Bibliografía Anotada

Alexander Enrique Urieles Nieto
aeurielesn@unal.edu.co

Maestría en Sistemas y Computación
Departamento de Ingeniería de Sistemas e Industrial
Universidad Nacional de Colombia

Bibliografía anotada

- **A. Amir and C. Benson. Efficient two-dimensional compressed matching. *Data Compression Conference, 1992*, 2002**

  Propone el problema de búsquedas de patrones sobre texto comprimido sin descompresión explicita, anteriormente no definido. Dejando claro la necesidad de investigación en el tema, especialmente en algoritmos de búsquedas en texto comprimido para algoritmos de compresión en varias dimensiones. No se concentra en el problema de *Run-Length Encoding* (RLE) debido a la aparente trivialidad de las búsquedas en el texto comprimido de éste. En cambio, se concentra en el problema de *two-dimensional run-length compressed matching* empleando técnicas de periodicidad en dos dimensiones definidas en [2].

  Presenta un algoritmo subóptimo para el problema de *two-dimensional run-length compressed matching*. En términos generales, el algoritmo propuesto se compone de dos fases: una fase de pre-procesamiento y una de búsqueda. En la primera fase se hayan las posiciones de discordancia entre del patrón y el texto comprimido. En la segunda fase se consideran posibles candidatos para *matching* usando una porción del patrón, luego se crean bloques disyuntos de tamaño $\frac{m}{2} \times \frac{m}{2}$ que contengan a los posibles candidatos, y por último se verifican los posibles bloques candidatos contra el texto para asegurar una ocurrencia.

  No hay resultados experimentales del algoritmo presentado en el artículo.

- **A. Amir, G.M. Landau, and D. Sokol. Inplace run-length 2d compressed search. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 817–818. Society for Industrial and Applied Mathematics, 2000**

  Define un algoritmo *inplace*, una nueva medida para el problema de *compressed matching*, y propone el primer algoritmo inplace para el problema

de búsquedas en texto comprimido sin descompresión explicita para el algoritmo de compresión Run-Length en 2-dimensiones.

Un algoritmo es *inplace* cuando la cantidad de espacio extra requerido es proporcional al tamaño del patrón dado. El algoritmo inplace presentado se basa en una estrategia de *duelo* y presenta una forma de acceder los *witness* en tiempo constante razón por la cual anteriormente no se había podido aplicar esta estrategia a Run-Length Encoding (RLE).

Detalla altamente los pasos desarrollados por el algoritmo y el proceso en el artículo. En la matriz de entrada se descartan los casos triviales donde las todas filas contienen el mismo carácter por la alta incidencia de patrones que pueden empezar en posiciones arbitrarias. No hay resultados experimentales del algoritmo presentado.

- **Amihood Amir, Gary Benson, and Martin Farach. Let Sleeping Files Lie: Pattern Matching in Z-Compressed Files. *Journal of Computer and System Sciences*, 52(2):299–307, 1996**

  Introduce las primeros aproximaciones en la solución al problema de búsquedas de patrones sobre texto comprimido para algoritmos adaptativos como la familia de algoritmos LZ [5, 6, 7, 8]. También refuerza la necesidad de investigación en el tópico terminando con una lista de problemas abiertos requeridos en el mismo.

  Se presentan dos algoritmos subóptimos para realizar búsquedas de patrones sobre texto comprimido con LZW [7], mostrando el proceso para lograr compensación entre tiempo y espacio para el segundo algoritmo teniendo como base el primero. Introduce un nuevo criterio, el criterio del espacio adicional para algoritmos de compresión de información en cual un algoritmo óptimo no debe usar más de $O(n)$ espacio adicional e incluso de ser posible no usar más de $O(m)$ espacio adicional, siendo n y m los tamaños del texto comprimido y el patrón respectivamente.

  Se recalca que se presenta el primer algoritmo subóptimo conocido para el tópico de búsquedas de patrones sobre texto comprimido sin descompresión explicita. Afirma que la solución del mismo problema para algoritmos de compresión no adaptativos es trivial. No hay resultados experimentales de los algoritmos presentados.

- **T. Bell, M. Powell, A. Mukherjee, and D. Adjeroh. Searching BWT compressed text with the Boyer-Moore algorithm and binary search. In *Data Compression Conference, 2002. Proceedings. DCC 2002*, pages 112–121. IEEE, 2002**

  Presentan dos algoritmo para realizar *online pattern matching* en archivos comprimidos con la transformada de Burrows-Wheeler [10] sin necesidad de índices pre-calculados. El primer algoritmo basado en el algoritmo de Boyer-Moore (BM) [11] utiliza un arreglo adicional para acceder a caracteres arbitrarios directamente en el texto comprimido. La forma de construcción de este arreglo se presenta en el artículo. El segundo algoritmo

usa búsqueda binaria y se basa en el hecho que el algoritmo de Burrows-Wheeler genera una lista ordenada de todas las palabras presentes en el texto lo cual hace posible su aplicación directa.

Los resultados experimentales muestran que el primer algoritmo es mucho más rápido para búsquedas de un único patrón que la descompresión y subsiguiente búsqueda usando el algoritmo de BM sobre el texto descomprimido e incluso que el segundo algoritmo propuesto usando búsqueda binaria. Sin embargo, el algoritmo de búsqueda binaria es más rápido (casi indiferente con respecto a los resultados con un solo patrón) que ambos cuando se realizan búsquedas de múltiples patrones a la vez con la pequeña desventaja de los requerimientos de memoria que éste requiere.

