Compresión semistática orientada a palabras
Guión

Motivación

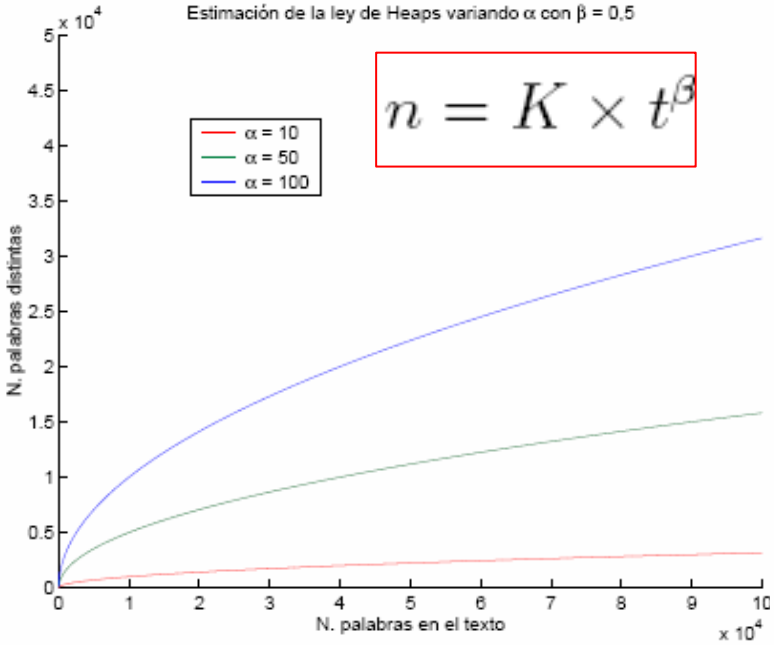
- Lenguaje Natural: leyes.
- Indexación orientada a palabras

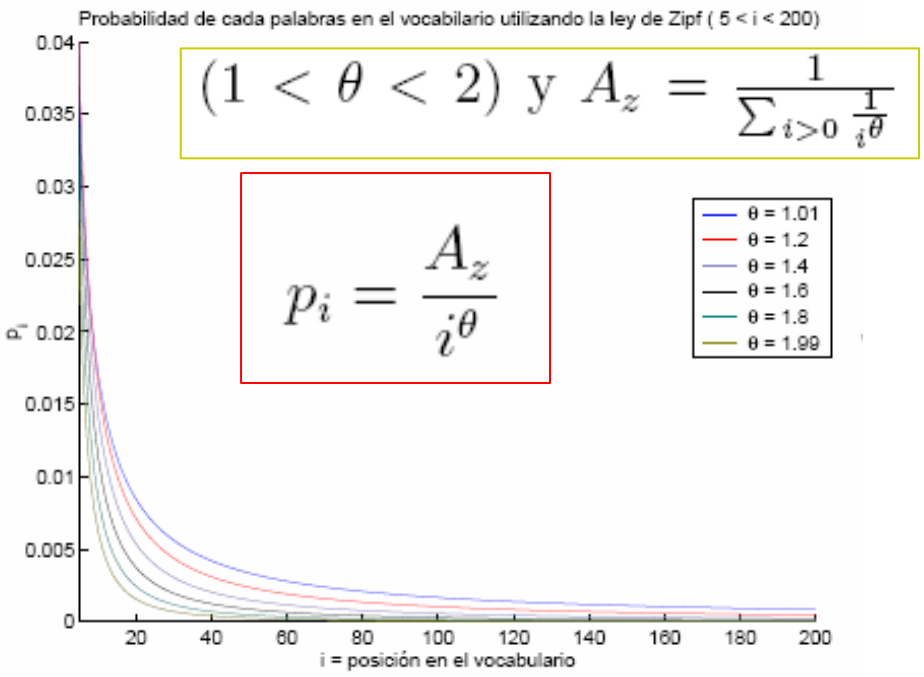
Compresión Huffman semistática

Compresores semiestáticos densos

Orientación a palabras ¿por qué?

- Lenguaje natural: Leyes de Zipf y de Heaps





Heaps:: el tamaño tiene poca importancia si tenemos textos grandes

Zipf:: La distribución de frecuencias de las palabras de un texto es muy sesgada

Orientación a palabras (refs)

- Refs básicas.

A. Moffat. Word-based text compression. *Software - Practice and Experience*, 19(2):185–198, 1989.

A. Moffat and J. Katajainen. In-place calculation of minimum-redundancy codes. In S.G. Akl, F. Dehne, and J.-R. Sack, editors, *Proc. Workshop on Algorithms and Data Structures (WADS'95)*, LNCS 955, pages 393–402, 1995.

Nivio Ziviani, Edleno Silva de Moura, Gonzalo Navarro, and Ricardo Baeza-Yates. Compression: A key for next-generation text retrieval systems. *IEEE Computer*, 33(11):37–44, 2000.

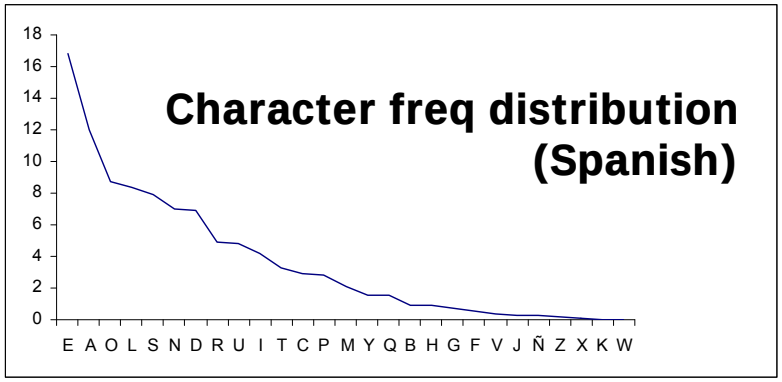
Compresión semistática orientada a palabras
Guión

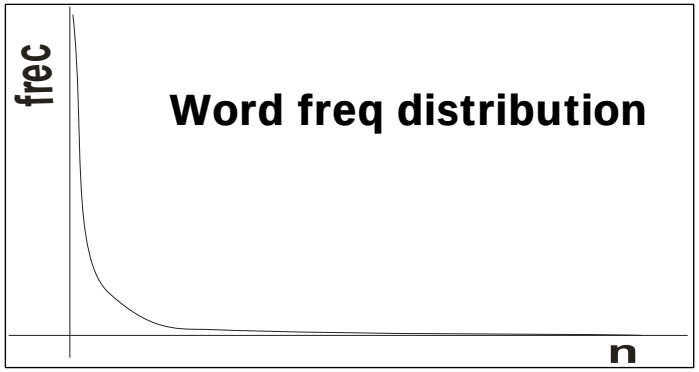
- Motivación
- Compresión Huffman semistática
  - Codificación Orientada a palabras
  - Huffman orientado a palabras: Plain y Tagged Huffman

- Compresores semiestáticos densos

Huffman orientado a palabras

- *Uso de orientación a palabras:*
  - *Bentley, Sleator, Tarjan, and Wei* (CACM-1986) ??
  - **Moffat** propuso usar palabras en vez de caracteres (+huffman)
- La distribución de frecuencias de las palabras es más sesgada





- Se consiguen ratios de compresión de hasta **25%** (en inglés)

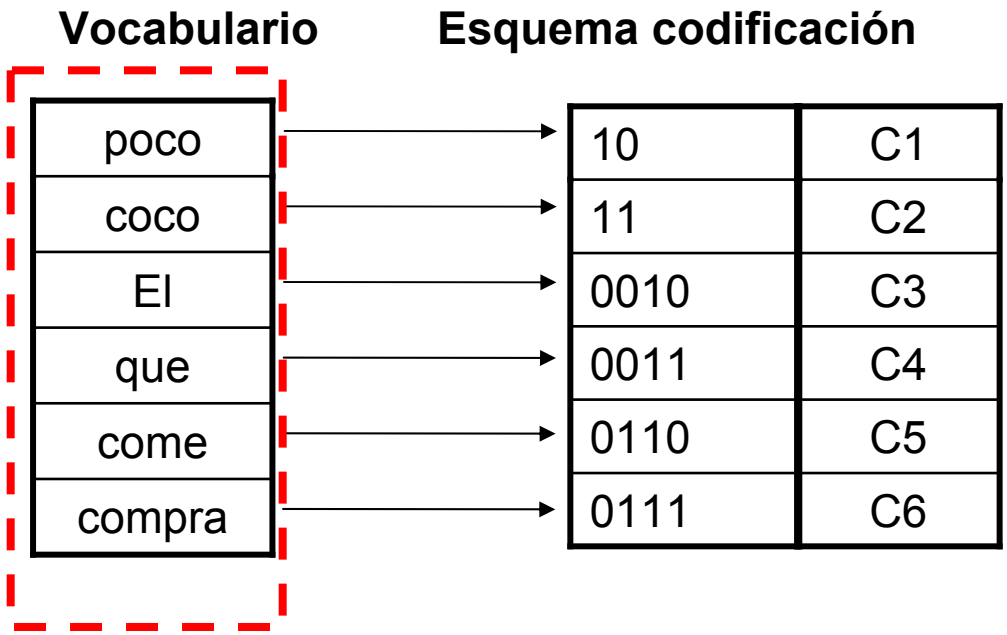
Así los elementos básicos para *compresión* y *Text Retrieval* son los mismos: ***las palabras***

Compresión + indexación

- Ejemplo:

Texto original: El que poco coco come poco coco compra

Índice invertido orientado a palabras:



Texto comprimido: 0010 0011 10 11 0110 10 11 0111

1 2 3 4 5 6 7 8 9 10 11 12

Compresión semistática orientada a palabras
Guión

- Motivación
- Compresión Huffman semistática
  - Codificación Orientada a palabras
  - Huffman orientado a palabras: Plain y Tagged Huffman

- Compresores semiestáticos densos

Plain Huffman y Tagged Huffman

- **1998: Moura, Navarro, Ziviani y Baeza:**
  - **2 nuevas técnicas**: Plain Huffman y Tagged Huffman

