Proyecto Fin de Carrera de Ingeniería en Informática

Algoritmos de compresión para secuencias biológicas y su aplicación en árboles filogénicos construidos a partir de ADN mitocondrial

Pablo Urcola Irache

Directora: Elvira Mayordomo Cámara

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior

Noviembre 2006

Comprimir es comprender
Jorge Wagensberg

Agradezco a todo el mundo que me ha ayudado o me ha ofrecido su ayuda en algún momento de la realización de este proyecto su interés y disposición. Como último favor les pido que me dejen dedicar este trabajo a mi abuelo.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Memoria

1

Introducción

En esta sección se describen los elementos fundamentales en que se enmarca el proyecto como pueden ser el objetivo, el alcance, el trabajo previo en que se apoya, el contexto en que se desarrolla o las herramientas que se han utilizado para su realización. Además se indica de forma abreviada el contenido y distribución del resto de las secciones de la memoria.

1.1. Objetivos

Los objetivos del proyecto, tal y como se indicaron en la propuesta, son los siguientes:

- Estudio de los algoritmos de compresión de secuencias biológicas utilizados en la actualidad, su comparación con los de propósito general y la obtención de unas pautas de diseño que se deben tener en cuenta al crearlos.

- Estudio de la aplicación de los algoritmos de compresión a la generación de árboles filogénicos utilizando la información contenida en el ADN mitocondrial y análisis de las características necesarias de los primeros así como los distintos métodos de construcción de los árboles.

1.2. Alcance

Definidos estos objetivos, la intención del proyecto no es superar los niveles de compresión o velocidad de otros algoritmos sino caracterizar de la mejor forma los fundamentos que un compresor de secuencias de adn debiera tener. Finalmente esta caracterización se ha implementado para probar en qué forma mejora los algoritmos existentes.

En relación a los árboles filogénicos, éstos no se pueden comparar con otros puesto que hay multitud de versiones válidas y respaldadas cada una de ellas por alguna teoría, así que se considera que se ha alcanzado un resultado bueno al encontrar un método que se aproxima a las ideas más aceptadas en la actualidad por la mayoría. Para la contrucción de los árboles se utiliza la información genética del ADN mitocondrial puesto que debido a su pequeño tamaño se hace más manejable sin perder capacidad de diferenciación entre los distintos individuos.

1.3. Trabajo previo

La documentación preexistente tanto en el campo de los algoritmos de compresión como en los árboles de filogenia es muy extensa y diversa, aunque, debido a que en general son bastante recientes, la validez de sus contenidos no está muy contrastada, por lo que hay que agudizar el sentido crítico frente a ellos.

Existe una gran variedad de algoritmos de compresión genéricos ([13, 12, 20]) y algunos dedicados a secuencias biológicas descritos en [4], [5] y [10], generalmente integrados en las grandes bases de datos de genomas que existen como NCBI[1] o GenBank[2]. Ambos tipos sirven como base para el proyecto puesto que se pretende saber por qué los primeros no funcionan con secuencia biológicas y qué características hacen a los segundos eficaces a la hora de comprimir este tipo de datos.

En cuanto a los árboles de filogenia, la variedad de métodos está relacionada directamente con la diversidad de hipótesis sobre la evolución de las especies, como se explica en [6], y con los distintos métodos numéricos de cálculo de distancias, descritos en [16].

1.4. Contexto

El desarrollo de este proyecto se enmarca dentro del grupo de investigación de complejidad computacional (GCC) que forma parte del Grupo de Ingeniería de Sistemas de Eventos Discretos (GISED) del Departamento de Informática e Ingeniería de Sistemas (DIIS). Este grupo investiga sobre el funcionamiento de los distintos algoritmos de compresión de uso general y quería averiguar por qué no eran útiles al comprimir secuencias biológicas. Debido a que se desarrolla en el seno de un grupo de investigación, el proyecto tiene una gran carga de trabajo de recopilación de información y un apartado más reducido de los elementos típicos de un proyecto de software.

[1] http://www.ncbi.nlm.nih.gov/

[2] http://www.psc.edu/general/software/packages/genbank/genbank.html

Este trabajo ha sido parcialmente financiado por el proyecto de investigación del Ministerio de Educación y Ciencia TIN2005-08832-C03-02, que cuenta con el apoyo de la empresa deCODE genetics.

1.5. Métodos, técnicas y herramientas

Gran parte del tiempo de trabajo del proyecto se invirtió en documentación, fundamentalmente lectura de artículos y consulta con otros investigadores que trabajan en el mismo tema. La disposición de los artículos y de algunos algoritmos fue dificultosa en algunos casos debido a las licencias restrictivas, lo que redujo la capacidad de investigación.

Una vez conseguida toda la información al alcance, se extrajeron las ideas para la implementación del software desarrollado para posteriormente verificarlo y compararlo con los programas ya existentes. Por último se diseñó un interfaz amigable que facilitara la utilización y la extracción de los datos de la ejecución para su posterior análisis. Además se pudo incluir uno de los algoritmos ya existentes [10] en el mismo interfaz gráfico gracias a que se facilitaron los ficheros fuentes para su utilización, lo que simplifica la labor de comparación de ambos algoritmos.

El trabajo con los árboles de filogenia no se decantaba tanto por la implementación sino por la realización de pruebas y el análisis de los resultados. Por eso se optó por el uso de un paquete de programas[3] que alcanzara todas las necesidades del proyecto, facilitando el trabajo con los árboles sin necesidad de una implementación propia de los métodos.

1.6. Contenido de la memoria

En una primera sección, se definen los conceptos relativos a secuencias biológicas y árboles de filogenia necesarios para la comprensión del trabajo realizado en el proyecto. La sección 3 aborda el proceso de documentación seguido hasta alcanzar el diseño final del compresor, indicando todos los pasos recorridos y las ideas adquiridas en cada uno de ellos. Posteriormente se profundiza en la mejora implementada utilizando los conocimientos adquiridos en la fase de estudio. La sección 5 está dedicada al trabajo realizado sobre los árboles de filogenia. Las dos últimas secciones contienen respectivamente información sobre las pruebas realizadas, los resultados que se obtuvieron y, por último, las conclusiones que se han podido extraer de todo el trabajo realizado.

[3] http://evolution.genetics.washington.edu/phylip.html

2

Conceptos previos

En esta sección se explican los conceptos necesarios para entender algunos elementos del proyecto que se alejan de la informática, fundamentalmente relacionados con la genética y con la teoría de la evolución.

2.1. Secuencias biológicas

Llamamos secuencia biológica a la representación de cualquiera de la sucesiones de monómeros que intervienen en la síntesis de proteínas a partir de la información genética de los organismos. Se pueden diferenciar tres tipos de secuencias distintas:

- Ácido desoxirribonucleico o ADN
- Ácido ribonucleico o ARN
- Proteínas

Cada uno de los elementos básicos que componen una de estas moléculas es un monómero, siendo de distinta naturaleza en cada una de ellas como se aprecia en el cuadro 2.1.

Molécula	**Monómero**
ADN	Base nitrogenada
ARN	Base nitrogenada
Proteína	Aminoácido

Cuadro 2.1: Moléculas y monómeros

Todas ellas están implicadas en el proceso de extracción de la información genética y su transformación.

2.1.1. Formación de proteínas

En el proceso de transcripción, el ADN se utiliza para la síntesis de ARN de diversos tipos. Sin embargo, no toda la información existente en el ADN queda reflejada luego en el ARN. Existen fragmentos, llamados ADN basura, de los que, hoy por hoy, no se conoce funcionalidad alguna. El ARN resultante está exclusivamente dedicado a la generación de moleculas de proteínas.

La fase de traducción es en la que se generan aminoácidos a partir de la información contenida en las moléculas de ARN. Para la construcción de un único aminoácido se necesitan tres bases de la secuencia de ARN. Esta traducción se realiza utilizando el llamado código genético por el que se hacen corresponder las tripletas de bases nitrogenadas con aminoácidos. Un mismo aminoácido puede ser sintetizado utilizando la información de distintas tripletas.

Figura 2.1: Procesos de transcripción y traducción

Desde el punto de vista práctico, el uso del ARN es prácticamente imposible debido a que es un elemento temporal en el conjunto del proceso y su inestabilidad impide su manejo en el laboratorio. Si analizamos el proceso atendiendo a la información almacenada, la fase de traducción implica una pérdida de información al sintetizar las proteínas usando el antes mencionado código genético. Por estas razones se decidió utilizar las secuencias de ADN para su almacenamiento y compresión, a pesar de que contienen información aparentemente inútil.

2.1.2. Estructura del ADN

Ya se ha dicho arriba que el ADN estaba formado por una sucesioón de bases nitrogenadas. Estas bases son cuatro que, por comodidad, se representan por su inicial:

- Adenina
- Citosina
- Guanina
- Timina

La molécula de ADN tiene una estructura de doble secuencia de bases complementarias. Dado que los emparejamientos están prefijados, sólo es necesario almacenar una de ellas para conocer la otra. La longitud de estas moléculas puede oscilar desde unos centenares a millones de bases.

