Desarrollo de un compresor de textos orientado a palabras basado en PPM*

Sandra Álvarez, Ana Cerdeira-Pena, Antonio Fariña, Susana Ladra

Laboratorio de Bases de Datos, Univ. de A Coruña, España.
{salvarezg,acerdeira,fari,sladra}@udc.es

Resumen Reducir el espacio de almacenamiento y el tiempo de transferencia se ha vuelto un aspecto fundamental en las Bases de Datos Textuales. En este trabajo se presenta un nuevo compresor, denominado PPM orientado a palabras (SWPPM), en el que se aplican los modelos estadísticos propios de PPM utilizando como símbolos de entrada las palabras. Presenta varios desafíos técnicos para los que es necesario aplicar nuevas estrategias y diseñar estructuras de datos adaptadas al problema. SWPPM alcanza ratios de compresión del 28 %.

1. Introducción

En los últimos años, las Bases de Datos Textuales han cobrado una gran importancia debido, entre otros factores, a la aparición y rápida expansión de las Bibliotecas Digitales, así como de la Web, que supone la mayor colección de textos existente. Es en este escenario, en donde las técnicas de compresión de textos resultan ser especialmente beneficiosas, ya que no sólo permiten reducir el espacio utilizado en disco, sino también el ancho de banda requerido en la transferencia de los datos y, dependiendo de las técnicas empleadas, también permiten mejorar el tiempo requerido para procesar dicha colección.

Lograr la máxima compresión supone por tanto una gran ventaja en multitud de aspectos, y es en esta línea de investigación donde destaca la técnica PPM (Prediction by Partial Matching)[4], un compresor que obtiene ratios de compresión del orden del 20 %.

En términos generales, las técnicas de compresión se clasifican según sus características. De acuerdo al vocabulario de entrada que utilizan pueden ser orientadas a caracteres o a palabras y, según su alfabeto de salida, pueden ser orientadas a bit o a byte. Además, los compresores pueden considerar un número fijo de símbolos de entrada para codificar, como es el caso general de los compresores estadísticos, o un número variable, típico de las técnicas basadas en diccionario [11]. Por último, los códigos resultantes de comprimir cada símbolo de entrada pueden tener longitud fija o variable.

* Parcialmente financiado por: el "Ministerio de Educación y Ciencia"(PGE y FEDER) ref. TIN2006-15071-C03-03; "Xunta de Galicia", ref. 2006/4; y el "Ministerio de Ciencia e Innovaciónref. AP2007-02484 (Programa FPU) para Ana Cerdeira-Pena

Los compresores estadísticos utilizan un modelo del texto (básicamente símbolos de entrada y sus frecuencias) para asignar códigos de menor longitud a aquellos símbolos que posean una mayor frecuencia. Estas frecuencias pueden venir prefijadas independientemente del texto que se comprima (*técnicas estáticas*) o depender del texto que se va a comprimir (técnicas semiestáticas o dinámicas) [1]. Las técnicas semiestáticas realizan una pasada previa sobre el texto para recopilar los diferentes símbolos existentes y sus frecuencias y utilizarlos para codificarlos adecuadamente en una segunda pasada. Por su parte, las técnicas dinámicas realizan una única pasada. En ella se va actualizando el modelo del texto que es utilizado para codificar los símbolos a medida que se van procesando [1]. El descompresor partirá del modelo previo en las técnicas estáticas y semiestáticas, mientras que en las técnicas dinámicas lo reconstruye del mismo modo que el compresor, a medida que descomprime el texto.

En función del modelo estadístico utilizado, hablamos de *modelos de orden 0*, cuando los símbolos de entrada se consideran de modo independiente (por ejemplo, Huffman [5]); y *modelos de orden n*, cuando la probabilidad de ocurrencia de cada símbolo viene determinada por el contexto de los n símbolos que lo preceden.

A lo largo de la historia han surgido numerosos compresores estadísticos orientados a carácter. La técnica de Huffman [5] es quizás el representante más conocido de esta familia de compresores, aunque sus ratios de compresión son pobres (en torno al 60 %), lejos del 20 % del PPM.

La creación de compresores orientados a palabras pueden mejorar las tasas de compresión [6]. Adaptaciones de la técnica clásica de Huffman, como la descrita en [9], las conocidas como Plain Huffman y Tagged Huffman [10] o técnicas con similares características como el End-Tagged Dense Code [2,3], mejoran los ratios de compresión obtenidos con Huffman clásico hasta un 30 %. Estas técnicas (orientadas a palabras), utilizan modelos de frecuencia de orden 0, es decir, no tienen en cuenta las palabras precedentes para calcular el modelo de frecuencias.

