EL4005 Principios de Comunicaciones

Clase No. 12: Algortimos de Codificación de Fuente

Patricio Parada | Néstor Becerra Yoma

Departamento de Ingeniería Eléctrica
Universidad de Chile

23 de Noviembre de 2011

Contenidos de la Clase

Codificadores de Fuente (1)

- En lo que sigue vamos a suponer que vamos a trabajar con fuentes discretas de información.
- Un codificador de fuente es un dispositivo que realiza una asignación entre el alfabeto de símbolos emitidos por una fuente de información $\mathcal{U}$ y una código $\mathcal{X}$.

$$\phi : \mathcal{U} \to \mathcal{X} \tag{1}$$

- El diseño de un codificador de fuente tiene varios grados de libertad. Entre ellos:
  - La **longitud de las palabras de código** (pueden ser de largo fijo e igual para todas las secuencias o de largo variable).
  - La **inyectividad del mapeo**. Si $\phi(\cdot)$ es inyectiva entonces la codificación es sin pérdidas, y si no se dice que produce pérdidas.

Codificadores de Fuente (2)

- Si asumimos que los símbolos pueden ser descritos mediante secuencias binarias de largo finito (k para la fuente y n para el código) entonces

$$\phi : \{0,1\}^k \to \{0,1\}^n$$
$$\mathbf{u} = (u_1, \ldots, u_k) \mapsto \mathbf{x} = (x_1, \ldots, x_n)$$

- La tasa de codificación se define como

$$R \doteq \frac{k}{n}. \tag{2}$$

- Diremos que la codificación es compresiva si $R > 1$

Teorema de Codificación de Fuente

- El resultado central que afecta al diseño de codificadores se denomina el **Teorema de Codificación de Fuente**.
- Establece dos hechos relativos a codificar a una tasa R bits/símbolo una fuente U con entropía H:
  - Si $H(U) < R$, entonces la probabilidad de cometer errores en la decodificación se va a 0 a medida que aumenta la longitud de las palabras de código.
  - Si $H(U) > R$ esta misma probabilidad está acotada lejos de 0, independiente de la complejidad del codificador.
- Este teorema, como otros de Shannon, provee de condiciones necesarias y suficientes para encontrar este código, pero no de una manera de construir tal código.

Algoritmo de Huffman (1)

- El objetivo de este algoritmo es codificar secuencias de símbolos de fuente de largo constante $\mathbf{U} = \{a_1, \ldots, a_m\}$ en secuencias de largo variable, que al ser utilizadas tengan un largo promedio menor.
- Este tipo de codificación recibe el nombre de *codificación fija-a-varaible*.
- La idea del algoritmo es mapear los símbolos que aparecen más frecuentemente a palabras de código de menor longitud, mientras que aquellas que aparecen menos frecuentemente deberán ser asignadas a palabras de código de mayor longitud.

Algoritmo de Huffman (2)

Paso 1. Ordenar los símbolos de fuente en orden decreciente según sus probabilidades de aparición.

Paso 2. Combinar los 2 símbolos menos probables en un sólo símbolo, cuya probabilidad es la suma de las probabilidades de los símbolos originales.

Paso 3. Si el número de símbolos restantes es 2, seguir al punto 4; Si no, volver a 1.

Paso 4. Asignar arbitrariamente 0 y 1 como palabras de código a las dos símbolos resultantes.

Algoritmo de Huffman (3)

Paso 5.
- Si una salida es el resultado de la combinación de dos símbolos de fuente, agregar 0 y 1 a la palabra que lo precede y repetir 5.
- Si la salida no es precedida por otra salida, detener el algoritmo.

Ejemplo de Aplicación del Algoritmo de Huffman (1)

- Considere un alfabeto de 7 símbolos $\{A, B, C, D, E, F, G\}$ cuyas probabilidades de aparición son las siguientes:

Símbolo	p_j
A	$3/8$
B	$3/16$
C	$3/16$
D	$1/8$
E	$1/16$
F	$1/32$
G	$1/32$

(3)

Ejemplo de Aplicación del Algoritmo de Huffman (2)

- Si utilizáramos un código de largo fijo, necesitaríamos símbolos de longitud:

$$l = \lceil \log_2 7 \rceil = \lceil 2{,}81 \rceil = 3 \text{ bits/símbolo} \tag{4}$$

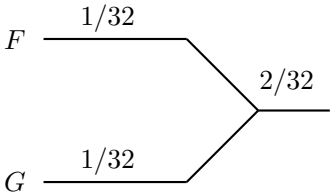
- La máxima compresión que podemos obtener está dada por la entropía de la fuente

$$H(\mathbf{p}) = -\sum_{i=1}^{7} p_j \log_2 p_j = 2{,}37 \text{ bits/símbolo} \tag{5}$$

- Ganancia de compresión teórica: $26{,}37\,\%$.

