# ANÁLISIS EMPÍRICO DEL EFECTO DEL TAMAÑO DE LA INFORMACIÓN DE ENTRADA EN EL DESEMPEÑO DE HERRAMIENTAS DE COMPRESIÓN SIN PÉRDIDA*

*Miguel Eduardo Torres Moreno***

*Germán Flórez Larrahondo****

**Resumen**: Este artículo presenta un estudio empírico del efecto que tiene el tamaño de la información de entrada en el desempeño de algoritmos de compresión sin pérdida. Se analizan tres medidas diferentes de desempeño y se crea un nuevo grupo de archivos basado en los *corpus* de Calgary y Canterbury. Este grupo de archivos también incluye dos archivos complejos. Se demuestra que para archivos grandes la tasa de compresión de algoritmos sin pérdida se mantiene relativamente constante y luego cambia por un pequeño factor cada 10 MB de información. Finalmente, se muestra que el tiempo de ejecución del proceso de compresión y descompresión es una función lineal basada en el tamaño del archivo de entrada.

**Palabras clave**: compresión de datos, algoritmos de compresión sin pérdida, desempeño de algoritmos.

**Abstract**: This paper presents an empirical study of the effect that different input sizes have on the performance of lossless data compression algorithms. We analyzed three different performance measures and created a new dataset based on the Calgary and Canterbury corpus. This dataset also includes two new "complex" files as well. We demonstrated that for large files the compression ratio of the lossless algorithms stays fairly constant and only changes by a small factor every 10MB. Finally, we have shown that the execution time for

* *Fecha de recepción: 13 de abril de 2004. Fecha de aceptación para publicación: 22 de julio de 2004.*

** *Ingeniero de sistemas, Universidad Nacional de Colombia. Master of Science, Mississippi State University. Profesor asistente, Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana. Correo electrónico: metorres@javeriana.edu.co*

*** *Ingeniero de sistemas, Universidad Nacional de Colombia. Master of Science y Candidato a Ph. D. en Computer Science, Mississippi State University Correo electrónico: gf24@cse.msstate.edu*

compressing and Decompression data is a linear function based on the size of the input.

**Keywords**: *Data compression, lossless algorithms, algorithm's performance.*

## INTRODUCCIÓN

La compresión de datos sin pérdida ha demostrado ser un mecanismo útil para mejorar el almacenamiento y la transmisión de información, ya que permite que la información comprimida pueda ser recuperada sin ningún tipo de error o modificación. Generalmente este tipo de compresión puede ser logrado mediante dos métodos diferentes [Bell *et al.*, 1989]: método de diccionario y método de codificación estadística. En el primero se almacena un historial de patrones bajo la suposición de que los símbolos recientemente observados pueden encontrarse nuevamente en las cadenas producidas de la información introducida que se está evaluando. En el segundo se crea un modelo estadístico de la información de entrada, y la información es comprimida de acuerdo con la probabilidad de ocurrencia de un símbolo en el modelo.

Un gran número de algoritmos de compresión sin pérdida —o también conocidos como algoritmos de compresión reversible— han sido publicados y muchos de ellos se han usado con éxito en la industria. Por ejemplo, herramientas como *compress* y *pack* fueron incluidas en las primeras versiones del sistema UNIX, para ejecutar codificación por los métodos de diccionario y estadística respectivamente.

Se han llevado a cabo muchos esfuerzos investigativos para contrastar y comparar los diferentes algoritmos [Arnold y Bell, 1997, y Bell *et al.*, 1989], sin embargo, no fue posible verificar la existencia de alguna investigación que relacione el tamaño y el tipo de mensaje[1] con el desempeño de la herramienta de compresión cuando el tamaño del mensaje es considerablemente grande (por ejemplo, 20 MB). Por esta razón, el propósito de esta investigación es realizar un análisis empírico del impacto de la información de entrada en el desempeño de herramientas de compresión sin pérdida. Como medidas de desempeño se han escogido la tasa de compresión —definida como la relación entre la longitud de salida y la longitud de entrada—, el tiempo real que se toma la herramienta para comprimir un archivo y el tiempo real que se toma la herramienta para descomprimirlo.

La primera parte de este artículo contiene una breve revisión de antecedentes de la bibliografía sobre este tema, la segunda presenta la metodología escogida para determinar cómo el desempeño de una herramienta de compresión se relaciona con la longitud del archivo de entrada, la tercera describe los experimentos llevados a cabo y, finalmente, en cuarta parte, se concluye.

1 *En este artículo las palabras* mensaje *y* archivo *son usadas indistintamente.*

## 1. Revisión de antecedentes en la bibliografía especializada

Hay extensas investigaciones en la comparación de algoritmos de compresión al usar diferentes medidas de desempeño. Las medidas más usadas son la tasa de compresión [Arnold y Bell, 1997; Bell *et al.*, 1989, y Cho y Cho, 1995], el tiempo consumido en realizar la operación [Cho y Cho, 1995, y Jones, 1991] y la velocidad de compresión (*kilobytes* por segundo) [Williams, 1991]. Otra medida interesante de desempeño es descrita en Jones [1991], que dice que el tiempo de compresión (representado en ciclos de reloj por bit) se representa como una función de la tasa de compresión.

Se han usado dos grupos de archivos para medir el desempeño de diferentes herramientas de compresión: los corpus de Calgary y los de Canterbury. El corpus de Calgary (1987) [Bell *et al.*, 1989] fue el primer grupo de archivos usados como un estándar de facto para probar algoritmos de compresión. El corpus de Canterbury [Arnold y Bell, 1997] apareció en 1997 como reemplazo del corpus de Calgary con un conjunto adicional de archivos, aunque, como indican Arnold y Bell [1997], el corpus de Calgary es aún un buen indicador de desempeño de la compresión.

Mano y Sato [1991], Cho y Cho [1995], Bender y Wolf [1990], entre otros, describen el impacto de la tasa de compresión debido al incremento en el tamaño del mensaje. En el trabajo de Arnold y Bell [1997], por ejemplo, los resultados de los experimentos demostraron que la tasa de compresión de los algoritmos LZ y LZW (codificación por el método de diccionario) se incrementa rápidamente cuando *log(n)* (*n* es el tamaño del mensaje) se incrementa, pero permanece constante cuando *log(n)* es mayor que doce (por ejemplo, 4.096 *bytes*). Sin embargo, la forma general de la función tasa de compresión frente a *log(n)* depende del tipo de archivo. Esta tasa de compresión constante puede también verse en los resultados de los experimentos con archivos de texto realizados por Cho y Cho [1995], donde se compararon métodos de diccionario usando cuatro tamaños diferentes de archivos hasta de 175 kb. Se encuentra que incluso luego de 100 kb nuevamente la tasa de compresión se mantiene constante.

Debido a restricciones tecnológicas, en algunos esfuerzos de investigación previos [Cho y Cho, 1995; Jianzhong y Srivastava, 2002, y Williams, 1991] se consideraron archivos de tamaños relativamente pequeños. Sin embargo, hasta el momento de realización del presente estudio no fue posible encontrar información de investigaciones elaboradas para comparar herramientas de compresión con archivos de un tamaño considerable.

## 2. Metodología utilizada

Con el fin de realizar un conjunto adecuado de experimentos que permitan conocer el impacto del tamaño de la información de entrada en el desempeño de herramientas de compresión sin pérdida se preci-

sa resolver preguntas como ¿qué herramientas de compresión se deben considerar?, ¿qué grupo de archivos de prueba utilizar?, y ¿cuál es el desempeño de una herramienta de compresión?

