Universidad Católica
"Nuestra Señora de la Asunción"

Facultad de Ciencias y Tecnologías

Ingeniería Informática

Trabajo Práctico de **TAI 2**

"TÉCNICAS DE COMPRESIÓN"

Alumnos:

Acuña, Pablo Mat. No. 42631
Basualdo, Hugo Mat. No. 37889

Profesor: Ing. Juan E. Urraza

Asunción – Paraguay

2/10/2003

Introducción

En este trabajo de investigación sobre técnicas de compresión hemos tratado de abarcar solo una pequeña parte de los diferentes algoritmos existentes en la actualidad. No pretendemos entrar en muchos detalles, aunque si introduciremos una pequeña referencia teórica de teoría de la información y códigos, que es el fundamento matemático de todas las técnicas de compresión y codificación en general. Aún en esta pequeña referencia omitiremos una serie de consideraciones teóricas del tipo demostrativo y en algunos casos nos guiaremos por la "fe" en que ya están probados y demostrados en su totalidad, y en otros casos nos guiaremos de manera intuitiva.

Después de la introducción teórica, que esperamos no sea tan aburrida, empezamos a citar y explicar de manera simple técnicas de compresión sin pérdida de información tan antiguas como el de Huffman hasta una técnica que promete ser la que revolucionaría la codificación de imágenes y de la que ya existen implementaciones a la fecha: la compresión fractal.

A continuación empezamos con un poco de teoría.

Algunas nociones teóricas

Para la implementación de varios de los algoritmos de compresión se tienen en cuenta una serie de consideraciones matemáticas, en especial en las técnicas de compresión sin pérdida de información. Aunque pueda parecer un poco tediosa una introducción matemática, en este trabajo trataremos de minimizar la complejidad y no entrar en detalles muy profundos cuando esto implique entrar en consideraciones del fundamento de algunas definiciones y/o fórmulas. Cuando este sea el caso, solo comentaremos a grandes rasgos la característica que deseemos definir o mostrar.

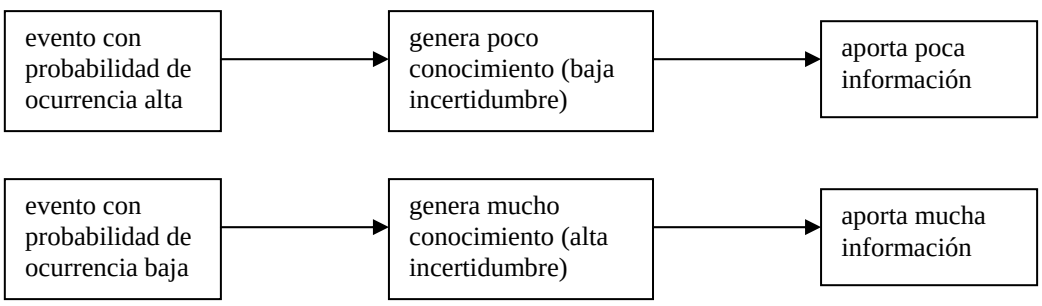
Empecemos con las definiciones y veremos un poquito de Teoría de la Información.

El primer concepto con el que debemos comenzar es el de **información.** Este es difícil de definir formalmente, aunque intuitivamente cualquiera sepa lo que significa.

Se puede considerar **información** como aquello susceptible de suministrar conocimiento, reduciendo el desconocimiento o incertidumbre en el receptor. Se debe distinguir entre datos e información. La transmisión de datos se puede considerar un proceso mecánico, que no tiene porque tener ligado la transmisión de información. Como ejemplo supóngase una fuente que siempre emita un mismo símbolo. Obviamente hay transmisión de datos puesto que quien observe la fuente recibirá un símbolo. Pero no existirá transmisión de información puesto que la recepción del símbolo no aumenta el conocimiento del receptor que estaba bien seguro de qué símbolo iba a recibir.

El problema surge cuando se pretende cuantificar de una forma objetiva la información, puesto que es muy difícil medir la cantidad de conocimiento que un mensaje genera en un receptor. La teoría matemática de la información está centrada sobre las señales solamente y su contenido de información, abstrayéndose de todos los usos humanos específicos. La expresión que se persigue para medir la información debe tener las siguientes propiedades:

- Debe ser proporcional al conocimiento que genera o, lo que es lo mismo, a la incertidumbre que disipa, de manera que:



- Tiene que resultar en valores nulos o positivos.
- Debe poseer una métrica lineal, de manera que la información proporcionada por dos mensajes sea igual a la suma de la información suministrada por cada uno de ellos

$$I(A + B) = I(A) + I(B)$$

siempre y cuando A y B sean independientes.

Y esta función que describa de forma matemática la información existe! Shannon propuso una fórmula para medir la cantidad de información *I* que proporciona un suceso en función de la probabilidad del suceso. Para un suceso *a* se tiene

$$I(a) = \log_b\left(\frac{1}{p(a)}\right) = -\log_b\ p(a)$$

donde *p(a)* es la probabilidad del suceso *a*. Si la base *b* del logaritmo es 2, la información es medida en *bits* (si no se pone la base, se supondrá base 2) . Si la base es 10, la información es medida en *dit*s. Si la base es el número *e* (logaritmos neperianos), la información es medida es *nats* (natural bits). El uso de la palabra bit puede llevar a confusión pues se utiliza como unidad de información y también como medida de los elementos de un mensaje (cuando éste es descrito mediante los símbolos 0 y 1). Así, un bit dentro de un mensaje puede llevar asociada una información mayor o menor que un bit de información.

La probabilidad *p* es la que determina cuanto de seguro o cierto hay en el suceso, es decir, el grado de incertidumbre, siendo más incierto cuanto menor sea su probabilidad.

Como ejemplo, la existencia de un tornado proporciona mucha más información que la de un nuevo amanecer, puesto que el amanecer era un suceso cierto que teníamos previsto, mientras que el tornado era un suceso incierto cuya ocurrencia no estaba contemplada, y por lo tanto nos aporta mucha más información. De hecho, la inexistencia de un amanecer aporta mucha más información que su existencia. La existencia del amanecer sólo confirma lo que estaba previsto. De esta forma, se observa cómo la cantidad de información de un suceso está íntimamente ligado a su grado de incertidumbre.

Aclarada esta parte con este último ejemplo, a continuación definiremos tres términos de manera menos formal que se manejan la teoría de la Información.

