Codificador aritmético enfocado a archivos PBM

Tarea 2
Tutor: Álvaro Martin

Alexis Baladón Ferreira de Araújo, 5574612-4

Compresión de Datos sin Pérdida 2021
Núcleo de Teoría de la Información
Facultad de Ingeniería. Universidad de la República
Montevideo, Uruguay

Abstract

En este informe se mostrará el proceso de implementación de un programa capaz de comprimir tanto como descomprimir archivos en base un codificador aritmético con ayuda de la asignación de probabilidad de Krichevsky-Trofimov. Este compresor será capaz de comprimir cualquier secuencia de bits, sin embargo, será modelado específicamente para archivos del tipo PBM. Finalmente se analiza la tasa de compresión del archivo al igual que su evolución a medida que el archivo es comprimido, observándose muy buenos resultados, aun mejores que los obtenidos con compresores convencionales como Zip.

1 Introducción

Los archivos de formato PBM permiten representar imágenes en blanco y negro, donde a bajo nivel se puede observar cómo cada bit del archivo representa el color de cada pixel del mismo (0 es blanco y 1 negro).

Es normal creer que una representación de la realidad como lo es una imagen siga ciertos patrones (por ejemplo, es normal asumir que en un paisaje selvático predominará el color verde), esto facilitará la compresión de un archivo en dichas condiciones si se elige correctamente un modelo que generalice sus características.

En este informe se logra implementar un compresor y descompresor enfocado a este tipo de archivos. Para esto fue necesario recurrir a la compresión aritmética, asumiendo que la distribución de probabilidades era markoviana de orden k, aproximando a la misma en tiempo de compilación con ayuda del estimador de Krichevsky-Trofimov. Se observó que aumentar el tamaño de k no necesariamente significaba mejorar la tasa de compresión, aunque esto tampoco implica que un valor pequeño de k sea el más óptimo. Esto dependerá de la cantidad de patrones hallados con cada modelo al igual que la penalización por memorizar una cantidad de datos redundantes.

Finalmente fueron obtenidos resultados realmente optimistas, con tasas de compresión mejores que las conseguidas con compresores como Zip, alcanzando en el mejor de los casos una tasa de 0.1446.

2 Problema a Resolver

El objetivo de este informe es programar un compresor (y descompresor) capaz de manipular archivos de tipo PBM con la ayuda de un codificador aritmético, el cual se apoyará de las probabilidades condicionales de ver un símbolo dado los anteriormente observados, las cuales serán aproximadas a medida que la secuencia es comprimida. Para obtener dichas probabilidades se asumirá que estas responden a un modelo markoviano de orden k, parámetro el cuál variará entre 0 y 64 a elección del usuario.

Este enfoque es equivalente a expresar a cada pixel en función a los pixeles de su izquierda (o de arriba si se alcanza el fin del ancho de la imagen), lo cual no es igual de eficiente que estudiar los pixeles espacialmente a su alrededor, sin embargo, es mucho más sencillo a nivel de implementación.

3 Metodología

3.1 Idea general

Para crear un programa capaz de cumplir con este objetivo se recurrirá a un codificador aritmético disponible públicamente [1]. De aquí será necesario implementar una estructura capaz de almacenar la frecuencia con la que se repite cada símbolo en determinado estado, para luego estimar las probabilidades condicionales de observar a cada símbolo individual dados los que ya lo han sido, con ayuda del estimador de Krichevsky-Trofimov.

Todo esto será logrado con la ayuda del lenguaje de programación C++ y la programación orientada a objetos.

3.2 Estructura implementada

Como el programa debe recordar un máximo de 64 bits a lo largo de su ejecución, se decidió usar como buffer una clase que contiene únicamente un atributo del tipo unsigned long long int (de 64 bits), el cual podrá ser actualizado con la ayuda de operadores booleanos.

Sobre este mismo buffer será utilizada la operación ”shift” al leerse un nuevo bit, para el cual se mantendrán en 0 los bits que no interesan recordar. Esto permitirá que cada buffer funcione a su vez como identificador de cada estado (el buffer tomará solo un valor para el mismo estado), por lo que puede ser aprovechado para el uso de diccionarios.