## 2.1 Herramientas de compresión

Como se mencionó anteriormente, existen los métodos estadísticos y los métodos de diccionario. Los métodos estadísticos de compresión asumen cierto grado de conocimiento de la información que va a ser comprimida, identificando la probabilidad de aparición de un símbolo en el mensaje, asignando códigos cortos de representación a símbolos de aparición altamente probable y códigos largos de representación para los símbolos menos probables [Bell *et al.*, 1989]. Las herramientas de compresión de esta categoría que se van a probar son:

- *Pack (SunOS versión 5.7)*: compresor UNIX que implementa el algoritmo semiadaptativo de Huffman, almacena las probabilidades como enteros de 32 bits, y los códigos de salida están limitados 24 bits.
- *Arithmetic Coding*: implementado por Moffat [2004], efectúa un método de compresión de caracteres por omisión con un tamaño de código de 32 bits y por omisión el número de bits usado para conteos de frecuencia es 27.

Los métodos de diccionario usan información histórica del mismo mensaje de entrada para construir un diccionario de cadenas comunes, a fin de mejorar la compresión. Aproximaciones similares usan un *buffer* de vista adelante (*look-ahead buffer*) con el fin de evitar almacenar información adicional. Las herramientas de compresión que implementan este tipo de algoritmos son y serán utilizadas en este estudio son:

- *Compress (SunOS versión 5.7):* implementa un algoritmo LZW con tamaño de palabra por omisión de 16 bits.
- *gzip (Versión 1.2.4)*: implementa una variante del algoritmo LZ77. Específicamente para esta versión el diccionario almacena en un *buffer* hasta 32 kb (32.768 *bytes*) y en un *buffer* de vista adelante 256 *bytes*. El código fuente se encuentra disponible en la dirección ftp://wuarchive.wustl.edu/mirrors/gnu/gzip/gzip-1.2.4.tar.gz.
- *LHA*: implementa un algoritmo LZ77 con una ventana de vista adelante de 258 bits. Usa una tabla Hash y árboles binarios, además de codificación Huffman secundaria por bloques. El código fuente se encuentra disponible en la dirección ftp://garbo.uwasa.fi/unix/arcers/lha101u.tar.Z

## 2.2 Grupos de archivos

Los tamaños de los archivos contenidos en los corpus de Calgary y los de Canterbury son muy pequeños para los propósitos de este trabajo. Con el fin de superar esta limitación se decidió crear un grupo propio de archivos basado en los ya existentes en los corpus de Canterbury y los

de Calgary. Adicionalmente, se incluyeron dos archivos 'grandes', representativos de archivos 'complejos'. La creación de este grupo de archivos fue un proceso desarrollado en dos partes. Primero se seleccionaron cuatro tipos principales de archivos que pueden ser considerados de alta ocurrencia [Bell *et al.*, 1989]: binario, texto en inglés, código fuente y otro tipo de archivos. A continuación se presenta una corta descripción de cada tipo de archivo identificado:

- Binarios: archivos que contienen código de máquina, formatos propietarios (por ejemplo, Microsoft ® Excel), imágenes, etc.
- Textos en inglés: archivos de texto que contienen frases en inglés (por ejemplo, alfabeto latino), incluido inglés técnico e inglés común.
- Textos no inglés: archivos que no solamente contienen alfabetización latina, sino también símbolos especiales. Ejemplos de ellos incluyen páginas HTML, páginas *man* de UNIX, código fuente C o LISP, entre otros.
- Otros archivos: aquellos que no encajan en las descripciones anteriores. La mayoría de ellos incluye información sintética y secuencias largas. Un ejemplo típico de este tipo de archivo sería una secuencia de ADN.

Para cada grupo se seleccionaron los archivos apropiados de los corpus de Calgary y los de Canterbury (véase Tabla 1) y se unieron usando el comando *cat* de UNIX hasta generar un solo archivo de 90 MB.[2] A continuación, se dividió el archivo usando el comando *split* y se crearon muestras que varían entre 1 kb y 90 MB. Sin embargo, el interés principal radica en archivos de tamaños mayores a 1MB, puesto que se conoce que para archivos pequeños el desempeño de las herramientas de compresión no sólo se ve afectado por la eficiencia del algoritmo, sino también por el cálculo adicional que representa identificar, por ejemplo, el tipo de archivo y los parámetros de compresión que se van a utilizar. Además, al usar archivos grandes se considera que se pueden pasar por alto detalles de implementación como la cantidad de memoria estática usada por cada herramienta. Es importante anotar que una aproximación sencilla para la creación del grupo de archivos podría consistir en generar un archivo pequeño (de por ejemplo 20 kb) y de éste crear todas las muestras. Sin embargo, se considera que la aproximación escogida le da más variedad a cada muestra.

En la segunda parte se buscaron nuevos archivos grandes para incluirlos en el grupo de archivos. Como se explicó, se espera que todos los archivos en un grupo sean muy similares. Sin embargo, las herramientas de compresión sin pérdida son usadas también diariamente para comprimir archivos con distribuciones altamente complejas de símbolos. Se tomaron como ejemplos[3] de este tipo de archivos el direc-

*2 Debido a restricciones de uso en los equipos utilizados para los experimentos se decidió trabajar con archivos de máximo 90 MB.*

*3 No se llevó a cabo ningún tipo de análisis formal para seleccionar dichos archivos; ellos fueron seleccionados según la experiencia de los autores.*

Tabla 1. Distribución de archivos de los *corpus* de Canterbury y de Calgary en el grupo de archivos propuesto

| Binario | No inglés | Inglés | Otro |
|---|---|---|---|
| Geo | bib | Alice29.txt | e.coli |
| Kennedy.xls | Cp.html | Asyoulik.txt | a.txt |
| Obj1 | Fields.c | Bible.txt | Aaa.txt |
| Obj2 | Grammar.lsp | Book1 | Alphabet.txt |
| Pic | Progc | Book2 | Pi.txt |
| Ptt5 | Progl | Icet10.txt | Random.txt |
| Sum | Trans | News | |
| | Xargs.1 | Paper1 | |
| | | Paper2 | |
| | | Paper3 | |
| | | Paper4 | |
| | | Paper5 | |
| | | Paper6 | |
| | | Plrabn12.txt | |
| | | World192.txt | |
| 2,36 MB | 384 kb | 9,33 MB | 5,67MB |

*Fuente*: Arnold y Bell [1997] y Bell *et al.* [1989].

torio */usr* de un sistema *Linux Mandrake* y el volcado de tráfico de una tarjeta de red. El primer archivo fue generado usando la utilidad *tar* de Linux. El archivo contiene cientos de archivos binarios, archivos de código fuente PERL, *scripts* de *shell*, páginas *man* de UNIX, etc. El segundo archivo fue tomado de MIT Lincoln Labs. [2004] y contiene tráfico simulado TCP/IP, que incluye transferencias FTP de diferentes tipos de archivos, correo electrónico, mensajes HTTP, etc. Ambos archivos fueron partidos para generar muestras desde 1 kb hasta 90 MB.

### 2.3 Medidas de desempeño

Como medidas de desempeño se utilizaron la tasa de compresión, el tiempo real (*real time*) tomado por las herramientas para comprimir un archivo y el tiempo real para descomprimirlo. Aunque también se midió el tiempo de usuario (*user time*) y el tiempo del sistema (*system time*) usando el comando time de UNIX, fue claro desde los experimentos preliminares realizados que la máquina usada consume la mayoría de ciclos de CPU ejecutando los algoritmos de compresión, por lo que el tiempo real es aproximadamente igual al tiempo de usuario más el tiempo del sistema.

## 3. Resultados experimentales

Los experimentos fueron realizados en la máquina *fantasia.cse.msstate.edu*, ubicada en el *Department of Computer Science* de la *Mississippi State University*, con sistema operativo SunOS, versión 5.7, con cuatro procesadores SPARC, cada uno corriendo a 296 MHz. El

grupo de archivos fue dividido en seis grupos principales para un total de 72 archivos. Los resultados reportados para el tiempo de compresión y descompresión corresponden a un promedio de tres ejecuciones por cada herramienta.

### 3.1 La tasa de compresión

Las figuras 1 a 4 muestran la tasa de compresión como función de los tamaños de archivos binarios, texto en inglés, texto no inglés y otros.

