UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS
FACULTAD DE MATEMÁTICA FÍSICA Y COMPUTACIÓN

PROGRAMACIÓN PARALELA DE ALGORITMOS BÁSICOS EMPLEADOS EN EL ANÁLISIS FILOGENÉTICO

Trabajo de Diploma

Autor: Jorge Enrique Moreira Broche

Tutores: Dra. María del Carmen Chávez Cárdenas
Dr. Robersy Sánchez Rodríguez

Consultantes: Dr. Ricardo Grau Ábalo
MSc. Leonardo Flavio del Toro Melgarejo

Santa Clara
2010

DICTAMEN

Hago constar que el presente trabajo fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____
Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____
Firma del tutor

_____
Firma del Jefe del Seminario

PENSAMIENTO

Nuestra ciencia es para la paz y el desarrollo, no para la guerra y la destrucción.

Fidel Castro Ruz

DEDICATORIA

A mi abuelo, porque ni siquiera la muerte pudo separarlo de nosotros

AGRADECIMIENTOS

Deseo agradecer a todas aquellas personas que de una forma u otra contribuyeron a que esta tesis fuera llevada a buen término.

De manera muy especial les doy las gracias a mis tutores, al profesor Grau y a los demás miembros del Grupo de Bioinformática por su invaluable apoyo cada vez que lo necesité.

Muchas gracias también a mi familia, sobre todo a mi madre, por su apoyo incondicional en todo momento.

Por último, pero no menos importantes, quiero agradecer a mis amigos, los de antes y los de ahora, porque con solo su presencia me dieron los ánimos necesarios para continuar trabajando.

RESUMEN

La existencia de grandes cantidades de información acerca de las secuencias genéticas de varias especies presenta un gran reto para la comunidad científica en su afán de extraer conocimientos de valor de las mismas. Las herramientas disponibles actualmente que permiten el análisis y comparación de estas secuencias, resultan ineficientes cuando se enfrentan a grandes volúmenes de datos debido al alto costo computacional asociado a la mayoría de los algoritmos que se utilizan con este fin. El pronóstico de nuevas mutaciones de influenza, que aborda el grupo de Bioinformática de la UCLV es un ejemplo de estos retos computacionales.

Este trabajo propone una implementación paralela en C++ de los algoritmos básicos del análisis filogenético usando el paradigma de paso de mensajes, con el objetivo de lograr una disminución sustancial de sus tiempos de ejecución. Se realizó un análisis de complejidad computacional de los algoritmos mencionados y se aplicaron las técnicas de programación paralela en aquellas etapas en que resultó propicio hacerlo.

La aplicación desarrollada fue puesta a prueba en un clúster de 6 computadoras y en una máquina con un procesador de 4 núcleos, comparando su desempeño con el de la implementación en Matlab ya existente y con el de otra implementación secuencial en C++. Para ello se utilizaron dos alineamientos de las secuencias de un gen del virus de la Influenza. Las pruebas realizadas permitieron comprobar la superioridad significativa de la implementación en paralelo frente a sus contrapartes secuenciales en el procesamiento de grandes cantidades de datos.

ABSTRACT

The existence of large amounts of information about the genetic sequences of several species presents a great challenge to the scientific community in its attempts to extract valuable knowledge from it. The available tools that currently allow the analysis and comparison of these sequences, behave inefficiently when faced to big volumes of data due to the high computational cost associated with most of the algorithms used with this purpose. The prediction of new mutations of influenza, which addresses the Bioinformatics Group at UCLV is an example of this computational challenges.

This work proposes a parallel implementation in C++ of the basic algorithms of phylogenetical analysis using the message-passing paradigm, with the objective of a substantial decrease of its execution times. A computational complexity analysis of the above mentioned algorithms was made and parallel programming techniques were applied in those stages in which it resulted propitious.

The developed application was tested in a 6 computers cluster and in a computer with a 4 cores processor, comparing its performance with that of the already existent Matlab implementation and another sequential implementation in C++. Two alignments of a gene of the Influenza virus were used with this purpose. The tests carried out proved the significant superiority of the parallel implementation when compared to its sequential counterparts in the processing of large amounts of data.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En la actualidad, la comunidad científica está fijándose nuevos objetivos, que comprenden la utilización de la gran cantidad de información que ha sido generada hasta el momento para su empleo en los análisis estructurales y funcionales de los genomas. Ahora, con la secuenciación completa del genoma humano y de otras especies puestas al alcance de los investigadores, el análisis detallado de las secuencias genómicas indudablemente mejorará el entendimiento de los sistemas biológicos, lo cual requiere de sofisticadas herramientas bioinformáticas y computacionales. De esta forma estamos asistiendo al paso de la "Era Genómica" a una "Era post-Genómica" en la cual, se va a analizar y comparar los genomas y cuáles son las relaciones existentes entre su estructura y su función, valiéndose de las herramientas bioinformáticas desarrolladas para este propósito (Yu, Lee et al. 2004).

Un aspecto clave dentro de las nuevas tendencias post-genómicas es la Genómica Comparativa, que analiza similitudes y diferencias de los genomas de distintos organismos, tanto estructurales como funcionales (Cliften 2004; Yu, Lee et al. 2004). La disponibilidad de muchos genomas completos ofrece la oportunidad de realizar la comparación de genomas de especies filogenéticamente relacionadas y distantes. Esta comparación no resulta trivial pues, por ejemplo, el genoma humano y el del chimpancé son comparables en el 96 % de su longitud siendo estas regiones idénticas en un 99 % (Consortium 2005). El número de diferencias genéticas entre los humanos y los chimpancés es aproximadamente 60 veces menor que entre los humanos y los ratones y unas 10 veces menos que entre los ratones y las ratas. Al mismo tiempo, la cantidad de disparidades genéticas entre un hombre y un chimpancé es unas 10 veces más que entre dos personas cualesquiera. De hecho, ambos genomas son idénticos casi un 99 % si no se tienen en cuenta en el análisis ciertos aspectos del ADN que se han reorganizado de forma distinta en las dos especies. Pero si se consideran las sustituciones de nucleótidos o bases, que difieren en el 1,23%, y otras variaciones como las repeticiones que ocurren en casi el 3%, las similitudes entre las secuencias de ADN de ambas especies sólo llegan al 96%. Luego, se origina la siguiente pregunta: ¿Si las similitudes entre el genoma del chimpancé y el del ser humano son tan grandes, entonces, por qué desde el punto de vista biológico somos tan diferentes? Desde este punto de vista se buscan las

diferencias en el análisis comparativo de pequeñas regiones de genes codificantes para proteínas vinculadas a funciones especiales del sistema nervioso y a enfermedades (Pennisi 2007). Evidentemente deben existir grandes diferencias en la arquitectura organizativa de estos genomas, las cuales no son fácilmente perceptibles utilizando las herramientas actuales simples de la Bioinformática.

Antecedentes y planteamiento del problema

A finales de 2008 se publicaron algunos criterios sobre los pronósticos de una eventual epidemia de Influenza A, que según se esperaba se desataría muy pronto. A nivel mundial, la mayoría de los científicos coincidía en que la cepa que tenía mayor probabilidad de generar una nueva pandemia era la H5N1. El Laboratorio de Bioinformática de la UCLV organizó entonces varios seminarios al respecto, convocando a otros laboratorios del Centro de Estudios de Informática, para seguir tales estudios.

Al mismo tiempo en el mencionado laboratorio se daban los toques finales a la publicación del nuevo modelo evolutivo sobre 5 bases, por lo que se decidió hacer algunas pruebas de este construyendo árboles filogenéticos a partir de los genomas de algunos mutantes del virus H5N1 reportados en Internet y la distancia definida en el nuevo modelo, para comparar el árbol consenso con aquel obtenible con el modelo clásico de Tamura-Nei (Tamura and Nei 1993). Esta aplicación del nuevo modelo fue publicada junto a otros ejemplos en (Sánchez and Grau 2009).

Tan pronto se declaró oficialmente la epidemia de la cepa H1N1, comenzó a trabajarse en esta. Se descargaron más de un centenar de secuencias del genoma completo, y muchas más secuencias de dos de sus segmentos más importantes por ser los principales sitios antigénicos del virus: la Hemaglutinina (HA) y la Neuramidasa (NA). Usando el nuevo modelo se construyeron árboles filogenéticos que en principio permitieron clasificar los linajes a nivel mundial, e inmediatamente el grupo se planteó la posibilidad de hacer predicciones de futuras mutaciones.

Con este objetivo se concibió un procedimiento, que en esencia calcula las distancias evolutivas de las secuencias previamente alineadas para construir el árbol consenso que describe los orígenes. Con las estimaciones de la distribución de probabilidades estacionarias de las 5 bases y las razones de cambio de cada una, que constituyen los parámetros

fundamentales del modelo, se hacen predicciones (*backward*) de los ancestros de las secuencias reales, y luego a partir de ellos, se hacen las predicciones (*forward*) de las futuras mutaciones en diferentes escalas de tiempo evolutivo.

Se usaron productos de software libre de bioinformática para el alineamiento de las secuencias y herramientas profesionales para la construcción de los árboles. Los cálculos de los parámetros del modelo fueron implementados en el MatLab adaptando para ello la herramienta MBEToolbox al modelo de 5 bases. Los primeros resultados positivos se obtuvieron en julio del 2009 y fueron de inmediato interés a varios centros del Polo del Oeste, pero especialmente al Centro Nacional de Sanidad Agropecuaria (CENSA) quien está encargado de perfeccionar constantemente el sistema de diagnóstico para este y otros virus.

En noviembre del 2009, después de intensos procesamientos, se habían pronosticado 100 millones de mutaciones de la HA, no todas diferentes entre sí. Se entregaron al CENSA las 20 mil variantes mutacionales que más frecuentemente aparecieron y además información sobre la conservación de los sitios para que pudieran comenzar a analizar la posible efectividad futura del sistema de diagnóstico actual que puede ser afectado por las mutaciones.

Lejos de terminar, el verdadero trabajo comenzaba allí por varias razones:

1. Puede parecer que 100 millones de predicciones son suficientes para que los resultados sean confiables; pero no es así. Téngase presente que en una gotita de líquido nasal puede haber varios miles de viriones. ¿Cuántas mutaciones pueden estar presentes en un estornudo de un solo paciente? Definitivamente, estimaciones confiables exigen cálculos con el orden mínimo del número de Avogadro (10^{23}).
2. Las predicciones tienen que ser constantemente actualizadas, en la misma medida en que aparecen nuevas secuenciaciones reportadas. Por tanto todo el procedimiento tiene que repetirse periódicamente.
3. El CENSA, solicitó en particular, la posibilidad de realizar el trabajo considerando especialmente las cepas del linaje americano, pues ellos tienen responsabilidad a nivel del continente en el análisis y perfeccionamiento del diagnóstico de varios virus.
4. Por la razón anterior, y motivado por los resultados, el CENSA solicita además que el trabajo se repita con las cepas H5N1, H7N1 y el virus de la Peste Porcina Clásica. La UCLV está interesada además en continuar sus pronósticos sobre el VIH

El gran problema es el tiempo computacional que requiere el procedimiento y las posibilidades reales de hacerlo sin interrupciones por falta de fluido eléctrico por muy bueno que sea el equipamiento. Para citar un ejemplo, el alineamiento de las cepas del linaje americano que solicitó el CENSA fue intentado 7 veces y frustrado por falta de energía eléctrica, al cabo de entre 7 y 15 días de procesamiento. Sólo se logró terminarlo cuando pudo ubicarse una buena microcomputadora en un local protegido por un grupo electrógeno y requirió aproximadamente un mes y medio. El procedimiento de estimación de las tasas que sigue a continuación requiere también un tiempo considerable. Pero incluso, aún cuando en situaciones idóneas no se tuviera este tipo de afectaciones, ¿cómo reducir los tiempos de análisis y respuesta?

Es así que surge este trabajo, con el cual se busca disminuir los tiempos de ejecución de estos algoritmos hasta el punto en que resulte posible su ejecución en las condiciones normales de la UCLV sin que las interrupciones del servicio eléctrico representen el problema tan serio que han sido hasta ahora. Una disminución significativa de los tiempos de ejecución también permitiría elevar considerablemente la capacidad del Grupo de Bioinformática de la UCLV para analizar las nuevas mutaciones y posibles predicciones, incluyendo los procesamientos que las mismas requieren.

Desde el punto de vista computacional, los algoritmos básicos empleados en el análisis filogenético han sido ampliamente implementados en múltiples paquetes de software destinados enteramente a esta temática (Kumar and Dudley 2007). Sin embargo, la mayoría de estos resultan inadecuados cuando se pretende analizar grandes volúmenes de datos, debido a su alto costo computacional en cuanto al tiempo de ejecución. Con el hardware y el software disponible en este momento solo resulta practicable el análisis de secuencias de una longitud de unos pocos miles de nucleótidos, que es, aproximadamente, la longitud de un gen; sin embargo, el análisis de segmentos aún mayores, como el genoma completo en el caso de virus como el de la influenza o el VIH, o el análisis de una mayor cantidad de secuencias, pudiera contribuir al alcance de un entendimiento más profundo del proceso evolutivo y el código genético.

Hasta el presente en la literatura disponible no se ha reportado la implementación en paralelo de los algoritmos básicos empleados en el análisis filogenético. Su rediseño e implementación

con estas características pudiera conducir a un aumento considerable del rendimiento que permita el análisis de mayor cantidad de información.

Estos hechos conducen al planteamiento de la siguiente pregunta de investigación:

¿En qué medida la implementación en C++ en paralelo para un clúster de computadoras o una máquina multiprocesador produce una reducción notable del tiempo computacional requerido para el procesamiento de grandes volúmenes de datos procedentes de secuencias de ADN y en particular en tareas de pronóstico de mutaciones de virus, como el de la Influenza?

Objetivos del trabajo

Este trabajo se propone como objetivo general:

- Disminuir sustancialmente el costo computacional asociado a la aplicación de algunos algoritmos básicos del análisis filogenético, particularmente, aquellos relacionados con estudios evolutivos, en presencia de grandes volúmenes de datos, mediante su implementación en paralelo, para ser ejecutados en un clúster de computadoras o en una computadora multiprocesador.

Como objetivos específicos:

- Encontrar aquellas fases de los algoritmos básicos del análisis filogenético en las que sea factible aplicar técnicas de paralelización y para las cuales no exista aún un software de este tipo.
- Desarrollar una aplicación que permita ejecutar estos algoritmos en paralelo y que pueda ser usada tanto en un clúster de computadoras como en una máquina multiprocesador.
- Evaluar el desempeño de la aplicación comparándolo con el de una versión secuencial de la misma.

Hipótesis de trabajo

Teniendo en cuenta los elementos teóricos antes expuestos, así como las interrogantes existentes, se plantea la siguiente hipótesis general de investigación:

La implementación en paralelo, usando el paradigma de paso de mensajes, de los algoritmos básicos empleados en el análisis filogenético para su ejecución en un clúster de computadoras o una computadora multiprocesador conduce a una disminución de no menos de 25% del costo computacional originado por el procesamiento de grandes volúmenes de secuencias de ADN.

Estructura del trabajo

El trabajo se estructura esencialmente en tres capítulos.

- El capítulo 1 está dedicado, en su primera parte, a la descripción de los fundamentos biológicos y las herramientas matemático computacionales utilizadas en el análisis filogenético. Se presenta luego un breve sumario de los algoritmos básicos del análisis filogenético, además se hace una introducción a las características de la programación en paralelo.
- En el capítulo 2 se presentan las principales herramientas computacionales utilizadas en la implementación de los distintos algoritmos. Se realiza además un análisis de la complejidad computacional de la versión secuencial de los algoritmos, para luego mostrar el procedimiento seguido en la paralelización de las partes más críticas. Por último se presenta un sencillo manual de usuario de la aplicación desarrollada.
- En el capítulo 3 se presentan y discuten los resultados de la ejecución de los algoritmos implementados en una computadora con un procesador multinúcleo y un clúster de computadoras utilizando secuencias concretas. Se discute, además la evaluación del desempeño de los mismos, comparados con los algoritmos secuenciales correspondientes.

1. FUNDAMENTOS BIOLÓGICOS, MATEMÁTICOS Y COMPUTACIONALES

En este capítulo se realiza una descripción de los fundamentos biológicos y las herramientas matemáticas y computacionales que se utilizan en el análisis filogenético. Se describen los términos biológicos, lo que significa hacer multialineamiento de secuencias, lo que son los modelos evolutivos y el análisis filogenético. Se explica el método de estimación de las tasas de mutación por sitio y se da una breve introducción a la programación en paralelo.

1.1. Introducción a los términos biológicos

Se presentan en esta sección algunas ideas básicas de biología molecular y algunos resultados anteriores relacionados con este trabajo. Los mismos son imprescindibles para la comprensión de las hipótesis de investigación y el resto de la tesis.

1.1.1. Las moléculas básicas de la vida

Los principales actores de la Biología Molecular son las moléculas del ADN, ARN y las proteínas. Estas moléculas son simultáneamente los principales actores de un poderoso sistema de comunicación, el sistema de información genética, entrelazados por el código de comunicación más maravilloso conocido por el hombre, el Código Genético.

La expresión de la información almacenada en el Ácido desoxirribonucleico (ADN) se produce a través de la transcripción lineal de la secuencia de nucleótidos en la secuencia de nucleótidos del Ácido ribonucleico mensajero (ARNm). La secuencia de nucleótidos del ARNm es traducida en una secuencia lineal de aminoácidos, de manera que el flujo de información es ADN $\rightarrow$ARNm $\rightarrow$Proteína.

