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

Extensión al proceso de simulación de poblaciones virales en secuencias genéticas, usando la programación paralela CUDA.

Autor:

Fredy Yasmany Chavez Ramirez

Tutores:

Dr. Daniel Gálvez Lio

MSc. Leonardo Flabio del Toro Mergarejo

Santa Clara, 2015

Declaración Jurada

El que suscribe, Fredy Yasmany Chavez Ramirez, hago constar que el trabajo titulado "Extensión al proceso de simulación de poblaciones virales en secuencias genéticas, usando la programación paralela CUDA.", fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios de la especialidad de Ingeniería Informática, 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(es) Firma del Jefe del Laboratorio

21 de junio de 2015.

A mi familia, mis profesores y mis amigos, por su incondicionalidad e infinito cariño.

No sé si fué obra del azar, el resultado de la conjunción de los astros, o la capacidad del ser humano para hacer que las cosas sucedan, y para encontrar las personas adecuadas en el momento oportuno. No sé si acaso resulta relevante. Lo que es importante, es que hay personas decisivas y trascendentales en el camino de uno, que les debemos parte de lo que somos y de lo que hacemos. Este trabajo no habría sido posible sin una serie de personas que sin saberlo, recorrían conmigo estos senderos, para llegar, por lo menos hasta aquí, a ellos quiero enviar mi agradecimiento:

A mi tutor, el Dr. Daniel Gálvez, por sus enseñanzas, apoyo y confianza en cada momento, y por dedicarme gran parte de su valioso tiempo.

A mi tutor, el MSc. Leonardo del Toro Melgarejo, por el tiempo dedicado durante la realización de este proyecto, incluso desde fuera del país, y por todas sus explicaciones y enseñanzas.

A todos mis profesores por su tiempo, su maravilloso ejemplo, por todo lo que me enseñaron, por ofrecerme cariño y amistad, y haber hecho amenos estos años de estudio.

A mi familia, sin cuyo amor y apoyo no fuese posible.

A mis compañeros por su amistad, y por el apoyo recibido durante todos estos años.

Agradezco a todas las personas que contribuyeron de una manera u otra al desarrollo de este trabajo.

Resumen

En este trabajo se exponen los resultados obtenidos en la simulación de la evolución de poblaciones virales, mediante la aplicación del método estocástico Monte Carlo y diferentes modelos evolutivos de Markov (JC69, K80, F81, y HKY85). Con el propósito de evaluar las tendencias evolutivas de las poblaciones virales, en ausencia o presencia de la presión selectiva del sistema inmune, fueron implementadas dos variantes del algoritmo de generación de secuencias mutantes, utilizando el lenguaje de programación C++, y las técnicas paralelas de CUDA.

Abstract

This paper presents the results obtained from the simulation of the evolution of viral populations, by means of the application of stochastic method Monte Carlo and different evolutionary models of Markov (JC69, K80, F81, and HKY85). In order to assess the evolutionary trends of viral populations, in the absence or presence of selective pressure of the immune system, were implemented two variants of mutant sequences generation algorithm by using the programming language C++, and the parallel techniques of CUDA.

Tabla de contenidos

INTRODUCCIÓN

Hace aproximadamente 3.500 millones de años surgió la vida en la Tierra. Es una historia de cambios y adaptación. Una adaptación que se refleja a muchos niveles, desde la maquinaria celular y los genes que gobiernan el desarrollo, hasta modificaciones morfológicas o de comportamiento. 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. Se han hecho innumerables investigaciones de los procesos evolutivos mediante la aplicación de técnicas de la Biología Molecular combinadas con los métodos matemáticos, en este sentido, cobra vital importancia el problema de simulación de la evolución molecular de poblaciones de virus patógenos, un fenómeno confrontado diariamente por el sistema inmune del hombre, fundamentalmente en el área de la medicina, en la introducción de tratamientos a los pacientes con nuevos antivirales contra virus tales como el de la influenza A/H1N1, el VIH y otros.