- Elementos comunes:
  - Basados en Huffman
  - Orientado a palabras
  - Usan bytes (no bits) (compresión ±30% pero más velocidad)

- Plain Huffman = Huffman sobre bytes (árbol 256-ario)
- Tagged Huffman **marca** el inicio de cada código

El primer bit es:
- "1" → para el 1er bit del 1er byte
- "0" → para el 1er bit de los bytes restantes

1xxxxxxx 0xxxxxxx 0xxxxxxx

Plain Huffman y Tagged Huffman Codificación

- Construcción Huffman b-ario:
  - Plain huffman aridad: 2^b=256, Tagged Huffman 2^b=128
  - Codificación huffman normal
    - Bottom-Up:
    - 1ª iteración: R=número de nodos último nivel
    - Restantes iteraciones: eligiendo 2^b nodos menos frecuentes

$$R = \begin{cases} n & \text{if } n < 2^b \ (only\ 1\ level\ in\ the\ tree) \\ 1 + ((n - 2^b)\ mod\ (2^b - 1)) & \text{if } (n - 2^b)\ mod\ (2^b - 1) > 0 \\ 2^b & \text{if } (n - 2^b)\ mod\ (2^b - 1) = 0 \end{cases}$$

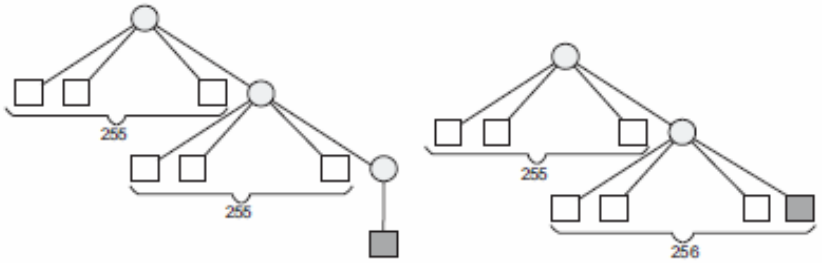


Figure 4.3: Shapes of non-optimal (a) and optimal (b) Huffman trees.

Plain Huffman y Tagged Huffman Ejemplos.

- **Asúmase (b=3, bytes "especiales" de sólo 3 bits)**
- **Obténgase la codificación Plain Huffman y Tagged Huffman para los vocabularios en los 2 escenarios siguientes:**

Word	Probab.
A	1/17
B	1/17
C	1/17
D	1/17
E	1/17
F	1/17
G	1/17
H	1/17
I	1/17
J	1/17
K	1/17
L	1/17
M	1/17
N	1/17
O	1/17
P	1/17
Q	1/17

Distribución uniforme: p_i=1/17

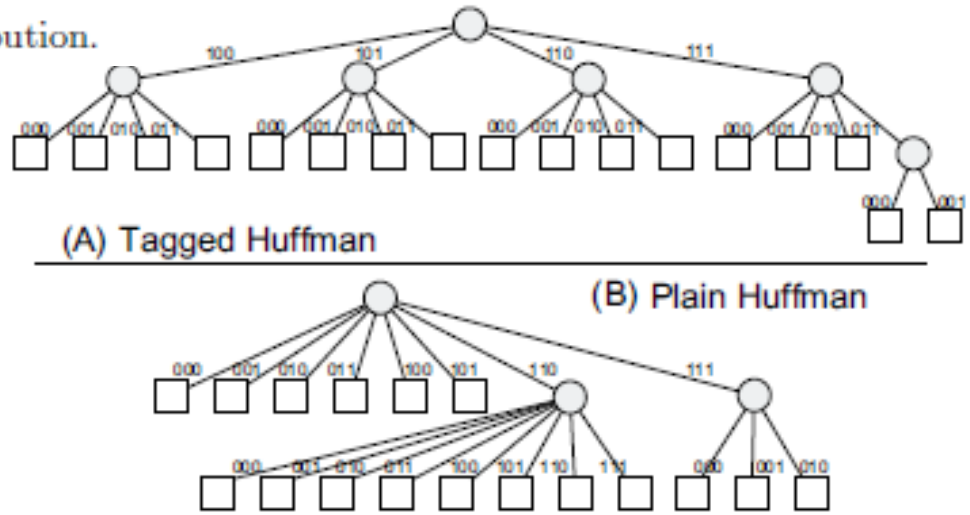
Word	Probab.
A	1/2
B	1/4
C	1/8
D	1/16
E	1/32
F	1/64
G	1/128
H	1/256
I	1/512
J	1/1024
K	1/2048
L	1/4096
M	1/8192
N	1/16384
O	1/32768
P	1/65536
Q	1/65536

Distribución exponencial: $p_i = 2^{-i}$

Plain Huffman y Tagged Huffman Ejemplo 1

Word	Probab.	Plain Huffman	Tagged Huffman
A	1/17	000	100 000
B	1/17	001	100 001
C	1/17	010	100 010
D	1/17	011	100 011
E	1/17	100	101 000
F	1/17	101	101 001
G	1/17	110 000	101 010
H	1/17	110 001	101 011
I	1/17	110 010	110 000
J	1/17	110 011	110 001
K	1/17	110 100	110 010
L	1/17	110 101	110 011
M	1/17	110 110	111 000
N	1/17	110 111	111 001
O	1/17	111 000	111 010
P	1/17	111 001	111 011 000
Q	1/17	111 010	111 011 001

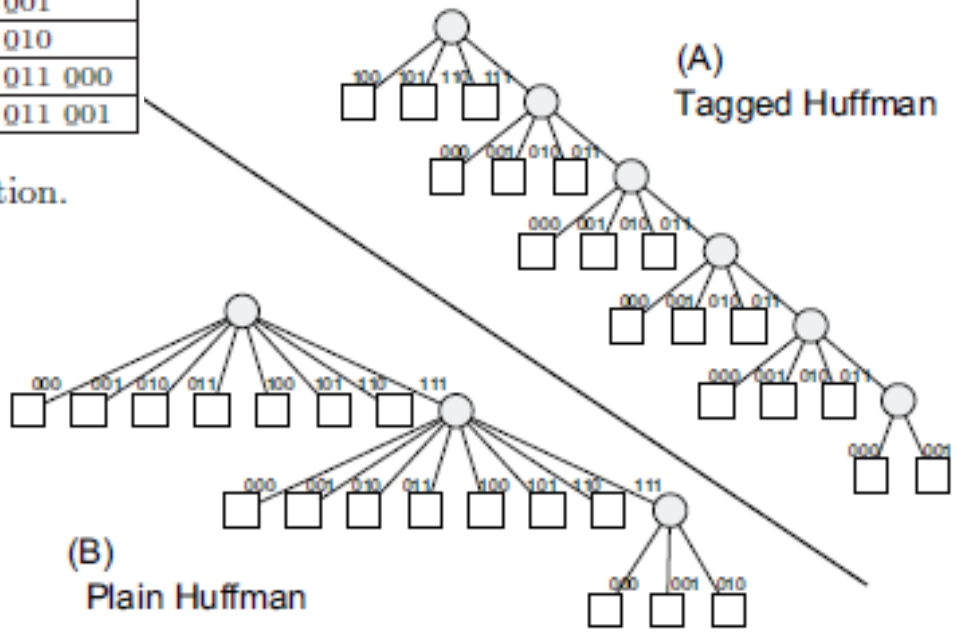
Table 4.1: Codes for a uniform distribution.



Plain Huffman y Tagged Huffman Ejemplo 2

Word	Probab.	Plain Huffman	Tagged Huffman
A	1/2	000	100
B	1/4	001	101
C	1/8	010	110
D	1/16	011	111 000
E	1/32	100	111 001
F	1/64	101	111 010
G	1/128	110	111 011 000
H	1/256	111 000	111 011 001
I	1/512	111 001	111 011 010
J	1/1024	111 010	111 011 011 000
K	1/2048	111 011	111 011 011 001
L	1/4096	111 100	111 011 011 010
M	1/8192	111 101	111 011 011 011 000
N	1/16384	111 110	111 011 011 011 001
O	1/32768	111 111 000	111 011 011 011 010
P	1/65536	111 111 001	111 011 011 011 011 000
Q	1/65536	111 111 010	111 011 011 011 011 001

Table 4.2: Codes for an exponential distribution.



Plain Huffman & Tagged Huffman
Búsquedas sobre texto comprimido

- Ejemplo (b=2): to be lucky or not

PLAIN HUFFMAN

be	00
or	01
not	10
lucky	11 00
to	11 11

TAGGED HUFFMAN

be	10
or	11 00
not	11 01 00
lucky	11 01 01 00
to	11 01 01 01

Busquemos “lucky”

to be lucky or not

1111 00 1100 01 10

Falso emparejamiento

to be lucky or not

11010101 10 11010100 1100 110100

Imposible emparejamientos falsos

- TH: Búsquedas mejoradas
  - → Búsqueda directa (comprimir el patrón y buscarlo)
  - → Empezar la búsqueda en cualquier lugar (y la descompresión)
  - → Búsqueda tipo Boyer-Moore es posible (saltando bytes)

Plain Huffman y Tagged Huffman

- Plain Huffman. (árbol de aridad 2^b=256)
  - Ratio +- 30-32%
  - Búsqueda simulando descompresión
    - (frases: *Shift-Or* ←→autómata)

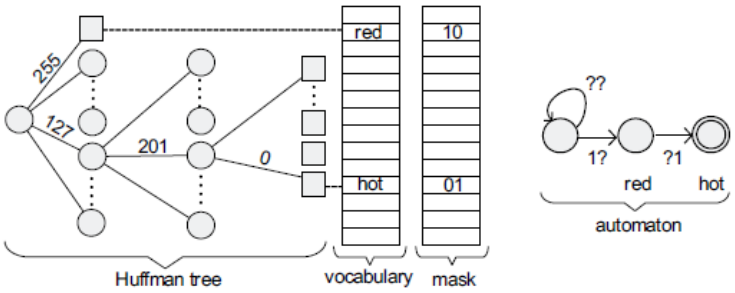


Figure 4.7: Searching Plain Huffman compressed text for pattern "red hot".