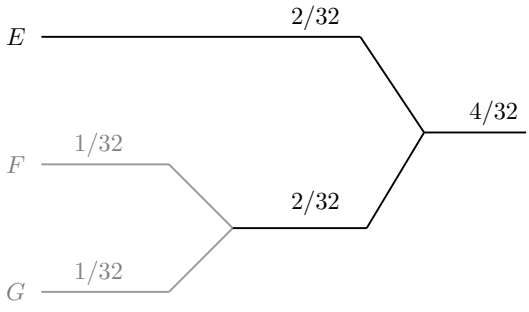
Ejemplo de Aplicación del Algoritmo de Huffman (3)

- La lista ya se encuentra ordenada de manera decreciente de acuerdo a las probabilidades de aparición de los símbolos.
- Comenzamos a agrupar los símbolos en parejas
- Primero F y G se combinan en un símbolo con probabilidad $2/32$.



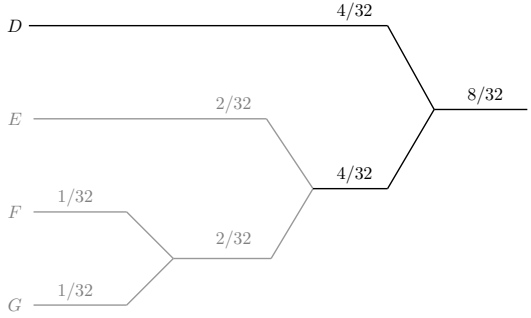
Ejemplo de Aplicación del Algoritmo de Huffman (4)

- Luego el símbolo resultante lo combinamos con E:



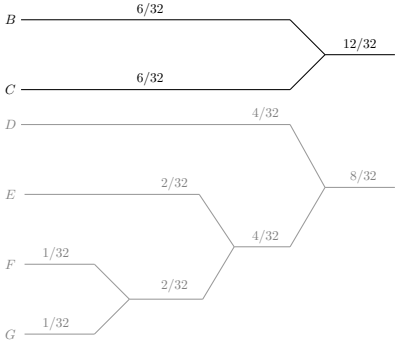
Ejemplo de Aplicación del Algoritmo de Huffman (5)

- Luego el símbolo resultante lo combinamos con D:



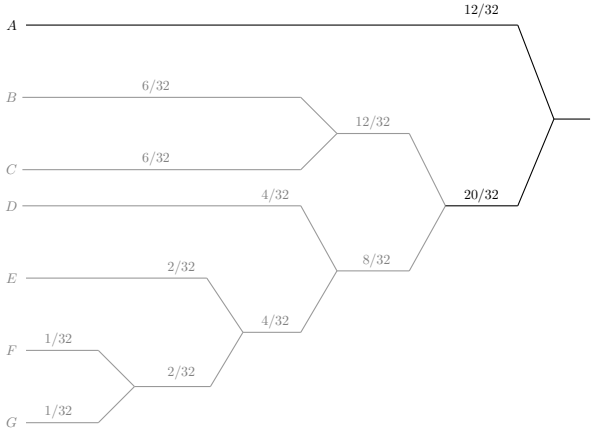
Ejemplo de Aplicación del Algoritmo de Huffman (6)

- Notamos que la combinación de símbolos D, E, F, G tiene mayor probabilidad que B o C, por lo que estos símbolos ahora son los menos probables y deben agruparse:



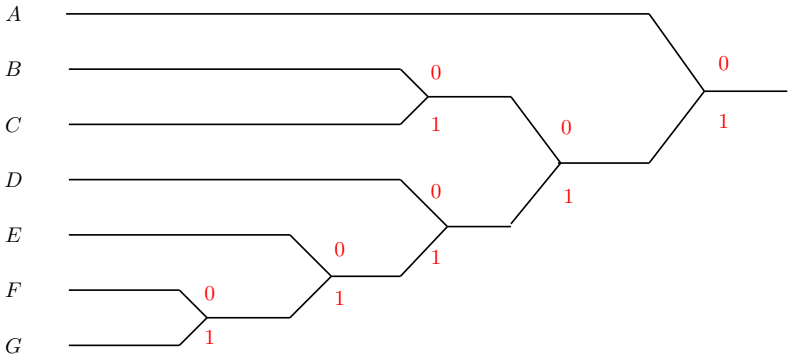
Ejemplo de Aplicación del Algoritmo de Huffman (7)

- Finalmente,



Ejemplo de Aplicación del Algoritmo de Huffman (8)

- Una vez construido este árbol, podemos etiquetar las ramas de modo de construir las palabras de código para cada símbolo.



