Compresión de Datos Mediante Meta-Símbolos

José de Jesús Galaviz Casas

2006

A
Marcela Casas,
Fabienne Herviou y
Carlos Casas
(mi madre y abuelos maternos, *in memoriam*).

A
Mónica Leñero,
Claudia Galaviz y
Arturo Galaviz
(mi esposa y mis hijos).

Agradecimientos

Este trabajo no es resultado de los esfuerzos solitarios del que aparece como autor y al que son atribuibles los posibles errores que en él se encuentran. Es el producto *iunctis viribus* de un muy nutrido conjunto de generosas personas, de las que me daré a la tarea de enumerar a las más obvias.

El Dr. Ángel Kuri es quien gestó la idea del método de compresión que aquí se estudia, es también quien invirtió denodados esfuerzos aleccionandome junto a mis condiscipulos, quien organizó nuestro grupo de trabajo, quien corrigió nuestros errores y discutió y analizó con nosotros cuanta peregrina idea se nos ocurría, quién, además de ser la brújula, fue también siempre el compañero de camino. En el ámbito puramente profesional, mis agradecimientos hacía el Dr. Kuri son infinitamente numerables. En el ámbito personal son continuos.

A las doctoras Katia Rodriguez y Cristina Verde, y los doctores Francisco Cervantes, Edgar Chavez, Marcelo Mejía y Liu Tang, quienes siempre estuvieron dispuestos a revisar mi trabajo y a invertir generosamente el propio durante el proceso, a prodigar valiosas sugerencias y señalar rutas nuevas. Agradezco especialmente a la Dra. Verde sus generosos consejos, meta-consejos y apoyo moral permanente.

A mi familia, entendida esta cómo el conjunto de personas con quienes tengo el privilegio de compartir la vida: a mi esposa Mónica por soportarme cuando ni yo mismo me soportaba, a mis aproximadamente pacientes hijos Claudia y Arturo, por cederme algo de su tiempo. A mis tíos, mis suegros (es en serio), mis primos, mis cuñados y hermanas virtuales por estar siempre *allí* y a mi madre y mis abuelos maternos por haber estado siempre *allí* (hasta que tuvieron que irse *allá*).

A mis buenos amigos Elisa Viso y Francisco Raggi, de quienes tengo el privilegio de aprender todos los días algo nuevo y con quienes, al igual que con los números naturales, siempre se puede contar.

Índice general

Resumen

La mayoría de los métodos de compresión comúnmente usados en la actualidad están basados en diccionarios: si una cadena de símbolos es encontrada repetidamente en los datos que se pretenden comprimir, esta cadena se añade a un diccionario y cada aparición de ella en los datos es reemplazada por una referencia a la posición que ocupa en dicho diccionario. Una generalización de este esquema consiste en considerar no sólo cadenas de símbolos consecutivos, sino también patrones constituidos por una combinación de símbolos y "huecos" o comodines de longitud arbitraria. En este trabajo se expone un método de compresión basado en esta generalización, tanto en sus aspectos teóricos como en aquellos involucrados con su realización práctica. Las principales aportaciones del trabajo son: el análisis de complejidad de los problemas involucrados en la compresión de datos en general y la compresión basada en patrones en particular y la formulación de una heurística que aproxima su solución.

1

Introducción

El término *compresión de datos* se refiere al proceso de transformar un conjunto de datos de entrada de cierto tamaño, en otro conjunto de datos de salida con un tamaño menor, preservando las cualidades esenciales que le dan significado a los datos. Es posible distinguir entre dos enfoques diferentes en el área: la llamada compresión de datos *sin pérdida* (o *lossless* en inglés) y la que involucra la pérdida de cierta información considerada no esencial, llamada *con pérdida* (o *lossy*).

El primer tipo de compresión es útil en todo contexto, pero sobre todo cuando se requiere recuperar los datos originales exactamente, a partir de los datos comprimidos; por ejemplo cuando se comprimen programas ejecutables o texto. El segundo tipo es útil en el contexto de la compresión de datos de audio, imágenes o video, en los que es usual considerar irrelevante la pérdida de información no audible o no apreciable a simple vista.

El método que motiva el presente trabajo pertenece a la clase de los *sin pérdida*.

1.1 Marcos conceptuales de la compresión de datos

Existen dos marcos teóricos dentro de los que se encuadra la compresión de datos: la hoy denominada *teoría de la información*, fundada por Claude E. Shannon en su célebre artículo de 1948 *Mathematical Theory of Communication* [Shannon48] y la llamada *teoría de la complejidad de Kolmogorov* o *teoría algorítmica de la complejidad*, descubierta independientemente por R.J. Solomoff, A.N. Kolmogorov y G.J. Chaitin en la década de los 60 del siglo pasado [Chaitin66, Kolmogorov65, Solomonoff64].

La teoría de la información se enfoca en la fuente que produce el mensaje a comprimir [Grünwald03]. La compresión se obtiene por medio de la codificación óptima (de mínima redundancia), determinada por las características de la fuente de información, la cuál es modelada mediante una cadena de Markov ergódica. Dada una fuente particular, la teoría de la información provee los medios para encontrar el código que minimiza la longitud promedio, necesaria para expresar cualquier mensaje producido por la fuente, pero no la codificación óptima para un mensaje particular.

En una situación práctica, sin embargo, es infrecuente tener el conocimiento de las características de la fuente de información que produce los datos a comprimir. En general sólo se poseen éstos, es decir, una muestra finita de lo que la fuente produce, a través de la que sólo es posible estimar, por medio de análisis estadístico, las características fundamentales para obtener el código óptimo, presuponiendo que dicha muestra es representativa. De esta forma, características tales como el orden de la cadena de Markov y su distribución de probabilidades a largo plazo, son calculadas con base en la evidencia recolectada de la muestra que se pretende comprimir.

En contraste, la teoría de Kolmogorov se enfoca en objetos individuales en vez de en la fuente que los produce [Grünwald03, Li+97]. La complejidad binaria de Kolmogorov de una secuencia de símbolos particular es la longitud, en bits, del programa de computadora más corto que la produce y luego se detiene. Formalmente hablando, esto es la longitud en bits de la descripción de la máquina de Turing que únicamente imprime la secuencia en cuestión. En el contexto de la teoría de Kolmogorov, la compresión se logra reemplazando la secuencia de símbolos por el programa que la imprime. De esto se concluye, de inmediato, que una secuencia incompresible es aquella que sólo puede describirse por ella misma, no posee ninguna regularidad que pueda ser sintetizada en un programa. Por desgracia la complejidad de Kolmogorov y por tanto el programa que la define, no son funciones computables en general [Grünwald03, Li+97].

El panorama está constituido entonces por dos vertientes complementarias: la una dedicada al modelaje de la fuente que produce los datos a comprimir, la otra dedicada a los datos mismos sin pretensión alguna de extrapolar sus características. La primera tiene dos problemas: por una parte nunca se puede estar seguro de que las estimaciones obtenidas con base en los datos disponibles sean correctas y por otra, las estimaciones de características generales pueden no proporcionar la manera óptima de expresar la muestra particular en la que están basadas, porque pueden haber sido derivadas de suposiciones incorrectas. La segunda vertiente tiene el inconveniente de que no es, en general, computable la expresión mínima de la muestra.

Los algoritmos de compresión tratan de resolver los problemas mencionados haciendo, implícitamente, un compromiso entre ambas alternativas. Utilizan lo que se denomina un *código universal de dos partes* o *two-part universal code* [Grünwald03, Li+97]. Este hace de puente entre los puntos de vista de la teoría de la información y la de Kolmogorov. La primera parte del código se construye con la evidencia recolectada del mensaje y constituye un modelo para las regularidades que se encuentran en él. En la segunda parte se describe, por un lado, todo aquello que constituye el mensaje y que no ha podido ser capturado por las regularidades descritas en la primera parte y por otro lado, haciendo uso del lenguaje propio de una máquina abstracta, la manera en que las regularidades han de ser utilizadas para reconstruir el mensaje original.

En los conocidos algoritmos de la familia Lempel-Ziv [Ziv+77, Ziv+78], por ejemplo, la primera parte es un diccionario de cadenas de símbolos encontradas frecuentemente en la muestra. La segunda parte está constituida por la descripción del contenido de la muestra que no está siendo descrita por el diccionario entremezclada con referencias a él. Durante el

proceso de descompresión la máquina virtual implícita en el algoritmo sabe como interpretar las referencias para reconstruir la muestra original.

En este contexto resulta oportuno hacer referencia al *Principio de Longitud de Descripción Mínima* o *Minimum Description Length Principle* (MDL) formulado por J. Rissanen en 1978 [Rissanen78]. En él se establece, en términos generales, que el mejor modelo para explicar un fenómeno es aquel cuya descripción es más breve; una reformulación del principio conocido como la *navaja de Occam* (*Occam razor*). Lo que se pretende con un método de compresión, visto como un código de dos partes, es describir, en la primera parte, una hipótesis acerca de la fuente que produjo la muestra (en términos de la teoría de la información) basada en la evidencia recolectada de la misma muestra; mientras que la segunda parte describe la muestra con ayuda de la hipótesis. El principio MDL establece que la mejor hipótesis es aquella que minimiza simultáneamente la longitud de ambas descripciones [Li+97, Rissanen78].

Más adelante se formulará el método de compresión que motiva este trabajo en términos del principio MDL.

1.2 Conceptos de teoría de la información

En el artículo de Shannon se formula el denominado *teorema de la codificación sin ruido*, también llamado *primer teorema de Shannon* en el que queda establecida una cota máxima para la compresión en el contexto de la teoría de la información.

En el mismo artículo se deriva un método para lograr representar los datos producidos por una fuente de información, descrita en términos de su modelo estadístico, de manera breve. La representación descrita es conocida hoy día como los *códigos de Shannon-Fano*.

En términos generales una *fuente de información* es un ente abstracto que produce símbolos en un alfabeto finito, los símbolos producidos por una fuente son una sucesión *estacionaria* de variables aleatorias, es decir, la probabilidad de que sea producida una cierta secuencia de símbolos no cambia con el tiempo. De hecho el modelo propuesto por Shannon para una fuente de información es una *cadena de Markov ergódica*, esto es, hay una distribución de probabilidades límite para los símbolos del alfabeto.

Haciendo uso de este modelo Shannon define la cantidad de información (en bits) de cada símbolo s_i del alfabeto Σ de una fuente de información $\mathcal{S}$ como:

$$I(s_i) = \log_2 \frac{1}{p_i} \tag{1.1}$$

donde p_i es la probabilidad de que el símbolo s_i sea producido por $\mathcal{S}$ y $\log_2$ denota el logaritmo en base dos (de ahi que la medida sea en *bits de información*). Por supuesto la cantidad de información de cada símbolo es también una variable aleatoria.

Habiendo definido el concepto de información de un símbolo, se procede a definir el concepto de *entropía de información* como el valor esperado de la cantidad de información de los símbolos producidos por la fuente. La entropía de la fuente $\mathcal{S}$ queda entonces

determinada por:

$$H(\mathcal{S}) = \sum_{s_i \in \Sigma} p_i \, I(s_i) = \sum_{i=1}^{q} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{q} p_i \log_2 p_i \tag{1.2}$$

suponiendo que $\Sigma = \{s_1, \ldots, s_q\}$.

Cabe hacer notar que la cantidad de información de un símbolo es tanto mayor cuanto menos probable es y que la entropía de una fuente es tanto mayor cuanto más uniforme sea la distribución de probabilidades de los símbolos de su alfabeto. Para culquier fuente $\mathcal{S}$ con un alfabeto de q símbolos, se cumple:

$$H(\mathcal{S}) \leq \log_2 q \tag{1.3}$$

De hecho, en el caso de una fuente de información $\mathcal{S}$ con q símbolos equiprobables (i.e. con probabilidad $1/q$) se cumple la igualdad en la expresión anterior. La entropía es pues, una medida del *desorden* de la fuente. Cuanto más impredecible sea, mayor será su entropía.

1.3 Codificación de redundancia mínima

El concepto de entropía guarda un nexo muy valioso con la compresión de datos. Si algo puede predecirse de una fuente, se pueden expresar sus datos de manera más breve eliminando la redundancia implícita en su predictibilidad. Si hay símbolos con una alta probabilidad de ser producidos, en detrimento de la de otros (dado que, como en toda distribución, las probabilidades deben sumar 1), es posible pensar en asociar a los símbolos una representación tanto más corta cuanto mayor sea su probabilidad. Esta es, en esencia, la idea detrás de los códigos descubiertos independientemente por Shannon y Fano.

También es ésta la idea detrás de los códigos de Huffman [Huffman52], que resultan ser los más eficientes (para una demostración de esto puede verse [Roman97]). La eficiencia de un esquema de codificación está determinada por la longitud[1] promedio de las palabras de código necesarias para expresar los datos producidos por la fuente. Es decir:

$$L(C, \mathcal{S}) = \sum_{i=1}^{q} p_i \ell(C(s_i)) \tag{1.4}$$

donde L denota la longitud promedio de palabra para una codificación C de los símbolos del alfabeto de la fuente $\mathcal{S}$. $C(s_i)$ denota la palabra de código asociada al símbolo s_i y $\ell(C(s_i))$ es la longitud de dicha palabra en bits.

El primer teorema de Shannon establece que la entropía de una fuente es la cota inferior de la longitud promedio de palabra de todos los esquemas de codificación cuya decodificación es *decidible* (llamados también *unívocamente decodificables*) lo que garantiza la recuperación

[1] Medida en bits, para ser consistente con la medida de entropía. Podría, arbitrariamente, cambiarse la base, siempre y cuando la usada en la entropía sea la misma que es usada para la longitud de palabra.

de los datos originales. Es decir, para cualquier esquema de codificación C unívocamente decodificable, se tiene que:

$$H(\mathcal{S}) \leq L(C, \mathcal{S}) \tag{1.5}$$

Este resultado es particularmente importante en compresión de datos dado que establece un límite infranqueable para la tasa de compresión alcanzable con esquemas de codificación unívocamente decodificables para una fuente $\mathcal{S}$.

De entre los esquemas del tipo mencionado, los de Huffman[2] poseen la cualidad de ser los más eficientes, como ya se ha señalado, porque tienen la mínima longitud promedio de palabra.

1.4 Descripción del trabajo

El presente trabajo está enfocado a la formulación de un método de compresión sin pérdida basado un diccionario de patrones de símbolos muy generales, el análisis de su factibilidad en términos de la complejidad de los problemas inherentes al método y la propuesta de heurísticas que permitan aproximar la solución de estos problemas dado que, como se verá, resultan ser intratables.

El capítulo 2 está dedicado a presentar el *estado del arte* en la compresión de datos sin pérdida, describiendo los métodos más sobresalientes y haciendo un análisis de ellos, que más adelante hace posible abstraer los elementos esenciales de la compresión y las diferencias entre los modelos en los que dichos métodos se basan.

En el capítulo 3 se definen formalmente, tanto el método propuesto, como los conceptos involucrados en él. Se formulan con precisión cada una de las tareas en las que puede dividirse el proceso de compresión.

El capítulo 4 presenta un análisis de la complejidad de los problemas inherentes a las tareas antes descritas. Se demuestra formalmente que estos resultan ser intratables.

El capítulo 5 está dedicado a formular algunas heurísticas que permitirán aproximar la solución de los problemas analizados y finalmente el capítulo 6 presenta los resultados experimentales obtenidos al probar el método propuesto, algunas conclusiones y perspectivas de trabajo futuro.

[2]De hecho los códigos de Huffman pertenecen a una subcategoría de los unívocamente decodificables, a los que se les denomina *instantáneos*.

2

Panorama de los métodos de compresión

2.1 Métodos basados en modelo estadístico

Siguiendo la idea expresada anteriormente, de asociar representaciones tanto más largas cuanto menos frecuentes sean los símbolos, resulta natural desear poseer un modelo estadístico de la fuente que produce los datos.

Probablemente el primer esquema de codificación que tomó ventaja de un modelo estadístico, para lograr una representación eficiente de los datos, fue el código Morse. Diseñado específicamente para transmitir mensajes en inglés, el código Morse asocia palabras hechas de dos símbolos a las letras del alfabeto, tanto más cortas cuanto más frecuentes en inglés. La aproximación al modelo estadístico del inglés usado por Morse, la adquirió visitando un taller de impresión y contando el número de letras en las cajas de tipos usadas por los impresores para componer las páginas. A la luz de los datos actuales el modelo de Morse no es del todo preciso, pero es una buena aproximación.