1.1.2. El ADN y ARNm

El ADN está compuesto por cuatro moléculas básicas llamadas nucleótidos las cuales solo se diferencian en que cada una posee una base nitrogenada diferente. Cada nucleótido contiene

un fosfato, un azúcar (desoxirribosa) y una de las cuatro bases nitrogenadas: *Adenina* (*A*), *Guanina* (*G*), *Timina* (*T*), *Citosina* (*C*) (Figura 1.1)

La estructura de la molécula de ADN es una doble hélice. Las bases nitrogenadas se aparean en la doble hélice formando enlaces por puentes de hidrógeno de acuerdo a las siguientes reglas: $G \equiv C$, $A = T$, donde cada "-" simboliza un enlace por puente de hidrógeno y cada par contiene una purina (*A* o *G*) y una pirimidina (*C* o *T*). Las bases que forman el par se denominan bases complementarias. Una hélice simple es una cadena de nucleótidos unidos por enlaces fosfodiester. Cada hélice simple se une con la hélice complementaria a través de los enlaces de puentes de hidrógeno que forman las bases, dando lugar a la doble hélice. (Figura 1.1)

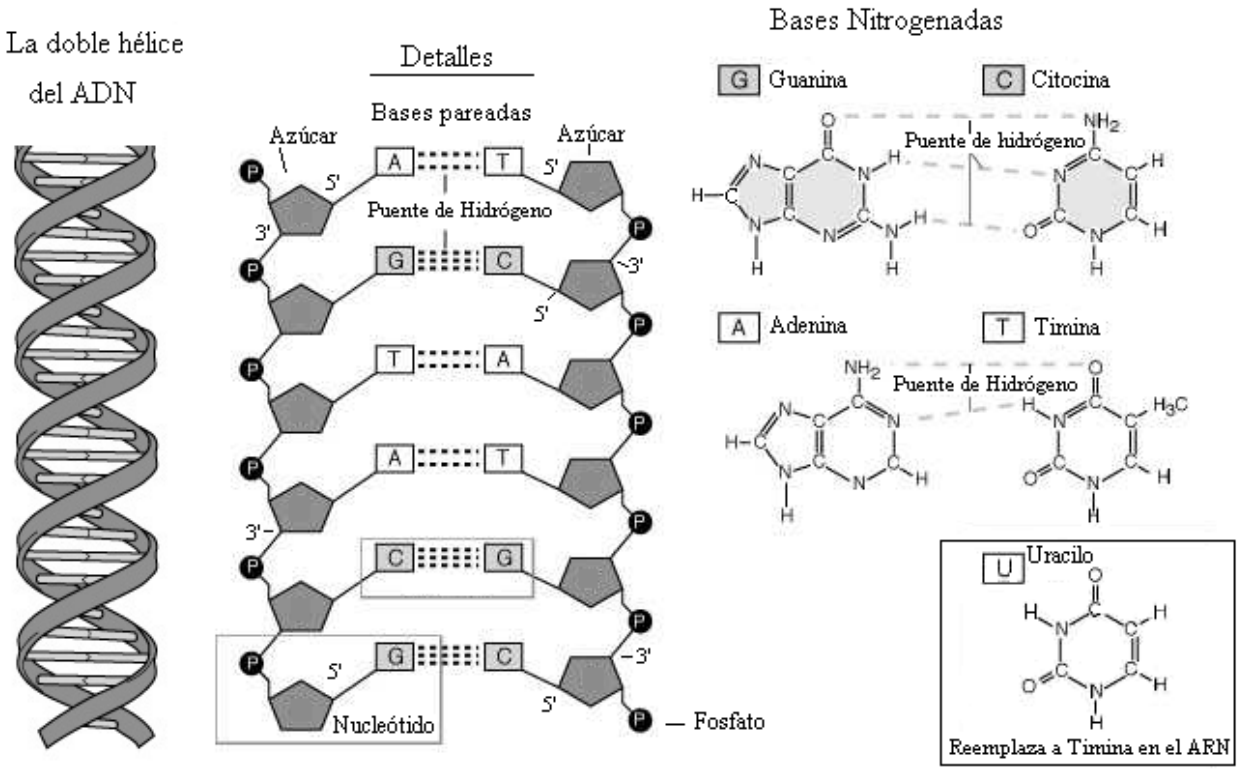

Figura 1.1. La doble hélice del ADN. Detalles de las bases nitrogenadas y su apareamiento en el ADN.

La molécula del ADN es una molécula direccional debido a la estructura asimétrica del azúcar desoxirribosa que forma el esqueleto de las hélices. Cada azúcar es conectado cadena arriba por el quinto carbono y cadena abajo por el tercer carbono, de manera que si una de las

cadenas tiene orientada el azúcar en la dirección 5'→3' (se lee de cinco prima a tres prima), la cadena complementaria estará orientada en la dirección 3'→5'. El ARNm está formado por una cadena de nucleótidos unidos por enlaces fosfodiester. En particular constituye una hélice en la cual el azúcar presente en el nucleótido es una ribosa, la base nitrogenada T es sustituida por la base Uracilo (U) y se mantienen las otras bases nucleotídicas: A, C y G como en el ADN. La función del ARNm es transportar hacia el citoplasma la información genética que se encuentra almacenada en el núcleo de la célula en la secuencia de nucleótidos del ADN.

Cuando se traten secuencias de ADN codificantes para proteínas, éstas se pueden escribir en la práctica utilizando las bases T o U indistintamente.

1.2. El multialineamiento de secuencias

El alineamiento múltiple de secuencias de ADN y proteínas es la piedra angular de la Bioinformática ya que es el punto de partida para los análisis de secuencias más populares desarrollados hasta el momento. Alinear es una forma de representar y comparar dos o más secuencias o cadenas de ADN, ARN, o estructuras primarias proteicas para resaltar sus zonas de similitud, que podrían indicar relaciones funcionales o evolutivas entre los genes o proteínas analizados. Las secuencias alineadas se escriben con letras (representando aminoácidos o nucleótidos) en filas de una matriz en las que, si es necesario, se insertan espacios para que las zonas con idéntica o similar estructura se alineen.

Los alineamientos múltiples de secuencias son útiles para identificar similitudes y diferencias entre secuencias, producir árboles filogenéticos, y desarrollar modelos de homología sobre estructuras de proteínas. Sin embargo, la relevancia biológica de los alineamientos no siempre es clara. Se asume a menudo que los alineamientos reflejan un grado de cambio evolutivo entre secuencias que descienden de un ancestro común; pero es formalmente posible que la convergencia evolutiva pueda darse para producir similitudes aparentes entre proteínas que no estén evolutivamente relacionadas, pero que lleven a cabo funciones similares y tengan parecidas estructuras.

Las representaciones visuales del alineamiento ilustran las diferencias entre las secuencias que pueden ser interpretados como mutaciones (un solo cambio de aminoácidos o nucleótidos) que aparecen como diferentes caracteres en una sola columna del alineamiento, o la inserción o

eliminación de caracteres (mutaciones in-del) que aparecen como huecos en una o varias de las secuencias en la alineación.

En la actualidad se disponen de varios paquetes de software que permiten realizar el alineamiento múltiple de secuencias. Además, el servicio de alineamiento de secuencias se oferta gratuitamente en servidores en Internet, aunque la mayoría de los programas más populares se pueden adquirir libremente. En particular, en este trabajo fueron utilizados los softwares MEGA 4 (Tamura, Dudley et al. 2007) y BioEdit (Hall 1999), los cuales se pueden obtener en los sitios web http://www.megasoftware.net/ y http://www.mbio.ncsu.edu/BioEdit/bioedit.html, respectivamente. En la Figura 1.2 se ilustra una región del alineamiento de secuencias codificantes para proteínas mitocondriales de 14 mamíferos obtenido con el MEGA 4.

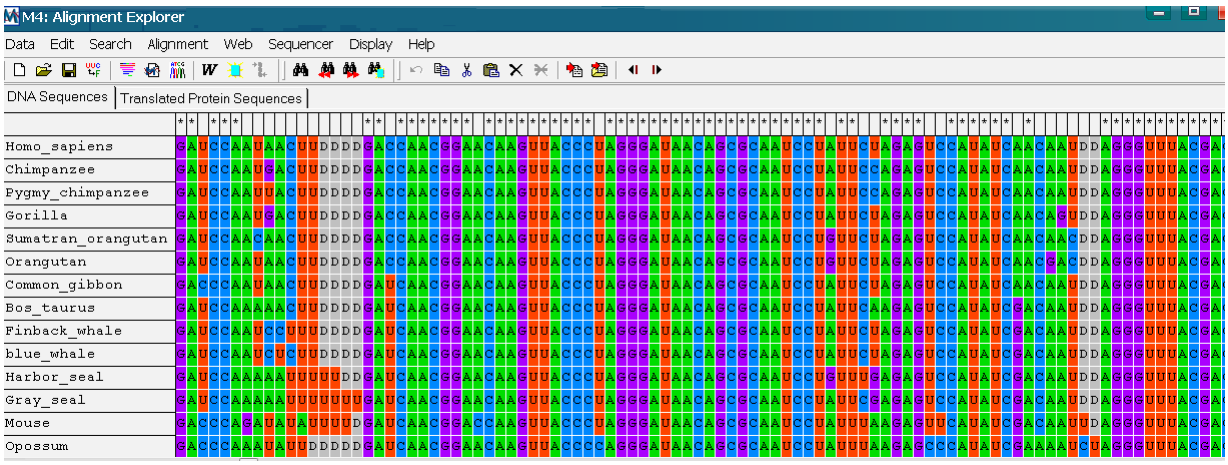

Figura 1.2. Ejemplo de un alineamiento múltiple de secuencias obtenido con el MEGA 4.

1.3. Modelos evolutivos y análisis filogenético

Los árboles filogenéticos son la herramienta más usada en la representación de las relaciones existentes entre las especies o entre individuos de una misma especie. Su uso se hace indispensable cuando se desea analizar un conjunto de secuencias genómicas de diversos organismos. En esta sección se abordan los principales temas relacionados con la reconstrucción de árboles filogenéticos. En particular se describen distintas formas de modelación del proceso evolutivo mediante cadenas de Markov continuas en el tiempo, que permite, entre otras cosas, realizar una estimación bastante aproximada de las distancias entre

secuencias, y se presentan algunos de los algoritmos más usados para la reconstrucción de dichos árboles.

1.3.1. Sumario sobre los modelos markovianos

Mientras la cantidad de secuencias disponibles aumenta cada día, el conocimiento actual de la Biología es solo una pequeña fracción de lo que aún resta por descubrir. Así, en la biología computacional, se hace inevitable el trabajo con un alto grado de incertidumbre: algunos datos se desconocen y otros son incorrectos (Baldi and Brunak. 2001). La modelación mediante procesos de Markov de las sustituciones nucleotídicas usada en el cálculo de la distancia entre secuencias, forma la base del análisis bayesiano y de verosimilitud de múltiples secuencias en una filogenia (Yang 2006).

Actualmente, uno de los primeros problemas que hay que enfrentar al analizar varias secuencias es la estimación de la distancia a que se encuentran unas de otras. La distancia evolutiva entre dos secuencias se define como el número esperado de sustituciones nucleotídicas por sitio.

Si se asume que la tasa de evolución se mantiene constante a través del tiempo, la distancia debe incrementarse linealmente con respecto al tiempo de divergencia. Una forma simple de medir dicha distancia, a veces llamada la distancia *p*, es tomar la proporción de sitios diferentes. Esta distancia no contempla el hecho de que en un mismo sitio puedan ocurrir varias mutaciones o distintas mutaciones en ambas secuencias que llevaron por caminos evolutivos distintos a la misma base, por lo que viola la propiedad de incremento lineal cuando las secuencias no son cercanas en el tiempo. Lo anterior hace que el uso de esta distancia solo sea apropiado en la comparación de secuencias muy similares.

Para estimar el número real de sustituciones entre secuencias relativamente alejadas en el proceso evolutivo, se necesita un modelo probabilístico que describa los cambios entre nucleótidos. Las cadenas continuas de Markov son la herramienta más usada con este propósito. Normalmente se asume que cada sitio de la secuencia evoluciona de forma independiente. Las sustituciones en cada sitio en particular son descritas mediante una cadena de Markov, que tiene como estados las bases nucleotídicas. La imposición de algunas restricciones en las tasas de sustitución entre nucleótidos lleva a diferentes modelos, como el

JC69 (Jukes and Cantor 1969), el K80 (Kimura 1980), o el TN93 (Tamura and Nei 1993). En el anexo 1 se presenta un resumen de los principales modelos descritos en la literatura.

A continuación se describe el modelo propuesto por Koichiro Tamura y Masatochi Nei en 1993 (Tamura and Nei 1993), para luego introducir la extensión del mismo propuesta por Robersy Sánchez y Ricardo Grau, que da lugar al "modelo de cinco bases" (Sánchez and Grau 2009), que es el principal modelo usado en este trabajo y será referido como FB de ahora en adelante.

El modelo TN93

Según este modelo las transiciones entre purinas ocurren a una razón distinta a la de aquella entre pirimidinas. También se asume que todas las transversiones ocurren con la misma velocidad, aunque esta puede ser distinta de las tasas de transición. Por último se permite que las cuatro bases tengan distintas distribuciones estacionarias.

Como resultado la matriz de las tasas de sustitución de nucleótidos es

$$Q_r = \begin{bmatrix} -(\alpha_2\pi_G + \beta\pi_Y) & \beta\pi_C & \alpha_2\pi_G & \beta\pi_T \\ \beta\pi_A & -(\alpha_1\pi_T + \beta\pi_R) & \beta\pi_G & \alpha_1\pi_T \\ \alpha_2\pi_A & \beta\pi_C & -(\alpha_2\pi_A + \beta\pi_Y) & \beta\pi_T \\ \beta\pi_A & \alpha_1\pi_C & \beta\pi_G & -(\alpha_1\pi_C + \beta\pi_R) \end{bmatrix} \tag{1.1}$$

donde π_A, π_C, π_G y π_T son las frecuencias estacionarias de los nucleótidos A, C, T y G respectivamente y $\pi_R=\pi_A + \pi_G$ y $\pi_Y=\pi_C + \pi_T$. Los parámetros α_1, α_2 y β son, respectivamente, la tasa de transiciones entre pirimidinas, entre purinas y la tasa de transversiones.

Cuando las frecuencias de nucleótidos se mantienen en equilibrio, o lo que es lo mismo, cuando la cadena de Markov se mantiene estacionaria, la tasa promedio de sustituciones para este modelo está dada por

$$\lambda = 2\pi_A\pi_G\alpha_1 + 2\pi_C\pi_T\alpha_2 + 2\pi_R\pi_Y\beta\,, \tag{1.2}$$

mientras que el número esperado de sustituciones nucleotídicas (distancia evolutiva) entre dos secuencias que divergieron hace t unidades de tiempo evolutivo ($d = 2\lambda t$) es

$$d = 4\pi_A\pi_G\alpha_1 t + 4\pi_C\pi_T\alpha_2 t + 4\pi_R\pi_Y\beta t$$

(1.3)

Para deducir una fórmula que permita estimar *d*, se debe conocer la proporción esperada de sitios con diferencias transicionales entre purinas (P_1) y entre pirimidinas (P_2) y de aquellos con diferencias transversionales (Q), expresadas en función de las tasas de sustitución y el tiempo evolutivo (Tamura 1992):

$$P_1 = \frac{2\pi_A \pi_G}{\pi_R}\left(\pi_R + \pi_Y e^{-2\beta t} - e^{-2(\pi_R \alpha_1 + \pi_Y \beta)t}\right) \tag{1.4}$$

$$P_2 = \frac{2\pi_C \pi_T}{\pi_Y}\left(\pi_Y + \pi_R e^{-2\beta t} - e^{-2(\pi_Y \alpha_2 + \pi_R \beta)t}\right) \tag{1.5}$$

$$Q = 2\pi_R \pi_Y \left(1 - e^{-2\beta t}\right) \tag{1.6}$$

Como P_1, P_2, Q, π_A, π_C, π_G y π_T pueden ser estimados a partir del alineamiento de las dos secuencias que se están comparando, se puede obtener entonces el siguiente estimado de *d*:

$$d' = -\frac{2\pi_A \pi_G}{\pi_R}\log_e\left(1 - \frac{\pi_R}{2\pi_A \pi_G}P_1' - \frac{1}{2\pi_R}Q'\right) - \frac{2\pi_C \pi_T}{\pi_Y}\log_e\left(1 - \frac{\pi_Y}{2\pi_C \pi_T}P_2' - \frac{1}{2\pi_Y}Q'\right)$$
$$- 2\left(\pi_R \pi_Y - \frac{\pi_A \pi_G \pi_Y}{\pi_R} - \frac{\pi_C \pi_T \pi_R}{\pi_Y}\right)\log_e\left(1 - \frac{1}{2\pi_R \pi_Y}Q'\right) \tag{1.7}$$

donde x' es el estimado del parámetro x correspondiente. El símbolo ' fue omitido en los estimados de las frecuencias para ganar en claridad. (Tamura and Nei 1993)

Es necesario también ser capaz de calcular la matriz de probabilidades de transición en el tiempo $P_{ij}(t)$ de la cadena, la cual puede ser derivada a partir de la siguiente ecuación diferencial:

$$\frac{dP(t)}{dt} = P(t)Q_r \tag{1.10}$$

De la solución de esta ecuación con condición inicial $P(0)=I$ resulta la matriz de probabilidad de transición

$$P(t) = e^{tQ_r} \tag{1.11}$$

La forma estándar de calcular una función algebraica, como la exponencial, de una matriz Q es por medio de la diagonalización siguiente, conocida también como descomposición espectral:

$$Q = U\Lambda U^{-1} \tag{1.12}$$

donde Λ es una matriz diagonal cuyos elementos son los valores propios de la matriz Q, U es la matriz formada por los vectores propios correspondientes a los valores propios en Λ y U^{-1} es la inversa de U. Siendo entonces ρ_1, ρ_2, ρ_3 y ρ_4 los valores propios de Q, se puede calcular la exponencial vista anteriormente como

$$e^{Qt} = U\begin{pmatrix} e^{\rho_1 t} & 0 & 0 & 0 \\ 0 & e^{\rho_2 t} & 0 & 0 \\ 0 & 0 & e^{\rho_3 t} & 0 \\ 0 & 0 & 0 & e^{\rho_4 t} \end{pmatrix} U^{-1} \tag{1.13}$$

Aunque existe una expresión analítica para los valores y vectores propios de la matriz de tasas de sustitución cuando se conocen los valores exactos de α_1, α_2 y β (Yang 2006), en la práctica estos se desconocen, por lo que el cálculo de los mismos se realiza mediante un método numérico. En este trabajo se utilizó la implementación del algoritmo de Francis (Francis 1961a; Francis 1961b) que aparece en la GNU Scientific Library.

El "modelo de cinco bases" como extensión del modelo TN93

La adición de una nueva base (D) al conjunto de cuatro ya existentes (A, C, G y T) produce varias modificaciones al modelo antes expuesto. Este incremento en complejidad está plenamente justificado por las dificultades encontradas por la comunidad científica al tratar de explicar el surgimiento de la vida y la evolución del código genético primitivo usando solo las bases presentes en el ADN actual, lo que llevó a considerar la posibilidad de la existencia de bases extra con pareos no específicos con las actuales (Orgel 1987; Levy and Miller 1998; Orgel 2004). Esta quinta base también puede ser usada para representar mutaciones *in-del* al ubicarse en los espacios o gaps generados por el alineamiento de secuencias, lo que permite que se utilice la información presente en esos sitios y no se deseche como es inevitable hacer al usar los modelos clásicos.

Como en el modelo TN93, en el presente la tasa de transición entre purinas (α_R) puede ser diferente de aquella entre pirimidinas (α_Y), la tasa de transversiones (β) es la misma para todas

las combinaciones de purinas y pirimidinas aunque puede ser distinta de α_1 y α_2 y se permite que las cinco bases posean distintas frecuencias. Dos nuevos parámetros, γ y N_D están relacionados con la quinta base y no aparecen en el modelo TN93, estos son, respectivamente, la tasa de sustitución entre la D y las otras cuatro y la correspondiente proporción esperada de sitios mostrando estas sustituciones (Sánchez and Grau 2009).