Pese a que, hasta el presente la comprensión de los procesos de evolución molecular son de los retos más encumbrados de la biología molecular evolutiva, en sentido general, los virus sufren cambios evolutivos al igual que los seres vivos. Los genomas virales están sujetos a la mutación con la misma frecuencia común a todos los ácidos nucleicos, y cuando las condiciones favorecen a un mutante en particular, éste es seleccionado, dando origen a una nueva cepa que paulatinamente sustituye a la anterior. Debido a que el proceso de evolución molecular, puede ser tratado como un proceso estocástico, la aplicación del método MCMC resulta apropiada para resolver el problema de simulación de la evolución molecular de poblaciones de virus patógenos. La simulación de Monte Carlo, y en particular, el método de Monte Carlo basado en Cadenas de Markov (Markov-Chain Monte Carlo, MCMC), es el fundamento para formar una familia de algoritmos de generación de muestras aleatorias en un tiempo continuo y razonable.

Para este fin hemos propuesto, el uso de un modelo estocástico evolutivo que permita simular matemáticamente la aparición de nuevas variantes mutacionales en las poblaciones virales a partir de la introducción aleatoria de mutaciones en las secuencias tomadas como ancestros; solo que este proceso requiere, una alta capacidad de procesamiento, merced del gran tamaño que casi siempre presentan las secuencia genética virales, Ahora bien, en los últimos años, ha ido

tomando como solución a este problema de alta capacidad de procesamiento, la utilización de un modelo de programación de la GPU en lenguaje de alto nivel, y una arquitectura de programación paralela; estos dos elementos conforman lo que NVIDIA denomina (CUDA).

Antecedentes y actualidad del tema (Contexto).

Los algoritmos de simulación, han sido ampliamente empleados en el análisis evolutivo de las secuencias genómicas de ADN en el campo de la Bioinformática. Son varios los trabajos dedicados al desarrollo de métodos estadísticos en la evolución molecular; incluso, han sido desarrollados varios productos de software destinados enteramente a esta temática, por ejemplo *HyPhy*, cuyo sistema está disponible en el sitio web www.hyphy.org.

La popularidad de las unidades de procesamiento gráfico (GPU) en los computadores modernos brinda un vasto potencial de paralelismo. La tecnología CUDA (*Compute Unified Device Architecture*) de NVIDIA se ha convertido en una poderosa herramienta para el desarrollo de diversas aplicaciones paralelas sobre las GPU, posibilidades que el grupo de Programación e Ingeniería del Software, del Centro de estudios Informáticos, ha empleado para explotar sus beneficios en trabajos previos aplicados a la Bioinformática {Alejo, 2012 #33}, {Alba, 2013 #40}.

El grupo de Bioinformática aplica y desarrolla diferentes modelos para el análisis del proceso evolutivo en secuencias genéticas de virus. Para ello se emplean varios modelos evolutivos {Tamura, 1993 #68}; {Sánchez, 2009 #64} y el método de Simulación de Monte Carlo. El empleo de estos modelos, y la estimación de los parámetros del mismo partiendo de los datos disponibles, hace posible el empleo de la simulación de Monte Carlo en la predicción de posibles variantes mutacionales de genes con el propósito de evaluar las tendencias evolutivas de las poblaciones virales, en ausencia y bajo la presión selectiva del sistema inmune {Sánchez, 2011 #65}.

Se han desarrollado varias tesis, tanto de pre-grado, como de maestría, implementando modelos evolutivos, como el TN93 por Omar Enrique López {López González, 2014 #92}{López González, 2014 #92}, así como la reconstrucción de secuencias ancestrales (ASR), también ha sido implementado en paralelo algunos modelos evolutivos sobre un clúster de computadoras (Jorge E. Moreira en 2011 {Moreira, 2011 #61}), entre otros.

Problema científico

La existencia de otros modelos evolutivos no implementados usando la tecnologia CUDA conlleva a diseñar y desarrollar una implementación con esas caracteristicas de esos modelos de manera que se pueda disponer de los mismos en simulación de la evolución de poblaciones virales, basado en los modelos evolutivos de Markov propuestos en el campo de la Biología molecular evolutiva y que se pueda disponer de ellos para ser incorporados en la herramienta que para tales fines se desarrolla en colaboración con el Laboratorio científico de Bioinformatica.