Fuente de información: es una entidad que genera un flujo de símbolos.

Alfabeto: Es el conjunto de todos los símbolos que puede generar una fuente de información

Lenguaje: Es el conjunto de todas las secuencias de símbolos permitidas.

Una fuente de información se puede modelar mediante una variable aleatoria discreta *X* que genera símbolos x_i. Cuando la aparición de los símbolos de una fuente son estadísticamente independientes, la fuente se denomina *sin memori*a, y a cada símbolo x_i del alfabeto de la fuente se le asocia una probabilidad de ocurrencia $p(x_i)$. El conjunto de estas probabilidades forman la distribución de probabilidades *p(x)* de forma que

$$\sum_i p(x_i) = 1$$

Entropía de la información

A partir del concepto anterior podemos definir la ***entropía de una fuente*** como el valor medio de la información proporcionada por la fuente, y se calcula como la esperanza matemática de la información de cada uno de los posibles valores que puede tomar X. Para las fuentes sin memoria el valor de la entropía denotado H(X) viene dado por la formula siguiente, conocida también como el *primer teorema de Shannon*:

$$H(X) = E[I(X = x_i)] = \sum_{x_i \in X} p(x_i)\log\left(\frac{1}{p(x_i)}\right)$$

Este valor es independiente de la representación utilizada para la información, depende de la aleatoriedad de la fuente, o sea que representa el grado de “desorden” de una fuente de datos.

Si todos los elementos del alfabeto de la fuente son equiprobables la formula se simplifica y *H(X)* = log(n), siendo *n* el número de elementos del alfabeto.

Ahora estamos en condiciones de continuar con los conceptos de codificación.

Codificación: Es el establecimiento de una correspondencia entre dos alfabetos no necesariamente distintos.

Puede considerarse como una función que asocia un símbolo o conjunto de símbolos del *alfabeto código* a cada símbolo del alfabeto de fuente (original). Matemáticamente se expresa como:

C: A → B

siendo C la función de codificación, A el conjunto de símbolos del alfabeto fuente y B el conjunto de símbolos del alfabeto código.

Como entrada al proceso de codificación podemos encontrar tanto símbolos de fuente aislados como agrupaciones de los mismos, formando los denominados ***mensajes***. Cada secuencia resultante de este proceso de codificación se le denomina ***palabra código***. La función inversa a la codificación es la ***decodificación***, o sea, revierte el proceso de codificación de la fuente y obtiene el mensaje original a partir del mensaje codificado. Matemáticamente se expresa como:

$$C^{-1}: B \rightarrow A$$

siendo C^{-1} la función de decodificación, B el conjunto de símbolos del alfabeto código y A el conjunto de símbolos del alfabeto original.

Parece obvio que para garantizar la reversibilidad del código, éste debe mantener una correspondencia uno a uno entre los símbolos de la fuente o mensajes y las palabras código. Cumpliendo este requisito se pueden contemplar dos alternativas: códigos de longitud fija y códigos de longitud variable. Suponiendo una fuente con alfabeto {A,B,C,D} y distribución de probabilidades (0.5,0.25,0.125,0.125), a continuación se proponen dos códigos binarios.

Símbolo	**Probabilidad**	**Longitud fija**	**Longitud variable**
A	0.5	00	0
B	0.25	01	10
C	0.125	10	110
D	0.125	11	111

Las palabras código del código de longitud fija son todas de dos bits que es la mínima cantidad que garantiza, en este caso, que la correspondencia entre los símbolos de la fuente y las palabras código sea uno a uno. Para el código de longitud variable la única precaución que se ha tenido presente ha sido asignar las palabras código de menor tamaño a los símbolos de la fuente de mayor probabilidad.
Para cuantificar la eficiencia de un código se utiliza la ***longitud media*** obtenida mediante la expresión

$$L_m = \sum = p(x_i)l(x_i)$$

donde x_i representa cada una de las palabras código, $p(xi)$ es la probabilidad que coincide con la del símbolo de la fuente correspondiente pues entre ambas existe una correspondencia uno a uno, y $l(xi)$ es la longitud de la palabra código dada por el número de símbolos.
El código de longitud fija propuesto tiene una longitud media de 2 bits, y el código de longitud variable de 1.75 bits. Este resultado refleja que los códigos de longitud variable son generalmente más eficaces en el sentido que para representar la misma información utilizan menos símbolos en media (siempre y cuando se asignen las palabras código de menor tamaño a los símbolos de la fuente de mayor probabilidad). En aquellos casos en los que los símbolos de la fuente son equiprobables, los códigos de longitud fija y variable son equivalentes.

Símbolo	**Código**
A	0
B	10
C	01
D	010

Cuando se trabaja con códigos de longitud variable aparece un problema en el lado decodificador. ¿Cómo se reconoce cada palabra código? ¿Dónde empieza y acaba cada palabra código?. Usando la tabla anterior, el codificador escribe la palabra código 010 correspondiente al símbolo D. El decodificador recibe 010 y tiene 3 posibles alternativas de decodificación:

- 010 corresponde al símbolo D.
- El primer 0 corresponde al símbolo A y el resto (01) al símbolo B1.
- Los dos primeros bits (10) corresponden al símbolo C y el resto (0) al símbolo A.

Las tres alternativas son igualmente válidas, lo que impide que el decodificador pueda averiguar el símbolo transmitido debido a la ambigüedad generada por no poder delimitar las palabras código recibidas. Esta clase de código se llama ***no singular***

Existe un tipo de código conocido como ***instantáneo o prefijo*** cuyo único requisito es que ninguna palabra código puede ser prefijo de otra, es decir, si dadas dos palabras código distintas *wj* y *wi* se cumple que no existen palabras código de la forma:

$$w_j = w_i w$$

donde $w_i w$ representa la concatenación de $w_i w$ y w.
El siguiente es un ejemplo de código instantáneo:

