UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

Análisis de genomas basado en técnicas de compresión de datos

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

ANTONIO CONTRERAS GARCÍA

DIRECTOR DE TESIS:
DR. PEDRO EDUARDO MIRAMONTES VIDAL

2009

Datos del Jurado

1. Datos del alumno.
   Contreras
   García
   Antonio
   55 79 90 31
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Ciencias de la Computación
   096320606

2. Datos del asesor
   Dr.
   Pedro Eduardo
   Miramontes
   Vidal

3. Datos del sinodal 1
   Dr.
   Germinal
   Cocho
   Gil

4. Datos del sinodal 2
   Dr.
   José Antonio
   Neme
   Castillo

5. Datos del sinodal 3
   Dr.
   José de Jesús
   Galaviz
   Casas

6. Datos del sinodal 4
   M. en C.
   José Luis
   Gutiérrez
   Sánchez

7. Datos del trabajo escrito
   Análisis de genomas basado en técnicas de compresión de datos.
   120 p.
   2009

A mis padres, mis hermanos
A Neli
A mis abuelas y en recuerdo de mis abuelos.

Agradecimientos

A mis padres José y Malena por todo el apoyo brindado en mi vida, sin él no habría consumado este logro, el cual también es obra de ustedes. Les agradezco su paciencia, sus palabras de aliento, su exigencia, su confianza, pero sobre todo el tenerlos como padres y guías en este camino, tanto en lo personal como en lo profesional y que aún no termina.

A mis hermanos Bere y Angel que siempre han estado apoyándome en todo momento. Ustedes han sido un ejemplo, porque siempre hemos compartido momentos que nos han ayudado a crecer como personas.

A mis abuelos y abuelas, a Luis porque siempre estuvo conmigo dándome momentos de aliento y que nunca dudó de esta satisfacción como resultado de poder realizar este sueño; siempre serás parte de él. A Margarita que siempre ha estado con ese apoyo y cariño tan valioso y que nunca deja de dar. A Joel y Antonia que son un ejemplo de vida del que se aprende cada día.

A mis tíos que me han brindado su apoyo y conocimiento a lo largo de mi vida y que son parte fundamental en mi desarrollo como persona. A mis primos por su compañia y su entusiasmo.

A mi profresor y tutor Dr. Pedro Miramontes por su paciencia, su apoyo, sus regaños, por su capacidad de transmitir y compartir el conocimiento así como de ser un formador de científicos en la facultad. Gracias a él, tengo el gusto por la investigación y por su confianza para realizar este proyecto. También agradezco al Dr. Antonio Neme por su valioso tiempo en este proyecto, así como de sus oportunos comentarios. A los demás sinodales, el Dr. Germinal Cocho, el Dr. José Galaviz y el M. en C. José Luis Gutierrez por haber revisado este trabajo.

A mis compañeros(as) y amigos(as) de la facultad, por todos los momentos que compartimos durante la carrera. Por esos días en que las tareas, las prácticas y los exámenes nunca faltaban. A los fantásticos, a Gustavo que durante todo este tiempo nunca falto su apoyo y su ayuda, Emmanuel que siempre brindó toda la ayuda posible para concluir este proyecto, Omar por

todos los buenos momentos en la carrera, Jesús y Vicente por su apoyo personal y académico; a Vania por sus palabras de aliento para terminar la tesis, pero sobre todo por la amistad que encontré en ustedes.

Al Dr. Eduardo Andrade y a todos los integrantes del laboratorio del acelerador Van de Graaff de 5.5 MeV. del IFUNAM; Julio por todos sus comentarios, Miguel, Oscar y Corina por su apoyo y comprensión para concluir esta etapa.

A mis amigos de la prepa, los otros fantásticos Cristian, Fernando y Johnatan. A Erika, Olivia, Angélica y Alba que siempre están presentes y por sus palabras de apoyo.

A Rubén por su apoyo bibliotecario y sus comentarios que siempre son positivos.

Pero sobre todo le agradezco a Neli, que me ha acompañado en este largo camino, apoyándome en los momentos difíciles, soportando y entendiendo los obstáculos que se han presentado y teniendo la tranquilidad para alentarme en todo momento. También por toda la ayuda y confianza que me ha brindado y que sin ella hubiera sido más difícil. Gracias por tu compañia.

Por mi raza hablará el espíritu.
México, 2009.

Science is facts; just as houses are made of stones, so is science made of facts; but a pile of stones is not a house and a collection of facts is not necessarily science.

Henri Poincaré

Índice general

1. Introducción **1**

2. DNA **3**
2.1. Cromosomas y el material genético 4
2.2. La estructura del DNA 6
2.3. DNA - Proteína 9
2.3.1. Replicación del DNA y síntesis de RNA 10
2.3.2. Código genético 13
2.3.3. Síntesis de proteínas 15
2.4. Mutación 16
2.5. Proteínas; estructura y función biológica 19

3. Fundamentos teóricos de la complejidad algorítmica **23**
3.1. Teoría de la información 23
3.1.1. Codificación de la información 24
3.1.2. Bases matemáticas para la compresión 26
3.2. Complejidad algorítmica 30
3.2.1. Complejidad de Kolmogorov 31
3.2.2. Teoría de información algorítmica 33

4. Compresión de datos **37**
4.1. Conceptos básicos 40
4.2. Técnicas de compresión 44
4.2.1. Compresión *lossless* 44
4.2.2. Compresión *lossy* 46
4.2.3. Medidas de desempeño 46
4.3. Compresión *lossless* 48
4.3.1. Algoritmos de compresión estadísticos 49

4.3.2. Algoritmos de compresión basados en diccionario . . . 56
4.3.3. Compresión mediante transformada 66

5. Análisis de secuencias de DNA **71**
5.1. Eucariontes, bacterias y arqueas 71
5.2. Compresión y DNA 73
5.2.1. Biocompress-2 75
5.3. Análisis de genomas basado en técnicas de compresión de datos 80

6. Resultados y discusión **99**
6.1. Conclusiones 102

Bibliografía **106**

Índice de figuras

2.1. Cromosomas humanos. 4
2.2. Composición y organización de los genes en los cromosomas. . 5
2.3. Las bases genéticas presentes en los ácidos nucleicos. 6
2.4. Bases complementarias. 7
2.5. Estructura de la molécula del DNA 8
2.6. Replicación del DNA 11
2.7. Transcripción, síntesis del RNA 12
2.8. Código genético. 14
2.9. Síntesis de proteínas 17
2.10. Tipos de mutación 18
2.11. RNA mensajero y proteínas 19
2.12. Proteínas; estructura y función 20

4.1. Compresión y reconstrucción 44
4.2. Creación de un árbol binario dadas las probabilidades de los símbolos en el mensaje a comprimir 52
4.3. LZ77 59
4.4. Diccionario Lempel-Ziv. 63

5.1. Estructura secundaria del RNA. 76
5.2. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, con compresión *gzip*. 85
5.3. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, con compresión *bzip2*. 86

5.4. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, compresión con *biocompress-2*. 87
5.5. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 87
5.6. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 5,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 88
5.7. Porcentajes de compresión para el cromosoma 21 del genoma del *H. sapiens* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 89
5.8. Porcentajes de compresión para el cromosoma 21 del genoma del *H. sapiens* con *intervalos* de 5,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 90
5.9. Porcentajes de compresión para el cromosoma x de la *D. melanogaster* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 91
5.10. Porcentajes de compresión para el cromosoma x de la *D. melanogaster* con *intervalos* de 5,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 92
5.11. Porcentajes de compresión para el cromosoma 1 del genoma de *C. elegans* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 93
5.12. Porcentajes de compresión para el genoma del *Mycoplasma pulmonis* con *intervalos* de 1,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 94
5.13. Porcentajes de compresión para el genoma del *Mycoplasma pulmonis* con *intervalos* de 2,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 94
5.14. Porcentajes de compresión para el genoma de *Sulfolobus tokodaii* con *intervalos* de 1,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 96
5.15. Porcentajes de compresión para el genoma de *Sulfolobus tokodaii* con *intervalos* de 2,000 bases, con *biocompress-2*, *bzip2* y *gzip*. 96
5.16. Porcentajes de compresión *secuencia pseudo-aleatoria* con *intervalos* de 1,000 bases, con *biocompress-2*, *bzip2* y *gzip*. . . . 97

6.1. Gráfica que muestra el promedio del valor del %(C+G) del cromosoma 21 del *Homo sapiens* como una línea constante y al final se observa la zona en la que este porcentaje está por encima del promedio. 100
6.2. Gráfica donde se muestra que al analizar el %(C+G) del genoma de *Mycoplasma pulmonis*, la localización de los *vsa genes* es en el mismo lugar donde se obtienen porcentajes de compresión en el análisis de genomas. 101

Índice de cuadros

3.1. Codificación binaria de los dígitos decimales 24
3.2. Código binario I 25
3.3. Código binario II 26

4.1. Códigos de Huffman 53
4.2. Valores de las variables en la codificación aritmética 54

5.1. Genomas de organismos 81

Capítulo 1

Introducción

A lo largo de los años, la sociedad se ha caracterizado por tener grandes cambios sociales, culturales, económicos, científicos, etcétera. Estos cambios han permitido un gran avance en la ciencia y de nuevas tecnologías para facilitar y explorar nuevos campos de investigación con la finalidad de entender mejor nuestro entorno.

Debido al rápido crecimiento en las comunicaciones, gran parte del conocimiento se da gracias a que cada día existen nuevas formas de comunicación y acceso a un mundo de información que anteriormente era casi imposible tener. Esto permite que surgan nuevas disciplinas de estudio en las que se intercambian conocimientos para fines comunes y en los que se ha enfocado el desarrollo de nuevas investigaciones dentro de las ciencias.

La necesidad de tener y procesar más información ha hecho posible que las ciencias de la computación avancen, ya que cada vez es mas indispensable el uso de las computadoras y la capacidad de cómputo en la mayoría de las investigaciones así como en las aplicaciones que requiere el ser humano. El desarrollo de nuevas tecnologías han permitido que, por ejemplo, se haya llegado a tener una red mundial en la que sólo basta tener acceso a una computadora conectada a esta red y de ahí poder navegar por miles de sitios alrededor del mundo en busca de cualquier tipo de información.

En nuestros días, están surgiendo nuevos paradigmas en las ciencias, en las que algunas disciplinas científicas como las matemáticas, las ciencias de la computación y la biología se han reunido dando lugar a una interdisciplina,

en las que cada una de éstas se complementa y enriquece. Este enriquecimiento entre disciplinas ha permitido avances tecnológicos que hacen posible generar nuevos conocimientos y aplicaciones en diferentes ramas de la ciencia como es el caso de la biología y se permite la creación y el desarrollo de nuevas disciplinas como la bioinformática que a su vez, con todos sus avances, y la gran cantidad de estudios sobre el conjunto de información genética (DNA o genomas) de los seres vivos dieron origen a la genómica.

Debido a que es posible obtener los genomas completos de los organismos, se busca obtener información sobre las características de estas secuencias. Una herramienta que permite hacer este tipo de análisis son las técnicas de compresión de datos y este estudio se divide en las siguientes partes.

En el capítulo 2 se hace una descripción del DNA como molécula encargada de llevar la información de cada uno de los seres vivos. Se describe también el proceso de traducción de DNA a proteínas y el papel que desempeña la molécula del RNA.

En el capítulo 3 se revisan los fundamentos teóricos de la complejidad algorítmica para entender de la mejor manera posible los algortimos de compresión de datos, su funcionamiento, su desempeño y el análisis de las secuencias genéticas.

En el capítulo 4 se estudian las clasificaciones, las características y las implementaciones de algunos algoritmos de compresión. Se revisan las técnicas de compresión mas comunes como lo son los algoritmos de compresión *lossless*, utilizados en su mayoría para la compresión de datos y en este trabajo para la compresión de los genomas que se estudian.

Con la teoría analizada en éstos, en el capítulo 5 se presenta un algoritmo de compresión muy específico para secuencias sobre alfabetos particulares de 4 letras, que en este caso, corresponden a los nucleótidos del DNA. También se presenta un análisis de las secuencias de DNA o genomas utilizando las técnicas de compresión de datos obteniendo interesantes resultados para entender y obtener información contenida en ellas, así como su análisis en las diversas ramas de la ciencia que tienen como tarea, el estudio y entendimiento del DNA. A continuación se muestran las conclusiones.

Capítulo 2

DNA

Los seres vivos están conformados por células. Los más sencillos, los procariontes, como las bacterias, están constituidos por una sola célula, es decir son unicelulares. Los demás, son organismos que se denominan eucariontes, tales como el humano y las plantas que son pluricelulares, aunque también existen eucariontes unicelulares, como la levadura o las amibas.

Las moléculas de una célula pueden ser grandes o pequeñas; las moléculas grandes, conocidas como macromoléculas, son de tres tipos: DNA, RNA y proteínas. Estas son las moléculas de mayor interés, y se hacen a partir del ensamble de ciertas moléculas pequeñas en polímeros, que más adelante se describen.

Las células eucariontes tienen, en promedio, un tamaño varios miles de veces mayor al de las células procariontes y su estructura, organización y complejidad es también mucho mayor que de las bacterias.

La diferencia más importante entre las células de los procariontes y los eucariontes radica en que estos últimos tienen cromosomas contenidos en una membrana nuclear, conformando lo que se conoce con el nombre del núcleo de la célula eucarionte. En él, se lleva a cabo el proceso de transcripción del DNA en RNA, y el fenómeno del procesamiento del RNA, para formar moléculas maduras de RNA mensajero, de transferencia y ribosomal. El procesamiento del RNA es muy importante porque, en muchos casos, da lugar a varias moléculas de RNA mensajero diferentes a partir de un solo gen.

2.1. Cromosomas y el material genético

En las células de todo organismo existen dos tipos de macromoléculas biológicas en las que reside la información genética y funcional, mediante la cual la célula lleva a cabo sus funciones. El primer tipo de macromoléculas informacionales son los ácidos nucleicos: el DNA y el RNA; el otro tipo de macromolécula informacional son las proteínas.

El ácido desoxirribonucleico DNA, es la molécula en la que reside la información genética y esta molécula da origen a los cromosomas. Los cromosomas son estructuras formadas por la asociación de DNA con proteínas y también con moléculas de ácido ribonucleico RNA (Frankel, 2003). Las proteínas asociadas a los cromosomas llevan a cabo funciones reguladoras y estructurales. Al conjunto de material genético (DNA) de cada célula se le llama *genoma*.

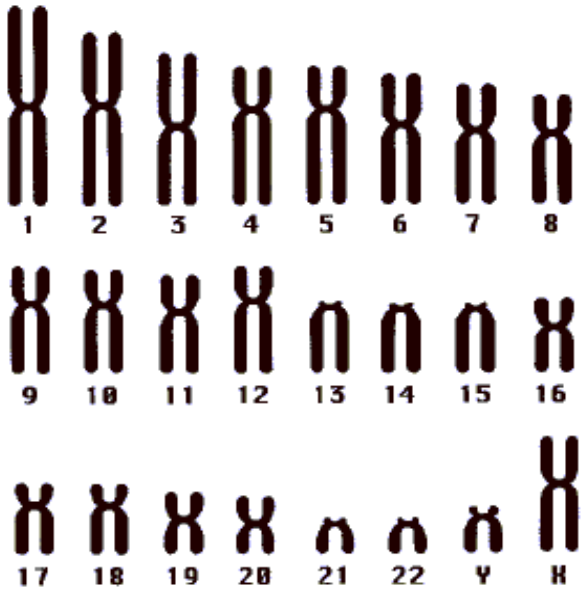


Figura 2.1. Cromosomas humanos.

La información genética reside en el DNA y los genes son segmentos específicos de esta cadena genética. Cada ser vivo tiene un número específico y diferente de cromosomas, con relación a los demás organismos vivos. En el caso del humano, este número es de 46 cromosomas (23 pares); en la mosca

de la fruta son 8 (4 pares); en el maíz, 20 (10 pares). La figura 2.1 muestra los 22 cromosomas somáticos y los dos cromosomas sexuales humanos.

En las células del cuerpo humano hay 23 pares de cromosomas. Cada cromosoma humano está formado por una sola molécula de DNA, la cual está asociada a miles de moléculas de proteínas, principalmente las llamadas histonas, cuya función principal es proporcionarle estructura al cromosoma. En todas y cada una de las células existen 23 pares de cromosomas (con excepción de los gametos: espermatozoides y óvulos, donde hay sólo 23 cromosomas); cada juego de 23 cromosomas proviene originalmente de cada uno de los padres (figura 2.1).

Los genes son segmentos de la molécula de DNA presente en cada cromosoma y los genes están organizados secuencialmente en los cromosomas, donde cada gen corresponde a un segmento del DNA que codifica una proteína específica (figura 2.2). Los genes son responsables de las características físicas de los individuos y se transmiten, como parte de los cromosomas, de padres a hijos.

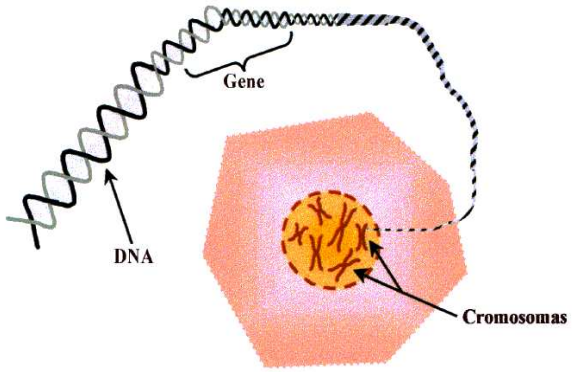


Figura 2.2. Composición y organización de los genes en los cromosomas.

2.2. La estructura del DNA

El DNA es la base de la herencia, es un polímero lineal, conocido como la "doble hélice", compuesto de pequeñas moléculas llamadas *desoxirribonucleótidos*. Un desoxirribonucleótido está formado por la unión de una base nitrogenada, una molécula de fosfato y una desoxirribosa. Estas bases nitrogenadas son cuatro: *adenina* (A), *citosina* (C), *guanina* (G) y *timina* (T). En este trabajo, la cadena de DNA se considera como una secuencia o un texto sobre el alfabeto de cuatro letras, $\mathcal{A} = \{A, C, G, T\}$. Una cadena de RNA, a su vez, será considerada como una secuencia o un texto sobre el alfabeto de los ribonucleótidos: $\mathcal{A} = \{A, C, G, U\}$, donde la timina (T) se reemplaza por el *uracilo* (U). (Arias, 2004)

Las proteínas también son polímeros y aquí las palabras están sobre un alfabeto de 20 aminoácidos, que más adelante se definen.

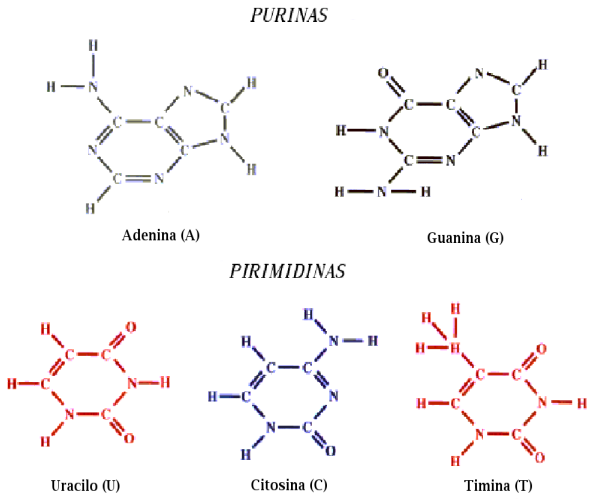


Figura 2.3. Las bases genéticas presentes en los ácidos nucleicos.

La figura 2.3 muestra las bases guanina (G), adenina (A), citosina (C), timina (T) y el uracilo (U) que existen en el DNA y el RNA. Estas bases, o letras genéticas, están unidas covalentemente al azúcar desoxirribosa y a la molécula de fosfato para formar los nucleótidos o monómeros del DNA (ver

figuras 2.4 y 2.5).

Las bases pueden pertenecer a dos tipos de familias, las purinas y las pirimidinas. La A y G pertenecen a las purinas y como están formadas por dos anillos se les llaman moléculas *grandes*, y por otro lado la C y T pertenecen a las pirimidinas y al estar formadas solamente por un anillo se les llama moléculas *pequeñas*.

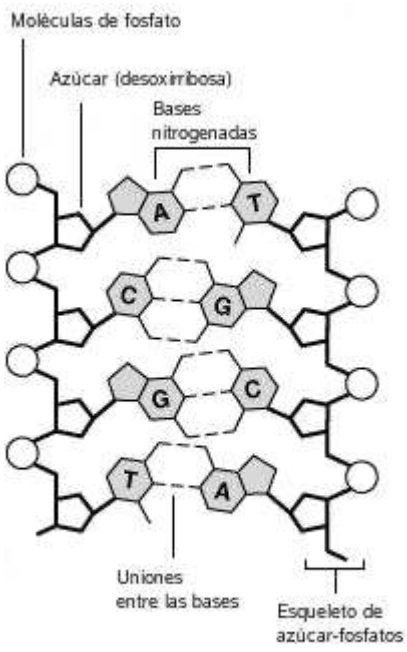


Figura 2.4. Bases complementarias.

El DNA está formado por dos hebras o cadenas complementarias. Las dos hebras permanecen unidas entre sí a través de puentes de hidrógeno, que se establecen entre cada dos nucleótidos complementarios. Lo anterior significa que en una molécula de DNA, siempre habrá la misma cantidad de adenina (A) que de timina (T) y la misma de citosina (C) que de guanina (G). En la figura 2.4 se muestran las uniones tipo puente de hidrógeno que se forman entre los pares de bases complementarios T=A y C≡G.

Como las bases A y T se unen por medio de dos puentes de hidrógeno, les llamaremos *débiles*, y como C y G lo hacen por medio de tres puentes de hidrógeno, a éstas las llamaremos *fuertes*.

La estructura del DNA es la misma en todos los seres vivos; además, la organización y regulación de los genes, que son fragmentos específicos de la doble hélice, funcionan igual en todos los organismos. Es relevante insistir en que la estructura general del DNA es exactamente la misma en todos los seres vivos, desde las bacterias hasta los humanos, es decir, el DNA es una doble hélice formada por dos polímeros antiparalelos y complementarios. Cada una de estas dos hélices o polímeros está, a su vez, integrada por miles y en algunas especies, por millones de nucleótidos (monómeros), para formar un polímero. Además, en todo tipo de DNA, a un nucleótido con la base adenina (A) le corresponde siempre, en el nucleótido de la hebra o hélice complementaria, uno con la base timina (T), y a todo nucleótido con la base guanina (G) corresponde un nucleótido con la base citosina (C) en la hebra complementaria (figura 2.5). La diferencia fundamental entre todas las moléculas de DNA que forman los diferentes cromosomas de los seres vivos, es la secuencia de los millones de estos cuatro tipos de nucleótidos con sus bases, A, T, G, C en cada molécula de DNA (genomas).

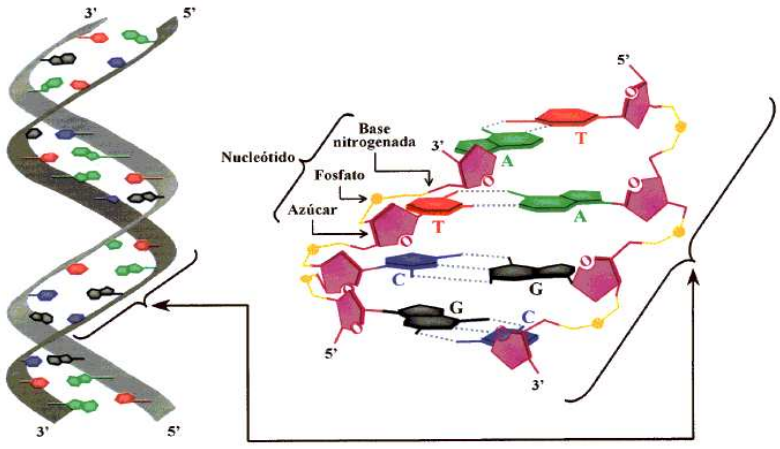


Figura 2.5. Estructura de la molécula del DNA

En la figura 2.5 se muestra cómo el DNA es una doble hélice donde cada

hebra o cadena es un polímero integrado por millones de nucleótidos que son los monómeros del polímero. Cada nucleótido contiene una molécula de azúcar llamada desoxirribosa, una base (nitrogenada) púrica o pirimídica y un grupo fosfato. Debido a que las dos cadenas de DNA son antiparalelas y se unen entre sí a través de enlaces tipo "puentes de hidrógeno", se obtiene una estructura tipo doble hélice, donde las bases de los nucleótidos se encuentran orientadas hacia el interior de la doble hélice y los grupos fosfato así como las azúcares desoxirribosas, hacia su exterior. En la doble hélice, la dirección de los nucleótidos en una hebra (3'→ 5') es opuesta a la dirección en la otra hebra (5'→ 3') según el enlace que existe en la molécula del azúcar.

Otra clasificación de las bases nitrogenadas depende de su naturaleza amínada-cetónica, en la que las amínadas son las bases A y C, y las cetónicas corresponden a las bases T y G. Entonces, las tres formas de clasificarlas son:

$$YR = \begin{cases} \text{Grandes} & A, G \\ \text{Pequeñas} & C, T \end{cases}$$

$$WS = \begin{cases} \text{Fuertes} & C, G \\ \text{Débiles} & A, T \end{cases}$$

$$MK = \begin{cases} \text{Amínadas} & A, C \\ \text{Cetónicas} & T, G \end{cases}$$

Saber cuánto DNA necesita un organismo para funcionar, nos diría mucho para poder analizar a los organismos, sin embargo se puede plantear otra pregunta: ¿cuánto DNA tiene un organismo?, ya que se ha podido secuenciar el DNA de distintas especies, por ejemplo, la bacteria *Escherichia coli* (*E. coli*), es un organismo con una célula y tiene alrededor de 5 x 10^6 bases. Comparado con la simple *E. coli*, el genoma del *humano* tiene aproximadamente 3 x 10^9 letras. Y cada célula humana contiene el mismo DNA.

2.3. DNA - Proteína

El DNA es el medio por el cual los organismos transfieren la información genética a sus descendientes. En organismos con un núcleo (eucariontes), el DNA permanece en el núcleo; mientras que las proteínas se producen en el

citoplasma fuera del núcleo. La molécula intermedia que lleva la información fuera del núcleo es el RNA.

A partir de la información localizada en la doble hélice, una célula sintetiza todas sus proteínas mediante dos mecanismos: la transcripción, que es la síntesis de moléculas de RNA usando regiones específicas o genes del DNA como molde y la traducción, que es la síntesis de proteínas a través de la "lectura" de las moléculas del RNA mensajero en los ribosomas.

2.3.1. Replicación del DNA y síntesis de RNA

Existen tres mecanismos moleculares fundamentales dentro de la célula: a) la replicación de su material genético y su transferencia a las siguientes generaciones; b) la síntesis de proteínas a partir de la información genética que reside en el DNA y c) la expresión de los genes en los cromosomas (Zweiger, 2002).