2.1.1 Códigos de Shannon-Fano

En la segunda demostración del teorema de la codificación sin ruido, Shannon expone un método para obtener una codificación eficiente de los símbolos producidos por una fuente ([Shannon48], pag. 17). Las longitudes de las palabras del código (binario) obtenido $\ell_1, \ell_2, \ldots, \ell_k$ satisfacen la desigualdad:

$$\log_2 \frac{1}{p_i} \leq \ell_i \leq \log_2 \frac{1}{p_i} + 1$$

donde p_i es la probabilidad de que el i-ésimo símbolo del alfabeto sea producido por la fuente. Esto implica que:

$$L(SF, \mathcal{S}) < H(\mathcal{S}) + 1$$

donde SF denota los esquemas de codificación de Shannon-Fano para los símbolos de la fuente $\mathcal{S}$. Lo que significa que esta codificación es sumamente eficiente.

El método de construcción expuesto por Shannon y descubierto independientemente por Fano, es el siguiente:

1. Dados el conjunto de símbolos del alfabeto $\{s_1, s_2, \ldots, s_n\}$ $(n < 1)$ ordenados de tal forma que sus probabilidades son, respectivamente: $p_1 \geq p_2 \geq \cdots \geq p_n$, se calcula el código binario de cada uno de ellos bit por bit de izquierda a derecha. Sea $j = 0$.

2. Si la lista A de símbolos y probabilidades tiene sólo un elemento se regresa código vacío.

3. Si no, se procede a dividir la lista A en dos sublistas A_1 y A_2 de tal forma que la suma de las probabilidades de los símbolos en A_1 difiera lo menos posible de la suma de los de A_2.

4. Se coloca un cero en el bit j de todos los símbolos de A_1 y un uno en los de A_2.

5. $j = j + 1$

6. Se aplica recursivamente el algoritmo a partir del paso 2 con cada sublista.

Este método, en esencia construye un árbol, llamado *árbol de decodificación*[1]. Se comienza por la raíz del árbol y cada descenso recursivo se bifurca un nodo dando lugar a dos subárboles binarios. En la figura 2.1 se muestra el algoritmo de Shannon-Fano en pseudocódigo; *idxini* e $idxfin$ son, respectivamente, el índice inicial y final de la partición a procesar de la lista original, evidentemente la primera vez que se llama al algoritmo son el índice inicial y el final de la lista completa. La función suma obtiene la suma de las frecuencias relativas o probabilidades, de los elementos en la lista que están entre los índices que recibe como argumento. La función arbolbin construye un árbol binario de un solo nodo (la raíz) cuyo contenido es el argumento de la función. En la segunda llamada a arbolbin (línea 16) el argumento no importa (*cualquiersimb*). Las funciones setSubIzq y setSubDer establecen el subárbol izquierdo y derecho de la instancia de árbol binario que las llama, en este caso *arbol* es el identificador de dicha instancia. Hay que notar que en las líneas 17 y 18 se llama recursivamente al proceso de construcción. El **if** de la línea 1 establece la condición de terminación de la recursión. Al final el proceso regresa un árbol binario que constituye el árbol de decodificación de Shannon-Fano.

Por supuesto en la implementación práctica de los códigos de Shannon-Fano las probabilidades son reemplazadas por sus estimadores, las *frecuencias relativas de los símbolos* en la muestra finita que se posee.

[1] Todo código instantáneo tiene asociado un árbol de decodificación.

```
SHANNON-FANO(idxini, idxfin, lista)
 1  if (idxini = idxfin) then
 2      return (arbolbin(lista[idxini]))
 3  else
 4      idxpart = idxini
 5      ssup = suma(lista, idxini, idxpart)
 6      sinf = suma(lista, idxpart + 1, idxfin)
 7      dif = | ssup − sinf |
 8      difant = dif
 9      while (difant ≥ dif) do
10          idxpart = idxpart + 1
11          ssup = suma(lista, idxini, idxpart)
12          sinf = suma(lista, idxpart + 1, idxfin)
13          difant = dif
14          dif = | ssup − sinf |
15      endwhile
16      arbol = arbolbin(cualquiersimb)
17      arbol.setSublzq ( SHANNON-FANO (idxini, idxpart − 1, lista))
18      arbol.setSubDer ( SHANNON-FANO (idxpart, idxfin, lista))
19      return arbol
20  endif
21  end
```

Figura 2.1: Algoritmo de construcción del árbol de decodificación de Shannon-Fano.

2.1.2 Códigos de Huffman

En 1952 David Huffman [Huffman52] inventó el esquema de codificación que lleva su nombre y que constituyó, durante dos décadas, el mejor medio para comprimir datos y aún hoy en día subyace en algoritmos de compresión más elaborados. La tasa de compresión alcanzada con códigos de Huffman es similar a la de Shannon-Fano, sin embargo, como ya se mencionó, el esquema de Huffman es el más eficiente de entre aquellos de decodificación instantánea, así que aquel siempre es, a lo más, tan eficiente cómo este. De hecho cuando las probabilidades son potencias exactas de 2 ambos son igualmente eficientes.

Para calcular los códigos de Huffman de los símbolos de la fuente, se construye el árbol de decodificación a partir de las hojas (a diferencia del de Shannon-Fano que comienza por la raíz).

1. Dados el conjunto de símbolos del alfabeto $\{s_1, s_2, \ldots, s_n\}$ $(n < 1)$ ordenados de tal forma que sus probabilidades son, respectivamente: $p_1 \geq p_2 \geq \cdots \geq p_n$, se asocia a cada símbolo s_i un nodo hoja aislado N_i de un árbol binario. Sea A la lista de las ternas (s_i, p_i, N_i).

2. Sean $A[r]$ y $A[t]$ las dos entradas de la lista A que corresponden a los dos símbolos con menor probabilidad.

3. Se eliminan $A[r]$ y $A[t]$ de A.

4. Se construye un nuevo nodo $N_{r,t}$ que constituye la raíz de un nuevo árbol binario cuyos descendientes son N_r y N_t.

5. Se asocia con este nuevo nodo un símbolo irrelevante $\rho_{r,t}$ y una probabilidad $p_{r,t} = p_r + p_t$.

6. Se añade la terna $(\rho_{r,t}, p_{r,t}, N_{r,t})$ a la lista A.

7. Regreso al paso 2 hasta que A tenga sólo una terna.

8. Se etiquetan las aristas del árbol de la única terna de A: las aristas de salida a la izquierda con un 0 y las de la derecha con 1 (o viceversa, pero consistentemente). El resultado es el árbol de decodificación de Huffman.

En la figura 2.2 se muestra el algoritmo de Huffman en pseudocódigo. En él se está suponiendo que cada elemento de la lista es en realidad una estructura constituida por un árbol binario, al que se denotará con ab, y una frecuencia relativa o probabilidad, que será denotada frec. El atributo frec contiene la suma de las frecuencias de aquellos símbolos que se encuentran en el árbol ab. La función getMin se encarga de encontrar y regresar el elemento de la lista con la mínima frecuencia y eliminaMin de eliminarlo. Las funciones setSubIzq y setSubDer establecen al subárbol izquierdo y derecho, respectivamente, de la instancia de árbol que las llama, en el caso de la figura 2.2, el nombre de la instancia

```
HUFFMAN(lista)
 1  while (lista.longitud > 1) do
 2      elem1 = lista.getMin
 3      lista.eliminaMin
 4      elem2 = lista.getMin
 5      lista.eliminaMin
 6      arbol = arbolbin(cualquiersim)
 7      arbol.setSubIzq(elem1.getArbol)
 8      arbol.setSubDer(elem2.getArbol)
 9      elemnuevo.setArbol(arbol)
10      elemnuevo.setFrec(elem1.getFrec + elem2.getFrec)
11      lista.insertaOrden(elemnuevo)
12  endwhile
13  return (lista.getElem(0).getArbol)
14  end
```

Figura 2.2: Algoritmo de Huffman

es *arbol*. getArbol regresa el atributo ab del elemento que llama a la función y setArbol establece dicho atributo, getFrec y setFrec hacen lo análogo para el atributo frec. getElem entrega el elemento de la lista indexado por el argumento de la función, se supondrá que el índice inicial es el 0. insertaOrden inserta el elemento que recibe como argumento en la lista preservando el orden respecto al atributo frec.

Una vez construido el árbol de decodificación, para obtener el código de un símbolo s_i, simplemente hay que escribir de izquierda a derecha, las etiquetas de las aristas que hay que recorrer, desde la raíz del árbol, para llegar a la hoja que asociada con s_i.

A pesar de que el código de Huffman es el más eficiente, sólo cuando las probabilidades de los símbolos son potencias negativas exactas de 2 la longitud promedio de palabra es igual a la entropía (lo que significa que no puede haber código más eficiente).

Extensiones de una fuente. La manera de incrementar la eficiencia del esquema de Huffman (y de hecho, de cualquier otro) es considerando la codificación, ya no de los símbolos individuales del alfabeto de la fuente, sino de secuencias de ellos. En una fuente con alfabeto binario $\{a, b\}$, sin importar las probabilidades de a y b, Huffman asignaría un bit a cada símbolo. Esto no es, ciertamente, lo más eficiente si, por ejemplo, $p(a) = 0.9$ y $p(b) = 0.1$. Pero si consideramos entonces las probabilidades de todos los digramas $\{aa, ab, ba, bb\}$, se considera cada digrama como un símbolo y a la fuente como productora de digramas y ya no de símbolos individuales; entonces la codificación de Huffman de esta nueva fuente asignará menos de un bit, en promedio, a los símbolos a y b originales. De hecho les asignará $L_2/2$ bits, donde L_2 denota la longitud promedio de palabra en bits, para la fuente de digramas.

A la fuente obtenida considerando los digramas como símbolos se le denomina *segunda*

extensión de la fuente. Por supuesto es posible pensar en trigramas, tetragramas, etc. Es decir en la n-ésima extensión de la fuente original, con n un entero positivo arbitrario. Generalmente $\mathcal{S}^n$ denota la n-ésima extensión de la fuente $\mathcal{S}$.

Resulta que la codificación de Huffman para $\mathcal{S}^n$ posee una longitud promedio de palabra L_n, tal que, el número de bits promedio asociado a cada símbolo del alfabeto de la fuente original L_n/n tiende a la entropía de la fuente original $H(\mathcal{S})$. Es decir, se puede ser tan eficiente como se desee usando sólo códigos de Huffman. El problema, por supuesto, es que esto no es práctico. El modelo estadístico de $\mathcal{S}$ tiene q probabilidades, donde q es el número de símbolos en el alfabeto original. Pero el de $\mathcal{S}^n$ tiene q^n probabilidades, las que hay que calcular o almacenar para poder reconstruir el árbol de decodificación al momento de descomprimir. Conviene llamar la atención del lector a este resultado que vincula la tasa de compresión con la complejidad del modelo estadístico de la fuente, más adelante se hará referencia a él.

Codificación de Huffman dinámica. Uno de los factores desfavorables para el uso de la codificación de Huffman en compresión es que, como se ha mencionado, se requiere guardar, junto con los datos comprimidos, el modelo estadístico de la muestra que se comprimió, para que el proceso de decodificación pueda llevarse a cabo y garantizar que se genera el mismo árbol de decodificación que se usó para comprimir. Otro inconveniente es que, para obtener el modelo hay que dar una "pasada" de análisis a la muestra para calcular el modelo y luego darle una segunda pasada para comprimir. Con el propósito de eliminar estos inconvenientes se propuso [Gallager78, Vitter87] ir construyendo el modelo a medida que se lee la muestra y se codifica, en la misma pasada. Es decir cada símbolo leído de la muestra es codificado usando el modelo estadístico actual y al mismo tiempo, el símbolo modifica el modelo haciéndolo más preciso. Por supuesto al inicio el modelo es nulo. A esta alternativa se le suele llamar *codificación de Huffman adaptable* o *dinámica*.

2.1.3 Codificación Aritmética

Cómo ya se dijo la codificación de Huffman sólo alcanza la eficiencia máxima cuando las probabilidades (o las frecuencias relativas que las estiman) son una potencia exacta de 2 (o en general de la base del alfabeto). Sólo en ese caso la longitud promedio de palabra de Huffman es igual a la entropía de la fuente. En otro caso la longitud promedio sólo se aproxima a la entropía por arriba. El error de aproximación puede reducirse a menos de ε considerando la k-ésima extensión del alfabeto, para k suficientemente grande, como se ha señalado.

Con esto en mente se propuso en [Rissanen76] un esquema de codificación que no requiriera la consideración de extensiones del alfabeto, al menos no explícitamente. El método, bautizado como *codificación aritmética*, se retomo y refinó en [Rissanen+79, Langdon84]. Para una excelente exposición de la forma en que se conoce ahora, se pueden consultar [Witten+87, Salomon00, Nelson+96b].

El objetivo es codificar una muestra completa mediante un único número real en el

intervalo $[0, 1)$, en vez de usar códigos para bloques de k símbolos (de la k-ésima extensión de la fuente).

Se parte del modelo estadístico de primer orden, es decir, la tabla de frecuencias relativas o probabilidades de los símbolos individuales que se supondrá son $\{s_1, s_2, \ldots, s_n\}$ con sus respectivas $p_1 \geq p_2 \geq \cdots \geq p_n$ ordenadas decrecientemente (aunque esto no es necesario). Aprovechando el hecho de que la suma de las probabilidades debe ser 1, se asocia a cada símbolo s_i, un subintervalo $I_i = [B_i, U_i) \subset [0, 1)$, de tal forma que $U_i - B_i = p_i$. Estos intervalos cumplen con:

$$\begin{aligned} \bigcup_{i=1}^{n} &= [0, 1) \\ I_i \cap I_j &= \emptyset, \text{ si } i \neq j \end{aligned}$$

es decir, son una partición de $[0, 1)$.

La idea es que, una vez leído el primer símbolo s_{r1} de una muestra que se desea codificar, el código asociado a toda la muestra está contenido en el intervalo asociado a s_{r1}. Al leer el segundo símbolo s_{r2}, el intervalo en el que está el código de la muestra se reduce, anidando el que corresponde a s_{r2} en el anterior, como si este fuera el $[0, 1)$. Se procede así hasta que se anida el intervalo del último símbolo de la muestra. Cualquier número real en el último intervalo codifica a la muestra completa unívocamente.

1. $B = 0$, $U = 1$, $i = 1$
2. Mientras haya símbolos en le muestra
3. Leer el siguiente símbolo s_{ri}
4. $rango = U - B$
5. $U = B + rango\ U_{ri}$
6. $B = B + rango\ B_{ri}$

En la figura 2.3 se muestra en pseudocódigo el algoritmo descrito. $\mathsf{superior}(s)$ e $\mathsf{inferior}(s)$ denotan, respectivamente el límite superior e inferior del intervalo asociado al símbolo s, que se han denotado con U_i y B_i.

Para decodificar se procede a hacer las operaciones inversas. Es decir se toma el código de la muestra y se observa en cuál de los intervalos de símbolo está, esto recupera el primer símbolo de la muestra. Luego se procede a reescalar el intervalo que queda luego de restarle al original el límite inferior del intervalo del símbolo. Esto genera un nuevo intervalo con el que se procede a repetir el procedimiento. En la figura 2.4 se muestra el pseudocódigo de este algoritmo.

La ventaja fundamental de la codificación aritmética estriba en que su tasa de compresión es muy alta. El número de dígitos necesarios para expresar la muestra, dividido por el número de símbolos, es muy cercano al estimador de la entropía medido sobre la muestra.

```
CodifAritm(muestra)
1  B = 0
2  U = 1
3  for (s ∈ muestra) do
4      rango = U − B
5      U = B + rango ∗ superior(s)
6      B = B + rango ∗ inferior(s)
7  endfor
8  return (B)
9  end
```

Figura 2.3: Algoritmo de codificación aritmética.

```
DecodAritm(cod)
 1  i = 1
 2  while (cod > 0) do
 3      s = simbolo(cod)
 4      muestra[i] = s
 5      i = i + 1
 6      rango = superior(s) − inferior(s)
 7      cod = cod − inferior(s)
 8      cod = cod/rango
 9  endwhile
10  return (muestra)
11  end
```

Figura 2.4: Algoritmo de decodificación aritmética.

Sin embargo la codificación aritmética, tal como se ha presentado, requiere del manejo de números reales de precisión arbitraria. De allí que en [Moffat+98] se haya decidido usar aritmética entera en lugar de números reales.

2.1.4 Codificación de Markov dinámica

Este método pretende encontrar la cadena de Markov de primer orden que mejor modela los datos contenidos en la muestra, fue propuesto por primera vez en [Cormack86]. A diferencia del método de Huffman, que pre-supone que los datos de muestra son una cadena de símbolos generados independientemente con una cierta distribución estacionaria, el método de cadena de Markov dinámica (DMC por sus siglas en inglés) supone que existen correlaciones entre los símbolos.