En los tres primeros grupos de archivos la tasa de compresión alcanzada en la compresión de archivos pequeños (1 kb, 100 kb y 1 MB) mejora al incrementarse el tamaño del archivo. En todos los experimentos se puede observar que la tasa de compresión parece mantenerse constante cuando el tamaño del mensaje de entrada se incrementa (tamaño >10 MB). Sin embargo, como se muestra en la Tabla 2, mientras la tasa de compresión para archivos binarios y no inglés mejora por un pequeño factor (aproximadamente entre 1% y 2%) por cada 10 MB, la tasa de compresión para el grupo de archivos en inglés y otros archivos se incrementó (entre un 1% y un 5%). En promedio, los algoritmos estadísticos (*arithmetic compression* y el código Huffman) mantuvieron una tasa constante para los grupos de archivos en inglés y no inglés. Vale la pena mencionar que al comprimir archivos grandes con estos algoritmos resulta una tasa pobre de compresión en comparación con los otros algoritmos usados.

Tabla 2. Cambio promedio de la tasa de compresión cada 10MB (para archivos > 10MB)

| Herramienta | Binarios | Inglés | No inglés | Otros archivos |
|---|---|---|---|---|
| ./arithcompress.sh | 0,98 | 1 | 1 | 1,05 |
| ./lahcampress.sh | 0,97 | 1,01 | 0,99 | 1,01 |
| compress | 0,97 | 1 | 0,99 | 1,01 |
| gzip | 0,97 | 1,01 | 0,99 | 1,01 |
| pack | 0,98 | 1 | 1 | 1,04 |

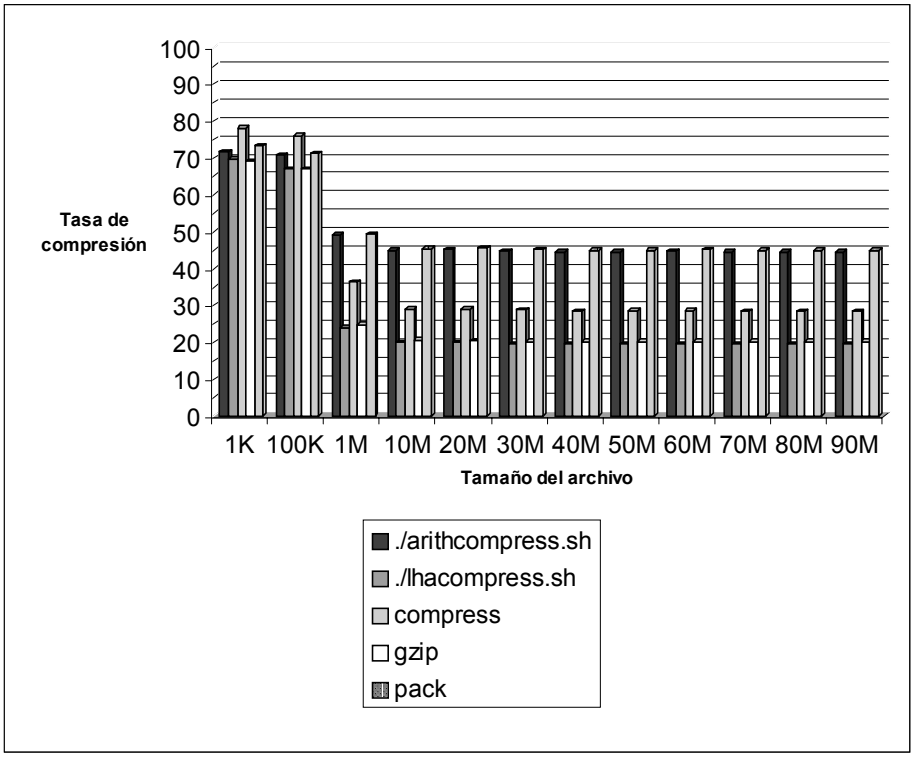
*Fuente*: elaboración propia.

Nota: cada celda indica la cantidad de cambio de la tasa de compresión por cada 10 MB de la entrada. Por ejemplo, la primera celda (*arithmetic compression* para el grupo de archivos binarios) indica que si la tasa de compresión de un archivo de 10 MB es λ, la tasa esperada de compresión para el archivo de 20 MB es (0,98) λ, la tasa de compresión esperada para un archivo de 30 MB es (0,98)(0,98)λ=(0,9604)λ, y así sucesivamente. Esto es, 0,98 indica un mejoramiento en el desempeño del 2%, mientras que 1,01 indica una desmejora del 1%.

En el grupo de archivos binario (Figura 1), todas las herramientas lograron una tasa de compresión pobre (más del 70%) con archivos pequeños (menos de 1 MB), comparado con los otros grupos de archivos. Esto da la idea de que esos archivos binarios contienen una baja cantidad de cadenas similares y que la probabilidad de ocurrencia de

los símbolos es muy uniforme. Sin embargo, cuando el tamaño del archivo se incrementa a 10 MB, se puede observar un mejoramiento drástico en el desempeño del algoritmo.

Figura 1. Tasa de compresión para el grupo de archivos binarios

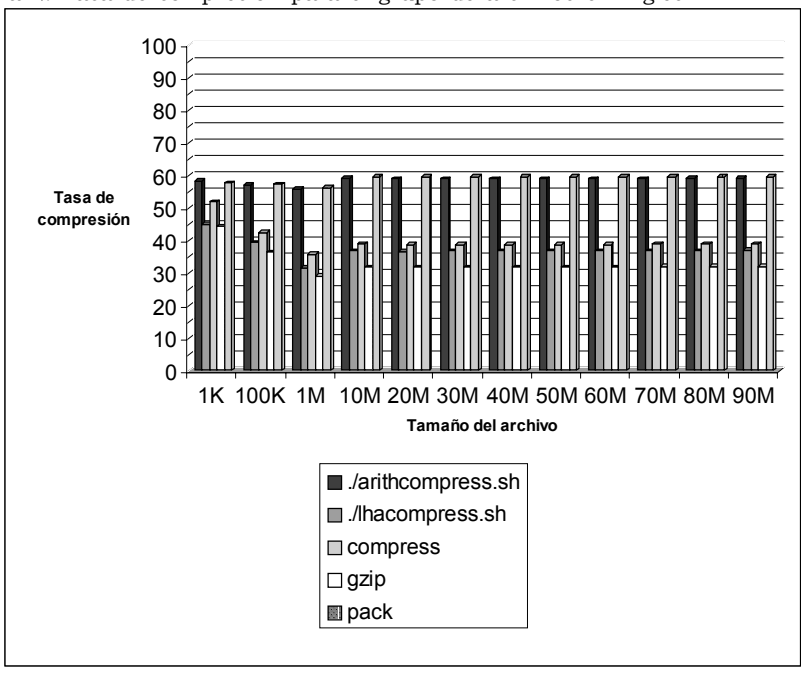



*Fuente*: elaboración propia.

Los resultados del experimento con archivos de texto en inglés y archivos de texto no inglés (figuras 2 y 3) fueron bastante similares, sólo hubo un pequeño incremento en la tasa de compresión cada 10 MB para los de texto en inglés usando *LHA* y *gzip* (véase Tabla 2). Sin embargo, es posible que la diferencia en la compresión total para cada grupo de archivos se deba al hecho de que los experimentos basados en texto en inglés contienen un pequeño conjunto de caracteres que incluye alfabeto romano y símbolos de puntuación. Por el contrario, el grupo de archivos no inglés no solamente contiene alfabeto romano, sino también una gran variedad de símbolos especiales, como caracteres reservados de lenguaje C o PERL.