Símbolo	Código
A	0
B	10
C	110
D	111

La construcción de códigos instantáneos impone ciertas limitaciones sobre el tamaño de las palabras código. Supongamos una alfabeto de tres símbolos A, B y C que se desea codificar con un código binario instantáneo. Con este fin nos aventuramos a fijar la longitud de cada palabra código como {1,1,2}.
Con estas longitudes es fácil observar que es imposible obtener un código instantáneo, donde no haya ninguna palabra código que sea prefijo de otra. El conjunto de longitudes de palabras código que posibilitan la construcción de un código instantáneo es indicado mediante el siguiente teorema llamado ***Teorema de Kraft.***
Dado un conjunto de longitudes de palabras código {*l*0, *l*1, ..., *lr*-1} y un alfabeto código de *s* símbolos, existe un código instantáneo con las longitudes especificadas si y solo si se cumple que

$$s^{-l0} + s^{-l1} + s^{-l2} + s^{-l3} + ... + s^{-lr} <= 1$$

El teorema de Kraft indica la condición que deben cumplir las longitudes de las palabras código para que formen un código instantáneo, pero no indica cuál de ellas conduce a una longitud media mínima. Para que un código instantáneo sea *óptimo* se debe cumplir la condición de que su longitud media debe ser la mínima posible. Esto se obtiene minimizando la expresión de la longitud media *Lm* con la restricción indicada por el Teorema de Kraft, resultando:

$$l_i = log_s \, 1/p_i$$

siendo *s* el tamaño del alfabeto código, en cuyo caso la longitud media coincide numéricamente con el valor de la entropía *Hs*.

En esta última expresión se basa la matemática de una gran cantidad de las técnicas de compresión cuyo concepto pasamos a dar ahora:

Compresión: Es obtener secuencias de palabras códigos lo más concisas posible, reduciendo la longitud media del mensaje final respecto del mensaje original.

Básicamente, en algunas técnicas de compresión se trata de aprovechar la redundancia de información, en el caso que la fuente sea una imagen o sonido, lo que llevó a que se desarrollaran técnicas que aprovechan una cierta correlación entre palabras código adyascentes (cambio ligero en la tonalidad de imágenes o de frecuencia en sonidos), lo que lleva comprimir aún más la información eliminando lo que para el ojo o el oido ya es imperceptible.

La teoría anterior sirve más bien para las técnicas de compresión sin pérdida de información que se aplican mayormente a textos.

A continuación pasamos a describir unas cuantas técnicas que consideramos interesantes de ver, entre ellas técnicas de compresión de textos e imágenes pero ya no hablaremos de técnicas de compresión de sonido.

Lossless (compresión sin pérdida)

Los métodos de compresión sin pérdida de información (*lossless*) se caracterizan porque la tasa de compresión que proporcionan está limitada por la entropía (redundancia de datos) de la señal original. Entre estas técnicas destacan las que emplean métodos estadísticos, basados en la teoría de Shannon, que permite la compresión sin pérdida. Por ejemplo: codificación de Huffman, codificación aritmética y Lempel-Ziv. Son métodos idóneos para la compresión dura de archivos.

Compresores Estadísticos

Utilizan la información de las probabilidades de los mensajes de la fuente para construir una codificación.

Ejemplos de compresores estadísticos:

* Compresores del tipo Huffman ó Shannon-Fano
* Compresores aritméticos
* Compresores predictivos

Los codificadores estadísticos son la piedra angular de la compresión.

La idea básica de su trabajo es la siguiente: El compresor predice la entrada y escribe menos bits en su salida si la estimación ha sido correcta. El descompresor debe ser capaz de realizar las mismas predicciones que el compresor para poder decodificar bien lo transmitido, ya que la transmisión es diferente dependiendo de la predicción.

Este tipo de compresores parten de:

* Una fuente de información de *n* mensajes
* Las probabilidades de aparición de cada mensaje de la fuente (que pueden ser extraídas a priori de forma experimental o pueden ser dadas y fijas)

Un *alfabeto* de salida que consta de una serie de símbolos (por ejemplo, el alfabeto binario consta de los símbolos 0 y 1).

Esta codificación debe ser tal que explote la redundancia en la información dada por la fuente para producir compresión.

Para simplificar, supondremos que nuestras fuentes de información son ficheros ASCII de 8 bits y que cada carácter es un mensaje de la fuente (256 mensajes).

El conocer las probabilidades de cada mensaje implica que el compresor debe realizar una primera pasada por el archivo para recuperar las frecuencias de cada mensaje.

Esto puede ser un problema, sobre todo para dispositivos de cinta de una sola pasada. Sin embargo, se puede argumentar que la codificación se puede realizar sobre bloques ó utilizarse algoritmos adaptativos.

Huffman o Shannon-Fano

Ambos algoritmos terminan construyendo un árbol que representa la codificación que de los mensajes de la fuente se ha realizado, de manera que los nodos hoja contienen cada uno de los mensajes emitidos por la fuente.

En el caso más simple, el alfabeto de salida en el que se realiza la codificación es binario. Esto quiere decir que de cada nodo partirán dos ramas, una para el 0 y otra para el 1.

El código para cada mensaje se construye siguiendo el camino desde el nodo raíz hasta la hoja que representa el mensaje.

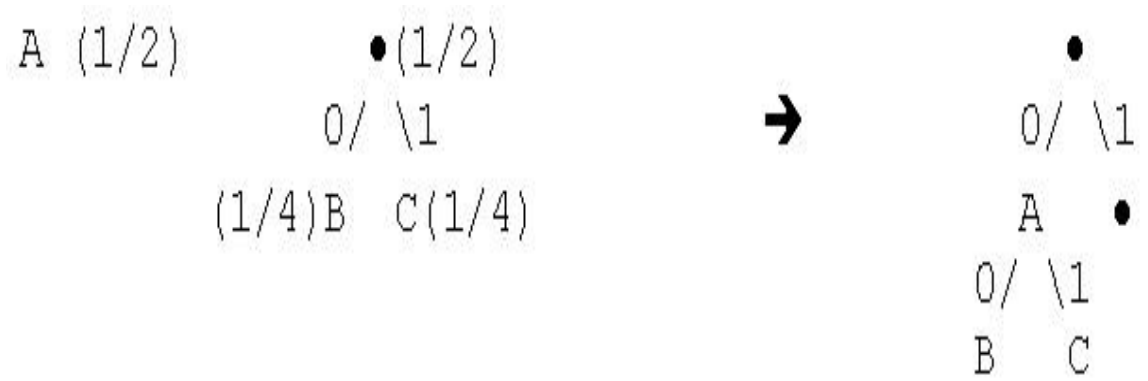
Lo verdaderamente importante es que la codificación resultante de la aplicación de estos algoritmos asigna longitudes de codificación **inversamente proporcionales** a la probabilidad de aparición de cada mensaje.

Es decir, el mensaje que más aparezca tendrá una codificación más corta, con lo que se ahorrará espacio en la transmisión.