Las estrucuras utilizadas para el diccionario fueron arreglos y tablas de hash, modelándose como distintas estrategias de la clase Diccionario, que se instanciara como una o la otra según el valor de k. El uso de arreglos en la práctica demostró ser siempre más rápido que las tablas de hash, sin embargo, si queremos almacenar cada estado posible deberíamos crear un arreglo de tamaño 2^k, por lo que a partir del valor $k = 16$ (inclusive) se utilizará el diccionario ”unordered_map” previsto por las librerías STL, el cual está implementado con tablas de hash (que ayudarán a encontrar las estadísticas de cada estado en $O(1)$ promedio).

4 Análisis Experimental

4.1 Descripción del análisis

Los experimentos realizados sobre este apartado están centrados en la tasa de compresión del programa variando el valor del parámetro k anteriormente descrito en su dominio predefinido ($0 \leqslant k \leqslant 64$).

Con el fin de analizar los resultados del programa serán estudiados 4 archivos, de los cuales importará cómo varía la tasa de compresión en fución del parámetro k y la posición del archivo comprimido. Para este último punto a estudiar se evaluará la tasa de compresión en cada intervalo de tamaño $n/5$ del archivo.

A tener en cuenta: Para esta sección se utiliza el término 'tasa de compresión' para referirse al cociente del tamaño del archivo comprimido y del archivo original.

4.1.1 Imagen A (128KB)

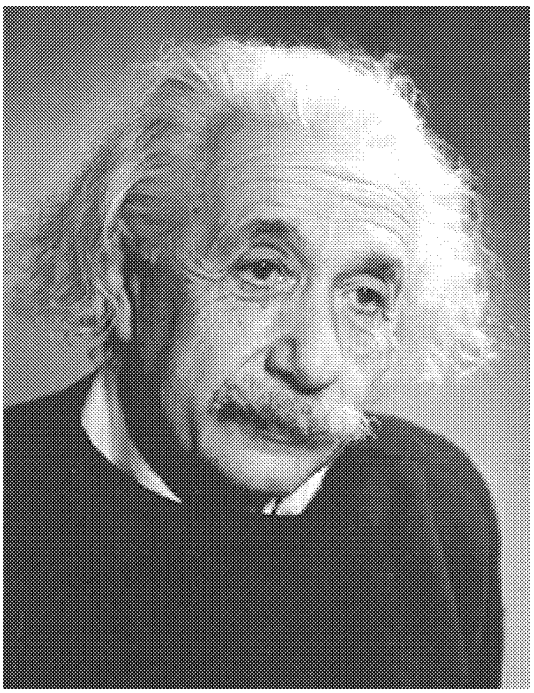

Fig. 1. "Imagen A"

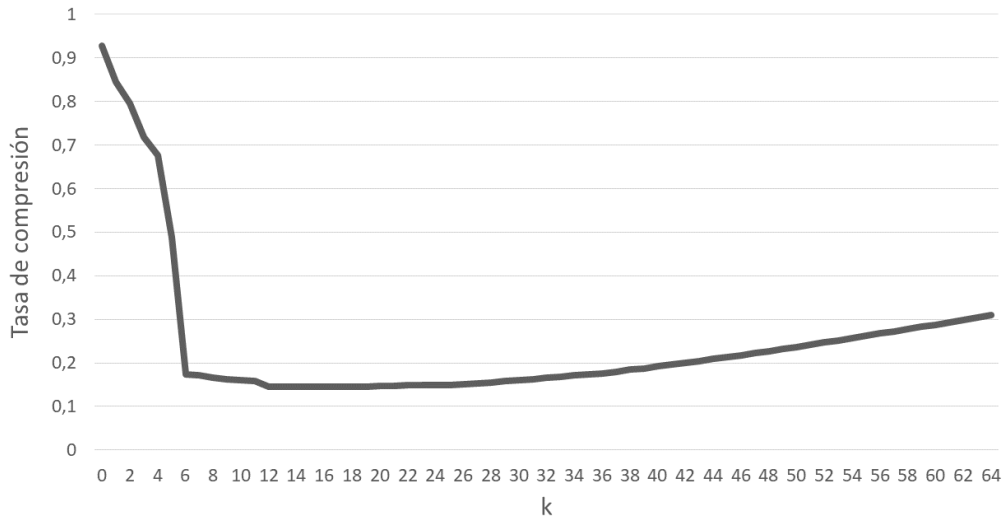

Fig. 2. Tasa de compresión en función del parámetro 'k' (Imagen A)

En esta gráfica se observa como al variar el parámetro k se comienza con una tasa de compresión considerablemente mayor a la alcanzada para mayores valores de k. A partir de aquí se llega a un mínimo absoluto, aunque localmente no parecen haber grandes saltos en la pendiente de la gráfica (como sí los hay para órdenes menores que 6).