Cabe decir que los estudios en esta rama de la ciencia están en pleno desarrollo lo que impide en estos momentos otorgar significado alguno a las distintas combinaciones de bases de forma que pudiera repercutir en los métodos utilizados durante la compresión. Por este motivo se decidió que el algoritmo de compresión no debería hacer distinciones entre las distintas subcadenas de ADN.

2.2. Árboles de filogenia

Según la teoría de la evolución de las especies de Charles Darwin, *todos los seres vivos son la evolución de un antepasado común.* Un árbol de filogenia es un grafo conexo acíclico en el que cada nodo representa a un organismo y cada arco representa la relación de descendencia o evolución, de forma que se puede establecer qué seres tienen el antecesor común más cercano. Podríamos considerar el árbol genealógico de una familia como un árbol de filogenia aunque los que se tratan en la realidad son más genéricos y, en lugar de tratar con individuos, tratan con especies. Sin embargo, un ejemplo con un árbol genealógico servirá para explicar el uso y la construcción de estas estructuras.

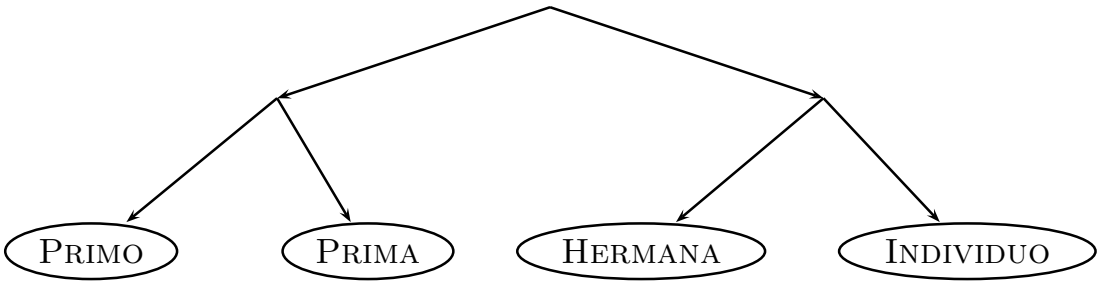

Figura 2.2: Ejemplo de árbol de filogenia de cuatro individuos

La figura 2.2 es el árbol filogénico para cuatro individuos. Se aprecia cómo se crea el árbol de ancestros lógico según los parentescos dispuestos. Cabe resaltar que se crean nuevos individuos en un principio imaginarios que son ancestros de los reales o de otros creados de esta forma. Las conclusiones que se pueden extraer es que primo es más cercano a prima que al resto y lo mismo podemos decir de hermana e individuo. En este ejemplo, los nodos creados por el algoritmo corresponderían al abuelo, al padre y al tío del individuo.

La complejidad de su construcción reside en decidir qué datos de cada individuo se usan para el estudio y en extraer información que sirva para construir el árbol a partir de los datos de cada individuo. Por supuesto, en este proyecto sólo nos interesan los obtenidos a partir de las secuencias de ADN mitocondrial de los distintos individuos.

Este tipo de ADN, el mitocondrial, es algo especial puesto que sólo aparece en las mitocondrias, que son unos orgánulos presentes en las células de los organismos pluricelulares que realizan la respiración celular. Son los encargados de transformar el oxígeno en energía. La información genética almacenada es realmente extraña puesto que no sufre alteraciones durante la reproducción sino que es transmitida de forma directa a través del óvulo materno. De esta forma las variaciones sufridas a lo largo del tiempo son mucho menores que el resto de las cadenas de ADN, aunque estudios muy recientes apuntan cada vez más hacia la gran relevancia de dichas variaciones [15].

3

Desarrollo del estudio

En este apartado se refleja el proceso de estudio de los distintos compresores de secuencias biológicas que han sido investigados, desde simples adaptaciones de algoritmos de compresión de carácter general a otros ya creados específicamente para secuencias de ADN.

3.1. Adaptación del algoritmo de Lempel-Ziv

La primera opción como algoritmo de compresión para secuencias biológicas que sugirió la directora del proyecto fue la adaptación de los algoritmos de Lempel-Ziv al alfabeto formado por las cuatro bases nitrogenadas {A, C, G, T}. Generalmente, los algoritmos implementados de este tipo trabajan con bits o bytes, de forma que había que introducir alguna pequeña variación para que funcionaran con las secuencias de ADN. De entre la gran variedad de algoritmos de la familia LZ se eligió el LZ77 porque ofrecía la máxima simplicidad sin pérdida de capacidades.

Este algoritmo recorre la entrada buscando en la parte aún no procesada una cadena que también esté en la parte ya recorrida, tal y como se explica en [20]. En caso de encontrar una repetición suficientemente larga, escribe una referencia a ella en lugar de la propia subcadena. Si no se encuentra tal repetición, se escribe el primer símbolo aún no procesado directamente en la salida y se continua con el siguiente símbolo hasta terminar el fichero.

La búsqueda puede limitarse hasta cierta ventana de comparación o comprobar todo el fichero. Si está limitada en un cierto rango, la escritura de la referencia es más sencilla puesto que sólo pueden escribirse cierto rango de valores, lo que se puede hacer con un número de bits fijo. Sin embargo, al limitar la búsqueda disminuye la probabilidad de encontrar una repetición y como consecuencia, la capacidad de compresión, por esta razón se eligió un algoritmo que hiciera una búsqueda completa. Por tanto, las referencias tuvieron que escribirse usando una

codificación de longitud variable [1]. En todos los casos se exigía que una repetición tuviera una longitud mínima tal que escribir una referencia a ella ahorra espacio en comparación a escribirla de forma directa. Esta parte sufrió la mayor variación junto con la codificación de los símbolos puesto que sólo se necesitan dos bits para codificar cualquiera de las cuatro bases nitrogenadas.

3.1.1. Resultados

Los resultados obtenidos mediante esta adaptación no son en general buenos puesto que no se alcanzan niveles de compresión significativos. Usando como unidad el número de bits que necesita una base para ser escrita, el mejor de estos algoritmos conseguía alcanzar una ratio de 1.927. Teniendo en cuenta que sin utilizar ningún método de compresión, al ser únicamente cuatro elementos los que componen las secuencias de ADN, se puede alcanzar unos niveles de compresión de 2 bits por base, el resultado no es muy bueno. De hecho, si se considera que el coste en tiempo de las búsquedas era bastante grande, no merecía la pena usar el algortimo implementado frente a la codificación directa de las bases.

3.2. Análisis de los algoritmos de compresión específicos

Al quedar patente que con la adaptación del algoritmo de Lempel-Ziv no se alcanzaban resultados muy buenos se decidió analizar en profundidad los algoritmos creados para la compresión de secuencias de ADN. No resultó dificil encontrar el que parecía dar mejores resultados, DNACompress [5]. Es el último de los algoritmos conseguidos por un mismo grupo de trabajo que también fue el responsable del GenCompress [4].

3.2.1. GenCompress

Este algoritmo es muy similar al algoritmo de Lempel-Ziv presentado arriba. También recorre secuencialmente el fichero buscando repeticiones en la parte que ya ha sido analizada. La diferencia clave es que introduce la capacidad de detectar repeticiones inexactas. Al parecer, las secuencias de ADN, al estar sujetas a mutaciones puntuales, pueden poseer repeticiones de bastante longitud en las que unos pocos elementos difieren entre ambas apariciones.

Al introducir esta posibilidad la forma de referenciar repeticiones cambia pues debe adaptarse a las nuevas capacidades. Ésto se traduce en que una referencia inexacta se codifica como una referencia exacta más una serie de operaciones de

edición (inserción, sustitución y borrado) que afectan a algunos de los elementos de esa repetición.

La forma en que decide si una repetición debe tenerse en cuenta o no depende ahora de más factores. Para simplificar, se asignan puntos positivos por cada elemento repetido y negativos en caso de elementos diferentes. Esta puntuación se calcula en función del coste de la codificación de la cadena de operaciones que generaría. Cuando la puntuación de una repetición baja por debajo de un umbral se considera que no aportará más compresión si se continua con la misma referencia así que se decide acabar allí la repetición.

La idea de aceptar repeticiones inexactas es innovadora pero los resultados empíricos no nos convencían mucho. En general, los elementos diferentes de las repeticiones aparecen al final de las repeticiones puesto que son estos los que hacen bajar la puntuación hasta el límite permitido. Por esta razón no se consideró tan importante la capacidad de incluir repeticiones inexactas, al menos consideradas de esta forma.

3.2.2. Pattern Hunter

La siguiente propuesta de este grupo de trabajo proponía un cambio radical. El compresor DNACompress se basa en los resultados de un motor de búsqueda de repeticiones llamado Pattern Hunter [8]. Este programa independiente devuelve una lista de las repeticiones inexactas más largas que tienen en común dos o más secuencias de ADN. DNACompress utiliza las salida producida por Pattern Hunter al ejecutarlo con un mismo fichero como entrada para comprimir dicho fichero cogiendo las mejores repeticiones.