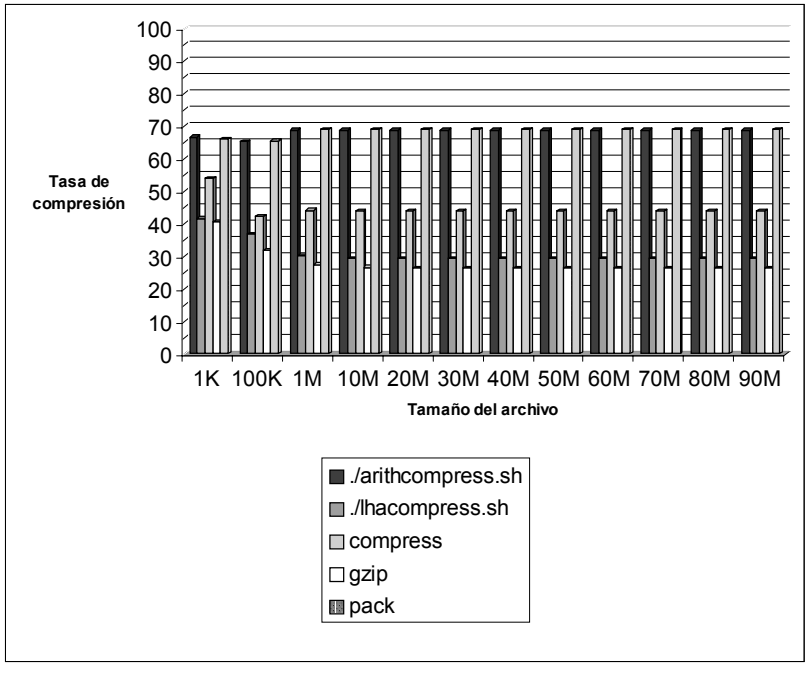
En el grupo de otros archivos (Figura 4), el desempeño de las herramientas decreció en un factor pequeño del 1% al 5% (véase Tabla 2). Es importante anotar cómo la tasa de compresión para todos los tamaños de archivos usando *arithmetic coding* y *pack* fue significativamente mejor que en experimentos previos. Por el contrario, los

Figura 2. Tasa de compresión para el grupo de archivos en inglés



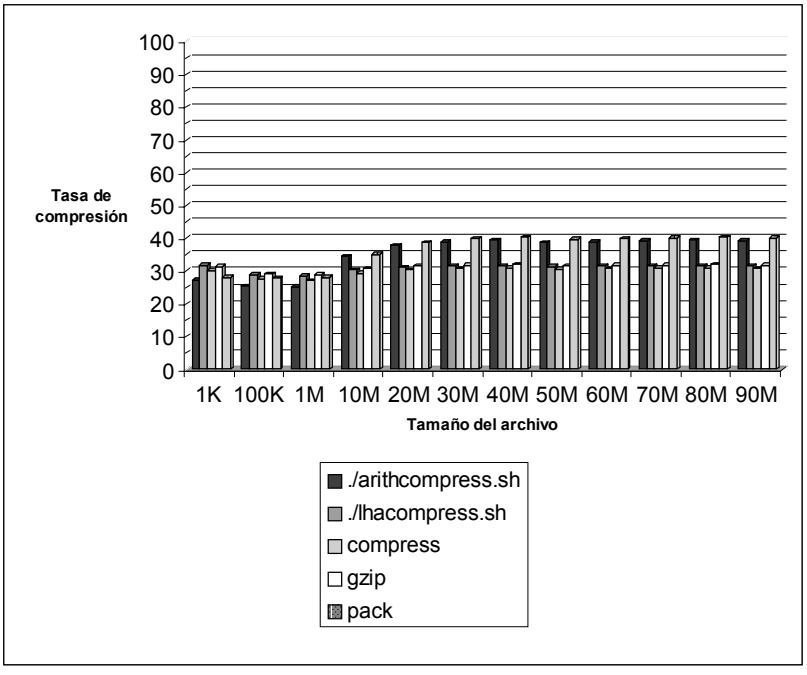

*Fuente*: elaboración propia.

Figura 3. Tasa de compresión para el grupo de archivos no inglés




*Fuente*: elaboración propia.

Figura 4. Tasa de compresión para el grupo de otros archivos



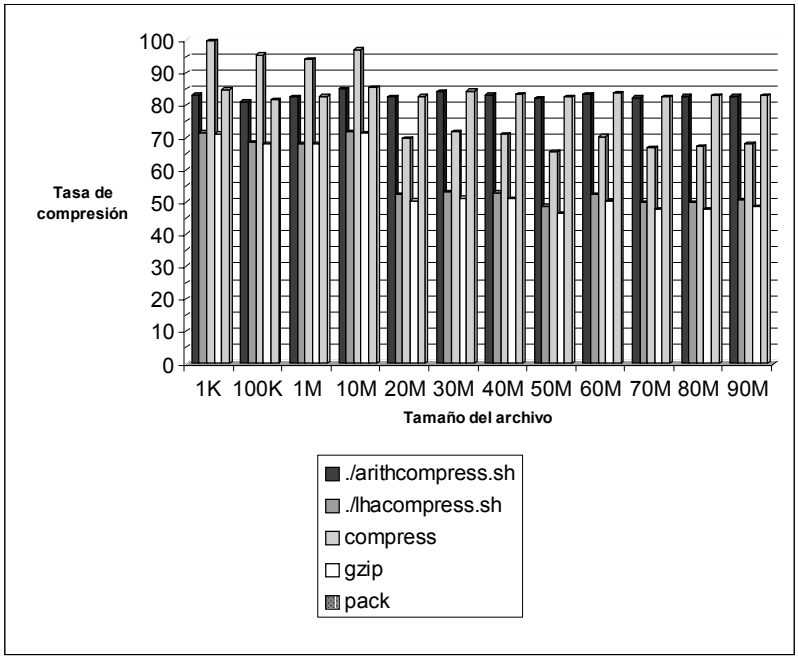

*Fuente*: elaboración propia.

métodos de compresión basados en diccionario parecieron alcanzar la misma tasa de compresión que la lograda en el grupo de archivos no inglés. Éste fue, además, el único experimento donde *compress* (*LZW*) superó en desempeño a las otras herramientas de compresión evaluadas, lo cual puede explicarse por la aparición de subcadenas en los archivos evaluados.

Las figuras 5 y 6 muestran la tasa de compresión en función del tamaño de los grupos creados con LinuxTar y Tcpdump. Como fue descrito, estos grupos de archivos contienen patrones bastante complejos de símbolos y tipos variados de archivos.

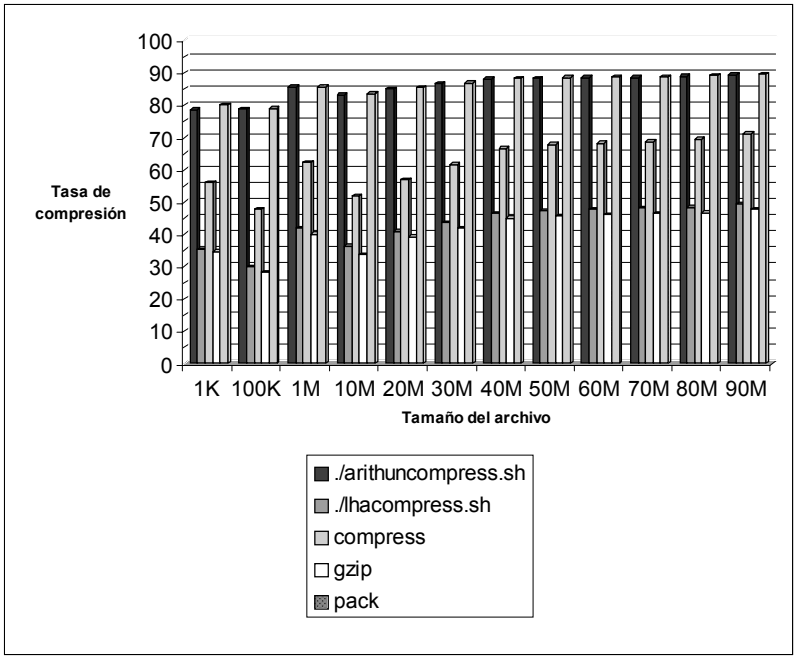
Para el grupo de archivos LinuxTar (Figura 5), la tasa de compresión fue más sesgada. De nuevo, dicha tasa promedio se mantuvo relativamente constante, pero no fue posible calcular una relación entre el tamaño de la entrada y el desempeño de las herramientas de compresión. La Tabla 3 muestra la desviación estándar de la tasa de compresión para el grupo de archivos *LinuxTar*. Ahí se observa que los algoritmos *arithmetic compress* y *pack* poseen una variación muy pequeña en cuanto a su tasa de compresión para archivos mayores a 20 MB, debido a que utilizan las probabilidades de aparición de los símbolos para efectuar la compresión. Es importante anotar que debido a la uniformidad de símbolos en los diferentes grupos de archivos, la desviación estándar para estos dos algoritmos es la de menor variación.