El valor del parámetro k que minimiza la tasa de compresión es $k = 18$, logrando una tasa de 0.1447, valor extremadamente pequeño si se compara con la compresión de un archivo de cualquier otro formato del mismo tamaño.
Esto puede deberse a la efectividad del programa tanto como el formato de la imagen. Estudiando la compresión del mismo archivo mediante la realizada por Zip se observa una tasa de 0.2038, esto es un resultado importante si se tiene en cuenta el trabajo que hay detrás de un compresor como lo es Zip.

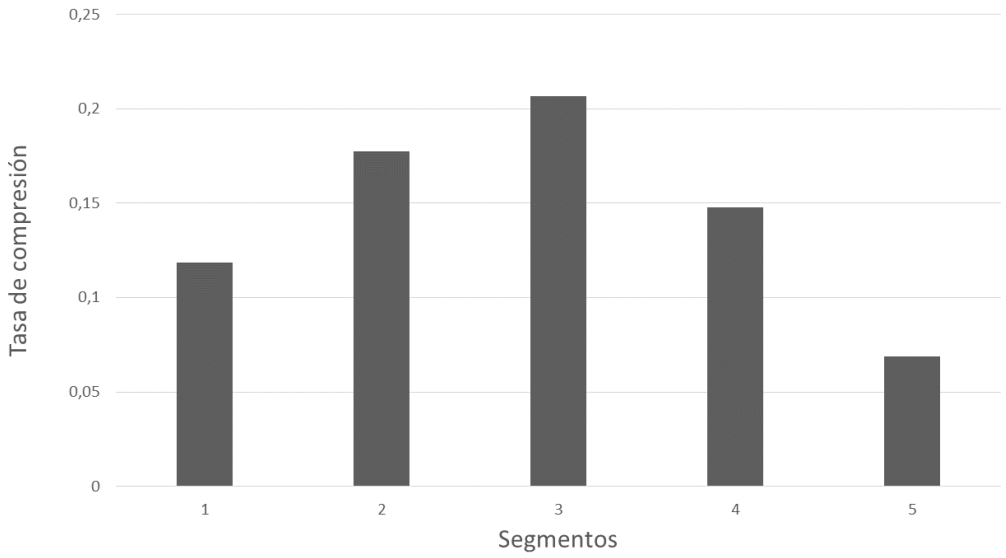

Fig. 3. Tasa de compresión en función del segmento del archivo para k = 18 (Imagen A)

Observarción: Los datos exhibidos son **de la tasa de compresión de cada segmento por separado** y **no** desde el inicio de la compresión hasta el momento indicado.

La tasa de compresión en la anterior gráfica crece hasta alcanzar un máximo para todos los intervalos de tamaño $n/5$ en el archivo, donde comienza a decrecer nuevamente.

Analizando la ”Imagen A” a simple vista, el comportamiento de la tasa de compresión parece justificarse por la distribución de pixeles en la misma. Particularmente en el centro de la imagen (estudiándola de arriba hacia abajo) se denota una menor uniformidad en la aparición de colores, a diferencia de la parte inferior por ejemplo, donde predominan pixeles oscuros.