Huffman recoge los dos nodos con **menor probabilidad** del árbol. Construye entonces un nodo padre de ambos y se la asigna la probabilidad suma de ambos hijos.
Este proceso hace que el árbol crezca y los nodos con menor probabilidad queden más al fondo.

La implementación de este algoritmo es muy sencilla si utilizamos una estructura de datos conocida como *heap*.

Un ejemplo de la construcción de un árbol de Huffman lo podemos ver así. Sean los mensajes y probabilidades siguientes
A (1/2) B (1/4) C (1/4)



En donde primero se agrupan B y C en un único nodo de probabilidad ½ y posteriormente se construye el árbol final: A = 0, B = 10 y C = 11 (en binario) es decir, 1.5 bits de media por mensaje (no 8 bits / byte como en un fichero con sólo A's, B's y C's).

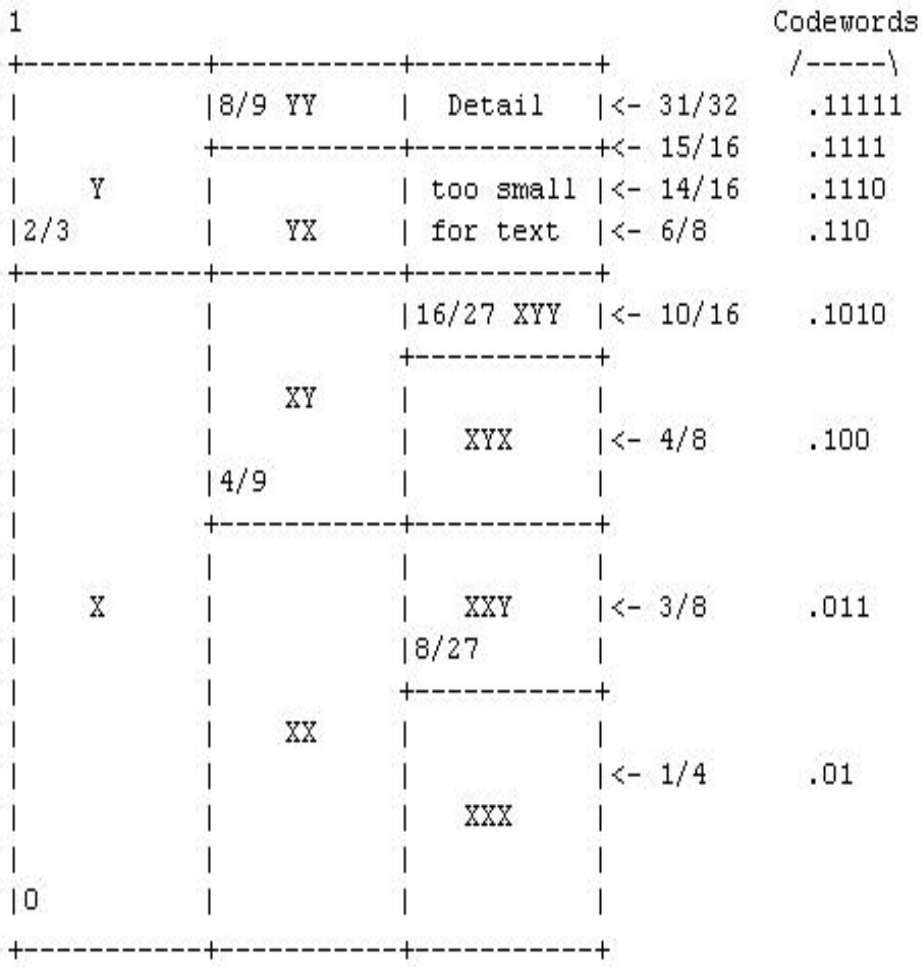
Nótese que este esquema nos permite que si, a la hora de descomprimir, el decodificador Huffman o Shannon-Fano poseen el mismo árbol que se hizo para comprimir, la decodificación es tan sencilla como leer *bits* de la fuente a descomprimir y seguir el camino desde la raíz hacia abajo bifurcando hacia un lado o hacia otro dependiendo del valor del bit. Eventualmente llegaremos a una hoja, que representa al mensaje que se estaba recibiendo.
El principal problema de estos algoritmos es que el compresor debe realizar primero un recorrido del archivo (o de la porción del mismo, si se realiza por bloques) a comprimir para reunir las frecuencias de los mensajes de la entrada.
Como el descompresor no tiene esa posibilidad, ya que sólo recibe los códigos asignados a los mensajes, el árbol ya procesado o las frecuencias de cada mensaje, junto con los datos, deben ser pasados al descompresor. Esto introduce una sobrecarga que hay que evitar de alguna manera.

Compresores aritméticos

Los compresores aritméticos se basan en las probabilidades de ocurrencia de los mensajes a la entrada.
Se basan en la representación de un valor del intervalo [0,1] con más decimales (más precisión) cuanto más información contengan los datos a comprimir.

Ejemplo:
Supongamos que queremos codificar una entrada que consta de dos símbolos, X e Y, con probabilidades p(X) = 2/3 y p(Y) = 1/3.

Al recibir una X, dividimos el intervalo [0,1] y nos quedamos con los 2/3 inferiores, por ejemplo. Tendremos entonces el intervalo [0, 2/3]. En caso de haber recibido una Y, habríamos cogido el intervalo del tercio inferior, es decir, [2/3, 1]. En cada paso, dividimos por los 2/3 inferiores o el tercio superior el intervalo que tengamos. Al final, el código que emitimos, en binario, es un número que cae en el intervalo. Se elegirá aquel que utilice menos bits en su representación. Este número representa a toda la entrada Con este número, representado por los bits que necesite, junto con la información del número de elementos codificados y la probabilidad de cada uno, el descompresor puede reconstruir la entrada.



Compresores predictivos

Procuran **predecir** el siguiente mensaje de la entrada tomando como base de conocimiento la entrada procesada hasta ese momento (en el fondo, también probabilidades). Si el mensaje que se encuentra a la entrada coincide con el predicho, su codificación se podrá hacer con menos bits. Si no, su codificación se hará con más bits, que permitirán entonces sincronizar al descompresor para que mantenga sus tablas internas idénticas a las del compresor sin pasárselas explícitamente.