Debido a la estructura de doble hélice del DNA, es capaz de dar lugar, mediante el fenómeno llamado replicación, a dos dobles hélices idénticas a partir de la doble hélice original. En este fenómeno, cada una de las cadenas de la doble hélice original sirve de molde para la síntesis de una nueva cadena complementaria, generándose así dos dobles hélices iguales, una de las cuales se transfiere a la siguiente generación y la otra permanece en el organismo original (figura 2.6).

La replicación del DNA es el fenómeno que permite el copiado de una doble hélice de DNA, para generar dos dobles hélices idénticas a la original. La información genética contenida en las moléculas de DNA se mantiene mediante la replicación. Durante este fenómeno, las dos cadenas se separan y cada una sirve de molde para sintetizar una nueva cadena complementaria. En la figura 2.6 se muestra que este proceso tiene dos características intrínsecas: I) es "semiconservado", ya que cada una de las dos cadenas de la molécula original pasa intacta a cada una de las moléculas hijas, mientras que la otra cadena se sintetiza de nuevo, y II) siempre va en la dirección 5'→ 3'.

Es posible describir los mecanismos celulares que están involucrados en la decodificación de la información genética por la célula para dar lugar a la

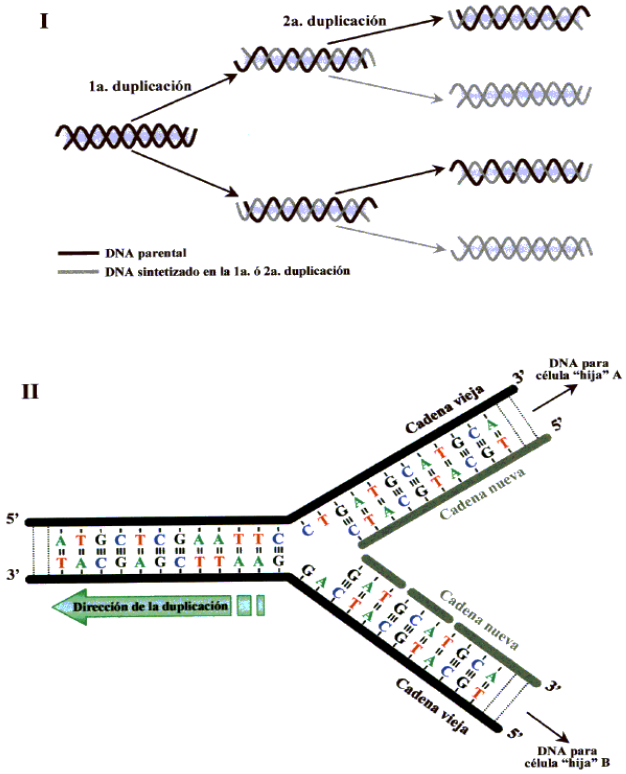


Figura 2.6. Replicación del DNA

síntesis de proteínas, mediante los mecanismos de transcripción del DNA en RNA mensajero (RNAm) y traducción de este RNA en proteína.

El primer paso en la síntesis de proteínas, a partir de la información presente en los genes a nivel del DNA, es la síntesis o formación de una molécula de RNA, específica para cada gen, usando para ello como molde un segmento (un gen) de una de las dos cadenas del DNA (figura 2.7). El RNA es una molécula químicamente muy parecida a una de las hebras del DNA ya que también está formado por cadenas lineales de nucleótidos, por lo que la información genética contenida en el DNA, es decir, el orden de desoxirribonucleótidos de una de las hélices del DNA, de uno o varios genes, se transfiere uno a uno, a una secuencia de ribonucleótidos complementaria en el proceso de la síntesis del RNA. Este proceso de "transcripción" o síntesis del RNA es un proceso enzimático mediado por la enzima RNA polimerasa

que siempre ocurre, al igual que la replicación del DNA, en la dirección 5'→ 3', y normalmente sólo una de las dos cadenas del DNA es transcrita en una molécula de RNA (figura 2.7).

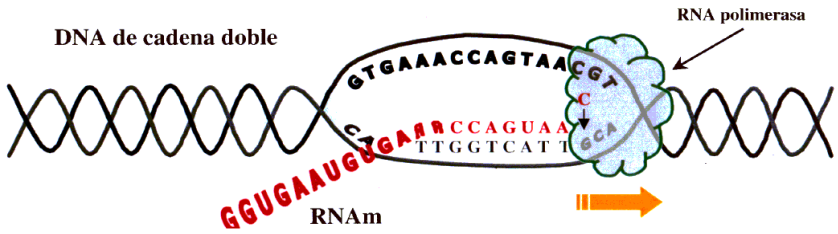


Figura 2.7. Transcripción, síntesis del RNA

Las moléculas de RNA, son transcritas o producidas por la enzima RNA polimerasa a partir de regiones específicas de material genético (que contienen uno o varios genes), utilizando como molde una de las dos cadenas del DNA (ver figura 2.6). Las moléculas de RNA son polímeros lineales de centenas de cuatro tipos diferentes de nucleótidos: A, G, C y U, en donde la diferencia primaria con el DNA es que el uracilo (U) es utilizado en lugar de la timina (T), durante su síntesis. Todas las moléculas de RNA, que son intermediarias en la síntesis de las proteínas, son exportadas del núcleo en el caso de las células eucariontes y luego, en el citoplasma celular, son leídas y su información es utilizada para sintetizar proteínas (ver figura 2.9). Cada molécula de RNA mensajero es recipiente de la información para sintetizar una o varias proteínas específicas.

En el caso de los procariontes que no tienen membrana nuclear, las moléculas de RNA que se transcriben de los genes, son inmediatamente traducidas a nivel de los ribosomas para sintetizar las proteínas (figuras 2.7 y 2.9). Sin embargo, en el caso de los eucariontes que sí tienen membrana nuclear, los RNA transcritos a partir de los genes, deben ser transportados del núcleo al citoplasma, a través de esta membrana. Además, a diferencia de las bacterias, los genes de los eucariontes contienen las estructuras llamadas *intrones*. Estas estructuras, que son también DNA, están interrumpiendo las regiones del gen que codifican para la proteína y que son llamadas *exones*. Como resultado de esta situación, al sintetizarse RNA a partir de un gen durante el proceso de transcripción, la molécula de RNA resultante incluye tanto las regiones de los intrones como la de los exones. Ahora, las moléculas de RNA

que llevan intrones y exones resultantes del proceso de transcripción que se lleva a cabo en los núcleos de las células de los eucariontes, deben ser "procesadas" para dar lugar a las moléculas maduras de los RNA mensajeros, más pequeñas, que son exportadas del núcleo de la célula al citoplasma, donde son traducidas en proteínas (ver figura 2.9).

2.3.2. Código genético

Las proteínas son el otro tipo de moléculas que existen en los organismos y tienen la información que les permite realizar la mayor parte de las funciones y los trabajos celulares; las proteínas también son polímeros biológicos que están constituidos por decenas o centenas de veinte tipos diferentes de monómeros llamados *aminoácidos* (Alberts et al., 2002). No puede haber correspondencia de un aminoácido por cada uno de los nucleótidos que integran los genes y la consecuencia es que cada uno de los aminoácidos que integran una proteína debe estar "codificado" por un grupo de nucleótidos. Cada aminoácido está codificado por un grupo de tres nucleótidos al que se le denomina *triplete* o *codón*, y más de un triplete codifica para un mismo aminoácido. Algunos tripletes codifican para señales de terminación o iniciación para la síntesis proteica.

El código genético es universal, es decir, es utilizado de la misma manera en casi todos los seres vivos (figura 2.8). Este código permite a la célula de cualquier organismo traducir en proteínas la información genética almacenada en los genes que se localizan en el DNA, mediante la lectura en bloques de tres nucleótidos (tripletes o codones) de la información genética presente en el RNA mensajero. Las proteínas son polímeros de decenas o centenas de aminoácidos, en las cuales cada aminoácido es un monómero (figura 2.9). Son veinte diferentes aminoácidos con los que cuenta la célula para integrar las proteínas de los seres vivos. El orden de los aminoácidos en la proteína es importante ya que de esto depende su significado o función biológica.

Cada uno de los veinte diferentes aminoácidos está codificado por un triplete o codón, de tres nucleótidos, a nivel del RNA mensajero. El RNA mensajero es, entonces, una molécula formada por una secuencia de nucleótidos. Esta información es traducida en proteínas al ser leídos estos nucleótidos, de tres en tres, por los ribosomas como se muestran en la figura 2.9.

Aminoácidos:

- Alanina	Ala	A	- Leucina	Leu	L
- Arginina	Arg	R	- Lisina	Lys	K
- Asparagina	Asn	N	- Metionina	Met	M
- Ac. aspártico	Asp	C	- Prolina	Pro	P
- Cisteína	Cys	D	- Serina	Ser	S
- Fenilalanina	Phe	F	- Tirosina	Tyr	Y
- Glicina	Gly	G	- Treonina	Thr	T
- Ac. Glutámico	Gln	Q	- Triptófano	Trp	W
- Glutamina	Glu	E	- Valina	Val	V
- Histidina	His	H	- Terminación de		
- Isoleucina	Ile	I	la traducción	fin	

Nucleótidos:

- Guanina	G
- Adenina	A
- Timina	T
- Citosina	C

NUCLEÓTIDO EN SEGUNDA POSICIÓN

NUCLEÓTIDO EN PRIMERA POSICIÓN	G	A	T	C	NUCLEÓTIDO EN TERCERA POSICIÓN
G	GGG, GGA, GGT, GGC: Gly	GAG, GAA: Glu; GAT, GAC: Asp	GTG, GTA, GTT, GTC: Val	GCG, GCA, GCT, GCC: Ala	G, A, T, C
A	AGG, AGA: Arg; AGT, AGC: Ser	AAG, AAA: Lys; AAT, AAC: Asn	ATG: Met; ATA, ATT, ATC: Ile	ACG, ACA, ACT, ACC: Thr	G, A, T, C
T	TGG: Trp; TGA: fin; TGT, TGC: Cys	TAG, TAA: fin; TAT, TAC: Tyr	TTG, TTA: Leu; TTT, TTC: Phe	TCG, TCA, TCT, TCC: Ser	G, A, T, C
C	CGG, CGA, CGT, CGC: Arg	CAG, CAA: Gln; CAT, CAC: His	CTG, CTA, CTT, CTC: Leu	CCG, CCA, CCT, CCC: Pro	G, A, T, C

Figura 2.8. Código genético.

En un código en el que se tiene un alfabeto de cuatro letras genéticas (A, C, G y T) organizado en tripletes, ya que cada aminoácido consta de tres letras, puede haber 64 diferentes tripletes, debido a que en cada una de las tres posiciones de los tripletes se tiene la posibilidad de poner cada uno de los cuatro nucleótidos; entonces, en la primera posición hay 4 posibilidades, en la segunda otras cuatro y en la tercera otras cuatro y esto da: $4*4*4=4^3=64$.

De esta manera, el código genético, es una tabla de asignación en la que a cada triplete le corresponde un aminoácido. Ahora, al contar solamente con veinte aminoácidos pero teniendo un código con 64 posibilidades, existen sinónimos en los aminoácidos y de esta manera un aminoácido puede ser representado por más de un triplete en la tabla.

Existen aminoácidos que están codificados hasta por seis diferentes tripletes, como leucina (leu), y hay aminoácidos como triptofano (trp), que sólo está codificado por un triplete (en este caso TGG). Existe un codón ATG, que codifica para metionina y que es el codón o triplete con el que se inicia la síntesis de la mayor parte de las proteínas; existen también tres codones TGA, TAA y TAG, que son tripletes que al leerse en los ribosomas son responsables de que finalice el proceso de traducción, esto es, se termina en este punto la síntesis de una molécula de proteínas y ésta se libera de los ribosomas para ser utilizada por la célula de acuerdo con su función biológica (figuras 2.9 y 2.10).

2.3.3. Síntesis de proteínas

La información genética contenida en cada molécula de RNAm, es traducida en moléculas de proteínas a través de un proceso enzimático que se realiza en los organelos celulares llamados ribosomas. En este mecanismo llamado "traducción", participan principalmente tres tipos distintos de RNA: el RNA ribosomal (RNAr), que junto con varias proteínas forman los ribosomas; el RNAm, que transporta la información genética contenida en segmentos específicos (genes) del DNA y finalmente, el RNA llamado de transferencia (RNAt), que sirve como adaptador específico para cada aminoácido, durante el ordenamiento lineal de éstos en la síntesis de proteínas (figura 2.9). La síntesis de proteínas, es la traducción de la secuencia de nucleótidos presentes en el RNAm, se lleva cabo también en dirección 5'→ 3', mediante la polimerización de aminoácidos en proteínas, a nivel de los ribosomas en donde la secuencia del RNAm, se lee de tres en tres nucleótidos de acuerdo con el código genético, incorporando en cada paso de lectura un aminoácido de la proteína, como en la figura 2.9. En la célula, el RNA mensajero lleva la información y los ribosomas leen y traducen la información en proteínas.

El RNA mensajero (RNAm) que se muestra en la figura 2.9, es el intermediario en la síntesis de proteínas. El RNA mensajero al ser copia del DNA, lleva la información genética de los genes hacia los ribosomas, donde ésta es traducida en proteína. Las proteínas son sintetizadas cuando los ribosomas se mueven leyendo, sobre las moléculas de RNAm, donde el extremo $5'$ del RNAm es leído o traducido primero. Una sóla molécula de RNA mensajero normalmente es utilizada por la célula para sintetizar varias moléculas de la

misma proteína.

Durante el proceso de lectura del RNA mensajero por los ribosomas, los codones o tripletes del RNAm se asocian con los anticodones complementarios de los RNA de transferencia, que a su vez se encuentran "cargados" con los aminoácidos respectivos de acuerdo con el código genético. Inmediatamente ocurre un proceso de transferencia del aminoácido que llega y así se incorpora a la cadena de proteína naciente compuesta por varios aminoácidos previamente unidos entre sí.

2.4. Mutación

La información genética contenida en el DNA es susceptible de sufrir cambios, los cuales se denominan *mutaciones*. El gen se describe como un segmento de la secuencia del DNA, consistente en una serie de bases que codifican la producción de una proteína particular. Así, las mutaciones son consecuencia de cambios de las bases en las moléculas de DNA. El DNA hace incontables réplicas de él mismo sin equivocarse. Repite el orden exacto de las bases que representan el código genético; sin embargo, puede ocurrir algo mal en alguna de estas duplicaciones y se comete un error en la copia de una o más bases. Una base puede ser modificada, omitida o añadida. En cualquier caso el mensaje genético se altera y puede manifestarse una mutación.

Todos los organismos experimentan mutaciones como resultado de errores cometidos al replicar el DNA. Son varios los tipos de cambios que puede sufrir el DNA y éstos pueden alterar sólo un nucleótido o varios. Alteraciones en la secuencia de los nucleótidos de un gen pueden ocasionar un cambio en la fase de lectura del gen, a nivel del RNA mensajero, y por ello generarse una proteína con su secuencia de aminoácidos diferente, la cual ya no sea capaz de realizar su función original. Sin embargo, puede haber cambios en la secuencia del DNA que no afecten el funcionamiento de la proteína codificada por el gen particular (figura 2.11). Estos cambios son responsables de la aparición de los llamados "polimorfismos genéticos" en los genes de una especie como la humana.

La figura 2.10 muestra los diferentes tipos de mutaciones que pueden ocu-

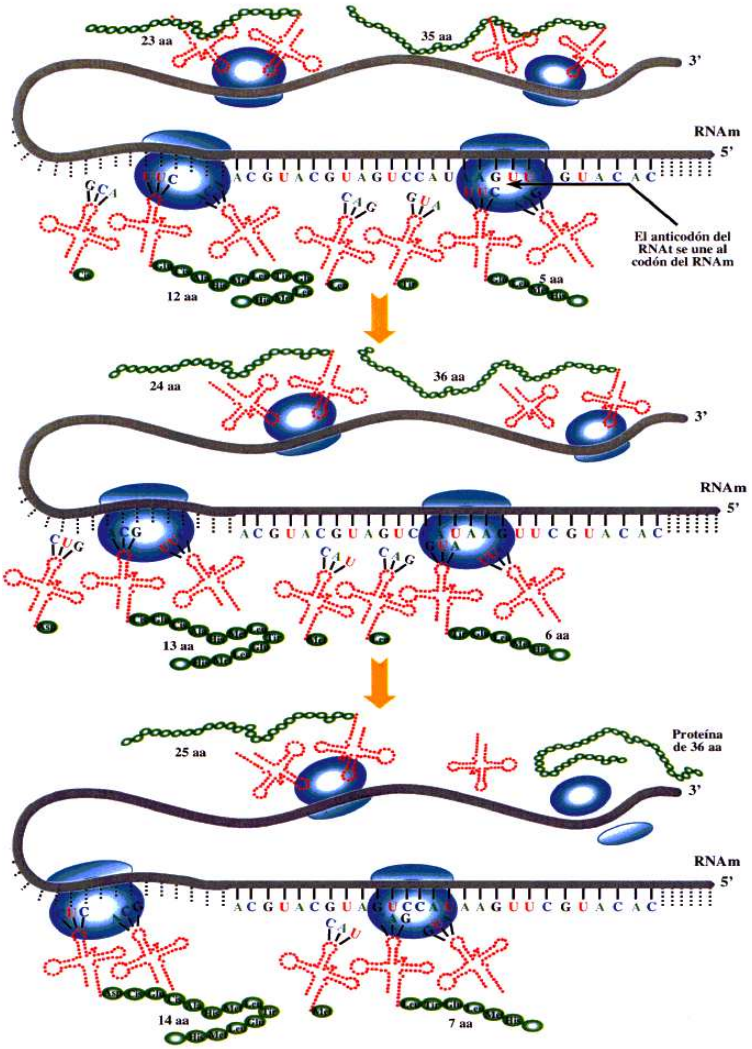


Figura 2.9. Síntesis de proteínas

rrir para cambiar la secuencia original de nucleótidos en cualquier molécula de DNA: a) sustitución, b) adición y c) eliminación. Se muestra el efecto de la sustitución, adición y eliminación de un par de nucleótidos; sin embargo, las adiciones o eliminaciones pueden involucrar uno o muchos pares de nucleótidos.

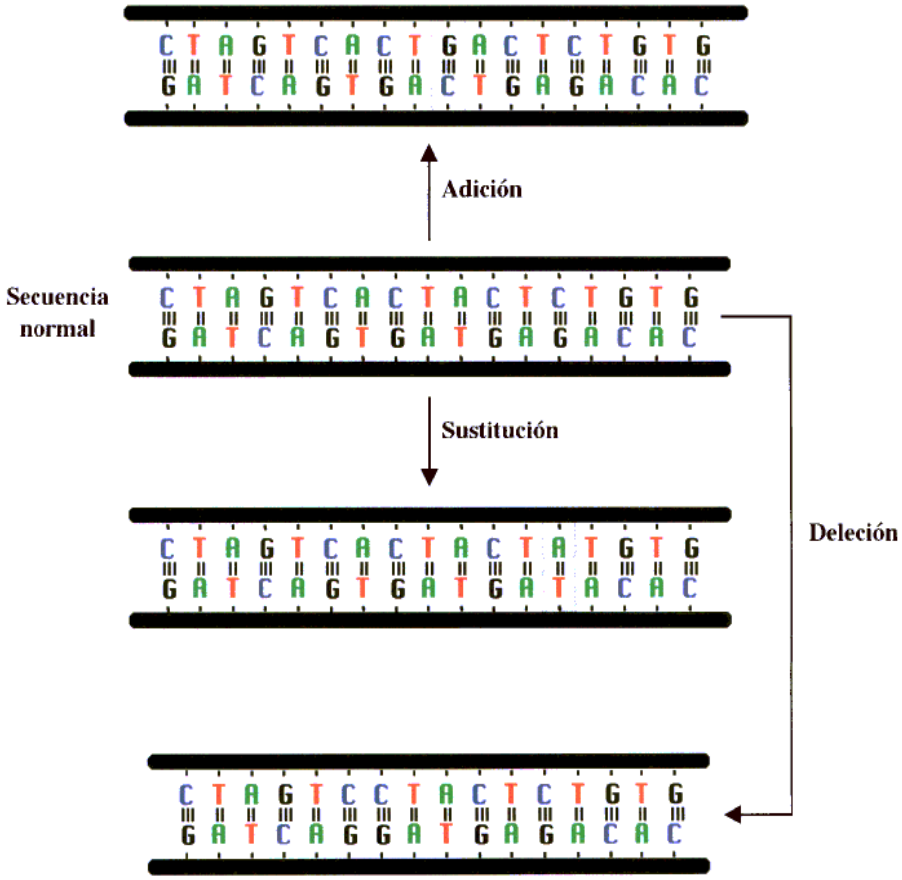


Figura 2.10. Tipos de mutación

Al cambiar la secuencia de nucleótidos de un gen, puede tener resultados diversos sobre la proteína para la que codifica. Los cambios en los genes, debido a las mutaciones, son elemento fundamental en el proceso de la evolución de los seres vivos, ya que permiten que haya variabilidad genética y, por lo tanto, biológica.

La secuencia de los nucleótidos en el gen es responsable de la secuencia de los nucleótidos del RNA mensajero. La estructura final de la proteína depende del orden, a nivel primario, es decir de la secuencia de los aminoácidos que la integran (ver figura 2.12). Si esta secuencia se altera, la función de la proteína puede también alterarse. Las mutaciones o cambios que ocurren en un gen pueden dar lugar a su vez a cambios en los aminoácidos en la proteína que se obtiene a partir de dicho gen.

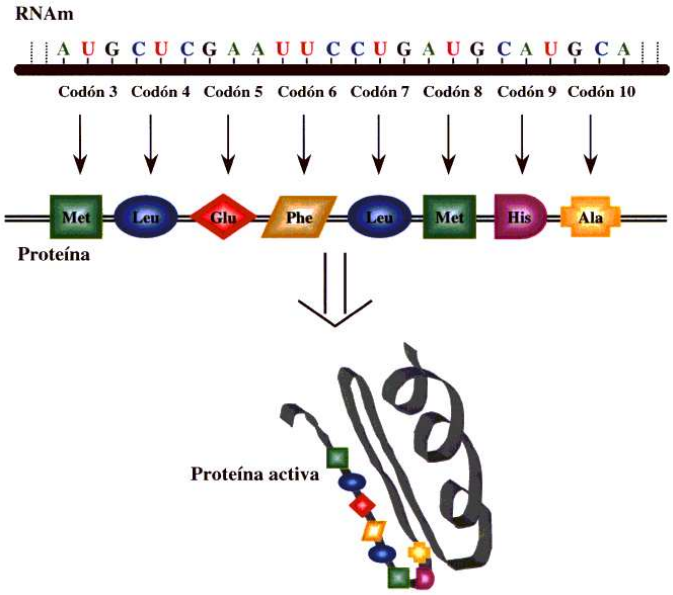


Figura 2.11. RNA mensajero y proteínas

2.5. Proteínas; estructura y función biológica

Las proteínas son moléculas informacionales, pero a diferencia del DNA, que es la molécula en donde reside la información genética para sintetizar las proteínas, éstas son las herramientas que tienen las células para llevar a cabo la mayor parte de sus funciones; en otras palabras, en las proteínas reside la información funcional de la célula.

Como se mencionó, las proteínas son polímeros que están constituidos por decenas o centenas de veinte tipos de monómeros diferentes, los aminoácidos. Cada proteína tiene una secuencia específica de aminoácidos dada por la secuencia de nucleótidos del gen que la codifica y esta secuencia es la que se conoce como la estructura primaria de la proteína. Gracias a esta secuencia primaria, la proteína puede adquirir una estructura secundaria que puede ser fundamentalmente de dos tipos: (a) hélice o (b) plegada. Las estructuras secundarias, a su vez, permiten el doblamiento de las proteínas en estructuras terciarias y finalmente, las estructuras terciarias permiten la asociación de varias moléculas de proteínas en lo que se conoce como estructura cuaternaria (figura 2.12).

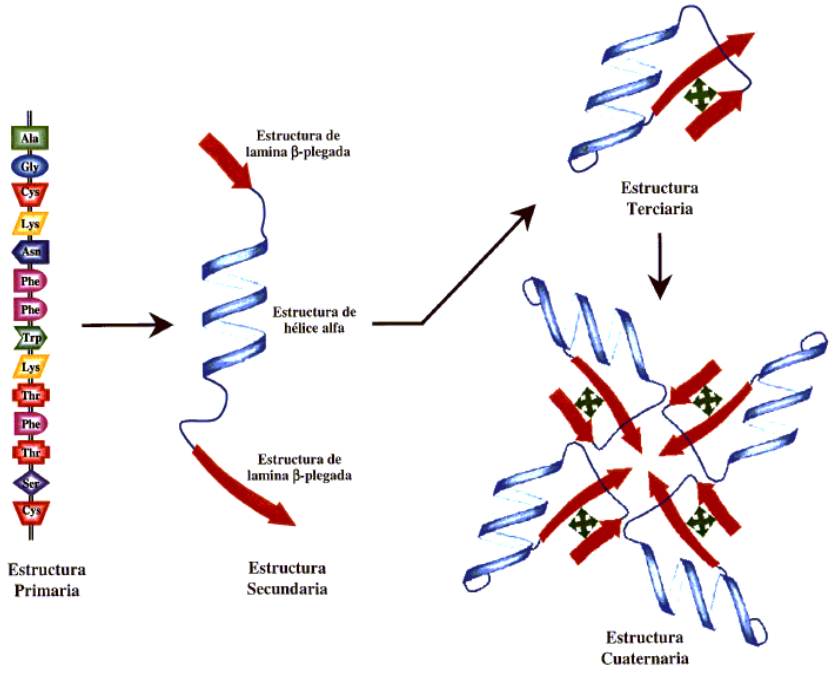


Figura 2.12. Proteínas; estructura y función

La función de cada proteína es específica y está determinada por la estructura tridimensional de la propia proteína. La estructura tridimensional (terciaria y cuaternaria) de cada proteína, está determinada por la estructura primaria, que es en realidad la secuencia de aminoácidos de la proteína.

En este capítulo se ha hecho una descripción de la molécula del DNA y de algunos procesos moleculares que ocurren dentro de la misma. Como se mencionó, se considera al DNA como una secuencia sobre un alfabeto de 4 letras también definido como *genoma* y que será el elemento base para el análisis por medio de algoritmos de compresión de datos propuesto en este trabajo. Para poder dar una explicación de los métodos de compresión, será necesario estudiar ciertas propiedades de la teoría de la información que veremos a continuación.

Capítulo 3

Fundamentos teóricos de la complejidad algorítmica

Para describir y entender qué son las técnicas o algoritmos de compresión de datos, como herramienta para el análisis de secuencias de DNA, en este capítulo se hace una descripción general de la teoría de la información y la complejidad algorítmica (Abramson, 1986), (Li y Vitányi, 2002).

3.1. Teoría de la información

La *teoría de la información* es la disciplina que se encarga del estudio y cuantificación de los procesos que se realizan sobre la *información*. La cantidad de información que nos proporciona cierto dato es menor cuanto más esperamos ese dato. La idea de la *probabilidad de ocurrencia* de cierto evento que nos da información puede modelarse por una variable aleatoria y su conjunto de mensajes y probabilidades asociadas. Al recibir un mensaje de esa variable aleatoria (fuente de información), se obtiene una cantidad de información, que depende sólo de la probabilidad de emisión de ese mensaje. La cantidad de información es una función creciente e inversa de la probabilidad, es decir, un conjunto de datos con menor probabilidad proporciona mayor cantidad de información. En 1948, Claude E. Shannon (Shannon, 1948) propuso una medida para la información que se verá mas adelante en este capítulo.