Es conocido que las técnica PPM, que utilizan modelos de órdenes elevados, obtienen muy buenos ratios de compresión. Además, se han visto, por ejemplo con el caso del Huffman clásico, que tomar una técnica orientada a caracteres para producir una orientada a palabras es efectivo, mejorando el ratio de compresión. De aquí surge nuestra propuesta de realizar un compresor semiestático orientado a palabras basado en PPM (SWPPM), que utilice contextos, formados por las palabras precedentes, para calcular la probabilidad de cada símbolo.

2. Conceptos básicos

Prediction by Partial Matching PPM es un compresor estadístico cuya idea fundamental consiste en utilizar los contextos de los últimos símbolos que han sido procesados para predecir cuál es el símbolo que los sucede. Para ello se mantienen modelos de diferentes órdenes, que se traducen en contextos de tamaño variable, utilizando para codificar el símbolo aquel orden de mayor magnitud del que dispongamos información [8]. Naturalmente, uno de los aspectos más críticos de

la implementación de esta técnica es cómo almacenar los contextos y los símbolos que lo siguen (y con qué frecuencia).

El proceso básico de compresión consiste en utilizar el contexto de mayor orden para calcular la probabilidad del símbolo que se va a codificar, emitiendo símbolos de escape en todos aquellos órdenes en los que no se disponga de información sobre el símbolo.

PPM es un compresor dinámico orientado a caracteres, por lo que la información contenida en los modelos de diferentes órdenes se va modificando durante la compresión, añadiendo nuevos contextos con sus símbolos y probabilidades cuando sea necesario, y actualizando las probabilidades de los ya existentes. Esto produce un tiempo de adaptación inicial en los primeros pasos de la compresión, que provoca que los símbolos se codifiquen de forma poco eficiente mientras no se disponga de suficiente información sobre los contextos.

Una vez se disponga de la probabilidad del símbolo actual en el contexto que se pretende codificar, se le proporciona dicha información a un módulo de codificación aritmética, que es el que proporciona el texto comprimido final. Cuando no se dispone de la información necesaria para un símbolo en un contexto dado, es necesario emitir un símbolo de escape. que es necesario emitir cuando no se dispone de la información Dicho símbolo ha de ser codificado como un símbolo más en dicho contexto, por lo que debe tener asignada una determinada probabilidad en cada contexto.

Codificación aritmética Esta es una técnica estadística que, partiendo de un conjunto de símbolos de entrada produce como salida un número entre 0 y 1 que lo identifica unívocamente [7].

La idea principal consiste en mantener un intervalo, que representará en todo momento a los símbolos ya procesados. El intervalo se divide en subintervalos de tamaño proporcional a las probabilidades de los símbolos, convirtiéndose en el intervalo actual aquel subintervalo que se corresponda con el símbolo a codificar.

Esta codificación resulta ser muy adecuada para PPM, puesto que permite definir en cada paso qué probabilidad se le asigna a cada símbolo y cuál es su codificación resultante. Para ello, PPM debe proporcionar en cada paso la frecuencia del símbolo en el contexto actual, la frecuencia total del contexto y la frecuencia acumulada inferior del símbolo, por lo que es necesario determinar un orden de los símbolos dentro del contexto.

3. PPM orientado a palabra (SWPPM)

El factor determinante en este desarrollo ha sido el diseño de una estructura de datos que sea eficiente y compacta, esto es, eficiente para recuperar la información de los contextos y compacta para ser almacenada en disco como parte del texto comprimido final, sin comprometer el rendimiento del compresor.

PPM utiliza contextos de orden $0, ..., n$, por lo que necesita tener recopilada toda esta información durante la compresión. En un compresor orientado a caracteres, el número de contextos diferentes está relativamente acotado, debido a que el número de símbolos de entrada está limitado a 256 caracteres. Sin embargo, utilizar palabras aumenta este número en varios órdenes de magnitud.

La familia de compresores PPM sigue una aproximación dinámica. En un compresor orientado a palabras, el tiempo de adaptación inicial sería demasiado elevado, debido al gran número de contextos diferentes que existen. Es por esto que se ha implementado un compresor semiestático. En una primera pasada sobre el texto se recopila toda la información sobre los contextos y sus probabilidades, utilizando toda esta información en una segunda pasada para codificar el texto.