- **NR Brisaboa and A Fariña. Lightweight natural language text compression. *Information Retrieval*, 2007**

  Presenta dos nuevos esquemas estadísticos semi-estáticos de compresión para lenguajes naturales basados en *Tagged Huffman Codes* [13]. Estos esquemas permiten una codificación sencilla y rápida, tienen mejores índices de compresión que su predecesor, permiten búsquedas en texto comprimido sin necesidad de descompresión explicita y permite descompresión local de forma aleatoria en el texto comprimido.

  Los dos esquemas de compresión: End-Tagged Dense Codes y (s, c)-Dense Codes. Los *End-Tagged Dense Codes* se diferencian de los Tagged Huffman Codes por la ubicación del bit de señalización de inicio de una palabra clave, usa el bit menos significativo en vez de usar el bit más significativo. Este cambio beneficia el rango de posibilidades para la generación de palabras claves.

  Los (s, c)-*Dense Codes*, s por *stoppers* y c por *continuers*; son una generalización de los End-Tagged Dense Codes en la cual se busca optimizar la cantidad de stoppers y continuers usados. Entre mayor es el valor de s se gana compresión en las palabras más frecuentes y se pierde compresión en las menos frecuentes. Razón por la cual se busca hallar valores óptimos de s y c. En la práctica es suficiente con aplicar una búsqueda binaria para localizar el valor mínimo para computar los mejores s y c.

  Con unas pequeñas modificaciones se puede usar cualquier algoritmo de búsqueda. En este caso, los autores usan el Boyer-Moore (BM) [11] que puede ser aplicado casi sin modificación alguna. Una verificación es realizada después de cada ocurrencia para asegurar que el algoritmo encontró efectivamente el inicio de palabra.

  Los *Dense Codes* se generan aproximadamente entre un 45 % y 60 % más rápido que los códigos Huffman.

  Los resultados experimentales muestran que End-Tagged Dense Codes son entre un 2 % y 5 % más rápidos que Tagged Huffman Codes. Las búsquedan en (s, c)-*Dense Codes* son entre un 2,5 % y 7 % más rápidas que en End-Tagged Dense Codes, y entre un 5 % y 10 % más rápidas que en Tagged

Huffman Codes. (s, c)-*Dense Codes* presenta mejores tiempos de búsqueda para textos de tamaño medio y Tagged Huffman Codes para textos largos.

- **M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm, 1994**

  Presenta un esquema de compresión que aplica una *transformada reversible* a un bloque de texto para explotar las redundancias en el texto y hacerlas más accesibles para otras técnicas de compresión.

  El algoritmo transforma una cadena generando sus rotaciones cíclicas, ordenándolas lexicográficamente y extrayendo el último carácter de cada rotación. El autor demuestra que la cadena original se puede obtener a partir del último carácter de cada rotación y la posición de la cadena original dentro del conjunto de rotaciones.

  También explica porqué este esquema genera buenos resultados para compresión.

- **L. Chen, S. Lu, and J. Ram. Compressed pattern matching in DNA sequences. In *CSB '04 Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference*. IEEE Computer Society, 2004**

  Presenta un algoritmo basado en Boyer-Moore (BM) [11] para búsqueda en texto comprimido de secuencias de ADN.

  El algoritmo de compresión utiliza dos bits para la codificación de cada carácter A, G, T, C; logrando un 75 % de compresión y además ayuda en el rendimiento del algoritmo al incrementar el tamaño del alfabeto a 256 valores.

  Los resultados experimentales muestran que el algoritmo propuesto es hasta aproximadamente 14 veces más rápido para patrones de tamaño mayor a 50 que el software *Agrep*, una herramienta estado-del-arte para búsquedas en texto. En general es más rápido que aplicar BM en el texto descomprimido para patrones de tamaño mayor a 20, e ineficiente para patrones cortos.

- **M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Text compression using antidictionaries. *Automata, Languages and Programming*, pages 702–702, 1999**

  Presenta un esquema de compresión basado en antidiccionarios. Un antidiccionario es una colección de palabras que no aparecen en el texto y por lo tanto pueden ser predecibles. Este conjunto de palabras predecibles son consideradas redundantes y pueden ser eliminadas del texto.

  El algoritmo de compresión genera un autómata finito de las palabras no aceptadas por el alfabeto. Durante el procesamiento del texto original si se llega a un estado final entonces se escribe la otra letra.

Esta técnica genera decompresores muy rápido. Los compresores y descompresores pueden ser paralelizables debido a que para la descompresión es únicamente necesario una pequeña parte a la izquierda de una ocurrencia. El algoritmo de compresión es estático: necesita procesar dos veces el texto, una para la creación del antidiccionario y otra para aplicar la compresión.

- **E. S. De Moura, G Navarro, N Ziviani, and R Baeza-Yates. Direct pattern matching on compressed text. In *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings*, pages 90–95, 1998**

  Presenta un esquema de compresión que permite búsquedas directamente en el texto comprimido sin necesidad de descompresión explicita. El esquema también permite búsquedas aproximadas.

  El esquema hace uso de codificación Huffman usando bytes en lugar de bits, y el árbol tiene grado 128 en lugar de grado 2. Un bit es usado para especificar el comienzo de una palabra, y los 7 restantes para la representación de las palabras claves (*codewords*). La elección de estas características para el algoritmo de compresión permite descompresión local en cualquier parte del texto comprimido, además permite una descompresión más rápida puesto que no se necesitan operaciones de desplazamiento ni enmascaramiento a nivel de bits para el procesamiento del texto comprimido.

  El algoritmo de búsqueda primero comprime el patrón dado usando la misma codificación que usada para la compresión del texto, y seguido se realiza la búsqueda del patrón directamente en el texto comprimido. Para búsquedas exactas usaron el algoritmo de Boyer-Moore (BM) [11]. Para búsquedas aproximadas se divide el patrón en partes y usando una heurística basada en el tamaño y el valor del código de cada parte se selecciona una parte y se aplica un algoritmo de búsqueda de multi-patrones cuando se encuentra una ocurrencia, se revisan las demás partes para ver si hubo una ocurrencia en la posición dada. Este algoritmo no descomprime el texto de ninguna manera a diferencia de [17].