4.1.2 Imagen B (63KB)

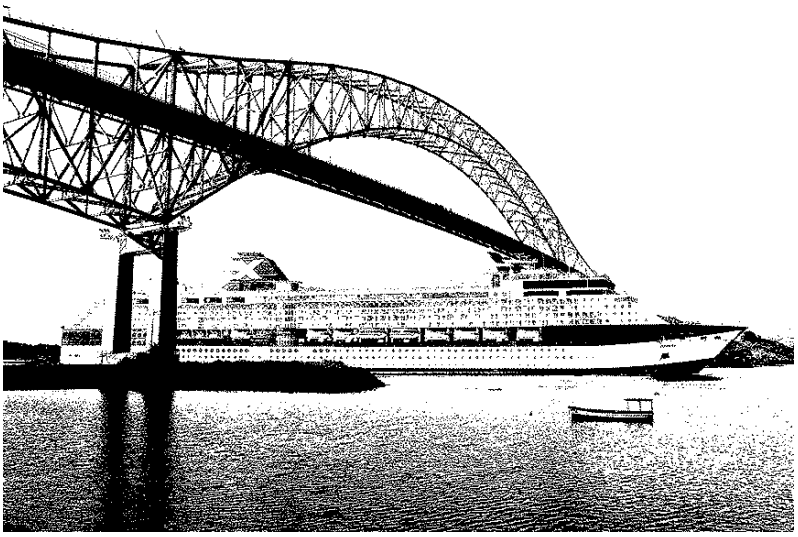

Fig. 4. ”Imagen B”

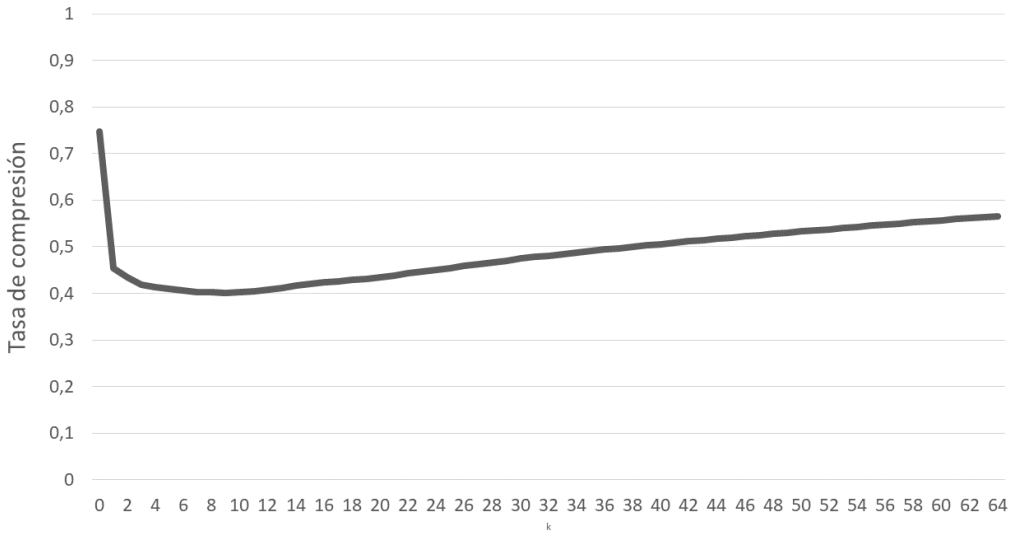

Fig. 5. Tasa de compresión en función del parámetro ’k’ (Imagen B)

De aquí se observa un comportamiento similar al archivo anterior. Se da un valor que minimiza la tasa de compresión aunque sin una diferencia notable con la tasa

que brindan valores de k cercanos. Este mínimo se da en $k = 9$, con una tasa de compresión de 0.4016, la cual no es tan optimista como la obtenida en el caso anterior, aunque lo sigue siendo comparándose con la compresión Zip: 0.4349.