3.1.1. Codificación de la información

Con objeto de exponer las ideas básicas de la teoría de la información, consideremos algunos ejemplos de transmisión de información. Se considerará un tipo particular pero importante de información, la información binaria.

La información contenida en los mensajes transmitidos mediante sistemas hay-no hay, prendido-apagado, todo-nada o la información almacenada en los elementos biestables de computadoras, constituyen unos cuantos ejemplos de esta clase de información. Esta es una manera de simplificar las consideraciones acerca de la codificación. Como dato se tiene que la representación binaria de la información ha sido la base y es fundamental en el desarrollo de las tecnologías actuales.

El cuadro 3.1 muestra un ejemplo sencillo de representación de información no binaria en función de dígitos binarios 0 y 1.

Dígito decimal	Representación binaria
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Cuadro 3.1. Codificación binaria de los dígitos decimales

La correspondencia entre los dígitos decimales y binarios definida por el cuadro 3.1 constituye un ejemplo de código. Las 10 secuencias binarias se denominan palabras código y los 10 dígitos decimales, símbolos mensaje. Es evidente que mediante el código de el cuadro 3.1 podremos deducir la secuencia de dígitos binarios correspondiente a cualquier secuencia de dígitos

decimales (símbolos mensaje). Recíprocamente, de una secuencia de dígitos binarios perteneciente a este código, podemos obtener una única secuencia de dígitos decimales.

La posibilidad de establecer a partir de una serie de palabras código binarias los correspondientes símbolos mensaje no es siempre una operación inmediata. Por ejemplo, el código definido en la cuadro 3.2 .

Símbolos mensaje	Palabras código
s_1	0
s_2	01
s_3	001
s_4	111

Cuadro 3.2. Código binario I

Dada una secuencia de palabras código de la tabla, no se puede estar en situación de deducir un único conjunto de símbolos mensaje. La secuencia binaria

$$111001 \tag{3.1}$$

puede provenir de

$$s_4 s_3 \tag{3.2}$$

o de

$$s_4 s_1 s_2 \tag{3.3}$$

Esto se puede objetar indicando que la simple inclusión de una coma (o espacio) es suficiente para evitar confusión. Naturalmente, esto es cierto, sin embargo, el empleo de una como (o espacio) no está de acuerdo con la definición de código binario. Si utilizamos una coma para separar las palabras, estamos empleando realmente tres símbolos diferentes, cero, uno y coma.

Símbolos mensaje	Palabras código
s_1	0
s_2	10
s_3	110
s_4	1110

Cuadro 3.3. Código binario II

Resulta sencillo encontrar un código que no presente los inconvenientes del del cuadro 3.2. A una secuencia de palabras código perteneciente al cuadro 3.3 puede asociarse un conjunto único de símbolos mensajes.

3.1.2. Bases matemáticas para la compresión

Como se verá en el siguiente capítulo, donde se hace una descripción más profunda, los esquemas de compresión se pueden dividir en dos clases: *lossy* y *lossless*. Los esquemas de compresión *lossy* implican la pérdida de cierta información y los datos que se han comprimido usando este esquema, no se pueden recuperar exactamente en la mayoría de los casos. Los esquemas *lossless* comprimen los datos sin pérdida de información y los datos originales se pueden recuperar exactamente de los datos comprimidos. Enseguida veremos algunas ideas en la teoría de la información que proporcionan el marco para el desarrollo de los esquemas *lossless* de compresión de datos, que se utilizarán en el o los algoritmos de compresión para comprimir los archivos de distintos genomas.

Shannon fue el precursor de la llamada teoría de la información; definió una cantidad llamada *self-information* (Shannon, 1948) que se trata a continuación.

Definición de información y sus fuentes.

Se define primero una medida de la información y se muestra que posee las propiedades deseables de cualquier otra definición.

Definición 1 *Sea E un suceso que puede presentarse con probabilidad P(E). Cuando E tiene lugar, decimos que hemos recibido*

$$I(E) = \log \frac{1}{P(E)} = -\log P(E). \tag{3.4}$$

unidades de información.

Esta cantidad, es la *self-information* asociada con E.

La elección de la base del logaritmo que interviene en la definición equivale a elegir una determinada unidad, ya que

$$\log_a x = \frac{1}{\log_b a} \log_b x \tag{3.5}$$

y por eso todavía no se ha especificado la base. El uso del logaritmo para obtener una medida de información no es una opción arbitraria y se verá más adelante porqué.

Primero se tiene que ver si el uso de un logaritmo en este contexto tiene sentido desde un punto de vista intuitivo. Como log (1) = 0 y -log(x) crece cuando x disminuye de uno hasta cero; entonces, si la probabilidad de un evento es baja, la cantidad de *self-information* asociada a él es alta; si la probabilidad de un evento es alta, la información asociada a él es baja. Incluso, si ignoramos la definición matemática de información y simplemente utilizamos la definición del lenguaje diario, ésta tiene un cierto sentido intuitivo. Por ejemplo, el ladrar de un perro durante un asalto es un evento con probabilidad alta y entonces, no contiene mucha información. Sin embargo, si el perro no ladra durante un asalto, es un evento con probabilidad baja y contiene bastante información. Aunque esta equivalencia de las definiciones matemática y semántica de la información es cierta muchas veces, en otras no. Por ejemplo, una cadena totalmente aleatoria de letras contendrá más información (en el sentido matemático) que un tratado bien pensado en teoría de información.

Si introducimos el logaritmo de base 2, la unidad correspondiente se denomina *bit*(binary digit).

$$I(E) = \log_2 \frac{1}{P(E)} \qquad \text{bits} \tag{3.6}$$

Si P(E) = 1/2, será I(E) = 1 bit. Es decir, *un bit es la cantidad de información obtenida al especificar una de dos posibles alternativas igualmente probables*. Esta situación se presenta al lanzar una moneda al aire o al examinar la salida de un sistema de comunicación binario.

Otra característica de esta definición matemática de información que tiene sentido intuitivo es que la información obtenida de la ocurrencia de dos eventos independientes es la suma de la información obtenida de la ocurrencia de los eventos individuales. Suponga que A y B son dos eventos independientes. La *self-information* asociada con la ocurrencia del evento A y del evento B esta dada por:

$$I(AB) = \log_b \frac{1}{P(AB)}$$

Como A y B son independientes,

$$P(AB) = P(A)P(B)$$

y

$$\begin{aligned} I(AB) &= \log_b \frac{1}{P(A)P(B)} \\ &= \log_b \frac{1}{P(A)} + \log_b \frac{1}{P(B)} \\ &= I(A) + I(B). \end{aligned}$$

La unidad de información depende de la base del logaritmo; así pues, si usamos log base 2, la unidad es *bits*, si usamos *log* base e, la unidad es *nats*, y si usamos *log* base 10, la unidad es *hartleys* (Sayood, 2000).

Nótese que para calcular la información en bits, necesitamos tomar el logaritmo base 2 de las probabilidades. Esto es debido a que

$$\log_b x = a$$

significa que

$$b^a = x.$$

Por lo tanto, si deseamos tomar el logaritmo base 2 de x

$$\log_2 x = a \Rightarrow 2^a = x,$$

queremos encontrar el valor de a. Podemos tomar el logaritmo natural (log base e) o el logaritmo base 10 en ambos lados, entonces

$$\ln(2^a) = \ln x \Rightarrow a \ln 2 = \ln x$$

y

$$a = \frac{\ln x}{\ln 2}$$

Ejemplo

Sean H y T los resultados de lanzar una moneda. Si no hay sesgos, entonces

$$P(H) = P(T) = \frac{1}{2}$$

y

$$I(H) = I(T) = 1 \quad bit$$

Si la moneda no es equilibrada, entonces esperaremos que la información asociada a cada evento sea diferente. Supongamos

$$P(H) = \frac{1}{8}, \quad P(T) = \frac{7}{8}.$$

Entonces

$$I(H) = 3 \quad bits, \qquad I(T) = 0{,}193 \quad bits.$$

Por lo menos matemáticamente, la ocurrencia de H transporta mucha más información que la ocurrencia de T.

Si tenemos un conjunto de eventos independientes A_i, los cuales son conjuntos de resultados de un cierto experimento S, tales que

$$\bigcup A_i = \Omega$$

donde Ω es el espacio muestral, entonces el promedio de *self-information* asociada con el experimento al azar está dado por

$$H = \sum P(A_i)I(A_i) = \sum P(A_i) \log_b \frac{1}{P(A_i)} = -\sum P(A_i) \log_b P(A_i)$$

A esta cantidad se le llama la *entropía* asociada con el experimento. Una de las muchas contribuciones de Shannon fue que demostró que si el experimento es una fuente que tiene como salida símbolos A_i de un conjunto A, entonces la entropía es una medida del número promedio de símbolos binarios necesarios para codificar la salida de la fuente. Shannon demostró que lo mejor que un esquema de compresión *lossless* puede hacer, es codificar la salida de una fuente con un número promedio de bits igual a la entropía de la fuente.

En resumen, se tiene que dada una fuente F de información de n mensajes F_i, cada uno con probabilidad p_i, la información al recibir un mensaje está dado por la ecuación:

$$I(F = F_i) = -log(p_i)$$

La medida media de información o *entropía* de F está dada por la esperanza matemática de la información de cada símbolo

$$H(F) = \sum_{i=1}^{n} p_i * I(F = F_i) = -\sum_{i=1}^{n} p_i * \log(p_i)$$

Al utilizar logaritmo en base 2, la medida estará en bits. Esta medida da una cota superior teórica de la cota de compresión que puede obtenerse de la información, no se puede tener ninguna codificación que consiga una longitud en bits media por símbolo emitido menor que la entropía de la fuente sobre la que se realiza la codificación. Los compresores no llegan a esta cota pero quedan muy cerca.

3.2. Complejidad algorítmica

La aproximación más natural para definir la cantidad de información es claramente definida en relación al objeto individual, en lugar de relacionarla

a un conjunto de objetos del cual el objeto individual puede ser seleccionado. Así, se podría definir la cantidad de información en un objeto en términos de el número de bits necesarios para describirlo. Una descripción de un objeto es útil sólo si se puede reconstruir el objeto desde esta descripción.

La definición de entropía está relacionada con el problema de transmisión de un mensaje sin perdida de información, es decir, el problema de codificación eficiente. La noción de información "absoluta" de objetos individuales, es la información que por sí misma describe el objeto completamente.

El problema de codificación óptima para un texto (o puede ser para una imagen o para cualquier tipo de información) es un campo de gran interés para su estudio. Shannon estudió que existe un límite para poder codificar una secuencia dada. Y este límite es la entropía de la secuencia. Existen varias definiciones equivalentes pero probablemente la mejor definición en este contexto es la complejidad de Kolmogorov que se dará más adelante.

Se sabe que hay objetos complicados y hay objetos que son simples. Por ejemplo, un número como 2^{1000} es de alguna manera simple (se expresa con unos cuantos bits); sin embargo, hay números de mil bits difíciles de ver y de hallar una descripción en la que se requiera muchos menos que mil bits. De esta manera, la dificultad para describir números implica abreviar adecuadamente sus descripciones.

Se requiere un método de descripción universal y un mecanismo para producir el objeto a partir de su supuesta descripción.

Por ejemplo, se sabe que ciertos lenguajes de programación favorecen cómputos simbólicos mientras que otros favorecen cómputos aritméticos, aún cuando todos son universales. La noción del contenido de información de objetos individuales puede ser útil solamente si la cantidad de información es un atributo del objeto en sí y es independiente a los medios con los que se describió.

3.2.1. Complejidad de Kolmogorov

A mediados de los años 60, durante la primera etapa de las ciencias de la computación y con la teoría general de las máquinas de Turing (Turing,

1936) bien entendida, los científicos necesitaban medir el cómputo y la información cualitativamente. La complejidad de Kolmogorov fue inventada por Ray J. Solomonoff (1964), Andrey N. Kolmogorov (1965) y Gregory J. Chaitin (1969), independientemente y en este orden cronológico. Ahora, esta teoría es aceptada extensamente como la aproximación estándar que estableció un debate sobre la noción de la aleatoriedad de un objecto individual, como oponer la mejor noción que se tiene de una variable aleatoria intuitivamente con ambos resultados individuales *aleatorio* y *no aleatorio*. La complejidad de Kolmogorov tiene una gran cantidad de aplicaciones en muchas áreas incluyendo las ciencias de la computación, las matemáticas, la física, la biología y las ciencias sociales (Li y Vitányi, 1997).

La cantidad de información en una secuencia finita es el tamaño (número de dígitos binarios o *bits*) del programa más pequeño que, sin datos adicionales y comenzado con una memoria en blanco, calcula la secuencia y después termina. Una definición similar se puede dar para las secuencias infinitas, pero en este caso el programa produce un elemento después de otro elemento siempre sin parar. Entonces una gran secuencia de 1's tales como

$$\underbrace{1111\ldots1}_{n \quad veces}$$

contiene poca información porque un programa de tamaño cercano a $log\ n$ puede generar esta cadena:

for $i := 1$ to n
print 1

Asimismo, el número transcendental π=3.1415..., una secuencia infinita de dígitos decimales aparentemente 'al azar', contiene una cantidad constante ($O(1)$) de información (hay un programa pequeño que produce los dígitos consecutivos de π). Con tal definición podría parecer que la cantidad de información en un objeto depende del lenguaje de programación particularmente usado. Puede ser mostrado que todas las opciones de los lenguajes de programación universales (tales como PASCAL, C++, Java o Lisp) conducen a la cuantificación de la cantidad de información "absoluta" en objetos individuales que sea invariante hasta una constante.

A esta cantidad se le llamará "complejidad de Kolmogorov" del objeto o secuencia. Si un objeto tiene regularidades o patrones en su contenido, entonces es posible formular una descripción compacta de él que haga referencia a estas regularidades y por tanto es "compresible". Formalmente, esto es formulado lo mejor posible en términos de las 'máquinas universales de Turing', la celebrada formulación rigurosa de computabilidad por Alan M. Turing (1936) que dió comienzo con la teoría y práctica del cómputo (Li y Vitányi, 1997).

La aplicación de la complejidad de Kolmogorov toma una variedad de formas; por ejemplo, usando el hecho de que algunas secuencias son extremadamente compresibles, que no son compresibles totalmente o que pueden ser comprimidas, pero hay que hacer un esfuerzo considerable para comprimirlas; así como utilizar la compresibilidad de secuencias como un criterio de selección.

Esta teoría es diferente de la teoría de información de Shannon que se ocupa de la información prevista en un mensaje de un conjunto probabilístico de mensajes posibles. La complejidad de Kolmogorov, por otra parte, mide la información en una secuencia o un mensaje individual. La *deficiencia de aleatoriedad* de una secuencia binaria de longitud n, es el número de bits por los cuales la complejidad no alcanza n − la máxima complejidad − y una cadena es más aleatoria si su complejidad es cercana a su longitud.

3.2.2. Teoría de información algorítmica

La complejidad de Kolmogorov $C(x)$ de una secuencia x es la longitud del programa más corto (para un lenguaje de programación universal) sin ninguna entrada adicional que genere x solamente como su salida y que entonces se detenga. Es la información necesaria para la recuperación de un objeto x de la nada. Una secuencia x es incompresible si $C(x)$ es por lo menos la longitud $|x|$ (número de bits) de x: la manera más corta para describir x es darlo literalmente. Similarmente, una secuencia x es "casi" incompresible si $C(x)$ es "tan grande como" $|x|$.

El estándar apropiado para "tan grande como" mencionado arriba, puede depender del contexto, una opción típica es $C(x) \geq |x|\text{-}O(|logx|)$. Similar-

mente, la complejidad *condicional* de Kolmogorov de x con respecto a y, denotada por $C(x|y)$, es la longitud del programa binario más corto que, con la información adicional y, genera x. Calcula la información necesaria para recuperar x dando solamente y. Y una secuencia x es incompresible en relación con y si $C(x|y)$ es grande en el sentido apropiado.

Por lo tanto, la complejidad es "información absoluta" en un objeto.

Si $C(x|y)$ es mucho menor que $C(x)$, entonces esto se puede interpretar como una indicación que y contiene una gran cantidad de información acerca de x.

Intuitivamente, se piensa en las secuencias que tienen pocos patrones como si fueran aleatorias y se utiliza el término "secuencia aleatoria" como sinónimo de "secuencia incompresible." Esto no es correcto pero da una idea de la respuesta a la pregunta fundamental sobre la existencia y la caracterización de los objetos individuales aleatorios (secuencias). Después de medio siglo de aproximaciones sin éxito y debates científicos decisivos, en 1965 el matemático sueco Per Martin-Löf resolvió el tema y dio una formalización rigurosa de la noción intuitiva de una secuencia aleatoria como una secuencia que pasa todas las pruebas efectivas de la aleatoriedad. Él dio una formulación similar para las secuencias aleatorias infinitas. El conjunto de secuencias aleatorias infinitas tiene medida 1 en el conjunto de todas las secuencias. La formulación de Martin-Löf utiliza teoría constructiva de la medida y tiene formulaciones equivalentes en términos de ser incompresible. Cada secuencia aleatoria de Martin-Löf es *universalmente* aleatoria en el sentido de que individualmente posee todas las características de aleatoriedad efectivamente probadas (se puede comparar esto con la noción de computabilidad intuitiva que es tomada exactamente de la noción de "cómputo de las máquinas de Turing", y cada cómputo de la máquina de Turing puede ser realizado por una máquina universal de Turing).

Se puede comparar la complejidad de Kolmogorov con la noción estadística de entropía de Shannon, la longitud mínima esperada, de palabra código, de mensajes de una fuente aleatoria usando el menor código posible. Es de notarse que, muchas leyes que se tienen para la entropía de Shannon, es decir, sobre el promedio, se sigan teniendo en cuenta para la complejidad de Kolmogorov de secuencias individuales aunque solamente con un término

aditivo logarítmico. Se denota por $C(x|y)$ la información en x dada por y (la longitud del programa más pequeño que calcula x a partir de y), y se denota $C(x,y)$ la longitud del programa más pequeño que calcula el par (x,y). Es ésta la versión (profunda y de mayor alcance) de la complejidad de Kolmogorov de la ley de "simetría de información" clásica. Considerando la siguietne expresión:

$$C(x, y) = C(x) + C(y|x) = C(y) + C(x|y)$$

Se interpreta $C(x) - C(x|y)$ como la información que y tiene acerca de x. De la misma manera, si la cantidad de información que y tiene acerca de x casi es igual a la cantidad de información que x tiene acerca de y, entonces la información es simétrica y se le llama información mutua.

La complejidad de Kolmogorov es una medida aceptable de aleatoriedad. Sin embargo, no es computable, lo cual impide obviamente algunas formas de uso práctico. Sin embargo, la no computabilidad no es realmente un obstáculo para la gran diversidad de aplicaciones de la complejidad de Kolmogorov, así como la no computabilidad de casi todos los números reales no impide su uso práctico.

Una vez que se han revisado algunos conceptos de teoría de la información y de la complejidad algorítmica como base de los métodos de compresión de datos, en el siguiente capítulo se describen las técnicas de compresión de datos y sus características.

Capítulo 4

Compresión de datos

En los últimos años hemos sido testigos de una transformación en la manera de como nos comunicamos y este proceso aún está en curso. Esta transformación, por ejemplo incluye al internet, el cual se ha convertido en un medio de información y comunicación indispensable, el rápido desarrollo de las comunicaciones móviles y la importancia cada vez mayor de la comunicación en vídeo. La compresión de datos es una de las tecnologías disponibles para que cada uno de estos aspectos de la comunicación se lleve a cabo. No sería práctico poner imágenes, menos todavía audio y vídeo juntos en un sitio web si no fuera por los algoritmos de compresión de datos.

Se ha dado un aumento tanto de la capacidad de almacenamiento de datos como en la velocidad de procesamiento en las computadoras. Junto con esto, la tendencia es la disminución de costos en memoria principal y secundaria así como también un aumento de velocidad de estos dispositivos de almacenamiento. Estos acontecimientos ponen en tela de juicio la necesidad de compresión de datos. Sin embargo, el auge que últimamente han tenido las redes de computadoras, demanda más prestaciones que están por encima de las posibilidades reales. El principal problema al que se enfrentan las redes de comunicación es la velocidad de transferencia de datos. El cambio a mayores velocidades no es tarea fácil, básicamente por razones como la no factibilidad de realizar cambios de infraestructura en las grandes compañías de redes WAN (cableado, tecnologías, entre otros) así como la falta de tecnología que acepte velocidades muy elevadas de transmisión.

En este entorno, para incrementar las prestaciones de velocidad, se debe

recurrir a técnicas que les permitan superar de alguna manera las deficiencias físicas de la red. La técnica más importante en este sentido es la compresión de datos. La compresión de datos es beneficiosa en el sentido de que el proceso de compresión-transmisión-descompresión es más rápido que el proceso de transmisión sin compresión. La compresión de datos no sólo se usa en la transmisión de datos sino también en el almacenamiento masivo. La necesidad de almacenamiento también crece por encima de las posibilidades del crecimiento de los discos duros o memoria. Por ejemplo, aplicaciones como el proyecto del genoma humano, experimentos físicos o los servidores de vídeo en demanda, requieren de varios gigabytes de almacenamiento. La motivación para aplicar compresión a los datos es la reducción de costos tanto en almacenamiento (se requiere menos espacio) como en la transmisión de los datos (se transmiten más rápidamente empleando el mismo ancho de banda). El precio que debe pagarse es cierto tiempo de cómputo para comprimir y descomprimir los datos (Witten et al., 1999); tradicionalmente, se ha establecido un compromiso entre los beneficios de compresión y costo computacional requerido (Stauffer y Hirschberg, 1993).

Para comprimir los datos, los métodos de compresión examinan los datos, buscan "redundancia" en ellos e intentan eliminarla. Una parte central en la compresión es la redundancia en los datos. Sólo los datos con redundancia pueden comprimirse aplicando un método o algoritmo de compresión que la elimine o remueva de alguna forma. La redundancia depende del tipo de datos (por ejemplo texto, imágenes, sonido), por ello, no existe un método de compresión universal que pueda ser óptimo para todos los tipos de datos.

Pero ¿qué es compresión de datos y por qué la necesitamos? En ciencias de la computación y teoría de información, la compresión de datos o la codificación de una fuente de datos, es el proceso de representar la información usando una cantidad menor de bits (u otras unidades de información) o bien, en una forma compacta. Por ejemplo, a partir de este momento, este trabajo se podría codificar con menos bits si uno acepta la convención de que la palabra "compresión" se codifique como "comp".

Al comprimir se crean representaciones compactas identificando y usando las estructuras que existen en los datos. Los datos pueden ser caracteres en un archivo de texto, escrito en cualquier idioma, números que son muestras de una imagen o secuencias de números que son generados por otros procesos.

Una razón por la que necesitamos la compresión de datos es que mucha de la información que se genera y utiliza está digitalizada (en la forma de números representados por bytes de datos).

Comprimir significa reducir el tamaño de algo. El objetivo principal de cualquier esquema de compresión es describir la misma información con un conjunto de datos de menor magnitud. Pero, ¿por qué es importante reducir el tamaño de los datos necesarios para representar una información determinada?

Primero para poder colocarlos en un espacio menor. Actualmente existen dispositivos con una gran capacidad de almacenamiento, permitiendo guardar en ellos datos de gran tamaño. Sin embargo, la complejidad y volumen de los datos aumenta en forma casi paralela al aumento de las capacidades de estos medios de almacenamiento. Además, si se desea transportar datos en otro medio (disquete, memorias flash, CD-ROM, DVD), el espacio está más limitado y puede no ser suficiente para contener los datos que se requiere transportar. Por ello hay que comprimirlos.

La segunda razón aparece fundamentalmente con las redes de comunicaciones como internet. Esta segunda razón es el tiempo necesario para acceder o descargar por la red archivos de gran tamaño. Por tanto, si esos datos están comprimidos, se tardará menos en enviarlos o recibirlos (por correo electrónico, web, ftp o cualquier otro protocolo de transferencia de datos).

Como en cualquier forma de comunicación, la comunicación de datos comprimida trabaja solamente cuando el remitente y el receptor de la información entienden el esquema de codificación.

La compresión es útil porque ayuda a reducir el consumo de recursos costosos, tales como espacio de disco para almacenar o ancho de banda en el caso de transmisión. El diseño de los esquemas de la compresión de datos por lo tanto, implica compensaciones entre varios factores, incluyendo el grado de compresión, la cantidad de distorsión introducida (si usa un esquema de compresión con pérdida de información) y los recursos de cómputo necesarios para comprimir y descomprimir los datos.

4.1. Conceptos básicos

En el análisis de la compresión se debe establecer una diferencia entre información y datos, ya que en muchas ocasiones se utilizan como sinónimos y no lo son. Los datos son una forma de representar la información; así, una misma información puede ser representada por distintas cantidades de datos. Por tanto, algunas representaciones de la misma información contienen datos redundantes. A continuación se presentan algunas definiciones de compresión de datos.

* La compresión consiste en tomar una secuencia de símbolos y transformarlos en códigos o claves. Si la compresión es eficiente, las códigos resultantes ocuparán menos espacio que los símbolos originales.

* La compresión de datos se define como el proceso de reducir la cantidad de datos necesarios para representar eficazmente una información. Es decir, la eliminación de datos redundantes.

La redundancia es una características de los datos, que hace que los símbolos de la fuente de datos no tengan un carácter aleatorio, lo que permite bajo ciertas condiciones, determinar una curva no uniforme de distribución de símbolos.

Tipos de redundancia:

* Repetición de caracteres.
* Repetición de patrones.
* Repetición de cadenas.
* Distribución de caracteres.

En el caso de las imágenes, existen tres tipos de redundancia:

* Código redundante.
* Píxeles redundantes.
* Redundancia visual.

En el proceso de compresión existen dos elementos fundamentales, un modelo y un codificador. Un modelo es simplemente una colección de datos

y reglas usados para procesar los datos de entrada; es aquí donde se extrae información sobre redundancias existentes en los datos y se describen en forma de modelo.

Como no hay modelos exactos, para mantener la correspondencia entre el modelo y las datos reales se debe generar una descripción del modelo y una descripción de las diferencias entre los datos reales y el modelo.

El codificador es usado para producir una codificación apropiada basada en los criterios establecidos en el modelo. Aquí se explotan las redundancias para representar la información en forma compacta. Por ejemplo en el caso de que se use un modelo probabilístico, el codificador se encargará de asignar un código a cada mensaje dependiendo de su probabilidad según el modelo.

Los conceptos de modelo y codificación son definiciones diferentes. El término "codificación" es un componente del proceso de compresión de datos mas no es todo el proceso de compresión. Por ejemplo, "codificación Huffman" y "codificación Run-Length" son descritas como técnicas de compresión de datos, cuando de hecho sólo son métodos de codificación usados en conjunción con un modelo de compresión de datos.