  Los resultados experimentales muestran que cuando se realizan búsquedas complejas o aproximadas, el algoritmo propuesto es hasta 8 veces más rápido que *Agrep*. Sin embargo, este esquema es únicamente útil para lenguajes naturales, lo cual reduce el espectro de aplicabilidad.

- **E.S. De Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast searching on compressed text allowing errors. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306. ACM, 1998**

  Presenta el esquema rápido de compresión y descompresión para lenguajes naturales que permite una aplicación eficiente de búsquedas en texto com-

primido sin descompresión explicita. Además presenta el primer algoritmo para búsqueda aproximada directamente en texto comprimido.

El esquema de compresión usa códigos de Huffman a nivel de bytes con un árbol de grado 256 porque permite alcanzar mejores velocidades de descompresión. La compresión y descompresión asociada con este esquema es extremadamente rápida, y logra mejores índices de compresión en lenguajes naturales que la familia de algoritmos LZ [5, 6, 7, 8]. Además permite descompresión local desde cualquier parte del texto comprimido.

El algoritmo de búsqueda es basado en Shift-Or [18] con un Autómata Finito No Determinista que acepta las palabras en el patrón para la verificación de falsos positivos con un filtro Boyer-Moore (BM) [11]. Para realizar una búsqueda primero se compara el patrón con cada patrón marcando en una máscara de bits la posición si hay una ocurrencia. En esta etapa si alguna de las palabras del patrón no se encuentra en el alfabeto, la búsqueda termina y el patrón no puede ser encontrado en el texto. Luego, se realiza la búsqueda en el texto comprimido byte por byte recorriendo el árbol de Huffman. Al llegar a una hoja se aplica la máscara de bits asociada con el código a un autómata para verificación del patrón.

El esquema puede ser adaptado para búsquedas aproximadas simplemente aplicando el algoritmo apropiado en la etapa de preprocesamiento del patrón contra el alfabeto.

Los resultados experimentales muestran que el algoritmo logra un índice de compresión del 30 %. Para patrones sencillos el algoritmo propuesto es hasta 2 veces más rápido que *Agrep* sobre el texto descomprimido. Y para búsqueda con patrones complejos es hasta 8 veces más rápido.

- **M. Farach and M. Thorup. String Matching in Lempel-Ziv Compressed Strings. *Algorithmica*, 20(4):388–404, 1998**

  Presenta el primer algoritmo aleatorio pseudo-óptimo no trivial para búsqueda en texto comprimido con LZ77 [5], del cual afirman que no hay manera no trivial de realizar una búsqueda en texto comprimido.

  Define un algoritmo *competitivo* como aquel que su tiempo de ejecución es $O(U+P)$ y un algoritmo *oportunista* si su tiempo de ejecución es $o(U+P)$. Un algoritmo *óptimo* es competitivo y oportunista. Donde U es el tamaño del texto descomprimido y P es el tamaño del patrón.

  El algoritmo propuesto hace uso del método de *Fingerprint* [20] y bajo la noción de que para cadenas con baja entropía la información tiende a aglomerarse en ciertas partes, a partir de esto se puede descomprimir únicamente las partes relevantes.

  La complejidad del algoritmo propuesto es $O(N \log^2(U/N) + P)$; donde N es el tamaño del texto comprimido, U es el tamaño del texto descomprimido y P es el tamaño del patrón.

- **DA Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 1952**

  Presenta un algoritmo para generación de códigos de mínima redundancia con la menor longitud posible del mensaje.

  El algoritmo es muy sencillo y puede ser resumido de la siguiente forma: a partir de una lista ordenada del alfabeto con las probabilidades para cada mensaje. Se toman los dos mensajes con menor frecuencia, se unen en un mensajes compuesto que tendrá como probabilidad compuesta la suma de las probabilidades de los elementos unidos. El proceso se repite hasta que únicamente queden dos mensajes restantes. La probabilidad compuesta de la unión éstos dos últimos mensajes debe dar 1,0.

  Los códigos son generados a partir de la raíz del árbol formado por el algoritmo anterior, dando los valores $\{0, 1\}$ a cada uno de los hijos de los nodos (e.g. 0 para el nodo izquierdo y 1 para el nodo derecho). El código de cada mensaje (los nodos hojas) es formado por el camino desde la raíz con su representación en binario.

- **T. Kida, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. A unifying framework for compressed pattern matching. In *String Processing and Information Retrieval Symposium, 1999 and International Workshop on Groupware*, pages 89–96. IEEE, 2002**

  Presenta un *framework* para el estudio de diferentes algoritmos de compresión de información basados en diccionarios. El framework propuesto es aplicable para métodos como la familia de algoritmos LZ [5, 6, 7, 8], Byte-Pair Encoding (BPE) [23] y Diccionario Estático.

  Define un *Collage System* como un sistema formal para representación de una cadena por medio de un diccionario D y una sequencia S de frases en D con operaciones de concatenación, truncamiento y repetición. Esta definición es usada como base para la presentación de un algoritmo general para búsqueda de patrones en texto comprimido basado en el algoritmo de [4] para búsquedas en texto comprimido con LZW [7]. El algoritmo generalizado para búsqueda de patrones en texto comprimido por *Collage Systems* esencialmente simula el movimiento del algoritmo Knutt-Morris-Pratt (KMP) [24] frase por frase sobre S reportando todas las ocurrencias del patrón dado. El algoritmo es fácilmente modificable para diferentes patrones con la personalización de dos funciones: $Jump$ y $Output$, que son la función de cambio de estado del autómata y la función de impresión de ocurrencias respectivamente.