Figura 5. Tasa de compresión para el grupo de archivos LinuxTar




*Fuente*: elaboración propia.

Figura 6. Tasa de compresión para el grupo de archivos Tcpdump




*Fuente*: elaboración propia.

Con el grupo de archivos TcpDump (Figura 6), el desempeño de *LHA*, *compress* y *gzip* se decrementó con archivos grandes. La Tabla 4 muestra el decremento promedio (por ejemplo, incremento de la tasa de compresión) cada 10 MB para el grupo de archivos *Tcpdump*.

Tabla 3. Desviación estándar de la tasa de compresión para el grupo de archivos *LinuxTar* para archivos de tamaño superior a 20 MB

| Herramienta | Desviación estándar |
|---|---|
| ./arithcompress.sh | 0,92 |
| ./lahcompress.sh | 1,64 |
| Compress | 2,21 |
| Gzip | 1,77 |
| Pack | 0,6 |

*Fuente*: elaboración propia.

Tabla 4. Cambio promedio de la tasa de compresión cada 10 MB para archivos de tamaño superior a 20MB

| Herramienta | TcpDump |
|---|---|
| ./arithcompress.sh | 1 |
| ./lahcampress.sh | 1,02 |
| compress | 1,03 |
| gzip | 1,02 |
| pack | 1 |

*Fuente*: elaboración propia.

En los algoritmos basados en diccionario la tasa de compresión depende básicamente de la distribución de subcadenas comunes en el mensaje y en el tamaño del diccionario. Si el mensaje es lo suficientemente largo y contiene patrones repetidos de subcadenas, es posible esperar una buena tasa de compresión en comparación con los otros algoritmos usados. Sin embargo, en el caso del algoritmo LZ77 es importante observar que cuando el mensaje es más grande que el *buffer* del diccionario, la tasa de compresión puede decrementarse por aquellos patrones que una vez existieron en el *buffer*, pero que han debido ser descargados debido al uso de la ventana deslizante (*sliding window*).[4] En el caso de los algoritmos LZW, la tasa de compresión no puede ser mejorada cuando el diccionario se llena. En el caso de *compress*, el diccionario (almacenado en 16 bits) es vaciado cuando la compresión decrece para la siguiente cadena en el mensaje.

En el otro lado, el mayor problema que los métodos estadísticos tratan de resolver consiste en asignar la longitud mínima posible a todos los símbolos en el mensaje para reducir la entropía [Arnold y Bell, 1997]. El algoritmo Huffman ordena las probabilidades de los símbolos y luego mezcla las de los símbolos menos probables [Arnold y Bell, 1997] mientras que *arithmetic code* asigna un número flotante a la probabilidad

4 *Para* gzip *ésta es de 32 kb.*

conjunta de los símbolos en una cadena [Arnold y Bell, 1997, y Livingston *et al.*, 1994]. Por lo tanto, si el número de símbolos y la probabilidad de distribución de dichos símbolos se mantiene constante al incrementar el tamaño del archivo, no se espera una alta tasa de cambio en la tasa de compresión.

### 3.2 Tiempo de compresión

Las figuras 7 a 12 muestran el tiempo de compresión para cada grupo, el tamaño de la muestra y la herramienta.

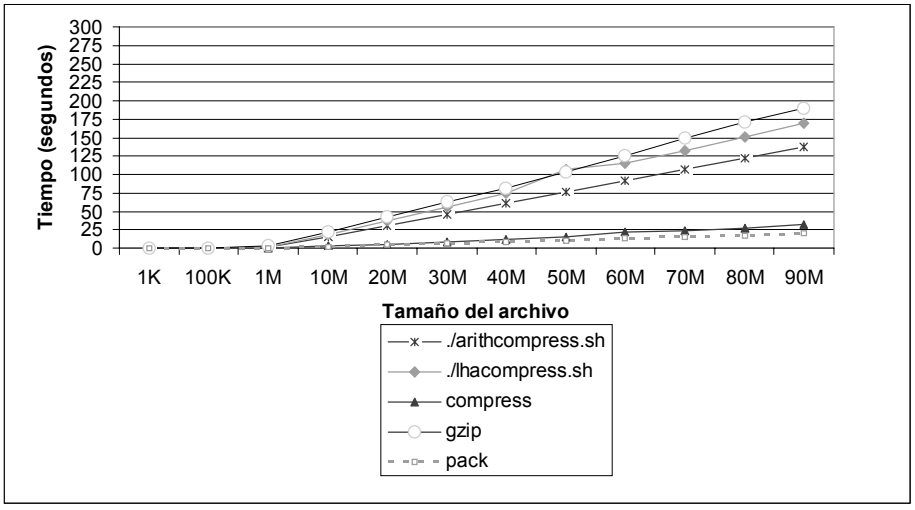
En todos los experimentos es claramente visible una relación lineal entre el tamaño del archivo (mayor a 1 MB) y el tiempo de compresión. Además, se puede concluir que *pack* y *compress* obtuvieron el mejor desempeño. En *pack* (codificación Huffman) la primera estación del proceso de compresión es la generación del modelo. Esto implica que la herramienta necesita leer el archivo dos veces, con lo que se aumenta el tiempo de ejecución del compresor a medida que el archivo incrementa su tamaño. Sin embargo, como puede verse en las figuras 7 a 12, la generación del código (la segunda parte del algoritmo semiestático de Huffman) es en realidad un proceso eficiente: encontrar el símbolo en el árbol y colocar el código en la salida. Ya que la profundidad del árbol no depende del tamaño del archivo (depende de la cantidad de símbolos y la probabilidad de aparición asociada con cada uno de ellos), el peor tiempo de búsqueda (una constante) multiplicado por el número de símbolos arroja un límite superior del tiempo de codificación. LZW (*compress*) incluye dinámicamente en el diccionario los patrones más recientes y busca nuevas cadenas. Sin embargo, se esperaba que LZW se desempeñara mejor que Huffman en el tiempo de ejecución. Éste no es el caso en el experimento, si bien el desempeño es muy similar, esto puede deberse a aspectos relacionados con la implementación de los algoritmos.

El resultado más interesante corresponde al grupo de otros archivos (Figura 10). Allí *gzip* presenta el peor desempeño con un incremento promedio de 75 segundos cada 10 MB. Esto se debe a que los archivos en el grupo contienen cadenas de caracteres compuestas por patrones largos (por ejemplo, del corpus de Canterbury se incluyó un archivo que contenía la letra *a* 100.000 veces). En el caso de LZ77, el tiempo de búsqueda en el *buffer* de historial está limitado por una función exponencial de la cantidad de símbolos que hagan juego en la ventana deslizante (*look-ahead buffer*).[5] Por lo tanto, buscar símbolos que hagan juego con cadenas tan largas afectó negativamente el tiempo de ejecución de LZ77.