Para comprimir los datos se utilizan distintos algoritmos, especialmente diseñados para cada tipo de dato. El resultado de la compresión será un archivo en un formato específico, de menor o igual tamaño que el original, dependiendo de qué tan eficiente sea el algoritmo de compresión. Se puede presentar el caso en el cual el archivo comprimido es de mayor tamaño que el archivo original, esto generalmente se debe a que se aplicó un algoritmo de compresión no adecuado. Es decir, el algoritmo aplicado no explota la redundancia asociada a los datos de entrada; cabe decir que existen varios tipos de redundancia y que los datos pueden poseer uno o varios tipos de redundancia asociados, dependiendo de su formato.

La compresión suele llevar asociada una descompresión, con la que se puede recuperar el formato original de esos datos para poder utilizarlos. Dependiendo de la forma en que se lleve acabo el proceso de compresión/descompresión, la compresión de datos se puede clasificar tomando como base los siguientes factores (Wayner, 2000):

Según el tiempo necesario para comprimir / descomprimir:

* *Simétricos*: en este tipo de compresión el proceso de descompresión consume una cantidad de tiempo igual o muy cercana a la que se emplea en el proceso de compresión. Este tipo de compresión es de uso general en los procesos que implican transmisión de datos en tiempo real. Por ejemplo las vídeo conferencias, voz sobre IP y aplicaciones de telemetría.

* *Asimétricos*: en este tipo de compresión el tiempo de compresión y descompresión difieren en un alto margen; generalmente, el proceso de compresión tarda más que el de descompresión. Estos algoritmos de compresión son utilizados generalmente para comprimir grandes volúmenes de datos, con la característica de que el proceso de compresión generalmente se realiza sólo una vez, mientras que el de descompresión se lleva a cabo muchas veces. Por ejemplo, en la compresión de vídeo, el proceso de compresión es muy lento y si no se cuenta con una máquina de buen rendimiento, el tiempo que tarda puede ser excesivo, pero con la ventaja de que sólo se realiza una vez, después se almacena el vídeo comprimido en un medio y desde allí se puede reproducir tantas veces como sea necesario porque se tiene un proceso de descompresión de relativa baja complejidad en comparación con el proceso de compresión.

Según el modelo utilizado los algoritmos de compresión se clasifican en:

* *No adaptativo*: En este tipo de compresión el modelo se mantiene constante y ha sido predefinido previo al proceso de compresión. Estos compresores son rápidos, pues no es necesario un reconocimiento previo de los datos a comprimir. La implementación del algoritmo de compresión es muy sencillo, pero la razón de compresión generalmente es muy baja.

* *Semi-adaptativo*: los algoritmos de compresión semi-adaptativos definen el modelo con base en un examen previo de los datos. Es decir, se establece un modelo basado en una visión panorámica del total de datos a comprimir y después se aplica dicho modelo de forma invariable en el proceso de compresión. Para el funcionamiento de estas técnicas se requiere que los datos a comprimir estén almacenados o que su transmisión se lleve acabo dos veces (en la primera transmisión se establece el modelo y en la siguiente se aplica dicho modelo para realizar la codificación). Esto es un inconveniente ya que los datos no pueden ser tratados conforme fluyen.

* *Adaptativo*: este tipo de compresión se caracteriza por la actualización dinámica del modelo. Es decir, conforme se realiza la compresión, el modelo aplicado varía dependiendo del comportamiento de los datos de entrada.

Existen dos ventajas principales de estos compresores sobre los anteriores. En primer lugar, se realiza un único recorrido de los símbolos codificados. Aunque esto pueda parecer una simple mejora, en determinadas situaciones es esencial ya que la secuencia no tiene que ser almacenada para ser comprimida y, así, dicho proceso puede realizarse simultáneamente a la llegada de los símbolos al compresor (la secuencia de símbolos puede ser tratada como una corriente).

En segundo lugar, presentan mejores razones de compresión; esto se debe principalmente, a que la codificación de los símbolos varía durante el transcurso de la compresión. Así, la longitud de los códigos que son asignados a los símbolos, aumenta o disminuye según las variaciones de los patrones del modelo, los cuales están directamente ligados a los datos de entrada y su flujo en el tiempo.

También tienen desventajas como las siguientes:

En primer lugar son más lentos que los anteriores. En segundo lugar su implementación es más compleja, pues deben resolver dos problemas:

a) La actualización del modelo.
b) La inclusión de nuevos símbolos en el modelo.

La compresión de datos ahorra espacio de almacenamiento y reduce el ancho de banda necesario para las transmisión de datos; a cambio, consume tiempo, recurso que generalmente no se está dispuesto a dejar de lado, es por eso que no todos los datos se almacenan o se transmiten comprimidos.

4.2. Técnicas de compresión

Cuando se habla de técnica de compresión o algoritmo de compresión, se hace referencia a dos algoritmos. Existe el algoritmo de compresión que toma una entrada X y genera X_c que es la representación que requiere menos bits y también existe un algoritmo de reconstrucción, el cual funciona sobre la representación comprimida X_c para generar la reconstrucción Y. Este proceso se muestra en la figura 4.1. Así, se seguirá con la convención y se hará referencia a ambos algoritmos de compresión y de reconstrucción juntos para definir el algoritmo de compresión

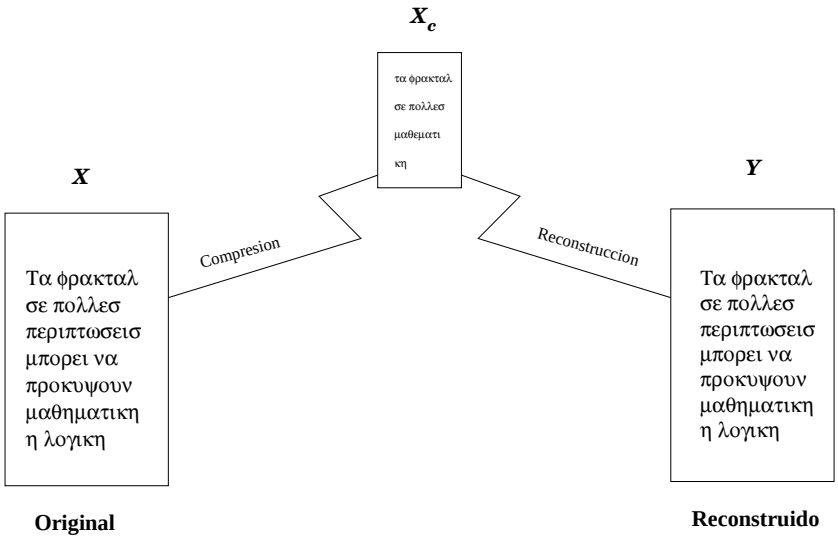


Figura 4.1. Compresión y reconstrucción

Basándose en los requerimientos de reconstrucción, los esquemas de compresión de datos pueden ser divididos en dos clases generales: compresión *lossless* en la cual Y es idéntico a X y compresión *lossy* que permite que Y sea diferente a X.

4.2.1. Compresión *lossless*

Las técnicas de compresión *lossless*, como su nombre lo indican, no implican alguna pérdida de información. Si los datos se han comprimido con

compresión *lossless*, los datos originales se pueden recuperar exactamente de los datos comprimidos. La compresión *lossless* generalmente se utiliza en aquellas aplicaciones que no pueden tolerar ninguna diferencia entre el original y los datos reconstruidos.

La compresión de texto es una parte importante para la compresión *lossless*. Es muy importante que la reconstrucción sea idéntica a la original del texto, ya que diferencias muy pequeñas pueden dar lugar a declaraciones en el texto con significados muy diferentes. Se podría dar el caso en el que al reconstruir algún texto, hubiera cambiado alguna letra o varias y esto modificaría el significado. Algo similar pasa con la información de los archivos en cómputo y para ciertos tipos de datos tales como los expedientes bancarios. (Sayood, 2000)

Si los datos de cualquier tipo serán procesados después para producir mas información, es importante que la integridad de la información se preserve. Por ejemplo, suponer que se comprime una imagen radiológica por medio de una compresión *lossy* y la diferencia entre la reconstrucción Y y la original X es visualmente imperceptible. Si esta imagen fue aumentada más adelante, las diferencias previamente imperceptibles pueden causar la aparición de artefactos[1] que podrían engañar seriamente al radiólogo. Debido a esto, es preciso tener cuidado al usar un esquema de compresión que genere una reconstrucción que sea diferente de la original. Otro ejemplo es el de los datos obtenidos desde los satélites: frecuentemente se procesan tiempo después para medir la vegetación, la deforestación, etcétera, entre una variedad de datos obtenidos. Si los datos reconstruidos no son idénticos a los originales, el proceso puede ser irreversible e imposible obtener los mismos datos otra vez. Por lo tanto, no es recomendable permitir que haya diferencias en el proceso de compresión.

Existen investigaciones que requieren de la compresión, así como de que la reconstrucción sea idéntica al original y en este trabajo, al analizar secuencias de DNA, es de gran importancia utilizar este tipo de compresión. Hay otro tipo de problemas en los que es posible permitir que no se conserve la identidad y así conseguir mayor compresión; entonces se utilizan las técnicas

[1]Distorsión percibida o cualquier otro error atribuible al procedimiento de observación o medición que puede provocar una mala interpretación o resultados erróneos.

de compresión *lossy*.

4.2.2. Compresión *lossy*

Las técnicas de compresión *lossy* implican una cierta pérdida de información y los datos que han sido comprimidos usando estas técnicas generalmente no se pueden recuperar o reconstruir exactamente. A cambio de aceptar esta diferencia en la reconstrucción, es posible obtener una medida de compresión, en general, mucho más altas que al usar la compresión *lossless*.

En muchas aplicaciones, esta falta de una reconstrucción exacta no es un problema. Por ejemplo, cuando se almacena o transmite una conversación se puede dar el caso que ciertos fragmentos de la conversación no tengan la misma claridad pero no es necesario ya que dependiendo de la calidad necesaria para la conversación reconstruida es posible tolerar cierta cantidad de pérdida de la información. Esto hace que dependiendo del medio en el que se reproducirá la conversación, es posible decir cuanta información perdida se puede tolerar. De la misma manera, al ver una reconstrucción de una secuencia de imágenes o vídeo, el hecho de que la reconstrucción sea diferente de la original no es generalmente importante mientras las diferencias no den lugar a artefactos que pudieren llegar a ser muy perceptibles. Así, para comprimir vídeo e imágenes, generalmente se usa la compresión *lossy*.

Una vez que se ha desarrollado un esquema de compresión de datos, es necesario poder medir su funcionamiento y desempeño. Debido a que existen diferentes áreas para sus aplicaciones, se han desarrollado diversos términos para describir y para medir su funcionamiento.

4.2.3. Medidas de desempeño

Un algoritmo de compresión puede ser evaluado de diferentes maneras. Se puede medir la complejidad del algoritmo, la memoria requerida para implementar el algoritmo, qué tan rápido se desempeña un algoritmo en alguna máquina en específico, la cantidad de compresión y cómo la reconstrucción de algún dato se asemeja al original.

Una manera de medir el desempeño de un algoritmo de compresión sobre un serie de datos es ver la razón del número de bits necesarios para representar los datos antes de la compresión y el número de bits necesarios para la representación después de la compresión (Arndt, 2001). Esta razón se definirá como *compression ratio* ó *razón de compresión*.

$$Razón\ de\ compresión = \frac{No.\ bytes\ archivo\ original}{No.\ bytes\ archivo\ comprimido}$$

De la misma manera existe otra medida de desempeño la cual es la razón recíproca a la anterior.

$$Factor\ de\ compresión = \frac{No.\ bytes\ archivo\ comprimido}{No.\ bytes\ archivo\ original}$$

Entre mayor redundancia exista en los datos, mejor razón (factor) de compresión será obtenido.

Como en este trabajo se emplearán secuencias de DNA de diversos organismos, supongamos que almacenar un genoma de una bacteria y en particular el de una bacteria tipo *mycoplasma* por ejemplo, requiere alrededor de 1 millón de bytes. Este mismo genoma, al ser comprimido, requiere de aproximadamente 300,000 bytes. Se dice que la razón de compresión es casi $3.\overline{3}$:1. Por otro lado, también se puede representar la razón de compresión, expresando la reducción en la cantidad de datos necesarios como un porcentaje de los datos originales. En este ejemplo particular la razón de compresión calculada de esta manera será del 70 %. A este porcentaje también se le llamará *tasa de compresión*.

Otra manera de reportar o medir el desempeño de compresión es el de proveer el número promedio de bits que se requieren para representar una muestra. A este número se le llama *índice*. Por ejemplo, si se diera el caso en el que la tasa de compresión de un genoma comprimido fuese del 75 %, para representar cada base de la secuencia de DNA es necesario un byte, y si para representar un byte son necesarios 8 bits, entonces el número promedio de bits por base en la representación comprimida es 2. Por lo tanto se dice que el índice de compresión es 2 bits por base de la secuencia.

4.3. Compresión *lossless*

Compresión de datos sin pérdida o *lossless*, en el sentido de que guarda absolutamente toda la información original (es reversible). Se utilizan para la compresión de datos, en los que no se puede dar pérdida de información, los datos originales pueden ser recuperados en un 100 % después de aplicar sobre los datos comprimidos un algoritmo compatible de descompresión.

Cuando se desea disminuir de tamaño una imagen, una canción o un vídeo, se está dispuesto a perder un poco de información (calidad) a cambio de una tasa de compresión alta. Es decir, se pierde información a cambio de recursos (espacio de almacenamiento, ancho de banda); pero jamás se estaría dispuesto a perder la totalidad de la información a cambio de recursos. Si no se tiene información, para qué se necesitan recursos.

La información se representa mediante símbolos, estos símbolos se pueden codificar de distintas formas, lo que conduce en el caso de las computadoras digitales a archivos con múltiples estructuras de datos (formatos). Existen archivos en los cuales algunos o todos los bits que componen su estructura de datos se vuelven indispensables al momento de ser usados o ejecutados; allí, cambiar el estado de un bit dentro de un archivo, puede provocar una alteración del archivo y la consiguiente pérdida de la totalidad de la información en él contenida. Entonces, ¿como comprimir un archivo con estas características?

La respuesta está en los algoritmos de compresión sin pérdida. Estos algoritmos tienen la propiedad de disminuir el tamaño y conservar la totalidad de la información, la imagen reconstruida es matemáticamente y visualmente idéntica a la original; pero como siempre, hay un precio que pagar. La compresión sin pérdida logra un índice de compresión muy bajo (aproximadamente 2:1), comparado con su complemento la compresión con pérdida (por ejemplo, con las imágenes se puede llegar a un índice de compresión de 25:1 con una pérdida mínima de la calidad).

Esta forma de compresión se caracteriza porque la tasa de compresión que proporciona está limitada por la entropía (redundancia de datos) de la señal original. Es decir, existe un límite teórico de compresión para los compresores sin pérdida; este límite fue enunciado por Shannon en 1948 (véase el capítulo 3), y dice que la longitud media de un código siempre es mayor

o igual a la entropía de la fuente, de lo que se concluye que el código más corto que puede existir para codificar la salida de una fuente de mensajes está limitado por la misma fuente.

Los algoritmos de compresión sin pérdida pueden clasificarse en dos tipos: sustitucionales (o basados en diccionario) y estadísticos. La principal distinción entre ambos es que en el caso de los métodos estadísticos, la codificación de un símbolo se basa en el contexto en el que éste ocurre, mientras que los métodos sustitucionales agrupan símbolos creando un tipo de contexto implícito (Witten et al., 1999). En general, un compresor estadístico logra mejores razones de compresión que un compresor sustitucional, pero la complejidad computacional y los requerimientos de memoria para los compresores estadísticos son mucho mayores que para los compresores sustitucionales (Jung y Burleson, 1995).

4.3.1. Algoritmos de compresión estadísticos

Los compresores estadísticos son los que utilizan la información de las probabilidades de los mensajes de la fuente para construir una codificación.

Los algoritmos de compresión estadísticos usan las propiedades estadísticas de los datos que van a comprimirse para asignar códigos de longitud variable a los símbolos individuales en los datos. Existen varios algoritmos estadísticos propuestos, la principal diferencia entre ellos es la forma en la que cada uno obtiene las probabilidades de los símbolos (Witten et al., 1999). Los algoritmos estadísticos emplean un modelo para obtener las probabilidades de los símbolos, la calidad de la compresión que se logra depende de qué tan bueno sea ese modelo. Para obtener buena compresión, la probabilidad se estima con base en el contexto en el que ocurren los símbolos. Dadas las probabilidades de los símbolos, la codificación se encarga de convertir dichas probabilidades en una cadena de bits para obtener los datos en forma comprimida.

No parece extraño que, habiendo usado una medida de información basada en las probabilidades, se hallaran compresores que utilizan las propiedades estadísticas de la fuente de información para mejorar la codificación (el conjunto de símbolos de salida asociados a cada mensaje emitido por la fuente)

de los mensajes de la fuente. Éstos son los compresores estadísticos.

Este tipo de compresores parten de:

- Una fuente de información de n mensajes.
- Las probabilidades de aparición de cada mensaje de la fuente (que pueden ser extraídas de forma experimental o pueden ser dadas y fijas).
- Un alfabeto de salida que consta de una serie de símbolos (por ejemplo, el alfabeto binario consta de los símbolos 0 y 1).

y su objetivo es asignar una codificación para los mensajes de la fuente. Una codificación es una función que a cada mensaje de la fuente le asigna una cadena de símbolos del alfabeto de salida. Esta codificación debe ser tal que aproveche la redundancia en la información dada por la fuente para producir compresión. Si se utiliza un alfabeto binario, la esperanza matemática de la longitud de las cadenas de símbolos de la codificación se debe acercar a la cota teórica de Shannon. Si es igual, la compresión es perfecta: hay una máxima eficiencia.

Entre los algoritmos de compresión que usan un modelo estadístico es posible encontrar varios tipos:

a) Algoritmos de compresión del tipo Huffman ó Shannon-Fano.
b) Algoritmos de compresión aritméticos.
c) Algoritmos de compresión predictivos.

Compresión Huffman

La codificación de Huffman es un método propuesto en 1952 por David Huffman (Huffman, 1952). La idea detrás de la codificación de Huffman es asignar códigos binarios lo más cortos posibles a aquellos símbolos que ocurren con mayor frecuencia en los datos. Los símbolos con poca frecuencia tendrán asignado códigos binarios de longitud más grande. El óptimo desempeño del algoritmo se consigue cuando el número de bits asignado a cada

carácter es proporcional al logaritmo de la probabilidad del mismo.

El algoritmo de Huffman construye un árbol binario mediante el cual, asigna los códigos a los símbolos de entrada. Se realiza una primera pasada sobre los datos a comprimirse para obtener las estadísticas de los símbolos; estos símbolos son ordenados en una lista de acuerdo a su probabilidad. Los elementos de esta lista ordenada de símbolos serán los nodos iniciales para la construcción del árbol.

Debido a su facilidad de cómputo, es ampliamente utilizada en programas de compresión de datos y en esquemas de compresión de imágenes como JPEG. Los compresores Huffman presentan las siguientes características:

a) Usa códigos de longitud variable, asignando a los símbolos más frecuentes un código de compresión de menor longitud implicando asignar códigos de mayor longitud a los menos frecuentes.

b) El número de bits por código de compresión es un número entero. Esto provoca que si la entropía de un símbolo es 2.5 bits, el código de Huffman asignará un código con una longitud de 2 o 3 bits, pero no 2.5 bits. Debido a esto, la codificación de Huffman no puede ser considerada como un método de codificación óptimo, pero si es la mejor aproximación para construir códigos de compresión con un número entero de bits. En consecuencia, sería imposible la codificación eficiente de fuentes binarias (que sólo generan dos símbolos) como es el ejemplo de una imagen binaria (que sólo tiene dos tonos). En estos casos la extensión de la fuente es la solución ya que, en general, el número de símbolos-bloque (formados por la concatenación de símbolos) aumentará respecto del número de símbolos y si la fuente no es aleatoria, una representación de longitud variable siempre será conveniente.

c) Aunque los códigos asociados a símbolos diferentes tienen diferente longitud, éstos pueden ser decodificados de forma única e instantánea. Esto debido a que en la codificación Huffman se usan códigos instantáneos. Dichos códigos se basan en que ningún código es prefijo (forma parte inicial) de otro.

Para construir un código de Huffman normalmente se utiliza un árbol binario que se construye a partir de las probabilidades de los símbolos que deseamos codificar. El diseño de dicho árbol se realiza de la siguiente forma:

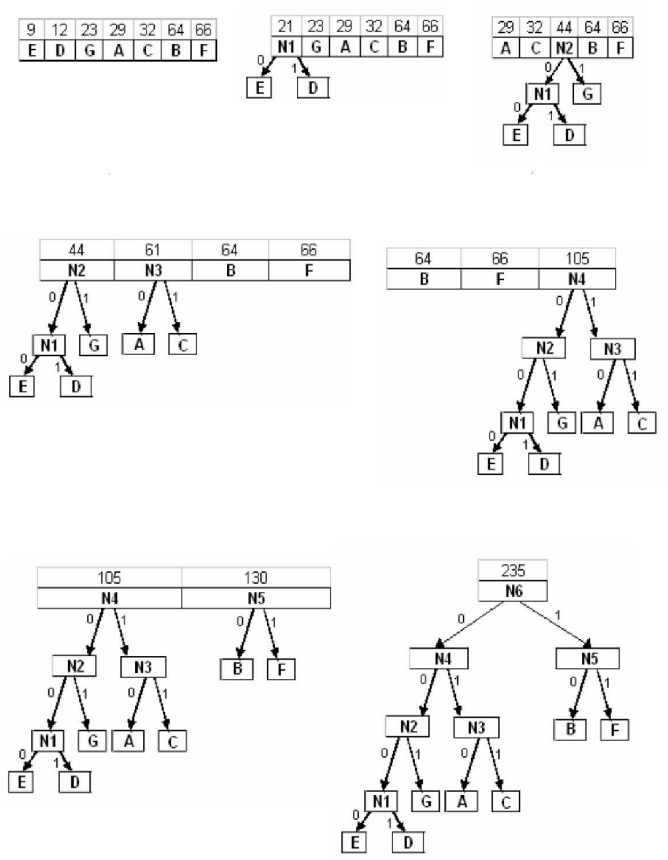


Figura 4.2. Creación de un árbol binario dadas las probabilidades de los símbolos en el mensaje a comprimir

Paso 1. Crear tantos nodos (que serán las hojas del árbol) como símbolos vayamos a codificar. En cada nodo se almacena el símbolo codificado y su probabilidad. Estos nodos forman la lista de nodos sin procesar.

Paso 2. Extraer dos nodos (L y R) de la lista de nodos sin procesar que tengan una probabilidad mínima.

Paso 3. Insertar en la lista un nodo que sea padre (F) de los dos nodos seleccionados. Este nodo no tiene asociado ningún símbolo y su probabilidad

será la suma de las probabilidades de los nodos hijo. Se construye un subárbol siendo F el padre, L y R los hijos izquierdo y derecho respectivamente. La arista entre los nodos F y L se etiqueta como 0 y la arista entre los nodos F y R se etiqueta como 1.

Paso 4. Repetir los pasos 2 y 3 hasta que únicamente se tenga un nodo en la lista de nodos sin procesar, nodo que debe tener una probabilidad igual a 1 y que se llama nodo raíz del árbol de Huffman.

Debido que en cada iteración se extraen dos nodos de la lista y se inserta otro, el número de iteraciones necesarias para construir un árbol con N hojas (el número de símbolos que se desea codificar) es N. Una vez construido el árbol, el código de compresión asociado a un símbolo se calcula recorriendo el árbol desde la raíz hasta el símbolo codificado (situado siempre en una hoja).

En la figura 4.2 se muestra un ejemplo de la construcción de un árbol binario de Huffman.

En el cuadro 4.1 se muestran los códigos correspondientes al ejemplo de la figura 4.2

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

Cuadro 4.1. Códigos de Huffman

Compresión Aritmética.

La codificación aritmética (Witten et al., 1987) es una técnica que obtiene una buena razón de compresión, aunque es un método lento que requiere por lo menos de una multiplicación por cada símbolo de entrada. En teoría,

la codificación aritmética asigna un único *codeword* a cada posible conjunto de datos. La compresión aritmética se basa también en las probabilidades de ocurrencia de los mensajes emitidos por la fuente de información. Se basa en la representación de un valor del intervalo [0,1] con más decimales (más precisión) entre más información contengan los datos a comprimir. La codificación aritmética se realiza a partir de los pasos siguientes:

1. Inicialmente, el intervalo actual es [L, H) = [0, 1), donde L es el valor inferior del intervalo y H es el valor superior.
2. Para cada símbolo s en la entrada se realizan dos acciones:
   - El intervalo actual se divide en subintervalos. El tamaño del subintervalo es proporcional a la probabilidad estimada de los símbolos.
   - Se selecciona el subintervalo que corresponde a s, este intervalo es ahora el intervalo actual.
3. Cuando el archivo se ha procesado, la salida final como resultado de la compresión es un número dentro del intervalo actual, generalmente, aquel que ocupe menos bits.

Para cada símbolo procesado, el intervalo se hace más pequeño y requiere de más bits para representarlo. Supóngase que se desea codificar la secuencia de caracteres ABCBAA. Las frecuencias de cada uno de los símbolos son: 3 A's, 2 B's y 1 C. Las probabilidades y valores de las variables para cada uno de los símbolos se muestran en el cuadro 3.5.

Símbolo	Frecuencia	Probabilidad	Rango[L,H]	LR	HR
A	3	3/6=0.50	[0.00,0.5)	0.00	0.50
B	2	2/6=0.33	[0.5,0.83)	0.50	0.83
C	1	1/6=0.16	[0.83,1.0)	0.83	1.00

Cuadro 4.2. Valores de las variables en la codificación aritmética

Los símbolos y frecuencias del cuadro anterior se escriben en el archivo de salida antes de que inicie la compresión del archivo. La codificación inicia definiendo dos variables L = 0 y H = 1 formando el intervalo inicial [0, 1).

Conforme los caracteres a codificar aumentan, el cálculo se vuelve más complicado. El hecho de realizar cálculos sobre números flotantes implica mayor tiempo de procesamiento haciendo al método lento respecto a otros. Otra de las ventajas de la codificación aritmética frente a la codificación de Huffman es que la representación se calcula requiriendo de menos memoria (Witten et al., 1999). Este algoritmo es muy eficiente, sólo que para ejecutarse es necesario conocer las probabilidades de ocurrencia.

Compresión Predictiva.

Estos algoritmos de compresión intentan predecir el siguiente mensaje o símbolo de entrada tomando como base de conocimiento los datos procesados hasta ese momento. La codificación predictiva presenta las siguientes características:

Una alta velocidad de compresión/descompresión, al actuar sobre un mensaje cada vez y realizar una predicción que generalmente suele ser de baja complejidad computacional.

- Son totalmente adaptativos.
- Su implementación generalmente es sencilla.
- En comparación con los anteriores(semi-adaptativos), la razón de compresión es moderada.

En este tipo de codificación la compresión está sujeta al éxito o fracaso de una predicción, si el mensaje que se encuentra a la entrada coincide con el predicho, su codificación se podrá hacer con menos bits. Si no, su codificación se hará con más bits.

La codificación predictiva no está representada por un único algoritmo; aún así, se tratará de describir un patrón en los pasos que se deben llevar acabo para codificar una cadena de símbolos usando un algoritmo predictivo, se debe tener un cuenta que estos pasos no son genéricos. Así, los pasos serían los siguientes:

Paso 1: Se parte de una base de conocimiento inicial, ésta nos ayudará a que las predicciones iniciales tengan una probabilidad de acierto alta.