3.1. Estructura del compresor

El proceso de compresión se divide en dos fases principales:

Cálculo de probabilidades Se realiza una primera pasada sobre el texto, calculando las frecuencias de todas las palabras. Como se trata de un modelo orden-n, las probabilidades estarán condicionadas por las $0, ..., n$ palabras precedentes.

Compresión En una segunda pasada sobre el texto, se utilizan los contextos y símbolos extraídos para comprimir el texto, utilizando siempre, al codificar un símbolo, el contexto de mayor orden del que se disponga información. Además, el texto comprimido final debe incluir los contextos, símbolos y frecuencias que se han utilizado, para poder realizar posteriormente la descompresión.

En una primera fase del desarrollo se decidió probar los resultados de compresión obtenidos almacenando todos los contextos encontrados, pero no se obtuvieron resultados competitivos. Por lo tanto, se decidió aplicar un *filtro de contextos* tras el cálculo de los modelos completos, eliminando aquellas cadenas de texto con una baja frecuencia, ya que previsiblemente no se utilizarán un gran número de veces. Así, el coste de almacenarlos en el texto comprimido final puede no ser compensado con el beneficio en la compresión que proporcionan. Es necesario determinar una frecuencia límite que marque qué elementos se eliminarán, denominada *umbral*. Su valor se determinará experimentalmente (ver Sección 5).

Tras realizar el filtrado, el proceso para comprimir consiste en analizar el contexto de longitud n de la palabra a codificar. Si no se encuentra información sobre este contexto, se desciende al orden inmediatamente inferior (consultando las $n - 1$ palabras anteriores), y se repite el proceso. Si por el contrario se encuentra el contexto pero no se dispone información sobre el símbolo actual, se emite un símbolo de escape y se desciende del mismo modo al orden inferior para volver a realizar la comprobación con un contexto de la longitud inmediatamente inferior. Por último, si el símbolo aparece en este contexto, se utiliza el codificador aritmético proporcionándole la probabilidad de la palabra en su contexto.

Nótese que a pesar de tratarse de un compresor semiestático, sigue siendo necesaria la emisión de símbolos de escape, ya que al haber eliminado los símbolos que aparecen un número reducido de veces en el contexto, puede ser necesario descender de orden, e indicarle al descompresor este hecho. Por tanto, es necesario que el símbolo de escape disponga de una frecuencia asignada en ese contexto.

Una vez se ha comprimido el texto, es necesario representar los contextos y sus frecuencias de una forma compacta, ya que es necesario que formen parte del texto comprimido final para permitir la posterior descompresión.

3.2. Estructura del descompresor

El proceso de recuperación del texto original a partir del texto comprimido se realiza de forma simétrica al proceso de compresión. El cálculo de probabilidades se simplifica, puesto que sólo se han de recuperar los contextos que el compresor ha producido como salida en la compresión. Una vez se dispone de toda esta información, descomprimir el texto supone básicamente realizar un proceso análogo al de la compresión, descodificando siempre en el contexto de mayor orden posible, y si el símbolo descodificado se trata de un símbolo de escape, descendiendo de orden y repitiendo el mismo proceso.

4. Diseño de estructuras para SWPPM

4.1. Almacenamiento de contextos

Toda la información sobre los modelos de orden $0, ..., n$ debe ser almacenada en una estructura que permita un rápido acceso durante la fase de cálculo de contextos y de compresión. En el almacenamiento de contextos, se precisa conocer si una secuencia de palabras (contexto y la palabra actual) ya existe en la estructura, para modificar su frecuencia o, en caso contrario, para insertarla con frecuencia 1.

Al igual que PPM orientado a caracteres, SWPPM también utiliza codificación aritmética. Así, para calcular el subintervalo que le corresponde al símbolo actual y poder codificarlo, se necesita obtener la *frecuencia absoluta del contexto*, la *frecuencia del elemento en ese contexto* y la *frecuencia acumulada inferior* a ese símbolo, esto es, de los otros símbolos anteriores en el contexto actual. Para ello, es preciso determinar un orden para los símbolos del mismo contexto, y tener acceso a ellos para poder sumar sus frecuencias y así calcular su frecuencia acumulada inferior. Estos tres valores son los que requiere el codificador aritmético para procesar el símbolo.