Acorde a esto hemos planteado llas siguientes **preguntas de investigación**:

1. ¿Cómo es posible extender con otros modelos evolutivos las herramientas desarrolladas con anterioridad?
2. ¿Cómo implementar de manera paralela los segmentos de mayor costo computacional de los nuevos modelos evolutivos?
3. ¿Qué diferencias en la calidad de los resultados se obtendrán entre los diferentes modelos evolutivos?

El **objetivo general** de la investigación consiste en:

Implementar algoritmos paralelos, en diferentes etapas del proceso de simulación de variantes mutacionales para los modelos evolutivos HKY85, JC69, K80, F81 y el algoritmo MCMC para evaluar tendencias evolutivas en poblaciones virales, utilizando la tecnología CUDA.

Este se desglosa en los siguientes **objetivos específicos**:

1. Caracterizar los principales modelos markovianos de la evolución molecular y el algoritmo MCMC.
2. Implementar los modelos evolutivos (K80, JC69, F81, HKY85) usando CUDA.
3. Analizar los resultados obtenidos por los modelos implementados, comparando sus tiempos de ejecución con el de sus versiones secuenciales.
4. Realizar comparaciones entre los resultados obtenidos en la evaluación de tendencias evolutivas de poblaciones virales de los diferentes modelos evolutivos implementados.

Justificación de la investigación:

Esta investigación contribuirá en el grupo de Bioinformática de la UCLV, como parte de un proyecto nacional del Ministerio de Ciencia, Tecnología y Medio Ambiente (CITMA), que lleva a cabo investigaciones sobre la predicción de mutaciones y la resistencia antiviral de virus como el VIH, el virus H1N1 y otros, a esto se añade que en estos momentos resultan más accesibles las tecnologías como CUDA soportadas en tarjetas de video NVIDEA, por lo que el uso de la programación paralela sobre esta tecnología es beneficioso.

Estructura del trabajo

El trabajo se estructura esencialmente en tres capítulos.

- El Capítulo 1 está dedicado a la descripción de los fundamentos biológicos, matemáticos, y computacionales, permitiéndole al lector tener un conocimiento general del tema abordado y una mejor comprensión de los capítulos siguientes.
- El Capítulo 2 se propone a alcanzar los objetivos 2 y 3 del trabajo, describiendo los algoritmos de simulación que permiten generar aleatoriamente las secuencias mutantes de ADN, bajo o sin presión selectiva, también se hace una descripción de nuestra propuesta de solución, y se plantean sus modelados fundamentales.
- El Capítulo 3 se dedica a los últimos objetivos del trabajo, presentando los resultados de las diferentes pruebas y comparaciones.
- Finalmente aparecen las conclusiones, recomendaciones, referencias bibliográficas y anexos.

1 Fundamentos biológicos, matemáticos y computacionales.

En este capítulo se abordarán descriptivamente los sustentos teóricos de este trabajo, puesto que en su desarrollo, se aplican varias herramientas matemáticas y computacionales, dentro de las cuales destaca el uso de la tecnología CUDA, además se trabajará con secuencias de ADN genómico, resultando necesario, para la comprensión de este trabajo, conocer una serie de elementos que se explican a continuación. Los contenidos abordados anteriormente en la tesis precedente se tratarán a modo resumen, centrándonos en los elementos fundamentales, y ahondando en los nuevos.

1.1 Introducción a los elementos biológicos

En este acápite se abordan los elementos básicos de la biología molecular, y las interacciones entre los fundamentales sistemas de la célula.

1.1.1 Las moléculas básicas

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 conocido como Código Genético. La expresión de la información almacenada en el ácido desoxirribonucleico (ADN) se produce a través de la trascripción lineal de la secuencia de nucleótidos en la secuencia de nucleótidos del ácido ribonucleico mensajero (ARNm). La secuencia de nucleótidos del ARNm es seguidamente traducida en una secuencia lineal de aminoácidos y a partir de estos se forman las proteínas, de manera que el flujo de información es ADN →ARNm → Proteína {Minh, 2010 #60}.