El método parte de un modelo muy simple, una cadena de Markov de un sólo estado, suponiendo una muestra binaria, este estado tiene transiciones hacia sí mismo luego de recibir un cero o un uno. Las transiciones poseen una cierta probabilidad de ocurrencia que se va adecuando de acuerdo con las frecuencias relativas observadas al leer la muestra. Rápidamente este modelo simple adquiere mayor complejidad, se añaden nuevos estados en un proceso llamado *clonación*, en el siguiente paso la cadena de Markov posee dos estados uno en el que se permanece viendo ceros y otro en el que se permanece viendo unos, si se ve el bit contrario se transita al estado complementario. En general un estado e, es clonado cuando sus transiciones de llegada poseen probabilidades muy diferentes, porque esto significa que, de llegarse a uno de los estados conectados a la salida de e, se perderá la información histórica acerca de la ruta usada para llegar hasta él y los caminos ocurren con muy distintas probabilidades, lo que significa que muy posiblemente haya una correlación entre el estado al que se llega desde e y alguno de los que llegan a e; ante esta situación se decide clonar a e generando un nuevo estado e' que recibe algunas de las transiciones de e y posee las mismas salidas que él.

Estando en un estado particular, el modelo de Markov posee una tabla de probabilidades estimadas para todos los símbolos del alfabeto. Esta tabla se utiliza para codificar aritméticamente, el símbolo que realmente tuvo lugar, por supuesto la ocurrencia de este símbolo modifica el modelo mismo. En [Cormack86] se utiliza una variante de la codificación aritmética llamada codificación de Guazzo, en la implementación mostrada en [Yu96] se utiliza la codificación aritmética tal como fue descrita aquí.

2.1.5 Predicción por concordancia parcial (PPM)

PPM es generalmente considerado como el mejor método de compresión, el *estado del arte*. Fue propuesto en [Cleary94] e implementado en [Moffat90]. En cierta forma es muy parecido a DMC: se posee un modelo estadístico inicial muy simple que se va refinando conforme se procesa la muestra y las probabilidades del modelo se usan para hacer codificación aritmética de los símbolos.

Sin embargo PPM es menos complejo que DMC dado que no mantiene un conjunto de

estados y transiciones que modelen el contexto en el que aparece cada símbolo. En PPM el modelo de partida es la tabla de frecuencias relativas de cada símbolo, al principio los símbolos son codificados usando esta información y cada nuevo símbolo actualiza el modelo. Después el modelo se complica considerando no sólo las probabilidades estimadas de cada símbolo, sino la probabilidad condicional del siguiente símbolo dado el actual, es decir las probabilidades condicionales de primer orden. En un texto en español, por ejemplo, la probabilidad de "u" es 0.0401 aproximadamente, pero la probabilidad de "u" luego de "q" es 1, así que su código será bastante menor si se consideran las estadísticas de primer orden.

Formalmente hablando, el modelo de PPM puede continuar incrementando su complejidad arbitrariamente. Pero por supuesto el número de probabilidades condicionales a considerar depende del tamaño del alfabeto en relación exponencial con el orden del modelo estadístico, así que se vuelve prácticamente inmanejable con rapidez. Generalmente se utiliza un modelo de hasta orden 3.

Es de notar el vínculo que esto tiene con las extensiones de la fuente, o del alfabeto de la fuente, mencionadas anteriormente. PPM pretende acercarse a la cota de la entropía, considerando justamente las extensiones de digramas y trigramas del alfabeto. La ventaja es que PPM no tiene en consideración digramas o trigramas que no haya visto previamente, es decir, no mantiene permanentemente la información de todos los posibles contextos.

Cuando el contexto actual, constituido por, digamos, los tres símbolos previos, no se encuentra en el árbol de contextos de PPM, se inserta un símbolo especial llamado *escape* y se busca un subcontexto menor considerando sólo los dos símbolos previos. De no hallarse este se inserta un nuevo escape y se buscan, paulatinamente, contextos menores, hasta hallar uno que empate con el actual. En caso de no hallar ninguno, ni siquiera de orden cero, lo que significa que el símbolo actual no se había visto antes, se procede a introducir $N-1$ símbolos de escape (donde N es el orden del modelo de PPM) y codificar el símbolo asignándole una probabilidad $1/|\Sigma|$, donde $|\Sigma|$ es el tamaño del alfabeto.

2.1.6 Codificación predictiva

El método de codificación predictiva fue propuesto inicialmente en [Elias55] y ha sido retomado en múltiples ocasiones ([Hamming80] [Einarsson91, Jagmohan02], por ejemplo). Probablemente la exposición más detallada sea la contenida en [Hamming80] (pags. 88–94).

El objetivo es predecir el siguiente símbolo de una muestra con base en un modelo estadístico de la misma. La predicción es tanto más exacta cuanto mayor sea el orden del modelo, pero, como se ha mencionado en la exposición de métodos anteriores, los modelos de orden mayor a tres resultan prácticamente inmanejables.

Supóngase que se posee una muestra binaria. Con base en los anteriores k bits leídos de la muestra y el modelo estadístico, se predice el siguiente bit, digamos el j-ésimo. Si se acierta, se coloca un 0 en el lugar j de una cadena binaria, en otro caso se coloca un 1, indicando que la predicción genera un error. Al final, luego de predecir todos los bits de la muestra, se posee una nueva cadena binaria de errores de predicción que, se espera,

tenga secuencias largas de ceros consecutivos interrumpidas por unos aislados y esporádicos. Estas secuencias de ceros pueden codificarse usando codificación por longitud de corridas (*run-length encoding*), usando un cierto número de bits para indicar cuantos ceros hay en cada corrida antes de encontrarse el siguiente 1. Luego se puede aplicar algún otro método de compresión a la secuencia de números resultante. En [Hamming80] se hace un análisis para determinar la longitud en bits, de cada uno de los números que indican las longitudes de las corridas de ceros, con base en la probabilidad de error de predicción.

2.2 Métodos basados en diccionario

Una alternativa a los métodos estadísticos expuestos, la constituyen los métodos basados en diccionario. Lo que se pretende, en términos generales, es tener un diccionario en el que se incluyan secuencias de símbolos *consecutivos* que aparecen frecuentemente en la muestra a comprimir. Cada vez que se encuentra una de estas cadenas en la muestra, es reemplazada por una referencia al lugar del diccionario donde se encuentra, por supuesto la longitud de la referencia se espera que sea mucho menor que la cadena que substituye.

Hay dos posibles maneras de concebir al diccionario:

- Estático. Esto es, el diccionario está pre-definido, contiene palabras de uso frecuente en el contexto de la muestra (lo que hace que el compresor/descompresor sea sólo de propósito específico) o bien frecuentes en un contexto más amplio (lo que hace que el diccionario sea de tamaño inmanejable).

- Dinámico. El diccionario se construye durante el proceso de compresión y es característico de la muestra. El diccionario se incluye como parte de la muestra comprimida, por lo que se hace necesario un segundo proceso de compresión del diccionario mismo (que generalmente es un codificador de los descritos en la sección anterior, por ejemplo Huffman o aritmético).

2.2.1 LZ77

El método fué propuesto en [Ziv+77] (por lo que debería llamarse Ziv-Lempel y no Lempel-Ziv). El procedimiento para comprimir la muestra consiste en recorrerla con una ventana deslizante dividida en dos partes: una región de búsqueda o *search buffer* y una región de revisión llamada *look-ahead buffer*. Una vez colocada la ventana deslizante en una posición determinada de la muestra a comprimir, se busca la subcadena más larga posible que esté presente en la región de búsqueda y que coincida con el inicio del *look-ahead*. Una vez encontrada esta cadena, se reemplaza su aparición en el *look-ahead* por una referencia hacia atrás, a su aparición en la región de búsqueda. Es decir, el diccionario es la región de búsqueda de la ventana deslizante, misma que se va desplazando símbolo a símbolo hasta terminar con la muestra. Típicamente la región de búsqueda es del orden de miles de símbolos, mientras que el tamaño del *look-ahead* es de unos cientos.

A pesar de su simplicidad este método resulta ser uno de los mejores por su alta tasa de compresión. Ha dado lugar a variantes que, hoy día, están en la mayoría de los programas de compresión más exitosos (`gzip/gunzip`, por ejemplo).

Nuevamente conviene notar el vínculo entre este método y las extensiones del alfabeto de una fuente. Implícitamente LZ77 está buscando elementos de la k-ésima extensión de la fuente, donde k es el tamaño del *look-ahead*. Pero no pretende generar un modelo general para la k-ésima extensión, sólo está tomando en consideración aquellas cadenas de orden k que realmente aparecen en la muestra y no *todas las posibles*, lo que como es sabido, sería inmanejable. Pero tampoco pretende obtener todas las cadenas de orden k que se repiten, sólo las que le permite ver la ventana deslizante, esto es una desventaja; significa que la tasa de compresión será tanto mayor cuanto mayor sea la longitud de la ventana, pero entonces el proceso de búsqueda del mayor empate posible se complica, aunque permanece de complejidad lineal.

2.2.2 LZ78

Ya se mencionó la desventaja inherente de LZ77 al tener un diccionario de alcance limitado a la longitud de la región de búsqueda de la ventana. Para eliminar este problema, los mismos autores propusieron otro método un año después de LZ77 [Ziv+78].

En LZ78 es posible que las apariciones muy alejadas de una cadena hagan la misma referencia al diccionario. Una vez establecida una entrada del mismo esta permanece allí durante todo el proceso, no desaparece: el diccionario no puede decrecer. Conforme se procesa la muestra se añaden al diccionario cada vez más cadenas y cada vez más largas. lo que dificultaría las búsquedas si no se utilizara una estructura de datos adecuada. En LZ78 el diccionario está contenido en un árbol no binario, si se encuentra una cadena que comienza como alguna ya incluida que termina en cierto nodo ν, en el árbol del diccionario, se coloca la referencia en la muestra comprimida y se añade el último símbolo de la cadena como hijo de ν.

En [Welch84] se hizo una implementación de LZ78 que sirvió de base para el formato gráfico GIF y para el programa `compress` de UNIX.

2.3 Otros métodos

2.3.1 La transformada de Burrows-Wheeler

En 1994 fue propuesto un método [Burrows94] basado en un reacomodo de la muestra a comprimir con el fin de hacerla susceptible de una mayor compresión. En [Nelson96b] se describe este método, conocido cómo el algoritmo de Burrows-Wheeler, de manera sencilla.

El algoritmo de Burrows-Wheeler (BW) opera en tres fases:

1. BWT o transformación Burrows-Wheeler. Esta es la fase fundamental para lograr alta tasa de compresión. Básicamente, como se verá, consiste en reacomodar los datos de entrada para poder comprimirlos.

2. Algoritmo *Move-to-front*. En esta fase, a partir de la salida producida por la anterior, se producen datos a los que se les puede comprimir eficazmente con algoritmos conocidos que se aplican en la etapa siguiente.

3. Codificador basado en la entropía. en esta última fase se aplica un algoritmo de compresión que puede ser el de Huffman o el de compresión aritmética.

Las tasas de compresión alcanzadas por el algoritmo de Burrows-Wheeler son, en general, competitivas e incluso llegan a ser mayores que las de otros algoritmos populares, como puede observarse en los resultados presentados en [Burrows94]. Ocasionalmente, sin embargo, la aplicación del algoritmo puede resultar contraproducente[2].

A grandes rasgos el algoritmo opera como sigue:

1. BWT. Se toma un bloque de datos de N bytes. A partir de ese bloque se genera una matriz de $N \times N$ en la que el renglón inicial (de índice cero) contiene el bloque (cadena) de caracteres leídos tal cual y en su renglón i-ésimo aparece el mismo bloque rotado i lugares a la derecha.

   Considerando cada renglón como una cadena de caracteres, se ordenan los renglones crecientemente, en el renglón de índice cero de la matriz resultante aparece el renglón de valor mínimo lexicográficamente hablando. Por supuesto al menos un renglón de la matriz es el bloque original, sea r el índice de ese renglón.

   La salida de esta etapa consiste de:

   - r, el índice del renglón donde quedó la cadena original.
   - L, la última columna de la matriz ordenada (la del extremo derecho).

   La transformación hecha en esta etapa es reversible. Es decir se puede regenerar la cadena original a partir de la salida producida por esta etapa. Sin embargo las transformaciones hechas hacen posible obtener una buena tasa de compresión, dado que la última columna de la matríz tiende a estar constituida de corridas relativamente largas con el mismo símbolo. A esta columna se le aplica el siguiente proceso.

2. *Move-to-front*. Leyendo secuencialmente la cadena constituida por la columna L, se genera una cadena mixta M en la que hay tanto números como caracteres. La primera vez que se ve un caracter este se copia a la posición actual de M y es colocado en el tope de una pila inicialmente vacía. En las siguientes apariciones del caracter, se coloca en M su profundidad en la pila actual, en vez del caracter mismo y se reposiciona este en la pila, para volverlo a poner en el tope.

   El resultado esperado es que M esté constituida por corridas largas de "0", corridas un poco menos largas de "1", otras aún menos largas de "2" y así sucesivamente, todas ellas interrumpidas por caracteres que se ven por vez primera. Se espera que la frecuencia de aparición de los números posea una distribución exponencial.

[2] Cómo se advierte en el manual en línea de `bzip2`, utilería Unix que implementa el algoritmo BW.

3. Ahora se puede aplicar a M la codificación de Huffman (que es la utilizada en [Burrows94] y en `bzip2`).

2.3.2 Compresión con redes neuronales

Hay en general dos diferentes maneras de aplicar las redes neuronales a la compresión de datos.

En una de ellas [Verma+99] se fragmenta la muestra a comprimir en r fracciones que puedan ser suministradas a las k neuronas de la capa de entrada de una red neuronal tricapa. La capa oculta de la red está constituida de relativamente pocas neuronas ($j \ll k$), en comparación con la capa de entrada y la de salida (que también tiene k neuronas). Los fragmentos de la muestra son utilizados como conjunto de entrenamiento de la red, la salida esperada coincide perfectamente con la entrada (i.e. se pretende que la red repita lo que vio como entrada). La versión comprimida de la muestra la constituyen dos elementos:

1. Los k pesos de la capa de salida.

2. El conjunto de r salidas de la capa oculta.

Para recuperar los datos originales se utilizan los pesos del inciso 1 como los pesos de un conjunto de k neuronas que harán las veces de la capa de salida original. Se utilizan además las r salidas de la capa oculta de la red original para ser suministradas como entradas a las k neuronas de la "capa de salida". Las salidas de estas k neuronas reconstruyen los datos originales.

La otra manera [Schmidhuber92, Scmidhuber96] de usar redes neuronales para comprimir es, de hecho, usar la red como un *predictor* para utilizarlo en un esquema de compresión predictiva. La red es entrenada al tiempo que se procesa la muestra, si la red produce una salida que coincide con el siguiente símbolo en la muestra se indica que la predicción es correcta. Al descomprimir los pesos de la red son inicializados de la misma manera y se utiliza el mismo algoritmo de adecuación de pesos que cuando se comprimió, se corrigen aquellas posiciones de la muestra que ya se sabe que la red predice incorrectamente.

2.4 Discusión

Luego de la revisión hecha en este capítulo es claro que los algoritmos de compresión pretenden, directa o indirectamente, que la cantidad de bits promedio necesarios para representar cada símbolo de la muestra, sea tan próxima a la entropía estimada en la muestra como sea posible. Es claro que para lograr este objetivo hay que pensar en extensiones del alfabeto de la fuente, o lo que es lo mismo, pensar en modelos de orden superior para la muestra, que vinculen la aparición de un símbolo con el contexto histórico de los símbolo previos. El problema estriba en que, en principio, habría que considerar un modelo con historia ilimitada para acercarse tanto como se desee a la deseada cota de la entropía. Esto por supuesto es imposible y, aunque no lo fuera, resulta impráctico pensar en considerar

contextos históricos grandes, dado que el número de posibles contextos a considerar crece exponencialmente con la longitud de la historia.

Los algoritmos de compresión más exitosos y aquellos considerados como *el estado del arte* tienen la cualidad de adaptar el tamaño del contexto de acuerdo con la evidencia encontrada: los algoritmos LZ al encontrar una cadena repetida de longitud s suponen, implícitamente, que hay vínculos de orden $1, 2, \ldots, s$ entre los símbolos; PPM, de no estar acotado su contexto, supone permanentemente la existencia de vínculos de orden arbitrario por determinar.

La alternativa de LZ es muy razonable. De entre todas las posibles cadenas de longitud s de un alfabeto, sólo se consideran útiles para comprimir la muestra aquellas que aparecen en ella. En vez de suponer *a priori* que cualquiera puede aparecer y merece ser considerada, sólo se toman en cuenta las que realmente están presentes.