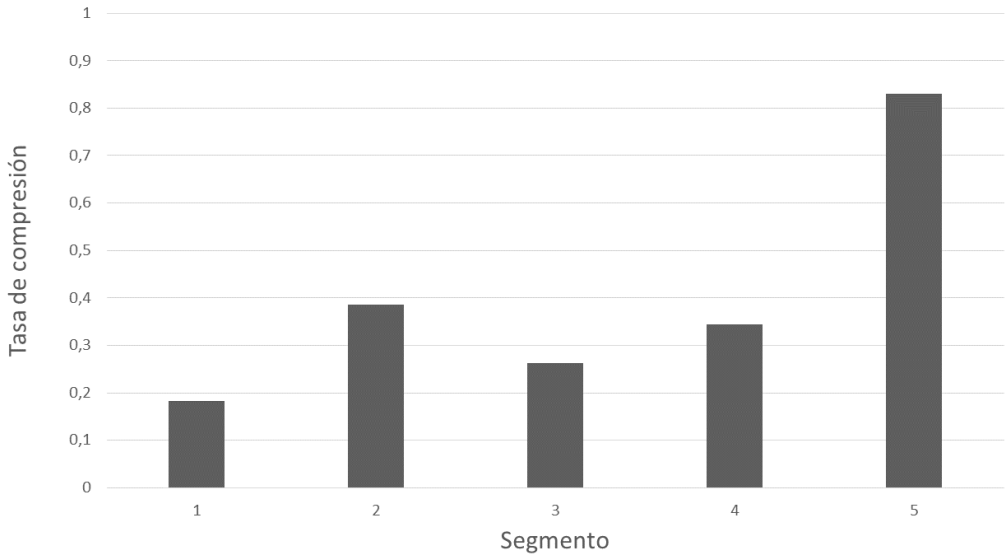

Fig. 6. Tasa de compresión en función del segmento del archivo para k = 9 (Imagen B)

A diferencia del caso anterior se observa una tasa creciente, lo cual de nuevo parece ser un comportamiento esperado, ya que los primeros bits que se codifican son casi enteramente blancos en la sección superior de la imagen, mientras en la inferior no parece seguirse ningún patrón en particular.

4.1.3 Imagen C (7KB)

Fig. 7. ”Imagen C”

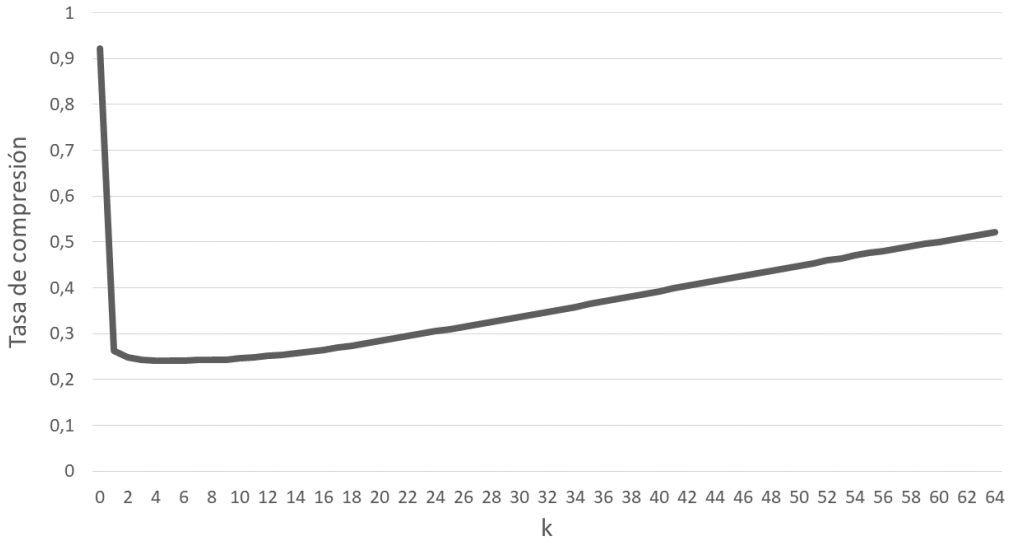

Fig. 8. Tasa de compresión en función del parámetro ’k’ (Imagen C)

Si se compara esta gráfica con las obtenidas para las dos imágenes previas se observan resultados casi idénticos, no parece haber indicios de que este

comportamiento vaya a cambiar. El mínimo en este caso se da en $k = 5$ con una tasa de 0.2410, nuevamente superando a la obtenida con Zip: 0.3055.