En los experimentos, las herramientas de compresión fueron ejecutadas usando los parámetros por omisión de cada una de ellas. Para *arithmetic coding* esto implica que cada símbolo es codificado y

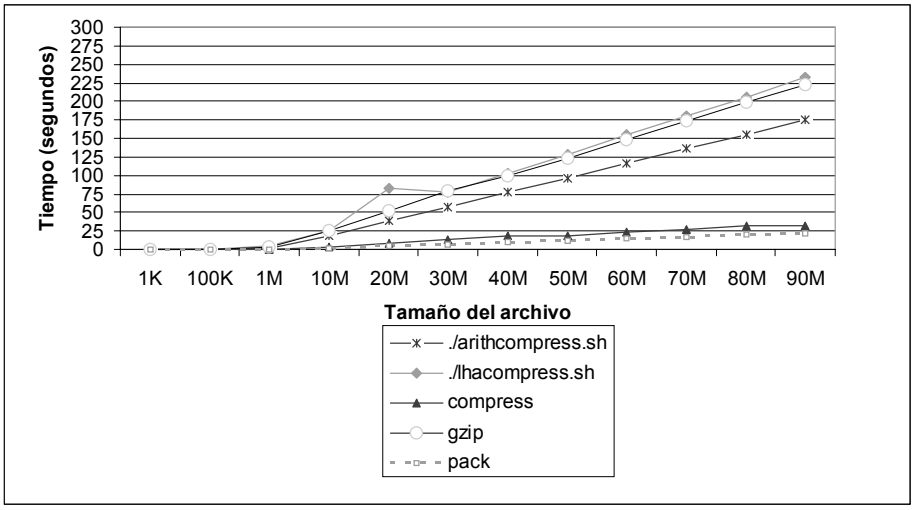
5 *Para* gzip *es de 256* bytes.

Figura 7. Tiempo de compresión para el grupo de archivos binarios



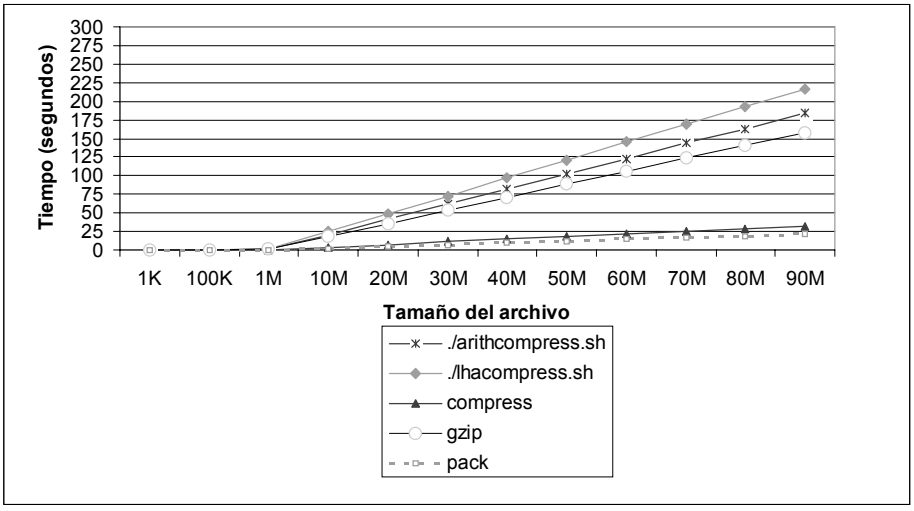

*Fuente*: elaboración propia.

Figura 8. Tiempo de compresión para el grupo de archivos de texto en inglés



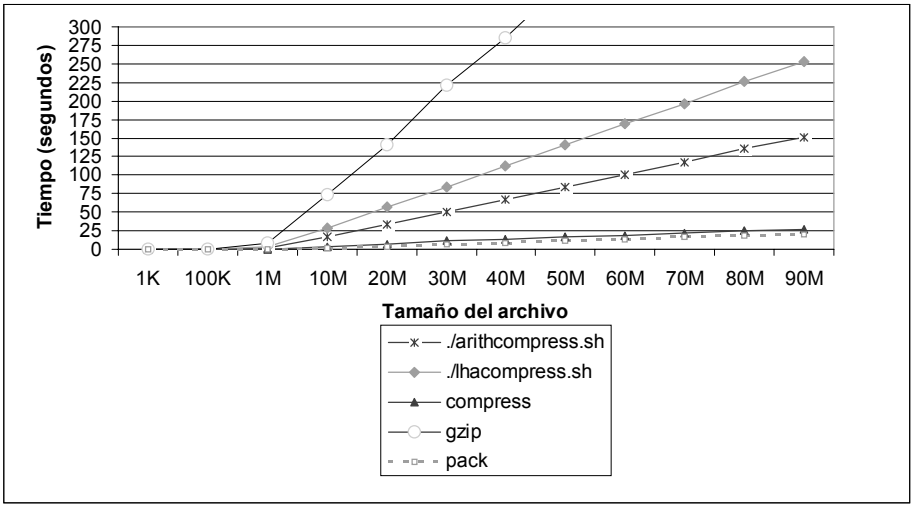

*Fuente*: elaboración propia.

Figura 9. Tiempo de compresión para el grupo de archivos no inglés



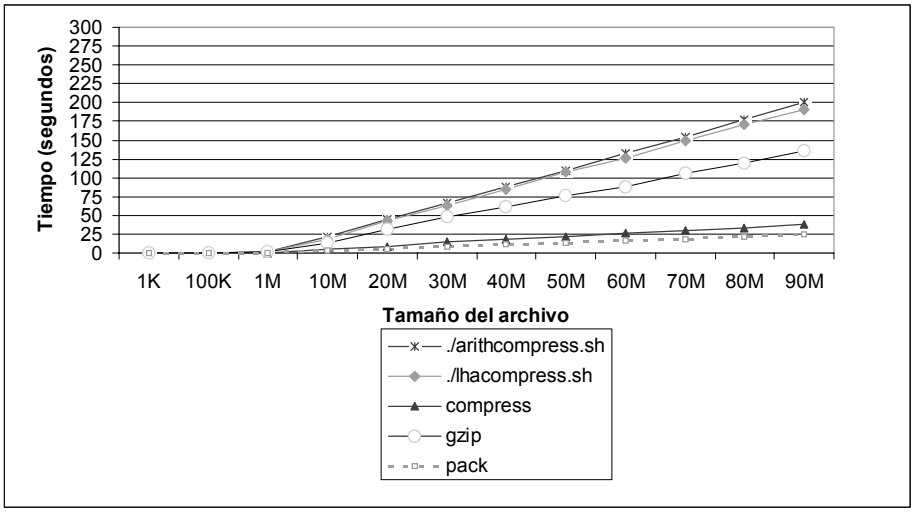

*Fuente*: elaboración propia.

Figura 10. Tiempo de compresión para el grupo de otros archivos



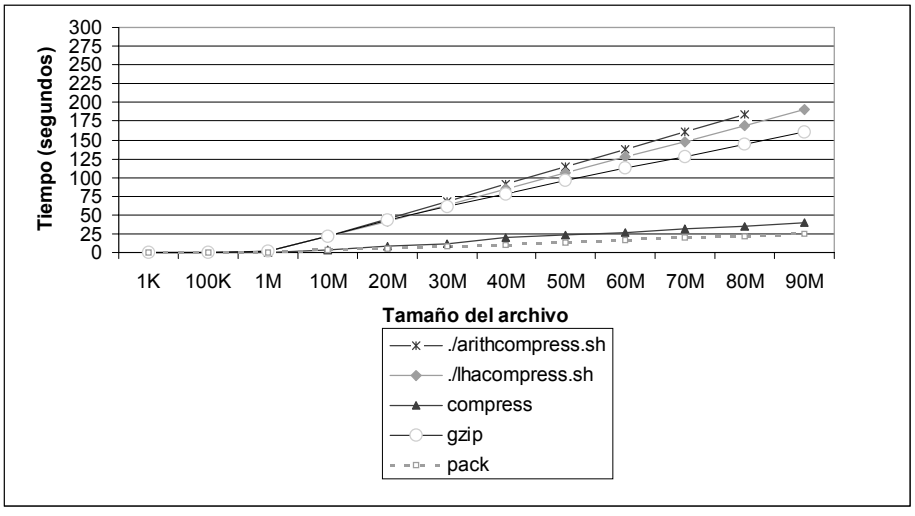

*Fuente*: elaboración propia.