  Los autores afirman que el truncamiento presente en algoritmos como LZ77 y LZSS [5, 8] ralentiza el proceso de búsqueda de patrones en texto comprimido.

- **T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Shift-And**

approach to pattern matching in LZW compressed text. In *Combinatorial Pattern Matching*, pages 1–13. Springer, 1999

Presenta un algoritmo para búsqueda en texto comprimido para LZW [7] basado en la técnica Shift-And [18]. Debido a la restricciones de Shift-And, el algoritmo únicamente puede ser usado para patrones de longitud menor o igual a 32 caracteres.

los resultados experimentales muestran que el algoritmo propuesto es aproximadamente 1,5 veces más rápido que la solución trivial de descomprimir y luego buscar sobre el texto descomprimido con la técnica Shift-And, y además de un algoritmo anterior de los mismos autores basado en Aho-Corasick (AC) [26]. Sin embargo, el algoritmo propuesto no puede ser mejor que la búsqueda directa sobre el texto descomprimido usando Shift-And.

El algoritmo puede ser adaptado para búsquedas aproximadas y a búsqueda de múltiples patrones.

- **T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In *Data Compression Conference, 1998. DCC'98. Proceedings*, pages 103–112. IEEE, 2002**

  Presenta dos algoritmo: uno que reporta las ocurrencias de múltiples patrones directamente sobre texto comprimido sin necesidad de descompresión explicita, y un segundo algoritmo para búsquedas de patrones simples. La técnica de compresión objetivo es LZW [7] y *Collage Systems* [22].

  El algoritmo de multi-patrones básicamente simula el movimiento de una máquina Aho-Corasick (AC) [26]. La búsqueda de patrones simples está basada en paralelismo de bits. Los resultados experimentales muestran que ambos algoritmos son más rápidos que una descompresión seguida de una búsqueda con *Agrep*. También cabe resaltar que los algoritmos son incluso más rápidos que una búsqueda directa sobre el texto descomprimido en el caso de tener discos de almacenamiento remotos.

  La complejidad del algoritmo propuesto para búsqueda de múltiples patrones es de $O(n + m^2 + r)$ haciendo uso de $O(n + m^2)$ de espacio; donde m es la longitud del patrón, r el número de ocurrencias del patrón. El algoritmo para búsqueda de patrones simples tiene por complejidad en tiempo de $O(\lceil m/w \rceil (n + r))$, después de un preprocesamiento en tiempo y espacio de $O(m)$.

- **S.T. Klein and D. Shapira. A new compression method for compressed matching. In *Data Compression Conference, 2000. Proceedings. DCC 2000*, pages 400–409. IEEE, 2002**

  Se plantea un nuevo algoritmo de compresión de substitución de texto basado en el algoritmo LZSS [8] que en vez de tener referencias a posiciones anteriores utiliza referencias a bloques de compresión siguientes. En general, una tripleta de elementos $(off, len, slide)$ es usada para la compresión

haciendo referencia a los *len* últimos elementos a una distancia off de la posición actual, moviendo la posición actual *slide* posiciones.

El método no presenta buenos índices de compresión. Las pruebas empíricas presentan mejores resultados que el algoritmos original.

- **J Kärkkäinen, G Navarro, and E Ukkonen. Approximate String Matching on Ziv-Lempel Compressed Text. *Journal of Discrete Algorithms (JDA)*, 1(3/4), 2003**

  Presenta el primer algoritmo no-trivial para búsquedas aproximadas de patrones directamente en texto comprimido para la familia de algoritmos LZ [5, 6, 7, 8]. El algoritmo propuesto es basado en *Edit Distance* y es capaz de reportar todas las ocurrencias del patrón.

  El archivo comprimido se procesa bloque por bloque, razón por la cual el algoritmo propuesto tuvo que ser modificado para recibir bloques de compresión de LZ en vez de carácter por carácter. Para cada bloque se calcula la descripción y se actualiza el estado dependiendo de ésta. Las ocurrencias se comprueban por patrones terminados en el bloque actual y por patrones contenidos en el bloque actual.

  El algoritmo propuesto presenta una complejidad de peor caso de $O(kmn+R)$ para encontrar todas las R ocurrencias de un patrón de tamaño m permitiendo k errores, y un tiempo promedio de $O(k^2n + R)$.

- **A. Lempel and J. Ziv. On the complexity of finite sequences. *Information Theory, IEEE Transactions on*, 22(1):75–81, 2002**

  Se presenta un nuevo criterio para evaluación de la complejidad de secuencias finitas basado en la cantidad de pasos necesarios para la construcción del texto haciendo uso de una regla general de construcción de subcadenas a partir de subcadenas anteriormente encontradas en el texto.

  Los autores presentan un estudio riguroso.

- **U Manber. A text compression scheme that allows fast searching directly in the compressed file. *ACM Transactions on Information Systems (TOIS)*, 1997**

  Presenta un nuevo esquema de compresión de información que busca facilitar las búsquedas directamente en el texto comprimido sin descompresión explicita y mejorar los tiempos de búsqueda al tener que revisar una menor cantidad de bits. La compresión funciona a nivel de bytes para mejorar el desempeño práctico del algoritmo debido a que se evitan realizar operaciones de desplazamiento y enmascaramiento cuando se trabaja a nivel de bits. La aplicación de cualquier algoritmo de búsqueda sobre cadenas de caracteres puede ser usado con este esquema sin necesidad de modificación alguna.