Se ha optado por almacenar los contextos en una *tabla hash*, donde cada elemento contiene la información de una cadena de texto, que está formada por un contexto y un símbolo que le sigue, con su frecuencia. Un contexto de k palabras se almacena recursivamente como una referencia al contexto de $k-1$ palabras y el símbolo correspondiente. El símbolo es una referencia a la palabra en el *vocabulario* almacenado de forma independiente.

Para el cálculo de la frecuencia inferior es necesario disponer de información que relacione entre sí los elementos que comparten el contexto. Para ello, cada elemento contiene otras dos referencias a otros elementos de la tabla: *SiguienteContexto* y *PrimerDescendiente*. De esta forma, se implementa un árbol de contextos sobre la tabla hash. La función hash se calcula a partir de las palabras que forman el contexto y la palabra que sigue a dicho contexto.

4.2. Filtro de contextos

Se descartan los contextos que no superen un determinado umbral de frecuencia, de forma que no serán utilizados para la compresión. Al eliminar un

contexto, también se tienen que eliminar los contextos que desciendan de él en el árbol de contextos. Nótese que esto es consistente, pues si un contexto tiene una frecuencia menor al umbral, necesariamente los contextos que desciendan de él serán poco frecuentes.

Un aspecto crítico en la eliminación de contextos consiste en mantener el árbol de contextos correctamente enlazado. El proceso de eliminación de contextos se hará de la siguiente forma:

- **Paso 1:** Se asigna el valor 0 a la frecuencia de aquellos elementos que no superen el umbral, siempre que no sean de orden 0 (que representa la frecuencia total de cada palabra independientemente de su contexto), ya que se precisa un orden en el que los elementos se puedan codificar con total seguridad (como se verá en el proceso de compresión).
- **Paso 2:** Se restauran los punteros *PrimerDescendiente*. Si apuntan a una fila cuya frecuencia sea 0, se recorre la lista de ese contexto mediante las referencias de *SiguienteContexto* hasta que se encuentre una fila cuya frecuencia sea distinta de 0, asignando la referencia de esa fila a *PrimerDescendiente*.
- **Paso 3:** Se restauran los punteros *SiguienteContexto* del mismo modo que se realizó con *PrimerDescendiente*.

Para permitir posteriormente un almacenamiento compacto de estas estructuras, resulta conveniente que la lista de elementos que comparten contexto (representada por punteros *PrimerDescendiente* y *SiguienteContexto*) esté ordenada alfabéticamente según las palabras que representan. Esta operación modifica solamente los valores de los campos *PrimerDescendiente*, *SiguienteContexto* y la referencia a la palabra, puesto que también se reordena el vocabulario. El proceso consta de varias fases:

Ordenación del vocabulario: en esta fase se reordena el vocabulario alfabéticamente y se proporciona un vector de *equivalencias* para poder realizar posteriormente la actualización de las referencias a la palabra.

Ordenación de las listas: una vez ordenado el vocabulario, el proceso de ordenación de las listas se vuelve sencillo. Para cada elemento, se sigue su lista accediendo a su campo *PrimerDescendiente* y posteriormente utilizando los enlaces con *SiguienteContexto*, almacenando los valores de palabra que se vayan procesando. Dado que el vocabulario está almacenado ahora alfabéticamente, ordenar los valores de palabra consiste en ordenar numéricamente ese vector. Una vez realizada esta ordenación, se recorre de nuevo la lista asignando por orden los valores de palabra encontrados a las referencias *SiguienteContexto*.

Al comenzar a realizar el procesamiento del texto, no se conoce el número de elementos que va a contener la *tabla de contextos*. Una estrategia simple consiste en crear una tabla que ocupe la mitad de la memoria disponible. Una vez calculados los contextos, y descartados los que aparecen pocas veces, se puede reajustar a un tamaño apropiado (por ejemplo a dos veces la cantidad de contextos).

4.3. Compresión

En este proceso, con toda la información disponible de la primera pasada sobre el texto, se realiza una segunda pasada en la que se codifica cada símbolo mediante el contexto de mayor orden posible, emitiendo un símbolo de escape si no se dispone información de los órdenes superiores.

En el proceso de codificación se utiliza el contexto de mayor longitud del que se disponga de información. En el caso de que no aparezca este contexto seguido del símbolo procesado, se enviará un símbolo de escape. Esta emisión puede ser muy elevada si se eliminan muchos elementos de la tabla de contextos (umbral alto). A la hora de la codificación, el codificador aritmético manipula este símbolo del mismo modo que las palabras. La frecuencia asignada al símbolo de escape influye en la frecuencia total del contexto, con lo que asignar una frecuencia óptima a dicho símbolo repercutirá en la eficacia del compresor.