Poseen ciertas ventajas sobre los anteriores algoritmos, entre ellas su velocidad:

Al actuar sobre un mensaje cada vez y realizar una predicción que generalmente suele ser de cálculo sencillo, son capaces de dar una alta velocidad de compresión/descompresión.

A la vez que rápidos, resultan sencillos de programar, por lo que se pueden convertir en una solución barata para sistemas de compresión transparente en tiempo real, con unas relaciones de compresión aceptables.

Ejemplo: PPP Predictor Compression Protocol (1987)

Predice el siguiente carácter a partir de los dos anteriores de la entrada. Para ello construye una matriz de 256*256 que guarda en cada casilla m[i,j] el byte que anteriormente siguió a dos entradas consecutivas de valores ASCII i y j.

Cuando se va procesando la entrada, el algoritmo siempre sabe qué dos mensajes precedieron al actual, por ejemplo p1 y p2.

Con esta información, su predicción para el carácter actual, pongamos c, será m[p1,p2]. Si acierta, es decir, si c=m[p1,p2], la salida será sólo un bit, que, puesto a 1 informa de que se ha logrado predecir el mensaje actual.

Si no acierta, su salida será un bit puesto a 0, indicando que no se predijo y a continuación el mensaje no predicho. Además, actualiza la tabla para que la vez siguiente sea capaz de predecir: m[p1,p2] = c.

Hay otras alternativas, como la que se incluye en *PK-ZIP 1.x* con el algoritmo "Reducing".

Consiste en considerar un conjunto de seguidores de cada carácter (*follower sets*) que guardan los caracteres que han seguido con más probabilidad a un carácter dado. Como el conjunto de seguidores es pequeño, para identificar un seguidor se pueden utilizar menos bits.

El trabajo del compresor es aquí difícil, ya que tiene que calcular el tamaño óptimo de los conjuntos de seguidores de manera que no se pierdan bits inútilmente. El problema aquí también es que los conjuntos deben ser pasados al descompresor.

Compresores en tiempo real

Con el aumento de velocidad de los ordenadores la compresión se está aplicando cada vez más a entornos en tiempo real o de compresión transparente

En cuanto a los usos, tenemos los siguientes:

- Compresión de disco en tiempo real: **Stacker**, **NTFS.**
- Compresión transparente en comunicaciones
- Soporte de .ZIP y .GZ directo en Unix,
- Tratamiento, en algunos sistemas operativos, de los archivos comprimidos como directorios.
- Sistemas de ayuda comprimidos como los **.HLP** de Windows o los **Portable Document Format (PDF)** de Adobe Systems.
- Formatos de vídeo comprimidos (de forma *lossless*) para juegos, presentaciones, etc.

Lossy (compresión con pérdida)

Los métodos de compresión con pérdida de información (*lossy*) logran alcanzar unas tasas de compresión más elevadas a costa de sufrir una pérdida de información sobre la fuente original. Por ejemplo: JPEG, compresión fractal, EZW, SPIHT, etc. Para la compresión de imágenes se emplean métodos *lossy*, ya que se busca alcanzar una tasa de compresión considerable, pero que se adapte a la calidad deseada que la aplicación exige sobre la imagen objeto de compresión.

Al ir aumentando la velocidad de los computadores, algoritmos que antes no se podían considerar de tiempo real pueden ser utilizados en estos entornos, dando una media superior de compresión.

El interés actual se desvía a los compresores *lossy* dado el auge que las técnicas multimedia han experimentado.

Con la venida de la televisión digital, por ejemplo, la búsqueda de algoritmos de compresión de imágenes y sonido pasa a primer plano.

Ejemplos de los avances en este tipo de compresores son, por ejemplo, el estándar MPEG-2, usado actualmente por las compañías de televisión en sus enlaces por satélite, y lo que se denomina *MPEG layer 3*, utilizado en compresión de sonido de calidad CD, con una reducción normal de 8 a 1.

Compresión De Imágenes Sin Pérdidas

Cuando un conjunto de datos se comprime, como un documento de texto o un dato numérico, se hace siempre para que la descompresión subsecuente produzca el dato original exacto. Si el dato reconstruido no es exactamente igual al original, el documento de texto podría tener caracteres errados, o un computador podría tener unas entradas equivocadas. Debido al tipo de datos que se manejan en estos ejemplos, una aproximación no funciona bien. Para estos casos, los datos deben reconstruirse exactamente igual que su forma original, o el esquema de compresión es inutilizable. El tipo de esquema

de compresión donde los datos comprimidos se descomprimen a su forma original exacta se llama compresión sin pérdidas. Está desprovisto de pérdidas, o degradaciones, de los datos.

Se han desarrollado una variedad de esquemas de compresión de imágenes sin perdidas. Muchas de estas técnicas vienen directamente del mundo de compresión de datos digital y se han adaptado meramente para el uso con datos de la imagen digitales.

Codificación De Longitud Fija

En la codificación de longitud fija, se asignan palabras de código de longitud iguales a cada símbolo en un alfabeto A sin tener en cuenta sus probabilidades. Si el alfabeto tiene M símbolos diferentes (o bloques de símbolos), entonces la longitud de las palabras de código es el entero más pequeño mayor que $\log_2 M$.

Dos esquemas de codificación de longitud fija comúnmente usados son los códigos naturales y los códigos Gray, que se muestran en el siguiente cuadro para el caso de una fuente de cuatro símbolos. Nótese que en la codificación Gray, las palabras de código consecutivas difieren en un solo bit. Esta propiedad de los códigos Gray puede proveer una ventaja para la detección de errores.

Simbolo	Código Natural	Código Gray
a_1	00	00
a_2	01	01
a_3	10	11
a_4	11	10

Puede mostrarse que la codificación de longitud fija sólo es óptima cuando:
* El número de símbolos es igual a una potencia de dos
* Todos los símbolos son equiprobables.

Sólo entonces podría la entropía de la fuente ser igual a la longitud promedio de las palabras código que es igual a la longitud de cada palabra código en el caso de la codificación de longitud fija. Para el ejemplo mostrado en el cuadro anterior, la entropía de la fuente y la longitud media de la palabra código es 2, asumiendo que todos los símbolos son igualmente probables. A menudo, algunos símbolos son más probables que otros, donde sería más ventajoso usar codificación de la entropía. Realmente, la meta de un sistema de compresión de imágenes es obtener un conjunto de símbolos con una distribución de probabilidad inclinada, para minimizar la entropía de la fuente transformada.