La matriz de las tasas de sustitución para este modelo es

$$Q_r = \begin{pmatrix} * & \beta\pi_C & \alpha_R\pi_G & \beta\pi_T & \gamma\pi_D \\ \beta\pi_A & * & \beta\pi_G & \alpha_Y\pi_T & \gamma\pi_D \\ \alpha_R\pi_A & \beta\pi_C & * & \beta\pi_T & \gamma\pi_D \\ \beta\pi_A & \alpha_Y\pi_C & \beta\pi_G & * & \gamma\pi_D \\ \gamma\pi_A & \gamma\pi_C & \gamma\pi_G & \gamma\pi_T & * \end{pmatrix} \tag{1.14}$$

Cuando las frecuencias de nucleótidos están en equilibrio la tasa promedio de sustituciones nucleotídicas por sitio para este modelo está dada por

$$\lambda = 2\pi_A\pi_G\alpha_R + 2\pi_C\pi_T\alpha_Y + 2\pi_R\pi_Y\beta + 2\pi_D(1-\pi_D)\gamma \tag{1.15}$$

mientras que el número esperado de sustituciones nucleotídicas (distancia evolutiva) entre dos secuencias que divergieron hace *t* unidades de tiempo evolutivo ($d=2\lambda t$) es

$$d = 4\pi_A\pi_G\alpha_R t + 4\pi_C\pi_T\alpha_Y t + 4\pi_R\pi_Y\beta t + 4\pi_D(1-\pi_D)\gamma t\,. \tag{1.16}$$

Un estimado de esta distancia puede ser obtenido a partir de

$$P_1 = 2\frac{\pi_A\pi_G}{\pi_R}\left(\pi_R + \frac{e^{-2\gamma}\pi_D\pi_R}{1-\pi_D} + \frac{\pi_Y}{1-\pi_D}e^{-2(\beta+(\gamma-\beta)\pi_D)t} - e^{-2(\gamma\pi_D+\beta\pi_Y+\alpha_R\pi_R)t}\right) \tag{1.17}$$

$$P_2 = 2\frac{\pi_C\pi_T}{\pi_Y}\left(\pi_Y + \frac{e^{-2\gamma}\pi_D\pi_Y}{1-\pi_D} + \frac{\pi_R}{1-\pi_D}e^{-2(\beta+(\gamma-\beta)\pi_D)t} - e^{-2(\gamma\pi_D+\beta\pi_R+\alpha_Y\pi_Y)t}\right) \tag{1.18}$$

$$Q = 2\frac{\pi_R\pi_Y}{1-\pi_D}\left(e^{-2\gamma} - e^{-2((\beta-\gamma)(1-\pi_D)+\gamma)} + \left(1-e^{-2\gamma}\right)(1-\pi_D)\right) \tag{1.19}$$

$$N_D = 2\left(1-e^{-2\gamma}\right)(1-\pi_D)\pi_D \tag{1.20}$$

de donde resulta

$$d' = -2\left[\frac{\pi_A\pi_G}{\pi_R}\log\left(1-\frac{P_1'\pi_R}{2\pi_A\pi_G}-\frac{Q'}{2\pi_R}-\frac{N_D'}{2(1-\pi_D)}\right)+\frac{\pi_C\pi_T}{\pi_Y}\log\left(1-\frac{P_2'\pi_Y}{2\pi_C\pi_T}-\frac{Q'}{2\pi_Y}-\frac{N_D'}{2(1-\pi_D)}\right)\right.$$

$$+\frac{1}{(1-\pi_D)}\left(\pi_R\pi_Y-\frac{\pi_A\pi_G\pi_Y}{\pi_R}-\frac{\pi_C\pi_T\pi_R}{\pi_Y}\right)\log\left(1-\frac{Q'(1-\pi_D)}{2\pi_R\pi_Y}-\frac{N_D'}{2(1-\pi_D)}\right)$$

$$-\frac{\pi_D}{1-\pi_D}\left(\pi_A\pi_G+\pi_C\pi_T+\pi_R\pi_Y-(1-\pi_D)^2\right)\log\left(1-\frac{N_D'}{2(1-\pi_D)\pi_D}\right)\Bigg] \tag{1.21}$$

donde x' es el estimado del parámetro x correspondiente, se omitió el símbolo ' en los estimados de las frecuencias para ganar en claridad.

La matriz de probabilidad de transición en el tiempo $P_{ij}(t)$ se deriva, como en el modelo TN93, de la ecuación diferencial

$$\frac{dP(t)}{dt}=P(t)Q_r \tag{1.22}$$

Cuya solución, con condición inicial $P(0)=I$, es

$$P(t)=e^{tQ_r} \tag{1.23}$$

y sus elementos pueden ser calculados por medio de la descomposición espectral, para la cual también existe una solución analítica si se conocen los valores de α_R, α_Y, β y γ (Sánchez and Grau 2009), pero se debe usar igualmente el método numérico.

1.3.2. Algoritmo para la construcción de árboles filogenéticos

Una vez que se han alineado las secuencias y se ha seleccionado el modelo probabilístico a utilizar suele procederse a reconstruir el árbol filogenético. Existe una gran variedad de métodos para lograr este objetivo. La inmensa mayoría de ellos se basa en encontrar el árbol que maximice algún criterio de optimalidad (Kumar and Filipski 2008). Como la cantidad de posibles topologías crece de forma exponencial con respecto a la cantidad de secuencias que se analizan (Yang 2006) una búsqueda exhaustiva es prácticamente imposible. Afortunadamente, la calidad de los árboles filogenéticos producidos por métodos heurísticos más rápidos es similar a la obtenida con búsquedas extensivas (Kumar and Filipski 2008).

Los criterios más usados en la reconstrucción de árboles filogenéticos son el de mínima evolución (ME), el de máxima parsimonia (MP) y el de máxima verosimilitud (MV). Bajo el criterio de MP, la topología que requiera el menor número de cambios de nucleótidos para ajustarse a las secuencias observadas es la seleccionada (Fitch 1971), su principal ventaja es que se considera no paramétrico en el sentido de que no depende de un modelo en específico,

pero ha sido demostrado que presenta inconsistencias bajo ciertas condiciones, o sea que, aún para secuencias arbitrariamente largas, no produciría el árbol correcto si se dieran dichas condiciones.

En los métodos de MV se selecciona la topología de mayor verosimilitud de acuerdo con algún modelo probabilístico de sustituciones nucleotídicas (Felsenstein 1981), este criterio es reconocido por producir los resultados más precisos, pero tiene como desventajas la dependencia del modelo probabilístico y, además, suele ser muy costoso computacionalmente en términos de tiempo de ejecución por lo que no permite una amplia exploración del espacio de búsqueda.

En los métodos basados en ME, primero se obtiene una matriz de las distancias entre cada par de secuencias, para luego calcular la longitud total de las ramas de cada topología necesaria para ajustarse a la matriz de distancias y se selecciona aquella que requiera la menor longitud.

Un método usado frecuentemente por su eficiencia computacional es el neighbour-joining (Saitou and Nei 1987). Este está basado en, aunque no necesariamente optimiza, el criterio de ME. El mismo es un algoritmo de clusterización que comienza con un árbol en forma de estrella y en cada paso se seleccionan dos nodos hijos de la raíz para ser agrupados de forma que se minimice la suma de las longitudes de las ramas. Es un método goloso que no garantiza encontrar el resultado óptimo, pero en la práctica ha sido ampliamente aplicado con muy buenos resultados.

Otro algoritmo muy utilizado es el de mínimos cuadrados ordinarios (OLS), en el cual dada una topología se ajustan las longitudes de las ramas a las distancias observadas entre las secuencias por medio de los mínimos cuadrados, la mejor topología se selecciona de acuerdo al criterio de ME o la que produzca un mejor ajuste a la matriz de distancias. Este método es susceptible de ser modificado mediante el uso de pesos, de forma que dos subárboles hermanos tengan siempre el mismo peso, en contraposición al método clásico no ponderado (el OLS) en el cual cada hoja tiene el mismo peso y cada subárbol tiene un peso igual a la suma de sus hojas. A esta variante se le conoce como mínimos cuadrados balanceados (balanced_LS) y ha demostrado ser aún más precisa que la versión clásica y que todas las variantes de NJ, además de que, aunque es de ejecución ligeramente más lenta que OLS (por un factor de aproximadamente log(n)), significa una mejora notable en este aspecto con

respecto al NJ, lo cual la hace ideal para su uso con árboles relativamente extensos (Desper and Gascuel 2002; Desper and Gascuel 2004).

En este trabajo se usó el FastME (Desper and Gascuel 2002), un software disponible libremente en internet en http://www.atgc-montpellier.fr/fastme/, que permite el uso de los algoritmos basados en el criterio de ME mencionados. El mismo construye siguiendo una estrategia golosa un árbol inicial por medio de uno de los métodos NJ, BIONJ, balanced_GME o OLS_GME para, partiendo de este, buscar el árbol óptimo usando intercambios de vecinos más cercanos (NNI del inglés *nearest neighbor interchange*), ya sea por medio del OLS_NNI o del balanced_NNI. En la Figura 1.3 se muestra un árbol obtenido mediante este software.

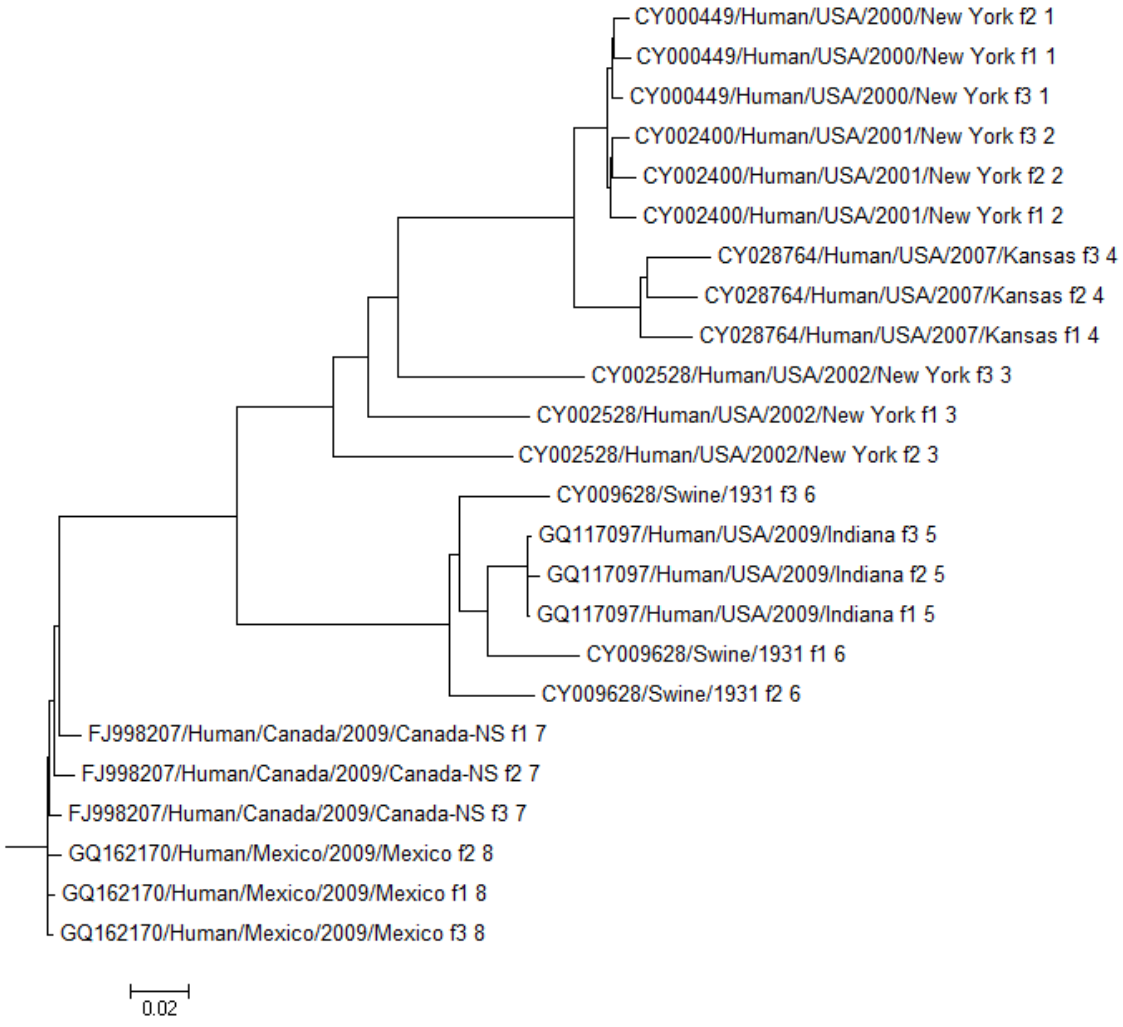

Figura 1.3. Árbol filogenético de 24 secuencias del virus de la Influenza A H1N1 obtenido con el FastME.

1.4. Método de estimación de las tasas de sustitución por sitio

En esta sección se describe el procedimiento realizado para la estimación de las tasas de sustitución por sitio. En la primera parte se explica el uso del método bayesiano empírico con este propósito. En este método se hace uso de la distribución gamma, la cual depende de dos parámetros que deben ser estimados con anterioridad, sobre esto trata la segunda parte de esta sección, donde se introduce el método de máxima verosimilitud, que permite calcular una primera aproximación de las tasas de sustitución, a partir de la cual se pueden calcular los parámetros α y β de la distribución gamma, necesaria para el cálculo final de las mismas.

1.4.1. El método bayesiano empírico

La estimación de las tasas de sustitución por sitio resulta imprescindible cuando se desean conocer las secuencias más probables que tenían los ancestros de los que descienden los individuos actuales o las que probablemente tendrán los descendientes de estos. Comúnmente se asume que las tasa de mutación es una variable aleatoria, de la cual se conoce que sigue aproximadamente una distribución gamma, a partir de lo cual esta puede ser estimada usando la distribución condicional (a posteriori) de la misma, dados los datos en el sitio (Yang 2006):

$$f\left(r|x;\theta\right)=\frac{f\left(r|\theta\right)f\left(x|r;\theta\right)}{f\left(x|\theta\right)} \tag{1.24}$$

Donde r es la tasa de mutación, x los datos observados en las secuencias y θ los parámetros del modelo.

En particular se puede tomar como estimado de la tasa de mutación por sitio el valor esperado de esta distribución (Yang and Wang 1995):

$$r'=E\left(r|x;\theta\right)=\int_0^\infty rf\left(r|x;\theta\right)dr=\frac{\int_0^\infty rf\left(x|r;\theta\right)f\left(r\right)dr}{f\left(x|\theta\right)}, \tag{1.25}$$

donde el cociente se calcula de la siguiente manera:

$$f\left(x|\theta\right)=\int_0^\infty f\left(x|r;\theta\right)f\left(r\right)dr \tag{1.26}$$

Sustituyendo (1.26) en (1.25) nos queda la expresión para la estimación a posteriori de la tasa de mutación en el sitio

$$r' = \frac{\int_0^\infty r f(x|r;\theta) f(r) dr}{\int_0^\infty f(x|r;\theta) f(r) dr} \tag{1.27}$$

En esta expresión $f(x|r;\theta)$ es la verosimilitud del árbol, la forma en que esta se calcula se describe más adelante, y $f(r)$ es la densidad probabilística *a priori* de las tasas de mutación, o sea, la función de densidad de la distribución gamma.

Las integrales que aparecen en la expresión (1.27) no suelen tener una solución analítica por lo que se acostumbra discretizarlas de la siguiente forma:

$$\int_0^\infty r f(x|r;\theta) f(r) dr = \sum_j r_j f(x|r_j;\theta) P(r = r_j)$$

$$\int_0^\infty f(x|r;\theta) f(r) dr = \sum_j f(x|r_j;\theta) P(r = r_j),$$

donde los r_j son las distintas clases discretas de las tasas de sustitución (Nielsen 2005; Yang 2006).

Como esta expresión solo depende de la topología del árbol, de las longitudes de las ramas y de los nucleótidos presentes en las secuencias en el sitio en cuestión, las tasas obtenidas por este método para dos sitios mostrando los mismos datos (o lo que es lo mismo, mostrando el mismo patrón de nucleótidos) serán idénticas. Este hecho permite que en la práctica, se agrupen los sitios en clases de equivalencia según el patrón de nucleótidos que presentan y solo se calcule la tasa de mutación para un representante de cada clase, asignándoles a los demás ese mismo valor, logrando así que se disminuya el costo computacional del procedimiento.

1.4.2. El método de máxima verosimilitud

En la sección anterior se explicó cómo se pueden estimar las tasas de sustitución por sitio por medio del método bayesiano empírico, bajo el supuesto de que las mismas siguen aproximadamente la distribución gamma. Esta distribución posee dos parámetros (α y β) que deben ser estimados previo a la aplicación del procedimiento descrito. En esta sección se

describe el método de máxima verosimilitud, el cual es otra forma de estimar los valores de las tasas de sustitución. En el presente trabajo se utilizan los resultados de la aplicación de este método para estimar los parámetros de la distribución gamma, o sea la distribución de probabilidad *a priori* de las tasas de mutación.

El concepto de la verosimilitud de un árbol filogenético fue introducido por Joseph Felsenstein en 1981 como un criterio de optimización para ser usado en la reconstrucción del mismo (Felsenstein 1981). Se define verosimilitud de un árbol filogenético como la probabilidad de obtener los datos de las secuencias que se tienen a partir del mismo; como se asume que cada sitio evoluciona independientemente de los demás, se puede calcular la verosimilitud del árbol en cada uno de ellos por separado. Si se asume un árbol fijo y se varía la tasa de mutación, se puede usar este criterio como función objetivo y encontrar los valores de las tasas que maximicen la verosimilitud en cada sitio, normalmente haciendo uso de un método numérico. En este caso se escogió el método de Brent (Brent 1973). Este método combina la rápida convergencia a la solución buscada de la interpolación parabólica (convergencia cuadrática) con la robustez del método de la sección de oro, más lento (convergencia lineal); además la aplicación de este algoritmo no requiere el cálculo de las derivadas parciales de la función objetivo, lo que garantiza su aplicabilidad en un mayor número de problemas (Press, Teukolsky et al. 1992).

Al igual que en el método bayesiano empírico, la verosimilitud solo depende de la topología del árbol, de las longitudes de las ramas y de los nucleótidos presentes en las secuencias en el sitio en cuestión, por lo que también es posible agrupar los sitios de las secuencias según los patrones de nucleótidos que éstos presentan para ganar en eficiencia computacional.