  El algoritmo de compresión reemplaza los pares de caracteres más frecuentes del texto por una palabra clave (*codeword*) especifica. Encontrar

la mejor combinación de parejas de tal manera que se reduzca óptimamente la longitud total del texto comprimido lo resuelven usando una algoritmo aleatorizado con un 100 iteraciones, debido a que en experimentos iniciales se encontró que con 20 iteraciones se encontraba una solución con menos de 3 % de error.

El algoritmo de búsqueda después de procesar la lista de parejas al inicio del archivo aplica el algoritmo de compresión al patrón dado y luego aplica el algoritmo de búsqueda sobre el texto comprimido con el patrón comprimido obteniendo una lista de ocurrencias. A continuación, cada parte del texto donde ocurrió una ocurrencia es descomprimida y se aplica, de ser necesario, nuevamente el algoritmo de búsqueda sobre el texto descomprimido para filtrar la salida.

El esquema de compresión no obtiene buenos índices de compresión, aproximadamente 30 % de compresión.

- **T. Matsumoto, T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Bit-parallel approach to approximate string matching in compressed texts. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 221–228. IEEE, 2002**

  Presenta el primer algoritmo para búsqueda aproximada en texto comprimido para *Collage Systems* [22] basado en paralelismo de bits. El algoritmo hace uso de un Autómata Finito No Determinista (AFN) para el problema de búsqueda aproximada junto con paralelismo de bits para agrupar conjunto de estados del autómata en una palabra de computadora. Sin embargo, el algoritmo no funciona para encontrar todas las ocurrencias del patrón.

  Además, los resultados experimentales muestran que el algoritmo no es práctico, puesto que es más lento que la solución trivial de descomprimir y luego realizar la búsqueda aproximada sobre el texto descomprimido.

- **A. Moffat and A. Turpin. Efficient construction of minimum-redundancy codes for large alphabets. *IEEE Transactions on Information Theory*, 44(4):1650–1657, July 1998**

  Presenta un algoritmo para construcción de códigos de mínima redundancia para alfabetos largos que además puede ser usado para generación de códigos de mínima redundancia con restricción de longitud. El algoritmo usa Run-Length Encoding (RLE) en una primera étapa, luego genera un grafo a partir del resultado de RLE y se realizan las uniones de los nodos en orden descendiente de pesos sin pérdida de los nodos unidos. Al final se recorre el grafo por orden descendiente de pesos y frecuencias generando la lista de códigos claves (*codewords*).

  También se puede modificar el algoritmo propuesto para generación aproximada de códigos Huffman dividiendo la lista de elementos en r partes

y asignando la misma frecuencia a cada parte. La generación de códigos con restricción de longitud también pueden ser generados.

La complejidad en tiempo y espacio es de $O(r + r\log(n/r))$, la cual es $o(n)$ cuando r es $o(n)$.

- **E Silva de Moura and G Navarro. Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems (TOIS)*, 2000**

  Se presenta una nueva técnica de compresión de información usando códigos Huffman basados en palabras [34] que permite realizar diferentes tipos de búsquedas de forma rápida. También se plantean dos algoritmos para la búsqueda directa en el texto comprimido usando códigos Huffman con marcas especiales para representación del comienzo de una palabra y otro algoritmo usando códigos Huffman sin marcas combinado con el algoritmo Shift-Or [18] para la verificación de los patrones. Los resultados experimentales son buenos para el algoritmo.

  La nueva técnica no presenta índices de compresión equiparables con la familia de algoritmos LZ [5, 6, 7, 8]. Además, sólo es útil en textos de lenguaje natural con codificación de 1-byte y que hacen uso de espacio como separador de palabras. Sin embargo, presenta muy buenos resultados para las búsquedas directas sobre el texto comprimido teniendo como base textos largos (mayores a 10MB). Es preferible el uso del segundo algoritmo propuesto.

- **G Navarro. Regular Expression Searching on Compressed Text. *Journal of Discrete Algorithms (JDA)*, 1(5/6):423–443, 2003**

  Se presenta el primer algoritmo de búsqueda usando expresiones regulares sobre texto comprimido sin descompresión explicita. Se basa especialmente sobre los algoritmos LZ78 y LZW [6, 7], y en técnicas de *paralelismo de bits*. El algoritmo es una variantes de un algoritmo de un Autómata Finito Determinista (AFD) basado en paralelismo de bits modificado para revisar por bloques de información (de LZ78) en vez de caracteres. Cada bloque puede estar relacionado con otros bloques o caracteres.

  En términos generales el algoritmo propuesto revisa un texto comprimido $Z = b_1, b_2 \dots b_n$, expresado como un conjunto de n *bloques*; bloque por bloque procesando la cadena representada por éste en el texto original. Presenta resultados experimentales usando *Compress* como de algoritmo de compresión y dos algoritmos de búsqueda, uno basado en AFD con paralelismo de bits y *nrgrep*. En las pruebas realizadas el algoritmo resulta ser al menos 20 % más rápido que descomprimir y luego aplicar bit-FDA o Nrgrep.

- **Gonzalo Navarro and Mathieu Raffinot. A General Practical Approach to Pattern Matching over Ziv-Lempel Compressed Text.**

In Maxime Crochemore and Mike Paterson, editors, *Combinatorial Pattern Matching*, volume 1645 of *Lecture Notes in Computer Science*, pages 14–36. Springer Berlin / Heidelberg, 1999

Se presenta un algoritmo de búsqueda en texto comprimido con LZ77 [5]. Se encuentra que el mismo algoritmo puede ser aplicado a LZ78 [6] y que además se desempeña mejor. Se presenta una nueva técnica de compresión de información basada en las características del algoritmo de compresión LZ77 manteniendo la misma velocidad de búsqueda sobre texto comprimido notada en LZ78.