Para ello, hemos estudiado dos alternativas. La primera consiste en asignar al símbolo de escape una frecuencia dada como porcentaje de la suma de las frecuencias de los símbolos existentes para ese contexto (*porcentaje del intervalo*). La segunda opción asigna al símbolo de escape la suma de todas las frecuencias de símbolos que hayan sido eliminados en un determinado contexto, ya que ésta será una medida bastante aproximada de las veces que el símbolo de escape se emitirá (*frecuencias eliminadas*). Experimentos preliminares muestran que con esta última aproximación se obtienen unos mejores resultados de compresión, y será la que utilicemos en la Sección 5.

Para calcular la frecuencia acumulada inferior de cada símbolo, es necesario recorrer la lista de símbolos de un contexto, formada por la referencia *Primer-Descendiente* del contexto y *SiguienteContexto* de los elementos que lo siguen. Este proceso ralentiza la compresión, por lo que se realizó una mejora consistente en calcular previamente las frecuencias acumuladas inferiores de los símbolos, así como la de los símbolos de escape. De este modo, recorriendo cada lista una sola vez se almacenan todos los valores necesarios, permitiendo en el proceso de compresión realizar la codificación accediendo directamente al elemento necesario y al elemento de su contexto.

4.4. Compresión de las estructuras auxiliares

El compresor y el descompresor precisan disponer del conjunto de elementos que serán utilizados para realizar la compresión y la descompresión. Así, partiendo de la tabla de contextos creada, es necesario transformar esa misma información de una forma que ocupe el menor espacio posible en disco.

El proceso consiste en dividir dicha tabla en vectores siguiendo una aproximación diferencial (es decir, se almacena la diferencia con respecto al valor inmediatamente anterior) que da lugar a valores bajos y repetitivos, y a continuación aplica compresión Huffman, sobre las diferencias. Para ello, se precisa tener ordenadas alfabéticamente las listas de elementos que comparten el contexto.

Estructura intermedia En primer lugar, se precisa transformar la tabla a una estructura intermedia que facilite la conversión a los vectores finales. Dicha estructura contendrá los elementos (contextos y las palabras que los siguen) ordenados según el orden del contexto y posteriormente por orden alfabético. Esta estructura está formada por varios campos:

- *Índice*: indica en qué fila de la estructura original se encuentra el elemento.
- *Contexto*: indica en qué fila de esta estructura intermedia está el contexto del elemento. Para los elementos de orden 0, este elemento apunta al vocabulario. Para los demás, este valor hace referencia a un elemento de orden inmediatamente inferior.
- *Palabra*: indica en qué fila de esta estructura está la palabra del elemento. En el caso de los elementos de orden 0 este valor es nulo, y en el resto de los casos, hace referencia a un elemento de orden 0.

Vectores finales Con el apoyo de esta estructura, se calculan los vectores que posteriormente serán comprimidos con Huffman. Tal y como se ha mencionado previamente, se utiliza una aproximación incremental. Se debe calcular:

- *LongitCont*: longitud del vector *Contextos*.
- *TotalSeg*: la posición i indica el número de elementos en el vector *Segundos* que hay de orden $0, ..., i-1$.
- *Contextos*: para cada elemento de la estructura intermedia de orden superior a 0, indica la fila de dicha estructura en la que se encuentra el contexto de ese elemento. Las posiciones son relativas, es decir, si el contexto es de orden n, el número indica el incremento de posición desde el primer elemento de orden n que hay en la fila intermedia. Esta estructura sólo almacena los contextos diferentes.
- *Numveces*: número de descendiente del contexto que ocupa esa misma posición en *Contextos*.
- *Segundos*: indica las palabras que suceden al contexto definido por *Contextos* y *NumVeces*. Para cada contexto, el primer elemento en este vector se corresponde con la posición de la palabra en la estructura intermedia en el orden 0. Las siguientes palabras con el mismo contexto se codifican con la aproximación incremental. Esto produce números relativamente y bajos y con un mayor número de repeticiones, debido a que las cadenas de texto están en orden alfabético, y la mayoría de los elementos de este vector son la distancia entre elementos próximos de orden 0.
- *Frecuencias*: frecuencia de todos los elementos de la estructura. La frecuencia se almacena para los elementos de todos los órdenes, incluido el orden 0, por lo que la longitud de este vector es $TotalSeq[n+1]$.