El algoritmo de poda

Existe una expresión general para la verosimilitud de un árbol filogenético en un sitio dado de la cadena, pero esta tiene como desventaja que para n individuos tiene 2^{2n-2} términos, lo que hace que en la práctica no pueda ser utilizada cuando el número de secuencias crece. Afortunadamente existe un método, conocido como algoritmo de poda, que se basa en el concepto de verosimilitud condicional y permite reducir considerablemente los cálculos necesarios. En este se hace un recorrido en pos-orden de árbol calculando en cada nodo la

probabilidad de que aparezca cada nucleótido a partir de las probabilidades antes calculadas en sus nodos hijos.

Se define $L_S(k)$ como la verosimilitud basada en los datos en o por debajo del nodo k en el árbol (se asume un árbol invertido, con la raíz arriba y las hojas abajo), dado que se conozca que el nodo k se encuentre el estado s en el sitio en consideración. Si el nodo k es una hoja entonces $L_S(k)= 0$ para todo s, excepto para aquel estado en que conocemos que se encuentra, para el cual $L_S(k)= 1$. Cuando k es un nodo intermedio, con descendientes inmediatos i y j a distancia v_i y v_j respectivamente, se puede calcular para todos los valores de s_k

$$L_{S_k}(k)=\left(\sum_{S_i} P_{S_k S_i}(v_i) L_{S_i}(i)\right)\left(\sum_{S_j} P_{S_k S_j}(v_j) L_{S_j}(j)\right)$$

donde $P_{ij}(t)$ es la probabilidad de transición en el tiempo de la cadena de Markov y depende del modelo probabilístico que se haya seleccionado.

Una vez que se llega a la raíz del árbol (nodo *0*) se han calculado todas las verosimilitudes condicionales para cada uno de los posibles estados. La verosimilitud del árbol en su conjunto para el sitio en consideración es entonces

$$L=\sum_{S_0} \pi_{S_0} L_{S_0}(0)$$

donde π_x representa la probabilidad *a priori* de encontrar el nodo *0* en el estado x, o lo que es lo mismo, la distribución de equilibrio de la cadena de Markov. Este valor también suele ser representado como $f(x|\theta)$, o sea la probabilidad de encontrar el dato x dados el vector de parámetros θ (que incluye la topología del árbol, la tasa de mutación, etc.).

1.5. Características de la programación en paralelo

En los últimos años ha existido la tendencia en el diseño y la fabricación de procesadores a usar varios núcleos de procesamiento dentro de una misma unidad, en lugar de aumentar significativamente la frecuencia de los mismos. Esto se debe en parte a que el aumento de velocidad del procesador no ha podido ser igualado por el de las memorias RAM o el de los buses de direcciones y datos. Por esta y otras razones la programación paralela ha tomado un nuevo auge, dejando de ser un elemento exclusivo de las grandes supercomputadoras o los clústeres de máquinas. En esta sección se muestran las principales formas en que se puede alcanzar paralelismo al ejecutar una tarea. Se comienza por introducir los conceptos de

paralelismo implícito y explícito presentando algunos ejemplos, luego se da una breve descripción de los sistemas con espacio de direcciones compartido y del paradigma de paso de mensajes.

1.5.1. Paralelismo implícito y explícito

La descripción lógica tradicional de una computadora consiste en una memoria conectada a un procesador por medio de un camino de datos. Los tres componentes – procesador, memoria, y camino de datos – presentan cuellos de botella que afectan la capacidad de procesamiento final del sistema. Para enfrentar este problema se han desarrollado un número de innovaciones en la arquitectura de las computadoras, siendo la multiplicidad (en unidades de procesamiento, caminos de datos y unidades de memoria) una de las más importantes. Esta multiplicidad puede quedar totalmente oculta al programador o ser expuesta a él de diversas maneras (Grama, Gupta et al. 2003).

Cuando ocurre paralelismo en la realización de una tarea computacional sin la necesidad de la intervención del programador se dice que se está en presencia de paralelismo implícito. Un buen ejemplo de este lo constituye la capacidad que poseen los procesadores modernos de ejecutar varias instrucciones en un mismo ciclo de reloj, lo cual es controlado por el propio hardware de la máquina y resulta invisible al desarrollador de software. Por otro lado, cuando el programador tiene que especificar las tareas que se ejecutarán en paralelo, las formas de interacción entre las mismas, etc., se dice que se está en presencia de paralelismo explícito.

1.5.2. El Modelo de Comunicación

Cuando se diseña un software en el que varias tareas en paralelo deben cooperar en la realización de un mismo trabajo suele necesitarse que las mismas interactúen entre sí: que necesiten intercambiar datos, que una controle el funcionamiento de las demás, etc. A los mecanismos empleados con este propósito se le conoce como el modelo de comunicación. Las dos formas principales de intercambiar información entre tareas en paralelo son el acceso a un espacio compartido de direcciones (de memoria) y el intercambio de mensajes (Grama, Gupta et al. 2003).

Sistemas de espacio de direcciones compartido

Cuando todos los procesadores de una arquitectura tienen acceso a un conjunto de datos común a todos ellos, se dice que se trata de una plataforma de espacio de direcciones compartido. Los procesadores, y con ellos las tareas que ejecutan, interactúan modificando los datos almacenados en este espacio común. La memoria en las plataformas de espacio de direcciones compartido puede ser local (exclusiva a un solo procesador) o global (común a todos los procesadores). La presencia de un espacio de memoria global simplifica notablemente la programación para este tipo de plataformas. Todas las interacciones de solo lectura son invisibles al programador ya que se codifican de igual forma que en un programa secuencial, lo cual disminuye significativamente el esfuerzo necesario para programar en paralelo. Las interacciones del tipo lectura/escritura son más difíciles de programar que las de solo lectura, ya que estas tareas requieren exclusión mutua para accesos concurrentes. Los paradigmas de programación para espacio de direcciones compartido tales como los *threads* (POSIX, Windows) proveen mecanismos como candados o semáforos para lograr una efectiva sincronización de este tipo de operaciones (Grama, Gupta et al. 2003).

La principal desventaja de este modelo de comunicación, que a la postre determinó que no fuera usado en este trabajo, es que no es aplicable a todas las arquitecturas. En un clúster de computadoras, por ejemplo, donde cada nodo se comunica con los demás a través de una red de área local, no es recomendable el uso de un espacio de direcciones compartido ya que la red se convertiría en un cuello de botella que afectaría severamente el rendimiento total del clúster. Se necesita por tanto un modelo que permita llevar al mínimo la comunicación entre los nodos.

Sistemas basados en paso de mensajes

La forma lógica en que se acostumbra ver una plataforma con paso de mensajes es como un conjunto de nodos de procesamiento, cada uno con su espacio de direcciones exclusivo. Cada uno de estos nodos puede ser de un solo procesador o un multiprocesador con espacio de direcciones compartido, tendencia esta última que ha ido tomando auge en las arquitecturas modernas basadas en paso de mensajes (Grama, Gupta et al. 2003). En este tipo de plataformas las interacciones entre procesos que se ejecutan en nodos diferentes deben ser

llevadas a cabo mediante el envío de mensajes. Este intercambio de mensajes se usa para transferir datos, carga de trabajo o para sincronizar las acciones entre los procesos.

Las operaciones básicas de este paradigma de programación paralela, a partir de las cuales se desarrollan las demás, son *send* y *receive*. Como las operaciones de envío y recepción de mensajes tienen que especificar un destino y, a veces, una fuente; se usa un mecanismo para asignar una identificación o ID a cada uno de los procesos que ejecutan un programa paralelo. Este ID suele ser puesto a disposición del programa por medio de una función *whoami* (el verdadero nombre de la función varía de una API a otra, pero el significado es el mismo). Por último se necesita de una cuarta función para completar el conjunto básico de operaciones de paso de mensajes: *numprocs*, que indica el número de procesos trabajando en conjunto. Con estas cuatro operaciones básicas es posible escribir cualquier programa que se base en el envío de mensajes (Grama, Gupta et al. 2003). Las distintas APIs que siguen este paradigma, como Message Passing Interface (MPI) y Parallel Virtual Machine (PVM), soportan estas operaciones básicas además de otras de más alto nivel.

Es fácil emular una arquitectura de paso de mensajes con p nodos en una computadora de espacio de direcciones compartido con la misma cantidad de procesadores. Una forma de lograr esto es particionar el espacio de direcciones en p partes disjuntas y asignar cada una de estas a un procesador diferente. Un procesador puede entonces enviar o recibir un mensaje escribiendo o leyendo información en la partición correspondiente a otro procesador, usando una primitiva de sincronización apropiada para informar a la otra parte cuando termine la operación de escritura o lectura. Es por tanto el paso de mensajes la opción escogida en este trabajo ya que el software resultante debería producir resultados similares al ser ejecutado tanto en un clúster de computadoras como en una computadora multiprocesador, aunque no haya sido expresamente diseñado para esta última.

1.6. Herramientas computacionales que serán utilizadas

En esta sección se introducen las herramientas utilizadas en la implementación de los algoritmos. Se da una breve reseña histórica del lenguaje de programación C++ y se explican las razones por las que fue seleccionado. También se describen las dos bibliotecas que se utilizaron para los métodos numéricos y la paralelización.

1.6.1. El lenguaje de programación C++

El lenguaje de programación C++ surgió a principios de la década de los 80 cuando un joven ingeniero de los Laboratorios Bell, Bjarne Stroustrup, comenzó a experimentar con la incorporación de algunas extensiones al C. El nuevo lenguaje pronto comenzó a ganar en popularidad no solo dentro de los laboratorios Bell, sino que se creó toda una comunidad de desarrollo alrededor del mismo, apareciendo nuevas funcionalidades, nuevos compiladores, etc. En 1998 la Organización Internacional de Estándares (ISO) publicó las especificaciones de lo que sería el estándar de C++. En este momento el lenguaje soportaba la programación orientada a objetos (el paradigma más utilizado cuando se programa en C++), programación estructural y programación genérica, además de que mantenía prácticamente la misma eficiencia que su predecesor al no correr sobre una máquina virtual como otros lenguajes orientados a objetos de la época. Estas características convierten al C++ en un lenguaje de programación multi-paradigma de propósito general.

Los lenguajes interpretados, como otros lenguajes orientados a objetos, permiten el desarrollo de código fácilmente portable entre las distintas plataformas, pero traen aparejada una significativa pérdida de rendimiento. Por otro lado, C++ usa el modelo de compilación y ensamblado que heredó del C, que permite una generación y optimización de código muy eficientes. Aún así, el lenguaje C++ no carece en absoluto de portabilidad pues se requieren muy pocos o ningún cambio en los programas escritos para una plataforma cuando se quiere compilarlos en otra. Otro factor que influye notablemente en la eficiencia de este lenguaje es la ausencia de un recolector de basura, aunque obliga al programador a ser mucho más cuidadoso en el manejo de la memoria.

Otra de las razones que llevaron a la selección de este lenguaje para la realización de este trabajo fue su popularidad. Como su uso está tan ampliamente generalizado existe una gran disponibilidad de código previamente escrito y de buena calidad para un sinnúmero de aplicaciones. El uso de las bibliotecas disponibles facilita mucho la labor del programador al permitirle concentrarse en las características específicas de su aplicación y no preocuparse por los detalles de implementación de temas secundarios como, por ejemplo, los métodos numéricos, los cuales pueden ser muy engorrosos y propensos a errores.

1.6.2. La biblioteca GSL

En este trabajo fue necesario aplicar varios métodos numéricos de cierta complejidad, como el cálculo de los valores y vectores propios de una matriz. La implementación de este tipo de algoritmos puede resultar engorrosa y propensa a errores, pero, afortunadamente, ya estos han sido implementados en todas sus variantes y, lo que es más importante, estas implementaciones ya han sido depuradas por sus autores y por los usuarios de las mismas.

En este trabajo se utilizó la biblioteca GNU Scientific Library (GSL) para los cálculos numéricos, la cual es un software distribuido libre y gratuitamente bajo la General Public License (GPL). Esta biblioteca consta de varios centenares de funciones escritas en lenguaje C, por lo que la eficiencia es uno de los rasgos que la caracterizan. Entre las diversas funcionalidades que ofrece están el trabajo con vectores y matrices, solución de sistemas de ecuaciones lineales, minimización y cálculo de raíces en una o varias variables, combinatoria, funciones matemáticas elementales (sen, cos, etc.) y especiales (gamma, beta, hipergeométrica, etc.), integración y derivación numéricas, generación de números aleatorios con varias distribuciones, etc.

1.6.3. La biblioteca MPI

En la década de los 80 tuvo lugar un amplio desarrollo de la computación paralela, que propició el surgimiento de un gran número de herramientas de software para el desarrollo de programas de este tipo, las cuales no eran compatibles unas con otras. Muy pronto se hizo evidente la necesidad de llegar a un consenso. El estándar surgió entre los años 1992 y 1994 cuando se creó el Forum MPI (Message Passing Interface), en el cual participan actualmente más de 40 organizaciones, entre vendedores, investigadores, desarrolladores de bibliotecas de software y usuarios. MPI no es en realidad una biblioteca sino un estándar que especifica cómo debe ser una biblioteca MPI. Aunque MPI no es un estándar ISO o IEEE, es ampliamente aceptado como el estándar internacional del paradigma de paso de mensajes, por lo que su portabilidad es una de sus principales ventajas, ya que no es necesario hacer modificaciones sustanciales al código cuando se desea cambiar de plataforma. MPI se presta para ser usado en casi cualquier modelo de programación paralela, siendo originalmente concebido para sistemas de memoria distribuida, hoy en día es ampliamente utilizado también en sistemas de memoria compartida y sistemas híbridos como los clústeres de estaciones de trabajo.

El estándar MPI consta de un gran número de funciones (la versión 1.0 tenía 115), pero existe un subconjunto reducido de estas, de unas 25 funciones, que permiten realizar casi cualquier tarea en paralelo. Además de las cuatro operaciones básicas del paradigma de paso de mensajes, MPI requiere el uso obligatorio de una función de inicialización y una de terminación que realiza labores de limpieza, además, provee varias funciones de alto nivel que permiten enviar información hacia todos los procesos (broadcast), recibir información de todos los procesos (gather), funciones de reducción por medio de varias operaciones como la suma o el mínimo, funciones de tratamiento de errores, etc. (Kowalik 1999)

En el presente trabajo se utilizó MPICH, una implementación libre de este estándar.

2. IMPLEMENTACIÓN EN PARALELO DE LOS ALGORITMOS BÁSICOS DEL ANÁLISIS FILOGENÉTICO

En este capítulo se describe la forma en que se implementaron los algoritmos básicos del análisis filogenético. En la primera sección se introduce brevemente el lenguaje de programación C++, además de las bibliotecas utilizadas. Luego se hace un análisis de la complejidad computacional asociada a la versión secuencial de los algoritmos implementados, a lo cual le sigue la descripción de las técnicas utilizadas en la paralelización de los mismos. El capítulo termina con un sencillo manual de usuario de la aplicación desarrollada.

2.1. Descripción de la implementación

En esta sección se presenta una descripción del diseño general de la aplicación, introduciendo las razones que llevaron a la creación de dos ejecutables separados, además de un análisis de la complejidad computacional de la versión secuencial de los algoritmos implementados.

2.1.1. Diseño general

En este trabajo se desea desarrollar una aplicación que tome como entrada un alineamiento de secuencias, a partir del cual se calcule una matriz de distancias con la que se reconstruya el árbol filogenético, y a partir del alineamiento y el árbol se estiman los parámetros del modelo y las tasas de sustitución por sitio. Es objetivo de este trabajo implementar el cálculo de la matriz de distancias entre las distintas secuencias, la estimación de los parámetros y la estimación de las tasas de sustitución, pero no la implementación de los algoritmos de reconstrucción del árbol filogenético, ya que para esta tarea se dispone de un software muy eficiente y versátil (el FastME). Como esta reconstrucción es un paso intermedio de todo el proceso, se decidió que la mejor opción es tener dos ejecutables separados (*DistanceCalc* y *ParameterEstim*) de forma que la salida del primero sea la entrada del FastME y la salida de este se convierta en la entrada del segundo (ver Figura 2.1).

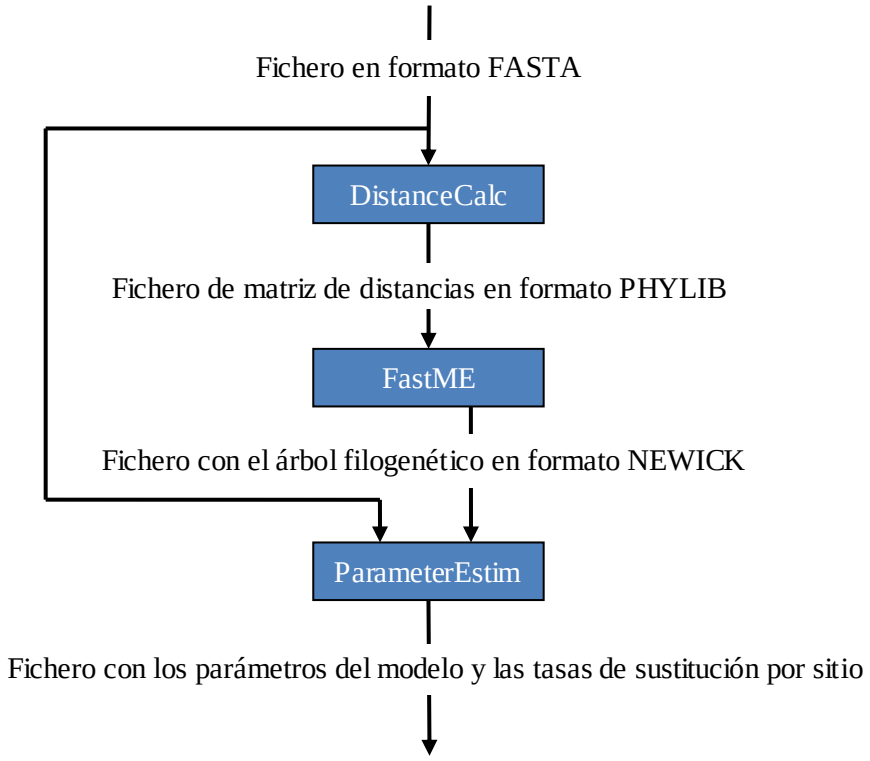

Figura 2.1. Diagrama general del proceso de ejecución de la aplicación

Teniendo en cuenta que el software que se desarrolla se ejecutará fundamentalmente en un clúster de computadoras, donde, por razones de eficiencia, no se dispone de interfaz visual, y también para facilitar la ejecución por lotes del mismo, se decidió que la aplicación funcione por consola y no mediante una interfaz gráfica de usuario (GUI). Cada programa toma como parámetros los nombres de los ficheros que necesita e imprime los resultados en el fichero correspondiente, los mensajes de error se muestran en la salida de error y la información que se brinda al usuario sobre el estado de la ejecución es impresa en la salida estándar.