El algoritmo que nos interesa es el de Pattern Hunter. La diferencia principal con su predecesor es que trata la entrada de una manera totalmente distinta. Lo primero que hace es sacar una cadena binaria de comparación elemento a elemento de las entradas en las que un 1 indica que los elementos son iguales y un 0 indica que son distintos. Podemos ver un ejemplo en la figura 3.1.

Secuencia A	aaaccccaagggt
Secuencia B	aagctcaaagcgg
Diferencia	1101010111010

Cuadro 3.1: Ejemplo de construcción de cadena binaria

Cuando la entrada es una única cadena la compara con una copia desplazada de forma que todas las posiciones sean comparadas. Para buscar las repeticiones simplemente tendría que buscar secuencias de unos en la cadena binaria obtenida

pero como también permite repeticiones inexactas, lo que hace es buscar apariciones de un determinado patrón fijo de longitud corta de forma que no require que todos los elementos sean unos. Por ejemplo, podría buscar en esa cadena binaria el patrón 11011 que indica que busque repeticiones inexactas que tengan dos bases repetidas alrededor de cada bases diferente. Una vez encontrados los patrones los intenta extender hacia ambos lados para obtener la mayor longitud posible. De nuevo, para considerar si un repetición es aceptable o no, utiliza el sistema de puntuación que también usa la versión anterior.

Los resultados avalan este método fundamentalmente en lo que a velocidad se refiere puesto que al usar cadenas binarias las comparaciones se pueden agilizar. Sin embargo mantiene el problema de su predecesor de dejar las inexactitudes en las repeticiones cercanas a ambos extremos de la cadena.

3.2.3. Pattern Hunter II

La última versión del compresor DNACompress [7] consiste únicamente en la modificación del motor de búsqueda de repeticiones. Básicamente se trata de ampliar la posibilidad de encontrar repeticiones inexactas utilizando más de un patrón sobre la cadena binaria lo que hace que sea un algoritmo más exhaustivo que su primera versión sin aumentar drásticamente el tiempo de ejecución. Se presenta pues un nuevo problema muy complejo consistente en seleccionar un conjunto de patrones que funcionen de la mejor forma posible.

Los resutados de esta nueva versión son muy similares a los de la primera versión aunque aumenta ligeramente la cantidad de repeticiones encontradas, lo cual mejora la compresión obtenida.

El problema de estos dos algoritmos, claramente los mejores, es que no hay información suficiente para estudiarlos en profundidad y mejorarlos sin tener que usar ingeniería inversa a partir del producto final para reconstruirlos. Esta razón hacía imposible su uso como base para implementar alguna mejora.

3.2.4. dnaX

A pesar de los buenos resultados de los algoritmos Pattern Hunter, no resultaba muy satisfactoria la solución basada en las repeticiones inexactas puesto que, analizando los resultados, los elementos distintos aparecen de forma muy escasa (nunca más de tres en una repetición) y generalmente situados hacia los extremos de las secuencias repetidas. Se consideró posible alcanzar los mismos niveles de compresión sin necesidad de tener en cuenta las repeticiones inexactas simplemente acortando las repeticiones.

El algoritmo dnaX [10] opta por esta solución y añade una característica nueva como es la búsqueda de repeticiones invertidas. Al consultar ésto con bioquímicos

se descubrió que no era descabellado encontrar inversiones de trozos de secuencias de longitudes que pueden llegar a casi un millón de bases [18].

Este algoritmo funciona de una forma algo distinta que los vistos con anterioridad. Básicamente trocea la secuencias en fragmentos de 20 bases y, tras la búsqueda de repeticiones exactas, codifica la entrada según estos bloques, diciendo qué partes de las 20 bases están ya repetidas y qué partes no, referenciando las primeras y escribiendo las últimas directamente en el fichero de salida. Las referencias a repeticiones deben ahora llevar un bit que indique si son repeticiones directas o inversas.

Según el artículo que lo describe, los resultados que presenta este algoritmo son similares a los del Pattern Hunter tanto en tiempo como en tasa de compresión. Además, el autor facilitó las fuentes del algoritmo lo que hacía mucho más fácil el trabajo partiendo de su código para introducir alguna mejora.

4

Implementación del compresor

En esta sección describo la solución implementada como compresor de secuencias de ADN. En la sección 6 se describen los resultados obtenidos con nuestro compresor y en comparación con los ya existentes.

4.1. Especificación del algoritmo

Algoritmo: Compresor de ficheros de ADN

Entrada: Fichero en el que únicamente aparezcan los caracteres A, C, G y T en mayúsculas o minúsculas.

Salida: Se crea un fichero que ocupa menos espacio que la entrada y que contiene la misma información, de forma que se pueda recuperar el original utilizando un descompresor.

4.2. Directivas de diseño

Tras el estudio realizado al problema se han obtenido ciertos elementos de diseño que se considera que el algoritmo debe tener presentes. Éstos se refieren tanto a las características particulares del tipo de datos tratado como al uso que se le va a dar a la aplicación obtenida.

4.2.1. Sobre los datos

- Las secuencias de ADN están formadas por cuatro elementos, por tanto con 2 bits por cada base se puede escribir el fichero. Cualquier archivo comprimido generado debería superar esta tasa de compresión.

- La longitud de las secuencias de ADN es enormemente diversa, desde unos pocos cientos a varios millones de bases.
- No se conoce ninguna estructura común a todas las secuencias de ADN suficientemente definida que permita dividirlo en secciones por razones semánticas o funcionales.
- La frecuencia de las distintas bases nitrogenadas es, en general, igual para todas ellas.
- El ADN suele contener repeticiones de longitudes muy variadas que no tienen por qué seguir cierta localidad espacial.
- Debido a la posibilidad biológica del ADN de sufrir mutaciones puntuales pueden aparecer repeticiones de secuencias no exactas que se diferencian en unas pocas bases.
- En las secuencias de ADN pueden aparecer fenómenos de inversión de ciertas subcadenas.

4.2.2. Sobre el uso del algoritmo

Las condiciones de uso del algoritmo son de vital importancia para tenerlas en cuenta al diseñarlo. En el caso que nos ocupa el proceso de trabajo con una secuencia de ADN es aproximadamente el siguiente.

1. Se extrae en el laboratorio la molécula de ADN que se desea obtener.
2. Se secuencializa, también por métodos químicos, la molécula extrayendo así la sucesión de bases nitrogenadas.
3. Se introduce la secuencia obtenida en una base de datos de ADN.
4. Se consulta todas las veces que se necesite para compararla con otras muestras obtenidas o para cualquier otro cometido.

Se puede deducir que la operación que más se realiza con este tipo de datos es la de consulta lo que nos sugiere que es ésta operación la que hay que agilizar en mayor medida, pudiendo invertir más tiempo en otras operaciones.

4.3. Estructura del algoritmo

El algoritmo se divide en tres fases diferenciadas para facilitar su comprensión y su mantenimiento y son el análisis de la entrada, la selección de las repeticiones y la codificación, como se ve en la figura 4.1.

Figura 4.1: Fases del algoritmo

4.3.1. Análisis de la entrada

En esta fase se trata de extraer la máxima información de la entrada que pueda servir posteriormente para comprimir al máximo. Fundamentalmente lo que se buscan son repeticiones dentro del fichero de forma que luego puedan sustituirse las subsecuencias por referencias a otras que son iguales.

La búsqueda de las repeticiones en el fichero es una de las partes más costosas del algoritmo y donde hay que refinar mucho el código para optimizar todo lo que se pueda el tiempo de ejecución. Sin embargo hay que tener en cuenta lo comentado anteriormente del proceso de trabajo con una secuencia de ADN en la que la compresión únicamente se realizaría una vez antes de almacenarla en la base de datos por lo que puede ser relativamente lenta comparado con la descompresión.

Dado que la velocidad del algoritmo no era una prioridad se implementó como primera versión una busqueda exahustiva simple en la que para cada posición se buscaba en todo el fichero la repetición más larga. Este código, aunque eficaz, era ya intratable para ficheros de unos pocos millares de bases pues hacía enormemente costoso realizar cualquier prueba.

La segunda opción surgió a partir de lo leído en [3] que indicaba que los árboles de sufijos eran un tipo de datos que se amoldaba muy bien a la tarea a realizar. La dificultad de implementación de las operaciones sobre estas estructuras hizo que se optara por una versión más sencilla aunque un poco más ineficiente, los vectores de sufijos [9, 17].

Los vectores de sufijos se basan en la ordenación de los sufijos de una cadena de forma que se pueden encontrar las subcadenas repetidas como los prefijos comunes de dos sufijos consecutivos en la lista ordenada. Esta solución era satisfactoria en velocidad pero presentó un problema en la fase de selección de repeticiones. En caso de que dos subcadenas repetidas estén solapadas o varias repeticiones se refieran a una misma sección de la secuencia se debía descartar su uso y no se podían utilizar ni siquiera parcialmente para otra posible repetición.