- Tagged Huffman. (árbol de aridad 2^{b-1} =128)
  - Pérdida en el ratio de compresión (3.5 puntos)
    - Ratio +- 33.5-35%
  - El bit de marca indica el inicio de los códigos:
    - Búsquedas directas mejoradas (al ser posible usar Boyer-Moore)
    - Descompresión aleatoria
      - **Sincronismo**: Es posible **(1)** ir a cualquier *offset* del texto comprimido, **(2)** buscar el principio de un código allí, y **(3)** comenzar la descompresión desde esa posición.

Compresión semistática orientada a palabras
Guión

- Motivación
- Compresión Huffman semiestática
- Compresores semiestáticos densos

- End Tagged Dense Code
- (s,c)- Dense Code
- Resultados Teóricos
- Resultados Empíricos

Compresores semistáticos densos
End-Tagged Dense Code

- Pequeño cambio: Una marca señala el final de un código

Primer bit es:

- "1" --> para el 1er bit del **último byte** → **1**xxxxxxx
- "0" --> para el 1er bit del resto de bytes → **0**xxxxxxx

Código libre de prefijo independ. de restantes 7 bits del byte

Ya no se necesita usar Huffman
Es posible usar ***TODAS las combinaciones*** de bits: ***Código Denso***

- Tiene bit de Flag → igual que Tagged Huffman en búsquedas

Compresores semistáticos densos
End-Tagged Dense Code

- Pequeño cambio: Una marca señala el final de un código

Primer bit es:
- "1" --> para el 1er bit del **último byte** → 1xxxxxxx
- "0" --> para el 1er bit del resto de bytes → 0xxxxxxx

Código libre de prefijo independ. de restantes 7 bits del byte

Códigos de 1 byte	1xxxxxxx
Códigos de 2 bytes	0xxxxxxx 1xxxxxxx
Códigos de 3 bytes	0xxxxxxx 0xxxxxxx 1xxxxxxx

Compresores semistáticos densos
End-Tagged Dense Code

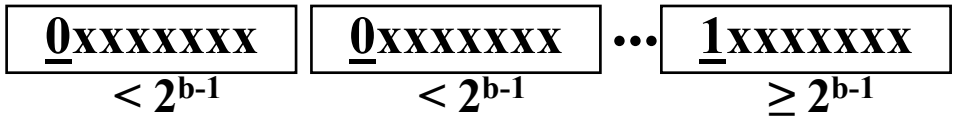
- Esquema de codificación

10000000 **1**0000001 ….. **1**1111111	*128* palabras más frecuentes (128 = **2^7** códigos de 1 byte)
00000000:**1**0000000 ….. **0**1111111:**1**1111111	***128^2*** palabras de **128+1** a **$128+128^2$** (128^2 **= 2^{14}** códigos de 2 bytes)
00000000:**0**0000000:**1**0000000 …… **0**1111111:**0**1111111:**1**1111111	**128^3** Las palabras **128+ 128^2+1** a **128 +128^2 +128^3** usan **tres bytes** (128^3 = **2^{21}** códigos)
…..	

- Los códigos dependen de la ***posición*** de la palabra en el ránking ***no*** de su ***frecuencia***

Compresores semistáticos densos
End-Tagged Dense Code

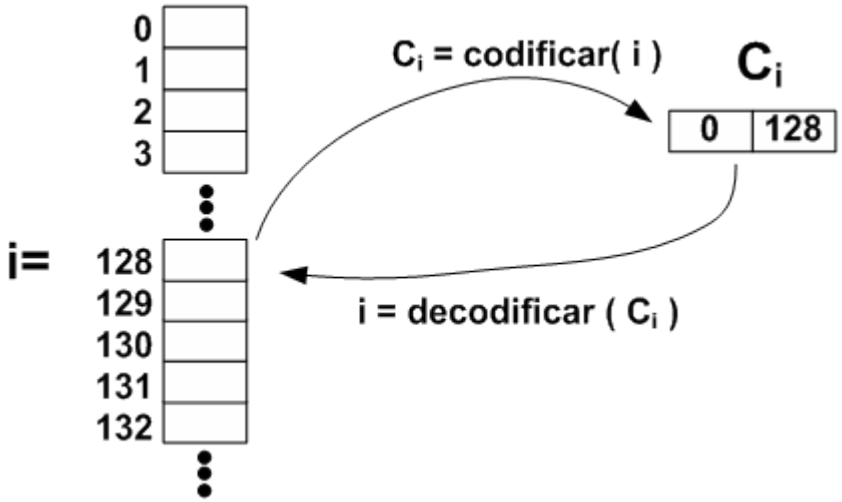
- Procedimiento de codificación **secuencial**
  - Ordenación de palabras por frecuencia
  - Asignación de códigos



- Procedimiento de codificación **directa** ("al vuelo")

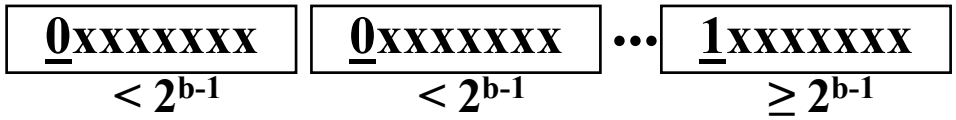
$C_i \leftarrow$ **codificar(i)**

i $\leftarrow$ **decodificar(C_i)**



Compresores semistáticos densos
End-Tagged Dense Code

- Procedimiento de codificación **secuencial**
  - Ordenación de palabras por frecuencia
  - Asignación de códigos




```
SequentialEncode (codeBook, n)
(1)  //input n: number of codewords that will be generated
(2)  firstKBytes ← 0;    //rank of the first word encoded with k bytes
(3)  numKBytes ← 128;    //number of k-byte codewords
(4)  p ← 0;    //current codeword being generated
(5)  k ← 1;    //size of the current codeword
(6)  while p < n    //n codewords are generated
(7)       paux ← p − firstKBytes;    //relative position inside k-byte codewords
(8)       while (p < n) and (paux < numKBytes) //k-byte codewords are computed
(9)            codebook[p].code[k − 1] ← (paux mod 128) + 128; //leftmost byte
(10)           paux ← paux div 128;
(11)           for i ← k − 2 downto 0
(12)                codebook[p].code[i] ← paux mod 128;
(13)                paux ← paux div 128;
(14)           p ← p + 1;
(15)           paux ← p − firstKBytes;
(16)      k ← k + 1;
(17)      firstKBytes ← firstKBytes + numKBytes;
(18)      numKBytes ← numKBytes × 128;
```

Compresores semistáticos densos
End-Tagged Dense Code

- Procedimiento de codificación **directa** (“al vuelo”)

C_i ← codificar(i)
i ← decodificar(C_i)

```
Decode (base, x)
(1)  //input  x: the codeword to be decoded
(2)  //output i: the position of the decoded word
(3)  i ← 0;
(4)  k ← 1; // byte of the codeword
(5)  while x[k] < 128
(6)        i ← i × 128 + x[k];
(7)        k ← k + 1;
(8)  i ← i × 128 + x[k] − 128;
(9)  i ← i + base[k];
(10) return i;
```

O(|x|) = O(log i)

```
Encode (i)
(1)  //input i: rank of the word being encoded 0 ≤ i ≤ n − 1
(2)  output ((i mod 128) + 128); //rightmost byte
(3)  i ← i div 128;
(4)  while i > 0 // remainder bytes
(5)        i ← i − 1;
(6)        output (i mod 128);
(7)        i ← i div 128;
```

O(log i)

base[1]=0, base[2]=128, base[3]=128 + 128^2, base[4]= 128 + 128^2 + 128^3...

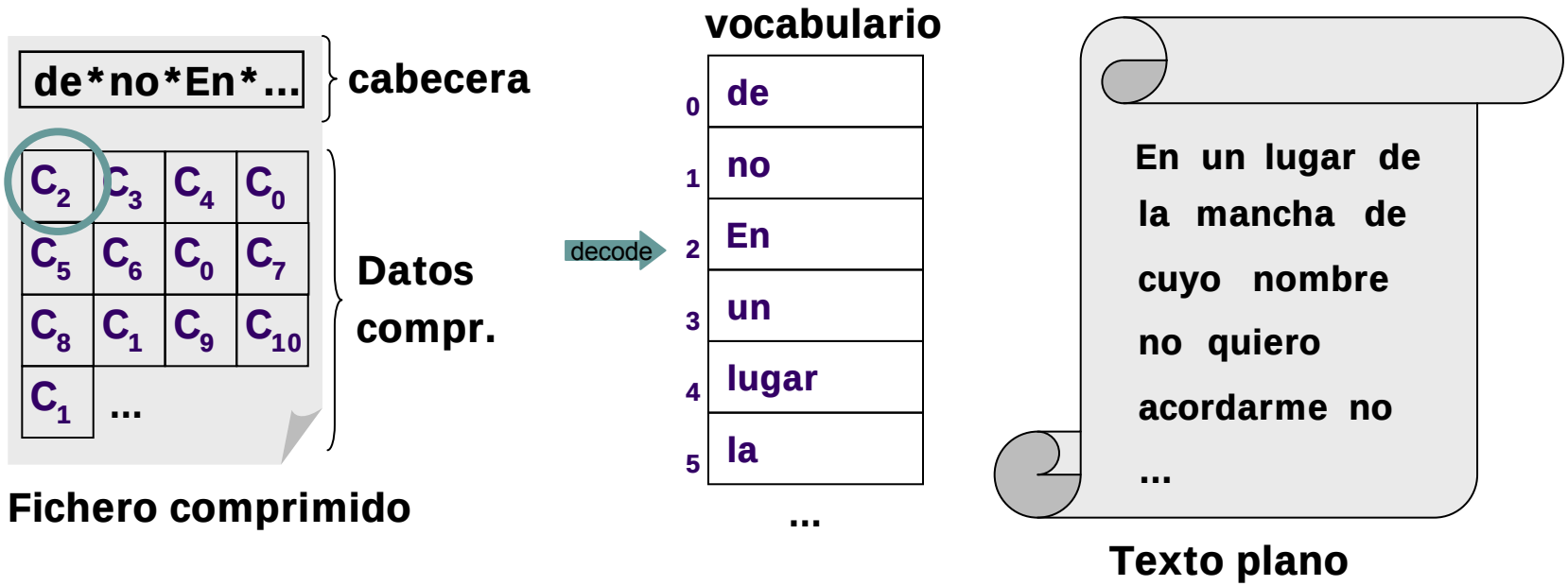
- Ej. i=decodifica(x)

$$x = x_1x_2x_3x_4$$
$$i = (((x_1 \times 128) + x_2) \times 128) + x_3) \times 128) + (x_4 - 128) + base[4]$$

operation		bitwise operation
$x \ mod \ 128$	$\longrightarrow$	$x \ and_b \ 0111111$
$x \ + \ 128$	$\longrightarrow$	$x \ or_b \ 1000000$
$x \ div \ 128$	$\longrightarrow$	$x \ >> \ 7$
$x \ \times \ 128$	$\longrightarrow$	$x \ << \ 7$