Sobre el diseño de la programación es necesario mencionar que se trata de un diseño orientado a objetos muy sencillo que se combina –además, con algunos elementos de la programación estructurada. Por la sencillez de los elementos a representar en el diseño de la programación no resultó necesario usar herencia ni polimorfismo, aunque sí se usan el encapsulamiento de la información y el almacenamiento conjunto de los datos y las operaciones sobre estos, propios del paradigma de programación orientada a objetos. En el Anexo 2 se muestra un diagrama de clases detallado de la aplicación.

2.1.2. Cálculo de la matriz de distancias

El cálculo de la distancia entre cada par de secuencias es de los procedimientos más sencillos que han sido implementados en este trabajo. El código para el mismo aparece en los ficheros *Sec.hpp*, *Sec.cpp* y *Main.cpp*. El fichero *Sec.hpp* contiene la especificación de la clase *Sec*, utilizada para representar una secuencia de nucleótidos (un alineamiento se puede representar entonces con un arreglo de secuencias - en este trabajo se usó la estructura de datos *vector*, que aparece en la biblioteca estándar de C++), además de algunas funciones relacionadas con esta clase aunque no forman parte de la misma; el fichero *Sec.cpp* contiene la implementación de las funciones declaradas pero no definidas en *Sec.hpp* y, por último, en el fichero *Main.cpp* aparece la función *main*, que constituye el punto de entrada a la aplicación.

A continuación se presenta una versión en pseudocódigo del algoritmo secuencial para facilitar el análisis de la complejidad temporal y espacial del mismo, se asume que el alineamiento de entrada tiene *S* secuencias de longitud *N* cada una.

Algoritmo: *Cálculo de la distancia entre dos secuencias*

Entradas:

sec1, sec2: Secuencias de N nucleótidos

Pseudocódigo:

1. *Inicializar* $\pi_A=0$, $\pi_C=0$, $\pi_G=0$, $\pi_T=0$, $\pi_D=0$, $P_1=0$, $P_2=0$, $Q=0$, $N_D=0$
2. *Para i=0, mientras i<N; inc(i)*
3. | *incrementar los valores de las variables que correspondan*
4. *Fin Para*
5. $\pi_A/=N$
6. $\pi_C/=N$
7. $\pi_G/=N$
8. $\pi_T/=N$
9. $\pi_D/=N$
10. $P_1/=N(N-1)/2$
11. $P_2/=N(N-1)/2$
12. $Q/=N(N-1)/2$

13. N_D/=N(N-1)/2
14. *dist=<fórmula de distancia>*

Algoritmo: *Cálculo de matriz de distancias*

Entradas:

secfile: Nombre del fichero que contiene las secuencias

matfile: Nombre del fichero de salida

Variables usadas:

sec: Matriz de nucléotidos de SxN, las filas representan las secuencias leídas

mat: Matriz de números reales, de SxS, donde se almacenan las distancias

Pseudocódigo:

1. *Leer secuencias desde secfile y almacenarlas en sec*
2. *Reservar espacio en memoria para mat*
3. *Para i=0, mientras i<S, inc(i)*
4. | *mat[n*i+i] = 0*
5. | *Para j=i+1, mientras j<S, inc(j)*
6. | | *mat[n*i+j]=distancia(sec[i],sec[j])*
7. | | *mat[n*j+i]=mat[n*i+j]*
8. | *Fin Para*
9. *Fin Para*
10. *Imprimir los resultados en matfile*

Durante la ejecución del algoritmo, en todo momento se necesita espacio en memoria para almacenar la matriz de *SxN* nucleótidos y la matriz de distancias de *SxS*, además de un número constante de variables auxiliares para la realización de los cálculos, por lo que se puede asegurar que la complejidad espacial es un $\Theta(SN+S^2)$). Para analizar la complejidad temporal es necesario notar que la lectura de las secuencias toma un tiempo $\Theta(SN)$, el cálculo de cada distancia un tiempo $\Theta(N)$, se calculan exactamente $N(N-1)/2$ distancias y la impresión en el fichero de salida de la matriz de distancias un tiempo $\Theta(N^2)$, el resto de las operaciones toman un tiempo constante que no varía cuando se cambia el tamaño de la entrada. Por tanto se puede

concluir que la complejidad temporal de todo el procesamiento es $\Theta(SN+N*N(N-1)/2+N^2)$, o lo que es lo mismo $\Theta(SN+N^3)$.

Solo se realiza el análisis del cálculo de las distancias según el modelo FB. El cálculo de la distancia según el modelo TN93 tiene la misma complejidad que el que se analizó. El modelo TN93 solo se usa cuando se indefine alguno de los logaritmos presentes en la fórmula de la distancia según el otro modelo, en estos casos se lanza una excepción y se recomienza el cálculo de la misma, por lo que el peor caso es aquel en que esto ocurre al analizar el último sitio de la secuencia y por tanto el cálculo de esta distancia toma un tiempo de $\Theta(2N) = \Theta(N)$, o sea, no afecta el resultado del análisis realizado. La cantidad de memoria necesaria también varía solo en una cantidad constante. Por tanto los resultados del análisis realizado para el modelo FB son válidos también cuando es necesario el uso del modelo TN93.

2.1.3. Estimación de los parámetros del modelo y las tasas de sustitución

Los procesamientos necesarios para la estimación de los parámetros del modelo en cuestión y de las tasas de mutación son más numerosos y significativamente más complejos que los necesarios para la estimación de las distancias entre las secuencias. En la Figura 2.2 se muestra un diagrama que describe el orden en que se aplican los algoritmos en esta etapa.

El código correspondiente a la implementación de esta fase aparece en los ficheros Main.cpp, utils.hpp, utils.cpp, Matrix.hpp, Matrix.cpp, PhyloTree.hpp, PhyloTree.cpp, Trie.hpp, Trie.cpp, Sec.hpp y Sec.cpp; estos dos últimos son los mismos que se utilizaron en el cálculo de la distancia evolutiva entre las secuencias, ya que el código que permite la lectura de las secuencias alineadas desde un fichero puede ser reutilizado.

Los ficheros Matrix.hpp y Matrix.cpp contienen la declaración y definición respectivamente de la clase Matrix, la cual constituye una interfaz de alto nivel al tipo de datos gsl_matrix y las funciones relacionadas con el mismo, que aparecen en la GNU Scientific Library. Esta clase se usa para representar matrices cuadradas de cualquier orden y permite realizar sobre las mismas las operaciones necesarias en este trabajo, como la inversión, normalización, cálculo de los valores y vectores propios, multiplicación por un escalar, etc.

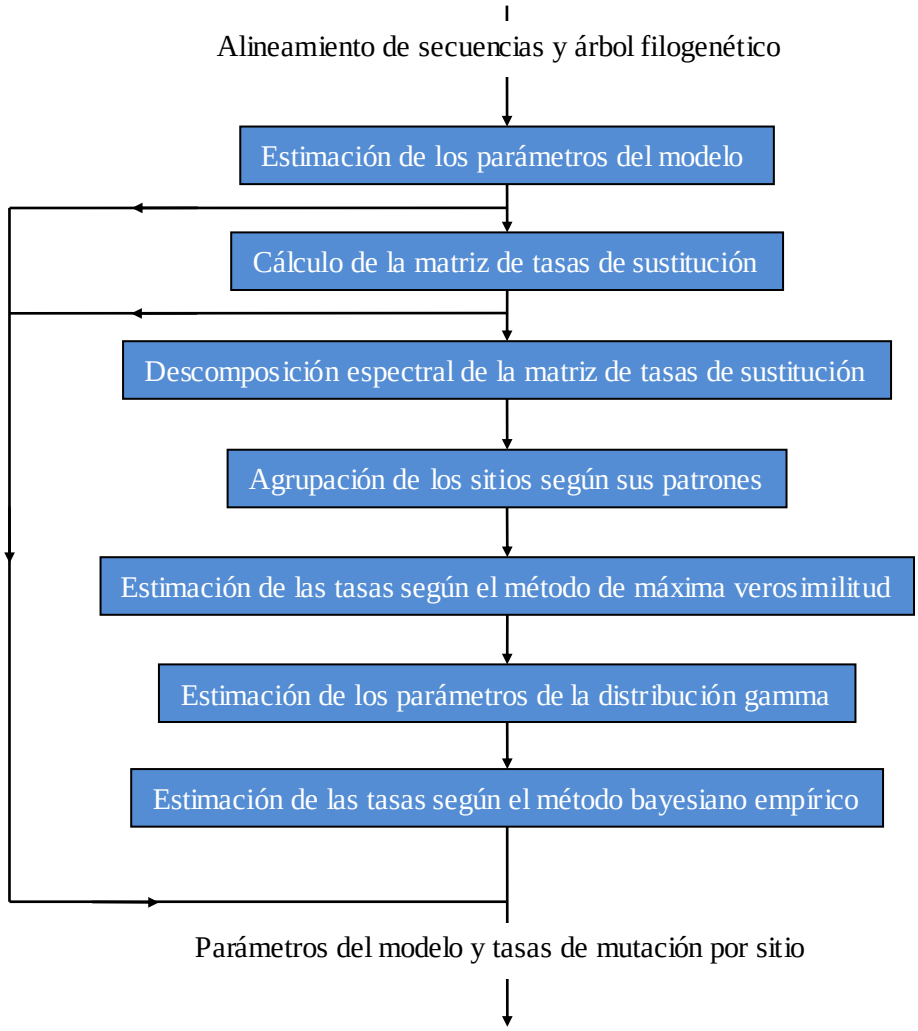

Figura 2.2 Diagrama de flujo del proceso de estimación de los parámetros del modelo probabilístico y las tasas de sustitución por sitio

Los ficheros PhyloTree.hpp y PhyloTree.cpp contienen la declaración y definición respectivamente de las clases PhyloTree, PTNode y PTBranch, las cuales son usadas para representar el árbol filogenético y proveen funciones que permiten leer el árbol de un fichero en formato NEWICK (Las especificaciones de este formato aparecen en http://evolution.genetics.washington.edu/phylip/newick_doc.html), el cálculo de la verosimilitud, etc.

Los ficheros Trie.hpp y Trie.cpp, contienen la declaración y definición respectivamente de la clase Trie, la cual se usa en el agrupamiento de forma eficiente de los sitios según sus patrones.

En los ficheros utils.hpp y utils.cpp aparecen varias funciones de diversa índole que no están directamente relacionadas con las clases mencionadas con anterioridad. Entre las funciones que se implementaron en estos ficheros están la que interpreta los parámetros de la línea de comandos y las opciones modificables por el usuario, las encargadas de estimar los parámetros de los diferentes modelos evolutivos, las que realizan la estimación de las tasas de mutación por sitio y las funciones relacionadas con la distribución gamma que no aparecen en la GSL o cuya implementación en dicha librería no satisface del todo los requerimientos de esta aplicación.

En el fichero Main.cpp aparece solamente la función main, que es el punto de entrada a la aplicación. En la misma se puede apreciar fácilmente la forma en que se ejecutan los distintos algoritmos siguiendo el diagrama de la Figura 2.2. A continuación se presenta el pseudocódigo de cada una de las etapas que componen esta fase con los correspondientes análisis de complejidad espacial y temporal.

2.1.4. Lectura de datos y opciones de configuración

Al inicio de la ejecución de este programa se deben realizar algunas acciones indispensables para el funcionamiento del mismo: leer de los ficheros correspondientes las secuencias alineadas, el árbol filogenético y los valores de las variables que el usuario puede personalizar (por ejemplo, la cantidad máxima de iteraciones permitidas en el algoritmo de maximización). La primera de estas operaciones es exactamente la misma que la que se realiza en el cálculo de la matriz de distancias, donde se vio que su costo computacional es un $\Theta(SN)$ tanto en tiempo como en espacio. Por otro lado, la operación de lectura de las opciones configurables por el usuario toma un tiempo constante y solo se necesitan unos cuantos bytes de memoria para su almacenamiento ya que se trata de unas pocas variables numéricas o booleanas, por lo que podemos afirmar que los costos espacial y temporal de esta etapa es de solo un $\Theta(1)$.

El análisis de la complejidad de la lectura desde fichero del árbol filogenético es un poco más complicado, aún más cuando es necesario, para garantizar una mayor eficiencia en operaciones futuras, guardar en cada nodo hoja del árbol una referencia a la secuencia nucleotídica correspondiente. El pseudocódigo de la lectura desde fichero del árbol filogenético es el siguiente:

***Algoritmo**: Lectura desde fichero del árbol filogenético*

***Entradas**:*

treefile: Fichero en formato NEWICW

Variables usadas:

tabla: Tabla Hash donde se almacenan (referencias a) las secuencias, tomando como llave el nombre de las mismas.

rt: Nodo raíz del árbol

Pseudocódigo:

1. *Para cada secuencia s*
2. *| Insertar(s, tabla)*
3. *Fin Para*
4. *LeerNodo(rt, treefile, tabla)*

***Función**: LeerNodo*

***Parámetros**:*

nodo: Nodo actual

treefile: Fichero en formato NEWICK

tabla: Tabla Hash con las secuencias

***Variables usadas**:*

token: Cadena de caracteres

Pseudocódigo:

1. *LeerSiguiente(token, treefile)*
2. *Si token = "("*
3. *| AdicionarHijo(nodo)*
4. *| LeerNodo(nodo.ultimoHijo, treefile, tabla)*
5. *| LeerDistanciaRama(nodo.ultimoHijo, treefile)*
6. *| LeerSiguiente(token, treefile)*
7. *| Si token = ","*
8. *| | Ir al paso 3*
9. *| Fin Si // si no es una coma, es un paréntesis cerrado y se termina de leer el nodo*
10. *Si no // si no es un paréntesis abierto, es un nombre, o sea, se trata de un nodo hoja*

11. | *nodo.nombre* ← *token*

12. | *nodo.secuencia* ← *tabla.obtener(token)*

13. Fin Si

14. Retornar

Para hacer un análisis de la complejidad espacial de esta operación es necesario notar que se deben almacenar cada uno de los nodos del árbol, pero el espacio que ocupa cada nodo en memoria no es el mismo: los nodos hojas deben almacenar una referencia a la secuencia a la que corresponden y los nodos intermedios deben almacenar una lista de nodos hijos. Este problema se puede evitar asumiendo que todos los nodos tienen reservado espacio para una referencia a una secuencia, aunque esta no sea conocida y, por tanto, en algunos casos esta será una referencia nula. Hacer esta generalización no altera el resultado de la complejidad computacional real ya que solo estamos añadiendo a algunos nodos una cantidad constante de memoria: si I es la cantidad de nodos internos (no hojas) y M es la cantidad de nodos total, la complejidad espacial cambia entonces de $\Theta(M)+\Theta(M-I) = \Theta(M)$ a $\Theta(M)+\Theta(M-I+I)= \Theta(M)$, o sea, no varía. Por otro lado, no es posible conocer de antemano cuántos hijos tiene cada nodo, pero si es posible conocer el resultado de la suma de todos estos, ya que cada nodo, excepto la raíz, es referenciado exactamente una vez, por lo que la complejidad de almacenar estas referencias es también un $\Theta(M)$. Solo falta expresar M en función de N, para ello debe notarse que el árbol con la menor cantidad de nodos es aquel que tiene forma de estrella que solo tiene las S hojas y la raíz, y el árbol con la mayor cantidad de nodos, si se tienen exactamente S hojas es aquel en que todos sus nodos internos tienen exactamente dos hijos (un nodo con un solo hijo no aparece nunca en un árbol filogenético obtenido por alguno de los algoritmos conocidos), la demostración de este hecho resulta muy sencilla ya que un árbol que no cumpla la condición anterior puede ser transformado en uno binario por medio de transformaciones como la que aparece en la Figura 2.3, las cuales solo aumentan, no disminuyen, el número de nodos. Finalmente, es conocido que un árbol de S hojas con bifurcaciones en todos sus nodos internos tiene en total $2S-1$ nodos, por lo que se puede deducir que $S+1 \leq M \leq 2S+1$, lo que implica que $\Theta(M)= \Theta(S)$ y, por tanto, la complejidad espacial generada al almacenar el árbol filogenético es un $\Theta(S)$. A esta expresión debe sumársele el costo correspondiente a almacenar la Tabla Hash con las S secuencias, que es

también un Θ(S), por lo que se puede concluir que la complejidad espacial de toda la fase de lectura de datos es: $\Theta(SN) + \Theta(1) + \Theta(S) = \Theta(SN)$.

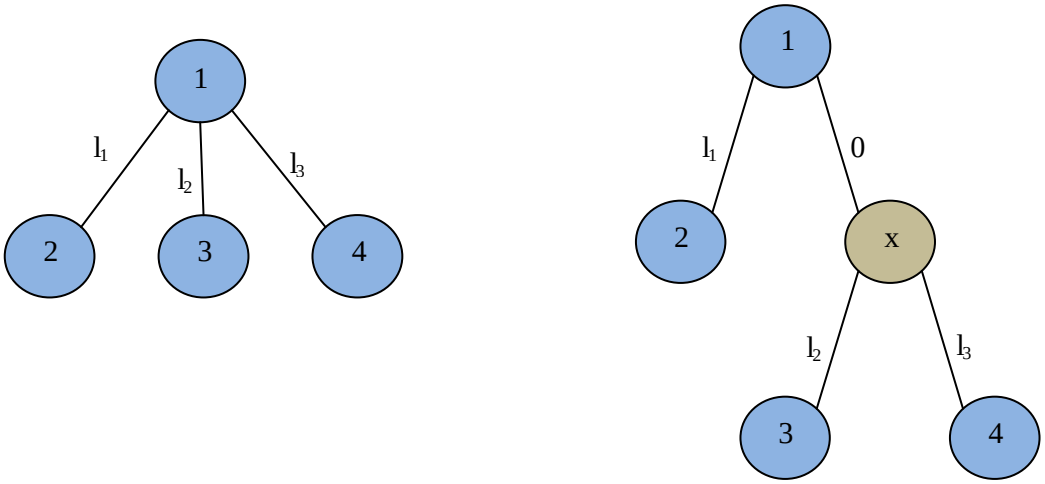

Figura 2.3 Ejemplo de transformación usada para convertir un árbol filogenético arbitrario en uno binario equivalente.