En el modelo más general, una muestra es producida por un proceso de Markov de orden k, del que se desconoce k, es decir la producción del i-ésimo símbolo depende de *todos y cada uno* de los previos k símbolos. LZ simplifica el problema restringiendo el valor de k, ya sea mediante la longitud del buffer de *look-ahead* (LZ77) o por la longitud de las cadenas encontradas (LZ78). Una simplificación menos restrictiva que la impuesta por LZ y que por tanto, proporciona una tasa de compresión potencialmente mayor, es suponer que el i-ésimo símbolo depende, en efecto, de los k anteriores, pero de algunos de ellos más que de otros dependiendo de los valores de los símbolos.

Por ejemplo. En español los trigramas ADE, ASE, ARE, AME están entre los 100 más frecuentes de todos los 27^3 posibles. El más frecuente de ellos es ADE. Pero por supuesto la probabilidad de que ocurra alguno de ellos es aún mayor (es la suma de las probabilidades de cada uno) y si se toma en cuenta además a los trigramas ANE, ATE, y a todos los que empiezan con A y terminan con E, la probabilidad de que alguno de ellos aparezca será todavía mayor, así que se puede pensar en que el patrón A*E, donde el * reemplaza a cualquier símbolo del alfabeto, es muy frecuente y por tanto su inclusión en un diccionario de patrones resultará provechosa. Cada vez que se encuentre en la muestra una A seguida de cualquier cosa y luego una E es posible hacer referencia al diccionario diciendo solamente quién está entre la A y la E.

Con esto en mente se puede pensar en un método de compresión basado en un diccionario de patrones de longitud arbitraria, con un número arbitrario de símbolos definidos separados por un número también arbitrario de *. Cuanto mayor y más frecuente sea un patrón, el costo de incluirlo en el diccionario se amortiza mejor. Implícitamente se están eliminando dos restricciones: no se presupone el orden del proceso de Markov que da lugar a la muestra, pero dado que la muestra es finita y no se posee evidencia de todos los vínculos a todos los ordenes, tampoco se supone que un símbolo depende en la misma medida de todos los anteriores, sino de algunos de ellos notablemente de acuerdo a la evidencia presentada por la muestra.

Luego del análisis de los diversos métodos de compresión es claro que todos ellos pretenden aproximarse, tanto como sea posible, a la cota establecida por la entropía estimada de la fuente que produjo la muestra que se desea comprimir. Las extensiones de alfabeto, que

se aproximan asintótica mente a la mejor tasa de compresión, constituyen el marco teórico fundamental. En la práctica, la aproximación se logra considerando el contexto previo de cada símbolo. La longitud del contexto considerado relevante es el elemento que mayor impacto tiene en la tasa de compresión alcanzada.

Los métodos de compresión basados en la codificación de Shannon-Fano o de Huffman, típicamente utilizan un contexto de orden cero, el modelo estadístico usado está constituido únicamente por la distribución de probabilidades del alfabeto, no se consideran relevantes las interacciones ningún orden entre los símbolos.

La capacidad de los métodos que consideran, explícitamente, modelos estadísticos de orden superior cómo PPM, DMC o la codificación predicativa está limitada severamente por el hecho de que el número de posibles contextos crece exponencialmente. La codificación aritmética, que implícitamente considera un modelo de orden igual al tamaño de la muestra, requiere de precisión potencialmente infinita para la representación numérica.

En el marco del principio de descripción mínima de Rissanen, el contexto constituye, de hecho, la hipótesis que se formula acerca de la fuente que produce los datos, la primera parte del código de dos partes que comprime la muestra

La capacidad de los métodos de diccionario que consideran dependencia de orden superior, sin generar un modelo estadístico que haga explícita dicha dependencia, se ve restringida al ignorar el hecho de que algunos elementos del contexto pueden ser más relevantes que otros.

En términos generales es posible caracterizar un método de compresión por el orden del contexto que toma en consideración y por la distinción que haga de la contribución que cada símbolo del contexto tiene para determinar el futuro. En los métodos revisados aquí, que toman en consideración un contexto de orden superior a cero, el orden máximo es acotado *a priori* y todos los símbolos en todas las posiciones son considerados igualmente importantes.

En cambio el método que se ha sugerido, basado en un diccionario de patrones, no acota *a priori* la longitud del contexto. Lo que resulta potencialmente peligroso por la explosión exponencial de posibles contextos. Entonces, para reducir la complejidad desechando lo irrelevante, propone diferenciar la influencia que cada símbolo del contexto tiene sobre los siguientes. Sólo los símbolos del contexto que la evidencia muestre que están claramente vinculados con los siguientes serán tomados en consideración. La generalidad de este método permite suponer que obtendrá tasas de compresión mayores que las obtenidas hasta ahora con los otros. Sin embargo habrá que hacer un análisis de su factibilidad.

3

Compresión de datos basada en patrones

Tal como se expuso en el capítulo previo, los métodos basados en un diccionario, incluyen en éste un conjunto de cadenas de símbolos consecutivos, de uso frecuente en la muestra que se pretende comprimir. Cada ocurrencia de una de estas cadenas en la muestra, es entonces reemplazada por una referencia a la posición que esta ocupa en el diccionario. Esta referencia debe ser, por supuesto, más corta que la cadena misma, con lo que se logra obtener la compresión de los datos preservando su integridad. Esencialmente este esquema es el mismo que fue formalizado en [Storer82] bajo el nombre de *substitución textual*.

3.1 Concepto de patrón

El objetivo del método que aquí se propone es construir un diccionario constituido, no sólo de cadenas de símbolos consecutivos, sino por patrones más generales, que admiten huecos o separaciones de tamaño arbitrario entre los símbolos que los constituyen.

Formalmente hablando, se define un patrón cómo sigue.

Definición 1 Dado un alfabeto finito de símbolos Σ, y un símbolo especial $\gamma \notin \Sigma$, llamado *separación*, *hueco* o *gap*. Un *patrón* o *meta-símbolo* p en Σ^+ es cualquier cadena en el lenguaje generado por la expresión regular:

$$\Sigma \cup [\Sigma(\Sigma \cup \{\gamma\})^\star \Sigma]$$

Este concepto es análogo al de *motif rígido* que suele usarse en biología computacional, como puede verse en [Rigoutsos+98a]. De acuerdo con esta definición, cualquier símbolo individual es un patrón, así como cualquier cadena que comience y termine con un símbolo del alfabeto y en medio exhiba una secuencia arbitraria de símbolos del alfabeto o huecos. El propósito de incluir un símbolo extra, que no pertenece al alfabeto de la muestra, es

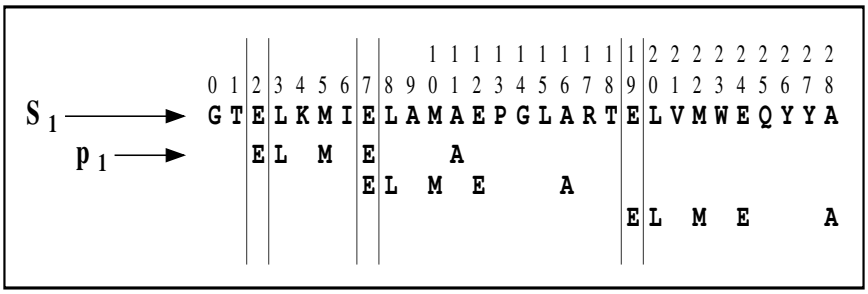

Figura 3.1: Ejemplo de patrón

denotar posiciones en las que puede aparecer cualquier símbolo del alfabeto. Así el hueco (*gap*) es un "comodín"[1] que puede reemplazarse por diferentes símbolos del alfabeto en las diferentes ocurrencias del patrón en la muestra. Los símbolos del alfabeto en el patrón serán llamados símbolos definidos o *sólidos*, constituyen la parte del patrón que permanece invariante en sus diferentes ocurrencias.

Si se utiliza el caracter "." cómo símbolo de hueco, entonces

$$p_1 = \texttt{EL.M.E...A}$$

es un patrón que aparece tres veces en la muestra

$$S_1 = \texttt{GTELKMIELAMAEPGLARTELVMWEQYYA}$$

tal cómo se muestra en la figura 3.1.

Existen diferentes cualidades asociadas con un patrón.

Definición 2 Asociadas a todo patrón p existen las siguientes características:

- Un entero no negativo llamado el *orden* del patrón, denotado como $o(p)$ y que es el número de símbolos sólidos en él. En el ejemplo $o(p_1) = 5$.
- Un entero no negativo llamado el *tamaño* del patrón, denotado como $|p|$ y que es el número total de símbolos sólidos y huecos contenidos en el patrón. El tamaño del patrón de ejemplo es $|p_1| = 10$.
- La *lista de ocurrencias* $L(p)$, que se refiere a las posiciones, listadas en orden creciente, en las que aparece el patrón en la muestra. La primera posición de la muestra es la cero. En el ejemplo la lista es $L(p_1) = \{2, 7, 19\}$ cómo se muestra en la figura 3.1.
- La *frecuencia* $f(p) \geq 0$ del patrón, es decir, el número de veces que aparece en la muestra. En el caso del ejemplo $f(p_1) = 3$.
- La *cobertura burda* $\mathrm{Cov}(p)$, que es el número formal total de símbolos de la muestra que son cubiertos por el patrón considerando todas sus ocurrencias. Esto no es más que el producto de la frecuencia y el orden del patrón. En el ejemplo $\mathrm{Cov}(p_1) = 5 \times 3 = 15$. Un concepto similar se presenta en [Klein00].

[1] Lo que en inglés se expresaría como *wildcard* o *don't care*.

- La *cobertura efectiva* $\mathrm{Cov}_{\mathrm{eff}}(p)$, que es el número total de símbolos de la muestra cubiertos al menos una vez por todas las ocurrencias del patrón. Es decir, la cobertura burda menos el número de símbolos que son cubiertos más de una vez por el patrón. En el ejemplo, dado que el símbolos en la posición 7 de la muestra S_1 es cubierto por dos diferentes ocurrencias del patrón, entonces la cobertura efectiva es: $\mathrm{Cov}_{\mathrm{eff}}(p_1) = 15 - 1 = 14$.

Para representar cada patrón en las implementaciones del método propuesto, se utilizó una representación compuesta de dos vectores:

- El *vector de contenido*, que denota cuáles son los símbolos sólidos del patrón. En el ejemplo $\mathbf{c}(p_1) = [E, L, M, E, A]$.
- El *vector de estructura*, que denota las distancias entre los símbolos sólidos en el patrón. En el ejemplo: $\mathbf{d}(p_1) = [1, 2, 2, 4]$.

El concepto de cobertura efectiva previamente definido, proporciona una medida del número de símbolos de la muestra que están contenidos en las diferentes ocurrencias del patrón, pero, cómo será evidente más adelante, esta puede no ser una medida conveniente cuando hay más patrones involucrados. Puede ocurrir, por ejemplo, que algunos símbolos cubiertos por un patrón ya estén cubiertos por algún otro que se traslapa con el primero en algunas de sus ocurrencias. Con esto en mente se define una nueva medida llamada *cobertura exclusiva*.

Definición 3 La *cobertura exclusiva* de un patrón q, respecto a una muestra S y un conjunto de patrones P, que se denotará cómo $\mathrm{Cov}_{\mathrm{exc}}(q, P)$, es el número de símbolos de S cubiertos por todas las ocurrencias de q y que no son cubiertos por ningún patrón en P.

3.2 Diccionario, tasa de compresión

Con estos conceptos en mente, es posible formular el objetivo general. Se desea un método de compresión basado en un código de dos partes:

1. Un diccionario de patrones frecuentes encontrados en la muestra de datos a comprimir.
2. La re-expresión de la muestra, reemplazando todas las ocurrencias de cada patrón en el diccionario, por una referencia a su posición en él.

De acuerdo con el principio MDL, es necesario encontrar el conjunto de patrones (diccionario) que minimiza simultáneamente la re-expresión del mensaje y el diccionario mismo.

Encontrar un diccionario en este contexto, significa implícitamente, que se está formulando una hipótesis acerca de la fuente que produjo la muestra. Se supone que la fuente no produce, realmente, símbolos individuales, sino que su "lenguaje natural", o mejor dicho, su alfabeto, está constituido, al menos, por los patrones encontrados en la muestra. Es decir,

se está suponiendo implícitamente que la fuente que produjo la muestra no se expresa (o no del todo) usando los símbolos atómicos del alfabeto obvio que se puede observar, sino con un alfabeto más sutil, constituido por *meta-símbolos* que se construyen acomodando símbolos atómicos en los patrones que encontramos. Ahora bien, encontrar no sólo un diccionario, sino el óptimo, significa, en consideración al principio MDL, haber encontrado la mejor hipótesis para la fuente que produjo la muestra.

Con el fin de poseer un criterio cuantitativo que permita evaluar diferentes hipótesis, se formula el concepto de *tasa de compresión*:

Definición 4 Dada una muestra finita S de símbolos en un alfabeto Σ y un diccionario D de patrones frecuentes encontrados en S. Sea $T(S, D)$ el tamaño en bits de la expresión del diccionario más el tamaño en bits de la re-expresión de la muestra S en términos de referencias al diccionario. La *tasa de compresión* se define cómo:

$$G(S, D) = 1 - \frac{T(S, D)}{|S|}$$

donde $|S|$ denota el tamaño en bits de la expresión original de la muestra.

Esta es una medida del tipo *Higher Is Better*[2] acotada, dado que $G(S, D) < 1$. Una tasa de compresión negativa significa expansión. Valores cercanos a cero representan tasas de compresión pobres.

Por supuesto, como se dice explícitamente en la definición, el tamaño de la muestra comprimida $T(S, D)$ tiene dos componentes, que corresponden respectivamente a las dos partes del código mencionado: el tamaño del diccionario y el tamaño de la muestra re-expresada. Una vez que se tiene un conjunto de patrones frecuentes Q que se considera un buen diccionario, se deben identificar aquellos símbolos de la muestra que no están contenidos en ningún patrón de Q. Estos símbolos constituyen por sí mismos un patrón q' que aparece sólo una vez en la muestra y que es tanto mejor cuanto más pequeño. Este será llamado el *patrón residual* o *relleno*. Por supuesto el patrón residual debe ser incorporado a Q para construir el diccionario $D = Q \cup \{q'\}$.

El tamaño del diccionario depende de:

- El número de meta-símbolos en él.
- El número de bits necesarios para codificar cada símbolo individual. Es posible distinguir dos casos diferentes.
  - Todos los símbolos utilizan un código de la misma longitud (código de bloque). En este caso el número de bits depende sólo de la cardinalidad del alfabeto $b = \lceil |\Sigma| \rceil$.

[2] El objeto evaluado es tanto mejor cuanto mayor sea el valor de la medida.

  - Cada símbolo posee un código cuya longitud está determinada en proporción inversa a su frecuencia (exclusivamente en el diccionario). Por supuesto en este caso, cada símbolo $s \in \Sigma$ tiene su propia longitud de código b_s. El valor medio de esta longitud puede aproximarse por el estimador de la entropía medido en el diccionario, suponiendo la fuente que produce símbolos en Σ. Es decir:

$$b = \bar{b}_s \approx \sum_{s \in \Sigma} f_{rel}(s)\ \log_2(1/f_{rel}(s))$$

    donde $f_{rel}(s)$ denota la frecuencia relativa del símbolo s.

- El número de bits necesarios para codificar cada distancia entre los símbolos sólidos de los meta-símbolos, es decir, el tamaño de las corridas de huecos. El valor óptimo para codificar cada una de estas corridas es: $g = \lceil \text{AvgGapSize}(D) \rceil + 1$, aplicando el mismo criterio que se expone en [Hamming80] a propósito de una codificación predictiva.

Así que el tamaño en bits del diccionario puede ser aproximado por:

$$\Delta(S, D) = \sum_{d \in D} \left[b\, o(d) + g\, (o(d) - 1) \right] \tag{3.1}$$

Por otra parte el tamaño en bits de la muestra re-expresada en términos de referencias al diccionario depende del número total de meta-símbolos en el diccionario y de el número total de referencias necesarias para re-expresar la muestra. Sea $r = \lceil \log_2 |D| \rceil$, entonces el tamaño en bits de la muestra re-expresada es:

$$\Gamma(S, D) = \sum_{d \in D} r\, f(d) \tag{3.2}$$

El tamaño total de la muestra comprimida $T(S, D)$ en la definición 4 es:

$$T(S, D) \approx \Gamma(S, D) + \Delta(S, D) \tag{3.3}$$

3.3 El proceso de compresión

El método de compresión que se propone en este trabajo, al que denotaremos PBDC por sus siglas en inglés (*Pattern-based Data Compression*), consiste de tres pasos generales que se harán explícitos un poco más adelante. Conviene, sin embargo, aclarar aquí que el primero de ellos, que consiste en encontrar un conjunto de patrones frecuentes óptimo (que maximice la tasa de compresión), puede ser dividido, a su vez, en dos fases

1. Encontrar un conjunto de patrones frecuentes que maximicen la tasa de compresión, tal cómo se expresa en la definición 4.