La nueva técnica propuesta es subóptima porque se desperdician bits para expresar *flags* y el almacenamiento de la información de compresión no es eficiente, aunque en los resultados experimentales prácticos demuestre casi tan buen índice de compresión como LZ77. Logra obtener mejores tiempos con la nueva técnica híbrida, con un algoritmo más sencillo que el propuesto para LZ77, para búsquedas en texto comprimido sin descompresión explicita que el paradigma trivial de descomprimir luego buscar usando algoritmos como Sunday [37]. Se deja claro que para algoritmos *adaptativos* como la familia de algoritmos LZ, la localidad referencial es el gran inconveniente para la resolución del problema de búsqueda sobre texto comprimido.

- **G. Navarro and J. Tarhio. LZgrep: A Boyer-Moore string matching tool for Ziv-Lempel compressed text. *Software: Practice and Experience*, 35(12):1107–1130, 2005**

  Presenta un algoritmo para búsqueda sobre texto comprimido sin descompresión explicita usando LZ78 y LZW basado en Boyer-Moore (BM) [11] aprovechando la propiedad para omitir comprobaciones innecesarias. Una herramiento *LZgrep* implementando esta técnica está disponible.

  El algoritmo BM hace uso de los caracteres explícitos de cada bloque de la compresión LZ78 y LZW para realizar los desplazamientos del patrón. También se modificó una versión para que acepte q-tuplas de caracteres que funciona mejor con alfabetos pequeños que la versión BM normal. Otra optimización para lograr desplazamientos por bloques completos de compresión fue implementada.

  En total 5 técnicas fueron implementadas, 3 técnicas diferentes a parte de las dos mencionadas anteriormente para comparación. Para patrones simples la versión multi-carácter de BM presenta los mejores resultados para secuencias de ADN y la versión normal de BM se comportó mejor en texto naturales en inglés.

  En búsquedas multi-patrones la versión BM con procesamiento de bloques se desempeña mejor para secuencias de ADN para patrones cortos y la versión multi-carácter para patrones largas.

  Los resultados experimentales muestran que se alcanzan hasta un 50 % más de velocidad en comparación con en el enfoque trivial de descomprimir y

luego realizar una búsqueda sobre el texto descomprimido.

- **G. Navarro and J. Tarhio. Boyer-Moore String Matching over Ziv-Lempel Compressed Text. In *Combinatorial Pattern Matching*, pages 166–180. Springer, 2000**

  Presenta un algoritmo basado en el algoritmo Boyer-Moore (BM) [11] para búsqueda sobre texto comprimido aplicable para las técnicas LZ78 y LZW [6, 7] que mejora en aproximadamente un 30 % el tiempo de búsqueda en comparación con los algoritmos más eficientes.

  El algoritmo hace uso de los caracteres visibles en el texto comprimido para realizar los desplazamientos del patrón de BM. Diferentes variaciones del algoritmo fueron implementadas para ser comprobadas experimentalmente.

  Todas las implementaciones tuvieron pésimos índices de compresión. Sin embargo, los tiempo de descompresión y búsqueda mejoraron. La implementación que más se destacó fue una para realizar desplazamientos en bloques completos de LZ78.

- **G Navarro, T Kida, and M Takeda. Faster approximate string matching over compressed text. *Data Compression Conference*, 2001**

  Presenta la primera solución práctica para el problema de búsqueda aproximada sobre texto comprimido con LZ78 y LZW [6, 7]. La técnica propuesta divide el patrón en $k+1$ partes disyuntas de igual longitud $\lfloor m/(k+1) \rfloor$ para luego aplicar un algoritmo de búsqueda de múltiples patrones sobre el texto comprimido. Cada vez que ocurra una ocurrencia de alguna de las partes, se aplica una descompresión local y se aplica un algoritmo de búsqueda aproximada sobre el texto descomprimido.

  Tres implementaciones fueron presentadas: una basada en Aho-Corasick (AC) [27], otra basada en Boyer-Moore (BM) [39], y la última basada en paralelismo de bits (PB) [25, 36].

  Los resultados experimentales muestran que esta técnica es 10 a 30 veces más rápida que los trabajos anteriores y 3 veces más rápida que la solución trivial de descomprimir luego aplicar una búsqueda aproximada estado-del-arte sobre el texto descomprimido para valores moderados de k/m (generalmente bajos), donde k es el número máximo de errores permitidos y m es la longitud del patrón dado. El algoritmo PB mostró mejores resultados en general, aunque el algoritmo BM se desempeña mejor para valores k/m menores a 10 %.

- **Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Byte Pair encoding: A text compression scheme that accelerates pattern matching, 1999**

  Presentan dos acercamientos al problema de búsqueda en texto comprimido usando Byte-Pair Encoding (BPE) [23]. El primero usa fuerza bruta

para descomprimir todos los posibles patrones para después aplica Aho-Corasick (AC) [26] y Shift-And [18], el segundo usa un autómata Knutt-Morris-Pratt (KMP) [24] modificado para procesar cambios de estados consecutivos.

Los resultados experimentales muestran que los algoritmos propuestos son aproximadamente entre 1,6 y 1,9 veces más rápidos que la búsqueda en sobre el texto original e incluso sobre LZW [7].

- **Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Speeding up pattern matching by text compression. *Algorithms and Complexity*, pages 306–315, 2000**

  Presenta un algoritmo para búsqueda en texto comprimido con Byte-Pair Encoding (BPE) [23]. El algoritmo simula el movimiento del algoritmo Knutt-Morris-Pratt (KMP) [24] modificado para recibir cada *token* de texto comprimido y cambiar los estados correspondientes a la frase representada. Los *tokens* tiene como tamaño 8 bits.