Para conocer la complejidad temporal del algoritmo, se debe notar que primero se realizan un total de *S* inserciones en la Tabla Hash, cada una de las cuales es conocido que toma un tiempo constante (se asume aquí que la longitud del nombre de las secuencias es siempre menor que una cierta constante, por ejemplo: 100), luego se hace una llamada a la función recursiva *LeerNodo*, en la cual solo se realiza un pequeño número de operaciones que toman un tiempo constante, además de las llamadas recursivas. La cantidad de llamadas a la función *LeerNodo* es exactamente *M*, el número de nodos. Por tanto la complejidad temporal de la lectura del árbol filogenético es un $\Theta(S) + \Theta(M)$, pero como ya se mostró que $\Theta(M) = \Theta(S)$, esta expresión se reduce a $\Theta(S)$, por lo que la complejidad temporal de toda la fase de lectura es entonces $\Theta(SN) + \Theta(1) + \Theta(S) = \Theta(SN)$.

2.1.4.1. Estimación de parámetros. Cálculo de la matriz de tasas de sustitución. Descomposición espectral.

Después de leídos los datos de entrada del programa se procede a la estimación de los parámetros del modelo, de los cuales depende la matriz de las tasas de sustitución. En el pseudocódigo que aparece a continuación solo se muestra la forma de estimar los parámetros

π_A, P_1 y β, para lograr claridad. La estimación del conjunto completo de parámetros tiene la misma complejidad que la estimación de estos tres ya que la cantidad de valores a estimar es siempre constante. Luego de la estimación se procede a crear la matriz y realizar su descomposición espectral. A continuación aparece el pseudocódigo correspondiente a esta fase del procedimiento:

Algoritmo: *Estimación de parámetros del modelo probabilístico*

Entradas:

alin: Alineamiento de S secuencias de longitud N

Variables usadas:

Q_r: matriz de tasas de sustitución

Pseudocódigo:

1. $\pi_A \leftarrow 0$, $P_1 \leftarrow 0$ *y* $\beta \leftarrow 0$
2. *Para cada sitio i*
3. | *cntA* ← *0, cntG* ← *0*
4. | *Para cada secuencia s*
5. | | *Si alin(s,i) = 'A'*
6. | | | $\pi_A \leftarrow \pi_A + 1$
7. | | | *cntA* ← *cntA + 1*
8. | | *Si no, Si alin(s,i) = 'G'*
9. | | | | *cntG* ← *cntG + 1*
10. | | | *Fin Si*
11. | | *Fin Si*
12. | *Fin Para*
13. | $P_1 \leftarrow P_1$ + *cntA + cntG*
14. *Fin Para*
15. $\beta \leftarrow$ *fórmula_beta(...)*
16. $Q_r \leftarrow$ *fórmula_matriz(...)*
17. *DescomposiciónEspectral(Q_r)*

Como en el cálculo de la distancia, si al estimar alguno de los parámetros del modelo FB aparece un valor negativo como argumento de un logaritmo, se lanza una excepción y se recomienza el procesamiento desde el principio, esta vez para el modelo TN93. El mismo análisis que se hizo para este caso en el cálculo de las distancias para determinar que la complejidad computacional no varía, es válido también aquí, excepto por el costo adicional que implica la eliminación de los sitios que presentan mutaciones de inserción-eliminación. Una forma eficiente de lograr esta eliminación es marcar los sitios en cuestión y obviarlos en todos los análisis posteriores, lo cual se puede conseguir a un costo en tiempo de un $\Theta(SN)$, aunque en la práctica la cantidad de sitios con estas características es lo suficiente pequeña como para que la eliminación real de los mismos no disminuya significativamente el rendimiento.

Es fácil ver, en el análisis de la complejidad espacial, que solo se adiciona a los datos almacenados en la fase anterior los valores de los parámetros y la matriz Q_r junto a su descomposición espectral, por lo que el incremento en complejidad es solo un valor constante que no depende del tamaño de la entrada. Por tanto la complejidad espacial de esta fase es $\Theta(SN) + \Theta(SN) + \Theta(1) = \Theta(SN)$.

En las instrucciones de la 1 a la 14 aparecen dos ciclos anidados de longitud N y S respectivamente, el resto de las operaciones toma un tiempo constante, por lo que la complejidad temporal en ese fragmento es un $\Theta(SN)$. Las instrucciones 15 y 16 también toman un tiempo constante, solo falta determinar entonces la complejidad de la operación de descomposición de la matriz, que usualmente depende del algoritmo utilizado. Afortunadamente el orden de la matriz, que es lo único de que depende el rendimiento de todos los algoritmos conocidos para esta tarea, siempre es, a lo sumo, 5, por lo que el tiempo que debe ser invertido en esta operación (y también el espacio en memoria) puede ser considerado un valor constante independientemente del método seleccionado. Por esta razón en todos los análisis de complejidad sucesivos se da por sentado que las operaciones sobre la matriz de tasas de sustitución tienen una complejidad de $\Theta(1)$, tanto en tiempo como en espacio. Entonces se puede concluir que la complejidad temporal de esta fase es de $\Theta(SN)+ \Theta(1) = \Theta(SN)$.

2.1.4.2. Cálculo de los patrones por sitio

Esta operación resulta muy sencilla al usar la estructura de datos *Trie*. Basta con ir adicionando cada sitio de las secuencias (que en la práctica son una cadena de caracteres de longitud S) a la estructura y la misma se encarga de comprobar si ese patrón ya ha sido insertado o no. Al final los patrones de los sitios quedan almacenados en las hojas del árbol construido. Se conoce que insertar una cadena de longitud S en un *Trie* tiene siempre una complejidad temporal de $\Theta(S)$, además el caso peor para el almacenamiento ocurre cuando todos los sitios son distintos y son necesarios $SN+1$ nodos en el árbol. Para extraer los patrones de la estructura basta con almacenar una referencia a ellos en el momento de su creación, lo que implica un aumento del tiempo de ejecución en una constante al final de cada inserción. Por tanto, como hay N sitios a examinar, se puede afirmar que el costo en tiempo de ejecución de esta operación es un $\Theta(N)\Theta(S)\Theta(1) = \Theta(NS)$, el aumento del costo en espacio es también un $\Theta(NS)$ en el caso peor y un $\Theta(S)$ en el mejor (cuando todos los sitios son iguales), por lo que el mismo se mantiene en $\Theta(NS)$.

2.1.4.3. Estimación de las tasas de mutación por sitio de máxima verosimilitud

Una vez que se conocen todos los parámetros del modelo evolutivo se procede a la estimación de las tasas de mutación por sitio. El método bayesiano empírico utilizado en este trabajo requiere que las mismas sean estimadas previamente de alguna otra forma. Para esto se seleccionó el criterio de máxima verosimilitud. El pseudocódigo del algoritmo correspondiente se muestra a continuación:

Algoritmo: *Estimación de las tasas de mutación, método de máxima verosimilitud*

Entradas:

patt: Arreglo que contiene referencias a los sitios representantes de cada patrón

tree: árbol filogenético

Variables usadas:

prates: tasas de mutación por patrón

srates: tasas de mutación por sitio

θ: parámetros del modelo

Pseudocódigco:

1. *Para cada patrón p en patt*
2. *| prates ← minimizar(-1*Verosimilitud(tree, p, θ), [0,∞))*
3. *Fin Para*
4. *srates ← procesar(prates)*

Función: *verosimilitud_N*

Entradas:

nodo: nodo actual

p: sitio representante de un patrón

θ: parámetros del modelo(incluye la tasa de mutación)

Salidas:

lhi: arreglo de 5 (4 si el modelo es el TN93) valores reales, correspondientes a la verosimilitud del nodo condicionada a la presencia en él, en ese sitio en particular, de cada una de las bases nitrogenadas

Pseudocódigo*:*

1. *Si Es_Hoja(nodo)*
2. *| lhi ← [0,0,0,0,0];*
3. *| lhi[p.base(nodo)] ← 1*
4. *Si no*
5. *| Para cada hijo h de nodo*
6. *| | verosimilitud_N(h, p, θ)*
7. *| Fin Para*
8. *| lhi ← fórmula(…)*
9. *Fin Si*
10. *Retornar lhi*

El análisis de la complejidad espacial en esta fase es muy sencillo, ya que el único incremento en los requerimientos de memoria es el espacio necesario para almacenar las referencias a los sitios representantes de cada patrón encontrado, las tasas de mutación por patrón, y luego las

tasas de mutación por sitio, lo que, en el peor caso, representa un incremento de un $\Theta(3N)=\Theta(N)$. Por tanto, la complejidad espacial para esta fase resulta $\Theta(SN)+\Theta(N)=\Theta(SN)$.

El análisis de la complejidad temporal en esta fase resulta un poco complejo. Primero debe notarse que en la línea 2 se hace una llamada a la función minimizar, para la cual el usuario puede seleccionar la tolerancia permitida y la cantidad máxima de iteraciones a realizar, por lo que es necesario introducir una nueva variable en el análisis, I, que representa la cantidad de iteraciones que realiza el algoritmo de minimización. El algoritmo de minimización de Brent que se utilizó en este trabajo garantiza que la complejidad temporal del mismo es un $\Theta(I*g(f))$, donde $g(f)$ es el costo asociado a una evaluación de la función objetivo en un punto dado, en este caso la función *Verosimilitud*. Esta función, a su vez, hace una llamada a la función *verosimilitud_N* en la raíz del árbol y retorna la suma del producto de sus resultados por las probabilidades estacionarias del modelo. Como la función que lee el árbol de fichero, *verosimilitud_N* se ejecuta exactamente una vez por cada nodo del árbol. La fórmula que aparece en la línea 8 tiene tantos términos como hijos tiene el nodo en cuestión, el modo de analizar esta situación es análogo al que se usó en la función de lectura. Por lo cual se puede concluir que el costo en tiempo de una evaluación de la función *Verosimilitud* es, al igual que para la función de lectura, un $\Theta(S)$. Por último, el costo de asignar a cada sitio su tasa correspondiente es un $\Theta(N)$, ya que esto se puede lograr mediante un simple recorrido por los mismos. Por tanto, la complejidad temporal de esta fase es un $\Theta(PIS)+\Theta(N)$, siendo P la cantidad de sitios distintos. Una cota superior al valor de P es N, la cantidad total de sitios, por lo que se puede afirmar entonces que la el costo en tiempo de ejecución de esta fase es un $O(NIS)+O(N)=O(NIS)$. En la práctica se observa que al aumentar el número de secuencias, la cantidad de sitios distintos tiende a aumentar a su vez, por lo que la sustitución de P por N resulta ser una aproximación aceptable cuando el volumen de datos aumenta.

2.1.4.4. Estimación de los parámetros de la distribución gamma y su discretización

En la aplicación del teorema de Bayes en la estimación de las tasas de mutación es necesario conocer las probabilidades *a priori* de las mismas. Se asume que las tasas siguen una distribución gamma, cuyos parámetros α y β, así como la media global, son estimados a partir de los datos obtenidos por el método de máxima verosimilitud. También es necesaria la discretización de esta distribución para eliminar la necesidad de resolver integrales complejas,

para lo cual se crean K clases discretas (donde K puede ser modificado por el usuario) para los valores de las tasas con igual probabilidad y con la media igual a aquella estimada por máxima verosimilitud.

La estimación de los parámetros α y β se realiza por el método de máxima verosimilitud (Hahn and Shapiro 1994), el cual tiene una complejidad de $\Theta(N)$ tanto en tiempo como en espacio. La discretización de la distribución gamma implica la evaluación K veces de la función inversa a la función de distribución acumulativa gamma. El costo de la evaluación de esta función, debido a su estimación de forma iterativa, es elevado; sin embargo, este puede ser considerado constante ya que no depende del tamaño de la entrada, ni de las decisiones del usuario. Por tanto, el costo de la discretización, tanto en tiempo como en espacio, es un $\Theta(K)$, por lo que el costo final de esta fase es un $\Theta(N+K)$.

2.1.4.5. Estimación de las tasas de mutación *a posteriori*

Una vez que se han completado las fases anteriores se puede proceder a la estimación de las tasas de mutación por sitio a posteriori mediante la aplicación del teorema de Bayes. Al igual que en el método de máxima verosimilitud, se hace solo para los sitios con patrones distintos con el propósito de ganar en eficiencia. El pseudocódigo de este algoritmo se muestra a continuación:

Algoritmo: *Estimación de las tasas de mutación a posteriori.*

Entradas:

patt: Arreglo que contiene referencias a los sitios representantes de cada patrón

tree: árbol filogenético

pr: arreglo de valores reales con las probabilidades a priori de las tasas (uniforme)

r: arreglo de valores reales con las clases discretas de las tasas

Variables usadas:

prates: arreglo de valores reales con la estimación de las tasas por patrón

srates: arreglo de valores reales con la estimación de las tasas por sitio

Pseudocódigo

1. *Para cada patrón p en patt*
2. | *prates[p] = 0*

```
3.  |   denom = 0
4.  |   Para i ← 0, mientras i < K; inc(i)
5.  |   |   prates[p] ← prates[p] + r[i]*pr[i]*Verosimilitud(tree, p, ..., r[i])
6.  |   |   denom ← denom + pr[i]*Verosimilitud(tree, p, ..., r[i])
7.  |   Fin Para
8.  |   prates[p] ← prates[p]/denom
9.  Fin Para
10. srates ← procesar(prates)
```

El análisis de la complejidad de esta fase resulta sencillo ya que la mayoría de las consideraciones necesarias ya han sido realizadas en el análisis del primer método. Es por ello que fácilmente se puede afirmar que la complejidad espacial de esta fase es un $\Theta(SN+K)$ y la temporal un $\Theta(PKN) + \Theta(N) = \Theta(PKN)$, que como habíamos dicho antes es un $O(KN^2)$.
Como esta es la última fase del proceso de estimación de parámetros del modelo y las tasas de sustitución, ya se está en condiciones de conocer la complejidad computacional global del procesamiento. La misma es, en el caso del costo en memoria, un $\Theta(SN+K)$ y, en el caso del costo en tiempo de ejecución, un $\Theta(SN + PIS + N + K + PKN) = \Theta(PIS + PKN)$ o un $O(NIS+N^2K)$.

2.2. Distribución de la carga computacional

Una vez que se conoce el algoritmo secuencial se puede proceder a buscar los segmentos que pueden ser ejecutados en paralelo, especialmente aquellos que presentan una complejidad computacional elevada y forman los cuellos de botella de la aplicación. Para que una fase de un procesamiento pueda ser llevada a cabo en paralelo eficientemente es necesario que en la misma se realicen secuencialmente varias tareas que no dependan unas de las otras, o que, al menos, la dependencia sea mínima para que el costo adicional inherente a la comunicación entre los distintos procesos no contrarreste la ganancia obtenida por la paralelización. Un buen ejemplo de estos segmentos son los ciclos en los cuales las operaciones que se realizan en cada iteración no dependen de aquellas que se realizaron en las iteraciones anteriores. En este trabajo se localizaron cuatro etapas susceptibles de ser paralelizadas:

- El cálculo de las distancias entre los distintos pares de secuencias
- El proceso de estimación de los parámetros del modelo
- La estimación de las tasas de mutación por sitio mediante MMV
- La estimación de las tasas de mutación por sitio mediante MBE

La forma de paralelizar las dos últimas es la más sencilla de todas, por lo que será expuesta primero. En ambas etapas se debe realizar P veces la misma operación sobre P conjuntos de datos distintos, siendo P la cantidad de patrones encontrados. Teniéndose k procesos ejecutándose en paralelo, se pueden distribuir las P operaciones lo más equitativamente posible entre ellos, haciendo que r procesos realicen $q+1$ operaciones y los otros $k-r$ realicen q operaciones, siendo $P = kq + r$, con $0 \leq r < k$. Al finalizar los últimos $k-1$ procesos las operaciones asignadas, estos envían sus resultados al proceso restante (el proceso 0) y este a su vez se encarga de realizar el procesamiento posterior y enviar los datos necesarios para continuar el trabajo a los demás o simplemente imprimir en la salida los resultados obtenidos entre todos.

En la estimación de los parámetros del modelo se hizo una distribución del trabajo parecida a la anterior. En esta, cada parámetro es estimado haciendo un conteo de los eventos que representa sobre el alineamiento de las secuencias, mediante la realización de un recorrido por columnas sobre el mismo. La cantidad de columnas a recorrer fue distribuida equitativamente entre los procesos de la misma forma que en la fase vista con anterioridad y al finalizar se hace una reducción por medio de la operación suma de los parámetros hacia el proceso 0, el cual se encarga de dividir entre los totales correspondientes y enviar los resultados hacia todos los procesos, para luego seguir el procesamiento normalmente.

La paralelización del cálculo de la matriz de distancias resulta un poco más compleja, ya que no se trata de un solo ciclo muy extenso, sino de dos ciclos anidados donde el más interno no siempre realiza la misma cantidad de operaciones, ya que al ser una matriz simétrica solo se calculan los elementos que están por encima de la diagonal principal. Esta característica presenta el problema de que si se paraleliza el ciclo más externo de la forma antes expuesta, la distribución de la carga computacional no resulta equitativa, y si, por el contrario, se paraleliza el ciclo más interno, solo se realiza una distribución equitativa en los ciclos de mayor longitud, pero en los más cortos se mantiene el mismo problema, además de que aumenta el costo

computacional asociado a las tareas de comunicación. Una solución más elegante es convertir estos dos ciclos anidados en uno solo de longitud $\frac{S(S-1)}{2}$ y paralelizar este como se ha visto antes. Para lograr este objetivo en necesario entonces establecer una correspondencia biunívoca entre los números enteros del 0 al $\frac{S(S-1)}{2}-1$ y los pares de números enteros [i , j], con $0 \leq i < \frac{S(S-1)}{2}$ e $i < j < \frac{S(S-1)}{2}$. En este trabajo se utiliza la correspondencia dada por la función biyectiva $f(i, j) = i\frac{(2S-i-1)}{2} + j$.

2.3. Manual de usuario

La sencillez es una de las características principales de la aplicación desarrollada. Ambos ejecutables tienen la forma de un comando clásico de UNIX, son ejecutados desde la línea de comandos y toman como parámetros las entradas que necesitan. Como se usó una biblioteca MPI para su desarrollo no debe llamárseles directamente, sino por medio de otro comando que depende de la plataforma y de la implementación específica de MPI que se use, típicamente *mpirun*.

La ejecución del programa *DistanceCalc* es la más sencilla de las dos. Este toma dos parámetros de la línea de comandos: el camino al fichero con las secuencias en formato FASTA y el camino al fichero donde se debe almacenar la matriz. Un ejemplo de ejecución de este comando es:

$ mpirun –np 3 DistanceCalc secuencias.fas matriz.txt

Ambos parámetros son obligatorios, intentar su ejecución con menos parámetros o con más solo resulta en un mensaje de error. Los caminos a los ficheros de entrada pueden ser relativos o absolutos, la única restricción es que los ficheros tengan los permisos de lectura y escritura correspondientes. Si el fichero de salida no existe el mismo es creado, si ya existe entonces se sobrescribe. El fichero de entrada debe contener la información en formato FASTA, si no es así el programa debería reaccionar y dar un mensaje de error, pero esto no se garantiza: en algunos casos pudiera ocurrir que se lea el fichero y se produzca el procesamiento con resultados erróneos. Se añadió al formato FASTA una restricción adicional que no aparece en la especificación original del mismo: Los nombres de las secuencias no pueden tener caracteres espacio (“ ”) ni dos puntos (“:”). Esto se hace porque estos nombres son copiados

íntegramente al fichero de salida, el cual sirve de entrada al FastME y este último no funciona bien en estos casos.