Compresores semistáticos densos
End-Tagged Dense Code

- Descompresión: dos pasos
  - Cargar el vocabulario ordenado
  - **i ← decodificar(C_i)** :: O(bytes T.Comp)



Compresores semistáticos densos
End-Tagged Dense Code: búsquedas TC

Búsquedas directas:

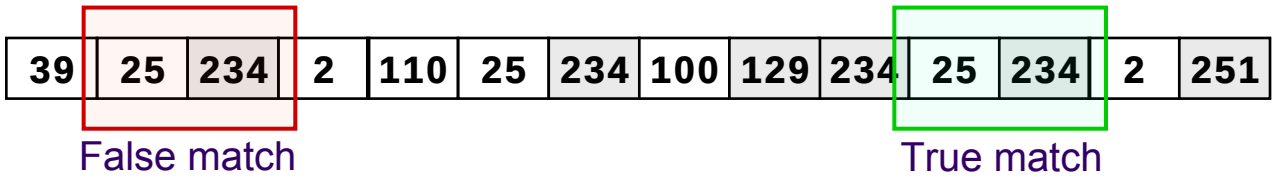
1) Obtener el código asociado al patrón **P → Cp**

2) Buscar el código **Cp** dentro del texto comprimido usando un algoritmo de tipo Boyer-Moore (skipping bytes)

3) **Tras un emparejamiento** chequear si es una ocurrencia real del patrón

   - Es una ocurrencia o el sufijo de un código más largo?
   - Byte previo **≥ 128** ?

— Ej. Búsqueda de: “Pedrito”→ C(“Pedrito”) = | 25 | 234 |



Compresores semistáticos densos
End-Tagged Dense Code: búsquedas TC

- Algoritmo Horspool modificado para ETDC.

Alg: Search process using Horspool's algorithm. It receives a codeword x and its length k, parameter c, the compressed text T to search, and its length z. It outputs all the text positions that start a true occurrence of x in T

Search $(x,\ k,\ c,\ T,\ z)$
(1) **for** $b \leftarrow 0$ **to** 255 **do** $d[b] \leftarrow k$
(2) **for** $j \leftarrow 1$ **to** $k-1$ **do** $d[x[j]] \leftarrow k-j$
(3) $i \leftarrow 0$
(4) **while** $i \leq z-k$
(5) $\quad j \leftarrow k-1$
(6) $\quad$ **while** $j \geq 0$ and $T[i+j] = x[j]$ **do** $j \leftarrow j-1$
(7) $\quad$ **if** $j < 0$ and $(i = 0$ or $T[i-1] \geq c)$ **output** i
(8) $\quad i \leftarrow i + d[T[i+k-1]]$

En ETDC, c=128

Programa C:

TRUCO para evitar (i=0)

T	39	25	234	2	110	25	234	100	129	234	25	234	2	251
	0	1	2	3										z-1

P	25	234
	0	k-1

Compresores semistáticos densos
End-Tagged Dense Code

- Es un código **denso**. Pueden utilizarse todos los códigos disponibles.
  - Comprime mejor que TH (2-3 puntos).
  - Es superado por PH (≤1 punto).
- **Marca** → mismas capacidades de búsqueda de Tagged Huffman
  - Búsqueda directa,
  - Acceso aleatorio.
- Codificación y decodificación eficiente
  - Procedimientos secuencial y directo
- ***Fácil de programar***

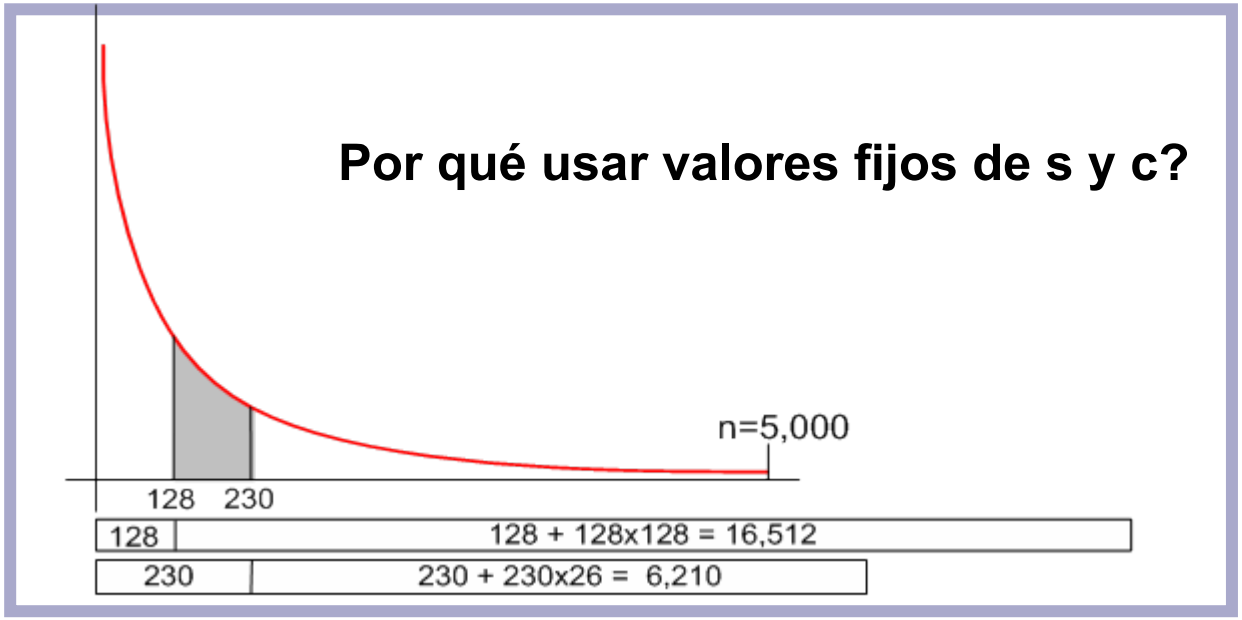
Compresión semistática orientada a palabras
Guión

- Motivación
- Compresión Huffman semiestática
- Compresores semiestáticos densos

  • End Tagged Dense Code
  • (s,c)- Dense Code
  • Resultados Teóricos
  • Resultados Empíricos

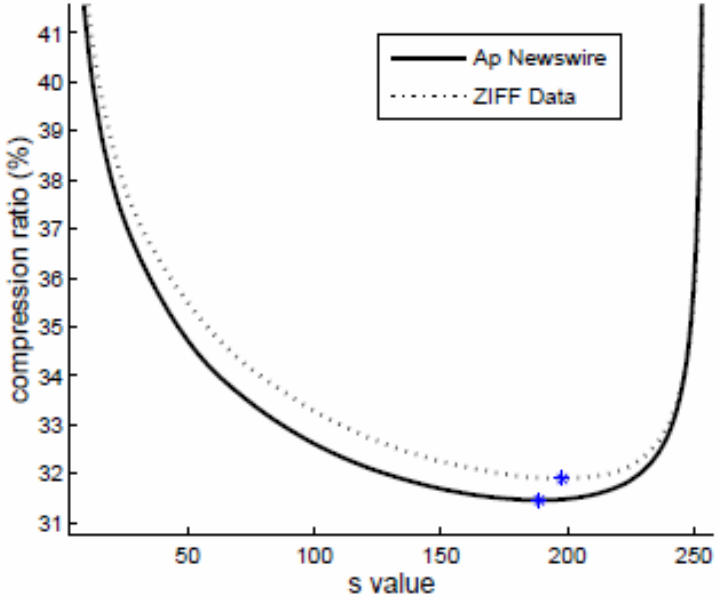
Compresores semistáticos densos
(s,c)-Dense Code

- End Tagged Dense Code
  - 128 valores disponibles [128, 255] para el último byte (***S***toppers)
  - 128 valores disponibles [0, 127] para los restantes bytes (***C***ontinuers)



- Adaptar (s,c) al vocabulario → s minimizando tamaño Texto Comp.
  - Número de palabras
  - Distribución de frecuencia de las palabras
- End-Tagged Dense Code es un (128,128)-Dense Code

Compresores semistáticos densos
(s,c)-Dense Code



s value	Ap Newswire Corpus		ZIFF Corpus	
	ratio(%)	size(bytes)	ratio(%)	size(bytes)
186	31.5927	79,207,309	32.0433	59,350,593
187	31.5917	79,204,745	32.0391	59,342,810
188	31.5910	79,202,982	32.0355	59,336,069
189	**31.5906**	**79,202,053**	32.0321	59,329,848
190	31.5907	79,202,351	32.0290	59,324,149
191	31.5912	79,203,574	32.0263	59,319,106
192	31.5921	79,205,733	32.0239	59,314,667
195	31.5974	79,219,144	32.0188	59,305,267
196	31.6002	79,226,218	32.0179	59,303,599
197	31.6036	79,234,671	32.0174	59,302,677
198	31.6074	79,244,243	**32.0174**	**59,302,609**
199	31.6117	79,254,929	32.0178	59,303,324
200	31.6166	79,267,125	32.0186	59,304,790
201	31.6220	79,280,683	32.0198	59,307,119