Debido a estos problemas se volvió a intentar la solución basada en los árboles de sufijos porque permitían salvar los fallos de la anterior propuesta. Siguiendo las explicaciónes de Ukkonen [19] se implementó esta solución. Sin embargo no resultó la definitiva por varios motivos. El primero es que no había una forma sencilla de ampliar los árboles de sufijos para que permitieran encontrar repeticiones invertidas y el segundo fue que era muy costoso encontrar las repeticiones porque aunque

la búsqueda de una subacadena es muy rápida, el número de subcadenas de una secuencia es muy grande como para que sea suficientemente rápido buscar todas y coger las mejores.

A pesar de esto, la visión que Ukkonen da a los árboles de sufijos tratándolos como autómatas sugirió la posibilidad de usar algún autómata para este trabajo. Precisamente ésto es lo que se describe en [14], donde se usan los autómatas para encontrar todas las repeticiones de subsecuencias en una secuencia.

El funcionamiento además es simple y curioso. Basta con construir un autómata finito no determinista capaz de reconocer todas las subcadenas de una cadena de ADN y luego, al aplicarle la tranformación que lo convierte en determinista, se puede recuperar la información correspondiente a las repeticiones encontradas.

Su sencillez permite a la vez extender el algoritmo descrito en el artículo para encontrar también las repeticiones invertidas simplemente modificando el autómata para que reconociera las subcadenas invertidas de la secuencia.

4.3.2. Selección de las repeticiones

Esta segunda fase recoge las repeticiones encontradas en la primera y selecciona las que mayor tasa de compresión permiten. La compresión que aporta una repetición se puede cuantificar teniendo en cuenta lo que cuesta codificar las bases sin referenciar y lo que constaría crear la referencia.

Una referencia consta de cinco elementos:

Original Posición de la primera aparición

Repeticion Posición de la segunda aparición

Longitud Longitud

Tipo Es directa o invertida

Los tres primeros son enteros positivos y el último un booleano. No se puede saber a priori la magnitud de los enteros puesto que no conocemos las posiciones relativas de las repeticiones y no hay ninguna longitud predilecta. Sin embargo, para tener en consideración la aparición de posibles mutaciones puntuales, una vez ordenadas las repeticiones por orden de posición dentro del fichero, se puede suponer que dos repeticiones consecutivas están muy próximas y se puede almacenar la distancia entre ambas en lugar de la posición absoluta.

La cuantificación del beneficio de una referencia queda determinado por la siguiente fórmula:

$$Beneficio = Referencia - Codificacin\ directa$$

$$Referencia = 3 \times Entero + 1$$
$$Codificacin\ directa = 2 \times Longitud$$

Además de seleccionar las repeticiones usando el beneficio como principal factor hay que tener en cuenta que no se referencien dos veces la misma zona del fichero por lo que hay que controlar que las repeticiones no se solapen.

Si tenemos en cuenta el algoritmo de descompresión se puede apurar aún más el uso de las repeticiones. La reconstrucción se realiza en orden secuencial de forma que no es necesario tener toda la referencia completa antes de escribir el trozo repetido. Se pueden utilizar repeticiones solapadas. Un ejemplo de un caso extremo pero muy clarificador es el siguiente:

Secuencia: `aaaaaaaa`

Si consideramos las repeticiones que no se solapen nos podría quedar un fichero comprimido con la siguiente información:

`aaaa, (0, 4, 4, normal)`

El paréntesis corresponde a la repetición que comienza en la posición 4, es igual a la que empieza en la posición 0 de longitud 4 en el orden normal.

Si consideramos las repeticiones solapadas, esta misma secuencia podría quedar de la siguiente forma:

`a, (0, 1, 7, normal)`

Vemos que ahora la repetición comienza en la posición 1 y que es igual a la que empieza en la posición 0 de longitud 7 en sentido normal. Esto es posible porque en cuanto se escribe la primera "a" repetida se incorpora a la cadena, lo que servirá a su vez para reconstruir el resto.

Esta posibilidad no se presenta cuado las repeticiones están invertidas porque se necesita el último elemento de la subsecuencia original para recuperar la primera base de la repetida. Así que a la hora de seleccionar las secuencias que se usarán en las referencias, las repeticiones directas son mejores.

4.3.3. Codificación de la salida

Ésta es la última fase del proceso pero puede intervenir en algunos aspectos del resto de las fases, fundamentalmente en el cálculo de los beneficios que se obtienen al considerar una repetición. La compresión obtenida depende del método de codificación empleado para escribir los enteros y también del que se usa para las bases.

Codificación de los enteros

Es uno de los puntos en los que más se ha trabajado para intentar reducir el tamaño de la codificación. El problema se reduce a escribir en el mínimo espacio un conjunto de enteros de los que se sabe bastante poco.

Lo primero que se desconoce es un rango suficientemente preciso. Es cierto que se puede hacer una estimación considerando que todos los enteros serán positivos y, como mucho, de la longitud de la secuencia. Sin embargo, esta estimación es muy mala puesto que los enteros más grandes son menos probables y una representación de longitud fija llevaría a un desperdicio de espacio. Hay que utilizar pues una codificación de enteros de longitud variable.

Este tipo de codificaciones puede llegar a ser muy complejo, como las basadas en los números de Fibonacci o en la rampa logarítmica, y esta complejidad se traduce en tiempo de cómputo que ralentiza la ejecuión en gran medida. Una solución sencilla y eficiente es la que se utiliza en el algoritmo dnaX [10] llamada Codificación de Bit de Continuidad (CBC). Se basa en dividir la codificación binaria del entero en grupos de bits de tamaño fijo e intercalar entre estos un bit que será un 0 entre los dos últimos grupos y un 1 en el resto. Se puede ver un ejemplo en el cuadro 4.1, donde se utiliza éste método formando grupos de tres bits y añadiendo el de continuidad.

Decimal	**Binario**	**CBC**
75	1001011	1001 1001 0011

Cuadro 4.1: Codificación de Bit de Continuidad

Aparte del método de codificación, se puede variar el dato en sí para hacerlo más pequeño. Esto es algo que hemos aplicado puesto que en lugar de escribir las posiciones de las repeticiones se escriben las diferencias con las posiciones de las repeticiones anteriores, de forma que escribir la resta entre dos enteros es siempre más económico que escribir cualquiera de ellos.

Lo único que permacece sin especificar de manera fija es el tamaño de los grupos que se utilizan en la codificación de bit de continuidad. Éste es un parámetro que hay que fijar de forma empírica y cuyo valor óptimo puede variar para cada fichero de ADN.

Codificación de las bases

Se sabe que las bases se pueden codificar utilizando 2 bits para cada una. La cuestión es si este límite se puede reducir. Para ello se han probado métodos estadísticos de compresión como pueden ser los códigos de Huffman, arítmetico o de rango descritos en [2, 11, 13]. El primero se muestra incapaz de bajar del límite puesto que las diferencias entre las frecuencias de las bases son mínimas. Los otros dos son de la misma familia y sí que consiguen en algunos casos reducir la codificación. El aritmético posee ciertas trabas por licencias así que se empleó el de rango para este proyecto.

El problema que presenta en la práctica es que en ocasiones supera el límite de 2 bits por base y realiza una codificación peor. Por eso es necesaria una comprobación y, en caso de detectar que no consigue la compresión, se utiliza la traducción directa a dos bits.

4.4. Consideraciones generales

Además de las directivas de diseño que se desprendían de los estudios de otros algoritmos ya existentes, se han tomado otras decisiones para incrementar las prestaciones del algoritmo desarrollado.

En cuanto a la mejora en velocidad se ha optado por limitar ciertos elementos para que se redujera el tiempo de cáculo. En este apartado cabría destacar la acotación de la búsqueda de repeticiones a las que que sean más largas que una cierta longitud mínima. Este límite inferior viene marcado por la fórmula de benificio, ya que no se aceptan las repeticiones que no aportan beneficio. Para ello se calcula para cada fichero cuál es la longitud mínima de las repeticiones en el mejor de los casos, es decir, con los demás datos de la referencia lo más pequeños posibles.

En el caso de la organización del fichero comprimido se contrastaron dos versiones distintas. En una, las referencias se almacenan separadas del resto de los elementos no referenciados. La otra opta por insertar una marca entre las bases no referenciadas que indique que allí hay una referencia. Esta segunda forma permite eliminar uno de los campos de las referencias sin embargo añade un nuevo símbolo al alfabeto lo que hace que 2 bits no sean suficientes para representarlos. Las pruebas demostraron que la primera opción era la que más comprimía.

Otro aspecto fundamental que se deseaba evitar era la parametrización. Gran parte de los algoritmos estudiados permitían variar ciertos parámetros para incrementar la compresión. Éstos parámetros no son intuitivos ni tienen relación directa con aspectos apreciables en la cadena de adn, de forma que cualquier usuario sería incapaz de decidir sin realizar varias pruebas cuál es el valor óptimo para cada parámetro. Dado que la compresión de una secuencia de ADN es una operación que se repite con poca frecuencia, se propuso que la elección del mejor valor la hiciera el propio algoritmo. Un ejemplo de este comportamiento es la búsqueda de la longitud mínima para las repeticiones, algo que otros algoritmos fijaban de forma estática.

