ty Pendlebury. Spotify vs. Apple Music: Is there a difference in sound quality? http://www.cnet.com/news/apple-music-vs-spotify-is-there-a-difference-in-sound-quality/, 5-07-2014. [Dirección web. Accedida el 13-12-2015].

[6] Geoffrey Morrison. What is HEVC? High Efficiency Video Coding, H.265, and 4K compression explained. http://www.cnet.com/news/what-is-hevc-high-efficiency-video-coding-h-265-and-4k-compression-explained/, 18-04-2014. [Dirección web. Accedida el 13-12-2015].

[7] Michael Robin and Michel Poulin. *Digital Television Fundamentals*. McGraw-Hill Education, 2nd edition, 2000.

[8] Mo Ladha. Skype's Pursuit of the Perfect Video Call. http://blogs.skype.com/2014/01/06/skypes-pursuit-of-the-perfect-video-call/, 6-01-2014. [Dirección web. Accedida el 13-12-2015].

[9] DSLR Camera Basics: Image Quality and Size. http://imaging.nikon.com/lineup/dslr/basics/26/01.htm, 2014. [Dirección web. Accedida el 13-12-2015].

[10] Image compression: Lossless and lossy compression. http://cpn.canon-europe.com/content/education/infobank/image_compression/lossless_and_lossy_compression.do, 2015. [Dirección web. Accedida el 13-12-2015].

[11] GNU Gzip. https://www.gnu.org/software/gzip/, 2010. [Dirección web. Accedida el 13-12-2015].

[12] WinRAR at a glance. http://www.win-rar.com/features.html?&L=0, 2015. [Dirección web. Accedida el 13-12-2015].

[13] Alexander J. Casson, Shelagh Smith, John S. Duncan, and Esther Rodriguez-Villegas. Wearable EEG: what is it, why is it needed and what does it entail? In *Engineering in medicine and biology society, 2008. embs 2008. 30th annual international conference of the ieee*, pages 5867–5870. IEEE, 2008.

[14] Fabiola Czubaj. Un equipo rioplatense creó el sistema que comprime las fotos. http://www.lanacion.com.ar/563203-un-equipo-rioplatense-creo-el-sistema-que-comprime-las-fotos, 12-01-2004. [Dirección web. Accedida el 13-12-2015].

[15] Jamie Beckett. HP on Mars: Labs technology used to send most accurate images possible. http://www.hpl.hp.com/news/2004/jan-mar/hp_mars.html, Enero 2004. [Dirección web. Accedida el 13-12-2015].

[16] Mia Tramz. Photographing Pluto: This Is How New Horizons Works. http://www.time.com/3944157/new-horizons-pluto/, 14-07-2015. [Dirección web. Accedida el 13-12-2015].

[17] The Johns Hopkins University Applied Physics Laboratory LLC. Science Operations Center. http://www.pluto.jhuapl.edu/Pluto/Science-Operations-Center.php, 2015. [Dirección web. Accedida el 13-12-2015].

[18] Robert G. Gallager and David C. van Voorhis. Optimal source codes for geometrically distributed integer alphabets (Corresp.). *IEEE Transactions on Information Theory*, 21(2), 1975.

[19] Solomon W. Golomb. Run-length encodings (Corresp.). *IEEE Transactions on Information Theory*, 12(3), 1966.

[20] Marcelo J. Weinberger, Gadiel Seroussi, and Guillermo Sapiro. The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. *IEEE Transactions on Image Processing*, 9(8):1309–1324, 2000.

[21] Tony Robinson. SHORTEN: Simple lossless and near-lossless waveform compression. *Technical Report 156, Engineering Department, Cambridge University*, Dec. 1994.

[22] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[23] Brockway McMillan. Two inequalities implied by unique decipherability. *IRE Transactions on Information Theory*, 2(4), 1956.

[24] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9), 1952.

[25] Robert F. Rice. Some Practical Universal Noiseless Coding Techniques—Parts I-III. *Jet Propulson Lab., Pasadena, CA, Tech. Reps. JPL-79-22, JPL-83-17, and JPL-91-3*, Mar. 1979, Mar. 1983, Nov. 1991.

[26] David Slepian and Jack K. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19(4), 1973.

[27] Abraham J. Wyner and Jacob Ziv. The rate-distortion function for source coding with side information at the decoder. *IEEE Transactions on Information Theory*, 22(1), 1976.

[28] Robert G. Gallager. Source coding with side information and universal coding. *Proc. IEEE Int. Symp. Information Theory*, 1976.

[29] Frédérique Bassino, Julien Clément, Gadiel Seroussi, and Alfredo Viola. Optimal prefix codes for pairs of geometrically distributed random variables. *IEEE Transactions on Information Theory*, 59(4):2375–2395, 2013.

[30] Aaron B. Kiely. Selecting the Golomb Parameter in Rice Coding. *IPN Progress Report, vol. 42-159, pp. 1–8*, 2, November 15, 2004.

[31] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.

[32] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 4th edition, 2012.

[33] Khalid Sayood. *Lossless Compression Handbook*. Academic Press, 2002.

[34] M. Hans and R. W. Schafer. Lossless compression of digital audio. *IEEE Signal Processing Magazine*, 18(4):21–32, 2001.

Glosario

Código Sea $\mathcal{D}$ un alfabeto, $\mathcal{D}^*$ su extensión y X una variable aleatoria que genera símbolos del alfabeto $\mathcal{X} = \{x_1, ..., x_n\}$. Un *código* C para X es un mapeo $C : \mathcal{X} \to \mathcal{D}^*$. Decimos que el código C *codifica* un símbolo x_i con la *palabra de código* $C(x_i)$. Con $l_C(x_i)$ denotamos el largo de $C(x_i)$.