Codificación De Longitud Variable

El método más simple de compresión de imágenes sin pérdidas consiste en reducir únicamente la redundancia de la codificación. Esta redundancia está normalmente presente en cualquier codificación binaria natural de los niveles de gris de una imagen. Dicha redundancia se puede eliminar construyendo un código de longitud variable que asigne las palabras código más pequeñas a los niveles de gris más probables.

Existen varios métodos de codificación de longitud variable, pero los mas usados son la codificación Huffman y la codificación aritmética.

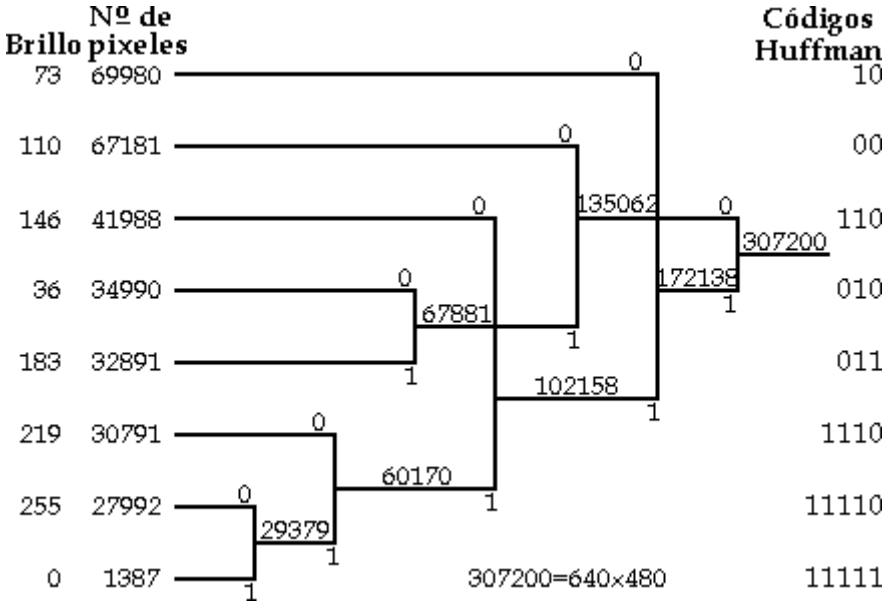
Codificación Huffman. La codificación Huffman convierte los valores de brillo de los pixeles de la imagen original en nuevos códigos de longitud variable, basado en su frecuencia de ocurrencia en la imagen. De esta manera, a los valores de brillo que ocurren más frecuentemente se les asignan los códigos más cortos y a los valores de brillo que ocurren con menos frecuencia se les asignan los códigos más largos. El resultado es que la imagen comprimida requerirá de menos bits para describir la imagen original.

El esquema de compresión Huffman comienza mirando el histograma de brillo de una imagen. Con el histograma, la frecuencia de ocurrencia para cada brillo en la imagen está disponible. Ordenando los valores de brillo por sus frecuencias de ocurrencia, se obtiene una lista donde el primer valor se encuentra más a menudo en la imagen, y el último valor se encuentra menos a menudo en la imagen. Con esta lista, el codificador Huffman asigna nuevos códigos a cada valor de brillo. Los códigos asignados son de longitudes variables; los códigos más cortos son asignados a los primeros (más frecuentes) valores m de la lista y, eventualmente, los códigos más largos se asignan a los últimos (menos frecuentes) valores de la

lista. Finalmente, la imagen comprimida es creada simplemente sustituyendo los nuevos códigos de valores de brillo de longitud variable por los códigos de valores de brillo originales de 1 byte. Por supuesto, la lista de códigos Huffman que acopla los valores de brillo originales a sus nuevos códigos Huffman variables se debe añadir a la imagen para el uso de la operación de descompresión Huffman

Los códigos Huffman son asignados creando un árbol de Huffman que hace combinaciones con los valores de brillo basado en la suma de las frecuencias de ocurrencia. El árbol de Huffman asegura que los códigos más largos se asignen a los brillos menos frecuentes y los códigos más cortos se asignen a los brillos más frecuentes. Usando el brillo clasificado en orden de sus frecuencias de ocurrencia, los dos del final de la lista (menos frecuentes) se combinan y se etiquetan como 0 y 1. Los brillos combinados son representados por la suma de las frecuencias de ocurrencia. Entonces, se determinan y se combinan las próximas dos frecuencias de ocurrencia más bajas. De nuevo, el siguiente par se etiqueta 0 y 1, y es representado por la suma de las frecuencias de ocurrencia. Esto continúa hasta que todo el brillo se ha combinado. El resultado es un árbol que, cuando se sigue del final hasta el principio, indica el nuevo código Huffman binario para cada brillo en la imagen.

El brillo se ordena basado en sus frecuencias de ocurrencia y entonces se combina en un árbol de Huffman (figura de abajo), como se describió anteriormente. Aunque todos los pixeles en la imagen original fueron codificados como valores de brillo de tres bits, los códigos Huffman son tan pequeños como un bit y pueden ser tan grandes como 7 bits. El código Huffman más largo nunca puede ser mayor que el número de valores de brillo diferentes en la imagen (en este caso 8) menos 1. Aunque una imagen codificada con Huffman puede tener un poco de brillo con códigos muy largos, sus frecuencias de ocurrencia siempre son estadísticamente bajas.

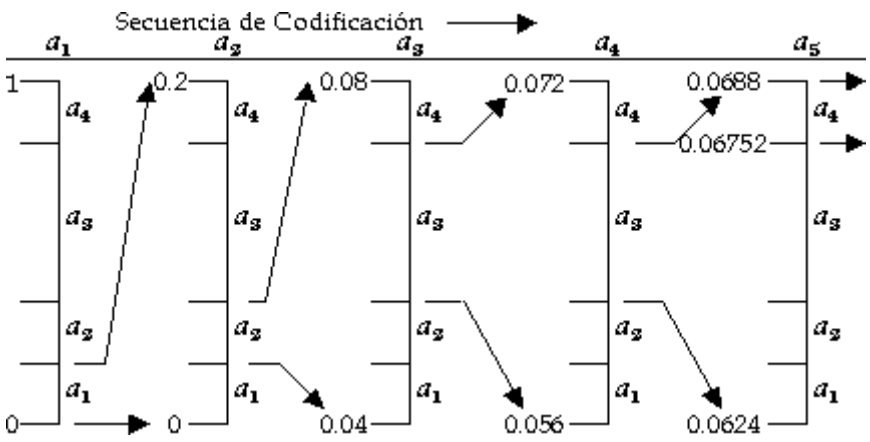


La cantidad de datos de la imagen original puede calcularse como 640x480x3 bits. La cantidad de datos de la imagen codificada por Huffman puede calcularse como la suma de las ocho frecuencias de ocurrencia multiplicadas por el respectivo número de bits en su código.