4.5. Interfaz gráfica

La última tarea que se realizó en la fase de implementación del compresor fue dotarlo de una interfaz gráfica que facilitara la extracción de los resultados y la

comparación entre los distintos algoritmos.

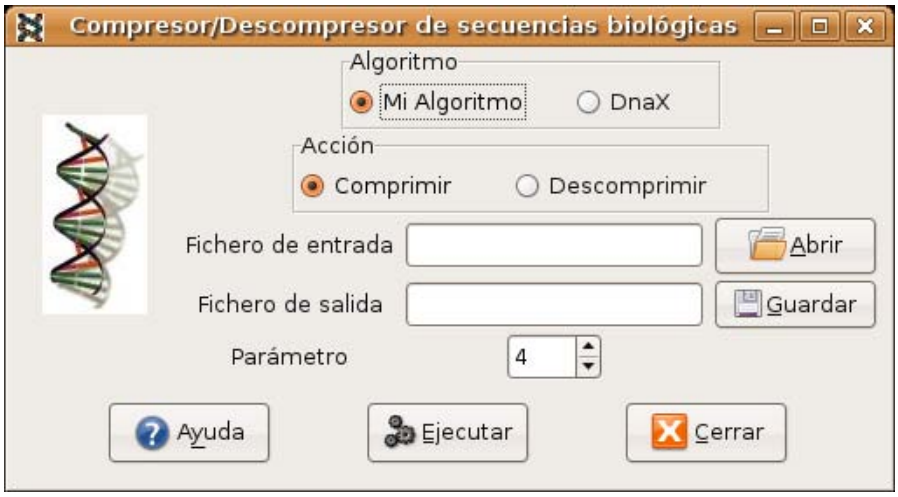

Figura 4.2: Ventana principal

Fundamentalmente consta de una ventana principal que nos muestra dos cuadros de selección donde se elige tanto el algoritmo que se va a usar como la acción (compresión o descompresión). También están los dos campos fundamentales correspondientes a los ficheros de entrada y salida del algoritmo. Por último, y en los casos que el algoritmo lo requiera, se puede especificar el valor de un parámetro que interviene en la codificación de los enteros. Por supuesto aparecen también los botones necesarios para que comience la acción elegida y el que provca que se acabe la ejecución del programa. Una imagen de esta ventana se puede ver en la figura 4.2.

Figura 4.3: Ventanas de estadísticas

Una vez concluida la ejecución de uno de los dos compresores, se muestra por pantalla en otra ventana la información del fichero que se ha comprimido y la

tasa de compresión que se ha alcanzado (en bits por base). Ésto permite realizar las pruebas de una forma sencilla y rápida. Se pueden ver unos ejemplo de estas ventanas en la figura 4.3.

5

Árboles filogénicos

En esta sección se tratan las características que debe tener un compresor de secuencias biológicas para poder ser usado como herramienta en la construcción de los árboles de filogenia.

Ya se ha comentado en secciones anteriores que los árboles filogénicos tienen como finalidad crear una estructura gráfica que represente las relaciones evolutivas de distintos individuos. En el caso de las secuencias biológicas, siven para conocer las ramas evolutivas que han llevado a la existencia de las distintas especies según la teoría de Darwin.

Todos los métodos que existen se podrían clasificar en dos grandes grupos dependiendo de cómo comparan las secuencias. El primer grupo utiliza toda la información de todas las secuencias para construir el árbol. El segundo, compara las secuencias dos a dos y obtiene una matriz numérica simétrica cuyos valores caracterizan la similitud entre las dos secuencias.

La intención del proyecto es la de utilizar el compresor como herramienta para generar el árbol así que se aprovechan las comparaciones que realiza el propio algoritmo descrito en la sección 4 para contabilizar la similitud entre dos secuencias. De esta forma, se construye una matriz de valores que luego servirá para generar el árbol.

5.1. Cálculo de la distancia entre dos secuencias

En el artículo [4] realizan una tarea similar aunque con pretensiones más teóricas, pues tratan de caracterizar la complejidad de Kolmogorov de una cadena tanto simple como condicionada a otra y usar los resultados existentes sobre este tema para conseguir un número que indique la similitud entre dos secuencias de ADN.

La medida que se usa en el proyecto como distancia entre dos secuencias es

básicamente la tasa de compresión que se alcanza al comprimir la concatenación de esas dos secuencias. Este método no asegura que la matriz sea simétrica pero experimentalmente se comprueba que la variación es practicamente nula entre las dos posibles permutaciones del par de secuencias. De todas formas, se ha considerado directamente la media de las tasas de las dos permutaciones como distancia entre la pareja de secuencias, lo cual asegura la simetría.

5.2. Características del compresor

Con estos planteamientos ya se pueden extraer algunas conclusiones de cómo debería ser el algoritmo de compresión para considerarlo como correcto en el cáculo de la similitud. Entre otras cosas, debe tener capacidad de almacenamiento de la parte de la secuencia ya procesada porque al tratar la parte correspondiente al segundo individuo, debe poder reconocer esos mismos elementos en la parte correspondiente al primero. Con este requerimiento se eliminan los compresores estadísticos como podrían ser los códigos de Huffman o la codificación aritmética.

Dentro de la familia de los algoritmos de Lempel-Ziv se deben excluir los que derivan del LZ78 porque el método que utiliza para crear el diccionario de las subsecuencias encontradas es muy inestable. Con esto me refiero a que la variación de una única base puede provocar la reconstrucción de todo el diccionario[1]. Por esa razón sólo se pueden considerar útiles aquellos algoritmos que almacenan las secuencias de una forma más consistente. En este grupo se incluyen por supuesto el DNACompress y el dnaX que, al ser específicos de las secuencias de ADN, recogen mejor las similitudes que los no dedicados como el gzip.

5.3. Métodos de cálculo

En este punto ya se dispone de una matriz de valores con las distancias entre los distintos individuos. Se presentan de nuevo multitud de métodos para conseguir la forma arbórea del esquema.

El más simple, busca la pareja más cercana, genera un individuo imaginario antecesor de ambas secuencias recalculando las distancias con el resto y repite el proceso hasta construir el árbol.

En el otro extremo están los que consideran todos los posibles árboles y se quedan con el más probable basándose en el mínimo cambio entre padre e hijo u otras leyes de evolución.

La elección del método de cálculo no es sencilla a priori puesto que la información que se maneja ya no es de tipo biológico sino matrices de números de las que

[1] Es un ejemplo de la catástrofe del bit [13, 12]

poca información se puede extraer. Por eso se tomó la decisión de utilizar varias alternativas y escoger la que mejores resultados diera.

5.4. Herramientas utilizadas

Como ya he dicho en la anterior sección, los métodos de construcción de árboles a partir de matrices numéricas están muy desarrollados por lo que se optó por usar algún paquete de software ya desarrollado. Se encontró de esta forma el paquete PHYLIP[2] del Departameto de Biología de la Universidad de Washington, que ofrece los algoritmos que se estaban buscando y otros que permiten la construcción gráfica del árbol filogénico. Los métodos que incorpora este paquete son los siguientes:

- Fitch
- Kitch
- Neighbor

El primero y el segundo calculan la filogenia a partir de la matriz de distancias usando el modelo que estima que las distancias deben igualar las sumas de las longitudes de las ramas de las distintas especies. Utilizan ambas el criterio de Fitch-Margoliash. La diferencia fundamental es que el método Kitch asume un reloj evolutivo lo que se traduce en que las longitudes totales de las ramas (desde la raiz a las hojas) deben ser iguales, pues son estas longitudes las que representan el tiempo. El método Fitch no considera esta restricción por lo que no se puede hacer ningún cálculo temporal sobre los resultados. El tercer método es el más simple de todos aunque también el más rápido y construye el árbol uniendo los *linages* más parecidos hasta llegar a unir todos. Por su simplicidad no considera tampoco reloj evolutivo por lo que no se pueden realizar cálculos temporales sobre sus resultados.

Las pruebas realizadas para comprobar qué método se ajustaba mejor a la construcción de árboles basados en cadenas de ADN y los resultados obtenidos se explican en detalle en la sección 6 de Resultados.

[2] http://evolution.genetics.washington.edu/phylip.html

6

Resultados

En esta sección se comentan las pruebas realizadas con el compresor y con los métodos de construcción de árboles de filogenia.

6.1. Resultados del compresor

En esta parte se especifican las pruebas realizadas al compresor y se expone una comparativa con los otros algoritmos de referencia utilizados.

6.1.1. Banco de pruebas

Las pruebas se han realizado en primer lugar sobre un banco de pruebas que es usado en todos los artículos que se han consultado. No se explica en ninguno de ellos la razón de su uso pero parece muy estandarizado. Los resultados sobre este banco de pruebas se pueden consultar en el cuadro 6.1. El tamaño está medido en número de bases y el nivel de compresión indicado en cada casilla se refiere a los bits por base que ocupa el fichero obtenido.