Ejemplo de Aplicación del Algoritmo de Huffman (9)

- El código resultante es el siguiente:

Símbolo	Código
A	0
B	100
C	101
D	110
E	1110
F	11110
G	11111

(6)

Ejemplo de Aplicación del Algoritmo de Huffman (10)

- Si asumimos que los símbolos aparecen de acuerdo a la distribución indicada en el ejemplo, entonces es número promedio de bits requeridos para representar la fuente es

$$\begin{aligned}\mathbb{E}[l] &= \sum_{j=1}^{7} p_j l(a_j) \\ &= \frac{3}{8} * 1 + \frac{3}{16} * 3 + \frac{3}{16} * 3 + \frac{1}{8} * 3 + \frac{1}{16} * 4 + \frac{1}{32} * 5 + \frac{1}{32} * 5 \\ &= 2{,}4375 \text{ bits/símbolo}\end{aligned}$$

- La compresión real lograda es de un $23{,}08\,\%$ cuando la codificación se hace símbolo a símbolo.

La Optimalidad del Algoritmo de Huffman (1)

- Notemos primero que el número promedio de bits $\mathbb{E}[l]$ utilizado en la codificación de Huffman satisface:

$$H(X) \leq \mathbb{E}[l] = H(X) + 1 \tag{7}$$

- Este resultado se basa en dos resultados:
- La aplicación de la desigualdad de Kraft:

$$\sum_{m=0}^{K-1} K^{-l_m} \leq 1 \tag{8}$$

donde el alfabeto de fuente es $\{a_0, \ldots, a_{K-1}\}$ y l_m es el número de símbolos empleados para codificar a_m.

La Optimalidad del Algoritmo de Huffman (2)

- El hecho que

$$\mathbb{E}[l] \geq H(\mathbf{p}) \tag{9}$$

y que

$$\mathbb{E}[l] \leq H(\mathbf{p}) + 1 \tag{10}$$

- En efecto, si

$$l_m = \lceil -\log p_j \rceil \tag{11}$$

al aplicar la desigualdad de Kraft se obtiene la cota.

La Optimalidad del Algoritmo de Huffman (3)

- Notemos que la idea del algoritmo de Huffman puede ser aplicada con bloques de símbolos de la fuente, es decir:

$$\mathbf{X}^n = (X_1, \ldots, X_n) \tag{12}$$

donde $X_j \in \{a_0, \ldots, a_{J-1}\}$.

- Si la fuente no tiene memoria, entonces

$$\lim_{n\to\infty} \mathbb{E}[l(n)] = H(X) = H(X_1). \tag{13}$$

Codificación Universal (1)

- Hasta ahora hemos visto que para codificar una fuente necesitamos conocer la distribución de probabilidad sobre los símbolos, $\mathbf{p} = (p_0, \ldots, p_{J-1})$.
- Sin embargo, en la práctica esta información se conoce de manera parcial o no se conoce.
- ¿Podemos construir un código que compacte la fuente de manera óptima o cercana a ella?
- La respuesta es "Sí", y estos códigos reciben el nombre de **código universal**.

Codificación Universal (2)

- Recordemos que una fuente de información queda completamente determinada por la medida de probabilidad $\mathbf{p}$ que la caracteriza.
- Asumamos que $\mathbf{p}$ puede ser parametrizada en términos de $\boldsymbol{\theta}$, de manera cada $\boldsymbol{\theta}$ identifica de manera única una fuente.
- Sea Θ denote la clase de fuentes de información descrita por $\mathbf{p}(\cdot|\boldsymbol{\theta})$.
- Este conjunto representa la información *a priori* que conocemos de la fuente. Si sabemos nada, entonces $\Theta = \mathcal{P}^{J^n}$.
- Luego, podemos construir un código binario de largo variable siguiendo el procedimiento de Huffman u otro similar.

Codificación Universal (3)

- Diremos que un **código es universal** si el largo promedio $\bar{l}_n(\boldsymbol{\theta})$ de una palabra de código asociada a un bloque de n símbolos de fuente $\mathbf{x} = (x_1, \ldots, x_n)$ satisface

$$\left| \frac{1}{n} \bar{l}(\boldsymbol{\theta}) - H_n(\theta) \right| < \epsilon \tag{14}$$

para cualquier $\epsilon > 0$ y cualquier fuente dentro la clase Θ, donde

$$H_n(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{\mathbf{x}} p^n(\mathbf{x}|\boldsymbol{\theta}) \log p^n(\mathbf{x}|\boldsymbol{\theta}).$$

y $p^n(\mathbf{x}|\boldsymbol{\theta})$ denota la distribución conjunta de $\mathbf{x}$.