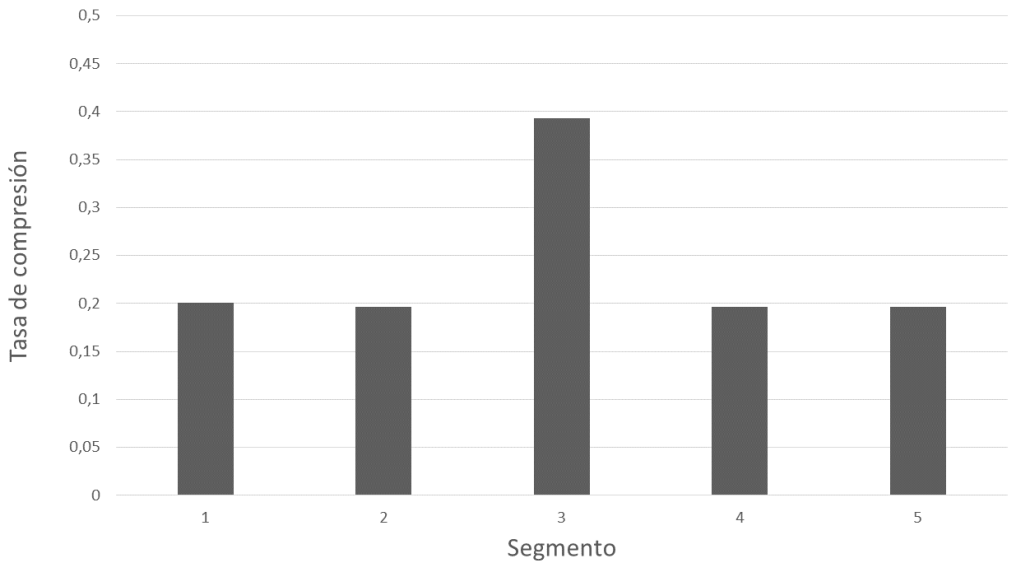

Fig. 9. Tasa de compresión en función del segmento del archivo para k = 5 (Imagen C)

Estos resultados no parecen tan fáciles de interpretar, si se analizan los primeros datos de la gráfica ellos no parecen ir en contra de la intuición, sin embargo, uno esperaría que el segmento final presente una tasa relativamente inferior a la inicial, donde se dan patrones menos complejos (muchos pixeles negros seguidos de pixeles negros y blancos seguidos de blancos). El factor que parece estar afectando la codificación puede estar relacionado con la ”memoria” del programa, ya que este no distingue entre el momento en el que los datos fueron obtenidos. Si al inicio de la imagen se dio un patrón específico y este cambia significativamente, la única forma de contrarrestar esta ”desinformación” es obtener suficientes datos empíricos para hacer insignificantes a los inicialmente obtenidos.

4.1.4 Imagen D (60KB)

Fig. 10. ”Imagen D”

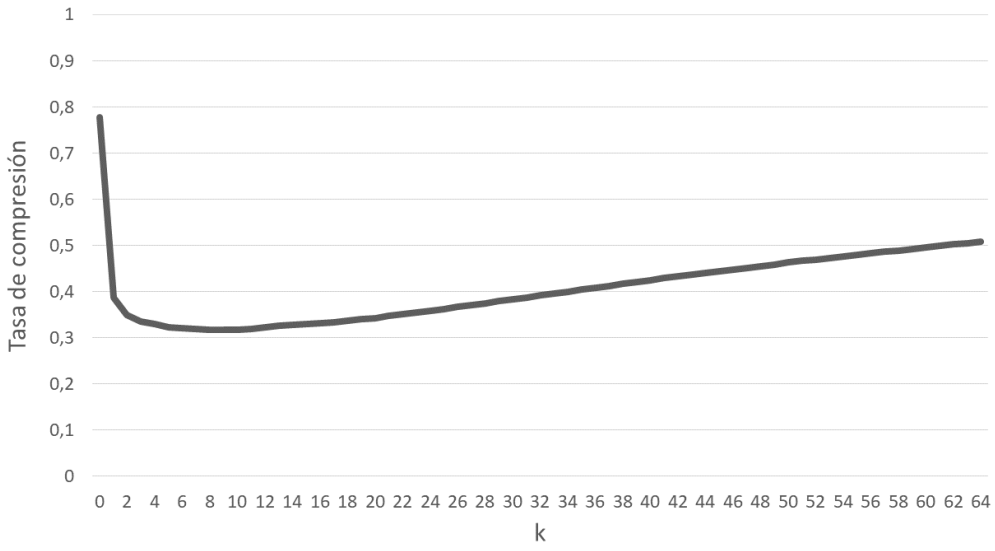

Fig. 11. Tasa de compresión en función del parámetro ’k’ (Imagen D)