  Los resultados experimentales muestran que el algoritmo es mejor que Shift-And [18] sobre texto comprimido y texto descomprimido, KMP sobre texto descomprimido. Sin embargo, *Agrep* sobre texto descomprimido sigue siendo más rápido.

  El tiempo de compresión es mejorado generando una tabla de substitución a partir de una parte del texto para mejorar el tiempo de compresión. Entre otras cosas, BPE permite descompresión local.

- **Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa. A Boyer-Moore Type Algorithm for Compressed Pattern Matching. In *Combinatorial Pattern Matching*, pages 181–194. Springer, 2000**

  Presenta un algoritmo de búsqueda de patrones en texto comprimido basado en Boyer-Moore (BM) [11], que puede ser aplicado a cualquier método de compresión de información definido dentro de un *Collage System* [22]. Los autores escogieron Byte-Pair Encoding (BPE) [23] como método de compresión porque cumple con las características deseadas de un Collage System.

  El algoritmo BM revisa el patrón de derecha a izquierda contra cada *token* del texto comprimido y desplaza el patrón usando una función de desplazamiento. Por cada *token*, primero reporta todas las ocurrencias contenidas en éste, después revisa y reporta todas las ocurrencias terminadas en el *token* actual, por último calcula la función de desplazamiento del patrón.

  Los resultados experimentales muestran que el algoritmo propuesto es aproximadamente entre 1,2 y 3,0 veces más rápido que el software *Agrep*, una herramienta estado-del-arte para búsqueda de patrones en texto.

- **Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Pattern matching in text compressed by using antidictionaries. In *Combinatorial Pattern Matching*, pages 37–49. Springer, 1999**

  Presenta un algoritmo para búsqueda de patrones en texto comprimido usando antidiccionarios [15] que es linealmente proporcional a la longitud del texto comprimido. El algoritmo simula el movimiento del autómata Knuth-Morris-Pratt (KMP) [24] para encontrar las ocurrencias del patrón en el texto.

  Los autores aseguran que el uso del algoritmo Shift-And [18] en vez del autómata KMP mejorará el rendimiento práctico para patrones de largo menor a 32 caracteres. No hay resultados experimentales del algoritmo propuesto pero se asegurará que los resultados preliminares demuestran la eficiencia del algoritmo propuesto.

- **M. Takeda, S. Miyamoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa. Processing text files as is: Pattern matching over compressed texts, multi-byte character texts, and semi-structured texts. In *String Processing and Information Retrieval*, pages 215–220. Springer, 2002**

  Presenta un algoritmo eficiente para búsqueda de patrones en texto comprimido que acepta lenguajes con codificación variable en múltiples bytes como el japonés que no emplea soluciones triviales para normalización de los caracteres. Además no sacrifica velocidad ni utiliza detectión de falsos positivos.

  El algoritmo propuesto, PMM, fusiona un Autómata Finito Determinista (AFD) para el reconocimiento de los palabras claves (*codewords*) y una máquina Aho-Corasick (AC) [26] modificada para correr byte por byte. Experimentos sobre texto comprimido con compresión Huffman demostró que PMM es más rápido que aplicar AC sobre el texto original.

  La técnica también es aplicable a documentos XML con una modificación para correr con una pila para el almacenamiento de las etiquetas encontradas. En comparación con una búsqueda usando AC, PMM demostró ser aproximadamente 1,2 a 1,5 veces más rápido. Los resultados experimentales demuestran que es aproximadamente de 7 a 10 veces más rápido que el software *sgrep*, una herramienta estado-del-arte para el procesamiento de documentos estructurados.

- **M. Takeda, Y. Shibata, T. Matsumoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa. Speeding up string pattern matching by text compression: The dawn of a new era. *Transactions of Information Processing Society of Japan*, 42(3):370–384, 2001**

  Se presenta una técnica enfocada en la compresión de cualquier tipo de texto, incluso texto en alfabeto Unicode como el japonés. La compresión es basada en Byte-Pair Encoding (BPE) [23]. También se presentan dos

algoritmos para las búsquedas en el texto comprimido uno que es una variante del algoritmo Boyer-Moore [11] y otro que es una variante del algoritmo Knuth-Morris-Pratt [24].

Se presentan resultados experimentales de ambos algoritmos en comparación con otros algoritmos presentes en la literatura. La técnica mostrada puede ser usada con textos con codificación en múltiples bytes, a diferencia de otras técnicas que son únicamente aptas para texto con codificación en 1-byte.

- **J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 2003(3):337—-343, 2002**

  Presenta un algoritmo universal para compresión de información secuencial que tiende a generar índices de compresión óptimos. El algoritmo propuesto consiste de una regla para generación de subcadenas a partir de un alfabeto A, las cuales no pueden sobrepasar un tamaño L, asignándoles palabras claves (*codewords*) únicos descifrables.

  El artículo presenta una extensa demostración del algoritmo propuesto. No hay resultados experimentales acerca del algoritmo pero la complejidad del mismo es demostrada en el artículo.

  Denominado LZ1 o LZ77.

- **J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530—-536, 2003**

  Presenta el algoritmo LZ2 o LZ78, que a diferencia de su predecesor el LZ77 [5] permite búsquedas de las subcadenas tanto en el pasado desde un diccionario como en el futuro leyendo desde el *buffer* de entrada.

Referencias

[1] A. Amir and C. Benson. Efficient two-dimensional compressed matching. *Data Compression Conference, 1992*, 2002.

[2] A. Amir and G. Benson. Two-dimensional periodicity and its applications. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 440–452. Society for Industrial and Applied Mathematics, 1992.