Figura 11. Tiempo de compresión para el grupo de archivos *LinuxTar*




*Fuente*: elaboración propia.

Figura 12. Tiempo de compresión para el grupo de archivos *Tcpdump*




*Fuente*: elaboración propia.

decodificado usando una serie de operaciones matemáticas. Además, para cada bit en el código el decodificador necesita actualizar el intervalo actual —esta versión de *arithmetic encoding* implementa una transmisión y recepción incremental como en Witten, Neal y Cleary [1987]; entonces, al codificar un símbolo, el tamaño del intervalo se duplica para evitar la pérdida de información, debido a la precisión aritmética del procesador—. Es importante observar que este algoritmo también mantiene un árbol dinámico para contar la frecuencia de las ocurrencias de los símbolos en el codificador y el decodificador.

### 3.3 Tiempo de descompresión

Las figuras 13 a 18 muestran el tiempo de descompresión para cada grupo de archivos, tamaño de la muestra y herramienta.

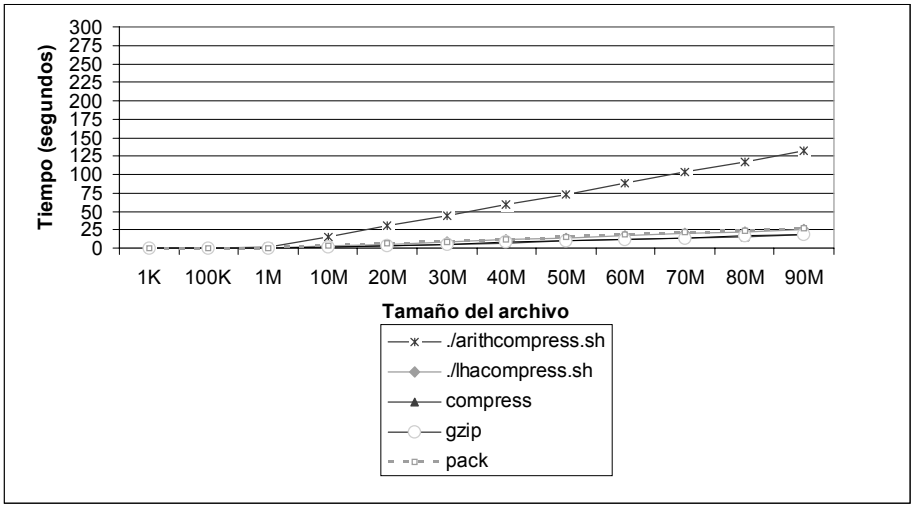
Nuevamente se puede observar que existe una relación lineal entre el tamaño de archivo (mayor de 1 MB) y el tiempo de ejecución de la herramienta de compresión. En todas las figuras se evidencia un comportamiento similar para LHA, *compress*, *gzip* y *pack,* tanto para los tiempos de descompresión como de compresión, con ligeras diferencias en las que los tiempos de descompresión son menores. Esto se debe a que la construcción del diccionario o árbol de subcadenas es más rápido para el proceso de descompresión.

Sin embargo, es claro que el peor desempeño en descompresión está dado por *arithmetic compression.* Como se explicó, tanto el codificador como el decodificador efectúan una serie de operaciones aritméticas por cada símbolo, por lo tanto, se puede esperar que el tiempo de compresión sea similar al tiempo de descompresión.

## 4. Conclusiones

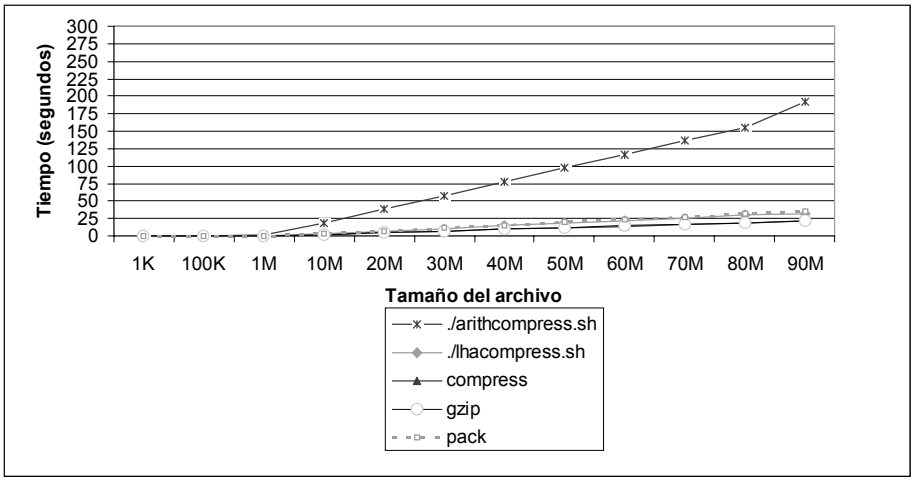
Se ha mostrado que para archivos grandes generados de los corpus de Calgary y de Canterbury, que contengan patrones de símbolos repetidos y una distribución relativamente constante de símbolos, la tasa de compresión de algoritmos sin pérdida se mantiene constante y solamente cambia por un pequeño factor cada 10 MB, debido a que con archivos grandes los métodos basados en algoritmos estadísticos y de diccionario son capaces de crear un modelo adecuado de la fuente. Sin embargo, ese pequeño factor depende del tipo de archivo y de la técnica utilizada. Específicamente, este estudio empírico mostró que usando el grupo de archivos binarios la tasa de compresión de los algoritmos sin pérdida se redujo en 2% cada 10 MB empleando herramientas estadísticas y 3% con herramientas basadas en algoritmos de diccionario. Con el grupo de archivos no inglés la tasa de compresión decreció en 1% usando métodos diccionario y se mantuvo constante para las herramientas estadísticas. Por el contrario, la tasa de compresión para el grupo de archivos en inglés y otros archivos se incrementa para el algoritmo LZW (en promedio el incremento es del 1%). La tasa de compresión para el grupo de archivos en inglés se mantuvo constante para los métodos estadísticos.

Figura 13. Tiempo de descompresión para el grupo de archivos binarios



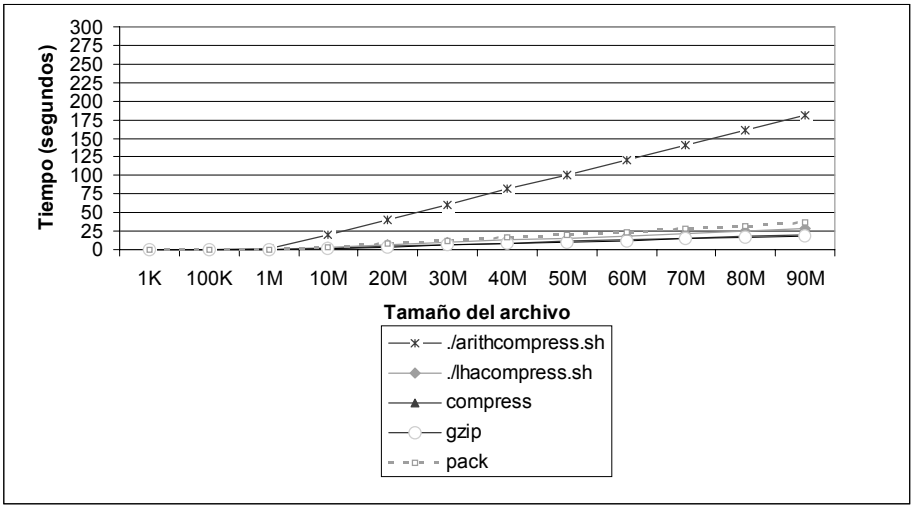

*Fuente*: elaboración propia.