SequentialFindBestS $(b,\ acc)$
(1) //inputs: b value ($2^b = c + s$) and acc, the vector of accumulated frequencies
(2) //output: The best s and c values
(3) $sizeBestS \leftarrow \infty$;
(4) **for** $i = 1$ **to** $2^b - 1$
(5) $\quad sizeS \leftarrow computeSizeS(i, 2^b - i, acc)$;
(6) $\quad$ **if** $sizeS < sizeBestS$ **then**
(7) $\quad\quad bestS \leftarrow i$;
(8) $\quad\quad sizeBestS \leftarrow sizeS$;
(9) $bestC \leftarrow 2^b - bestS$;
(10) **return** $bestS,\ bestC$;

computeSizeS $(s,\ c,\ acc)$
(1) //inputs: s, c and acc, the vector of accumulated frequencies
(2) //output: the length of the compressed text using s and c
(3) $k \leftarrow 1$; $n \leftarrow$ *number of positions in vector* $'acc'$;
(4) $total \leftarrow acc[n]$;
(5) $Left \leftarrow min(s, n)$;
(6) **while** $Left < n$
(7) $\quad total \leftarrow total + (acc[n] - acc[Left]$* k ;
(8) $\quad Left \leftarrow Left + sc^k$;
(9) $\quad k \leftarrow k + 1$;
(10) **return** $total$;

Num occs	20	15	12	11		8	8		3	3	2		1	1	
Frec. acum	20	35	47	58		…	…						…	1000	

Compresores semistáticos densos
(s,c)-Dense Code

http://vios.dc.fi.udc.es/codes

- Esquema de codificación
  - Stoppers: último byte. ***s*** valores entre [0,s-1]
  - Continuers: otros bytes. ***c*** valores entre [s, 255]

0
…
s-1

s palabras más frecuentes

s	0
s+1	1
…	…
255	s-1

sc palabras de ***s+1*** a ***s+sc***

s	s	0
…	…	…
255	255	S-1

sc² palabras de ***s+sc+1*** a ***s+sc+sc²***

Compresores semistáticos densos
(s,c)-Dense Code

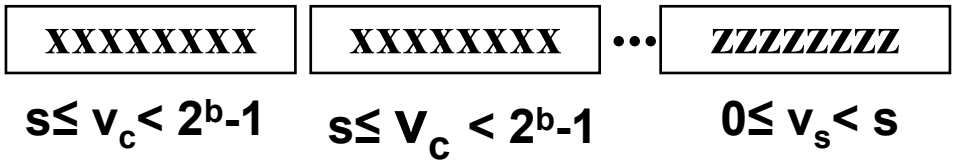
- Ejemplo (b=3)

Palabra	Freq	P.H.	(6,2)-DC	(5,3)-DC	(4,4)- DC	PH	(6,2)	(5,3)	ETDC
A	0,20	[000]	[000]	[000]	[000]	0,20	0,20	0,20	0,20
B	0,20	[001]	[001]	[001]	[001]	0,20	0,20	0,20	0,20
C	0,15	[010]	[010]	[010]	[010]	0,15	0,15	0,15	0,15
D	0,15	[011]	[011]	[011]	[011]	0,15	0,15	0,15	0,15
E	0,14	[100]	[100]	[100]	[100][000]	0,14	0,14	0,14	0,28
F	0,09	[101]	[101]	[101][000]	[100][001]	0,09	0,09	0,18	0,18
G	0,04	[110]	[110][000]	[101][001]	[100][010]	0,04	0,08	0,08	0,08
H	0,02	[111][000]	[110][001]	[101][010]	[100][011]	0,04	0,04	0,04	0,04
I	0,005	[111][001]	[110][010]	[101][011]	[101][000]	0,01	0,01	0,01	0,01
J	0,005	[111][010]	[110][011]	[101][100]	[101][001]	0,01	0,01	0,01	0,01
					Longitud media del código	1,03	1,07	1,16	1,30

- End-Tagged Dense Code es un $(2^{b-1},2^{b-1})$-DC

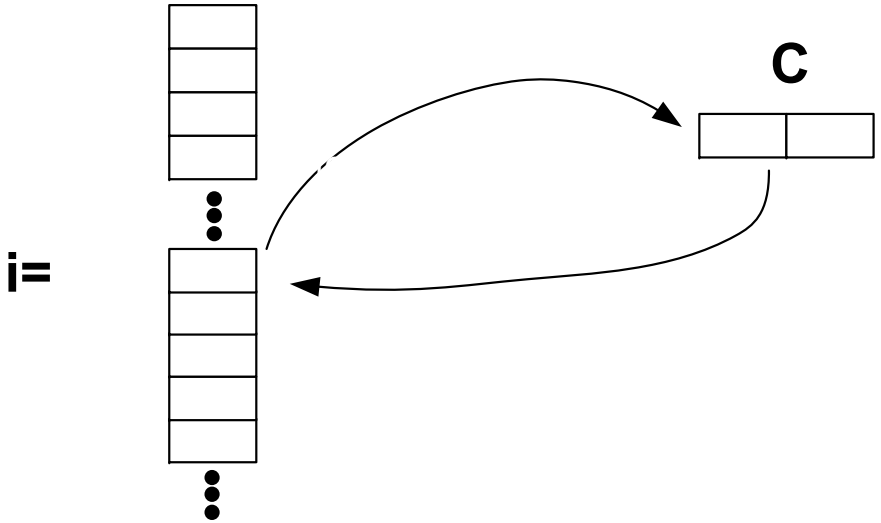
Compresores semistáticos densos
(s,c)-Dense Code

- Codificación Secuencial



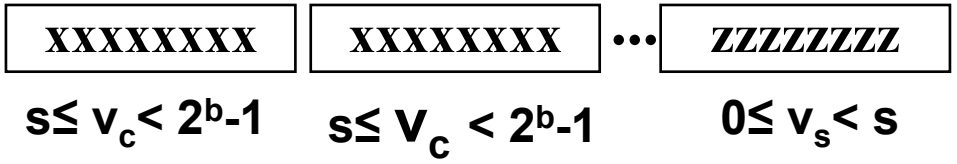
- Codificación directa

$C_i \leftarrow$ codifica(s, i)
$i \leftarrow$ decodifica(s, C_i)



Compresores semistáticos densos
(s,c)-Dense Code

- Codificación Secuencial



- Codificación directa

C_i ← codifica(s, i)
i ← decodifica(s, C_i)

```
Encode (i)
(1) //input: i, the rank of the word in the vocabulary
(2) //output: the codeword C_i from right to left
(3) output i mod s;
(4) x ← i div s;
(5) while x > 0
(6)       x ← x − 1;
(7)       output(x mod c) + s;
(8)       x ← x div c;
```

O(log i)

```
Decode (base, x)
(1) //input: x, the codeword to be decoded
(2) //output: i, the position of the decoded word
(3) i ← 0;
(4) k ← 1; // number of bytes of the codeword
(5) while x[k] ≥ s
(6)       i ← i × c + (x[k] − s);
(7)       k ← k + 1;
(8) i ← i × s + x[k];
(9) i ← i + base[k];
(10) return i;
```

O(|x|) = O(log i)

base[1]=0, base[2]=s, base[3]=s + sc, base[4]= s + sc+ sc^2 ...

Compresores semistáticos densos
(s,c)-Dense Code : búsquedas TC

- Algoritmo Horspool modificado para SCDC.

Alg: Search process using Horspool's algorithm. It receives a codeword x and its length k, parameter c, the compressed text T to search, and its length z. It outputs all the text positions that start a true occurrence of x in T

Search $(x,\ k,\ c,\ T,\ z)$
(1) **for** $b \leftarrow 0$ **to** 255 **do** $d[b] \leftarrow k$
(2) **for** $j \leftarrow 1$ **to** $k-1$ **do** $d[x[j]] \leftarrow k-j$
(3) $i \leftarrow 0$
(4) **while** $i \leq z-k$
(5) $\quad j \leftarrow k-1$
(6) $\quad$ **while** $j \geq 0$ and $T[i+j] = x[j]$ **do** $j \leftarrow j-1$
(7) $\quad$ **if** $j < 0$ and $(i = 0$ or $T[i-1] \geq c)$ **output** i
(8) $\quad i \leftarrow i + d[T[i+k-1]]$

En SCDC, $c=2^b-s = 256-s$

Compresores semistáticos densos
(s,c)-Dense Code

- Es un código denso
  - Comprime mejor que TH (3-4 puntos)
  - Comprime mejor que ETDC (0.5 puntos)
  - Es superado por PH (0.25 puntos)
  - RATIO: PH < SCDC << ETDC <<< TH
- Codificación y decodificación simple
- ¿Marca? → (byte valor < s)
  - Mismas capacidades de búsqueda que End-Tagged Dense Code y Tagged Huffman
- S óptimo entre 180 y 190

Guión de la exposición

- Motivación
- Compresión Huffman semiestática
- Compresores semiestáticos densos