[3] A. Amir, G.M. Landau, and D. Sokol. Inplace run-length 2d compressed search. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 817–818. Society for Industrial and Applied Mathematics, 2000.

[4] Amihood Amir, Gary Benson, and Martin Farach. Let Sleeping Files Lie: Pattern Matching in Z-Compressed Files. *Journal of Computer and System Sciences*, 52(2):299–307, 1996.

[5] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 2003(3):337—-343, 2002.

[6] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530—-536, 2003.

[7] T. Welch. Technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

[8] J.A. Storer and T.G. Szymanski. Data compression via textual substitution. *Journal of the ACM (JACM)*, 29(4):928–951, 1982.

[9] T. Bell, M. Powell, A. Mukherjee, and D. Adjeroh. Searching BWT compressed text with the Boyer-Moore algorithm and binary search. In *Data Compression Conference, 2002. Proceedings. DCC 2002*, pages 112–121. IEEE, 2002.

[10] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm, 1994.

[11] R S Boyer and J S Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[12] NR Brisaboa and A Fariña. Lightweight natural language text compression. *Information Retrieval*, 2007.

[13] E Silva de Moura and G Navarro. Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems (TOIS)*, 2000.

[14] L. Chen, S. Lu, and J. Ram. Compressed pattern matching in DNA sequences. In *CSB '04 Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference*. IEEE Computer Society, 2004.

[15] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Text compression using antidictionaries. *Automata, Languages and Programming*, pages 702–702, 1999.

[16] E. S. De Moura, G Navarro, N Ziviani, and R Baeza-Yates. Direct pattern matching on compressed text. In *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings*, pages 90–95, 1998.

[17] E.S. De Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast searching on compressed text allowing errors. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306. ACM, 1998.

[18] R. Baeza-Yates and G.H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74—-82, 1992.

[19] M. Farach and M. Thorup. String Matching in Lempel-Ziv Compressed Strings. *Algorithmica*, 20(4):388–404, 1998.

[20] R.M. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[21] DA Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 1952.

[22] T. Kida, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. A unifying framework for compressed pattern matching. In *String Processing and Information Retrieval Symposium, 1999 and International Workshop on Groupware*, pages 89–96. IEEE, 2002.

[23] P. Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994.

[24] D.E. Knuth, J.H. Morris Jr, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323, 1977.

[25] T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Shift-And approach to pattern matching in LZW compressed text. In *Combinatorial Pattern Matching*, pages 1–13. Springer, 1999.

[26] A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[27] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In *Data Compression Conference, 1998. DCC'98. Proceedings*, pages 103–112. IEEE, 2002.

[28] S.T. Klein and D. Shapira. A new compression method for compressed matching. In *Data Compression Conference, 2000. Proceedings. DCC 2000*, pages 400–409. IEEE, 2002.

[29] J Kärkkäinen, G Navarro, and E Ukkonen. Approximate String Matching on Ziv-Lempel Compressed Text. *Journal of Discrete Algorithms (JDA)*, 1(3/4), 2003.

[30] A. Lempel and J. Ziv. On the complexity of finite sequences. *Information Theory, IEEE Transactions on*, 22(1):75–81, 2002.

[31] U Manber. A text compression scheme that allows fast searching directly in the compressed file. *ACM Transactions on Information Systems (TOIS)*, 1997.

[32] T. Matsumoto, T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Bit-parallel approach to approximate string matching in compressed texts. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 221–228. IEEE, 2002.

[33] A. Moffat and A. Turpin. Efficient construction of minimum-redundancy codes for large alphabets. *IEEE Transactions on Information Theory*, 44(4):1650–1657, July 1998.

[34] A. Moffat. Word-based text compression. *Software: Practice and Experience*, 19(2):185–198, 1989.

[35] G Navarro. Regular Expression Searching on Compressed Text. *Journal of Discrete Algorithms (JDA)*, 1(5/6):423–443, 2003.

[36] Gonzalo Navarro and Mathieu Raffinot. A General Practical Approach to Pattern Matching over Ziv-Lempel Compressed Text. In Maxime Crochemore and Mike Paterson, editors, *Combinatorial Pattern Matching*, volume 1645 of *Lecture Notes in Computer Science*, pages 14–36. Springer Berlin / Heidelberg, 1999.

[37] D.M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132–142, 1990.

[38] G. Navarro and J. Tarhio. LZgrep: A Boyer-Moore string matching tool for Ziv-Lempel compressed text. *Software: Practice and Experience*, 35(12):1107–1130, 2005.

[39] G. Navarro and J. Tarhio. Boyer-Moore String Matching over Ziv-Lempel Compressed Text. In *Combinatorial Pattern Matching*, pages 166–180. Springer, 2000.

[40] G Navarro, T Kida, and M Takeda. Faster approximate string matching over compressed text. *Data Compression Conference*, 2001.

[41] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Byte Pair encoding: A text compression scheme that accelerates pattern matching, 1999.

[42] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Speeding up pattern matching by text compression. *Algorithms and Complexity*, pages 306–315, 2000.

[43] Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa. A Boyer-Moore Type Algorithm for Compressed Pattern Matching. In *Combinatorial Pattern Matching*, pages 181–194. Springer, 2000.

[44] Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Pattern matching in text compressed by using antidictionaries. In *Combinatorial Pattern Matching*, pages 37–49. Springer, 1999.

[45] M. Takeda, S. Miyamoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa. Processing text files as is: Pattern matching over compressed texts, multi-byte character texts, and semi-structured texts. In *String Processing and Information Retrieval*, pages 215–220. Springer, 2002.

[46] M. Takeda, Y. Shibata, T. Matsumoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa. Speeding up string pattern matching by text compression: The dawn of a new era. *Transactions of Information Processing Society of Japan*, 42(3):370–384, 2001.