La descompresión de imágenes Huffman invierte el proceso de compresión sustituyendo los valores de brillo originales de longitud fija de un byte por valores codificados de longitud variable. La imagen original se reconstruye exactamente. La compresión de imágenes Huffman generalmente proporcionará razones de compresión de alrededor de 1.5:1 a 2:1.

Codificación Aritmética. En la codificación aritmética no existe una correspondencia biunívoca entre los símbolos fuente y las palabras código. En cambio, se asigna una sola palabra código aritmética a una secuencia completa de símbolos fuente. La propia palabra código define un intervalo de números reales entre 0 y 1. Conforme aumenta el número de símbolos del mensaje, el intervalo utilizado para representarlo se va haciendo menor y se va incrementando el número de unidades de información necesarias para representar dicho intervalo. Cada símbolo del mensaje reduce el tamaño del intervalo según su probabilidad de aparición. Puesto que esta técnica no requiere, como sucedía con la técnica de Huffman, que cada símbolo de la fuente se traduzca en un número entero de símbolos del código (esto es, que los símbolos se codifiquen uno a uno), se alcanza (solo en teoría) el limite establecido por el teorema de codificación sin ruido.

La figura de abajo ilustra el proceso básico de la codificación aritmética. En este caso, se codifica una secuencia o mensaje de cinco símbolos, a1 a2 a3 a3 a4, generados por una fuente de cuatro símbolos. Al principio del proceso de codificación, se supone que el mensaje ocupa todo el intervalo semiabierto [0,1). Como se muestra en el cuadro de abajo, este intervalo se subdivide inicialmente en cuatro regiones en función de las probabilidades de cada símbolo de la fuente. Por ejemplo, se asocia el subintervalo [0, 0.2) al símbolo a1. Puesto que se trata del primer símbolo del mensaje a codificar, el intervalo del mensaje se reduce inicialmente a [0, 0.2). Así, en la figura de abajo el intervalo [0, 0.2) abarca toda la altura de la figura y se marcan los extremos con los valores del rango reducido. Posteriormente, se divide este rango reducido de acuerdo con las probabilidades de los símbolos de la fuente original, y el proceso continúa con el símbolo del mensaje. De esta forma, el símbolo a2 reduce el subintervalo a [0.04, 0.08), a3 lo reduce aún más, dejándolo en [0.056, 0.072), y así sucesivamente. El último símbolo del mensaje, que se debe reservar como indicador especial de fin de mensaje, reduce el intervalo, que pasa a ser [0.06752, 0.0688). Por supuesto, se puede utilizar cualquier número que esté dentro del subintervalo, como por ejemplo el 0.068, para representar el mensaje.



Símbolo fuente	Probabilidad	Subintervalo inicial
a_1	0.2	[0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1)

En el mensaje codificado aritméticamente de la figura de arriba, se utilizan tres dígitos decimales para representar el mensaje de cinco símbolos. Esto se traduce en 3/5, ó 0.6 dígitos decimales por símbolo fuente, lo que se aproxima bastante a la entropía de la fuente, que resulta ser de 0.58 dígitos decimales por símbolo. Conforme aumenta la longitud de la secuencia a codificar, el código aritmético resultante se aproxima a límite establecido por el teorema de la codificación sin ruido. En la práctica, existen dos factores que hacen que el rendimiento de la codificación se aleje de este límite:
La inclusión del indicador de fin de mensaje, necesario para separar un mensaje de otro.
La utilización de aritmética de precisión finita.

Como habiamos mencionado anteriormente los métodos de compresión con pérdida de información (*lossy*) logran alcanzar unas tasas de compresión más elevadas a costa de sufrir una pérdida de información sobre la fuente original. Un ejemplo es la compresión fractal.

Introducción a los Fractales

Hoy en día, después de que en 1975 Benoit Mandelbrot acuñara la palabra **fractal**, 21 años después no existe una definición generalizada del mismo. A pesar de esto, tomando como base el conjunto de Mandelbrot podemos aproximar una definición:
Un Fractal, palabra que proviene del latín 'fractus' o lo que es lo mismo, algo roto, algo **no entero**, comprende objetos geométricos de cierta entidad que pueden ser descritas en términos de dimensiones no enteras o dimensiones fractales.
Las características que definen un fractal son las siguientes:

- **Autosimilitud**: A diferentes escalas, un fractal conserva la misma apariencia, siempre existe una clara similitud entre partes muy distantes de una misma figura fractal.
- **Infinito Detalle**: Relacionada con la anterior característica, al ampliar un fractal, tanto más detalle revela este, sin que se tenga un límite en el que se aprecien bloques.
- **Dimensión no entera**: Al contrario de la geometría clásica, en la que la las figuras tienen 1, 2 o 3 dimensiones, un fractal puede desarrollarse en una dimensión no entera, como, por ejemplo la curva de Koch, que lo hace en la dimensión 1.26; esto es, ocupa parte del plano pero no llega a tener la entidad de figura bi-dimensional.
- Las fórmulas o algoritmos que los definen son relativamente **sencillos** y con un conjunto muy **reducido** de datos.
- Su **algorítmia** es definida por una característica clave: **la iteración**. Es gracias precisamente al ordenador lo que permite experimentar y descubrir nuevos conjuntos y sin él, probablemente Mandelbrot no hubiese llegado tan lejos. El ordenador fué y es imprescindible en cualquier campo que abarque los fractales.

Además de la belleza plástica que todos hemos contemplado en la generación de un fractal, es algo más que bellas e intrincadas imágenes generadas por ordenador; en la última década los fractales se utilizan para la representación y el análisis de una gran variedad de procesos complejos a lo largo de diversos campos, como pueden ser la Física, las Matemáticas, Biología, Química, Geología, ...

La facilidad de los fractales para expresar o simular fenómenos que suceden en la Naturaleza es debido a la autosimilud; los fenómenos naturales creados o en los que interviene el azar, la aleatoriedad, estadísticamente siguen una periodicidad o autosimilitud que puede ser caracterizada a través de la dimensión fractal.