2. Re-expresar la muestra substituyendo las ocurrencias de cada patrón por una referencia al diccionario.

3. Codificación eficiente del diccionario. Es posible distinguir dos alternativas para ello:

   - Representación simple. En la que cada símbolo individual en Σ es representado con un código de bloque de longitud fija del mismo tamaño para todos los símbolos.
   - Representación basada en frecuencia. En la que cada símbolo individual es representado por una palabra de un código de longitud variable. La palabra es tanto más corta cuanto más frecuente es el símbolo asociado a ella. Una representación con códigos de Huffman es un buen ejemplo de esto.

Podría pensarse que la codificación del diccionario usando un código de longitud variable es siempre más corta; sin embargo esto no ocurre necesariamente. Hay que recordar que, en ese caso, se está utilizando, nuevamente, un código de dos partes: a la representación codificada de los símbolos de Σ habría que añadir el modelo estadístico de primer orden que permite reconstruir los símbolos originales, es decir, la tabla de frecuencias de Σ. La inclusión de este modelo, por breve que resulte, puede redundar en un costo conjunto superior al de la representación simple. En los programas que implementan el método que aquí se expone se calculan ambos costos y se elige la representación que resulta más corta, una bandera en el encabezado del archivo que contiene la muestra comprimida indica cuál de las alternativas fue elegida.

El primer paso del proceso de compresión, a saber, encontrar el conjunto óptimo de patrones frecuentes para constituir el diccionario, es con mucho, el más complejo de todos. Más adelante se procederá a analizar la complejidad de las tareas involucradas en él con todo detalle. Por el momento, en un primer acercamiento, lo que se requiere es encontrar patrones con la mayor frecuencia y el mayor orden posibles. Cuanto mayor sea el número de símbolos definidos en un patrón y mayor sea su frecuencia, será mejor amortizado el costo de incluirlo en el diccionario. Es prudente enfatizar que deben buscarse patrones cuyo orden y frecuencia sean, simultáneamente, lo mayor posibles, no basta con maximizar sólo una de estas características: en un extremo, el patrón con mayor orden es la muestra completa, un único patrón con frecuencia 1, con lo que no se obtiene compresión en absoluto; en el otro extremo, los patrones con mayor frecuencia son los símbolos de Σ, patrones de orden 1, con lo que nuevamente, la compresión no existe. Existe pues, un conflicto intrínseco entre estas dos características, se deben buscar entonces patrones con el mejor compromiso entre ambas. Es debido a esto que se definieron las coberturas burda, efectiva y exclusiva: maximizar la cobertura es, de hecho, maximizar ambas cualidades.

4

Complejidad del proceso

En el capítulo previo se expuso el proceso del método de compresión que se propone en este trabajo. En el primer paso de este proceso se requiere encontrar un conjunto de patrones que maximice la tasa de compresión cuando es adoptado como el diccionario de meta-símbolos. Se procederá ahora a analizar la complejidad de las tareas involucradas en la obtención de dicho conjunto óptimo.

Este primer paso puede subdividirse en dos:

1. Encontrar el conjunto de todos los patrones frecuentes en la muestra.

2. Encontrar el subconjunto de estos que maximiza la tasa de compresión.

También es posible pensar en no subdividir este primer paso, sino tratar de recabar evidencia que permita suponer que ciertos patrones son "buenos", al tiempo que son descubiertos, ignorando algunos otros menos prometedores. Esta alternativa ha sido explorada por otro de los integrantes del proyecto de investigación en el que está inscrito este trabajo [Kuri04b]. Comparativamente, a grandes rasgos, el subdividir el proceso en dos sub-tareas disminuye, a voluntad, el riesgo de ignorar patrones que, aunque en fases tempranas del proceso de búsqueda resultan tener un menor desempeño que otros, a la larga resultan ser mejores. Sin embargo el costo computacional de la subdivisión es considerablemente mayor, cómo se podrá apreciar más adelante.

4.1 El problema subyacente a la búsqueda de patrones

Típicamente los algoritmos de búsqueda de patrones están basados en el pre-procesamiento de la muestra donde serán buscados, almacenándola en una estructura de datos arborescente, ya sea un *suffix tree* o un *suffix trie* [Vilo02, Wang+99]. En cualquiera de ellas resulta muy sencillo buscar subcadenas frecuentes en la muestra, de hecho el algoritmo de descenso utilizado posee una complejidad de $O(k)$ donde k es la longitud de la subcadena buscada. La construcción de la estructura resulta también ser un proceso eficiente. Los algoritmos

como el de McCreight o Ukkonen, los mejores para el caso, poseen una complejidad de $O(|S|)$, donde S es la muestra a pre-procesar; los peores tienen complejidad polinomial.

El problema subyacente a la construcción de una estructura arborescente, como las mencionadas, consiste en encontrar la subcadena más larga que es común a un conjunto de cadenas. En el caso específico que nos ocupa, puede pensarse que el conjunto de cadenas en cuestión es el de todos los sufijos de la muestra ordenados lexicográficamente. Este problema posee solución polinomial. En esencia sólo hay que explorar el conjunto de cadenas (o un subconjunto de él en iteraciones posteriores) de izquierda a derecha, en posiciones sucesivas, hasta que no ocurra un empate entre todas las cadenas del subconjunto y aplicar recursivamente el proceso. La exploración siempre es, desde una posición, hacia la inmediata siguiente, dado que se buscan subcadenas de símbolos consecutivos.

Podría pensarse entonces en una generalización del proceso, no ya para buscar subcadenas, sino para buscar patrones cómo los definidos en este trabajo. El problema, sin embargo, deja entonces de ser simple. En principio habría que buscar el máximo apareamiento posible en todas las posiciones subsecuentes a partir de una inicial y luego recorrer todas las posibles posiciones iniciales. El crecimiento del espacio de búsqueda en este caso, ya no es lineal y se convierte en exponencial.

Es posible formular este problema en términos precisos como sigue.

Definición 5 Dado un alfabeto finito Σ y un conjunto finito R de cadenas en Σ^+. El problema de MaximumCommonPattern (MCP) consiste en encontrar el patrón (de acuerdo con la definición 1) de mayor orden que ocurre en toda cadena de R.

Se probará que MCP es un problema NP-completo (originalmente la demostración fue publicada en [Galaviz05a]). Para ello será necesario hacer una reducción, es decir, un mapeo polinomial de algún problema NP-completo conocido a MCP. Se ha elegido utilizar el problema del cubrimiento de vértices: VertexCover o VC. Un problema similar, en el que no es necesario que empaten los huecos (*gaps*), sino sólo los símbolos sólidos, es el llamado MaximumCommonSubsequence que ha sido ubicado también entre los NP-completos [Maier78].

Definición 6 Dado un grafo no-dirigido $G = (V, E)$, donde V es el conjunto de vértices y E el de aristas, el problema de VertexCover consiste en encontrar el subconjunto de vértices $V' \subseteq V$ mas pequeño tal que toda arista $e \in E$ tiene al menos un extremo en V'.

Se especificará ahora un algoritmo polinomial que mapea cualquier instancia de VC en una instancia de MCP.

Sea $G = (V, E)$ un grafo no-dirigido con vértices $V = \{v_1, \ldots, v_k\}$ y aristas $E = \{e_1, \ldots, e_r\}$, la entrada de una instancia cualquiera de VC. Se supondrá, sin pérdida de generalidad, que se ha definido un orden arbitrario sobre el conjunto de vértices, es decir, los vértices se han etiquetado de manera que $v_i < v_j$ si $i < j$. Cada arista $e = (v_i, v_j)$ puede entonces ser especificada de manera que $v_i < v_j$.

1. Se define $\Sigma = \{s_1, \ldots, s_k\}$ un conjunto arbitrario de $k = |V|$ símbolos y f una correspondencia uno a uno: $f(v_i) = s_i$.

2. Sea $C = s_1 s_2 \ldots s_k$, la cadena en la que todos los símbolos aparecen en orden creciente respecto a su vértice asociado.

3. Sea $e_i = (v_j, v_m)$ la i-ésima arista, con base en ella y en C se define la cadena:

$$c_i = s_1 \ldots s_{j-1} \# s_{j+1} \ldots s_k s_1 \ldots s_{m-1} \# s_{m+1} \ldots s_k$$

   es decir, una cadena idéntica a C concatenada consigo misma, salvo que reemplaza, en la primera C el símbolo asociado con el vértice j por un "#" y en la segunda C hace lo mismo con el símbolo asociado al vértice m, los extremos de la i-ésima arista. Se supone que "#" no pertenece a Σ.

El conjunto $R = \{C, c_1, \ldots, c_r\}$ de $r + 1$ cadenas será la entrada del problema MCP. La obtención de este conjunto se hace en tiempo polinomial, dado que su complejidad es, de hecho, $O(2rk)$.

Se puede proceder ahora a demostrar el siguiente resultado.

Teorema 1 MAXIMUMCOMMONPATTERN *es* NP*-completo .*

Dem: Se debe mostrar primero que MCP está en NP. Es decir, que es P-verificable. Dado un conjunto R de r cadenas de longitud k, en el que se sabe que hay un patrón común de orden t, máximo y dado un patrón P de orden t, es posible verificar si P es común a todas las cadenas haciendo $O(r(k - |P|)|P|)$ comparaciones.

Por otra parte, el algoritmo previamente descrito reduce cualquier instancia de VC a una instancia de de MCP y esta transformación se lleva a cabo en tiempo polinomial, dado que la generación de las $r + 1$ cadenas toma $O(2rk)$ pasos, donde r es el número de aristas y k el de vértices.

Ahora bien, si se posee una solución para una instancia de VC; esto es, un conjunto $V' = \{v'_1, v'_2, \ldots, v'_s\} \subseteq V$ tal que cualquier arista en E tiene un extremo en V', entonces el patrón de máximo orden, común a todas las cadenas, es obtenido reemplazando con "#" en C, cada símbolo asociado con los vértices en V'. Esto se hace en $O(k)$ reemplazos. □

En la figura 4.1 se muestra un grafo, en el que han sido enfatizados los vértices que constituyen la cubierta mínima, a saber: $V' = \{3, 5, 8, 9\}$. Se han asociado a los vértices símbolos arbitrarios del alfabeto. En la tabla 4.1 se muestran las cadenas que el algoritmo de reducción obtendría. El subíndice de cada cadena corresponde con la etiqueta de la arista asociada a ella y los encabezados de la columnas son las etiquetas de los vértices. Cómo puede comprobarse el patrón de orden máximo es: `AB#D#FG##J`. Que resulta de reemplazar en la cadena `ABCDEFGHIJ`, los símbolos asociados con los vértices V'.

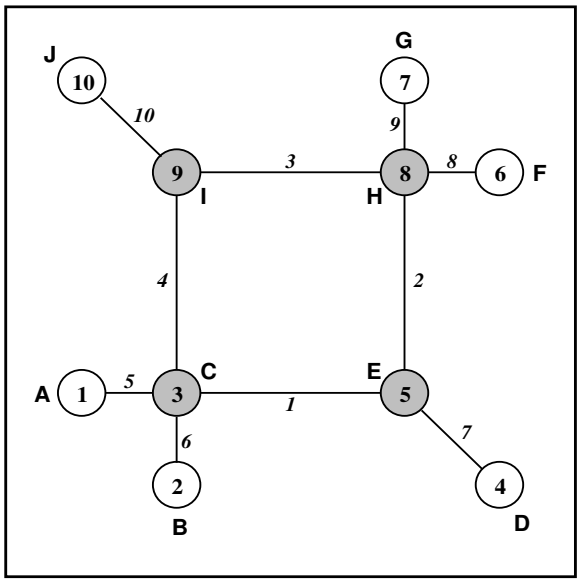

Figura 4.1: Grafo ejemplo para la reducción de VC a MCP.

Cad	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
S	A	B	C	D	E	F	G	H	I	J										
$\mathbf{S}_1$	A	B	#	D	E	F	G	H	I	J	A	B	C	D	#	F	G	H	I	J
$\mathbf{S}_2$	A	B	C	D	#	F	G	H	I	J	A	B	C	D	E	F	G	#	I	J
$\mathbf{S}_3$	A	B	C	D	E	F	G	#	I	J	A	B	C	D	E	F	G	H	#	J
$\mathbf{S}_4$	A	B	#	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	#	J
$\mathbf{S}_5$	#	B	C	D	E	F	G	H	I	J	A	B	#	D	E	F	G	H	I	J
$\mathbf{S}_6$	A	#	C	D	E	F	G	H	I	J	A	B	#	D	E	F	G	H	I	J
$\mathbf{S}_7$	A	B	C	#	E	F	G	H	I	J	A	B	C	D	#	F	G	H	I	J
$\mathbf{S}_8$	A	B	C	D	E	#	G	H	I	J	A	B	C	D	E	F	G	#	I	J
$\mathbf{S}_9$	A	B	C	D	E	F	#	H	I	J	A	B	C	D	E	F	G	#	I	J
$\mathbf{S}_{10}$	A	B	C	D	E	F	G	H	#	J	A	B	C	D	E	F	G	H	I	#

Tabla 4.1: Las cadenas asociadas con el ejemplo.

4.2 El subconjunto óptimo

Suponiendo que se ha podido salvar, de alguna manera, el problema de encontrar un conjunto de patrones frecuentes en la muestra que se pretende comprimir, aún hay que resolver el segundo de los problemas involucrados con el primer paso del proceso de compresión, a saber, encontrar el subconjunto de patrones que, de adoptarse como diccionario, maximiza la tasa de compresión.

A este lo hemos bautizado como el problema del *subconjunto de patrones óptimo* u OptimalPatternSubset (ops). Formalmente:

Definición 7 Dados:

- Una muestra finita de datos S, con $|S|$ símbolos.
- Un conjunto $P = \{p_1, p_2, \ldots, p_n\}$ de patrones de S con frecuencias $f(p_i) = f_i$ y ordenes $o(p_i) = o_i$.

el problema OptimalPatternSubset consiste en encontrar un subconjunto $D \subseteq P$ que maximiza la tasa de compresión G, de la definición 4, sujeta a la restricción:

$$\mathrm{Cov}(Q) \leq |S| \tag{4.1}$$

donde $\mathrm{Cov}(Q)$ denota la cobertura efectiva acumulada de todos los patrones en Q.

Este problema, al igual que el analizado anteriormente, también resulta intratable. Para hacer la demostración se recurrirá ahora al problema *de la mochila* (0,1) o $(0,1)$-KnapsackProblem ((0,1)-kp, véanse [Cormen01, Garey79]).

Definición 8 Dados:

- Un conjunto de objetos $I = \{i_1, i_2, \ldots, i_m\}$.
- Una función v que asigna a cada objeto i_j su *valor*: $v(i_j) > 0$.
- Una función w que asigna a cada objeto i_j su *peso*: $w(i_j) > 0$.
- Un entero positivo C llamado la *capacidad*.

el $(0,1)$-KnapsackProblem consiste en encontrar algún subconjunto $B \subseteq I$ tal que maximice:

$$\sum_{i_j \in B} v(i_j)$$

con la restricción:

$$\sum_{i_j \in B} w(i_j) \leq C$$

Teorema 2 OPTIMALPATTERNSUBSET *es* NP*-completo .*

Dem: Primero se debe probar que OPS está en NP, lo que significa que es verificable en tiempo polinomial. Dado un subconjunto $D \subseteq P$ y la máxima tasa de compresión γ, es posible calcular $G(S, D)$ (de acuerdo a la definición 4) en un tiempo que depende linealmente del tamaño de D (número de patrones en D) y verificar que coincide con el valor de γ.

Sea ahora I un conjunto arbitrario de objetos de una instancia cualquiera de $(0, 1)$-KP. Sean v y w las funciones de valor y peso respectivamente y C la capacidad. Finalmente sean b, g y r tres enteros positivos arbitrarios y $k = b + g$.

Más adelante será necesaria la siguiente cota inferior para la tasa de compresión, basada en la definición 4, tomando en consideración las expresiones 3.3, 3.1 y 3.2.

$$\begin{aligned} G(S,D) &= 1 - \frac{T(S,D)}{|S|} \\ &= 1 - \frac{\Gamma(S,D) + \Delta(S,D)}{|S|} \\ &= 1 - \frac{\sum_{d \in D} [b\, o(d) + g\,(o(d) - 1) + r\, f(d)]}{|S|} \\ &\geq 1 - \frac{\sum_{d \in D} [k\, o(d) + r\, f(d)]}{|S|} \end{aligned} \tag{4.2}$$

Nótese que maximizar esta cota implica, necesariamente, maximizar la tasa de compresión.