Construcción Práctica de un Código Universal (1)

- ¿Cómo podemos utilizar este resultado para construir un código universal?
- El no tener información de la distribución de probabilidad de símbolos significa que no tenemos cómo modelar la fuente de manera precisa.
- P: ¿Qué sabemos?
- I1: La fuente emite símbolos de un alfabeto discreto $\{a_0, a_1, \ldots, a_{J-1}\}$.
- I2: Sabemos que $\mathcal{P}^J$ contiene todas las fuentes de información posibles que se puede definir sobre este alfabeto.
- I3: Luego, $\mathcal{P}^{J^n}$ contiene todas las fuentes que codifican bloques de n símbolos de fuente.

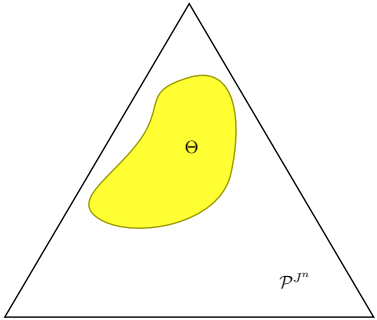
Construcción Práctica de un Código Universal (2)

- Podemos definir un subconjunto de $\mathcal{G} \subset \mathcal{P}^{J^n}$ que de alguna forma represente una gran cantidad de fuentes posibles, pero cuyo tamaño sea finito.
- Como desconocemos la fuente $\mathbf{p}^n(\mathbf{x})$ que representa la fuente de información verdadera, podemos codificar los símbolos utilizando cada $\mathbf{q}^n(\mathbf{x}) \in \mathcal{G}$.
- Para no confundir las palabras de código, deberemos incluir r bits en el preámbulo de cada palabra de código, de modo de identificar cada $\mathbf{q}^n(\mathbf{x}|\cdot)$, donde

$$r = \lceil \log_2 |\mathcal{G}| \rceil. \tag{15}$$

Algoritmo de Codificación Universal (1)

- $\mathcal{P}^{J^n}$ denota la familia de probabilidades conjuntas asociada a codificar la fuente original en bloques de largo n.
- Al fijar Θ define un subconjunto de $\mathcal{P}^{J^n}$.

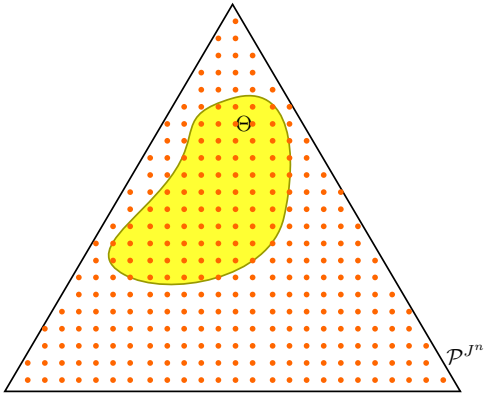


Algoritmo de Codificación Universal (2)

Paso 1. Crear una malla equiespaciada $\mathcal{G} \subset \mathcal{P}^{J^n}$ con 2^r puntos.

Paso 2. Utilizar r bits para representar cada punto de la grilla, los que están asociados a una única distribución de probabilidad, y por lo tanto, a una fuente.

Paso 3. Construir un código de largo variable para cada fuente en $\mathcal{G} \cap \Theta$.



Algoritmo de Codificación Universal (3)

- Luego, cada palabra de código tiene dos partes:
  - preámbulo: r bits que indican la fuente utilizada
  - código: $\bar{l}(\boldsymbol{\theta})$ bits (en promedio) asociados a los datos utilizando este código en particular.
- El costo que pagamos por no conocer la fuente es el *overhead* de r bits que debemos agregar como preámbulo a cada palabra de código.
- A medida que el largo de bloque n aumenta, el costo del preámbulo pierde importancia y uno puede realizar codificación óptima (en sentido asintótico, a medida de $n \to \infty$.

Algoritmo de Codificación Universal (4)

- El conocimiento *a priori* que uno tenga sobre la fuente, representado por Θ, puede emplearse para reducir apropiadamente el costo de la codificación universal.
- Se necesitarían $r = \lceil \log_2 |\mathcal{G} \cap \Theta| \rceil$ bits para representar cada fuente prototipo (codebook).
- El resto del procedimiento es el mismo.

Resumen

Hemos revisado:

- Teorema de Codificación de Fuente
- Algoritmo de Huffman para fuentes de información sin memoria.
- Optimalidad de la codificación de Huffman.
- Principios de codificación universal.

Lecturas

- Salehi & Proakis, *Communication System Engineering*, Sección 6.2-6.3.
- J. Rissanen, "Universal coding, information, prediction, and estimation," *IEEE Trans. on Information Theory*, Vol. 30, No. 4, pp. 629-636, July 1984.