Paso 2: Se hace una predicción acerca del mensaje que llegará. Si acertamos en la predicción, se procede a codificar el mensaje de una forma tal que ocupe menos bits. En caso contrario la codificación se hace mas extensa y la cantidad de bits usados aumenta.

Paso 3: Se modifica la base de conocimiento con base en el resultado obtenido en el paso 2. Si aún quedan más mensajes por codificar, entonces se regresa al paso 2. En caso contrario, la codificación ha finalizado.

4.3.2. Algoritmos de compresión basados en diccionario

Los métodos basados en diccionario usan el principio de reemplazar cadenas de datos con *codewords* que identifican a esa cadena dentro de un diccionario (Witten et al., 1999). El diccionario contiene una lista de subcadenas y *codewords* asociados a cada una de ellas. El diccionario que contiene las cadenas de símbolos puede ser estático o adaptable (dinámico). Los métodos basados en diccionario, a diferencia de los métodos estadísticos, usan *codewords* de longitud fija y no requieren de las estadísticas de los símbolos para realizar la compresión. Prácticamente, todos los métodos de compresión sustitucionales están basados en los métodos de compresión desarrollados por Abraham Lempel y Jacob Ziv en la década de los 70's, los métodos LZ77 y LZ78. En ambos métodos, una subcadena es reemplazada por un apuntador a donde dicha cadena haya ocurrido previamente en el diccionario, el cual, se crea dinámicamente. Las principales diferencias entre los métodos sustitucionales radican en cómo los apuntadores son representados y en las limitaciones que imponen sobre lo que los apuntadores pueden representar.

Estos algoritmos no requieren un conocimiento de la probabilidad de aparición de cada símbolo. Por ello serán útiles en aquellas aplicaciones donde no sea posible conocer tales probabilidades. Se basan en que algunas cadenas de símbolos se repiten frecuentemente. Las cadenas procesadas hasta el momento pasan a ser parte de un diccionario indexado, el cual puede definirse de forma explícita o implícita. Durante el proceso de compresión, si la cadena que se está procesando, coincide con una entrada en el diccionario, la

salida emitida por el compresor es el índice asociado a dicha cadena; índices que generalmente son códigos de menor longitud que la cadena a la que hace referencia.

Diccionario estático: Es apropiado si se conoce la estructura repetitiva de los datos. Se almacenan en el diccionario sólo los patrones más frecuentes y sus palabras código (suelen ser los índices); si los patrones son pocos, las palabras código son cortas.

Diccionario dinámico: Es apropiado para compresores generales adaptables a fuentes de distintas características. Construyen el diccionario conforme procesan la entrada. En el descompresor se reconstruye el mismo diccionario.

Existen varios algoritmos basados en el uso de diccionarios, la diferencia fundamental entre todos ellos es la manera en que cada uno crea y gestiona el diccionario. Los esquemas más conocidos son: LZ77 y LZ78. Estos esquemas se han ido optimizando desde el momento de su aparición. En 1982 James Storer y Thomas Szymanski presentaron una optimización del LZ77, la cual reducía el tamaño de los índices de referencia. Esta optimización fue llamada LZSS. En 1984 Terry Welch presentó y patentó la optimización más conocida del LZ78 y la llamó el LZW. Más adelante se describen estas versiones de los esquemas originales de Abraham Lempel y Jacob Ziv.

Compresión RLE

Es el compresor mas sencillo dentro de los compresores basados en diccionario. Consigue reducir el tamaño de las cadenas de entrada eliminando las secuencias repetidas de caracteres contiguos. Básicamente el funcionamiento de este compresor tiene como objeto sustituir la secuencia repetida contigua de un carácter por un par ordenado ("Carácter", "Número de veces que se repite").

Codificación RLE (Run-Length Encoding). A continuación se presentarán de la manera más general posible, los pasos que se deben llevar a cabo en este proceso de compresión:

Paso 1: Se establece un *apuntador* en el primer carácter de la *cadena de*

entrada.

Paso 2: Se lee el carácter apuntado y se almacena en *carácter actual*, se ubica en cero el *contador de repeticiones* y el *apuntador* se avanza al siguiente carácter .

Paso 3: Se lee el carácter apuntado por el *apuntador* y se compara con *carácter actual*; si ambos caracteres son iguales se incrementa en uno el *contador de repeticiones* y se regresa al paso 3.

En caso contrario, se examina el valor del *contador de repeticiones*; si éste es mayor que la longitud mínima de ganancia. Se concatena la codificación del par ordenado a la *cadena de salida*; de no ser así, se concatena a la *cadena de salida* el *carácter actual* tantas veces como lo indique *contador de repeticiones.*

Paso 4: Si el carácter apuntado es diferente del indicador de fin de cadena, entonces regresar al paso 2. De lo contrario la *cadena de salida* es la cadena comprimida y la compresión habrá finalizado.

Los pasos anteriormente descritos dejan abierta la forma en la que se codificarán los pares ordenados [“Carácter”, “Repeticiones”]; es precisamente en esa codificación donde algunos compresores toman ventaja sobre sus similares RLE.

Un ejemplo para este tipo de compresión, es el considerar una pantalla que contiene texto en negro sobre un fondo blanco. Habría muchas secuencias de este tipo con píxeles blancos en los márgenes vacíos y otras secuencias de píxeles negros en la zona del texto. Supongamos una única línea, con N representando las zonas en negro y B las de blanco:

BBBBBBNBBBBBBNNNBBBBBBBBBBBNBBBBBBB

Al aplicar la codificación run-length a esta línea, se obtiene lo siguiente:

6BN6B3N12BN7B

donde la secuencia obtenida tiene menor longitud al de la original, o sea, se comprimió.

Compresión LZ77

En 1977 A. Lempel y J.Ziv presentaron el método de compresión LZ77 (Ziv y Lempel, 1977), mediante el uso de un diccionario implícito que consiste en una porción de la secuencia previamente codificada, el codificador examina la secuencia de caracteres de entrada a través de una ventana deslizante tantas veces como cadenas de símbolos son codificadas y que consta de dos partes:

- Un *buffer de búsqueda*: porción de la secuencia previamente codificada que establece el diccionario actual.
- Un *buffer hacia el frente (look-ahead)*: porción de la secuencia no codificada hasta el momento.

como se ve en la figura 4.3. Inicialmente, el *buffer de búsqueda* puede rellenarse con algún símbolo inicial mientras que el *buffer hacia el frente* se rellena con los primeros símbolos del archivo de entrada. En la práctica, el *buffer de búsqueda* es de algunos cientos de bytes mientras que el *buffer hacia el frente* es de decenas de bytes.

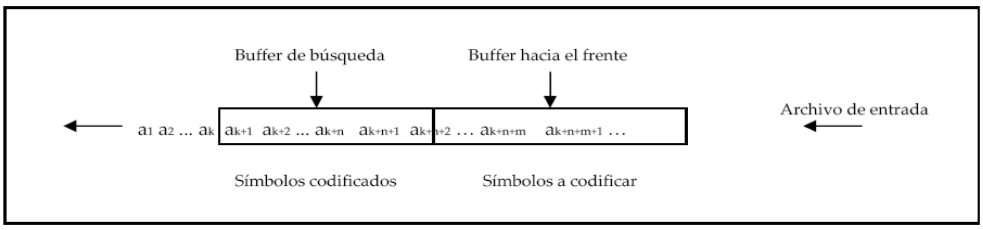


Figura 4.3. LZ77

El algoritmo para la codificación se muestra a continuación:

1. El codificador busca la subcadena más grande en el *buffer de búsqueda* (de derecha a izquierda) que sea el prefijo del *buffer hacia el frente*.
   a) Si tal subcadena existe, la codificación consiste de un triplete $T = (O, L, C)$, donde O es la distancia en símbolos desde inicio de la subcadena encontrada hasta el final del *buffer de búsqueda*, L es la

longitud de la subcadena encontrada y C es el símbolo siguiente a la subcadena encontrada en el *buffer hacia el frente*.

b) Si la cadena no existe, O y L son puestos a cero y C es el primer símbolo del *buffer hacia el frente* (en la figura 4.3, el símbolo a_{k+n+1}).

2. Ambos *buffers* se recorren hacia la izquierda $L + 1$ posiciones.

Para codificar la secuencia que hay en el *buffer hacia el frente look-ahead*, se mueve un apuntador sobre el *buffer de búsqueda* hasta que se encuentra una secuencia de caracteres que concuerda total o parcialmente con la cadena contenida en el *buffer hacia el frente*, posteriormente esta coincidencia se codifica como un triplete ordenado (O, L, C). La compresión se logra gracias a que la codificación de dicho triplete ordenado es de menor longitud que la codificación de la secuencia original.

La codificación de una cadena de símbolos se lleva a cabo haciendo uso de los dos *buffers*. Durante todo el proceso de codificación, la cadena de entrada se divide en pequeñas subcadenas que son almacenadas temporalmente en los *buffers* mientras son procesadas una a una. La secuencia que se está codificando es almacenada en el *buffer hacia el frente* y la secuencia de referencia o diccionario implícito es almacenada en el *buffer de búsqueda*.

Para codificar la secuencia que hay en el *buffer hacia el frente (look-ahead)*, se mueve un apuntador de búsqueda hacia atrás sobre el *buffer de búsqueda* hasta que encuentra una secuencia de caracteres que coinciden con una secuencia idéntica contenida en el *buffer hacia el frente*. Una vez que ha encontrado la coincidencia de mayor longitud, se codifica como un triplete ordenado (O, L, C) y se avanza la ventana deslizante en una cantidad igual a la longitud de la coincidencia (L). La distancia desde la ubicación del apuntador de búsqueda al inicio del *buffer hacia el frente* se denomina "Desplazamiento" (O) y "Siguiente símbolo" (C) es el símbolo contiguo al último símbolo incluido en la cadena que coincide. Si no encuentra coincidencia, se debe concatenar a la cadena de salida el primer símbolo del *buffer hacia el frente* y se avanza en un símbolo la ventana deslizante. La compresión se logra gracias a que la codificación de dicho triplete ordenado es de menor longitud que la codificación de la secuencia original.

La codificación de los tripletes ordenados (O, L, C) es de suma importancia y determina en gran parte la eficiencia de la compresión; el único requisito que debe cumplir esta codificación, es que el triplete ordenado pueda ser diferenciado de los demás símbolos codificados en la cadena de salida durante el proceso de decodificación. Una de las formas mas eficientes de codificar los tripletes es la presentada en 1982 por James Storer y Thomas Szymanski, en su modelo llamado LZSS (Storer y Szymanski, 1982). La diferencia principal es en la salida, LZ77 siempre da un triplete ordenado (O,L,C), aún si la coincidencia es de un sólo byte. La mejora introducida en el LZSS es el uso de banderas, que ocupan un sólo bit y nos informan si lo que viene luego es un símbolo o un triplete.

Existen distintas variantes mejoradas del algoritmo LZ77, cada una de estas mejoras obtiene una mayor eficiencia al optimizar uno de los siguientes aspectos del algoritmo básico, tales aspectos son:

(a) Codificación de tripletes: como se dijo anteriormente, entre menos bits se usen para codificar un triplete mayor será la compresión obtenida. La optimización en este aspecto se lleva a cabo mediante el uso de bits de bandera como en el caso del LZSS, que usa un bit para identificar si el símbolo leído es el inicio de un triplete. De otro modo también se puede optar por una codificación de longitud variable de los tripletes. Este algoritmo es usado en los programas de compresión tales como: *pkzip, zip, LHarc, png, gzip, arj.*

(b) Tamaño de los *buffers*: El algoritmo LZ77 sólo hace uso del pasado cercano. Es decir el espacio de búsqueda de repeticiones está restringido a un pequeño subconjunto de un total de todos los símbolos procesados con anterioridad; el uso de un *buffer de búsqueda* limitado tiene su justificación, en función de un ahorro en el número de bits necesarios para codificar los tripletes, pero hay situaciones en las cuales el uso de un *buffer* de mayor tamaño aumenta en un alto grado la eficiencia de la compresión, al aumentar el tamaño del *buffer de búsqueda* las probabilidades de encontrar coincidencias significativas también se incrementan, todo esto con un costo, que consiste en un aumento en el número de bits necesarios para codificar un triplete ordenado, de esta forma se eleva el número mínimo de símbolos en una secuencia coincidente para que ésta se considere significativa.

El método LZ77 no requiere de las frecuencias de ocurrencia de los símbolos presentes en los datos que se van a comprimir. El método sólo busca patrones de subcadena en el *buffer de búsqueda* que ocurran en el *buffer hacia el frente*, suponiendo que dichos patrones ocurren continuamente, al menos dentro de la subcadena que engloba ambos *buffers*. Si esta suposición se cumple en el conjunto de datos que se va a comprimir, el método logra buenas razones de compresión. El método LZ77 ha sido mejorado desde su invención, entre las mejoras destacan la utilización de campos O y L de longitud variable y el uso de diferentes tamaños de diccionario (longitud de la ventana). La utilización de una ventana más grande implica mayor tiempo de búsqueda de los patrones y obliga a la utilización de mejores estructuras de datos para realizar la implementación. Por otra parte, el utilizar una ventana más grande implica el uso de campos O y L más grandes con lo que la razón de compresión se ve afectada. Otro factor a considerarse es la posible expansión del archivo en vez de la reducción deseada. Debido a que se requiere un *codeword* de tres campos, esto puede producir la expansión del archivo de salida para los casos donde la subcadena encontrada sea de longitud 1 o 2. Una posible solución es escribir al archivo de salida *codewords* cuando la cadena encontrada sea de longitud mayor a 3 y escribir los símbolos sin comprimir cuando la cadena encontrada sea de longitud menor o igual a 3. Esto implica el uso de un símbolo especial que permita diferenciar al decodificador entre *codewords* y datos no comprimidos.

Un ejemplo del algoritmo Lempel-Ziv es el siguiente:

1. Inicializar el diccionario que contiene todos los bloques de longitud uno (D={a,b}).
2. Se busca el bloque más largo W que haya aparecido en el diccionario.
3. Se codifica W por su índice en el diccionario.
4. Se añade al diccionario W seguido por el primer símbolo en el siguiente bloque.
5. Regresar al paso dos.

y se obtiene el siguiente resultado (figura 4.4):

Diccionario			
Indice	Entrada	Indice	Entrada
0	a	7	b a a
1	b	8	a b a
2	a b	9	a b b a
3	b b	10	a a a
4	b a	11	a a b
5	a a	12	b a a b
6	a b b	13	b b a

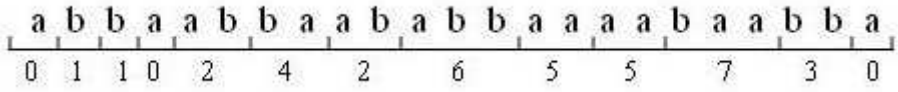


Figura 4.4. Diccionario Lempel-Ziv.

Compresión LZ78

Este algoritmo de compresión basado en diccionario fue presentado en 1978 nuevamente por A. Lempel y J. Ziv (Ziv y Lempel, 1978), en el que a diferencia de su anterior trabajo el LZ77, no hace uso de una ventana deslizante, el LZ78 a cambio construye explícitamente un diccionario. La ventaja de usar un diccionario explícito es que elimina la limitación de recurrencia cercana que suponía el LZ77 con su ventana deslizante, al hacer uso de toda la información procesada con anterioridad la probabilidad de hallar secuencias de símbolos repetitivas aumenta.

El diccionario consiste en una tabla con 2 columnas y n filas, la primera columna contiene un índice que se asocia al contenido de la segunda columna, la cual contiene secuencias de símbolos que se han encontrado hasta el momento en la cadena de entrada. El proceso de compresión inicia con un diccionario vacío y a medida que la entrada es procesada se colocan en el diccionario todas las secuencias que hacen su aparición por primera vez. Cuando en la entrada se presenta una secuencia que ya está en el diccionario, en la cadena de salida esta secuencia es representada por un par ordenado [índice, Siguiente_símbolo] cada uno de los campos de este par tiene el siguiente significado:

- *índice*: especifica la posición que ocupa en el diccionario la secuencia previa más larga que coincide con la entrada.
- *Siguiente_símbolo*: es el código del carácter que viene a continuación de la secuencia coincidente.

Si no se encuentra coincidencia, i = 0, en cualquier caso cada nueva pareja generada [*índice*, *Siguiente_símbolo*] se añade al diccionario.

Se tiene un diccionario que contiene las cadenas que han ocurrido previamente. El diccionario está vacío inicialmente y su tamaño está limitado por la memoria disponible. Para ilustrar la forma en la que el método funciona, se considera un diccionario (arreglo lineal) de N localidades con la capacidad de almacenar una cadena de símbolos en cada una de ellas. El diccionario se inicializa guardando en la posición cero la cadena vacía. El algoritmo de codificación se muestra a continuación:

- Mientras existan símbolos a la entrada
  1. $S = Null$, $Pos = 0$
  2. X = siguiente símbolo de entrada, $S = S \cdot X$
  3. Mientras S exista en el diccionario
     a) $Pos = Pos(S)$
     b) X = siguiente símbolo de entrada, $S = S \cdot X$
  4. Salida: (Pos, X)
  5. Guardar S en el diccionario

El proceso es iterativo y termina cuando ya no existen más símbolos a la entrada para codificar. En cada iteración S se inicializa a $Null$ ($S = Null$ indica una cadena vacía que siempre se encuentra en la posición cero del diccionario). El símbolo X del archivo de entrada se lee y se busca la cadena $S{\cdot}X$ (concatenación de S y X) en el diccionario, si la cadena $S{\cdot}X$ se encuentra en el diccionario, S es ahora $S{\cdot}X$ y se lee un nuevo símbolo X. Nuevamente, se busca $S{\cdot}X$ en el diccionario y si la cadena se encuentra, se vuelve a leer otro símbolo de entrada y el proceso se repite buscando nuevamente $S{\cdot}X$ en el diccionario. Si la cadena $S{\cdot}X$ no se encuentra en el diccionario, se guarda la cadena $S{\cdot}X$ en una posición disponible en el diccionario y se escribe al

archivo de salida la posición de S dentro del diccionario y el símbolo X.

A diferencia del método LZ77, ahora los *codewords* se componen sólo de dos campos, un campo apuntador o índice que identifica la posición de una cadena dentro del diccionario y un campo que contiene el último símbolo que se ha leído de la entrada. Entre más grande sea el diccionario, más cadenas son almacenadas y se tienen coincidencias de cadenas más grandes que mejoran la razón de compresión, pero se requieren apuntadores más grandes y el tiempo de búsqueda se incrementa.

El principal problema del LZ78 es que el tamaño del diccionario no está acotado. En la medida en que aparecen nuevas cadenas, éstas son insertadas en el diccionario; en la práctica, recursos como memoria y velocidad de procesamiento son limitados; además, un diccionario demasiado extenso implica el uso de índices largos que requieren gran cantidad de bits lo cual termina por degenerar la eficiencia del compresor, estas limitaciones hacen que en un momento dado se deba detener la construcción del diccionario y proceder con alguna de las siguientes alternativas:

* Eliminar el diccionario: Consiste en eliminar algunas de las entradas del diccionario, la selección de las entradas que serán eliminadas se puede hacer basándose en estadísticas que indican cuales son las entradas menos referenciadas. Es decir las cadenas menos frecuentes.

* Nuevo diccionario: Consiste en eliminar todas las entradas del diccionario hasta quedar con un diccionario vacío.

* Diccionario estático: Consiste en detener la inserción de nuevas entradas en el diccionario y a partir de ese momento continuar manejándolo como un diccionario estático.

Otro aspecto importante que se debe tener en cuenta a la hora de implementar un compresor tipo LZ78 es la definición y la administración de la estructura de datos del diccionario, la importancia de este aspecto radica en el uso extensivo que se hace de dicha estructura a lo largo del proceso de compresión/descompresión.

Compresión LZW

El método LZW (Welch, 1984) es una variante del método LZ78. El *codeword* consiste ahora de únicamente una referencia a la posición en el diccionario de la cadena encontrada. En el caso de codificar símbolos de 8 bits, las primeras 256 posiciones del diccionario se inicializan con los 256 símbolos antes de iniciar la codificación. Este paso inicial asegura que cualquier símbolo leído de la entrada siempre se localizará en el diccionario con lo que es posible manejar únicamente un apuntador a la posición del diccionario como código de salida. El codificador lee cada símbolo de la entrada y los acumula en una cadena S. Cada vez que un símbolo X es leído de la entrada, el símbolo se concatena con S y la cadena resultante se busca en el diccionario. Siempre que la concatenación es encontrada en el diccionario, un nuevo símbolo X es leído de la entrada, se concatena a S y se realiza la búsqueda de la nueva cadena $S{\cdot}X$. Cuando la concatenación $S{\cdot}X$ no se encuentra en el diccionario (S se encuentra en el diccionario pero $S{\cdot}X$ no se encuentra), el codificador escribe a la salida la posición en el diccionario de la cadena S, guarda $S{\cdot}X$ en una localidad disponible en el diccionario e inicializa S con el símbolo X.

4.3.3. Compresión mediante transformada

Esta técnica de compresión consiste en expresar los datos que se pretenden comprimir en una forma que proporcione algún beneficio, bien sea que aplicando la transformada, el volumen de los datos se reduzca o que al aplicar la transformada, la redundancia presente en los datos pueda ser aprovechada con más eficiencia por cualquier otra técnica o algoritmo de compresión.

Estas son transformadas sin pérdida o reversibles que generalmente consisten en una reordenación de los datos. Obteniendo como salida un volumen de datos igual al original; pero con la diferencia de que los datos de salida presentan una alta redundancia.

Transformada de Burrows-Wheeler (BWT).

En el año 1994, Michael Burrows y David Wheeler publicaron un reporte de investigación titulado, “A Block-sorting Lossless Data Compression Al-

gorithm", allí se describe un algoritmo para la compresión de datos que se desarrolló tomando como base una transformación descubierta por Wheeler en 1983 y que hasta ese momento no había sido publicada (Burrows et al., 1994).

Básicamente la BWT es un algoritmo que toma un bloque de datos y lo reorganiza usando un algoritmo de ordenación. El resultado es un bloque que contiene exactamente los mismos elementos que el bloque original, la única diferencia es la forma en la que están ordenados. La transformación es reversible, por consiguiente es posible recuperar el orden original de los elementos sin ninguna distorsión. Es decir, sin pérdida alguna de los datos originales.

El método descrito por Burrows y Wheeler es en realidad un compuesto de tres algoritmos diferentes:

- La transformación Burrows-Wheeler. Un preprocesador sin pérdida que es parte principal para la clasificación por bloque en la que los datos de entrada son reordenados para realizar la compresión. Es importante para tener una buena tasa de compresión.
- Codificador *move-to-front*. La idea de este esquema es mantener una lista que representa los símbolos de un alfabeto para posteriormente comprimirlos.
- Codificador estadístico. Este algoritmo es el que llevará a cabo la compresión y una opción, que generalmente se usa es el de Huffman.

Este algoritmo se usa en el programa de compresión *bzip2* que forma parte del análisis en el siguiente capítulo. En general, el desempeño de este algoritmo es bueno, teniendo mejor tasa de compresión que el usado por LZ77.

De estos tres métodos, los dos primeros constituyen la parte fundamental del algoritmo. Estos dos algoritmos acoplados forman una transformación completamente reversible (sin pérdida), permitiendo obtener los datos de entrada a partir de la salida de la transformación. El funcionamiento de este algoritmo es el siguiente:

1. El preprocesador de clasificación opera exclusivamente con base en bloques y lleva a cabo la transformación BWT. Una forma para entender esta idea es pensar en el bloque de entrada organizada como una matriz circular donde, para cada símbolo, los símbolos subsiguientes se utilizan como un predictor. Esta predicción se utiliza para agrupar los símbolos con los mismos vecinos del lado derecho y se realiza como un proceso en dos fases. La primer fase crea todos los cambios cíclicos del bloque de entrada, cuyo tamaño es normalmente una potencia de dos. La cadena original se mantiene sin cambios en la primera fila de la matriz resultante. Si la longitud del bloque es n entonces existen únicamente n rotaciones de la cadena original (a la izquierda). Estas rotaciones son ahora consideradas como los renglones de una matriz de N x N símbolos.

La segunda fase consiste en el ordenamiento de esta matriz. Este ordenamiento se hace a partir de su orden lexicográfico. Como cada renglón es una cadena de caracteres, los renglones se ordenan en forma creciente. Así, en el primer renglón (índice cero) de la matriz, se encuentra el renglón que tiene el mínimo valor lexicográficamente hablando y en el i-ésimo renglón de la matriz resultante se localiza el bloque original de entrada. De esta manera la salida de estas fases consta de:

- La última columna de la matriz resultante, o sea, la del lado derecho a la que se le llama L.
- i es el índice del renglón donde quedó el bloque o cadena original.

2. Codificador *move-to-front*. Aquí, la codificación de un carácter consiste en el número de caracteres distintos que se han visto desde su última aparición, en el caso de que ya haya aparecido. Esto significa ir metiendo a una pila los caracteres que vamos leyendo y el codificador obtiene el índice de la posición en la pila del carácter que se haya leído. Enseguida se mueve el carácter al tope de la pila siendo éste el que se acaba de leer.

3. Los valores obtenidos por el codificador *move-to-front* se aplican al codificador estadístico que en el caso de este algoritmo de compresión se utiliza la codificación de Huffman, así como en la implementación de *bzip2*.

La matriz de transformación descrita anteriormente puede requerir bastante espacio de almacenamiento y puede resultar un algoritmo no factible

como tal. Sin en cambio, al acoplarla con un algoritmo de compresión simple, resulta un buen método.

Un ejemplo de como funciona la BWT es el siguiente:

Se considera la cadena "good, jolly good", la cual es un ejemplo con redundancia alta. 1) En el primer paso se obtienen los cambios o rotaciones de la cadena de entrada.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	g	o	o	d	,		j	o	l	l	y		g	o	o	d
1	o	o	d	,		j	o	l	l	y		g	o	o	d	g
2	o	d	,		j	o	l	l	y		g	o	o	d	g	o
3	d	,		j	o	l	l	y		g	o	o	d	g	o	o
4	,		j	o	l	l	y		g	o	o	d	g	o	o	d
5		j	o	l	l	y		g	o	o	d	g	o	o	d	,
6	j	o	l	l	y		g	o	o	d	g	o	o	d	,	
7	o	l	l	y		g	o	o	d	g	o	o	d	,		j
8	l	l	y		g	o	o	d	g	o	o	d	,		j	o
9	l	y		g	o	o	d	g	o	o	d	,		j	o	l
A	y		g	o	o	d	g	o	o	d	,		j	o	l	l
B		g	o	o	d	g	o	o	d	,		j	o	l	l	y
C	g	o	o	d	g	o	o	d	,		j	o	l	l	y	
D	o	o	d	g	o	o	d	,		j	o	l	l	y		g
E	o	d	g	o	o	d	,		j	o	l	l	y		g	o
F	d	g	o	o	d	,		j	o	l	l	y		g	o	o

2) En este paso está la matriz después del ordenamiento de las cadenas.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		g	o	o	d	g	o	o	d	,		j	o	l	l	y
1		j	o	l	l	y		g	o	o	d	g	o	o	d	,
2	,		j	o	l	l	y		g	o	o	d	g	o	o	d
3	d	,		j	o	l	l	y		g	o	o	d	g	o	o
4	d	g	o	o	d	,		j	o	l	l	y		g	o	o
5	g	o	o	d	,		j	o	l	l	y		g	o	o	d
6	g	o	o	d	g	o	o	d	,		j	o	l	l	y	
7	j	o	l	l	y		g	o	o	d	g	o	o	d	,	
8	l	l	y		g	o	o	d	g	o	o	d	,		j	o
9	l	y		g	o	o	d	g	o	o	d	,		j	o	l
A	o	d	,		j	o	l	l	y		g	o	o	d	g	o
B	o	d	g	o	o	d	,		j	o	l	l	y		g	o
C	o	l	l	y		g	o	o	d	g	o	o	d	,		j
D	o	o	d	,		j	o	l	l	y		g	o	o	d	g
E	o	o	d	g	o	o	d	,		j	o	l	l	y		g
F	y		g	o	o	d	g	o	o	d	,		j	o	l	l

3) La cadena en la última columna y el índice del renglón de la cadena original.