Se puede comprobar que el algoritmo dnaX es mejor que el resto en prácticamente todos los ficheros. Al comprobar estos resultados y volver sobre el artículo que define este algoritmo [10], se sospechó que el algoritmo estaba hecho prácticamente a la medida de este banco de pruebas así que se decidió construir otro banco de pruebas independiente para comprobar si mantenía el mismo rendimiento.

El segundo banco de pruebas (Cuadro 6.2), se construyó cogiendo una secuencia genética de cada una de las clases en que las divide la base de datos NCBI [1] buscando a la vez diversidad en las longitudes de las secuencias.

[1] http://www.ncbi.nlm.nih.gov/

Secuencia	Tamaño	dnaX	DNACompress	Mi Algoritmo
chmpxx	121024	1.68	1.84	1.96
chntxx	155844	1.62	1.93	2.00
hehcmvcg	229354	1.85	1.97	2.00
humdystrop	38770	1.95	1.92	1.99
humghcsa	66495	1.38	1.93	1.32
humhprtb	56737	1.92	1.92	1.97
mpomtcg	186608	1.93	1.96	1.97
vaccg	191737	1.76	1.91	1.93

Cuadro 6.1: Banco de pruebas ajeno

Secuencia	Tamaño	dnaX	DNACompress	Mi Algoritmo
Aeromonas	11822	2.05	1.96	2.00
Anisakis	13916	1.84	1.77	1.85
Citrus Dwarf	294	3.95	2.37	2.12
Deinococcus	412344	1.87	1.88	1.98
Methanohalophilus	2158	2.44	1.89	2.02
Nanoarchaeum	490885	1.85	1.86	1.98
Plasmodium	643292	1.59	1.67	1.68
Porphyra Pluchra	6427	2.11	1.95	1.98
Virus de la Rabia	11932	2.07	1.97	2.00
Salmonella	46900	2.00	1.98	2.00

Cuadro 6.2: Banco de pruebas propio

6.1.2. Análisis de los resultados

El análisis realizado a los datos tenía como objetivos fundamentales dos aspectos sobre los algoritmos:

- Comparar los niveles de compresión de los distintos algoritmos.
- Averiguar si la eficiencia de los algoritmos era similar en los distintos bancos de pruebas.

Para conseguir estos objetivos se recurrió a realizar un estudio estadístico de los resultados.

Elección del estadístico

Este es un punto fundamental a la hora de realizar el análisis pues afecta a todo el proceso. Se presenta una duda entre dos opciones al elegir el estadístico más correcto.

La primera opción es la más simple y utiliza como elemento de medida la tasa de compresión (bits/base) obtenida en cada ejecución del algoritmo. Parece ser la más sencilla e intuitiva pero hay casos en los que perjudica de forma excesiva a cierto algoritmo. Si se observan los datos correspondientes al *Citrus Dwarf* del Cuadro 6.2, destaca de forma contundente el resultado correspondiente al algoritmo dnaX, que obtiene una tasa de 3.95 cuando el límite superior permitido es de 2 bits por base. Sin embargo, dado que el tamaño de esa secuencia es muy pequeño, sólo 294 bases, esto representa una pérdida de apenas 574 bits, lo que representa una cantidad nimia si tenemos en cuenta la ganancia que se obtiene en otros ficheros más grandes.

La segunda opción recoge este caso de forma que se considera como medida de compresión la tasa de compresión de todos los ficheros juntos, es decir, el total del tamaño comprimido dividido para el total de bases. De esta forma se diluye el nefasto resultado antes mencionado pues su ponderación en el resultado final es prácticamente despreciable.

Sin embargo esta segunda opción no refleja con exactitud todos los aspectos que intervienen en la compresión. Además de la obligatoria cabecera que se ha de introducir en los ficheros comprimidos y que hace que las tasas de los ficheros pequeños sean peores, la decisión de poner una longitud mínima fija para las repeticiones también influye en la capacidad de compresión, perjudicando de nuevo a los ficheros pequeños. Dado que este factor era uno de los tratados en la implementación realizada en este proyecto, parece fundamental que se vea reflejado en el análisis. Por eso se eligió la tasa de compresión de cada fichero como estimador.

Comparativa de los niveles de compresión

La cuestión que se aborda con este análisis es averiguar si el algoritmo implementado era igual, mejor o peor que los otros dos que se han tomado como referencia. Para conseguir ésto, la prueba más conveniente es una comparación de las medias de las tasas, basada en la t de Student, teniendo en cuenta que los resutados de ambos algoritmos provienen de los mismos datos, lo que es equivalente a decir que los datos están emparejados.

Con los resultados presentados en el Cuadro 6.3 se puede afirmar que las medias son iguales para las dos muestras a un nivel de confianza de 95 %. Es decir, en media, tanto el algoritmo implementado como el basado en Pattern Hunter obtienen la misma tasa de compresión.

	Mi Algoritmo	DNACompress
Media	1.93	1.93
Varianza	0.03	0.02
Observaciones	18	18
Grados de libertad	17	
Estadístico t	0.10	
P(T<=t) una cola	0.46	

Cuadro 6.3: Prueba t de Student de igualdad de medias 1

	Mi Algoritmo	dnaX
Media	1.93	1.99
Varianza	0.03	0.29
Observaciones	18	18
Grados de libertad	17	
Estadístico t	0.55	
P(T<=t) una cola	0.29	

Cuadro 6.4: Prueba t de Student de igualdad de medias 2

Con los datos del Cuadro 6.4 se puede concluir que ambas muestras tienen medias iguales a un nivel de confianza de un 95 %. Por lo tanto, el algoritmo que se ha implementado alcanza el nivel de compresión del dnaX en media.

Se puede decir entonces que el algoritmo que se ha implementado consigue al menos las mismas tasas de compresión que los tomados como referencia, el dnaX y el DNACompress, y que incluso supera al primero en algunos ficheros.

Comparación entre los dos bancos

Se pretende ahora comprobar si los algoritmos son igual de eficientes si se cambian los ficheros que se desean comprimir. Uno de los motivos de la creación de otro banco de pruebas era la sospecha de que los algoritmos estaban excesivamente basados en unos ficheros concretos y que podían perder efectividad con otras secuencias genéticas.

Para comprobar la independencia del algoritmo a los ficheros de pruebas, comparamos los resultados obtenidos con un banco y con el otro usando un test de la t de Student. En este caso los datos no están emparejados porque proceden de datos diferentes. Para la correcta realización de este test se debe conocer si las varianzas de las dos muetras a comparar son iguales o no. Para solucionar esto, se utiliza la

prueba F para varianzas de dos muestras.

	Mi Algoritmo		**dnaX**		**DNACompress**	
	B-1	**B-2**	**B-1**	**B-2**	**B-1**	**B-2**
Media	1.89	1.96	1.93	1.99	1.92	1.93
Varianza	0.05	0.01	0.04	0.44	0.001	0.03
Obs.	8	10	8	10	8	10
f	3.87		0.09		0.05	
P(F<=f)	0.03		0.002		0.0002	

Cuadro 6.5: Prueba de igualdad de varianzas

Según nos indica la fila de probabilidades del Cuadro 6.5, se deben considerar las varianzas de las dos muestras distintas en los tres casos que se tratan aquí con un nivel de confianza del 95 %. Por tanto la prueba que se debe realizar es la de comparación de muestras con varianzas distintas.

	Mi Algoritmo		**dnaX**		**DNACompress**	
	B-1	**B-2**	**B-1**	**B-2**	**B-1**	**B-2**
Media	1.89	1.96	1.93	1.99	1.92	1.93
Varianza	0.05	0.01	0.04	0.44	0.001	0.03
Obs.	8	10	8	10	8	10
Estadístico t	-0.76		-1.89		-0.13	
P(T<=t)	0.23		0.04		0.45	

Cuadro 6.6: Prueba t de Student de igualdad de muestras

Con los resultados que aparecen en la fila de probabilidades del Cuadro 6.6 podemos considerar que tanto mi algoritmo como DNACompress funcionan de igual manera en los ficheros de ambos bancos mientras que queda probado que el comportamiento del algoritmo dnaX con los dos grupos de secuencias no es el mismo.

6.2. Resultados de los árboles de filogenia

En esta sección se describen los ficheros de secuencias de ADN escogidos para realizar estas pruebas y los árboles obtenidos con los distintos métodos.

6.2.1. Corrección de las pruebas

Ya se ha explicado en la sección dedicada a los árboles de filogenia que las teorías sobre éstos son múltiples y muchas de ellas tienen bastante aceptación. Esto impide saber de una forma sencilla si un árbol filogénico es correcto o no. Las dos formas que menos complejidad presentan son las siguientes:

- Crear un árbol evolutivo artificial propio y comprobar si se puede recuperar apartir de los miembros separados.
- Utilizar un árbol cuya distribución sea conocida e intentar obtenerlo.

La primera de ellas es la más fiable puesto que el árbol es una creación propia y por lo tanto no hay duda de si es correcto o no. Sin embargo y dado el caso de los ficheros de ADN, esta opción se presentaba aún muy compleja si se pretendía obtener cierto tamaño en las secuencias de bases de los distintos individuos. Por eso se optó por la segunda, que puede presentar alguna duda sobre la corrección pero es mucho más simple.