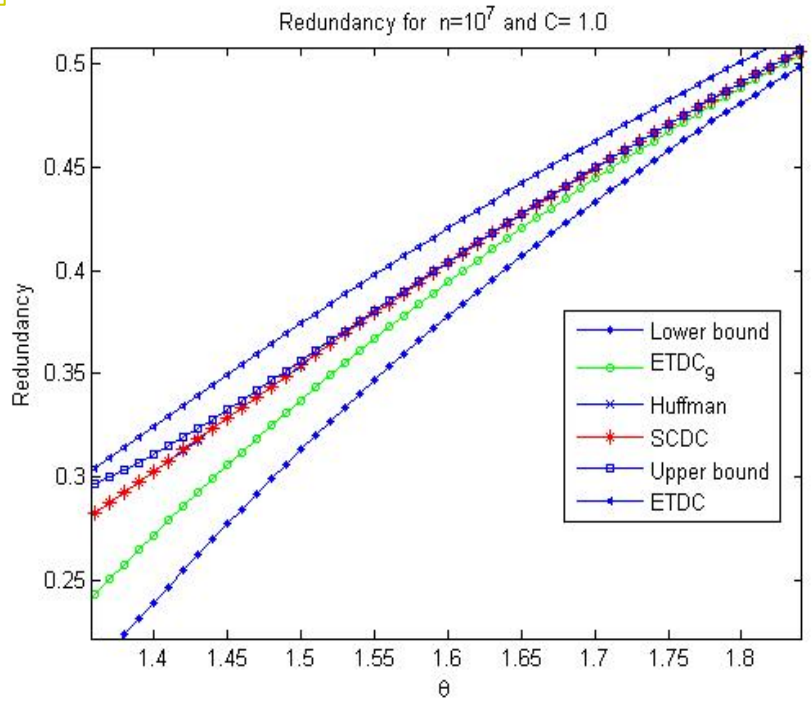
  - End Tagged Dense Code
  - (s,c)- Dense Code
  - Resultados Teóricos
  - Resultados Empíricos

Compresores semiestáticos densos

Acotación analítica de Huffman

- Es posible obtener nuevas cotas analíticas de la compresión que puede alcanzarse con Huffman usando (s,c)-DC

$$L_{d(D,D)} \leq L_{h(D)} \leq L_{d(s,D-s)}$$



Gonzalo Navarro and Nieves Brisaboa. **New Bounds on *D*-ary Optimal Codes**. *Information Processing Letters (IPL)* 96(5):178-184, 2005

Compresores semiestáticos densos

Acotación analítica de Huffman

- Nuevas cotas analíticas de Huffman usando (s,c)-DC

$$L_{d(D,D)} \leq L_{h(D)} \leq L_{d(s,D-s)}$$

Dada la entropía (*D* bits): $H_D = \sum_{0 \leq i < n} p_i \log_D \frac{1}{p_i}$, $H_D \leq L_{h(D)} \leq H_D + 1$

Siendo $c^{k-1}s$ el número de palabras codificables con *k* bytes,

$$w_k = \sum_{j=1}^{k} sc^{j-1} = s\frac{c^k - 1}{c - 1} \qquad w_0 = 0$$

indica el número de palabras codificables con *k* bytes

Por tanto, la probabilidad de los los símbolos codificados con hasta *k* bytes es:

$$f_k = \sum_{j=w_{k-1}+1}^{w_k} p_j$$

Obteniéndose:

$$L_{d(s,c)} = \sum_{k=1}^{K} k f_k = \sum_{k=1}^{K} k \sum_{j=w_{k-1}+1}^{w_k} p_j$$

$$= 1 + \sum_{k=1}^{K-1} k \sum_{j=w_k+1}^{w_{k+1}} p_j = 1 + \sum_{k=1}^{K-1} \sum_{j=w_k+1}^{w_K} p_j$$

$$K = \left\lfloor \log_c \left(1 + \frac{n(c-1)}{s}\right) - 1 \right\rfloor$$

Compresores semiestáticos densos

Acotación analítica de Huffman

Por la ley de Zipf: $p_i = A/i^\theta$, $n = \infty$, para ciertas constantes A y $\theta > 1$

donde: $$A = \frac{1}{\sum_{i \geq 1} 1/i^\theta} = \frac{1}{\zeta(\theta)}$$

Cota Superior

Partiendo de que $L_{h(D)} \leq L_{d(s,D-s)}$

$$L_{d(s,c)} \leq 1 + A \sum_{k \geq 1} \int_{w_k}^{\infty} 1/x^\theta dx = 1 + \frac{A(c-1)^{\theta-1}}{(\theta-1)s^{\theta-1}} \sum_{k \geq 1} \frac{1}{(c^k-1)^{\theta-1}}$$

$$= 1 + \frac{1}{(\theta-1)\zeta(\theta)s^{\theta-1}(1-c^{1-\theta})}$$

Sustituyendo *c*=*D*-*s* y minimizando, obtenemos una cota superior mínima cuando: $s = D - D^{1/\theta}$ y $c = D^{1/\theta}$

$$\min_s L_{d(s,D-s)} \leq 1 + \frac{1}{(\theta-1)\zeta(\theta)(D-D^{1/\theta})^{\theta-1}(1-D^{1/\theta-1})}$$

Compresores semiestáticos densos

Acotación analítica de Huffman

Cota Inferior

Análogamente:

$$L_{d(s,c)} \geq 1 + A \sum_{k \geq 1} \int_{w_k+1}^{\infty} 1/x^{\theta} dx = 1 + \frac{A(c-1)^{\theta-1}}{\theta - 1} \sum_{k \geq 1} \frac{1}{(sc^k - s + c - 1)^{\theta-1}}$$

Tomando $s = c = D$

$$L_{d(D,D)} \geq 1 + \frac{A(D-1)^{\theta-1}}{\theta - 1} \sum_{k \geq 1} \frac{1}{(D^{k+1} - 1)^{\theta-1}} = 1 + \frac{(D-1)^{\theta-1}}{(\theta-1)\zeta(\theta)D^{\theta-1}(D^{\theta-1} - 1)}$$

Puesto que : $L_{d(D,D)} \leq L_{h(D)}$ nuestra cota superior, viene dada por:

$$1 + \frac{(D-1)^{\theta-1}}{(\theta-1)\zeta(\theta)D^{\theta-1}(D^{\theta-1} - 1)} \leq L_{h(D)}$$

Compresión semistática orientada a palabras
Guión

- Motivación
- Compresión Huffman semiestática
- Compresores semiestáticos densos

  • End Tagged Dense Code
  • (s,c)- Dense Code
  • Resultados Teóricos
  • Resultados Empíricos

Compresores semistáticos densos

Resultados empíricos y Plataforma de prueba

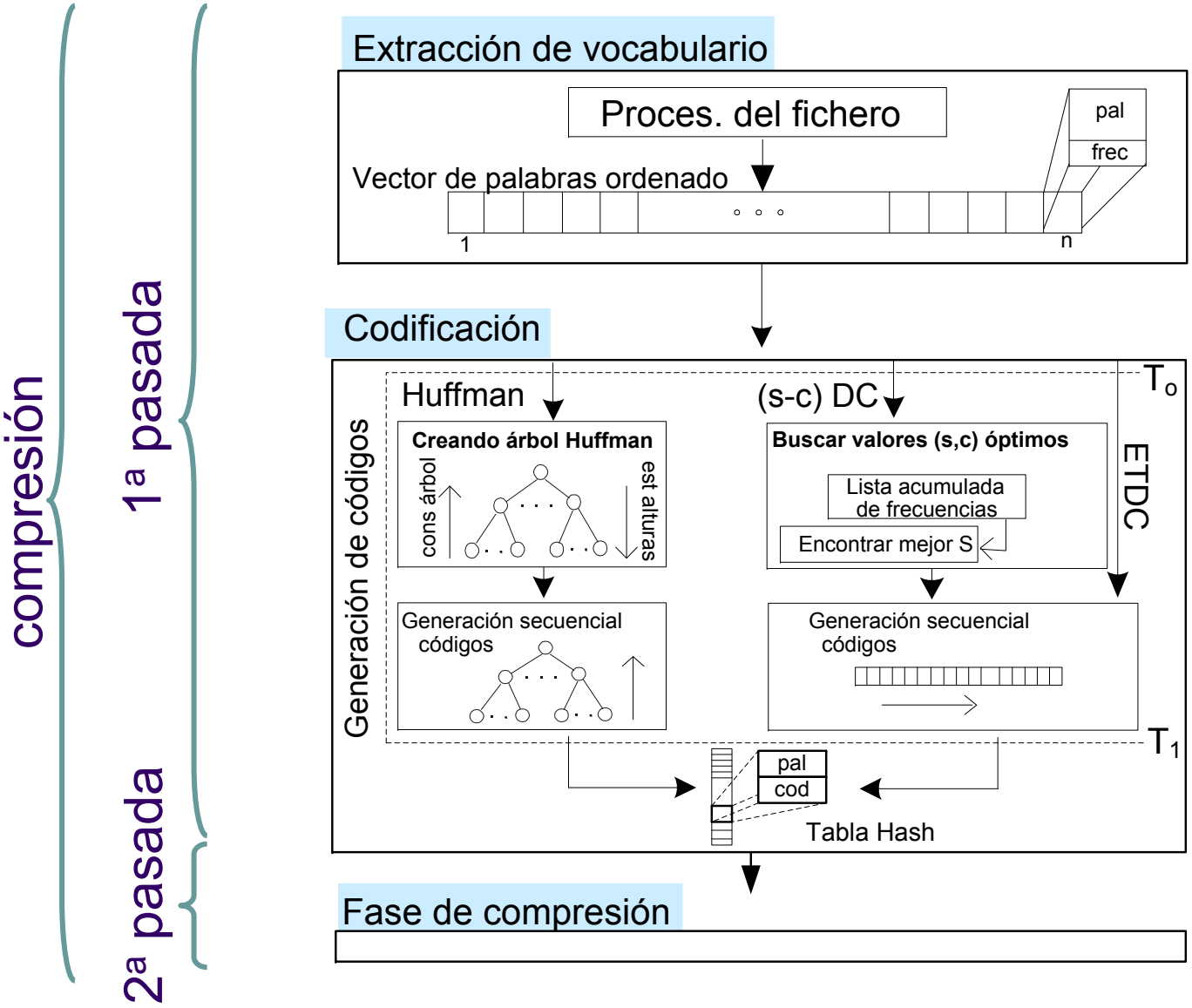
- Textos del **TREC-2** y **TREC-4**

CORPUS	Tamaño (bytes)	Nº palabras	Nº palabras diferentes
CALGARY	2,131,045	528,611	30,995
FT91	14,749,355	3,135,383	75,681
CR	51,085,545	10,230,907	117,713
FT92	175,449,235	36,803,204	284,892
ZIFF	185,220,215	40,866,492	237,622
FT93	197,586,294	42,063,804	291,427
FT94	203,783,923	43,335,126	295,018
AP	250,714,271	53,349,620	269,141
ALL FT	591,568,807	124,971,944	577,352
ALL	1,080,719,883	229,596,845	886,190