"y,dood oloojggl" , 5

4) Aplicando el codificador *move-to-front*.

121, 45, 102, 114, 0, 1, 36, 0, 1, 113, 1, 0, 112, 110, 0, 3, 5

5) Se convierte a hexadecimal.

79, 2D, 66, 72, 0, 1, 24, 0, 1, 71, 1, 0, 70, 6E, 0, 3, 5

6) Distribución estadística de los caracteres de la cadena de salida.

00: 4; 01: 3; 03: 1; 05: 1; 24: 1; 2D: 1
66: 1; 6E: 1; 70: 1; 71: 1; 72: 1; 79: 1

Se han estudiado diferentes tipos de algoritmos de compresión de datos, que manipulan las secuencias a comprimir de una manera diferente cada uno de ellos. Estos algoritmos resultan idóneos para analizar secuencias genéticas que constan de puros símbolos. En el siguiente capítulo se define otro tipo de algoritmo de compresión *biocompress-2* diseñado particularmente para el análisis de genomas y que formará parte del método propuesto. En este análisis se detectarán y obtendrán ciertas características a partir de la compresión de datos.

Capítulo 5

Análisis de secuencias de DNA

En los capítulos anteriores se han estudiado características de las secuencias de DNA de los organismos (*genomas*), así como la teoría en la que se basa la compresión de datos, la cual se usa como herramienta para el análisis de dichas secuencias de DNA.

El análisis de secuencias de DNA y proteínas es un área que ha tenido un crecimiento significativo en los últimos años. Existen grandes cantidades de información en bases de datos alrededor del mundo y se requiere cada vez más tiempo de procesamiento y mejores algoritmos para su análisis, lo que hace necesaria la utilización de herramientas matemáticas y computacionales, así como para poder encontrar información en las secuencias de DNA que, por medio de otros métodos, sería difícil. Por lo tanto, en este trabajo se propone un método bioinformático que conjunta dichas herramientas, el cual permite obtener información de los genomas de diversos organismos encontrando patrones o segmentos similares en el DNA durante el análisis.

5.1. Eucariontes, bacterias y arqueas

Los seres vivos se dividen actualmente en tres dominios: bacterias (*bacteria*), arqueas (*archaea*) y eucariontes (*eukarya*). En los dominios *archaea* y *bacteria* se incluyen los organismos procariontes, esto es, aquéllos cuyas células no tienen un núcleo celular diferenciado, mientras que en el dominio *eukarya* se incluyen a los animales, hongos, plantas y protistas.

Originalmente, el término *bacteria* se aplicó a todos los microorganismos procariontes. Sin embargo, la filogenia molecular ha podido demostrar que estos microorganismos procariontes se dividen en dos dominios denominados *bacteria* y *archaea* (Woese et al., 1990), que evolucionaron independientemente desde un ancestro común. Estos dominios, junto con el dominio *eukarya*, constituyen la base del sistema de tres dominios, que es actualmente el sistema taxonómico aceptado en biología (Gupta, 2000).

Archaea es similar a *bacteria* en la mayoría de los detalles de la estructura y del metabolismo de la célula pero difiere sobre todo en la composición de los lípidos de la membrana celular y en los aspectos genéticos. Así, la transcripción y traducción genéticas, los dos procesos centrales de la biología molecular, no presentan las características bacterianas típicas sino que son más similares en muchos aspectos a los encontrados en *eukarya*. Por ejemplo, la traducción usa factores de iniciación y de alargamiento del tipo eucarionte y la transcripción implica proteínas de unión como en los eucariontes. Muchos genes ARNr y ARNt de las *archaea* albergan intrones únicos que no se encuentran ni en eucariontes ni en bacterias.

Existen diferencias estructurales en cada uno de los dominios mencionados y esto se debe a la forma en que están distribuidos y organizados los nucleótidos dentro del genoma o secuencia de DNA de cada uno de los organismos. En este trabajo se buscan este tipo de estructuras utilizando los algoritmos de compresión como herramienta.

Estructuralmente se consideran algunos parámetros medibles que se refieren a la composición del genoma en términos de nucleótidos; por ejemplo, el %ACGT, el porcentaje de compresión como una medida de complejidad del genoma; el %C+G y el %A+T que aportan la información de qué tan rígida o flexible puede ser la secuencia, ya que es una clasificación de moléculas a partir de cómo es el enlace entre ellas. De manera semejante, varias características fisicoquímicas del DNA dependen de las interacciones entre las bases consecutivas, por lo que la identificación de patrones de las bases vecinas o más cercanas, dan lugar a una caracterización de la secuencia. La flexibilidad y la rigidez del DNA, dependen de la correlación local o de primeros vecinos de los nucleótidos y de su distribución en la secuencia genética. Así, el DNA es una molécula que tiene su propia organización estructural y puede ser un

factor en la evolución de los organismos de los dominios mencionados a partir de su evolución a nivel molecular. (Miramontes et al., 1995) y (Miramontes y Cocho, 2003)

5.2. Compresión y DNA

Uno de los pioneros en este campo es Eric Rivals (Rivals et al., 1996) quien introdujo conceptos básicos para el estudio y análisis de secuencias de DNA por medio de la compresión.

Como ya se ha mencionado, los algoritmos de compresión permiten calcular una nueva descripción de los objetos que se comprimen para generar una representación más corta. Para alcanzar esta reducción se suprimen algunas regularidades de la descripción original y se sustituyen mediante un código.

Por ejemplo, las repeticiones en el DNA son un tipo de regularidad, un segmento repetido dado en una secuencia del genoma es una regularidad en la secuencia. Puede ser sustituida por un código que indique donde está la ocurrencia anterior del segmento. Entonces se puede explicar que el origen del segmento es una duplicación de otro segmento y proporciona la posición y longitud exacta de la primera ocurrencia. Por lo tanto, dando un algoritmo de compresión y conociendo el tipo de regularidades que codifica, se puede estudiar la presencia de estas regularidades en una secuencia genética; o sea, localizar segmentos regulares y entender porqué son regulares al ver su código.

Se definirá el término de *compresibilidad* de una secuencia de la siguiente manera:

Si la secuencia que se utiliza, tiene regularidades o patrones de repetición secuencial o especular, se dice que es *compresible*.

Por ejemplo, considérese la siguiente secuencia:

$$\ldots\ldots \underbrace{ACG}_{patrón}\underbrace{ACG}\underbrace{ACG}\underbrace{ACG}\underbrace{ACG}\underbrace{ACG}\ldots\ldots$$

en la cual el patrón ACG se repite varias veces. Por lo tanto, la secuencia es

compresible.

Ahora, se considera la siguiente secuencia:

$$\ldots\ldots AAGACGTAGCTGATTGGTCTAGATATGCTAGCCG\ldots\ldots$$

en esta secuencia no hay regularidad alguna con la que se pueda comprimir o reducir en tamaño. Entonces, esta secuencia es *no compresible*.

Una manera de ver estos términos es tomar en cuenta que una secuencia puede tener su información de tal manera que preserve un orden, o de la misma manera que esté en completo desorden.

Dado que los algoritmos de compresión detectan y codifican segmentos regulares dentro de los objetos, en este caso, secuencias de DNA, se tiene que al aplicar un algoritmo de compresión a este tipo de secuencias, detecta qué tan ordenado es el genoma mediante la asociación simple:

$$\begin{array}{rcl} \text{Orden} & \longrightarrow & \text{Compresión alta} \\ \text{Desorden} & \longrightarrow & \text{Compresión baja} \end{array}$$

5.2.1. Biocompress-2

En 1994, se propuso un algoritmo de compresión *lossless* muy específico, llamado *biocompress-2* (Grumbach y Tahi, 1994), diseñado para la compresión de información contenida en las secuencias de DNA, debido a que este tipo de secuencias es muy particular y específico. Este algoritmo está basado en la detección de regularidades, tal como la presencia de palíndromos.

El algoritmo combina los métodos sustitucionales y estadísticos y obtiene una tasa de compresión mayor en el caso de genomas, de tal manera que se consigue una mejor idea de la relación necesaria entre la compresión y comprensión de secuencias genéticas. Este algoritmo es específico ya que nuestras secuencias de DNA (genomas) son textos sobre un alfabeto de cuatro letras $\{A, C, G, T\}$.

Biocompress-2 es una extensión del algoritmo *biocompress* (Grumbach y Tahi, 1993), el cual usa codificación aritmética. Como se dijo, este algoritmo combina un método sustitucional; principalmente se basa en la esencia del método de compresión introducido por J. Ziv y A. Lempel (Ziv y Lempel, 1977) y un método estadístico, la codificación aritmética. Este algoritmo codifica textos de nuestro alfabeto de cuatro letras en una secuencia binaria.

Dentro de los genomas hay redundancias específicas, llamadas *palíndromos* que se obtienen a partir del proceso de plegado del RNA. Tales repeticiones no existen en textos de lenguaje natural y en lenguajes de programación, *biocompress-2* las detecta y codifica.

En el capítulo 2 se describió el proceso de como el DNA codifica proteínas y el papel que tiene el RNA. Asimismo, se explicó que existen varios tipos de moléculas de RNA que llevan a cabo el proceso de codificación de proteínas.

Las moléculas de RNA se pliegan de manera complicada. Su estructura terciaria (i.e., de tres dimensiones) determina sus funciones. Esta estructura está ligada particularmente por el acoplamiento entre las bases complementarias, es decir, entre G - C, A - U. Esto implica que haya pares de subsecuencias complementarias de RNA que tienen que acoplarse. A estos pares de subsecuencias se les llama palíndromos o secuencias inversas. La estructura secundaria del RNA (figura 5.1) es una gráfica en dos dimensiones con los

pares de bases complementarias ligadas. La estructura del RNA es el origen de la abundancia de los palíndromos en el DNA.

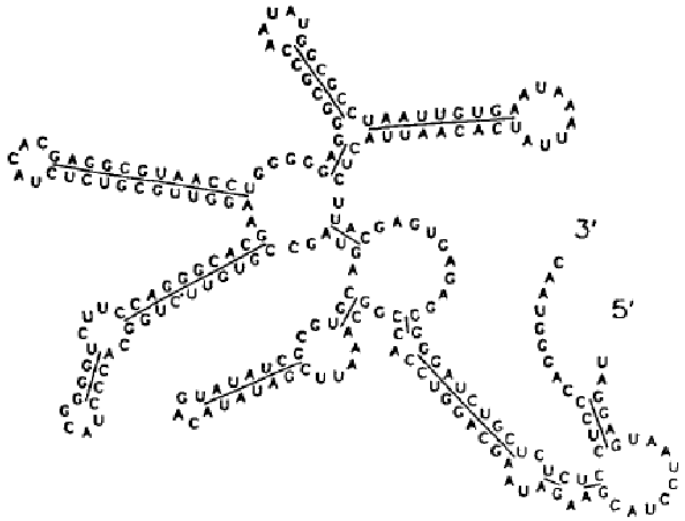


Figura 5.1. Estructura secundaria del RNA.

Las secuencias genéticas están orientadas (capítulo 2). Siempre se leen en la misma dirección, como por ejemplo, en los textos del lenguaje natural o en lenguajes de programación. En el caso de la doble hélice del DNA, las dos hebras son leídas en direcciones opuestas.

Por lo tanto se definen formalmente los palíndromos en el DNA de la siguiente manera:

Se considera una secuencia de caracteres f sobre el alfabeto $\{A, C, G, T\}$:

$$a_1 a_2 \ldots a_n$$

$\bar{f}$ denota la secuencia inversa:

$$a_n a_{n-1} \ldots a_1$$

f' denota la secuencia obtenida de f mapeando cada carácter a su carácter complementario, A es mapeado en T y viceversa, C es mapeado en G y viceversa. Por ejemplo, la secuencia:

$$A\ A\ C\ G\ T\ T$$

es mapeada en la secuencia:

$$T\ T\ G\ C\ A\ A.$$

El par de factores f y $\bar{f}'$ es llamado un *palíndromo*.

Para detectar todos los factores independientemente de su tamaño o posición, el algoritmo *biocompress-2* utiliza una ventana de longitud arbitrariamente grande desde la primera posición en la secuencia, hasta la actual posición. La estructura de datos usada para reconocer los factores es un autómata simple.

En cada paso de la compresión, se elige el factor más largo que comienza en la posición actual y buscando un factor el cual empiece en la parte de la cadena ya codificada. Este factor puede ser un factor idéntico o un palíndromo. Entonces es codificado por un *codeword copia*, es decir, un par de enteros $(l,\ p)$, donde l es la longitud del factor y p la posición de su primera ocurrencia.

Cada carácter A, C, G o T puede ser codificado en dos bits de la siguiente manera; al carácter A le corresponde el "00", a C el "01", a G el "10" y a T el "11". Éstos son *codewords literal*. Puede suceder que el tamaño del *codeword* (l,p) asociado a un factor sea más grande que el tamaño del factor mismo (donde los caracteres son codificados en dos bits). Para evitar la extensión de la secuencia, se codifica un factor f por el par (l,p) solamente si el tamaño del *codeword* no es demasiado grande relativamente al tamaño del factor, es decir,

$$|(l,p)| + c \leq 2l,$$

donde $|(l,p)|$ es el tamaño de la representación de (l,p) y c es una constante proporcional al tamaño de control de bits. Si ningún factor satisface el requisito anterior, el primer carácter se agrega a un *buffer* y la búsqueda se repite con los factores que comienzan en el carácter siguiente.

Este proceso da lugar a una alternancia de secuencias de caracteres sin codificar y de secuencias de pares de números enteros. Las secuencias de

caracteres sin codificar se codifican entonces utilizando la codificación aritmética, cuando el tamaño de la codificación aritmética es más corto que el tamaño de la secuencia misma. De lo contrario, la salida son los *codewords literal* correspondientes.

El algoritmo de codificación aritmética es aplicado repetidamente en cada secuencia de caracteres entre los *codewords copia*. La tabla que contiene las frecuencias se actualiza dinámicamente en cada secuencia y se mantiene para la siguiente secuencia. Por lo tanto, el código de salida contiene tres tipos de *codewords*: caracteres en pares de bits (*codewords literal*), valores numéricos (*codewords aritmético*) y pares de enteros de la forma (*l*,*p*) (*codewords copia*). Enfrente de cada secuencia de *codeword literal*, *codeword aritmético* o *codeword copia*, se añade un *codeword control* indicando el cambio de semánticas de la codificación. Esto permite distinguir entre los tres tipos de secuencias. Los *codewords control* son enteros especificando el número de de caracteres (*codewords literal*), bytes (*codewords aritméticos*) o pares de enteros (*codewords copia*) respectivamente.

Por ejemplo, la siguiente secuencia:

$$AACTGTTGTTGTTAGAACTGTTGTT$$

se traduce en la expresión:

$$6AACTGT1(7,4)2AG1(10,1)$$

La eficiencia de la compresión depende del espacio requerido para la codificación de los enteros. Para los *codewords control*, se reserva un espacio variable. Esto es porque el número de caracteres no codificados por el método sustitucional, varía entre uno y algunos miles de caracteres para genomas grandes. Para codificar tales enteros, se utilizó su codificación en binario. Existen otras codificaciones para los enteros, como por ejemplo, la codificación de Fibonacci (Apostolico y Fraenkel, 1987) que es muy utilizada, pero para los fines de este trabajo, basta con la codificación binaria de cada entero.

Este algoritmo se implementó y en la siguiente sección se muestra su funcionamiento y desempeño en comparación con algunos de los métodos de compresión mencionados en el capítulo anterior.

El algoritmo *biocompress-2* se muestra a continuación:

```
begin
i=1;
bool = F;
while (i ≤ n)
    do begin

        encontrar el factor mas largo f
            empezando en una posición menor que i
            comparar el factor en la actual posición i
        sea l1 la longitud y p1 la posición de f

        encontrar el palíndromo mas largo f̄'
            empezando en una posición menor que i
            comparar el factor en la actual posición i
        sea l2 la longitud y p2 la posición de f̄'

        if |(l1,p1)| + c >2l1 then
            if |(l2,p2)| + c >2l2 then
                salida, el caracter en la posición i en el buffer
                bool = T;
                i = i +1;
            else
                if bool = T then
                    salida, codificación aritmética o literal del buffer;
                salida, el codeword (l2,p2);
                vaciar buffer;
                bool = F;
                i = i + l2;
        else
            if bool = T then
                salida, codificación aritmética o literal del buffer;
            vaciar buffer;
            bool = F;

            if (|l1| >|l2| ó |(l2,p2)| + c >2l2 then
                salida, el codeword (l1,p1);
                i = i + l1;
            else
                salida, el codeword (l2,p2);
                i = i + l2;
    end
end
```

Algoritmo *biocompress-2*

5.3. Análisis de genomas basado en técnicas de compresión de datos

Muchas secuencias de DNA contienen patrones repetitivos. Se utilizarán las técnicas de compresión de datos para la detección de tales patrones o regiones en las que se repitan. Esta detección es una parte importante del análisis de secuencias de DNA.

Para empezar con la descripción, fue necesario obtener las secuencias de DNA o genomas en su formato *FASTA*[1] de la base de datos del "National Center for Biotechnology Information" (NCBI) (Entrez, 2009); en el caso de los organismos pertenecientes a los eucariontes se obtienen cromosomas de su genoma. Estas secuencias se guardaron como un archivo de texto sin la primer línea descriptiva, sin espacios ni saltos de línea. Cabe recordar que los genomas de los organismos son una secuencia de letras que pertenecen al alfabeto A, C, G y T, que son los cuatro nucleótidos.

```
>gi|30698537|ref|NC\_003074.4| Arabidopsis thaliana chromosome 3, complete sequence
ACGCCGGCCAGTGAATTGTAATACGACTCACTATAGGGCGAATTGGGCCCTCTAGATGCATGCTCGAGCG
GCCGCCAGTGTGATGGATATCTGCAGAATTCGCCCTTCCCTAAACCCTAAACCCTAAACCCTAAACCCTA
AACCCTAAACCCTAAACCCTAAATCCATAAATCCCTAAAACCATAATCCTAAATCCCTTAATTCCTAAAT
CCCTAATACTTAGACCCTAATCTTTAGTTCCTAGACCCTAATCTTTAGTTCCTAGACCCTAAATCCATAA
TCCTTAATTCCTAAATTCCTAAATCCCTAATGCTAAATCTCTAAATCCCTAGCAATTTTCAAGTTTTGCT
TGATTGTTGTAGGATGGTCCTTTCTCTTGTTTCTTCTCTGTGTTGTTGAGATTAGTTTGTTTAGGTTTGA
TAGCGTTGATTTTGGCCTGCGTTTGGTGACTCATATGGTTTGATTGGAGTTTGTTTCTGGGTTTTATGGT
TTTGGTTGAAGCGACATTTTTTTGTGGAATATGGTTTTTGCAAAATATTTTGTTCCGGATGAGTAATATC
TACGGTGCTGCTGTGAGAATTATGCTATTGTTTTGCAGGTCCTGTTCTTAAT.................
```

Ejemplo de un archivo *fasta*.

Los genomas de los organismos que se utilizaron para este análisis corresponden a los que se encuentran en el Cuadro 5.1.

Este método para analizar genomas a través de medidas de compresión de datos no considera la totalidad del genoma ya que el genoma no es homogéneo y no se puede analizar en su totalidad sin considerar que éste cambia su

[1] El formato FASTA, es un formato basado en texto para representar las secuencias de ácidos nucleicos o las secuencias de proteína.

Organismo	Especie
Homo Sapiens cromosoma 21	eucarionte
Arabidopsis thaliana cromosoma 3	eucarionte
Drosophila melanogaster cromosoma x	eucarionte
Caenorhabditis elegans cromosoma 1	eucarionte
Saccharomyces cerevisiae cromosoma 14	eucarionte
Methanococcus jannaschii	arqueobacteria
Sulfolobus tokodaii	arqueobacteria
Mycoplasma gallisepticum	bacteria
Mycoplasma pulmonis	bacteria
Bacillus Subtilis	bacteria
Chlamydia trachomatis	bacteria
Escherichia Coli	bacteria

Cuadro 5.1. Genomas de organismos

estructura a través del espacio. Por lo tanto, la compresión de las secuencias de DNA se hace para intervalos relativamente pequeños respecto a la cadena total de un genoma.

Para obtener los intervalos del genoma que se someterán a compresión se divide la cadena total de DNA en *ventanas* iguales de longitud n hasta recorrer todo el genoma de la siguiente manera:

$$\underbrace{ACGTCGCGTA\ldots\ldots}_{n}\underbrace{ATCGTAGGCT\ldots\ldots}_{n}GCTTACG\ldots$$

Sea n el tamaño de la ventana. Para obtener cada una de las ventanas se empieza desde el primer elemento del genoma de izquierda a derecha y se consideran los primeros n elementos. Después se seleccionan los siguientes n elementos a partir del elemento $n+1$. El siguiente paso es obtener los siguientes n elementos a partir del elemento $2n+1$ y así sucesivamente hasta que se terminan los elementos del genoma.

En caso de que la longitud del genoma no sea múltiplo del tamaño de las ventanas, entonces la última subsecuencia del genoma cuyo tamaño será menor al de la ventana no se considera y no se comprime.

Para el análisis en este trabajo se consideraron las ventanas sobre el genoma con traslape, esto es, no empezando en la primera letra de la que consta el genoma, sino que la ventana se definió a partir de un cierto número de letras a partir de la primera. Por ejemplo:

$$ACGT\underbrace{CGCGTA\ldots\ldots ATCG}_{n}\underbrace{TAGGCT\ldots\ldots GCTT}_{n}ACG\ldots$$

Sin embargo, al aplicar la compresión con este tipo de ventana con traslape, así como al utilizar la que empieza con la primera letra del genoma, los resultados obtenidos no cambiaron significativamente, por lo que es lo mismo considerar cualquiera de estos dos tipos de ventana para este análisis. Aquí se consideraron las ventanas sin traslape para recorrer el genoma desde el inicio.

Cada ventana se comprime con algunas técnicas de compresión de datos mencionada en el capítulo anterior. Para este trabajo se utilizaron los siguientes programas:

- *gzip* - programa basado en el algoritmo de codificación Lempel Ziv (LZ77).
- *bzip2* - programa basado en los algoritmos de compresión Burrows Wheeler y codificación de Huffman.
- *biocompress-2* - programa basado en los algoritmos de codificación Lempel Ziv (LZ77) y codificación aritmética.

para la compresión de cada una de las ventanas de tamaño n a lo largo de los genomas y así, poder calcular su tasa de compresión.

Las versiones de los programas *gzip* y *bzip2* son las versiones de software libre, dentro de las distribuciones de *linux*. El programa *biocompress-2* es una implementación diseñada para este trabajo en el manejo y análisis de las ventanas de compresión. Se implementó en el lenguaje C++ en el sistema operativo linux, siendo este programa, el que obtiene una mejor tasa de compresión para el análisis de secuencias de genomas o secuencias de letras específicas como lo son las secuencias de ADN {A,C,G,T}. Este resultado se verá a continuación en las gráficas obtenidas en este análisis con ventanas sobre los diversos genomas y con diferentes tamaños de ventanas.

Como se ha mencionado, los genomas son secuencias de letras que pueden llegar a miles o millones de letras, por ejemplo, se supone que el genoma de algún organismo consiste en una cadena de 1,000,000 de letras, entonces se puede dividir esa cadena en 1,000 intervalos o ventanas de tamaño 1,000 y aplicarle el algoritmo de compresión a cada ventana obteniendo como resultado una serie de 1,000 tasas de compresión, es decir, una tasa de compresión por cada ventana.

Cada *tasa de compresión* se calcula de la siguiente manera:

$$compresión = \frac{tamaño\ original - tamaño\ codificado}{tamaño\ original}$$

donde este valor es una proporción de compresión y al multiplicarlo por 100 se obtiene el porcentaje de compresión.

Así, para cada ventana de tamaño fijo se calcula su tasa de compresión; estos porcentajes se almacenan uno por uno en un archivo hasta recorrer todo el genoma y obtener las tasas de todas las ventanas de longitud n de las que consta el genoma.

Debido a que la longitud de los genomas completos de los distintos organismos varía mucho, existirán genomas en los que se tenga un número menor

o mayor de ventanas. Como en el caso, por ejemplo, del genoma del virus bacteriófago[2] que tiene aproximadamente $5x10^3$ pares de bases a diferencia del tamaño del genoma del *Homo sapiens* de aproximadamente $3x10^9$ pares de bases.

Las series de las tasas de compresión se almacenan en un archivo para cada organismo y para cada ventana de diferente longitud; éstas a su vez son graficadas como se verá mas adelante.

En este análisis de genomas se han variado los siguientes parámetros externos:

- La cadena de DNA (genoma de un organismo)
- El algoritmo de compresión
- El tamaño de cada ventana o intervalo dentro del genoma.

El tamaño de cada ventana se varió entre 1,000 y 10,000 para diferentes genomas.

De acuerdo con el análisis que se realizó, se encontró que la gráfica de tasas de compresión para genomas iguales, con tamaños de ventanas iguales, pero con algoritmos de compresión diferentes, son cualitativamente iguales. Anteriormente se hizo notar que el mejor algoritmo de éstos es el *biocompress-2* pues las tasas de compresión resultaron ser mejores y más altas que las obtenidas con *gzip* y *bzip2*. Sin embargo, esta mejora no se vió reflejada en las cualidades de las gráficas.

A continuación se muestran los resultados de las gráficas para diferentes genomas de organismos a los cuales se les aplicó el análisis. Como se verá, en los siguientes casos, se ejecutó el análisis variando el tamaño de los intervalos en que se divide el genoma notándose que mantiene cierta estructura en la gráfica y mostrando que hay zonas en las que el genoma tiene ciertas regularidades que pueden indicar la existencia de alguna estructura dentro del genoma.

[2]Virus que infecta exclusivamente a bacterias.

Las gráficas representan los porcentajes de compresión de cada intervalo en que se divide el genoma, el *eje X* corresponde al *número de ventana* en que se dividió el genoma y el *eje Y* corresponde a la *tasa de compresión* de cada una de esas ventanas.

El primer organismo al que se le aplicó el análisis fue a la planta *Arabidopsis thaliana* y se analizó su cromosoma 3.