6.2.2. Ficheros de prueba

Una vez decidido que se iba a usar un árbol filogénico real, había que decidir qué ficheros de ADN representarían a los indivíduos. Leyendo los distintos artículos sobre filogenia, parece que es muy común usar las secuencias de un cierto material genético llamado ADN mitocondrial para establecer la filogenia entre los distintos seres vivos eucariotas, es decir, los que poseen este tipo de secuencias.

Para hacerlo más sencillo se optó por escoger únicamente los mamíferos puesto que son más conocidos y se puden apreciar los errores de forma más sencilla. Además, una clasificación similar ha sido usada en [6], lo que ofrece otro elemento de comparación para verificar el correcto funcionamiento de los métodos usados.

El software utilizado[2] limitaba también en cierta medida la cantidad de individuos que se podían utilizar para la generación de los árboles, por lo que se eligió un subconjunto de los que se habían usado en el artículo consultado.

Preparación de los ficheros

Como se ha explicado en la sección 5, los métodos de construcción de árboles filogénicos usados son los basados en matrices de valores. Por eso se creó la matriz que determinaba las distancias entre los distintos individuos, calculando la media de las tasas de compresión obtenidas durante el procesado de los ficheros creados al concatenar las secuencias de ADN de dos individuos, en las dos permutaciones posibles. Los valores correspondientes a la diagonal se fijaron a 0 por conveniencia.

[2] http://evolution.genetics.washington.edu/phylip.html

6.2.3. Algoritmos y métodos utilizados

Para comprobar la bondad de los métodos y algoritmos utilizados para la creación de los árboles, se realizaron pruebas tanto con los que se esperaba buen resultado como con los que se podía suponer que no conseguirían el objetivo buscado.

Los algorimos de compresión utilizados para las prueba fueron:

- Codificación aritmética
- Códigos de Huffman
- RLE
- Gzip
- 7zip
- El algoritmo creado

Los tres primeros algoritmos no se esperaba que funcionasen correctamente puesto que ni siquiera consideraban repeticiones largas dentro del fichero. Un salto de calidad se creía que se obtendría con los basados en los algoritmos de Lempel-Ziv aunque sin llegar a los resultados proporcionados por los algoritmos específicos de ADN como el que se ha implementado.

En cuanto a los métodos numéricos que permiten la generación del árbol a partir de la matriz de distancias, dado que se desconocía cuáles podrían dar buenos o malos resultados, se utilizaron los que ofrecía el paquete de software utilizado, que resultaban ser estos tres:

- Fitch
- Kitsch
- Neighbour

Los tres se explican en la parte dedicada a los árboles filogénicos (sección 5).

6.2.4. Resultados obtenidos

Se muestran aquí algunos de los árboles generados indicando cuáles son los más correctos en función del resultado.

El primero que se muestra en la figura 6.2.4 es el que resulta del uso del algoritmo RLE para la compresión de los distintos ficheros. Se puede observar que la estructura que consigue no refleja las familias en las que se dividen los distintos individuos. Este comportamiento se observa en todos los algoritmos de compresión

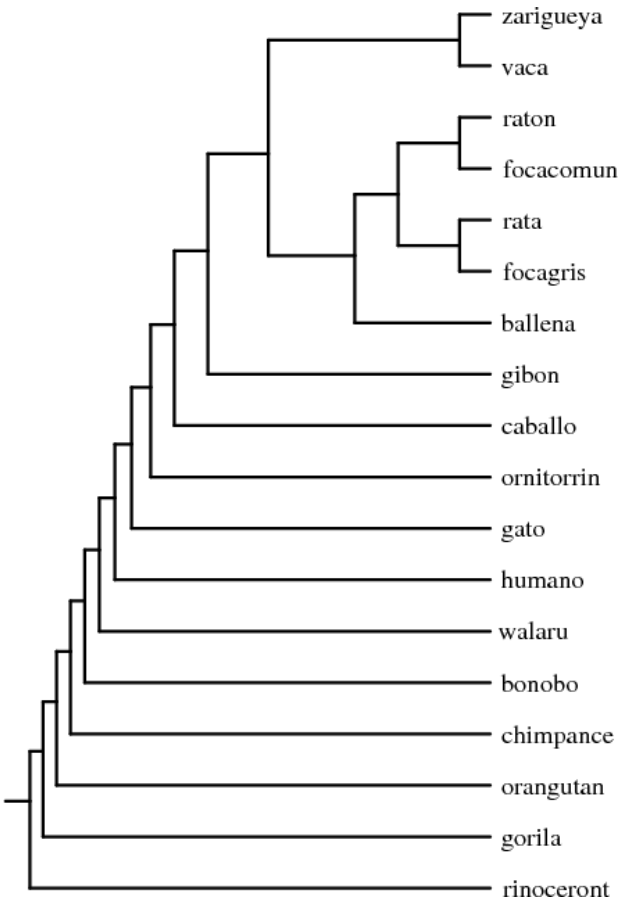

Figura 6.1: Árbol generado por el algoritmo RLE

de tipo estadístico puesto que no poseen buenos diccionarios que permitan reflejar las similitudes entre las cadenas biológicas.

En segundo lugar se puede ver el árbol generado por los algoritmos de la familia Lempel-Ziv (figura 6.2.4). Se puede apreciar cómo los individuos están organizados bajo cierta jerarquía de familias, sin embargo algunas especies no llegan a entrar en esta clasificación y quedan como líneas de evolución separadas.

El último de los casos es el correspondiente al algoritmo de compresión de secuencias de ADN (figura 6.2.4) y posee una completa estructuración de los individuos según las familias que se definen en la biología clásica. Las cuatro grandes ramas corresponderían, de izquierda a derecha, a las familias de los ungulados, primates, roedores y marsupiales.

Se observa pues que se obtiene el resultado deseado con el algoritmo implementado, al menos en las grandes clasificaciones. La estructura a niveles más pequeños

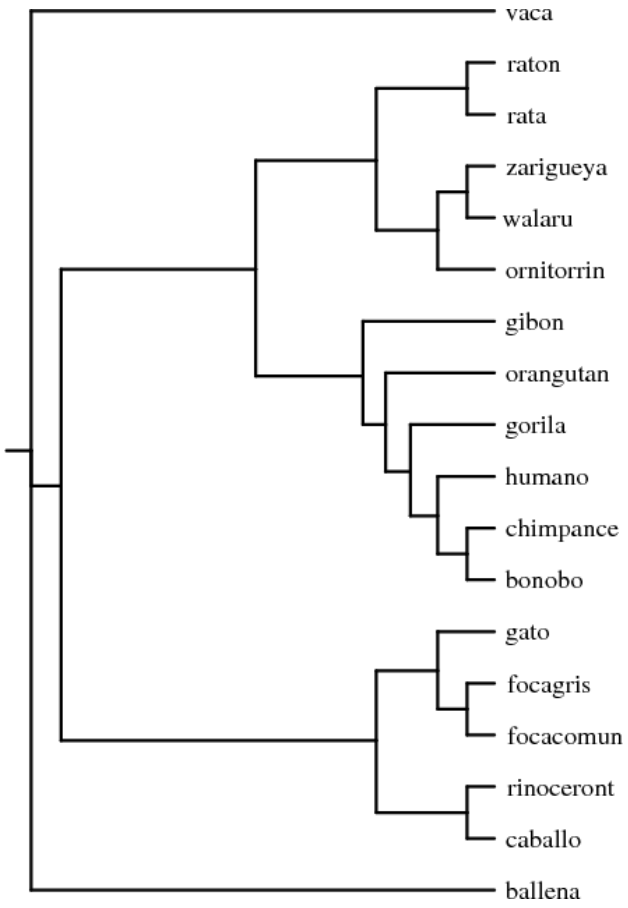

Figura 6.2: Árbol generado por el algoritmo gzip

se deben en gran medida a los métodos numéricos de construcción del árbol. El que mejor resultado a dado en las pruebas realizadas ha sido el *kitsch*, debido fundamentalmente a que en los cálculos que realiza no considera las distancias por pares únicamente sino que tiene en cuenta las distancias con el resto de una manera más determinante que los otros dos métodos.