Se procederá ahora a establecer la reducción de $(0, 1)$-KP a OPS. El algoritmo para mapear, en tiempo polinomial, una instancia cualquiera de $(0, 1)$-KP a una de OPS, procede como sigue:

Para cada objeto $i_j \in B$ se calcula:

$$w'(i_j) = \begin{cases} \frac{(v(i_j)-C)^2}{4kr\, w(i_j)}, & \text{if } w(i_j) \geq \frac{(v(i_j)-C)^2}{4kr} \\ w(i_j), & \text{en otro caso} \end{cases}$$

Esto significa que:

$$4kr\, w'(i_j) < (v(i_j) - C)^2 \tag{4.3}$$

Ahora es posible establecer los valores para el tamaño de muestra, las frecuencias y los ordenes para la instancia respectiva de OPS:

$$C_B = \frac{C}{|B|} \tag{4.4}$$

$$f(i_j) = \frac{(C_B - v(i_j)) + \sqrt{(v(i_j) - C_B)^2 - 4kr\, w'(i_j)}}{2r} \tag{4.5}$$

$$o(i_j) = \frac{w'(i_j)}{f(i_j)} \tag{4.6}$$

donde (4.3) garantiza que la raíz cuadrada de (4.5) es una solución real de:

$$r\, f^2(i_j) + f(i_j)\,(v(i_j) - C_B) + k\, w'(i_j) = 0$$

De donde se obtiene:

$$v(i_j) = \frac{C_B\, f(i_j) - k\, w(i_j) - r\, f^2(i_j)}{f(i_j)}$$

Finalmente, usando (4.4) y (4.6):

$$v(i_j) = \frac{C}{|B|} - (k\, o(i_j) + r\, f(i_j))$$

De (4.6) se tiene:

$$w'(i_j) = f(i_j)\, o(i_j)$$

En la instancia de $(0,1)$-KP se requiere maximizar:

$$\begin{aligned} \sum_{i_j \in B} v(i_j) &= \sum_{i_j \in B} \left[\frac{C}{|B|} - (k\, o(i_j) + r\, f(i_j)) \right] \\ &= C - \sum_{i_j \in B} (k\, o(i_j) + r\, f(i_j)) \end{aligned}$$

Si se establece $|S| = C$ para OPS entonces, maximizar la última expresión también maximiza:

$$\frac{\sum_{i_j \in B} v(i_j)}{|S|} = 1 - \frac{\sum_{i_j \in B} (k\, o(i_j) + r\, f(i_j))}{|S|}$$

que es la cota de la tasa de compresión definida en (4.2).

La restricción se transforma entonces en:

$$\sum_{i_j \in B} w'(i_j) = \sum_{i_j \in B} f(i_j)\, o(i_j) \leq \sum_{i_j \in B} w(i_j) \leq C = |S|$$

En términos de OPS la restricción significa que la cobertura conjunta de los patrones en B no debe exceder el tamaño original de la muestra.

Así, la solución para el problema de la mochila, usando la transformación descrita, puede mapearse en una para el problema OPS en tiempo polinomial. □

Los resultados de este capítulo, prueban que los problemas subyacentes a las tareas que deben ser completadas en la compresión son intratables. Es por esto que, en el capítulo siguiente, se expondrán heurísticas que permitan aproximar la solución de estos problemas.

5

Aproximación de soluciones

Los resultados de complejidad exhibidos en el capítulo previo, muestran que implementar el método planteado, pretendiendo encontrar determinísticamente la mejor tasa de compresión posible, no resulta computacionalmente factible. Se debe entonces pensar en métodos de aproximación que permitan acercarse a las soluciones para los dos problemas enunciados, a saber: (a) encontrar un conjunto de patrones frecuentes y (b) encontrar un subconjunto de estos que maximice la tasa de compresión de ser adoptado como diccionario. Se procederá entonces a describir los algoritmos de aproximación utilizados.

5.1 Búsqueda de patrones frecuentes

Existen numerosos algoritmos para la búsqueda de patrones (*Pattern Discovery*), como puede apreciarse en [Brazma+97, Buhler+02, Vilo02, Wang+99]. La complejidad de estos es creciente conforme se imponen menos restricciones al tipo de patrones buscados. En particular, para el tipo de patrones que se han definido aquí: con un número desconocido de huecos entre los símbolos definidos, la complejidad de los algoritmos para encontrarlos es proporcional a 2^n donde n es la longitud de la muestra donde se buscan los patrones ([Vilo02], Págs. 531–54). Lo que, como se ha señalado, proviene esencialmente del hecho de que MaximumCommonPattern es NP-completo .

Si se imponen, sin embargo, restricciones adicionales sobre el tipo de patrones buscados, por ejemplo, acotando el tamaño, orden o frecuencia, es posible formular algoritmos eficientes (i.e. de complejidad polinomial). Probablemente uno de los más conocidos sea *Teiresias* [Floratos+98, Rigoutsos+98a, Rigoutsos+98b], desarrollado por un grupo de investigación de IBM.

Este algoritmo recibe, como parámetros de operación, tres valores: el orden mínimo requerido en los patrones que se buscan (μ), el tamaño máximo permitido de ellos (λ) y su frecuencia mínima (ϕ). Estos valores constituyen restricciones que reducen el dominio de búsqueda considerablemente, Teiresias sin embargo, impone una restricción adicional, a saber, que los patrones encontrados sean *maximales*, formalmente:

Definición 9 Un patrón p de una muestra S es un *patrón maximal* si no es posible encontrar en S un patrón más específico q, tal que cualquier aparición de p sea también una de q.

Por ejemplo, el patrón q =`KL..PK.D` es más específico que p =`KL...K`. Evidentemente cada aparición de q es también una aparición de p. Pero si además cada aparición de p es también una de q (i.e. nunca aparece p por sí solo, sin ser realmente q), entonces p no es maximal, porque puede hacerse más específico sin perder frecuencia.

Esto es particularmente interesante para el problema que nos ocupa porque entonces es posible reducir la búsqueda a patrones maximales en el sentido mencionado arriba. Si se considerara algún patrón no maximal para su inclusión en un diccionario se estaría desperdiciando espacio, sería más eficiente incluir, en su lugar, alguno de los patrones maximales en los que está "contenido" (abusando del lenguaje).

Si se denota con $|S|$ el tamaño de la muestra, la complejidad de Teiresias [Floratos+98] es:

$$O\left(\lambda^{\mu}\,|S|\log|S| + \lambda(|S|+t)\sum_{p\ maximal} o(p)\right)$$

donde t es el tiempo necesario para buscar un patrón en una tabla *hash* que contiene a todos los patrones maximales que cumplen con las restricciones de orden, tamaño y frecuencia, el mismo conjunto sobre el que se itera la suma.

La ventaja de un algoritmo como este es que pueden fijarse los parámetros de tal forma que el tiempo de ejecución sea pequeño, tanto como se quiera. A fin de cuentas cuanto mayor sean la frecuencia y el orden y menor el tamaño exigidos en los patrones habrá menos patrones que incluir entre los maximales, con lo que, en principio t será menor (menos colisiones) al igual que el factor definido por la suma (que tiene que ver con determinar si un patrón es maximal o no), lo que es aún más significativo. Es como si se pasaran los patrones por un cedazo, cuanto mayores sean los orificios en él, tanto menor será el número de patrones que se queden en el cedazo. Cómo la complejidad del algoritmo está dada en términos proporcionales al número de patrones en el cedazo, menor será ésta.

La desventaja es que al hacer más grandes los orificios del cedazo pueden perderse patrones que realmente podrían ser útiles. Bien podría ocurrir que una muestra fuera construida con varios patrones de frecuencia 5 y uno más de frecuencia 7. Si el parámetro de frecuencia mínima fuera fijado en 6, por ejemplo, entonces sólo se encontraría el patrón de frecuencia 7 mientras que los otros patrones constructores nunca serían descubiertos. Se está en presencia de un nuevo compromiso entre dos medidas de desempeño: la rapidez y la riqueza del conjunto de patrones. Cuanto menor es el riesgo de ignorar patrones útiles tanto más próxima a lo exponencial es la complejidad en tiempo del algoritmo.

Teiresias opera en dos fases [Rigoutsos+98a]: la de búsqueda de patrones elementales y la de convolución. Durante la primera se encuentran todos los patrones que tienen, al menos la frecuencia mínima especificada y el orden mínimo sin exceder el tamaño máximo. Luego, en la segunda fase, estos patrones son combinados para formar patrones de mayor orden, eliminando a todos aquellos que no son maximales en cuanto se les descubre.

5.2 Búsqueda de un subconjunto óptimo

Se ha mencionado ya la conveniencia de poseer patrones de orden y frecuencia tan grandes como sea posible, maximizar simultáneamente ambas cantidades amortiza mejor el costo de incluir el patrón en el diccionario. Con base en ello fue que se definieron los diferentes conceptos de cobertura de patrones. A partir de ellos es posible definir una estrategia heurística para aproximar un conjunto de patrones óptimo.

Si se denota con D_0 el subconjunto que aproxima al diccionario óptimo, con P al conjunto total de patrones frecuentes encontrados en la muestra, con $\mathrm{Cov}(D_0)$ a la cobertura conjunta de todos los patrones en D_0 y con S, como hasta ahora, a la muestra; el algoritmo que se propone para aproximar el diccionario óptimo es el siguiente:

```
CoverageBased(P)
1  D_0 ← ∅
2  repeat
3        d ← p  that maximizes  Cov_exc(p, P \ {p})
4        P ← P \ {d}
5        D_0 ← D_0 ∪ {d}
6  until (P = ∅) ∨ (Cov(D_0) ≥ |S|) ∨ (Cov_exc(p, D_0) = 0)
7  return  D_0
8  end
```

La estrategia *greedy* del algoritmo puede describirse en términos más intuitivos:

1. Establecer el subconjunto D_0 de patrones seleccionados como vacío.
2. Seleccionar el patrón $p \in P$ que cubra el mayor número posible de símbolos de la muestra que no hayan sido cubiertos por los elementos de D_0.
3. Incluir p en D_0
4. Eliminar de P a p, el patrón seleccionado.
5. Repetir los tres pasos previos hasta que:
   - P quede vació o bien
   - La cobertura alcanzada por D_0 sea el total de la muestra o bien
   - El número de símbolos de la muestra cubiertos por p y que no han sido previamente cubiertos por algún elemento de D_0 sea cero
6. D_0 es la aproximación propuesta para constituir el diccionario óptimo.

Como en toda heurística, encontrar la propuesta óptima no está garantizado, así que se hizo necesario acudir a algoritmos que mejoren la propuesta obtenida por este algoritmo.

5.3 Escaladores

Se ha visto que, en el caso de problemas de optimización combinatoria NP-completos, los esquemas de búsqueda local pueden mejorar considerablemente una propuesta inicial [Halldorsson95], en particular los métodos basados en el reemplazo de un elemento de una colección por otro.

Con esto en mente y teniendo en consideración el esquema de codificación elegido para la heurística anterior, se formularon dos escaladores (*hill climbers*) que operan sobre una propuesta inicial mejorándola a través de un proceso iterativo. El esquema de codificación mencionado es el más simple: se listan ordenadamente los patrones del conjunto total P de patrones frecuentes hallados en la muestra, se construye un vector booleano con $|P|$ entradas, un 1 en la i-ésima entrada significa que el patrón $P[i]$ es incluido en la propuesta de subconjunto óptimo, un 0 significa lo contrario. Por supuesto se parte del vector con 1's en las entradas que corresponden a los elementos de D_0, la propuesta encontrada por la heurística.

MSHC Escalador de Paso Mínimo (*Minimum Step Hill Climber*). Busca una mejor propuesta de entre aquellas que difieren en sólo un bit en el vector binario de la propuesta inicial. Es decir se incluye un patrón ausente de la propuesta o se elimina alguno que había sido incluido. Formalmente:

```
MSHC(V)
 1  V_new ← V
 2  for i ∈ {0, ..., |V| − 1} do
 3      V[i] ← Not(V[i])
 4      if  V is better than V_new then
 5          V_new ← V
 6      endif
 7      V[i] ← Not(V[i])
 8  endfor
 9  return  V_new
10  end
```

MRHC Escalador de Reemplazo Mínimo (*Minimum Replacement Hill Climber*). Busca una mejor propuesta de entre aquellas que reemplazan un patrón incluido por otro que no había sido incluido. Formalmente:

```
MRHC(V)
 1  V_new ← V
 2  for i ∈ {0, ..., |V| − 1} do
 3      if V[i] = 1 then
 4          V[i] ← 0
 5          for j ∈ {0, ..., |V| − 1} do
 6              if V[j] = 0 ∧ i ≠ j then
 7                  V[j] ← 1
 8                  if V is better than V_new then
 9                      V_new ← V
10                  endif
11                  V[j] ← 0
12              endif
13          endfor
14          V[i] ← 1
15      endif
16  endfor
17  return V_new
18  end
```

Ambos escaladores se iteran hasta que no haya más mejoras o hasta que se haya efectuado un cierto número de evaluaciones.

6

Resultados

6.1 Diseño de los experimentos

El método de compresión descrito en los capítulos anteriores (PBDC) fue probado con dos tipos de experimentos: con muestras construidas con patrones cuya estructura, frecuencia y contenido estaba decidido *a priori* y con muestras de archivos reales en diferentes formatos. Su desempeño fue comparado con algunos métodos de compresión populares, descritos anteriormente en este mismo documento: Huffman (HUFF), Huffman adaptable (AHUF), codificación aritmética (ARIT), Lempel-Ziv-Welch (LZW, que cómo se recordará es una variante de LZ78) y el compresor libre de Unix `gzip` (GZIP, variante de LZ77). La medida de desempeño utilizada fue siempre la tasa de compresión tal como aparece en la definición 4 (página 26).

El número de muestras sobre las que se probó fue variable. Se determinó como el mínimo necesario para proporcionar un intervalo de confianza del 95% alrededor de la media, con un tamaño no mayor del 5% alrededor de ella.

Los tamaños de las muestras a comprimir fueron de 128, 256, 384, 512, 1024 y 2048 bytes. Con estos tamaños basta para tener una perspectiva clara de la manera en que la tasa de compresión depende del tamaño de la muestra. Tamaños mayores no hubieran aportado información relevante y hubieran requerido tiempos de procesamiento que rápidamente se hubieran vuelo inmanejables[1]. Para la fase de búsqueda de patrones se utilizó una implementación propia del algoritmo mencionado en el capítulo previo. Los parámetros de este algoritmo se eligieron para capturar casi todos los patrones (lo que contribuye a explicar lo mencionado a propósito de la magnitud del tiempo de ejecución), es decir: frecuencia mínima de 2, orden mínimo de 2 y tamaño máximo del 50% del tamaño de la muestra.

Para explicar la construcción de las muestras hechas con patrones, se debe hablar un poco acerca del contexto del que surgió el algoritmo que las generó. Un problema de interés

[1]El algoritmo consume hasta 7 días para comprimir una sola muestra de 2 Kb. Como sabemos, esto se debe a la naturaleza intratable de los problemas subyacentes que se han analizado aquí. Esto además, significa que incrementar el poder de cómputo no retribuye sensiblemente, ni en una disminución del tiempo de procesamiento, ni en un incremento del tamaño de los problemas que se resuelven en un tiempo dado.

en geometría combinatoria es el de determinar la configuración, si existe, que deben adquirir un conjunto de figuras, llamadas *polyominoes*, para cubrir de la mejor manera posible una cuadrícula de un tamaño dado. Los polyominoes a acomodar están también hechos de un cierto número de cuadrados (n) del mismo tamaño de los que constituyen la cuadrícula. El valor de n le da nombre a las figuras, si cada figura está hecha de 5 cuadrados, por ejemplo, se denomina *pentómino*, mientras que si está hecha de seis se llama *hexómino*. El numero de formas en que pueden acomodarse cinco cuadrados para hacer una figura conexa es finito, 12 para ser exactos o 63 si se toman en cuenta reflexiones y rotaciones de los 12 básicos. Es posible pensar entonces que hay un "meta-alfabeto" de 63 pentóminos bidimensionales que se acomodan a manera de mosaico para cubrir la cuadrícula. Si luego cada fila de la cuadrícula se pone a continuación de la anterior se obtiene un segmento hecho de patrones originalmente bidimensionales que ahora han sido puestos linealmente y por tanto han quedado "inconexos" o con huecos entre los cuadrados que los constituían. En la figura 6.1 se muestra un ejemplo de esto.