En la figura 5.2 se muestra la gráfica de la serie que contiene los tasas de compresión para el cromosoma 3 del genoma de la *Arabidopsis thaliana* con tamaño de la ventana 10,000 y con el programa de compresión *gzip*. Como se ve en la gráfica, la mayor parte de los porcentajes de compresión está entre el 68 % y el 70 %, pero hay ciertas ventanas en las que se eleva mucho el porcentaje sobresaliendo de los demás e indicando la presencia de cierta estructura dentro de la secuencia.

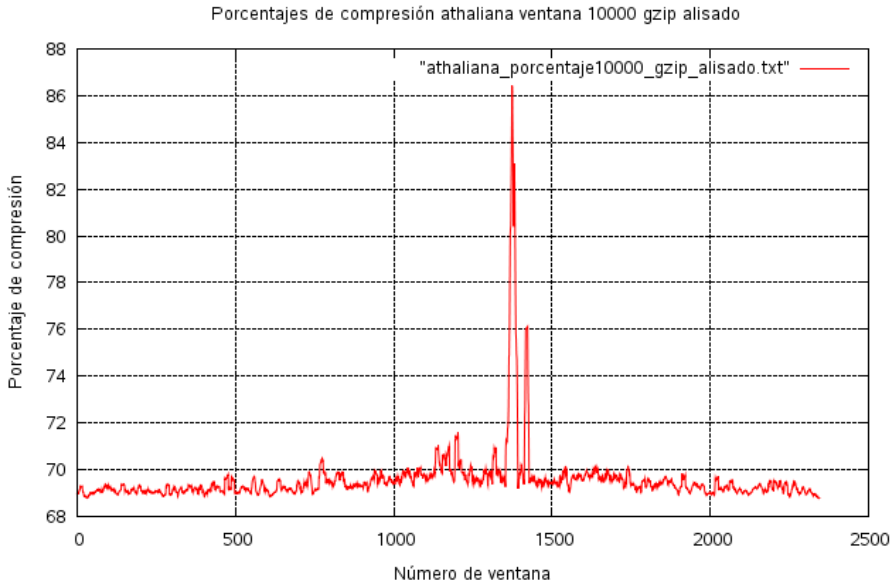


Figura 5.2. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, con compresión *gzip*.

En la figura 5.3, únicamente se cambió el algoritmo de compresión a *bzip2*, manteniendo el cromosoma 3 del genoma de la *Arabidopsis thaliana* y el tamaño de la ventana (10,000). Con este algoritmo de compresión, mejoró la tasa de compresión pero se mantuvo el patrón de la gráfica con respecto al obtenido con el *gzip*.

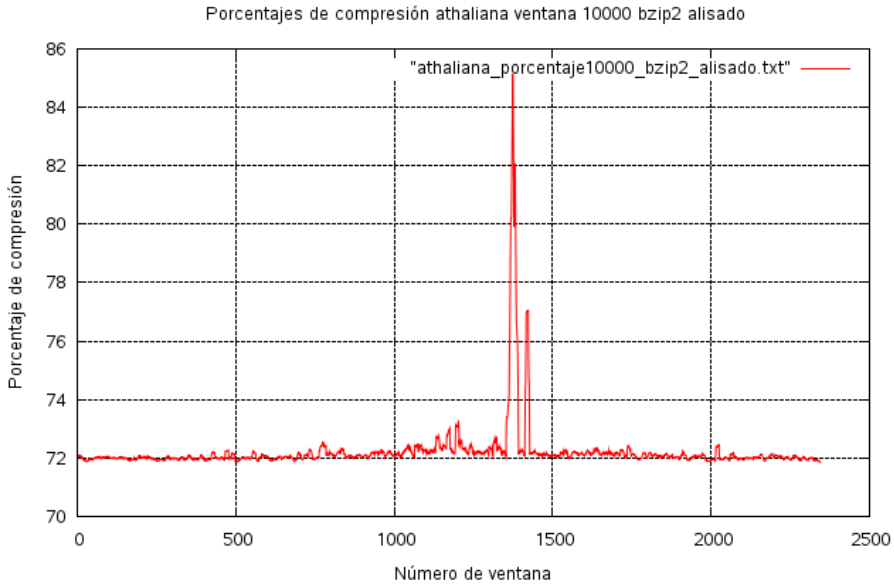


Figura 5.3. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, con compresión *bzip2*.

En la figura 5.4, nuevamente se ejecutó el análisis cambiando el algoritmo de compresión al *biocompress-2*, con el mismo genoma y el mismo tamaño de la ventana, y también se mejoraron los porcentajes de compresión, oscilan por el 74 % la mayoría de ellos, pero sin modificar cualitativamente el patrón de la gráfica.

Ahora, en la figura 5.5, se muestran las tres gráficas juntas de cada uno de los algoritmos de compresión (*gzip*, *bzip2* y *biocompress-2*) utilizados para este tamaño de ventana fijo (10,000). Como se puede apreciar, el *biocompress-2* alcanza mejores tasas de compresión que los otros dos, pero la gráfica mantiene su estructura con cualquiera de los algoritmos.

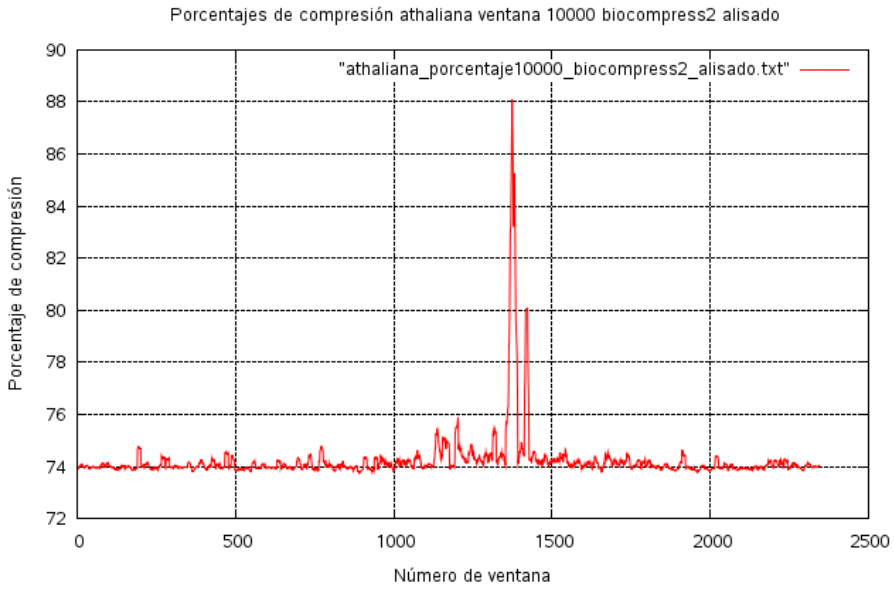


Figura 5.4. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, compresión con *biocompress-2*.

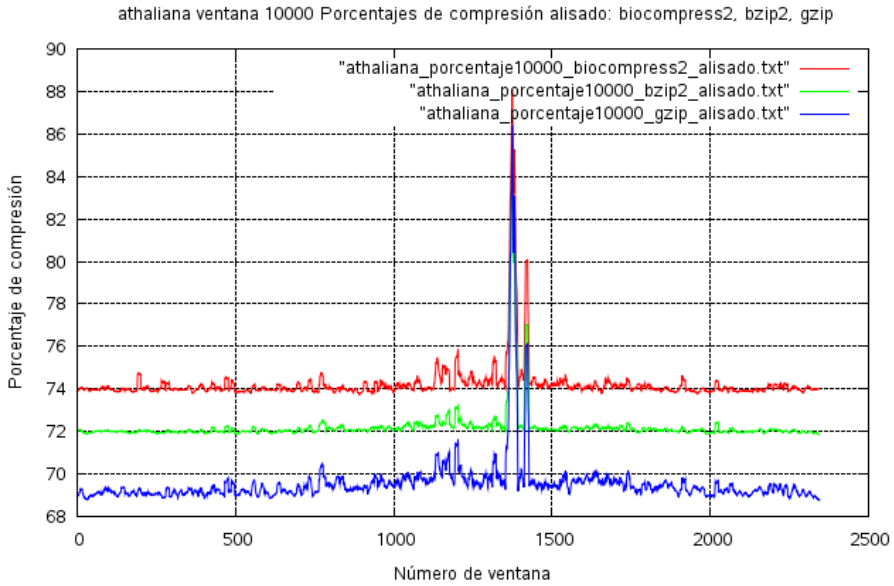


Figura 5.5. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

A diferencia de la figura anterior, en la figura 5.6 se fijó el tamaño de la ventana en 5,000 para recorrer el cromosoma 3 del genoma de la *Arabidopsis thaliana*. Se puede observar que nuevamente el *biocompress-2* obtiene mejores porcentajes, pero se mantiene el patrón en la gráfica, para los tres algoritmos de compresión.

De las figuras 5.5 y 5.6 se muestra que el genoma de la *Arabidopsis thaliana* contiene zonas en las que tiene mayor compresibilidad, esto es, en donde el porcentaje de compresión se eleva. Como se ve en las gráficas, estas zonas de mayor compresibilidad coinciden al variar el programa de compresión, por lo que independientemente del algoritmo que se use, en esa parte de la secuencia de DNA del genoma existe alguna estructura más ordenada que en el resto y estos algoritmos de compresión son capaces de detectarla. En el capítulo anterior, se mencionó que cada uno de estos programas se basan en diferentes algoritmos o métodos de compresión dentro del esquema *lossless*. En el caso del *biocompress-2*, esta compresibilidad se debe a que en esta zona del genoma encuentra palíndromos, pero también encuentra redundancia de

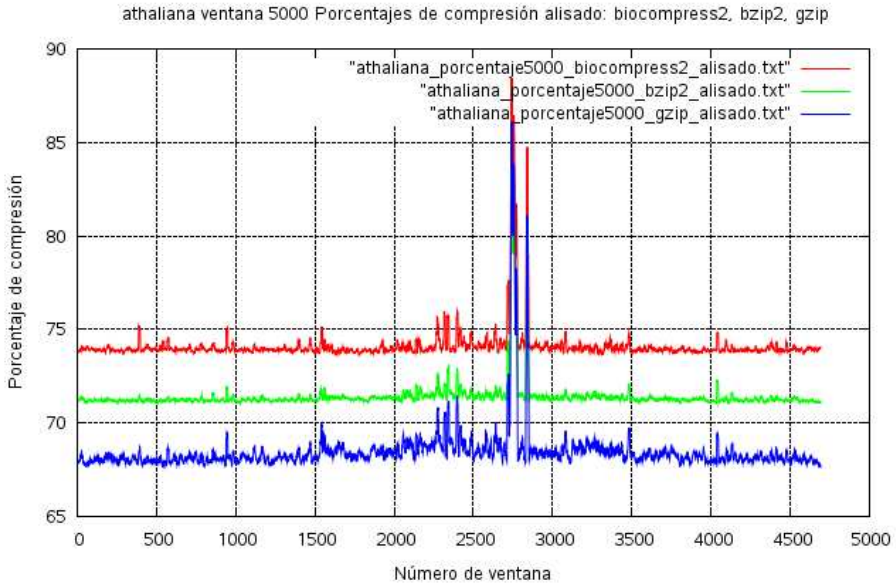


Figura 5.6. Porcentajes de compresión para el cromosoma 3 del genoma de *A. thaliana* con *intervalos* de 5,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

información dentro del genoma.

La obtención de mejores tasas de compresión con el programa *biocompress-2* se debe a que este algoritmo de compresión está diseñado para la compresión de secuencias sobre alfabetos de 4 letras {A,C,G,T}.

Para el estudió del *Homo sapiens* se consideró el cromosoma 21, uno de los más pequeños de todo el genoma. Los resultados del análisis con los algoritmos de compresión se muestran a continuación.

Se eligen ventanas de tamaño 10,000 y 5,000 para cada uno de los programas de compresión utilizados en este método.

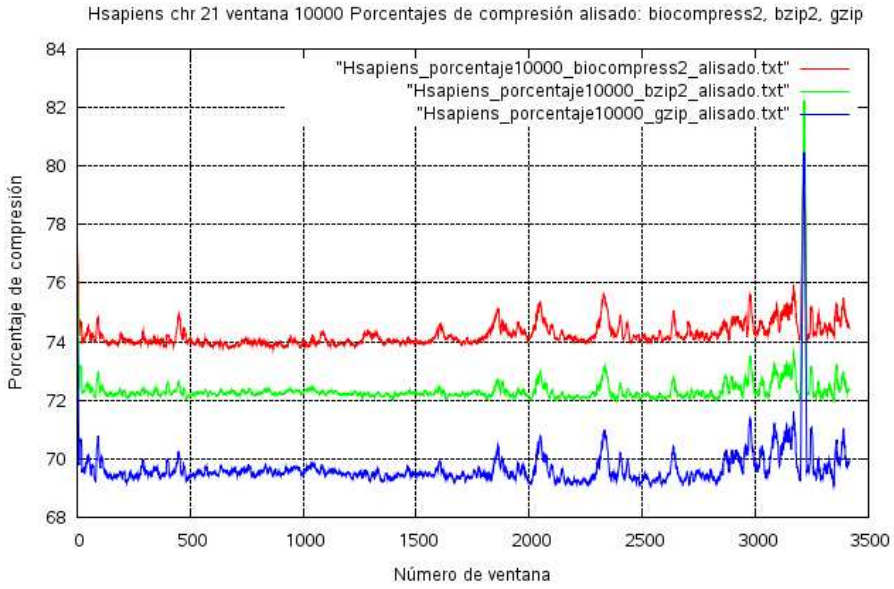


Figura 5.7. Porcentajes de compresión para el cromosoma 21 del genoma del *H. sapiens* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

De la misma manera que en el análisis para el genoma de la planta *A. thaliana*, en el cromosoma 21 del *Homo sapiens*, existe una zona en la que se nota una mayor compresibilidad hacia el final del genoma independientemente del programa de compresión que se haya utilizado, esto se observa en

las figuras 5.7 y 5.8.

Las figuras 5.7 y 5.8, corresponden al tamaño de ventana 10,000 y 5,000 respectivamente en las que se muestran los tres programas de compresión utilizados y en los que nuevamente el programa *biocompress-2* obtiene mejores tasas de compresión.

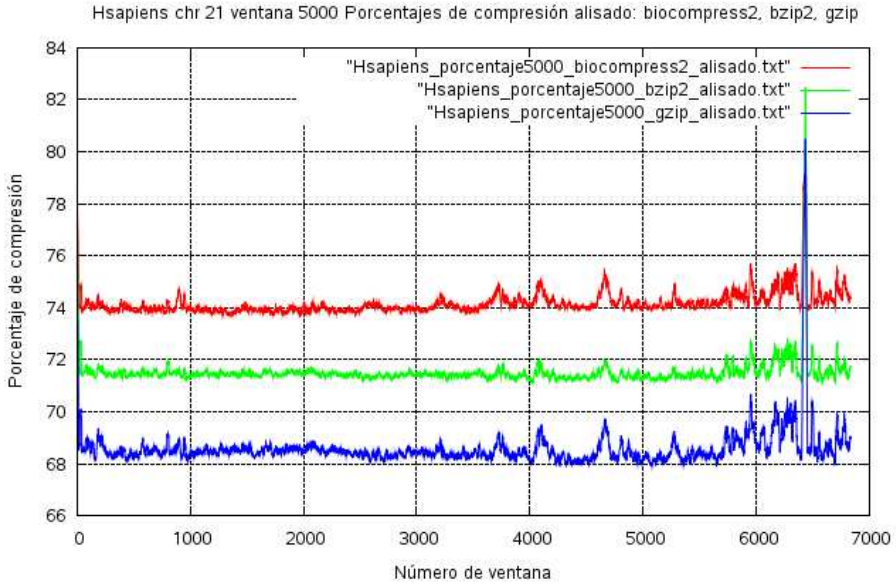


Figura 5.8. Porcentajes de compresión para el cromosoma 21 del genoma del *H. sapiens* con *intervalos* de 5,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

Como se ha mostrado con estos organismos hasta el momento, se observa que en algunas zonas de los genomas existen subsecuencias con un valor de tasa de compresión mucho más alto que en el resto del genoma. En esta zona de mayor compresión, existe cierta redundancia en la secuencia capaz de localizarse independientemente del algoritmo de compresión que se utilice.

Otro organismo perteneciente al dominio de los eucariontes es la *Drosophila melanogaster* también conocida como la mosca de la fruta. Para este organismo se analizó el cromosoma *X*.

Para esta secuencia genética de la misma manera se hizo el análisis para ventanas de tamaño 10,000 y 5,000, en las cuales se tiene una estructura diferente a los demás eucariontes analizados.

En la figura 5.9 la estructura del genoma muestra un poco más de variedad, siendo que tiene más zonas que alcanzan una tasa de compresión mayor a la que predomina a lo largo del genoma.

De igual forma en la figura 5.10 se observa la misma estructura que cuando se analizó para las vantanas de tamaño 10,000 para el genoma de la mosquita, por lo que tampoco depende del número de ventanas que se consideren para cambiar la estructura al comprimir. Esto pasa para todos los genomas analizados.

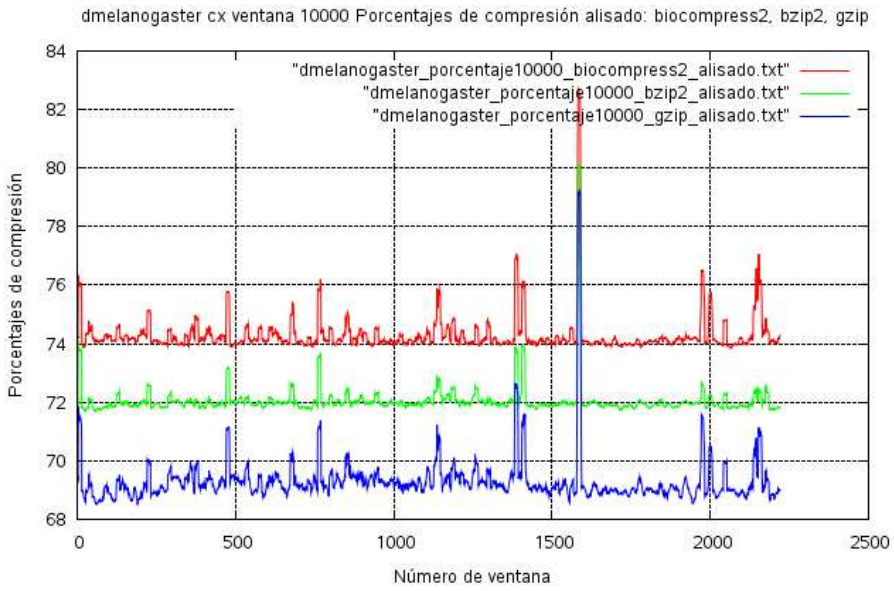


Figura 5.9. Porcentajes de compresión para el cromosoma x de la *D. melanogaster* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

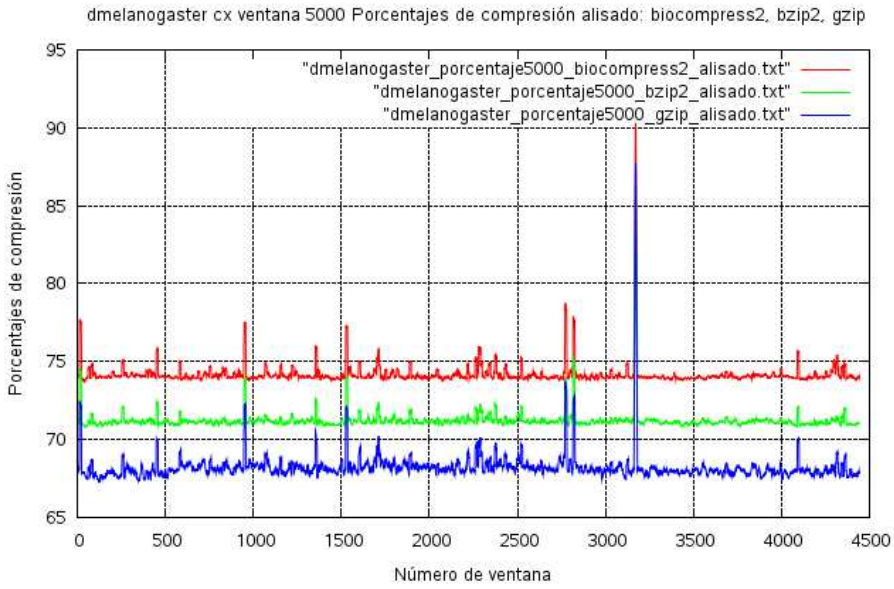


Figura 5.10. Porcentajes de compresión para el cromosoma x de la *D. melanogaster* con *intervalos* de 5,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

Otro organismo eucarionte es el gusano *Caenorhabditis elegans* para el cual se analizó su cromosoma 1.

En la figura 5.11 se muestra el análisis para el cromosoma 1 del genoma del gusano *Caenorhabditis elegans* en el que aparecen zonas con alta compresión a lo largo de la secuencia, dando información acerca de que existe alta periodicidad en estos segmentos, En este caso, existen más zonas con tasas altas de compresión que los anteriores eucariontes. Nuevamente los algoritmos de compresión detectan las mismas zonas en las que tiene patrones característicos para esta secuencia genética solamente que con diferentes porcentajes de compresión. A diferencia de los anteriores organismos, el *C. elegans* tiene zonas de mayor compresión con respecto al promedio que se obtiene a lo largo de su cromosoma. Para este genoma se presenta la gráfica utilizando una ventana de tamaño 10,000.

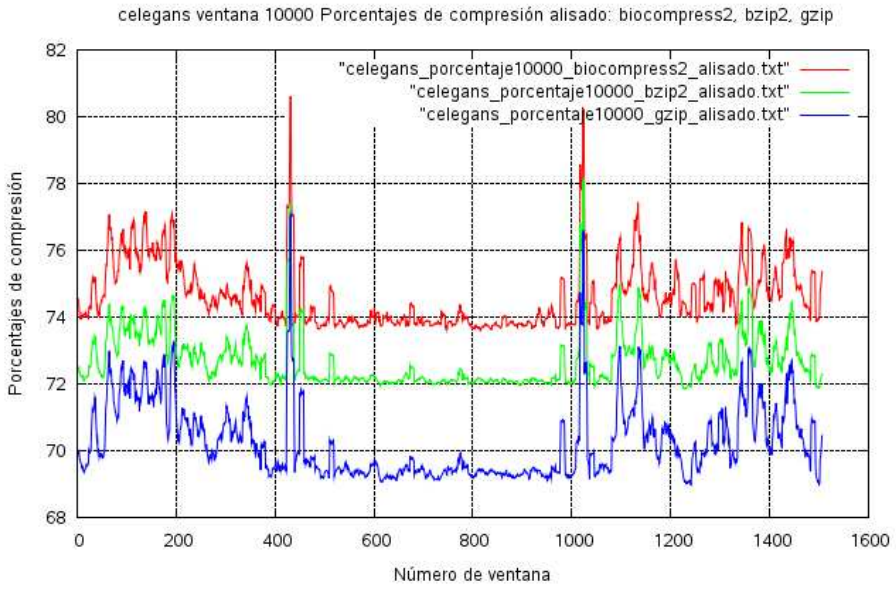


Figura 5.11. Porcentajes de compresión para el cromosoma 1 del genoma de *C. elegans* con *intervalos* de 10,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

El análisis se ha realizado con genomas correspondientes a organismos *eucariontes*, por lo que a continuación se presentarán algunos que pertenecen a los *procariontes* (bacterias y arqueobacterias).

El organismo *Mycoplasma pulmonis* es una bacteria responsable de enfermedades respiratorias en los ratones, sin embargo es interesante el análisis con esta bacteria

Para el caso del *Mycoplasma pulmonis*, perteneciente al dominio de las bacterias, es muy evidente la zona en la que el porcentaje de compresión es alto con respecto a las demás ventanas, dando una estructura al genoma muy particular. Por lo que estas ventanas con altas tasas de compresión dentro del genoma de este organismo se diferencian en su secuencia y en la que debe existir alguna otra función dentro del organismo (figuras 5.12 y 5.13).

El tamaño de las ventanas a comprimir para este genoma se modificó,

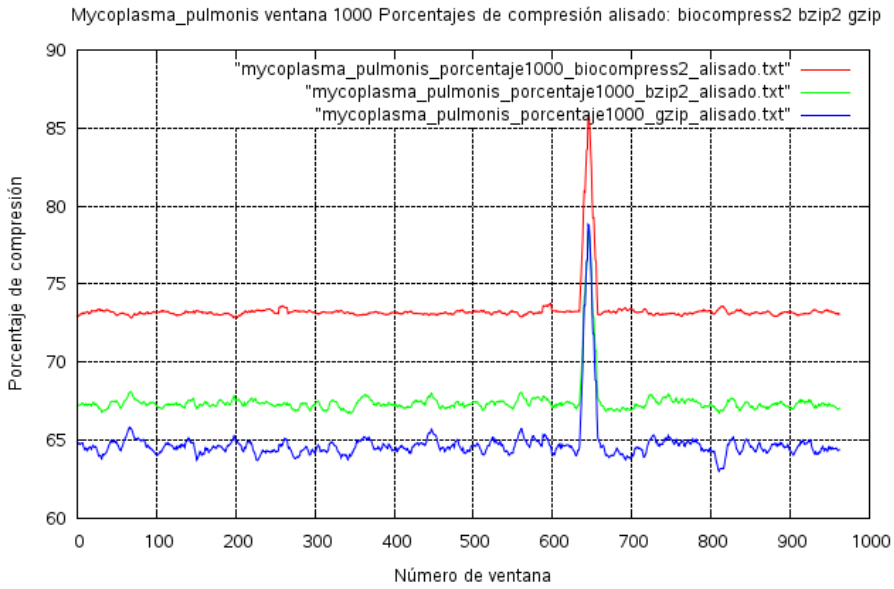


Figura 5.12. Porcentajes de compresión para el genoma del *Mycoplasma pulmonis* con *intervalos* de 1,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

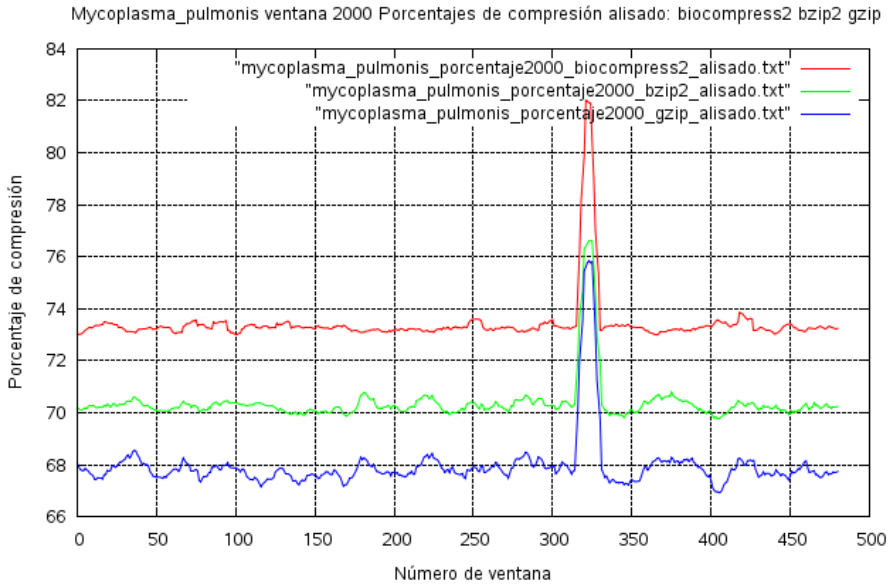
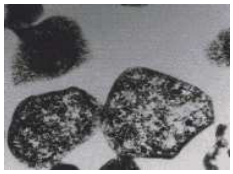


Figura 5.13. Porcentajes de compresión para el genoma del *Mycoplasma pulmonis* con *intervalos* de 2,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

fijando cada una de ellas en 1,000 y 2,000, debido a que el tamaño del genoma total es menor en comparación de los genomas de los eucariontes, pero es suficiente para seguir obteniendo características importantes en el análisis.