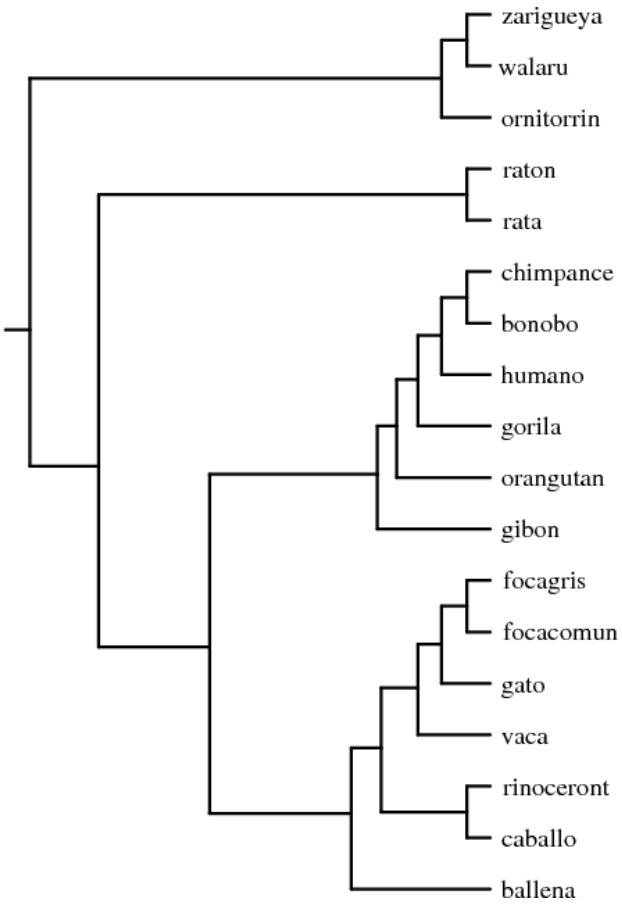

Figura 6.3: Árbol generado por el compresor de ficheros de ADN

7

Conclusiones

En esta parte se resumen las conclusiones obtenidas de la elaboración del proyecto, tanto las referentes a la propia realización del proyecto como las personales. Además se plantean posibles líneas para continuar con el trabajo realizado.

7.1. Conclusiones sobre el compresor

Se puede decir que los algoritmos dedicados a la compresión de secuencias de ADN deben tener ciertas propiedades que los hagan realmente útiles. Entre las principales encontramos las siguientes:

- No se pueden construir algoritmos basados en elementos funcionales puesto que en el punto en que se encuentra la investigación en genética no se dispone de suficiente información para poder dividir de alguna forma concreta las secuencias de ADN.

- Los algoritmos deben de estar basados en diccionarios para la búsqueda de repeticiones en las cadenas pues es ésta la forma de conseguir la compresión deseada. Además deben incluir la capacidad de detección de repeticiones invertidas, muy comunes en estas secuencias.

- Debido a que el uso de las secuencias hace que la operación de compresión sea poco común, merece la pena utilizar métodos exhaustivos que mejoren la tasa compresión, aun a riesgo de aumentar el tiempo de ejecución.

- La codificación de los datos del archivo comprimido debe ser especialmente cuidada puesto que el límite de compresión es de por sí muy bajo y obliga a realizar un gran esfuerzo para conseguir una tasa mejor de compresión.

- Se debe evitar la parametrización de los algoritmos al máximo por dos razones fundamentales. La primera es que la diversidad de las cadenas de ADN hace casi imposible encontrar un conjunto de valores óptimo en todos los casos para los parámetros del algoritmo. La segunda es que los parámetros que se manejan en los algoritmos no tienen correspondencia directa con características que se puedan extraer de forma directa de la cadena, obligando al uso de prueba y error para fijar los valores correctos.

Como líneas de continuación del proyecto referentes al compresor se pueden considerar las siguientes:

- Mejorar en el aspecto temporal y de uso de memoria el algoritmo implementado. Éste no era un objetivo del proyecto pero se hace imprescindible si se quiere conseguir que el algoritmo propuesto sea funcional para todas las secuencias de ADN existentes.

- Continuar con la eliminación de todos los parámetros del programa de forma que el propio compresor elija en su ejecución el óptimo. Por supuesto esta capacidad va en relación con la anterior propuesta puesto que el proceso de reducir el número de parámetros afecta en gran medida al tiempo de ejecución.

7.2. Conclusiones sobre los árboles de filogenia

Tras la realización de esta parte del trabajo se han llegado a las siguientes conclusiones:

- Para conseguir las estructuras deseadas en los árboles de filogenia no sirve cualquier compresor. De hecho, es necesario que el compresor refleje en buena medida las características de los datos que se manejan para obtener resultados buenos.

- Los métodos númericos que se usan para la construcción del árbol a partir de la matriz también deben de cuidarse, siendo importante que utilice el máximo de información para la toma de decisiones y así alcanzar un resultado óptimo.

En lo que se refiere a posibilidades de continuar con el trabajo en los árboles de filogenia, hay varios puntos que se podrían tener en cuenta:

- Realizar pruebas más exhaustivas, creando un árbol propio de filogenia y tratar de reconstruirlo con los métodos explicados aquí.

- Investigar en los posibles significados del uso de los distintos métodos numéricos en relación con la evolución y extraer de allí el que pudiera ajustarse de mejor forma a la teoría.

7.3. Conclusiones personales

Personalmente valoro muy positivamente esta experiencia, principalmente, debido a que este proyecto requería una parte muy importante de investigación, relegando aspectos más mecánicos a un segundo plano.

También considero importante el haber podido colaborar o al menos consultar otras personas que trabajan sobre este mismo tema en el resto del mundo. Además he apreciado el distinto enfoque que le da cada uno, más académico o más comercial, influyendo luego en el grado de ayuda que han prestado.

En un aspecto más técnico, he tenido la posibilidad de aprender multitud de algoritmos y estructuras de datos nuevas para mi que han ampliado en gran medida mis conocimientos sobre este tema.

Bibliografía

[1] Alberto Apostolico and Aviezri S. Fraenkel. Robust Transmossion of Unbounded Strings Using Fibonacci Representations. In *IEEE transactions on information theory*, volume it-33. IEEE, Marzo 1987.

[2] Eric Bodden, Malte Clasen, and Joachim Kneis. Arithmetic Coding in revealed. In *Proseminar Datenkompression 2001*. RWTH Aachen University, 2002. German version available: Proseminar Datenkompression, Arithmetische Kodierung.

[3] Javier Campos. Material de clase de Técnicas Avanzadas de Programación, 2004/2005. http://webdiis.unizar.es/asignaturas/TAP/.

[4] Xin Chen, Sam Kwong, and Ming Li. A Compression Algorithm for DNA Sequences and Its Applications in Genome Comparison. In *Proceedings of the 4th Annual International Conference on Computational Molecular Biology*, 2000.

[5] Xin Chen, Ming Li, Bin Ma, and John Tromp. *DNACompress: fast and effective DNA sequence compression*, volume 18, pages 1696–1698. Bioinformatics, 2002.

[6] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. The Similarity Metric. In *IEEE transactions on information theory*, volume 50. IEEE, Diciembre 2004.

[7] Ming Li, Bin Ma, Derek Kisman, and John Tromp. PatterHunter2: Highly Sensitive and Fast Homology Search, 2004.

[8] Bin Ma, John Tromp, and Ming Li. *PatternHunter: faster and more sensitive homology search*, volume 18, pages 440–445. Bioinformatics, 2002.

[9] Udi Manber and Gene Meyers. Suffix arrays: A new method for on-line string searches. Technical report, University of Arizona, 1989.

[10] Giovanni Manzini and Marcella Rastero. A Simple and Fast DNA Compressor, 2004.

[11] G. N. N. Martin. Range encodig: an algorithm for removing redundancy from a digitised message, Marzo 1979. Presented to the Video & Data Recording Conference, Southampton.

[12] Elvira Mayordomo. El algoritmo de compresión de datos LZ78, 2003.

[13] Elvira Mayordomo. Las técnicas de compresión de datos sin pérdida de información (lossless). Un resumen personal, 2003.

[14] Bořivoj Melichar. Repetitions in Text and Finite Automata.

[15] Raquel Moreno-Loshuertos, Rebeca Acín-Pérez, Patricio Fernández-Silva, Nieves Movilla, Acisclo Pérez-Martos, Santiago Rodriguez de Cordoba, M Esther Gallardo, and José Antonio Enríquez. *Differences in reactive oxygen species production explain the phenotypes associated with common mouse mitochondrial DNA variants*, pages 1261 – 1268. Number 38. Nature Genetics, Noviembre 2006.

[16] Roderic Page. Introduction to Tree Building.

[17] Klaus-Bernd Schürmann and Jens Stoye. An Incomplex Algorithm for Fast Suffix Array Construction.

[18] Hreinn Stefansson, Agnar Helgason, Gudmar Thorleifsson, Valgerdur Steinthorsdottir, Gisli Masson, John Barnard, Adam Baker, Aslaug Jonasdottir, Andres Ingason, Vala G Gudnadottir, Natasa Desnica, Andrew Hicks, Arnaldur Gylfason, Daniel F Gudbjartsson, Gudrun M Jonsdottir, Jesus Sainz, Kari Agnarsson, Birgitta Birgisdottir, Shyamali Ghosh, Adalheidur Olafsdottir, Jean-Baptiste Cazier, Kristleifur Kristjansson, Michael L Frigge, Thorgeir E Thorgeirsson, Jeffrey R Gulcher, Augustine Kong, and Kari Stefansson. *A common inversion under selection in Europeans*, pages 129 – 137. Number 37. Nature Genetics, Febrero 2005.

[19] Esko Ukkonen. *On-line construction of suffix trees*, pages 249–260. Number 14. Algorithmica, 1995.

[20] Christina Zeeh. The Lempel Ziv Algorithm, January 2003. Seminar "Famous Algorithms".