Códigos de Golomb Son una familia de códigos instantáneos óptimos para codificar fuentes con distribución geométrica. Dentro de la familia, un código de Golomb particular queda definido por un parámetro entero $l \in \mathbb{Z}_{>0}$. Utilizamos la notación G_l para el código de Golomb con parámetro l.

Códigos de Golomb GPO2 Son códigos de Golomb cuyo parámetro l es potencia de 2. Utilizamos la notación $\mathcal{G}_k$ para el código GPO2 con parámetro $l = 2^k$, con $k \in \mathbb{Z}_{\geq 0}$.

Código de Huffman Utilizando el algoritmo de Huffman es posible construir un código instantáneo óptimo para una distribución de probabilidad conocida sobre un alfabeto finito. Al código construido se le llama *código de Huffman*. Utilizamos la notación $CH_{\mathbf{p}}$ para designar al código obtenido al aplicar el algoritmo sobre una distribución de probabilidad dada por un vector $\mathbf{p} = (p_1, .., p_n)$. Además, con la notación CH_X designamos al código de Huffman construido para una variable aleatoria X cuyas probabilidades son conocidas.

Código GolombBN Código implementado en el proyecto, utilizado para codificar una variable aleatoria con distribución binomial negativa. Tiene dos parámetros, $l \in \mathbb{Z}_{>0}$ y $\lambda \in \mathbb{Z}_{>0}$. Su implementación es sencilla y rápida desde el punto de vista computacional, pero está lejos de ser un código óptimo.

Código óptimo Decimos que C es un *código óptimo* para una fuente si el largo medio obtenido al codificarla con el código C es menor o igual al largo medio obtenido al codificarla con cualquier otro código.

Código T Código implementado en el proyecto, utilizado para codificar una variable aleatoria con distribución binomial negativa. Tiene dos parámetros, $\alpha \in \mathbb{Z}_{\geq 0}$ y $\beta \in \mathbb{Z}_{>0}$. Su implementación es compleja y costosa desde el punto de vista computacional, pero está cerca de ser un código óptimo.

Código unario El código unario $U : \mathbb{Z}_{\geq 0} \to \mathcal{B}^*$ codifica un entero no negativo i con una palabra de código formada por una cadena de i ceros seguida de un uno. La palabra de código $U(i)$ tiene largo $l_U(i) = i + 1$. A modo de ejemplo, tenemos $U(0) = 1$, $U(1) = 01$, $U(2) = 001$ y $U(3) = 0001$.

Distribución binomial negativa (con parámetros 2 y p) Consideramos una secuencia de ensayos de Bernoulli independientes con idéntica probabilidad de éxito $p \in \mathbb{R}_{(0,1)}$. Definimos la variable aleatoria $Y \in \mathbb{Z}_{\geq 0}$ como el número de ensayos exitosos realizados antes de que ocurran los primeros dos fallos. Decimos que la variable aleatoria Y sigue una *distribución binomial negativa* con parámetros 2 y p, y nos referimos a ella utilizando la notación $Y \sim BN(2, p)$. La distribución de probabilidad de Y está dada por

$$\mathrm{P}(Y = i) = (1-p)^2(i+1)p^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{H.4}$$

Distribución geométrica Consideramos una secuencia de ensayos de Bernoulli independientes con idéntica probabilidad de éxito $\theta \in \mathbb{R}_{(0,1)}$. Definimos la variable aleatoria $X \in \mathbb{Z}_{\geq 0}$ como el número de ensayos exitosos realizados antes de que ocurra el primer fallo. Decimos que la variable aleatoria X sigue una *distribución geométrica* con parámetro θ, y nos referimos a ella utilizando la notación $X \sim Geo(\theta)$. La distribución de probabilidad de X está dada por

$$\mathrm{P}(X = i) = (1-\theta)\theta^i\,, \quad i \in \mathbb{Z}_{\geq 0}. \tag{H.5}$$

Entropía Sea X una variable aleatoria que genera símbolos del alfabeto $\mathcal{X}$ siguiendo una función de probabilidad $p(x) = \mathrm{P}(X = x)$, $x \in \mathcal{X}$. Definimos la *entropía* de X como

$$\begin{aligned} H(X) &= E_p\Big[-\log_2 p(X)\Big] \\ &= -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x), \end{aligned} \tag{H.6}$$

donde E_p denota la esperanza con respecto a p. La entropía se expresa en bits y al calcularla utilizamos la convención $0 \log_2 0 = 0$.

Largo medio de código Sea X una variable aleatoria que genera símbolos del alfabeto $\mathcal{X}$ siguiendo una función de probabilidad $p(x)$. Definimos el *largo medio* de un código C para X como el valor esperado de $l_C(X)$, es decir

$$L(C) = \sum_{x_i \in \mathcal{X}} p(x_i) l_C(x_i). \tag{H.7}$$

Primer Teorema de Shannon (extensión) Sea C un código binario instantáneo óptimo para una variable aleatoria X sobre un alfabeto $\mathcal{X}$. Entonces su largo medio $L(C)$ cumple

$$H(X) \leq L(C) < H(X) + 1, \tag{H.8}$$

con $L(C) = H(X)$ si y solo si se tiene $p(x_i) = 2^{-l_C(x_i)}$ para todo $x_i \in \mathcal{X}$.