Figura 14. Tiempo de descompresión para el grupo de archivos en inglés



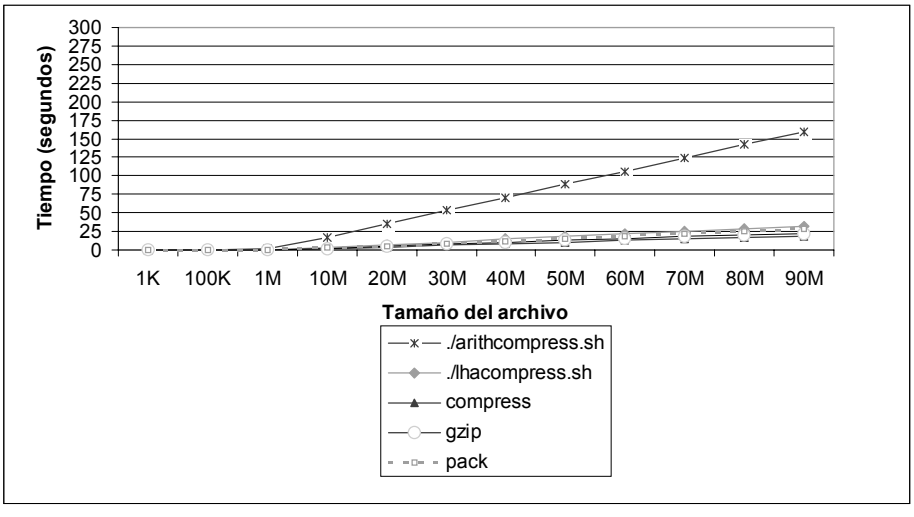

*Fuente*: elaboración propia.

Figura 15. Tiempo de descompresión para el grupo de archivos no inglés



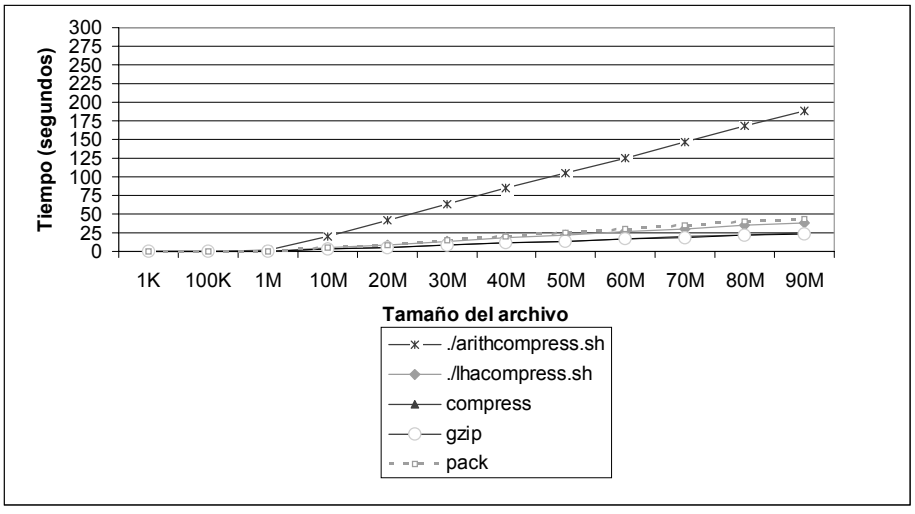

*Fuente*: elaboración propia.

Figura 16. Tiempo de descompresión para el grupo de otros archivos



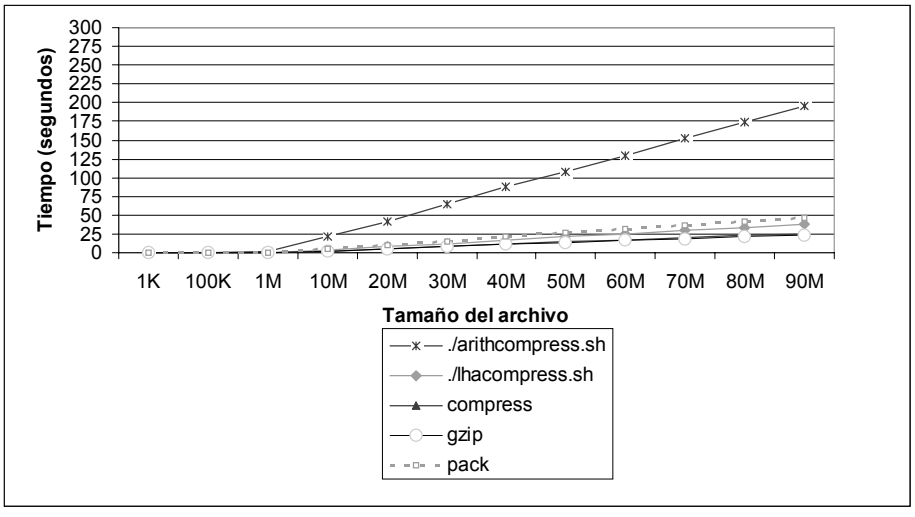

*Fuente*: elaboración propia.

Figura 17. Tiempo de descompresión para el grupo de archivos *LinuxTar*




*Fuente*: elaboración propia.

Figura 18. Tiempo de descompresión para el grupo *Tcpdump*




*Fuente*: elaboración propia.

Conclusiones similares se obtuvieron para los dos grupos de archivos nuevos, es decir, un *tar* del directorio */usr* de un sistema Linux y un archivo de *log* simulando tráfico de red, aunque la tasa promedio de compresión no es tan buena como los demás grupos de archivos de los corpus de Canterbury y de Calgary. Finalmente, se demostró que el tiempo de ejecución para la compresión y la descompresión de información es una función lineal del tamaño del archivo de entrada y de las labores de preparación para la ejecución.

## REFERENCIAS BIBLIOGRÁFICAS

Arnold, R. y Bell, T. A Corpus for the Evaluation of Lossless Compression Algorithms. En: *Proceedings of the IEEE Data Compression Conference.* Utah: Snowbird, 1997.

Bell, T. *et al.* Modeling for Text Compresión. En: *ACM Computing Surveys*, 21(4), diciembre, 1989, 557-591.

Bender, P. E. y Wolf, J. K. An Improved Sliding Window Data Compression Algorithm based on the Lempel-Ziv Data Compression Algorithm. En: *Proceedings on the Global Telecommunications Conference*, GLOBECOM '90, IEEE, 3, 1990, 1773-1777.

Cho, G. Y. y Cho, D. H. A Study on the Efficient Compression Algorithm of the Voice/Data Integrated Multiplexer. En: *Proceedings on the IEEE International Conference on Communications*, ICC 95, Seattle, 3, junio, 1995, 1438-1442.

Jianzhong, L. y Srivastava, J. Efficient Aggregation Algorithms for compressed Data Warehouses. En: *IEEE Transactions on Knowledge and Data Engineering*, 14(3), 2002, 515-529.

Jones, D. A Practical Evaluation of a Data Compression Algorithm. En: *Proceedings of the 1991 Data Compression Conference.* Utah: Snowbird, 1991, 372-381.

Livingston, F. *et al.* Lossless Data Compression in Real Time. En: *Proceedings of the Twenty-Eighth Asilomar Conference on Signals*. Systems and Computers, 2, 1994, 1247-1250.

Mano, Y. y Sato, Y. A Data Compression Scheme which Achieves Good Compression for Practical Use. En: *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*. septiembre, 1991, 442-449.

MIT Lincoln Labs. *Datasets for the 1999 Intrusion Detection Evaluation.* 2004. Disponible en: http://www.ll.mit.edu/IST/ideval/data/1999/training/week1/thursday/outside.tcpdump.gz.

Moffat, A. Arithmetic Coding. 2004. Disponible en: http://www.cs.mu.oz.au/~alistair/arith_coder/ (current May 10).

Williams, R. N. An Extremely Fast Ziv-Lempel Data Compression Algorithm. En: *On Data Compression Conference*, DCC '91, abril, 1991, 362-371.

Witten, I. H., Neal, R. M. y Cleary, J. G. Arithmetic Coding for Data Compression. En: *Communications of the ACM*, 30(6), 1987, 520-540.