El programa *ParameterEstim* también se ejecuta por línea de comandos, pero su formato es otro:

$ (mpirun –np …) ParameterEstim <secfile> <treefile> [<configfile>]

Donde <secfile> es un camino a un fichero de secuencias con las mismas especificaciones que en el programa anterior. El parámetro <treefile> es un camino a un fichero en formato NEWICK el cual contiene la especificación del árbol filogenético. El último parámetro es opcional e indica el camino a un fichero de configuración cuyo formato será explicado más adelante. Al igual que en el programa anterior, *ParameterEstim* intentará detectar cualquier inconsistencia que exista entre ambos ficheros y cualquier error de formato, pero algunos errores pueden pasar inadvertidos.

El fichero de configuración tiene un formato sencillo, al estilo de los ficheros de configuración del Sistema Operativo UNIX. Las líneas vacías y las que comienzan con un carácter "#" son ignoradas, el resto debe tener la forma <nombre_variable> <valor>, donde <nombre_variable> es el nombre del parámetro que se desea modificar y <valor> el nuevo valor asignado. Si uno de los posibles identificadores que pueden aparecer en el fichero no está presente se asume un valor por defecto. A continuación se muestra una lista de los nombres de todos los parámetros que se pueden configurar, con su valor por defecto y una breve descripción de su significado:

- *maxNumIter 20*: Este parámetro solo acepta valores enteros positivos, representa el número máximo de iteraciones que se permitirá que realice el algoritmo de minimización utilizado en el método de máxima verosimilitud de estimación de las tasas de mutación por sitio.
- *absErr 0.0000000001*: Este parámetro acepta solo valores reales positivos, representa el error absoluto máximo permitido al algoritmo de minimización.
- *relErr 0.00001*: Este parámetro acepta solo valores reales positivos, representa el error relativo máximo permitido al algoritmo de minimización.
- *forceTN93 0*: Este parámetro solo acepta valores booleanos, si es cero (0) se intentará usar el modelo FB y si falla éste se utilizará el modelo TN93 en su lugar, por el contrario si su valor es uno (1) se usará desde el principio el modelo TN93. El formato

de este parámetro fue escogido teniendo en cuenta que en un futuro se añadirán los demás modelos existentes en la literatura, a cada uno de los cuales se puede asignar un número distinto para su identificación, en este caso el nombre del parámetro se cambiará a model.

- *fileOutput 0*: Este parámetro solo acepta valores booleanos, si su valor es uno (1) el programa creará, en el directorio donde se realiza la ejecución, un fichero con nombre output.txt (si ya existe se sobrescribe) en el que aparecerán los parámetros del modelo y las tasas de mutación por sitio estimadas por ambos métodos.
- *ncate 25*: Este parámetro solo acepta valores enteros, representa la cantidad de clases que se utilizarán en la discretización de la distribución gamma.

3. RESULTADOS Y DISCUSIÓN

En el presente capítulo se introducen las principales formas de medir el rendimiento de un programa, en particular uno en paralelo. Luego se describen las distintas pruebas realizadas y se hace un breve análisis de los resultados obtenidos.

3.1. Métodos utilizados en la evaluación del desempeño

Existen diversas formas de medir el rendimiento de un programa, en especial de programas en paralelo. En este trabajo se mide el desempeño de la aplicación desarrollada mediante la comparación con otras dos implementaciones secuenciales de los mismos algoritmos.

Antes de la implementación en paralelo de los algoritmos tratados en el trabajo, en el Grupo de Bioinformática de la UCLV se disponía del toolbox del Matlab llamado *Molecular Biology and Evolution Toolbox* (MBEtoolbox) (Cai, Smith et al. 2005; Cai, Smith et al. 2006), el cual provee un gran número de algoritmos útiles para el análisis filogenético, además de otros adicionados por investigadores del mencionado Grupo, incluyendo todos los implementados en este trabajo. Esta herramienta fue la utilizada para comprobar que los resultados obtenidos por la aplicación desarrollada fueron correctos.

Para tener una medida exacta de los beneficios de la paralelización de un procedimiento computacional, es necesario hacer la comparación con la mejor versión secuencial conocida (Grama, Gupta et al. 2003). El MBEtoolbox no cumple con esta condición, pues está implementado en Matlab, que es un lenguaje interpretado y, por ende, su ejecución trae aparejado un incremento en el costo computacional en comparación con los lenguajes que generan código de máquina. Además la implementación de algunos algoritmos en esta herramienta deja mucho que desear en lo que se refiere a eficiencia, ya que en ocasiones se elige la forma más clara y sencilla en lugar de la más óptima.

Por todas estas razones se decidió implementar en C++ una versión secuencial análoga a la desarrollada para este trabajo, pero sin la carga adicional de la comunicación, ni el cálculo extra asociado a la distribución de las tareas entre los diferentes procesos.

3.1.1. Métricas utilizadas

Quizás, la métrica más popular en la comparación del rendimiento de dos programas es la conocida como *Speedup* o *Ganancia*, la cual expresa la razón entre los tiempos de ejecución de los mismos, dando una medida de cuán rápido es uno con respecto al otro. Esta está dada por la fórmula:

$$S = \frac{T_1}{T_2} \tag{3.1}$$

Donde T_1 y T_2 son los tiempos de ejecución de las versiones del procedimiento que se que comparan.

Cuando se trata de un programa en paralelo, la *Ganancia* por sí sola no es una medida lo suficientemente buena, ya que no tiene en cuenta todos los aspectos involucrados, como la cantidad de nodos de procesamiento. Para evaluar el desempeño de un software de este tipo se usa la *Eficiencia*, que es una medida del aprovechamiento eficiente de los recursos disponibles. Esta está dada por la fórmula:

$$E = \frac{T_S}{T_P p} = \frac{S}{p} \tag{3.2}$$

Donde T_S y T_P son los tiempos de ejecución de la mejor versión secuencial conocida y la versión en paralelo respectivamente y p es la cantidad de nodos de procesamiento utilizados. El valor de esta medida siempre pertenece al intervalo [0,1], siendo 1 el mejor valor posible. Si esta medida se multiplica por 100, representa el por ciento de los recursos que son aprovechado en el algoritmo principal y no se dedican a tareas de comunicación o sincronización, o simplemente se desperdician esperando por los demás procesos.

3.2. Comparaciones utilizando diferentes número y tamaño de secuencias

Los experimentos realizados se llevaron a cabo en dos plataformas diferentes, para probar el rendimiento de la herramienta desarrollada en las distintas configuraciones de hardware. Las primeras pruebas se realizaron en una sola computadora con varios procesadores, con un espacio de direcciones compartido. El procesador utilizado fue un Intel Core 2 Quad a 2.33 GHz, con una memoria RAM DDR2 de 2.0 GB y con el sistema operativo Ubuntu 9.04. En

esta máquina se ejecutó la versión en Matlab, la versión secuencial en C++ y la versión paralela con uno, dos, tres y cuatro procesos.

La segunda configuración de hardware utilizada fue un clúster de seis computadoras en una red privada de tipo GigaBit Ethernet. Cada una de estas computadoras posee un procesador Pentium IV a 3.0 GHz, memoria RAM DDR de 1.0 GB y sistema operativo Debian Etch. En esta plataforma solo se ejecutaron las versiones escritas en C++, llevando la versión paralela hasta seis procesos. La versión en Matlab no fue posible ejecutarla en el clúster, pero esta inconveniencia no representa un problema serio, ya que el Matlab no está diseñado para este tipo de arquitecturas y, por tanto, solo hubiera podido utilizar un único nodo de los disponibles, en el que debería mostrar un desempeño inferior al observado previamente al usar el otro hardware.

En ambas plataformas fueron utilizados dos conjuntos de datos diferentes, el primero es un alineamiento considerado de tamaño medio, con un total de 749 secuencias de 1702 nucleótidos. El segundo, con 2963 secuencias de 1744 nucleótidos, ya tiene un tamaño considerable. Cada prueba fue repetida varias veces y se tomó la media de los tiempos de ejecución observados para su comparación. Durante este proceso fue posible también comprobar que la aplicación desarrollada produce los resultados correctos al compararse los mismos con aquellos producidos por la implementación en Matlab.

En las tablas que aparecen a continuación se muestran los resultados obtenidos en las diferentes pruebas realizadas. En las mismas aparecen, para cada una de las combinaciones de hardware, datos de entrada y fases del procedimiento antes mencionados, la media de los tiempos de ejecución de cada una de las implementaciones, la ganancia que representan las implementaciones en C++ cuando se les compara con la implementación en Matlab, la ganancia de la implementación en paralelo cuando se ejecuta con 1 a 6 procesadores (en el caso de la computadora con varios procesadores solo se lleva el números de procesos hasta 4) comparada con la versión secuencial de la misma y por último la eficiencia en el uso de los recursos computacionales que presenta la implementación en paralelo.

En la Tabla 3.1 aparecen los resultados de las pruebas realizadas en la máquina multiprocesador con los datos de entrada más pequeños. En esta resulta interesante la diferencia entre los resultados obtenidos en la ejecución de la fase de cálculo de las distancias evolutivas con los obtenidos en la fase de estimación de parámetros. En la primera fase la

versión secuencial en C++ es la que mejor desempeño muestra, ya que, aunque la versión en paralelo presenta tiempos de ejecución inferiores, la eficiencia con que trabaja decrece de manera muy pronunciada, llegando a caer hasta el 46% cuando se utilizan 4 procesos. Por el contrario, en la segunda fase es la versión en paralelo ejecutada con 4 procesos la que mejor trabaja al conseguir disminuir el tiempo de la versión secuencial en C++ en 3.793 veces con una eficiencia del 94.8%. Esta diferencia entre el rendimiento de la versión paralela en las dos fases es causada por la diferencia en el costo computacional de las mismas, ya que en la primera fase este es tan pequeño que permite que el rendimiento sea significativamente afectado por el costo de las operaciones de comunicación entre los procesos, mientras que el costo de la segunda fase es lo suficientemente alto como para hacer insignificante en comparación el de las operaciones de comunicación. En ambas fases del procesamiento, el rendimiento de las implementaciones en C++ superan ampliamente el del Matlab, logrando una disminución del tiempo de ejecución máxima de 672.248 veces cuando se ejecutó la versión paralela con 4 procesos.

Tabla 3.1. Resultados de las pruebas con 749 secuencias en una máquina de 4 procesadores

		MBE Toolbox	*C++ secuencial*	*1 proceso*	*2 procesos*	*3 procesos*	*4 procesos*
Cálculo de las distancias	T E [a]	2m55.480s	6.601s	6.999s	4.324s	3.736s	3.587s
	G-MBET[b]	-	26.584	25.072	40.583	46.97	48.921
	G-C++ s.[c]	-	-	0.943	1.527	1.767	1.840
	Eficiencia[d]	-	-	0.943	0.763	0.589	0.460
Estimación de parámetros	T E	35h25m	11m59.596s	12m0.624s	6m5.264s	4m13.643s	3m9.695s
	G-MBET	-	177.213	176.96	349.123	502.762	672.248
	G-C++ s.	-	-	0.999	1.970	2.837	3.793
	Eficiencia	-	-	0.999	0.985	0.945	0.948

[a] TE: Tiempos de ejecución observados en cada una de las implementaciones.

[b] G-MBET: Ganancia de las implementaciones en C++ con respecto al MBEtoolbox.

[c] G-C++ s: Ganancia de la implementación en paralelo con distintas cantidades de nodos con respecto de la variante secuencial en C++.

[d] Eficiencia: Eficiencia alcanzada por la cada una de las configuraciones de la versión en paralelo.

En la Tabla 3.2 se muestran los resultados obtenidos en la misma máquina con el alineamiento de 2963 secuencias para ambas fases del procesamiento. En esta prueba ocurre algo similar a lo ocurrido en la anterior: la versión paralela se comporta mucho mejor en la segunda fase del procesamiento que en la primera. Sin embargo, se puede afirmar que en la primera fase su desempeño mejora con respecto al observado anteriormente, elevando su eficiencia con 4 procesadores hasta un 66.1%. Nuevamente la versión en Matlab es la que presenta el peor desempeño, de hecho, el proceso de estimación de parámetros usando esta implementación no se logró terminar a tiempo y tuvo que ser interrumpido tras 7 días de ejecución.

Tabla 3.2. Resultados de las pruebas con 2963 secuencias en una máquina de 4 procesadores

		MBE Toolbox	**C++ secuencial**	**1 proceso**	**2 procesos**	**3 procesos**	**4 procesos**
Cálculo de las distancias	T E	46m20.138s	1m54.876s	2m4.902s	1m9.294s	52.718s	43.397s
	G-MBET	-	24.201	22.265	40.125	52.734	64.06
	G-C++ s.	-	-	0.920	1.658	2.179	2.647
	Eficiencia	-	-	0.920	0.829	0.726	0.661
Estimación de parámetros	T E	>1semana	80m20.4s	80m29.7s	40m34.8s	28m10.5s	21m4.0s
	G-MBET	-	>150	>150	>300	>400	>600
	G-C++ s.	-	-	0.998	1.98	2.851	3.814
	Eficiencia	-	-	0.998	0.99	0.95	0.954

En la Tabla 3.3 aparecen los primeros resultados obtenidos en el clúster, correspondientes al alineamiento más pequeño. Con esta configuración de hardware se pueden ejecutar con la versión paralela, hasta 6 procesos simultáneamente de modo eficiente. Es en esta arquitectura donde la implementación paralela propuesta en este trabajo muestra toda su potencialidad disminuyendo en más de 5 veces en el caso del cálculo de las distancias, el tiempo de ejecución de la versión secuencial, con una eficiencia de 83.9%. En la fase de estimación de parámetros es absoluta la superioridad de la versión paralela, al lograr, con 6 procesos, un tiempo de ejecución de solo 4 minutos y 50.4 segundos, que al compararlo con los 28 minutos y 41.7 segundos de la versión secuencial representa una disminución de un 80% del costo computacional asociado al procedimiento, manteniéndose un excelente 98.8% de eficiencia en el uso de los recursos. Este resultado supera con creces la disminución del costo computacional planteada en la hipótesis del trabajo de un 25%, sobre todo si se tiene en cuenta

que el 80% de disminución se logró contra la versión secuencial, la cual, se vio con anterioridad, supera con creces el rendimiento de la versión en Matlab.

Tabla 3.3. Resultados de las pruebas con 749 secuencias en un clúster de 6 computadoras

		C++ secuencial	**1 proceso**	**2 procesos**	**3 procesos**	**4 procesos**	**5 procesos**	**6 procesos**
CD	T E	1m1.262s	1m1.769s	32.008s	22.008s	17.049s	14.048s	12.168s
	G-C++ s.	-	0.992	1.914	2.784	3.593	4.330	5.035
	Eficiencia	-	0.992	0.957	0.928	0.898	0.866	0.839
EP	T E	28m41.7s	28m48.5s	14m25.7s	9m37.5s	7m13.6s	5m47.6s	4m50.4s
	G-C++ s.	-	0.996	1.989	2.981	3.971	4.953	5.929
	Eficiencia	-	0.996	0.994	0.994	0.993	0.991	0.988

En la Tabla 3.4 aparecen los resultados de las pruebas realizadas en el clúster con el alineamiento de mayor tamaño, en las cuales se aprecia un comportamiento muy parecido al observado en las pruebas anteriores. La diferencia entre las más de 3 horas de tiempo ejecución de la versión secuencial en la segunda fase y los 32 minutos y 30.6 segundos de la versión paralela, indica una significativa superioridad de esta última sobre la primera, lo cual se hace mucho más evidente al observar el sobresaliente 99% de eficiencia alcanzado (el mayor entre todas las pruebas realizadas). En la primera fase del procesamiento también se desempeña muy bien la versión paralela, elevando su eficiencia hasta un 88.4%, el cual es ligeramente superior al logrado en las pruebas anteriores para esta misma fase en clúster y ampliamente por encima de los valores de eficiencia alcanzados en la computadora multiprocesador.

Tabla 3.4. Resultados de las pruebas con 2963 secuencias en un clúster de 6 computadoras

		C++ secuencial	**1 proceso**	**2 procesos**	**3 procesos**	**4 procesos**	**5 procesos**	**6 procesos**
CD	T E	16m24.6s	16m44.0s	8m33.2s	5m49.9s	4m28.4s	3m39.2s	3m5.7s
	G-C++ s.	-	0.980	1.919	2.814	3.668	4.492	5.302
	Eficiencia	-	0.980	0.959	0.938	0.917	0.898	0.884
EP	T E	3h13m7s	3h13m25s	1h37m6s	1h4m49s	48m35.3s	38m53.0s	32m30.6s
	G-C++ s.	-	0.998	1.989	2.98	3.975	4.967	5.941
	Eficiencia	-	0.998	0.994	0.993	0.994	0.993	0.990

3.2.1. Análisis de los resultados obtenidos

En los resultados expuestos anteriormente se aprecia la mejora que representa la nueva implementación con respecto al MBEtoolbox en lo que a rendimiento se refiere.

Usando la versión secuencial de la aplicación desarrollada se logran disminuir los tiempos de ejecución en más de 150 veces, llevándolos en el caso de 2963 secuencias, desde más de una semana a menos de dos horas, lo que disminuye considerablemente la probabilidad de que el procesamiento se detenga por causas ajenas al mismo, como pueden ser las interrupciones del fluido eléctrico, tan comunes en estos tiempos debido a la situación económica del país.

A pesar del gran aumento del rendimiento alcanzado a partir del uso del lenguaje de programación C++, la implementación secuencial tiene sus límites:

Como se vio en los análisis de complejidad computacional, el tiempo de ejecución de la fase de estimación de parámetros, que es la más costosa, es aproximadamente proporcional a la cantidad de secuencias y a la longitud de estas, sobre todo cuando estos valores son elevados. Por esta razón es de esperarse que un marcado incremento del volumen de datos a analizar provoque una elevación en los tiempos de ejecución que haga impracticable su uso.