En este caso se observa un mínimo en $k = 8$, con una tasa nuevamente superior a la obtenida con Zip: 0.3171 contra 0.3595. Si se observan los valores obtenidos previamente para k, se perciben mejores resultados para valores pequeños, lo cual

puede parecer inicialmente contraintuitivo, ya que se está perdiendo efectividad por ”recordar” más datos. Además, curiosamente, se observa una alta correlación entre el tamaño del archivo y el valor de k que optimiza la compresión (particularmente, si se calcula el coeficiente de correlación entre los tamaños en función de los valores de k, se obtiene un resulado para nada despreciable de 0.97). Esto puede deberse a la cantidad de pixeles utilizados para representar la realidad manifestada en la imagen (en general rostros humanos), ya que los mismos patrones encontrados por el compresor (por ejemplo, una nariz) en imágenes grandes, también son encontrados por el mismo en imagenes pequeñas, pero utilizándose menos bits en cada uno. Aun así debe tenerse en cuenta que el tamaño de la muestra es pequeño y que podrían darse casos que contradigan este hallazgo.

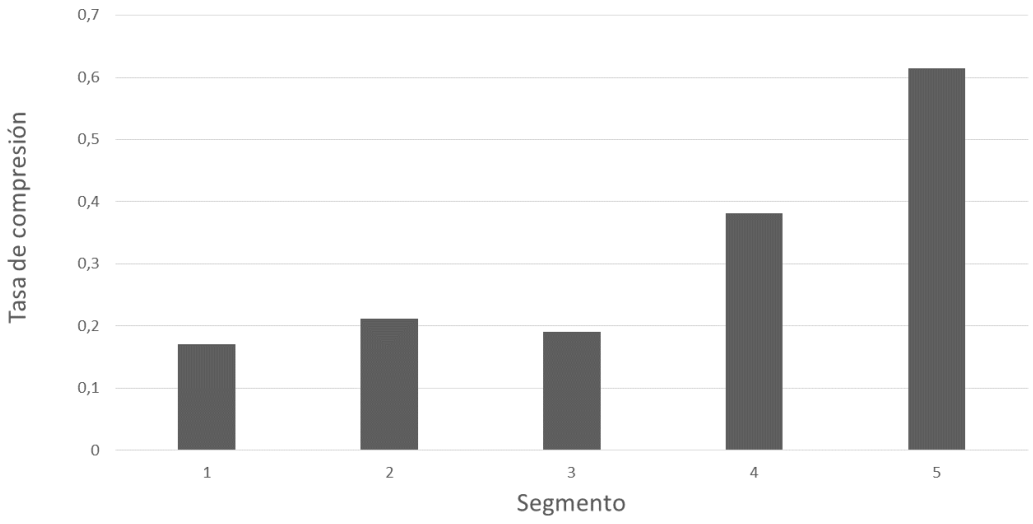

Fig. 12. Tasa de compresión en función del segmento del archivo para k = 8 (Imagen D)

Nuevamente se observan resultados intuitivos. Fue resaltado que el hecho de que la memoria del compresor no tenga sentido del tiempo es un dato relevante, aunque no parece realmente influir tanto en la efectividad del compresor como la repetición de patrones sin relación aparente (como lo es en este caso el sombreado de la ropa del hombre de la imagen D, comparado con el fondo blanco que hay detrás).

En caso de quererse tener en cuenta imágenes que presenten distintos contextos

dentro, podría ser considerado tanto olvidar memoria luego de haber visto determinado número de pixeles o favorecer la ponderación de valores vistos más recientemente (aunque para esto podrá será necesario almacenar cada dato observado de forma ordenada en vez de en un simple contador).

4.1.5 Costo por recordar más datos

Las más obvias desventajas de aumentar el parámetro k, son la cantidad de memoria utilizada para almacenar las apariciones de cada pixel en cada estado de dicho orden, y el tiempo de ejecución ocupado por el compresor. Ya se observó cómo el uso de memoria imposibilitaba la utilización de arreglos para valores altos de k (estructuras realmente rápidas a la hora de obtener el valor asociado a un identificador).