La utilización de la geometría fractal en investigaciones numéricas, teóricas y experimentales, ha hecho posible tener una visión práctica y predecible de problemas que antes eran prácticamente intratables.

Compresión Fractal de imágenes

Una de las aplicaciones más excitantes a través de los fractales y su aplicación en el mundo de las imágenes digitales es sin duda la **Compresión Fractal de imágenes**. A continuación se expresa una pequeña introducción a la compresión Fractal:
En 1987 Michael Barnsley, fundador de la empresa Iterated Systems junto con Alan Sloan, descubrió que era posible controlar el contenido de una imagen fractal de forma precisa y hacerla parecer increíblemente similar a una imagen del mundo real. Un ejemplo temprano de aproximación a una imagen real basado en fractales lo contituye el clásico helecho fractal.

*El helecho completo mantiene una apariencia similar en cada una de sus hojas, cada hoja con sus sub-hojas y así sucesivamente. Sigue un **patrón** claramente definido*

La generación de la anterior figura proviene, en su base, de un simple sistema de ecuaciones que opera en el plano a través de rotaciones, traslaciones y escalados; se trata de la **transformación afín** y constituye parte fundamental de la compresión fractal. Expresada en términos numéricos, no son más que los coeficientes del sistema de ecuaciones anteriormente citado, y por ello su sencillez y su capacidad de compresión. El conjunto de estos coeficientes constituye lo que se llama **mapa de afinidad**.

Transformación afín expresada de forma matricial.

$$W\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.2 & 0.4 \\ -0.5 & 0.1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 2 \\ 5 \end{pmatrix}$$

Concretando sobre la compresión fractal, esta se basa en un tipo concreto de transformación afín: las transformaciones afines **contractivas** y se caracterízan porque su resultado en el plano euclídeo es más pequeño que la imagen a la que se le aplicó la transformación afín (matemáticamente, la distancia entre dos puntos cualesquiera de la imagen original es siempre mayor o igual que la distancia entre ambos puntos tras haberles aplicado la transformación afín contractiva).

Para definir la imagen del helecho fractal tan solo son necesarias **24 bytes** que es lo que ocupan los mapas de afinidad de las 4 transformaciones afines que contituyen el IFS generador del helecho.
Básicamente, el proceso de compresión, muy a grandes rasgos, es el siguiente:

- La imagen origen es dividida en subconjuntos llamados **regiones de dominio**, sobre las que se buscarán redundancias dentro de la imagen.
- Para cada región de dominio se escoge una **región de rango**, mayor en tamaño que la región de dominio.
- Todas las posibles regiones de rango sin rotadas, escaladas y se las aplica una simetría (en definitiva, una transformación afín), eligiendo la región de rango que junto con la transformación afín mas se aproxime a la región de dominio.
- La elección de la región de rango, junto con la transformación afín, se almacenan en el fichero fractal, y contituirán los patrones para la descompresión y así reconstruir la imagen original.

El proceso de descompresión se resume en **iterar** un número suficiente de veces todas las transformaciones afines almacenadas sobre las regiones de rango hasta llegar a un conjunto invariante, al atractor, que es una buena aproximación de la imagen original (al tratarse de una compresión con pérdidas, nunca será una réplica pixel a pixel del original).

Características de la compresión Fractal: Ventajas y desventajas

- Tecnología de 'compresión con pérdidas' de reciente creación y superior al método JPEG.
- Sujeta a una patente comercial, perteneciente a Iterated Systems**.** A pesar de ello, Iterated Systems ofrece un gran aperturismo a través de Internet, intentando alcanzar que esta tecnología se convierta en un estandar 'de hecho'
- Zoom avanzado sin pixelización. Al ser independiente de la resolución, no produce la aparición de bloques ampliados (pixels). A pesar de ello, la ampliación no deja de ser una forma avanzada de interpolación.
- Ahorro de espacio de almacenamiento y, evidentemente, menores tiempos de transmisión de imágenes a través de Internet.
- El proceso de Compresión/Descompresión es acentuadamente **asimétrico**. Mientras que la descompresión es casi inmediata, la compresión requiere considerablemente más tiempo de computación.

Conclusion

Algunas consideraciones sobre la Compresion:

La compresion de datos, ya sean imagenes, videos, musica, textos, o cualquier tipo de archivo es un campo de constante investiagación ya que los avances en este campo han permitido que sea posible compartir datos a través de redes de computadoras y a través de Internet de una manera más eficiente y por supuesto con mayor velocidad.

Algunas Ventajas de la compresion son:
* Mayor rapidez de transmisión. Cuando el ancho de banda es limitado, por ejemplo una señal de video necesita aproximadamente 50 Mbits/seg.
* Mayor cantidad de información almacenada. Como sabemos, los medios físicos de almacenamiento de datos son finitos, esto es, la capacidad de almacenamiento es limitada, por lo que gracias a la compresión, es posible almacenar grandes cantidades de datos sin desperdiciar espacios físicos de almacenamiento. Ejemplo, es posible almacenar una enciclopedia Británica escaneada de 25 Gbytes.

Existen dos métodos generales de compresion que son: Los métodos de compresión con pérdida de información (Lossy) y los métodos de compresión sin pérdida de información (Lossless). Dentro de estos dos métodos existen una gran variedad de técnicas (algoritmos) para la compresión.

Bibliografía

- http://thunder.prohosting.com/~josuna/fractal/Intro/intro.htm (introduccion a la compresion fractal)
- http://www.pucmmsti.edu.do/materias/ptaveras/itt437/ (intro a los sistemas de codificacion)
- http://www.fuac.edu.co/autonoma/pregrado/ingenieria/ingelec/proyectosgrado/compresvideo/compresion_sin_perdidas.htm (compresión de imágenes sin pérdida)
- http://dis.eafit.edu.co/cursos/st780/material/medios/fund-compresion.pdf
- http://bioinfo.uib.es/~joemiro/aenui/ProcWeb/actas2001/gipra359.pdf
- http://www.ace.ual.es/~vruiz/investigacion/IR-1/html/informe.html
- http://lsi.ei.uvigo.es/~formella/doc/tc01/node1.html
- ftp://links.uwaterloo.ca/pub/Fractals/Papers/Waterloo
- http://dit.teleco.ulpgc.es/usuarios/profes/pablo/