En la Figura 1 se muestra el proceso de compactación de las estructuras auxiliares que posteriormente serán comprimidas usando Huffman.

En el conjunto de vectores finales de orden 0 sólo se almacenan las frecuencias. Esto es posible porque en el orden 0 se almacenan todas las palabras por orden alfabético, y esta ordenación ya se mantiene en el vocabulario, por lo que no es

	ÍND.	CONT.	PALAB.	
A	8	0	Ø	*ORD. 0*
B	17	2	Ø	
C	7	4	Ø	
D	47	6	Ø	
A A	5	0	0	*ORD. 1*
A C	25	0	4	
B A	27	2	0	
B B	15	2	2	
B C	56	2	4	
AA B	2	4	2	*ORD. 2*
AA D	9	4	6	
BB A	13	7	0	
BB D	21	7	6	
BC D	58	8	6	

LON PRIM:	5											
TOTALSEG	0		4		9		14					
	Ord 0		Ord 1		Ord 2		Ord 3					
CONTEXTOS	0		1		0		3		1			
	Ord 1				Ord 2							
NUMVECES	2		3		2		2		1			
	Ord 1				Ord 2							
SEGUNDOS	0		2		0		1		1	1	2	0	3	4
	Ord 1									Ord 2				
FRECUENCIAS	30		20		20		25		12	15	7	16	8	6	5	5	6	5
	Ord 0									Ord 1					Ord 2			

Figura 1. Ejemplo de compactación de las estructuras auxiliares

necesario volcar en disco ninguna información más sobre los elementos de este orden para poder reconstruirlos posteriormente.

Tras obtener los vectores finales que han de ser volcados a disco, un último paso previo al volcado consiste en comprimir los vectores de mayor longitud (*Contextos, NumVeces, Segundos* y *Frecuencias*) aplicando Huffman a cada uno de los vectores independientemente. Así, para cada vector se calcula la frecuencia de cada valor, y con ello se crea el árbol Huffman correspondiente. Con este árbol se codifica el vector y se vuelca a disco, ya comprimido, incluyendo también el árbol de Huffman, necesario para la posterior descompresión.

4.5. Descompresión

El proceso de descompresión se realiza de forma muy similar a la compresión. Una vez realizada la recuperación de contextos, se dispone de la tabla de contextos y el código para descomprimir. El proceso es el siguiente: tras indicarle al contexto la frecuencia total del mismo, el decodificador aritmético devuelve un número. El intervalo al que pertenezca dicho número indica el símbolo descodificado, y en caso de que se corresponda con un símbolo de escape, se desciende de nivel, indicándole la frecuencia del contexto de orden inferior y comprobando de nuevo a qué subintervalo pertenece el número devuelto por el descodificador.

5. Resultados experimentales

Los experimentos se ejecutaron en una máquina Intel®Pentium®-IV@ 3.00 GHz (16Kb L1 + 1024Kb L2 cache), con 4 GB dual-channel DDR-400Mhz RAM, sobre la que corría Debian GNU/Linux (kernel version 2.4.27). El compilador usado fue *gcc* version 3.3.5. Las pruebas, que miden el ratio de compresión, fueron realizadas sobre un conjunto de textos de la "Text Retrieval Conference"[1] (TREC); de TREC-2 se han utilizado "AP Newswire 1988" y "Ziff Data 1989-1990", y de TREC-4 "Congressional Record 1993", "Financial Times 1991, 1992, 1993 y 1994". A su vez *FTALL* agrega los corpus provenientes del Financial Times. Además, se utilizó el texto *English50* [2], resultado de los primeros 50

[1] http://trec.nist.gov/

[2] Obtenido de http://pizzachili.dcc.uchile.cl/texts.html

MegaBytes del texto *English*, consistente en una concatenación de textos en inglés del proyecto Gutemberg[3]

En un primer lugar, se analizó la influencia del orden máximo utilizado en el compresor. La Gráfica izquierda de la Figura 2 muestra los resultados obtenidos para el texto *English50*. Se muestra el orden máximo frente al ratio de compresión. Cada curva representa los ratios obtenidos para un determinado umbral. Se puede observar que en un texto en lenguaje natural el orden máximo óptimo es 2. Los resultados producidos indican que en un texto en lenguaje natural no es