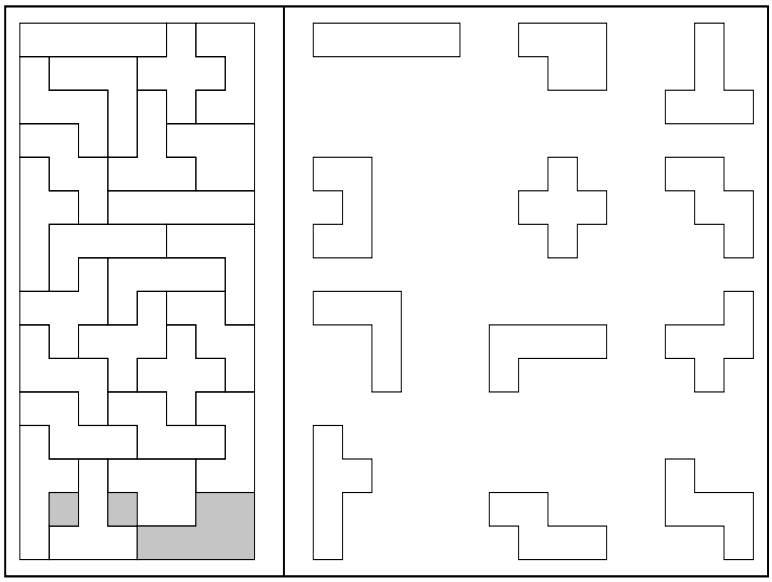

Figura 6.1: A la derecha se muestra un catálogo de 12 pentóminos que constituyen el "meta-alfabeto" con el que se ha construido el rectángulo de la izquierda (cuadrícula de $8 \times 16 = 64$ cuadrados). En gris los cuadrados que no se cubrieron con pentóminos.

Las muestras con patrones se construyeron utilizando un algoritmo de retroceso mínimo (*backtrack*), que obtiene las configuraciones de pentóminos que cubren un rectángulo de dimensiones dadas. Dado que se conocen el número, el orden y la estructura de los patrones involucrados en la construcción de las muestras, es posible calcular *a priori* cuál es la tasa de compresión óptima, que será etiquetada como "Teórico" en las gráficas.

Las muestras de archivos reales de los tres tipos mencionados (texto, ejecutables e

Tam	#	PBDC	±	DesvStd	AHUF	HUF	ARIT	LZW	GZIP	Teórico
128	12	0.340	0.0057	0.0100	0.118	0.070	0.254	0.150	0.001	0.297
256	40	0.506	0.0262	0.0844	0.104	0.080	0.342	0.260	0.202	0.586
384	63	0.601	0.0300	0.1216	0.181	0.163	0.426	0.336	0.311	0.680
512	66	0.680	0.0337	0.1397	0.319	0.314	0.513	0.420	0.430	0.729
1024	67	0.739	0.0357	0.1489	0.228	0.225	0.569	0.434	0.535	0.799
2048	68	0.764	0.0379	0.1596	0.241	0.241	0.679	0.500	0.704	0.836

Tabla 6.1: Comparación de los distintos métodos en muestras con patrones.

imágenes) se construyeron tomando fragmentos arbitrarios de archivos. Las de texto fueron generadas a partir de archivos de texto en inglés; para las ejecutables se tomaron programas en formato ELF (*Executable and Linkable Format*, formato estándar de archivo ejecutable en Unix) y para las muestras de imágenes se eligieron fragmentos de archivos en formato JPEG (*Joint Photograph Experts Group*, formato estándar para almacenamiento de imágenes). Para este último tipo de muestras se eligió JPEG por ser un formato que conlleva implícitamente una alta compresión con pérdida (*lossy*) de los datos, por lo que las muestras deberían ser casi incompresibles.

Los resultados mostrados aquí aparecen reportados parcialmente en [Kuri05b].

6.2 Muestras con patrones

Los resultados obtenidos para las muestras con patrones se muestran en la tabla 6.1. En la primera columna de (izquierda a derecha) aparece el tamaño de las muestras, en la segunda el número de muestras que fueron necesarias para obtener el intervalo de confianza indicado, en la tercera se muestra el valor medio de la tasa de compresión alcanzada por el algoritmo PBDC, en la cuarta se indica la amplitud del intervalo de confianza alrededor de la media de la columna anterior, la quinta columna muestra la desviación estándar y las siguientes cinco columnas muestran los valores medios de las tasas de compresión obtenidos por los diferentes algoritmos de compresión con quienes se comparó. Finalmente la columna del extremo derecho muestra el valor de la tasa de compresión teórica que era posible obtener si se encontraba exactamente el alfabeto de patrones con los que se construyó la muestra.

Cabe mencionar que hay una aparente discordancia en el resultado del primer renglón: la tasa de compresión de PBDC es superior a la teórica. Una inspección de lo ocurrido mostró que, en muestras pequeñas, es posible que accidentalmente surjan patrones que no forman parte del alfabeto y que, de hecho, "contienen" a alguno de ellos y resultan ser mejores que los patrones constructores. Este efecto desaparece en muestras mayores donde la probabilidad de estos "accidentes" es mucho menor. En la figura 6.2 se muestran los datos de la tabla graficados.

Cómo se puede observar, en muestras construidas con patrones el algoritmo con mejor tasa de compresión resulta ser PBDC.

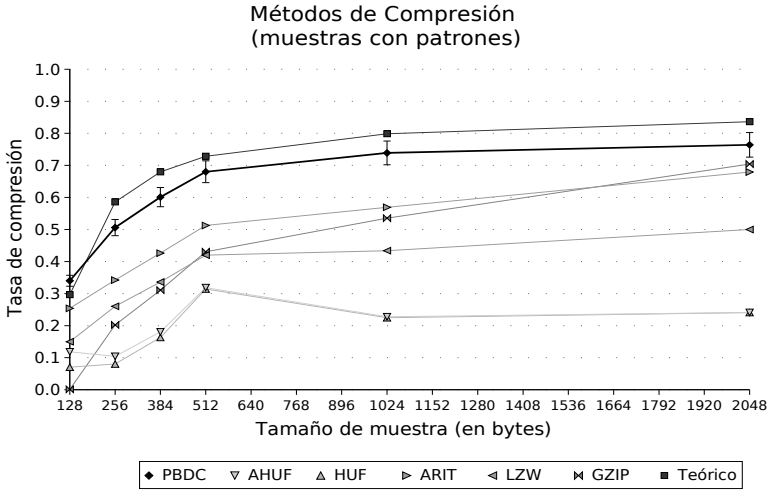

Figura 6.2: Comparación de las tasas de compresión promedio para los diferentes algoritmos en muestras construidas con patrones.

6.3 Muestras de archivos reales

En la tabla 6.2 se muestran los resultados de la comparación para muestras de archivos de texto y en la figura 6.3 la gráfica de los mismos. En la tabla 6.3 y la figura 6.4 se muestran los correspondientes a muestras de ejecutables, y en la tabla 6.4 y la figura 6.5 los de muestras de imágenes en formato JPEG.

Los datos experimentales indican que PBDC alcanza tasas de compresión competitivas. Ciertamente no son las mejores, pero hay que señalar que los algoritmos con los que se compara no enfrentan los problemas intratables que PBDC trata de resolver. En los archivos de imágenes, como era de esperarse, ninguno de los algoritmos utilizados obtiene tasas de compresión positivas. Los archivos JPEG carecen de redundancia explotable por cualquiera de ellos. El algoritmo de JPEG no sólo encuentra el modelo preciso que genera los datos, sino que lo recorta perdiendo precisión cuando se considera irrelevante.

Es de señalarse tambien que PBDC tiende a comportarse mejor conforme el tamaño de la muestra crece. Aparentemente, aún cuando el conjunto total de patrones crece exponencialmente con el tamaño de la muestra, lo que dificulta la labor de PBDC, la utilidad de los patrones identificados es redituable.

Tam	#	PBDC	±	DesvStd	AHUF	HUFF	ARIT	LZW	GZIP
128	56	0.04	0.0021	0.0082	0.28	0.14	0.25	0.09	0.00
256	60	0.21	0.0105	0.0417	0.36	0.30	0.35	0.20	0.24
384	55	0.27	0.0134	0.0509	0.39	0.34	0.38	0.25	0.31
512	38	0.34	0.0153	0.0480	0.40	0.36	0.40	0.26	0.36
1024	40	0.42	0.0149	0.0480	0.42	0.40	0.45	0.33	0.42
2048	46	0.45	0.0111	0.0385	0.43	0.42	0.50	0.38	0.47

Tabla 6.2: Comparación de los distintos métodos en muestras de texto.

Tam	#	PBDC	±	DesvStd	AHUF	HUFF	ARIT	LZW	GZIP
128	20	0.22	0.0061	0.0140	0.56	0.25	0.51	0.49	0.25
256	25	0.40	0.0187	0.0476	0.59	0.35	0.55	0.57	0.44
384	35	0.51	0.0195	0.0587	0.54	0.34	0.51	0.54	0.48
512	53	0.26	0.0129	0.0480	0.14	-0.06	0.30	0.24	0.22
1024	45	0.31	0.0157	0.0538	0.19	0.09	0.40	0.31	0.35
2048	48	0.63	0.0164	0.0578	0.48	0.44	0.61	0.59	0.65

Tabla 6.3: Comparación de los distintos métodos en muestras ejecutables.

Tam	#	PBDC	±	DesvStd	AHUF	HUFF	ARIT	LZW	GZIP
128	20	-0.46	0.0008	0.0019	-0.28	-1.00	-0.02	-0.01	-0.16
256	20	-0.53	0.0039	0.0089	-0.34	-0.72	-0.10	-0.06	-0.08
384	20	-0.33	0.0066	0.0151	-0.34	-0.55	-0.13	-0.12	-0.05
512	20	-0.28	0.0087	0.0199	-0.30	-0.42	-0.14	-0.14	-0.04
1024	20	-0.22	0.0118	0.0269	-0.20	-0.22	-0.17	-0.21	-0.02
2048	31	-0.18	0.0088	0.0249	-0.11	-0.11	-0.21	-0.29	-0.01

Tabla 6.4: Comparación de los distintos métodos en muestras de imágenes.

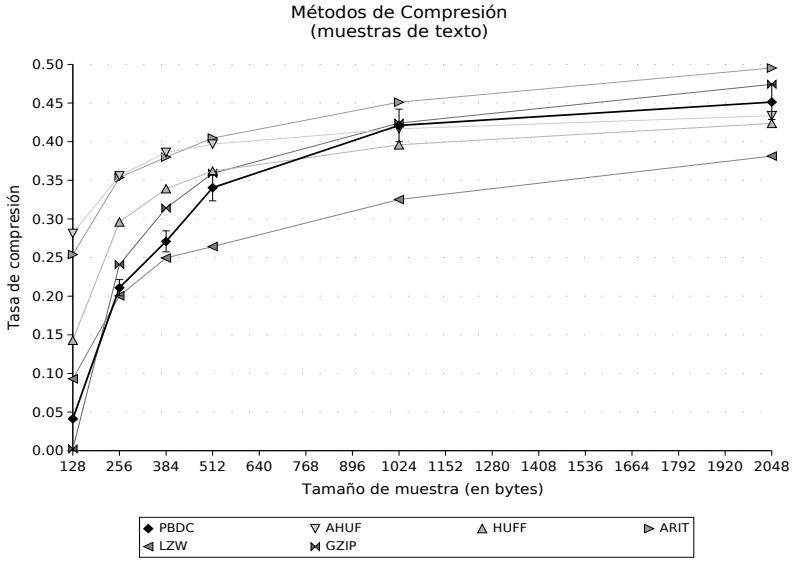

Figura 6.3: Comparación de las tasas de compresión promedio para los diferentes algoritmos en muestras de texto.

6.4 Síntesis y conclusiones

A lo largo de este trabajo se propuso un método de compresión basado en un diccionario. A diferencia de los métodos existentes, en los que el diccionario está integrado por cadenas de símbolos consecutivos que aparecen frecuentemente en la muestra a comprimir, en el método propuesto se generaliza el concepto de diccionario incluyendo en él patrones rígidos arbitrarios.

Haciendo referencia a la teoría de complejidad de Kolmogorov, a la teoría de la información de Shannon y especialmente al principio de descripción mínima de Rissanen (MDL), visto como puente entre las dos teorías anteriores, el método pretende encontrar el modelo (diccionario) que minimice la longitud de la descripción del diccionario mismo y de la muestra re-expresada en sus términos. El diccionario constituye así, la representación de la hipótesis más plausible para la fuente que produjo la muestra y supone, implícitamente, que el alfabeto de dicha fuente está constituido por meta-símbolos conformados por símbolos individuales acomodados en patrones.

Fueron abordados los problemas fundamentales del método propuesto y el análisis de ellos muestra que son computacionalmente intratables. Para aproximar la solución de uno

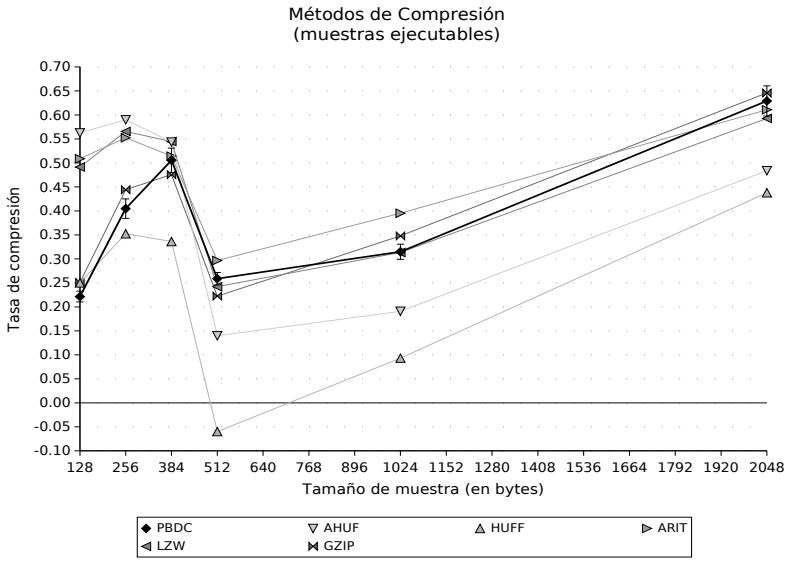

Figura 6.4: Comparación de las tasas de compresión promedio para los diferentes algoritmos en muestras ejecutables.

de ellos se propuso una heurística basada en las cualidades de los patrones encontrados y se introdujeron dos mecanismos de optimización local que refinan la solución propuesta por la heurística.

Los resultados experimentales muestran que, con muestras que se ajustan a las hipótesis del método propuesto, la tasa de compresión alcanzada por éste es superior a las de cualquiera de los otros métodos con los que fue comparado.

Por otra parte, dado que las hipótesis del método propuesto son, con mucho, menos exigentes que las que se suponen en los otros métodos, resulta plausible suponer que el método propuesto resulta potencialmente mejor que los que están actualmente en uso.

Sin embargo es de señalarse también que la naturaleza intratable de los problemas involucrados en el método, hace que éste posea tiempos de ejecución relativamente grandes comparado con los otros métodos. Existe un compromiso inherente entre el riesgo de no encontrar los mejores patrones o el mejor subconjunto de ellos y el tiempo invertido en la búsqueda de ellos. Esto no es sino una manifestación del hecho de que el cálculo de la complejidad de Kolmogorov es un problema no-computable.

Los resultados de muestras extraídas de archivos reales, parecen confirmar que el método propuesto obtiene tasas de compresión competitivas a pesar de que el espacio de búsqueda

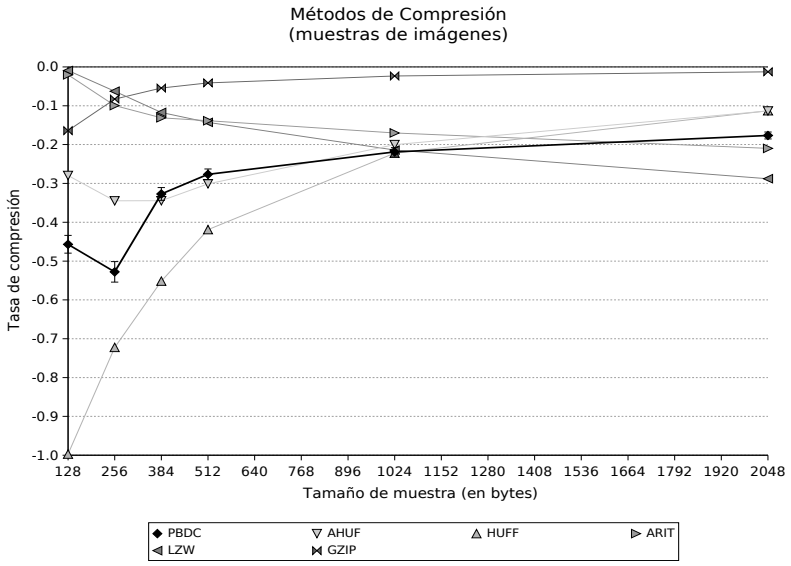

Figura 6.5: Comparación de las tasas de compresión promedio para los diferentes algoritmos en muestras de imágenes.

del diccionario óptimo crece, en general, un orden de magnitud respecto al tamaño de la muestra. Fenómeno del que no tienen que preocuparse los otros métodos.