Nuevamente se muestra que el mejor compresor es el *biocompress-2*, implementado para este trabajo y junto con los otros compresores *bzip2* y *gzip*, mantienen el patrón de la gráfica.

Con los resultados obtenidos de algunos genomas analizados, es posible señalar que al encontrar zonas en las que los porcentajes de compresión son altos, se demuestra que en esa zona tienen redundancia de información y eso hace que se encuentren alejados del *azar*, esto implica que las secuencias de DNA no sean en su totalidad como se pudiera pensar, una secuencia totalmente azarosa.



El siguiente organismo para analizar es *Sulfolobus tokodaii* que pertenece al dominio de las arqueobacterias proveniente de los manantiales de Japón y su crecimiento es óptimo en ambientes de temperatura de 80^{o}C.

La estructura de las gráficas obtenidas al aplicar los compresores a este organismo, tiene un patrón diferente a los que se han analizado ya que las regiones o subsecuencias del genoma donde se encuentran los porcentajes de compresión mas altos, precisamente se encuentran al inicio del genoma y al final de éste.

Por ser una arqueobacteria el genoma de *Sulfolobus todokaii*, su longitud también es pequeño a comparación de los eucariontes y por lo tanto se consideran 1,000 y 2,000 el tamaño de las ventanas con que se recorre el genoma para obtener los porcentajes de compresión, sin embargo a pesar de los algoritmos de compresión se sigue manteniendo la estructura cualitativa de la gráfica (figura 5.14 y 5.15).

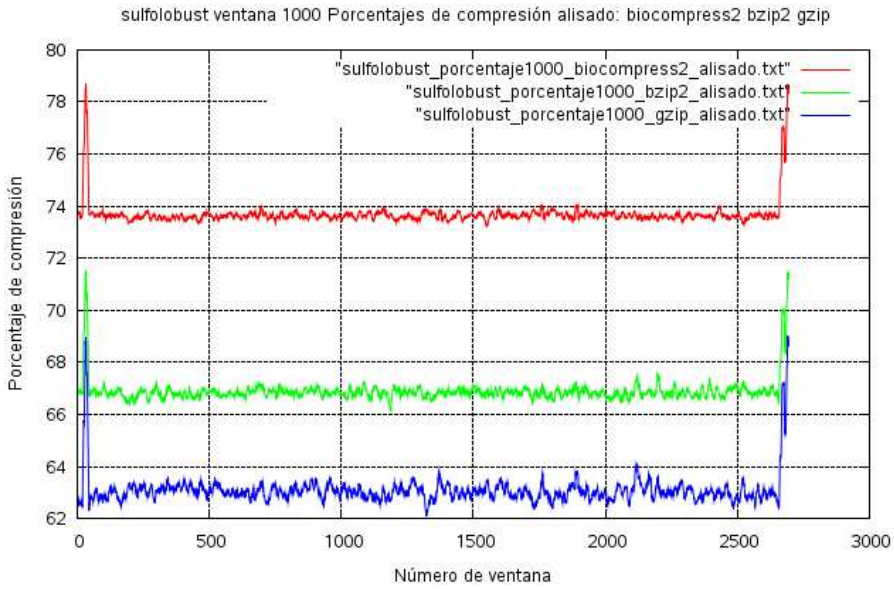


Figura 5.14. Porcentajes de compresión para el genoma de *Sulfolobus tokodaii* con *intervalos* de 1,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

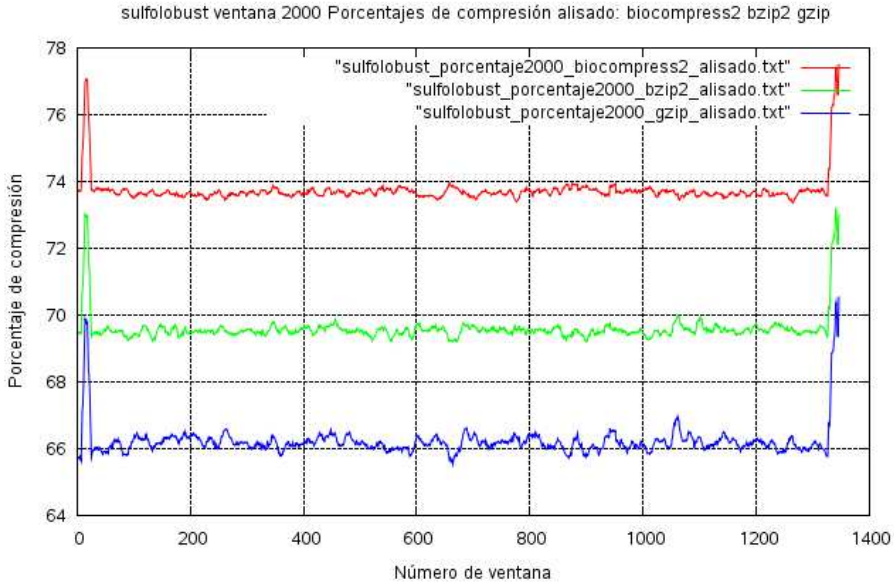


Figura 5.15. Porcentajes de compresión para el genoma de *Sulfolobus tokodaii* con *intervalos* de 2,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

Este análisis se ha hecho para organismos dentro de los dominios de los seres vivos pero que pasa cuando se genera una secuencia pseudo-aleatoria que simule una secuencia genética, esto es que contenga los cuatro nucleótidos del DNA.

La figura 5.16 muestra la diferencia de utilizar el genoma de un organismo, a utilizar una secuencia pseudo-aleatoria en la que no existe algún patrón dentro de la gráfica de porcentajes de compresión. Se observa que es casi constante todo el genoma y la secuencia no tiene alguna estructura que pueda comprimirse más que el resto del genoma. Por lo que no tiene zonas en las que pueda haber algún tipo de redundancia como es el caso de las secuencias genéticas de organismos. Esto hace posible señalar que los genomas de los organismos, no sean completamente azarosos, es decir, que existe cierta redundancia o repetición de información dentro de las subsecuencias.

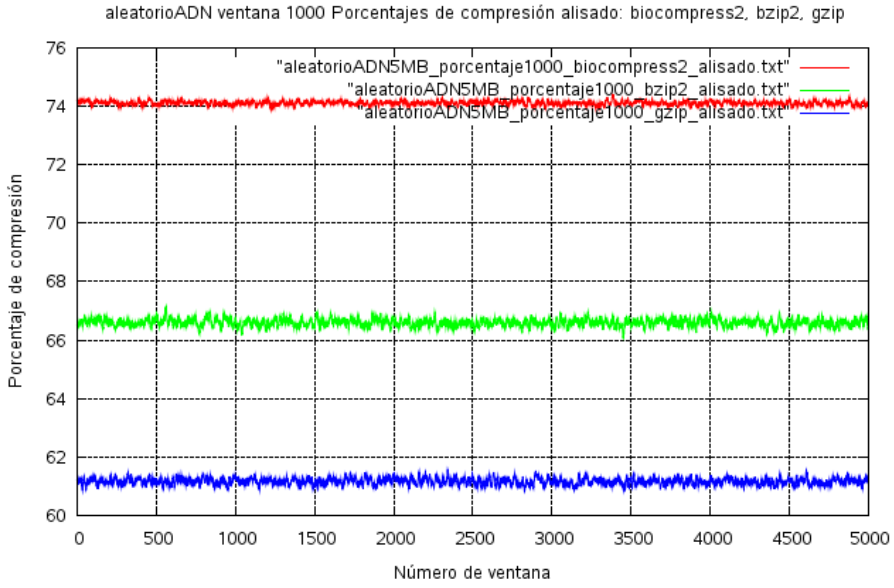


Figura 5.16. Porcentajes de compresión *secuencia pseudo-aleatoria* con *intervalos* de 1,000 bases, con *biocompress-2*, *bzip2* y *gzip*.

Capítulo 6

Resultados y discusión

Al aplicar este método para el análisis de genomas, se obtienen resultados que permiten observar características estructurales dentro de las secuencias de DNA analizadas.

En el análisis aplicado al cromosoma 21 del genoma del *Homo sapiens* se obtiene que al final de la secuencia, existe un porcentaje de compresión más alto en comparación con el que se tiene a lo largo del resto del cromosoma. Si se considera que este cromosoma está compuesto de poco más de 34 millones de nucleótidos, la zona donde existe mayor compresión ocurre en las últimas 6 millones de bases aproximadamente.

Analizando esa zona de la secuencia en particular, se obtiene que el porcentaje del número de C y G, $\%(C+G)$, por ventana en esta zona, está por encima del promedio de este porcentaje a lo largo del cromosoma (Li y Miramontes, 2006); entonces, si los algoritmos de compresión comprimen más, es porque existe más estructura en estas secuencias y posiblemente haya una periodicidad estructural en esta zona, por lo tanto estas subsecuencias donde aumenta el porcentaje son más compresibles. Al ser C y G los nucleótidos que presentan triple enlace en la doble hélice, se esperaría que en esta parte sea más rígida la molécula del DNA (figura 6.1).

Otro resultado que se obtuvo con este método es el que aparece cuando se analiza el genoma de la bacteria *Mycoplasma pulmonis*, el cual tiene un poco menos de un millón de nucleótidos. Los resultados mostrados anteriormente para este organismo muestran que existe un conjunto de secuencias

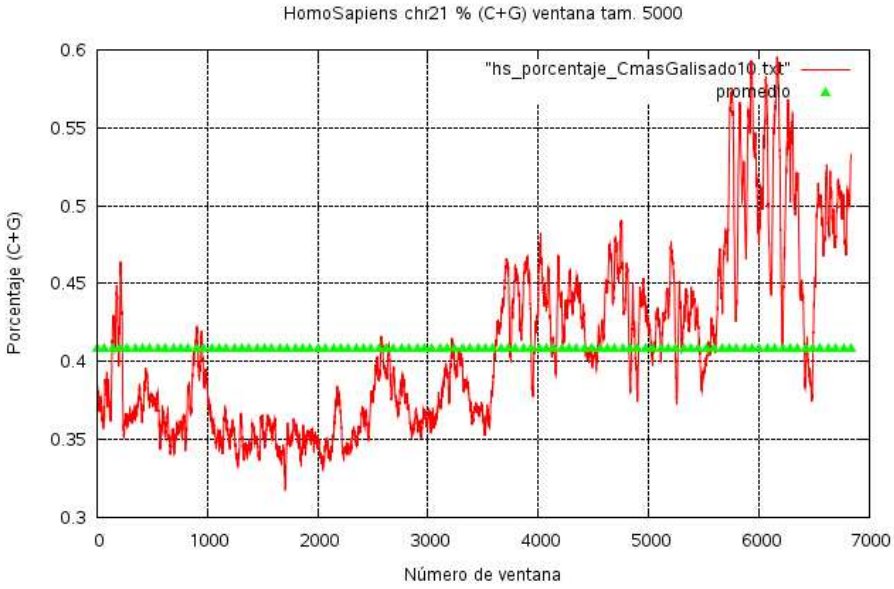


Figura 6.1. Gráfica que muestra el promedio del valor del %(C+G) del cromosoma 21 del *Homo sapiens* como una línea constante y al final se observa la zona en la que este porcentaje está por encima del promedio.

muy localizado en las que el porcentaje de compresión es más elevado que en el resto del genoma. Resulta interesante ya que es una zona relativamente pequeña donde este porcentaje se incrementa siendo muy notorio el cambio de la tasa de compresión.

Particularmente este método permite obtener las secuencias o segmentos que sobrepasan algún umbral del valor de compresión; esto es, en la gráfica de las tasas de compresión de los organismos los picos más altos que se encuentran en las zonas en que se comprime más la secuencia y así se procede a analizar por separado cada secuencia obtenida en este proceso.

Estas secuencias se analizaron comparando su contenido con algunos resultados obtenidos de un banco de genes. Al obtener las secuencias donde este porcentaje es alto y analizarlas en la base de datos de *BLAST*(Blast, 2009) resultó que exactamente en esa zonas de mayor compresión existe un tándem

de genes *vsa*[1] presente en este genoma (Shen et al., 2000). Cabe resaltar que no es fácil encontrar estos genes expresados en esta forma en algún otro genoma, pero implica que existe cierta periodicidad en esta parte del genoma y que los algoritmos de compresión los detectaron de una manera satisfactoria, demostrando que son una buena herramienta para detectar características estructurales en las secuencias de DNA. Este resultado se compara con lo reportado en (Chambaud et al., 2001) en el que se detecta estos mismos *vsa* genes por medio de otro método presentado en ese trabajo (figura 6.2).

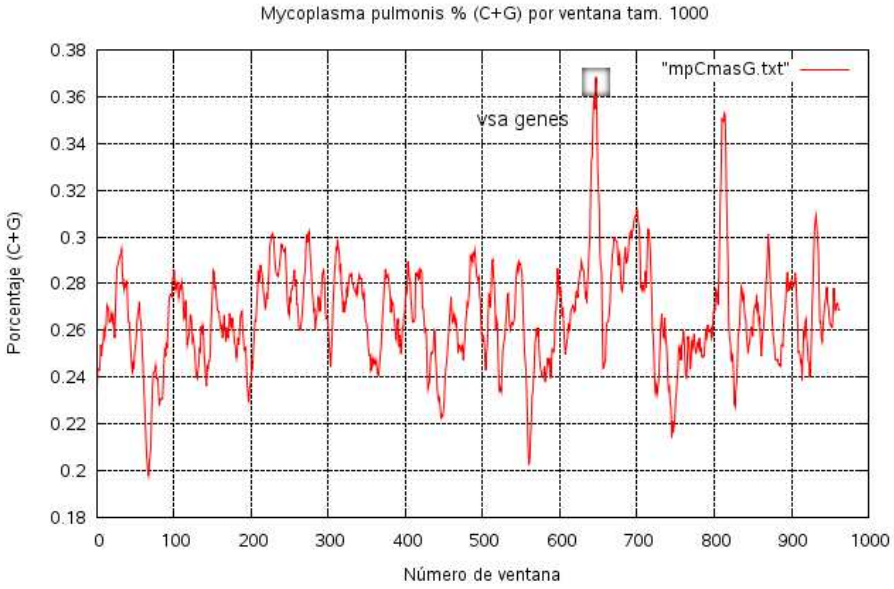


Figura 6.2. Gráfica donde se muestra que al analizar el %(C+G) del genoma de *Mycoplasma pulmonis*, la localización de los *vsa genes* es en el mismo lugar donde se obtienen porcentajes de compresión en el análisis de genomas.

Es interesante señalar que esta bacteria tiene un bajo contenido de $\%(C+G)$ pero en estas zonas donde existe un alto porcentaje de compresión, también existe un %(C+G) más alto al del resto de genoma, y por lo tanto al igual que en el caso de *Homo sapiens* existe una característica estructural

[1]Variable surface antigen. Estos genes codifican las proteínas de superficie V-1 y la variación de estas proteínas contribuyen a la gravedad de las enfermedades respiratorias en los ratones.

especial para este genoma.

Aquí la importancia de este análisis que utiliza herramientas computacionales y que se obtienen resultados comparables con los reportados con otras metodologías moleculares. Estos resultados son posibles al considerar a los genomas como cadenas de símbolos y más en específico de un alfabeto de 4 letras {A,C,G,T}.

El análisis para la arqueobacteria *Sulfolobus todokaii* nuevamente permite ver una estructura muy particular para su genoma. Al principio del genoma tiene un alto porcentaje de compresión así como al final de la secuencia, es interesante ya que si se unen los dos extremos resulta un genoma "circular". Es un caso que comparado con los eucariontes y bacterias permite diferenciarlo debido a esta estructura obtenida a partir de comprimir su secuencia con ventanas móviles.

En el caso de la secuencia pseudo-aleatoria, esta cadena demuestra no tener ninguna estructura, pero por otro lado el contenido de A, C, G y T está distribuido a lo largo de la secuencia pero sin ningún patrón de repetición comparado con cualquier genoma de organismos aquí analizados.

6.1. Conclusiones

El análisis que se presenta en este trabajo es una propuesta para el estudio de secuencias de DNA o genomas, ya que es un caso de estudio actual en disciplinas como la bioinformática, genómica o medicina en las que se trata de entender el DNA de los organismos a partir de su composición tanto estructural como funcional, así como de rescatar información contenida en el DNA. Esto es posible por la disponibilidad de los genomas completos de una gran variedad de organismos en el formato de 4 letras en los bancos de genes que en este caso, son el elemento principal para el análisis bioinformático.

Dado que la molécula del DNA tiene sus procesos biológicos en los que codifica información, en este análisis, se buscan secuencias con patrones o segmentos que nos permitan explicar esos procesos de codificación.

Los resultados obtenidos al analizar las secuencias de DNA, permiten ob-

servar que existen zonas dentro de los genomas en las que es evidente que no son azarosas y por lo tanto, se tiene mayor compresibilidad. A partir de esto, se dice que presentan redundancia. Debido a que los sistemas redundantes presentan mayor compresibilidad, es posible que sean tolerantes a la presencia de errores, pero sin pérdida de información. Esto permite que las secuencias de DNA sean capaces de sufrir algún error o alteración, como es el caso de alguna mutación, toleren el error y mantengan su estructura, ya que los organismos viven en una diversidad de ambientes y que esto implica que se puedan adaptar para sobrevivir dependiendo de la estructura de su genoma.

Este análisis conjunta herramientas y bases teóricas tanto matemáticas como computacionales, así como de la teoría de la información que permiten llevar a cabo un estudio dentro de las secuencias genéticas en su representación de texto o cadenas de letras y permiten descubrir características del DNA de los distintos organismos. También ayuda a encontrar diferencias y semejanzas en la distribución de la información en la molécula del DNA.

Para poder hacer este análisis, se hace un estudio global de lo que es el campo de compresión de datos para poder aplicarlo al método de analizar secuencias genéticas.

Un punto importante es que al utilizar técnicas de compresión de datos, los genomas se interpretan como un texto sobre el alfabeto de 4 letras, sin utilizar ninguna característica fisicoquímica del DNA y esto permite manipular de forma sencilla las secuencias para poder así, encontrar subsecuencias en las que se presentan determinadas características estructurales que representan zonas en las que el DNA aporta información que no podría ser vista de otra manera para el análisis de los organismos.

La implementación del algoritmo de compresión *biocompress-2* permite encontrar ciertas estructuras dentro de los genomas, a las que se llaman palíndromos y esto permite inferir que existen zonas dentro del DNA en las que la información está situada con cierto orden y por lo tanto se encuentran alejadas del azar ya que mantienen una estructura especial y se mostró que al aplicar los otros algoritmos de compresión, la estructura de las secuencias se mantiene invariante.

Se muestra que las secuencias alcanzan un promedio de porcentaje de compresión similar a lo largo de sus secuencias; sin embargo, se logra identificar diferentes estructuras en los porcentajes de compresión para los genomas mientras que la secuencia pseudo-aleatoria es constante, independientemente del algoritmo de compresión que se utilice.

El resultado de aplicar los algoritmos de compresión a las secuencias genéticas también implica obtener una medida de la información que hay en las cadenas por lo que la compresión puede ser interpretada como una medida de complejidad del DNA, ya que se demuestra que las secuencias no son aleatorias ni periódicas. Esto es que la información que se encuentra en el DNA pertenece a una zona muy particular entre lo homogéneo y la zona que no distingue algún patrón, o sea, el azar.

También se detectó que el lugar (las ventanas) en el que aparecen los nucleótidos C y G, tiene una mayor estructura en cada genoma y, gracias a esto, presentan ciertos patrones dentro de la secuencia y permiten distinguir propiedades de cada secuencia.

Durante el análisis se encontraron secuencias en las que dentro del genoma existen (codifican) segmentos con cierta información, como es el caso de los genes encontrados para el genoma del *mycoplasma pulmonis*. Esto se debe a que se detectan características estructurales y espaciales dentro de la secuencia genética y se espera que presenten algún tipo de información. Sin embargo, al encontrar segmentos que contienen cierta información o redundancia en las bases contenidas ahí, no garantiza saber el funcionamiento, ya que la funcionalidad del DNA es un concepto más complejo que el simple hecho de tener la secuencia.

Esto abre la oportunidad para trabajo futuro de profundizar en este análisis y compartir conocimiento con biólogos genéticos, genomistas y moleculares para interpretar si realmente estos segmentos que tienen mayor compresibilidad obtenidos en el análisis para diferentes genomas codifican algún gen o tienen un significado biológico que permita comprender y obtener la mayor información posible que aportan estos segmentos, como en el caso del *Homo sapiens* y de la bacteria *Mycoplasma pulmonis*.

De esta manera se pretende compartir conocimiento entre la biología y

las ciencias de la computación dando cauce a resultados de gran interés para ambas ciencias, proponiendo líneas de investigación en las que se tenga un mejor conocimiento de la evolución, comportamiento y funcionalidad de la molécula del DNA.

Por lo tanto, esto es una invitación al trabajo interdisciplinario, donde el estudio de las secuencias del DNA representa un problema de la biología que se aborda desde la perspectiva de las matemáticas, la teoría de la información y las ciencias de la computación bajo el enfoque de los sistemas complejos.

Bibliografía

Abramson, N. (1986). *Teoría de la información y codificación.* Paraninfo, Madrid.

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K. y Walter, P. (2002). *Molecular biology of the cell.* pub-GARLAND, cuarta edición, Nueva York.

Apostolico, A. y Fraenkel, A.S. (1987). "Robust Transmission of Unbounded Strings Using Fibonacci Representations". *IEEE Transactions on Information Theory*, **33**(2):238–245.

Arias, C. (2004). *Fundamentos y casos exitosos de la biotecnología moderna.* El Colegio Nacional, México.

Arndt, C. (2001). *Information Measures.* Springer, Berlín Heidelberg.

Blast. Página web ncbi. `http://blast.ncbi.nlm.nih.gov/BLAST.cgi`, (2009).

Burrows, M., Burrows, M., Wheeler, D. J. y Wheeler, D. J., (1994). A block-sorting lossless data compression algorithm. Technical report, Digital SRC Research Report.

Chambaud, I., Heilig, R., Ferris, S., Barbe, V., Samson, D., Galisson, F., Moszer, I., Dybvig, K., Wróblewski, H., Viari, A., Rocha, E.P.C. y Blanchard, A. (2001). "The complete genome sequence of the murine respiratory pathogen Mycoplasma Pulmonis". *Nucleic Acids Research*, **29**(10):2145–2153.

Entrez. Página web ncbi. `http://www.ncbi.nlm.nih.gov/`, (2009).

Frankel, E. (2003). *DNA, el proceso de la vida.* Siglo Veintiuno editores, México.

Grumbach, S. y Tahi, F. (1993). "Compression of DNA sequences". *DCC '93: Proceedings of the Conference on Data Compression*, páginas 340–350.

Grumbach, S. y Tahi, F. (1994). "A new challenge for compression algorithms: genetic sequences". *Information Processing & Management*, **30** (6):875–886.

Gupta, R. (2000). "The natural evolutionary relationships among prokaryotes". *Critical reviews in microbiology*, **26**(2):111–131.

Huffman, D. (1952). "A Method for the Construction of Minimum Redundancy Codes". *Proceedings of de IRE*, **40**(9):1098–1101.

Jung, B. y Burleson, W. P. (1995). "Real-time VLSI compression for high-speed wireless local area networks". *DCC '95: Proceedings of the Conference on Data Compression*, página 431.

Li, M. y Vitányi, P. (1997). *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, Nueva York ,EUA.

Li, M. y Vitányi, P. (2002). "Algorithmic Complexity". *International Encyclopedia of the Social & Behavioral Sciences*, páginas 376–382.

Li, W. y Miramontes, P. (2006). "Large-scale oscillation of structure-related DNA sequence features in human chromosome 21". *Physical Review E*, **74** (2).

Miramontes, P. y Cocho, G. (2003). "DNA dimer correlations reflect in vivo conditions and discriminate among nearest-neighbor base pair free energy parameter measures". *Physica A: Statistical Mechanics and its Applications*, **321**(3-4):577–586.

Miramontes, P., Medrano, L., Cerpa, C., Cedergren, R., Ferbeyre, G. y Cocho, G. (1995). "Structural and Thermodynamic Properties of DNA Uncover Different Evolutionary Histories". *Journal of Molecular Evolution*, **40**(6):698–704.

Rivals, E., Dauchet, M., Delahaye, J.P. y Delgrange, O. (1996). "Compression and Genetic Sequence Analysis". *Biochimie*, **78**(5):315–322.

Sayood, K. (2000). *Introduction to Data Compression.* Morgan Kaufmann, Segunda Edición, San Francisco California, EUA.

Shannon, C.E. (1948). "A Mathematical Theory of Communication". *Bell System Technical Journal*, **27**:379–423, 623–656.

Shen, X., Gumulak, J., Yu, H., French, C. T., Zou, N. y Dybvig, K. (2000). "Gene rearrangements in the vsa locus of Mycoplasma pulmonis". *J. Bacteriol.*, **182**(10):2900–2908.

Stauffer, L. M. y Hirschberg, D. S., (1993). Parallel text compression REVISED. Technical Report ICS-TR-91-44, Universidad de California, Irvine.

Storer, James A. y Szymanski, Thomas G. (1982). "Data compression via textual substitution". *J. ACM*, **29**(4):928–951.

Turing, A. M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem". *j-PROC-LONDON-MATH-SOC-2*, **42**:230–265.

Wayner, P. (2000). *Compression Algorithms.* Morgan Kaufmann, San Francisco California, EUA.

Welch, T. (1984). "A Technique for High-Performance Data Compression". *IEEE Computer*, **17**(6):8–19.

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

Witten, I.H., Moffat, A. y Bell, T.C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images.* Segunda Edicion, San Francisco, CA, Academic Press.

Woese, C., Kandler, O. y Wheelis, M. (1990). "Towards a natural system of organisms: proposal for the domains Archaea, Bacteria and Eucarya". *Proceedings of the National Academy of Sciences of USA*, **87**(2):4576–4579.

Ziv, J. y Lempel, A. (1977). "A Universal Algorithm for Sequential Data Compression". *IEEE Transaction on Information Theory*, **3**(3):337–343.

Ziv, J. y Lempel, A. (1978). "Compression of Individual Secuences via Variable Rate Coding". *IEEE Transactions on Information Theory*, **24** (5):530–536.

Zweiger, G. (2002). *El Genoma, Información, anarquía y revolución en las ciencias biomédicas.* McGraw Hill. México.