- Intel Pentium-III (x2) 800 Mhz con 768Mb RAM.
  - Debian GNU/Linux (kernel 2.2.19)
  - gcc 3.3.3 20040429 y optimización –O9
  - Time muestra **CPU user-time**

Compresores semistáticos densos

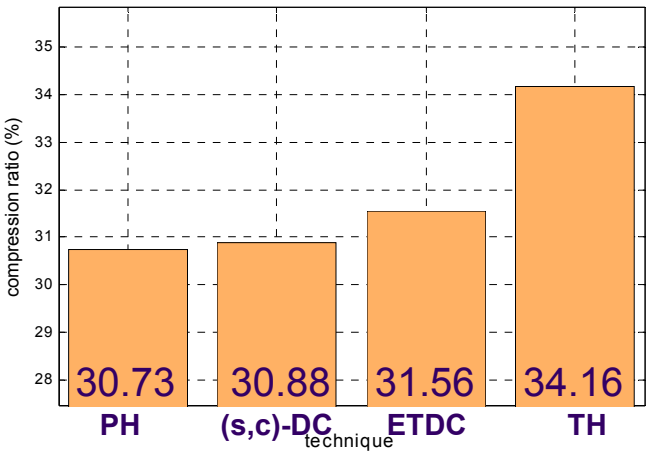
Tiempos de codificación y compresión



Compresores semistáticos densos

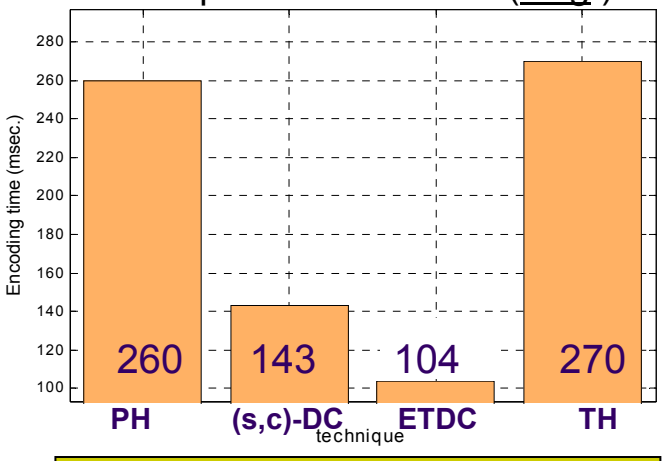
Resultados Empíricos

Ratio de compresión (%)



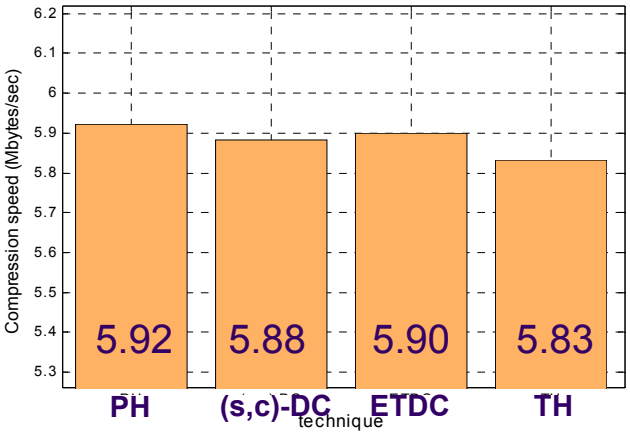
PH < (s,c)-DC < ETDC < TH
0.2 pp 0.8 pp 2.5 pp

Tiempo de codificación (msg.)



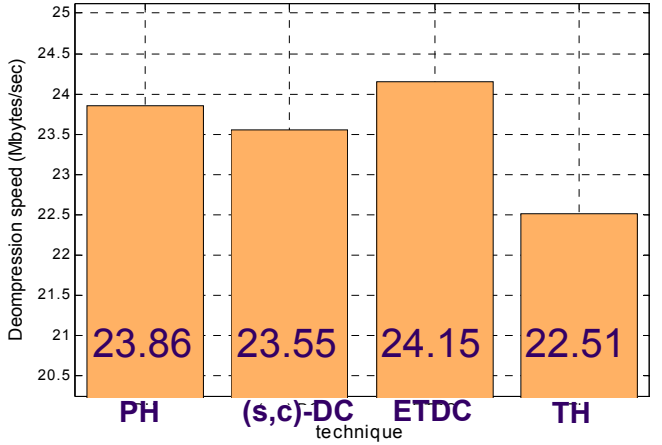
ETDC < (s,c)-DC < PH < TH
25% 45% 2%

Velocidad de compresión (Mb/sg.)



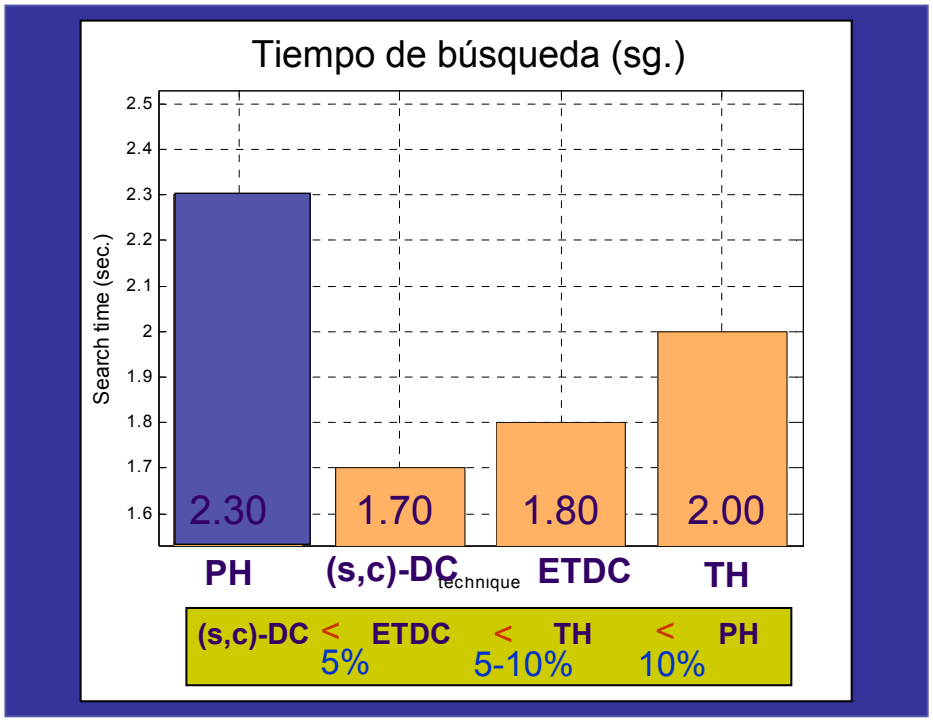
PH = ETDC > (s,c)-DC > TH

Velocidad de descompresión (Mb/sg.)



ETDC = PH > (s,c)-DC > TH
1,5% 4%

Compresores semistáticos densos
Búsquedas de patrones simples

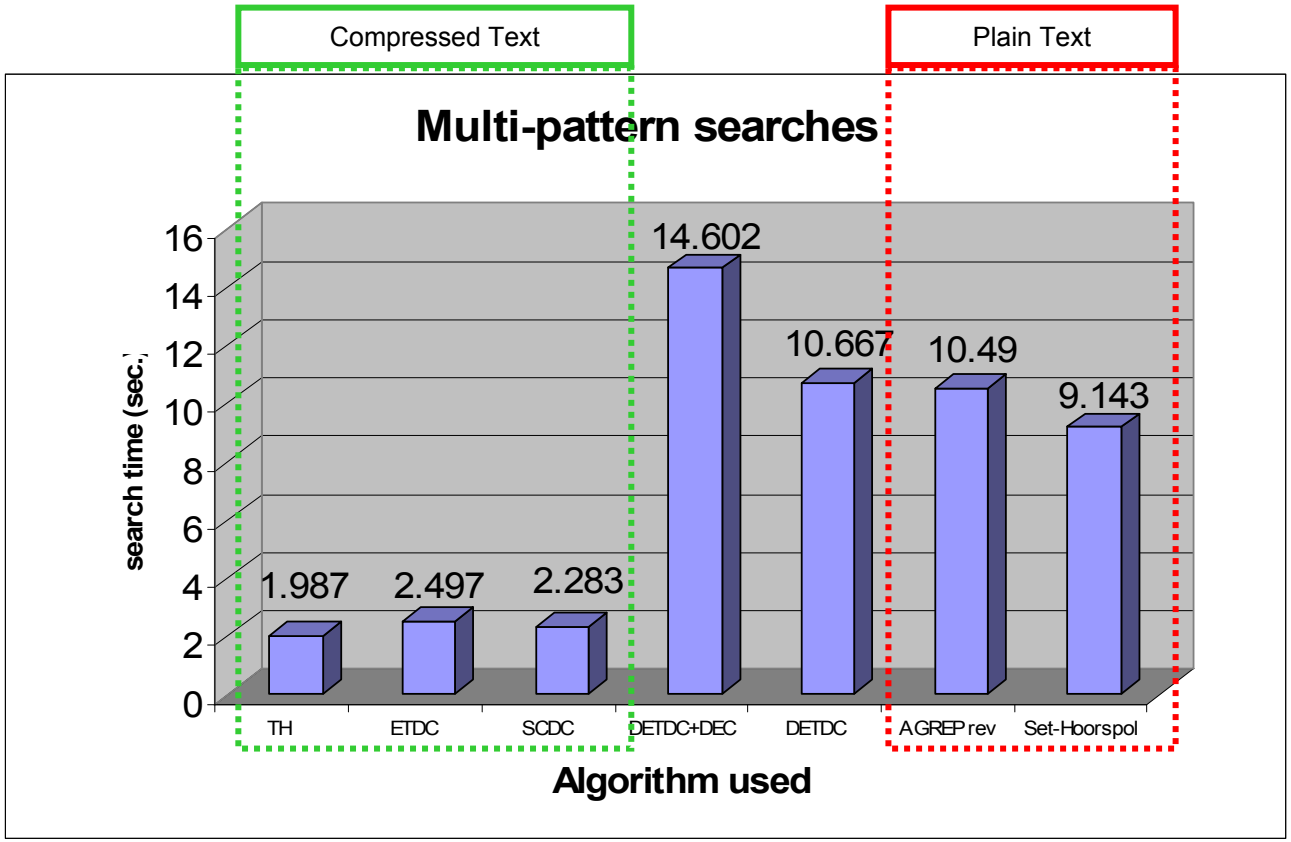


Buscando patrones:

- Formados por 1 única palabra
- Cuyos **<u>códigos tienen la misma longitud</u>**

Compresores semistáticos densos

Búsquedas multipatrón (100pats.)