6.5 Aportaciones

Las contribuciones más relevantes de este trabajo pueden sintetizarse como sigue:

- El análisis de la complejidad de dos problemas fundamentales para la realización del método propuesto: el problema de patrón común máximo (MaximumCommonPattern) y el de el subconjunto óptimo de patrones (OptimalPatternSubset). Se probó que ambos pertenecen a la categoría de los NP-completos.

- Luego de hacer un análisis acerca de las cualidades deseables de un patrón, se diseñó una heurística que aproxima el diccionario óptimo de patrones intentando optimizar simultáneamente dichas cualidades.

- Se diseñaron dos diferentes algoritmos de optimización local: el escalador de paso mínimo (*Minimum Step Hill Climber*) y el de reemplazo mínimo (*Minimum Replace*

Hill Climber). La acción encadenada de ellos mejora de manera significativa la aproximación obtenida por la heurística mencionada. En los experimentos mostrados aquí los escaladores mejoran hasta en un factor de 7.23 la tasa de compresión alcanzada con la heurística.

- El método de compresión propuesto constituye, de hecho, una manera práctica de aproximar la complejidad de Kolmogorov de una muestra de datos, cómo se ha mostrado en [Kuri05c]. Equivalentemente, puede considerarse como un mecanismo útil para determinar el modelo óptimo de una fuente que genera los datos [Kuri05d].

Pero quizás la aportación más relevante no es tan puntual como las ya descritas. A lo largo de este trabajo no sólo se ha expuesto un método de compresión, se han expuesto realmente los problemas fundamentales subyacentes a la compresión de datos en general. Los problemas que, de hecho, constituyen un puente entre los dos marcos teóricos en los que se encuadra la compresión. Podríamos decir que el método de compresión aquí planteado ha sido sólo el medio por el que se han desentrañado algunos de los problemas más profundos del área. Así que los resultados de complejidad no sólo exhiben la dificultad de los problemas que el método basado en patrones debió enfrentar, además hacen claro que todos los métodos de compresión los simplifican de una u otra manera, definen un modelo que, *de facto*, limita la capacidad de compresión, pero impone restricciones que permiten evadir las dificultades de los problemas hasta simplificarlos lo suficiente para hacerlos manejables. El éxito de los actuales métodos de compresión radica en encontrar alguno de los posibles equilibrios correctos entre capacidad de compresión y simplicidad de los problemas una vez que han sido impuestas las restricciones. Esta es la enseñanza fundamental del presente trabajo.

A la luz de lo expuesto en el párrafo anterior, resulta también plausible afirmar que los métodos que sean formulados en el futuro, y que superen las tasas de compresión de los actuales, indudablemente estarán basados en estrategias heurísticas que les permitan no explorar exhaustivamente el espacio de posibles modelos de la muestra, lo que indudablemente los remitiría a lidiar frontalmente con los problemas intratables, en el mejor de los casos, o no computables en el peor, que han sido expuestos en este trabajo.

6.6 Trabajo futuro

En los primeros ensayos del método propuesto se utilizaron algoritmos genéticos para resolver el problema de encontrar el subconjunto óptimo. El descubrimiento de la heurística basada en cobertura nos llevó a abandonar el uso de estas meta-heurísticas, por resultar relativamente más costosas en términos computacionales, que la heurística. Sin embargo las pruebas preliminares se hicieron en muestras de tamaño máximo de 256 bytes y construidas con patrones. A la luz de los resultados mostrados, resulta recomendable hacer una re-evaluación del compromiso costo-beneficio de los algoritmos genéticos u otro tipo de meta-heurísticas. El tamaño del espacio de búsqueda en las muestras no construidas con patrones resulta mucho mayor que en las muestras construidas a propósito. Probablemente un AG

con la población inicial sembrada usando la heurística de cobertura y los resultados del proceso de optimización local podría dar mejores resultados que los alcanzados hasta ahora [Kuri05d].

Es importante hacer notar que usando algoritmos genéticos es posible abreviar el proceso de compresión. Los AGs pueden usarse no sólo para aproximar el subconjunto óptimo una vez que se han encontrado los patrones frecuentes, de hecho, como se demostró en [Kuri04b, Kuri05d], es posible usarlos para encontrar dicho subconjunto al mismo tiempo que se encuentran patrones frecuentes. Aparentemente esto incrementa el riesgo de descartar prematuramente patrones prometedores, sin embargo aumenta notoriamente la velocidad a la que pueden procesarse muestras de tamaño mayor al usado aquí. Es necesario hacer una comparación cuidadosa de ambas alternativas, que considere no sólo la tasa de compresión cómo criterio de contraste, sino también el compromiso entre ésta y la velocidad.

Actualmente se está haciendo uso del método descrito en este trabajo en el análisis de secuencias proteicas del hongo Saccharomyces cerevisiae [Kuri05a, Kuri06]. Los resultados preliminares muestran que el método resulta de gran utilidad en este análisis, encontrando diccionarios de patrones que señalan características estructurales comunes a ciertos grupos de proteínas en los que ya se han encontrado características funcionales comunes. Es interesante obtener resultados de este tipo dado que la entropía estimada del conjunto de proteínas es cercana a la máxima posible, dada la cardinalidad de su alfabeto (20 aminoácidos) [Nevill+99]. Lo que lleva a pensar que no hay patrones que contengan información relevante por sí mismos fuera del contexto en que aparecen.

Referencias

[Brazma+97] Brazma, A., I. Jonassen, I. Eidhammer and D. Gilbert, "Approaches to the Automatic Discovery of Patterns in Biosequences", *Journal of Computational Biology*, Vol. 5, No. 2, 1997, pp. 277-304.

[Buhler+02] Buhler, J. y M. Tompa, "Finding Motifs Using Random Projections", *Journal of Computational Biology*, Vol. 9, No. 2, 2002, pp. 225-242.

[Burrows94] Burrows, M. y D.J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm", *Research Report 124*, Digital Systems Research Center, mayo de 1994.

[Chaitin66] G.J. Chaitin, "On the Length of Programs for Computing Finite Binary Sequences", *Journal of the ACM*, Vol. 13, No. 4, 1966, pp. 547–569.

[Cleary94] Cleary, J. G. y I. H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching", *IEEE Transactions on Communications*, Vol. 32, No. 4, 1984, pp. 396–402.

[Cormen01] Cormen, T.H., C.E. Leiserson, R.L. Rivest y C. Stein, *Introduction to Algorithms*, 2a Ed., MIT Press, 2001.

[Cormack86] Cormack, G. and R. N. Horspool, "Data Compression Using Dynamic Markov Modeling", *The Computer Journal*, Vol. 30, No. 6, 1986. pp. 541-550.

[Elias55] Elias, P. "Predictive Coding", *IRE Transactions on Information Theory*, Vol. 1, 1955, pp. 16–33.

[Einarsson91] Einarsson, G., "An Improved Implementation of Predictive Coding Compression", *IEEE Transactions on Communications*, Vol. 39, No. 2, febrero 1991, pp. 169–171.

[Floratos+98] Floratos, A. and I. Rigoutsos, "On The Time Complexity Of The Teiresias Algorithm", *IBM Research Report RC 21161*, april, 1998.

[Galaviz05a] Galaviz, J. y A. Kuri, "Discovery of Maximum Common Pattern in a Set of Strings", *Reportes de Investigación*, Depto. de Matemáticas, Facultad de Ciencias, UNAM, No. 2–05, abril 2005.

[Gallager78] Gallager, R., "Variations on a Theme by Huffman", *IEEE Transactions on Information Theory*, Vol. 24, No. 6, noviembre 1978, pp. 668–674.

[Garey79] Garey, M.R. y D.S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979.

[Grünwald03] P.D. Grünwald and P.M.B. Vitányi, "Kolmogorov Complexity and Information Theory. With an interpretation in terms of questions and answers", *Journal of Logic, Language, and Information*, Vol. 12, No. 4, pp. 497–529, 2003.

[Halldorsson95] Halldorsson, M., "Approximating Discrete Collections via Local Improvements", *Proceedings of 6th SIAM/ACM Symposium on Discrete Algorithms (SODA '95)*, 1995. pp. 160–169.

[Hamming80] Hamming, R.W., *Coding and Information Theory*, Prentice-Hall, 1980.

[Huffman52] Huffman, D. "A Method for the Construction of Minimum Redundancy Codes", *Proceedings of IRE*, Vol. 40, No. 9, 1952, pp. 1098-1101.

[Jagmohan02] Jagmohan, A., A. Sehgal y N. Ahuja, "Predictive encoding using coset codes", *Proceedings of IEEE International Conference on Image Processing 2002*, 2002, pp. 29–32 (Vol. 2).

[Klein00] Klein, Shmuel T., "Improving Static Compression Schemes by Alphabet Extension", *Proceedings of Combinatorial Pattern Matching, CPM 2000*, Springer Verlag, LNCS 1848, 2000, pp. 210–221.

[Kolmogorov65] A.N. Kolmogorov, "Three approaches to the quantitative definition of information", *Problems on Information Transmission*, Vol. 1, No. 1, 1965, pp. 1–7.

[Kuri04a] Kuri, A. y J. Galaviz, "Pattern-based Data Compression", en (Monroy, R., editor) *MICAI 2004: Advances in Artificial Intelligence*, Springer Verlag, LNAI 2972, 2004, pp. 1–10, 2004.

[Kuri04b] Kuri, A. y O. Herrera, "Efficient Lossless Data Compression using Advanced Search Genetic Operators and Genetic Algorithms", en (Figueroa Nazuno et al., editores) *Advances in Artificial Intelligence*, Research on Computer Science, Vol. 16, 2004, pp. 243–251.

[Kuri05a] Kuri, A. y M. Ortíz-Posadas, "A New Approach to Sequence Representation of Proteins in Bioinformatics", en *MICAI 2005, Advances in Artificial Intelligence*, LNAI 3789, 2005, pp. 880–889.

[Kuri05b] Kuri, A. y J. Galaviz, "Data Compression using a Dictionary of Patterns", *Research on Computing Science*, Vol. 16, 2005, pp. 173–182.

[Kuri05c] Kuri, A., O. Herrera, J. Galaviz y M. Ortiz-Posadas, "Practical Estimation of Kolmogorov Complexity using Highly efficient Compression Algorithms", *Research on Computing Science*, Vol. 16, 2005, pp. 193–202.

[Kuri05d] Kuri, A. J. Galaviz y O. Herrera, "Pattern-based Data Compression and the MDL Principle", enviado a *Information and Computation*, Elsevier, octubre de 2005.

[Kuri06] Kuri, A., A. Escalante y G. Coello, "Soft Computing unbiased Protein Classification in Saccharomyces cerevisiae", *WSEAS Transactions on Biology and Biomedicine*, No. 1, Vol. 3, enero 2006, pp. 21–26.

[Langdon84] Langdon, G.G., "An Introduction to Arithmetic Coding", *IBM Journal of Research and Development*, Vol. 28, No. 2, marzo de 1984, pp. 135–149.

[Li+97] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and its applications*, 2nd. edition, Springer Verlag, Graduate texts in Computer Science, 1997.

[Maier78] Maier, David, "The Complexity of Some Problems on Subsequences and Supersequences", *Journal of the ACM*, Vol. 25, No. 2, abril 1978, pp. 322-336.

[Moffat90] Moffat, A., "Implementing the PPM Data Compression Scheme", *IEEE Transactions on Communications*, Vol. 38, No. 11, 1990, pp. 1917–1921.

[Moffat+98] Moffat, A., R.M. Neal y I.H. Witten, "Arithmetic Coding Revisited", *ACM Transactions on Information Systems*, Vol. 16, No. 3, julio de 1998, pp. 256–294.

[Nelson96b] Nelson, M., "Data Compression with the Borrows Wheeler Transform", *Dr. Dobb's Journal*, septiembre de 1996.

[Nelson+96b] Nelson, M. y J. Gailly, *The Data Compression Book*, 2a ed., M & T Books, 1996.

[Nevill+99] Nevill–Manning C.G. y I.H. Witten, "Protein is incompressible", en (Storer J.A. y M. Cohn, editores) *Proceedings of IEEE Data Compression Conference*, pp. 257–266, 1999.

[Rigoutsos+98a] Rigoutsos, I. and A. Floratos, "Combinatorial Pattern Discovery in biological sequences: The Teiresias Algorithm", Bioinformatics, Vol. 14, No. 1, 1998, pp. 55–67.

[Rigoutsos+98b] Rigoutsos, I. and A. Floratos, "Motif Discovery Without Alignment Or Enumeration", *Proceedings of the 2nd annual Conference on Computational Molecular Biology (RECOMB'98)*, ACM Press, 1998, pp. 221–227.

[Rissanen76] Rissanen, J., "Generalized Kraft Inequality and Arithmetic Coding", *IBM Journal of Research and Development*, Vol. 20, mayo 1976, pp. 198–203.

[Rissanen78] J. Rissanen, "Modeling by shortest data description", *Automatica*, Vol. 14, 1978, pp. 465–471.

[Rissanen+79] Rissanen, J. y G.G. Langdon "Arithmetic Coding", *IBM Journal of Research and Development*, Vol, 23, No. 2, marzo 1979, pp. 149–162.

[Roman97] Roman, Steven, *Introduction to Coding and Information Theory*, Springer Verlag, 1997.

[Schmidhuber92] Shmidhuber, J., "Learning Unambiguous Reduced Sequence Descriptions", en *Advances in Neural Information Processing Systems*, Morgan Kaufmann, 1992, pp. 291–298.

[Scmidhuber96] Schmidhuber, J., "Sequential Neural Text Compression", *IEEE Transactions on Neural Networks*, Vol. 7, No. 1, 1996, pp. 142–146.

[Shannon48] Shannon, Claude E., "A Mathematical Theory of Communication", *The Bell System Technical Journal*, 1948, Vol. 27, julio: pp. 379-423, octubre: pp. 623-656.

[Salomon00] Salomon, D. *Data Compression, A Complete Reference*, 2a ed., Springer Verlag, 2000.

[Solomonoff64] R.J. Solomonoff, "A formal theory of inductive inference, part 1 and 2", *Information and Control*, 1964, Vol. 7, pp. 1–22 y 224–254.

[Storer82] Storer, James y Thomas Szymanski, "Data Compression via Textual Substitution", *Journal of the ACM*, Vol. 29, No. 4, october 1982, pp. 928–951.

[Verma+99] Verma, B.K., M. Blumenstein y S. Kulkarni, "A New Compression Technique Using an Artificial Neural Network", *Journal of Intelligent Systems*, Vol. 9, No. 1, 1999, pp. 39–53.

[Vitter87] Vitter, J.S, "Design and Analysis of Dynamic Huffman Codes", *Journal of the ACM*, Vol. 34, No. 4, 1987, pp. 825–845.

[Welch84] Welch, T.A., "A Technique for High-Performance Data Compression", *IEEE Computer*, Vol. 17, No. 6, junio de 1984, pp. 8-19.

[Vilo02] Vilo, Jaak, *Pattern Discovery from Biosequences*, PhD Thesis, Technical Report A-2002-3, Department of Computer Science, University of Helsinki, 2002.

[Wang+99] Wang, Jason T.L. *et al.*,"Pattern Discovery and Classification in Biosequences", en Wang, Shapiro y Shasha (editores), *Pattern Discovery in Biomolecular Data, tools, techniques, and applications*, Oxford University Press, 1999, pp. 55–74.

[Witten+87] Witten, I.H. R.M. Neal y J.G. Cleary, "Arithmetic Coding for Data Compression", *Communications of the ACM*, Vol. 30, No. 6, junio de 1987, pp. 520–540.

[Yu96] Yu, T. L., "Dynamic Markov Compression", *Dr. Dobb's Journal*, enero de 1996. pp. 30–32.

[Ziv+77] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, Vol. 23, No. 3, 1977, pp. 337–343.

[Ziv+78] J. Ziv and A. Lempel, "Compression of individual Sequences via Variable-Rate Coding", *IEEE Transactions on Information Theory*, Vol. 24, No. 5, 1978, pp. 530–536.