Se ha demostrado que la exactitud de los métodos utilizados depende en gran medida de la cantidad de datos que se analicen, por lo que resulta conveniente analizar el mayor volumen de secuencias posibles. La cantidad de información disponible hoy en día es más que suficiente para garantizar cualquier requisito de exactitud, solo hace falta la capacidad para procesarla, de ahí la utilidad de la implementación en paralelo. Nótese que con solo cuatro procesos se logró reducir el tiempo de ejecución unas 642.278 veces con respecto a la implementación en Matlab en las primeras pruebas realizadas, lo cual representa una disminución del costo computacional de más del 99%, absolutamente superior al valor esperado en la hipótesis.

El límite al incremento de rendimiento que se puede lograr, incrementando la cantidad de procesadores involucrados, está determinado por la eficiencia que se logre alcanzar en el uso de los recursos disponibles, la cual tiende a decrecer cuando se incrementa el número de nodos de procesamiento, debido al normal incremento del costo computacional asociado a las tareas de comunicación entre los mismos. Sin embargo, se pudo observar claramente en las pruebas realizadas que este decrecimiento se hace mucho menos marcado a medida que la cantidad de datos de entrada aumenta. A partir de esta observación, se pudiera afirmar que mientras mayor

sea el volumen de datos analizados, mayor debiera ser la posibilidad de aumentar el rendimiento de la aplicación a un costo razonable, permitiendo que los tiempos de ejecución se mantuvieran en un rango aceptable.

CONCLUSIONES

En la realización de este trabajo se obtuvieron los siguientes resultados:

- Se identificaron las cuatro fases de los algoritmos básicos del análisis filogenético estudiados en las que era factible la aplicación de técnicas de paralelización. Las fases escogidas fueron: El cálculo de las distancias evolutivas entre los distintos pares de secuencias, la estimación de los parámetros del modelo y la estimación de las tasas de mutación por sitio, tanto por el método de máxima verosimilitud como por el método bayesiano empírico.
- Se desarrolló una aplicación en C++ que permite ejecutar en paralelo dichos algoritmos, la cual funciona tanto en una máquina con varios procesadores como en un clúster de computadoras.
- En las pruebas realizadas se pudo comprobar la notable disminución alcanzada en los tiempos de ejecución de los algoritmos en la aplicación desarrollada con respecto a las versiones secuenciales, superando en sentido general, el porciento de reducción esperado en la hipótesis.

RECOMENDACIONES

- Incluir en la aplicación desarrollada otros algoritmos actualmente utilizados en la estimación de las tasas de mutación por sitio.
- Lograr la combinación del paradigma de paso de mensajes con el de espacio de direcciones compartido.
- Dado que, por la limitación del tiempo disponible para la realización de este trabajo, solo se incluyeron dos modelos evolutivos, se deben incluir en la aplicación desarrollada los modelos restantes mencionados en el Anexo 1.

REFERENCIAS BIBLIOGRÁFICAS

Baldi, P. and S. Brunak. (2001). Bioinformatics: The machine learning approach, Massachusetts Institute of Technology: MIT Press.

Brent, R. P. (1973). Algorihms for Minimizations without Derivatives. New Jersey, Prentice-Hall.

Cai, J., D. Smith, et al. (2005). "MBEToolbox: a MATLAB toolbox for sequence data analysis in molecular biology and evolution." BMC Bioinformatics **6**(1): 64.

Cai, J., D. Smith, et al. (2006). "MBEToolbox 2.0: An enhanced version of a MATLAB toolbox for Molecular Biology and Evolution." Evolutionary Bioinformatics Online **2**: 189-192.

Cliften, P. (2004). "The post-genomic era for a select few." Genome Biology.

Consortium, T. C. S. a. A. (2005). "Initial sequence of the chimpanzee genome and comparison with the human
genome." Nature **437**: 69-97.

Desper, R. and O. Gascuel (2002). "Fast and Accurate Phylogeny Reconstruction Algorithms Based on the Minimum-Evolution Principle." Journal of Computational Biology **9**(5): 687-705.

Desper, R. and O. Gascuel (2004). "Theoretical Foundation of the Balanced Minimum Evolution Method of Phylogenetic Inference and Its Relationship to Weighted Least-Squares Tree Fitting." Molecular Biology and Evolution **21**(3): 587-598.

Felsenstein, J. (1981). "Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach." Journal of Molecular Evolution **17**: 368-376.

Fitch, W. (1971). "Toward defining the course of evolution: minimum change for a specific tree topology." Systematic Zoology **20**(4): 406-416.

Francis, J. G. (1961a). "The QR transformation, part I." Computer Journal **4**: 265-272.

Francis, J. G. (1961b). "The QR transformation, part II." Computer Journal **4**: 332-345.

Grama, A., A. Gupta, et al. (2003). Introduction to Parallel Computing, Second Edition, Addison Wesley.

Hahn, G. J. and S. S. Shapiro (1994). Statistical Models in Engineering, John Wiley & sons.

Hall, T. A. (1999). "BioEdit: a user-friendly biologicla secuence alignment editor and analysis program for windows 95/98/NT." Nucleic Acids Symposium(41): 95-98.

Hasegawa, M., H. Kishino, et al. (1985). "Dating the human–ape splitting by a molecular clock of mitochondrial DNA." Journal of Molecular Evolution **22**: 160-174.

Jukes, T. H. and C. R. Cantor (1969). Evolution of protein molecules in Mammalian protein metabolism. New York, Academic Press.

Jukes, T. H. and C. R. Cantor (1969). Evolution of protein molecules. In Mammalian protein metabolism. New York, Academic Press.

Kimura, M. (1980). "A simple method for estimating evolutionary rate of base substitution through comparative studies of nucleotide sequences." Journal of Molecular Evolution **16**: 111-120.

Kowalik, J. S., Ed. (1999). MPI-The Complete Reference. Volume 1, The MPI Core. Second edition, The MIT Press.

Kumar, S. and J. Dudley (2007). "Bioinformatics software for biologists in the genomics era." Bioinformatics **23**(14): 1713-1717.

Kumar, S. and A. Filipski (2008). Molecular Phylogeny Reconstruction. Encyclopedia of Life Sciences (ELS), John Wiley & Sons.

Levy, M. and S. L. Miller (1998). "The stability of the RNA bases: implications for the origin of life." Proc. Natl. Acad. Sci. USA 95: 7933-7938.

Nielsen, R., Ed. (2005). Statistical Methods in Molecular Evolution. Statistics for Biology and Health. Ithaca, Springer.

Orgel, L. (1987). "Evolution of the genetic apparatus: a review." Cold Spring Harbor Symposia on Quantitate Biology: 9-16.

Orgel, L. (2004). "Prebiotic chemistry and the origin of the RNA world, Crit. Rev." Biochemistry and Molecular Biology **39**: 99-123.

Pennisi, E. (2007). "Genomicists Tackle. The Primate Tree." Science **316**.

Press, W. H., S. A. Teukolsky, et al. (1992). Numerical Recipes in C: The art of scientific computing. Cambridge
New York
Port Chester
Melbourne
Sydney, Cambridge University Press.

Saitou, N. and M. Nei (1987). "The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees." Molecular Biology and Evolution **4**: 406-425.

Sánchez, R. and R. Grau (2009). "An algebraic hypothesis about the primeval genetic code architecture." Mathematical Bioscienses **221**: 60-76.

Tamura, K. (1992). "Estimation of the Number of Nucleotide Substitutions when there are strong Transition-Transversion and G+C-Content biases." Molecular Biology and Evolution **9**: 678-687.

Tamura, K., J. Dudley, et al. (2007). "MEGA4: Molecular Evolutionary Genetics Analysis (MEGA) Software Version 4.0." Molecular Biology and Evolution **24**: 1596-1599.

Tamura, K. and M. Nei (1993). "Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees." Molecular Biology and Evolution **10**: 512-526.

Tavaré, S. (1986). "Some probabilistic and statistical problems on the analysis of DNA sequences." Lect. Math. Life Sci. **17**(57-86).

Yang, Z. (1994). "Estimating the pattern of nucleotide substitution." Journal of Molecular Evolution **39**: 105-111.

Yang, Z. (2006). Computational Molecular Evolution, Oxford University Press.

Yang, Z. and T. Wang (1995). "Mixed Model Analysis of DNA sequence Evolution." Biometrics **51**: 552-561.

Yu, U., S. H. Lee, et al. (2004). "Bioinformatics in the Post-genome Era." Journal of Biochemistry and Molecular Biology **37**(1): 75-82.

Zharkikh, A. (1994). "Estimation of evolutionary distances between nucleotide sequences." Journal of Molecular Evolution **39**: 315-329.

ANEXOS

Anexo 1. Modelos probabilísticos de sustitución de nucleótidos

Las principales diferencias entre los distintos modelos son fácilmente deducibles a partir de las matrices de tasas de sustitución de los mismos. A partir de estas es posible conocer las tasas de sustitución y con ellas deducir las distintas propiedades del proceso que se asumen en el modelo (por ejemplo, saber si la tasa de transición entre purinas es la misma o no que entre pirimidinas). A continuación se presentan las matrices de tasas de sustitución de los modelos markovianos de sustitución nucleotídica más comúnmente usados.

JC69 (Jukes and Cantor 1969)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \begin{array}{cccc} T & C & A & G \end{array} \\ \begin{pmatrix} * & \lambda & \lambda & \lambda \\ \lambda & * & \lambda & \lambda \\ \lambda & \lambda & * & \lambda \\ \lambda & \lambda & \lambda & * \end{pmatrix} \end{array}$$

K80 (Kimura 1980)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \begin{array}{cccc} T & C & A & G \end{array} \\ \begin{pmatrix} * & \alpha & \beta & \beta \\ \alpha & * & \beta & \beta \\ \beta & \beta & * & \alpha \\ \beta & \beta & \alpha & * \end{pmatrix} \end{array}$$

F81 (Felsenstein 1981)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \begin{array}{cccc} T & C & A & G \end{array} \\ \begin{pmatrix} * & \pi_C & \pi_A & \pi_G \\ \pi_T & * & \pi_A & \pi_G \\ \pi_T & \pi_C & * & \pi_G \\ \pi_T & \pi_C & \pi_A & * \end{pmatrix} \end{array}$$

HKY85 (Hasegawa, Kishino et al. 1985)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \begin{array}{cccc} T & C & A & G \end{array} \\ \begin{pmatrix} * & \alpha\pi_C & \beta\pi_A & \beta\pi_G \\ \alpha\pi_T & * & \beta\pi_A & \beta\pi_G \\ \beta\pi_T & \beta\pi_C & * & \alpha\pi_G \\ \beta\pi_T & \beta\pi_C & \alpha\pi_A & * \end{pmatrix} \end{array}$$

F84 (Felsenstein J, DNAML program since 1984)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \begin{array}{cccc} T & C & A & G \end{array} \\ \begin{pmatrix} * & (1+\frac{k}{\pi_Y})\pi_C & \beta\pi_A & \beta\pi_G \\ (1+\frac{k}{\pi_Y})\pi_T & * & \beta\pi_A & \beta\pi_G \\ \beta\pi_T & \beta\pi_C & * & (1+\frac{k}{\pi_R})\pi_G \\ \beta\pi_T & \beta\pi_C & (1+\frac{k}{\pi_R})\pi_A & * \end{pmatrix} \end{array}$$

TN93 (Tamura and Nei 1993)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \\ \begin{pmatrix} * & \alpha_1\pi_C & \beta\pi_A & \beta\pi_G \\ \alpha_1\pi_T & * & \beta\pi_A & \beta\pi_G \\ \beta\pi_T & \beta\pi_C & * & \alpha_2\pi_G \\ \beta\pi_T & \beta\pi_C & \alpha_2\pi_A & * \end{pmatrix} \end{array}$$

GTR (REV) (Tavaré 1986; Yang 1994; Zharkikh 1994)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \\ \begin{pmatrix} * & a\pi_C & b\pi_A & c\pi_G \\ a\pi_T & * & d\pi_A & e\pi_G \\ b\pi_T & d\pi_C & * & f\pi_G \\ c\pi_T & e\pi_C & e\pi_A & * \end{pmatrix} \end{array}$$

UNREST (Yang 1994)

$$\begin{array}{c} \\ T \\ C \\ A \\ G \end{array} \begin{array}{c} \\ \begin{pmatrix} * & qTC & qTA & qTG \\ qCT & * & qCA & qCG \\ qAT & qAC & * & qAG \\ qGT & qGC & qGA & * \end{pmatrix} \end{array}$$

FB (Sánchez and Grau 2009)

$$\begin{array}{c} \\ A \\ C \\ T \\ G \\ D \end{array} \begin{array}{c} \begin{array}{ccccc} A & C & G & T & D \end{array} \\ \begin{pmatrix} * & \beta\pi_C & \alpha_R\pi_G & \beta\pi_T & \gamma\pi_D \\ \beta\pi_A & * & \beta\pi_G & \alpha_Y\pi_T & \gamma\pi_D \\ \alpha_R\pi_A & \beta\pi_C & * & \beta\pi_T & \gamma\pi_D \\ \beta\pi_A & \alpha_Y\pi_C & \beta\pi_G & * & \gamma\pi_D \\ \gamma\pi_A & \gamma\pi_C & \gamma\pi_G & \gamma\pi_T & * \end{pmatrix} \end{array}$$

Anexo 2. Diagrama de clases

Como se dijo en el Capítulo 2 de este trabajo, para el diseño de la aplicación desarrollada se utilizaron elementos del paradigma de programación orientada a objetos combinados con otros del paradigma de programación estructurada, debido a las ventajas que ofrece el lenguaje de programación C++ en el uso de ambos. La Figura A2.1 muestra un diagrama de las clases implementadas en este trabajo y las relaciones existentes entre ellas.

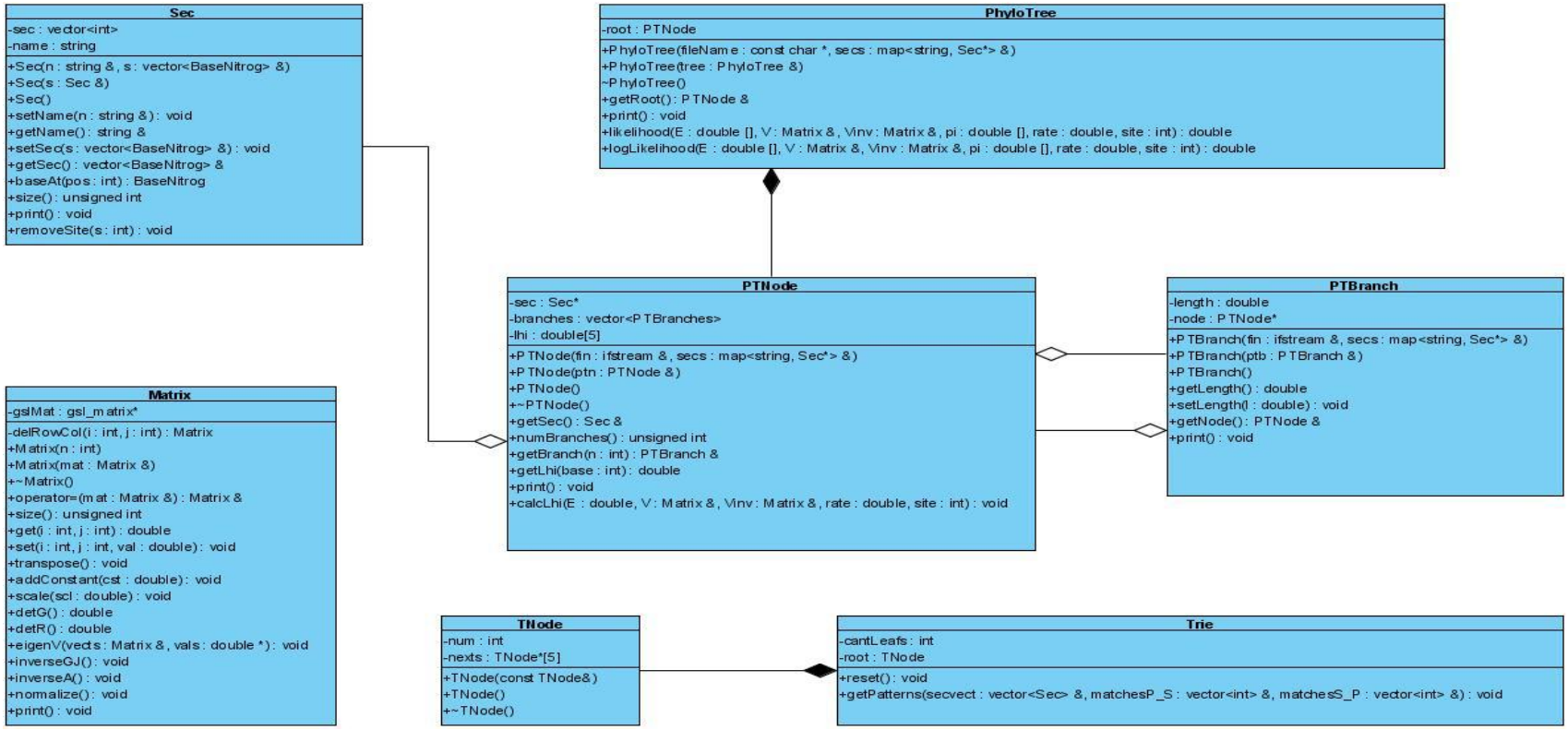

Figura A2.1. Diagrama de clases de la aplicación

Además de las operaciones propias de cada clase, se implementó un número de funciones independientes que trabajan con estos objetos al estilo de la programación estructurada. A continuación se presenta una lista con el prototipo de cada una de ellas:

- *main(argc: int, argv: char* []): int*
- *getSecs(fileName: char*, ret: vector<Sec>&): void*
- *distanceFB(sec1: Sec&, sec2: Sec&): long double*
- *distanceTN93(sec1: Sec&, sec2: Sec&): long double*
- *calcRateMatrixFB(secvect: vector<Sec>&, Q_r: Matrix&, pi: double [], procRank: int, cantProc: int): void*
- *calcRateMatrixTN93(secvect: vector<Sec>&, Q_r: Matrix&, pi: double*, procRank: int, cantProc: int): void*
- *removeGaps(secs: vector<Sec>&): void*
- *condProb(E: double [],V: Matrix&, Vinv: Matrix&, i: int, j: int, length: double, rate: double): double*
- *maximizeLkl(tree: PhyloTree&, eigvals: double [],V: Matrix&, Vinv: Matrix&, freqs: double [], site: int): double*
- *gammafit(data: vector<double>&): double*
- *gammadistrib(gdrate: double*, gdprob: double*, ncate: unsigned int, alpha: double): void*
- *gamma_Pinv(p: double, a: double, b: double): double*
- *siteRateEstim(tree: PhyloTree&, eigvals: double*, V: Matrix&, Vinv: Matrix&, freqs: double*, site: int, gdrate: double*, gdprob: double*, ncate: unsigned int, grate: double): double*
- *parseOptions(argc: int, argv: char* [], procRank: int): void*