Un precio no tan evidente que debe ser pagado por el compresor está relacionado con la tasa de compresión en función del valor de k. ¿Por qué es que mejorar la cantidad de datos memorizados afectaría negativamente en los resultados obtenidos? Para responder esta pregunta será de ayuda el siguiente teorema:

Theorem

Sea $\mathcal{C}_F = \{P_\theta\}_{\theta\in\Theta}$ la clase de modelos definida sobre un alfabeto finito $\mathcal{X}$ por una FSM F con conjunto de estados S, donde las distribuciones de probabilidad condicionales en los estados se definen mediante el parámetro $\theta \in \Theta$, de dimensión $|S|(|\mathcal{X}| - 1)$. Se cumple

$$MMR_n(\mathcal{C}_F) = \frac{|S|(|\mathcal{X}| - 1)}{2} \log n + \Theta(1)\,.$$

Fig. 13. ”Cota para MMR”

El programa implementado lee cada símbolo en forma de bits, por lo que el valor de $|X|$ del ejemplo sería igual a 2, mientras que $|S|$ sería igual a la cantidad de estados del modelo utilizado (2^k). Esto convertiría al Min-Max Regret del teorema

en 2^{k-1}log n, lo cual significa que el aumento de k juega un papel importante en la efectividad del compresor. Aunque si este teorema es cierto, ¿Por qué el mínimo no se da en $k = 0$, y qué lógica tiene esta fórmula más allá de lo matemático?

Para intentar explicar la primera pregunta se deben tener en cuenta la cantidad de factores que influyen en el cálculo del Regret de nuestra distribución. Es cierto que la diferencia entre la asignación brindada por el modelo real y la asignada por Krichevsky-Trofimov crece junto con k, sin embargo, cada modelo que nos posibilita imitar dicho parámetro puede representar de forma más precisa la realidad existente en la secuencia a comprimir.

Un ejemplo que permitirá comprender ambas preguntas será la codificación de la secuencia de bits 01100001 repetida indefinidamente dentro de un archivo (fue elegido el tamaño 8 en particular ya que es más común observar textos con caracteres ASCII, y suele ser más conveniente estudiar estas tiras como octetos y no bit a bit). Aquí si asumimos que el modelo es iid, el compresor será incapaz de observar patrones en la secuencia, aun siendo esta casi absolutamente redundante. Por otro lado, ¿Qué obtiene el compresor de recordar mas de 8 bits? Es aquí donde el teorema tratado anteriormente juega un rol importante, ya que no se obtienen ganancias por memorizar más datos, pero sí empeora la aproximación brindada por el estimador. Que un k no tan pequeño ni grande sea el óptimo es un reflejo de ambos factores, es el punto medio entre intentar recordar información innecesaria y no recordar la suficiente como para encontrar patrones.

5 Conclusiones

Durante la fase de análisis se observaron distintos resultados al variar el parámetro k dentro de su dominio, además de analizar estos resultados en cada segmento de forma individual.

Estudiando la efectividad del compresor se observaron resultados ampliamente optimistas, ya que estos superaban en el mejor de los casos a la compresión obtenida por la compresión Zip.

Se observó además que memorizar más datos no era equivalente a mejorar la compresión, ya que esto tiene un costo en la aproximación de la distribución de probabilidad del modelo elegido, además de claro, el tiempo de ejecución y almacenamiento de memoria que requiere el programa. Adicionalmente, parece haber una relación entre el tamaño del archivo y el del parámetro óptimo para el compresor.

Desde mi punto de vista, haber utilizado un compresor de uso libre facilitó enormemente la realización del programa. Además, visto en los resultados de la tasa de compresión, parecía estar realmente bien optimizado independientemente de la pequeña cantidad de líneas de código presentes en el mismo. Encuentro que el uso de las estructuras implementadas (arreglos y tablas de hash) fue realmente útil, especialmente al utilizar arreglos, aunque si hubiese creado una función de hash adecuada para el identificador utilizado los tiempos de ejecución hubiesen mejorado considerablemente. Aún asi, esto solo se nota al querer comprimir varios archivos, ya que la compresión de secuencias individuales es casi instantánea.

References

[1] Arithmetic Coding Statistical Modeling Data Compression https://marknelson.us/posts/1991/02/01/arithmetic-coding-statistical-modeling-data-compression.html