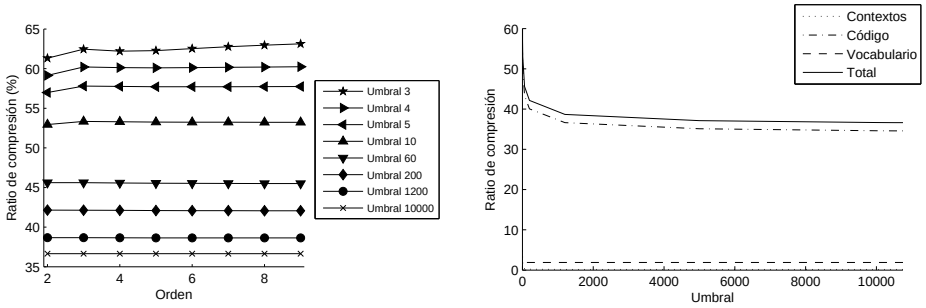

Figura 2. Ratios de compresión para un texto en lenguaje natural con diferentes umbrales y órdenes

muy beneficioso utilizar órdenes máximos elevados. Esto se debe a que utilizar un orden mayor supone almacenar un mayor número de contextos y, para que compense el incremento de almacenamiento producido, la información disponible debe producir una mejora en la compresión que implique un *código* mucho más reducido. Si las frases de esta nueva longitud máxima no se repiten mucho, incrementar este orden supone la mayoría de las veces introducir un símbolo de escape y posteriormente codificar en el orden inferior del mismo modo que se hacía previamente, produciendo un *código* peor debido al coste de emisión del símbolo de escape.

Una vez fijado el orden óptimo, se procedió a analizar la cantidad de contextos que se debían almacenar (determinado por el umbral). La Gráfica derecha de la Figura 2 muestra los ratios de compresión para los órdenes máximos probados empíricamente para el texto *English50*. La gráfica indica en el eje x el umbral y, en el eje y, el ratio de compresión. En ella se muestran cuatro curvas: tres de ellas representan la contribución al ratio de compresión de los *contextos*, del *código* y del *vocabulario*, y una cuarta indica la suma de las tres anteriores, y por tanto, el ratio de compresión final.

Se puede observar cómo el espacio ocupado por los contextos decrece a medida que aumenta el umbral, debido a que ello implica que se almacenen un menor número de contextos, mientras que el vocabulario permanece constante. El tamaño del texto comprimido también desciende a medida que aumenta el

[3] http://www.gutenberg.org/ebooks/

umbral. En principio, se puede pensar que debería aumentar, ya que se dispone de menos información sobre los contextos y símbolos, pero la forma de codificar los símbolos de escape hace que en los umbrales bajos (en los que se eliminan un menor número de contextos), los símbolos de escape tengan una frecuencia más baja asignada, produciendo una peor codificación de estos símbolos que repercute en el texto comprimido final. La suma de las tres curvas se vuelve óptima en los umbrales superiores, por lo que se puede concluir que los resultados óptimos se obtienen para un umbral de 10000 o superior con un orden máximo de 2, resultando un ratio del 36,61 %.

Una vez calculados los valores óptimos para nuestro compresor, comparamos su comportamiento con otros compresores conocidos. Los compresores utilizados en la comparativa fueron el ya mencionado PPMDI, ETDC [2,3], gzip [4] y el compresor bzip2 [5]. PPMDI fue probado con orden máximo 0 y 9. Por otra parte, el compresor gzip se utilizó con las opciones de ejecución más rápida (opción 1) y más eficiente, pero más lenta (opción 9), mientras que el compresor bzip2 se probó con los tamaños de bloques extremos (100k en la opción 1 y 900k en la opción 9 más rápido o mejor compresión respectivamente).

La Tabla 1 muestra los resultados obtenidos. En la primera columna se indica el nombre del corpus, mientras que las restantes muestran los ratios de compresión obtenidos con los diferentes compresores.

Los resultados obtenidos con SWPPM son competitivos, mejorando cualquier ejecución de gzip y de ETDC y con resultados próximos a la ejecución óptima de bzip2.