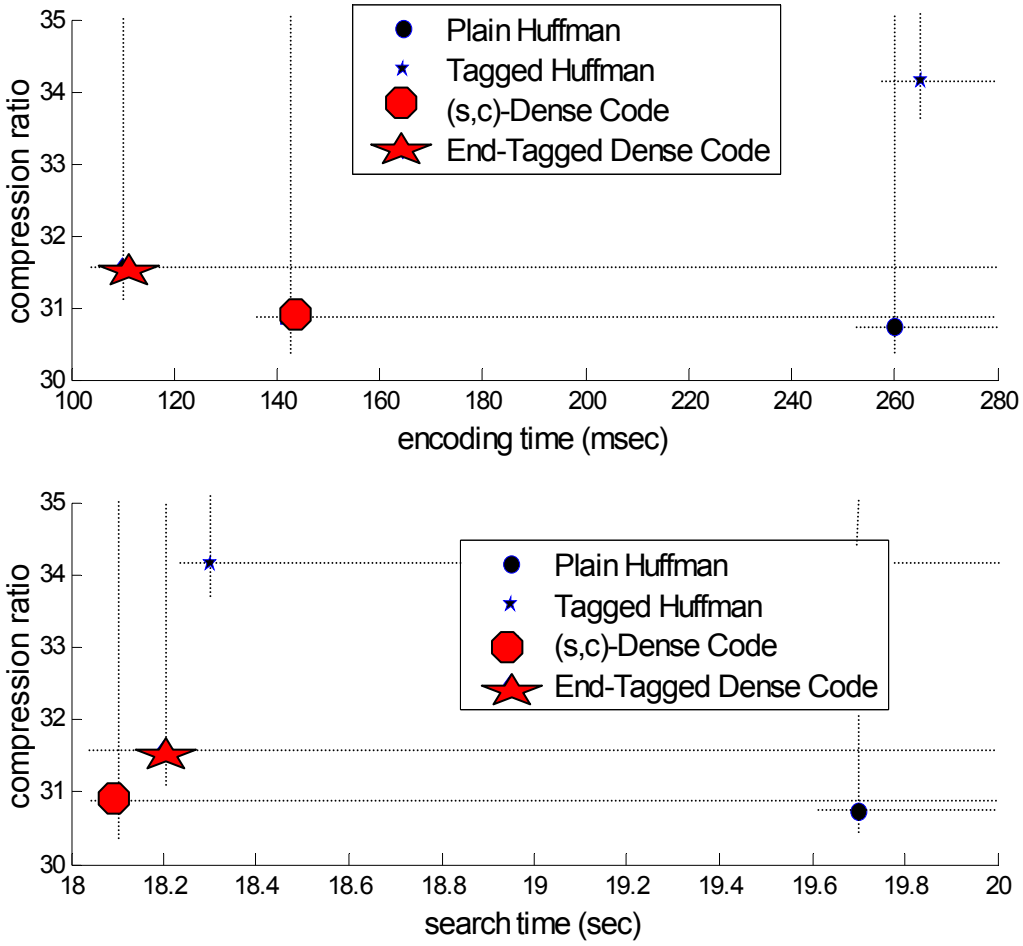
- Multipatrón

TH < SCDC < ETDC < Plain text

15-20% 5% 400%

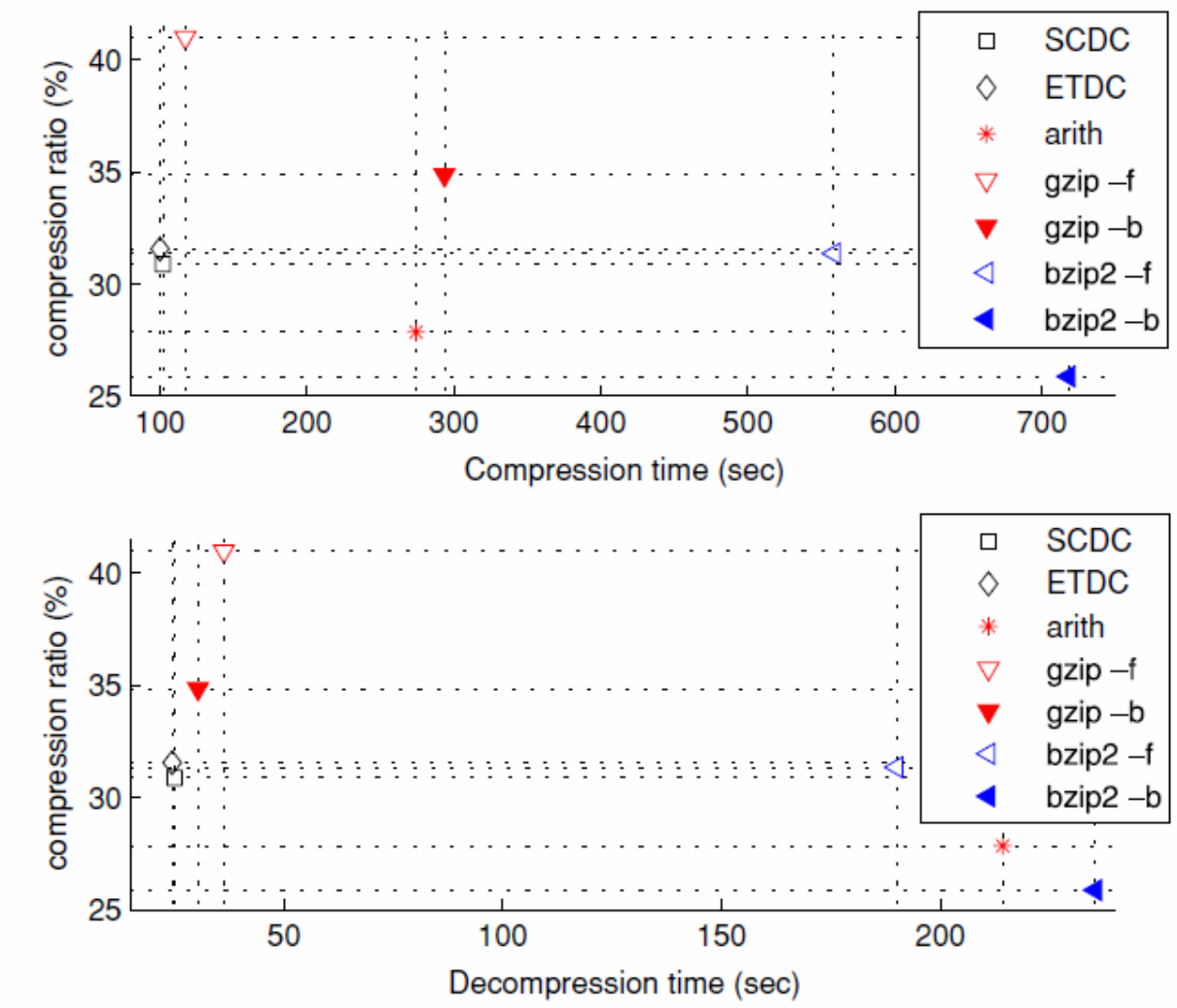
Compresores semistáticos densos

Resultados Empíricos : Resumen



Compresores semistáticos densos

Resultados Empíricos : Resumen



Compresores semistáticos densos
Resultados Empíricos : Resumen

- Compresores "**densos**" semiestáticos: **ETDC** y **SCDC**
- Codificación más simple y rápida que los basados en Huffman.
  - Codificación secuencial
  - Codificación directa ("al vuelo")
- Permiten búsqueda directa y acceso aleatorio
- **Velocidad**: Buena velocidad de compresión y descompresión
- Ratio de compresión próximo a Plain Huffman
- ***Superan*** a Tagged Huffman en (todo):
  - Ratio de compresión,
  - Velocidad de compresión y de descompresión
  - Velocidad de búsquedas.

Compresores semistáticos densos
Ejercicio

- Muestra la codificación ETDC que se obtiene para los vocabularios siguientes (asúmanse bytes de “sólo” 3 bits)

Word	Probab.
A	1/17
B	1/17
C	1/17
D	1/17
E	1/17
F	1/17
G	1/17
H	1/17
I	1/17
J	1/17
K	1/17
L	1/17
M	1/17
N	1/17
O	1/17
P	1/17
Q	1/17

Distribución uniforme: p_i=1/17

Word	Probab.
A	1/2
B	1/4
C	1/8
D	1/16
E	1/32
F	1/64
G	1/128
H	1/256
I	1/512
J	1/1024
K	1/2048
L	1/4096
M	1/8192
N	1/16384
O	1/32768
P	1/65536
Q	1/65536

Distribución exponencial: $p_i = 2^{-i}$

- ¿En qué se diferencian?
- ¿En qué caso se obtiene una compresión mejor? Justifica la respuesta.
- Y si el vocabulario tuviese 20.000 símbolos. ¿qué codigo le correspondería al símbolo en la posición 16.512 (para SCDC o ETDC)