TEXTO	PPMDI		BZIP2		GZIP		ETDC	SWPPM
	Op. 0	Op. 9	Op. -1	Op. -9	Op. -1	Op. -9		O2,U10000
FT92	29,40 %	23,42 %	32,38 %	27,10 %	42,59 %	36,39 %	32,85 %	29,87 %
ZIFF	26,48 %	21,77 %	39,66 %	32,98 %	39,66 %	32,98 %	33,81 %	31,02 %
FT93	27,66 %	21,68 %	30,62 %	25,32 %	40,23 %	34,12 %	32,91 %	29,09 %
FT94	27,61 %	21,68 %	30,53 %	25,27 %	40,24 %	34,12 %	32,86 %	28,81 %
AP	30,34 %	23,33 %	33,27 %	27,25 %	43,66 %	37,23 %	32,93 %	30,05 %
FTALL	28,20 %	22,24 %	31,15 %	25,98 %	40,99 %	34,85 %	32,56 %	28,37 %

Tabla 1. Comparativa de diferentes compresores para textos semiestructurados

6. Conclusiones

En este trabajo hemos presentado un nuevo compresor PPM orientado a palabras (SWPPM), que se diferencia de las otras versiones de la familia PPM, en la utilización de palabras como símbolos a comprimir en lugar de caracteres.

Tras varios diseños preliminares, se creó una variante que llamamos SWPPM que sólo considera los contextos con una frecuencia superior a un umbral dado. El compresor propuesto fue probado experimentalmente obteniendo ratios de compresión del 28 %-31 %. La comparación frente a otras técnicas existentes

[4] Puede encontrarse en el sitio Web http://www.gzip.org/
[5] Disponible en http://www.bzip.org/

permitió ver que el SWPPM comprime más que técnicas como el *gzip* (en todas sus variantes) y los compresores semi-estáticos orientados a palabras más utilizados en la actualidad (dense codes). En general, el SWPPM comprime un poco menos que el *bzip2*, y se ve superado por el *PPMDI*.

Los resultados prometedores obtenidos dejan la puerta abierta a futuras optimizaciones sobre la base del trabajo realizado, como por ejemplo, la mejora de la compresión de estructuras auxiliares que permiten mantener un mayor número de contextos durante la compresión. También estamos trabajando buscando una reducción del tamaño de los contextos en disco, realizando la compresión final con otras técnicas (ppm, gzip) en lugar de Huffman.

Otra línea de investigación consiste en aprovechar la información que proporciona el hecho de emitir un símbolo de escape para obtener una mejor estimación de las probabilidades usadas al codificar en el orden inferior. Además, el filtro de contextos elimina aquellos elementos con una menor frecuencia porque es probable que se usen menos veces, pero es posible que algunos de los restantes no se utilicen, ya que a los elementos de órdenes inferiores sólo se accede cuando no existe el orden superior. Se propone por tanto realizar una pasada intermedia, previa al proceso de compresión, en la que se simula el proceso de codificación marcando el número de veces que se utiliza cada elemento y los símbolos de escape emitidos en cada contexto. Así, se ahorraría el espacio de almacenamiento de los elementos y se ajustarían mejor las frecuencias de los símbolos.

Referencias

1. T. C. Bell, I. H. Witten, and J. G. Cleary. *Text compression.* Prentice Hall, Englewood Cliffs, N.J., 1990.
2. N. R. Brisaboa, A. Fariña, G. Navarro, and J. R. Paramá. Lightweight natural language text compression. *Information Retrieval*, 10(1):1–33, 2007.
3. N. R. Brisaboa, E. L. Iglesias, G. Navarro, and J. R. Paramá. An efficient compression code for text databases. In *Proc. 25th European Conf. on IR Ressearch (ECIR'03) - LNCS 2633*, pages 468–481, 2003.
4. J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32:396–402, 1984.
5. D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
6. A. Moffat. Word-based text compression. *Softw. Pract. Exper.*, 19(2):185–198, 1989.
7. A. Moffat, R. Neal, and I. H. Witten. Arithmetic coding revisited. In *Proc. IEEE Data Comp. Conf.*, pages 202–211. IEEE Computer Society Press, 1995.
8. A. Moffat and A. Turpin. *Compression and coding algorithms.* Kluwer Academic Publishers, Boston, London, 2002.
9. Turpin A. Moffat, A. Fast file search using text compression. In *Proc. 20th Australian Computer Science Conference*, pages 1–8, 1997.
10. E. Moura, G. Navarro, N. Z., and R. Baeza-Yates. Fast and flexible word searching on compressed text. *ACM TOIS*, 18(2):113–139, 2000.
11. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.