1.1.2 ADN

El ADN forma parte de todas las células, desde el punto de vista químico, es un polímero de nucleótidos (polinucleótido), es decir, un compuesto formado por unidades simples denominadas nucleótidos, que se encuentran conectadas entre sí. Cada nucleótido está

formado por la unión covalente de tres componentes: pentosa, ácido fosfórico, y una base nitrogenada, el ácido fosfórico actúa dentro del ADN como enlace entre un nucleótido y el siguiente, mientras que la base nitrogenada (adenina→A, timina→T, citosina→C o guanina→G) distingue a cada nucleótido.

> "La disposición secuencial de estas cuatro bases a lo largo de la cadena es la que codifica la información genética: por ejemplo, una secuencia de ADN puede ser ATGCTAGATCGC". {Alba, 2013 #40}.

1.1.3 ARN (ácido ribonucleico)

Es un ácido nucleico formado por una cadena de ribonucleótidos. El ADN transfiere mediante el ARN información vital (la información genética que se encuentra almacenada en el núcleo de la célula en la secuencia de nucleótidos hacia el citoplasma) durante la síntesis de proteínas (es la molécula que dirige las etapas intermedias de la síntesis proteica que necesita la célula para sus actividades y su desarrollo).

> Existen varios tipos de ARN, cada uno con una función distinta. A los que forman parte de las subunidades de los ribosomas se les denomina ARN ribosomal (rARN); los ARN que tienen la función de transportar los aminoácidos activados, desde el citosol hasta el lugar de síntesis de proteínas en los ribosomas, se les conoce por ARN de transferencia (tARN) y los ARN que son portadores de la información genética y la transportan del genoma (molécula de ADN en el cromosoma) a los ribosomas son llamados ARN mensajero **(mARN)** {Minh, 2010 #60}.

En el proceso de transformación de ADN a ARN tenemos que 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.

1.1.4 Los nucleótidos, los aminoácidos y el código genético

El código genético es el conjunto de reglas usadas para traducir la secuencia de ARNm a secuencia de proteína. Fue esclarecido en el año 1961 por Crick, Brenner y un grupo de colaboradores. La correspondencia entre nucleótidos y aminoácidos se hace mediante codones. Un aminoácido es una molécula orgánica, existen cientos de aminoácidos, pero solo 22 forman parte de las proteínas y tienen codones específicos en el código genético. Un codón es

una tripleta de nucleótidos, que codifica un aminoácido concreto. Los científicos demostraron que hay 61 tripletes -o codones- que codifican aminoácidos, muchos de los cuales son codificados por más de un codón, si un mismo aminoácido es codificado por varios codones, salvo Triptófano y Metionina que están codificados por un único codón, el código genético está degenerado. Los distintos aminoácidos son codificados por un número diferente de codones (algunos por 1, otros por 2, o por 3), e incluso existen tres tripletes que no codifican para ningún aminoácido. Encontramos aminoácidos como Leucina, Serina y Arginina, cada uno de los cuales posee 6 codones que codifican para el mismo, mientras que otros como el Triptófano y la Metionina solo poseen un solo codón. El codón de los aminoácidos Metionina es usualmente utilizado en la mayoría de los seres vivos como codón de iniciación de la cadena polipeptídica. Si combinamos tres bases (tripletes) para formar un aminoácido, obtenemos un total de 64 combinaciones (43 = 64), tomando en cuenta que de ellos solo 61 codifican aminoácidos, restan tres que no son codificantes, estos son utilizados como señales de terminación y son llamados codones de parada (stop codons).

El hecho de que 64 codones diferentes codifiquen solo una veintena de aminoácidos obliga a un cierto grado de degeneración en el código.

> " .. La degeneración del código implica solamente a la tercera posición del codón en la mayoría de los casos (son excepciones la Arginina, la Leucina y la Serina), de esta forma resulta que las dos primeras bases de cada codón son las determinantes principales de su especificidad. La posición tercera, esto es, el nucleótido situado en el extremo 3 del codón, tiene menor importancia y no encaja con tanta precisión". {Minh, 2010 #60}

Los codones que codifican un mismo aminoácido en muchos casos comparten los dos primeros nucleótidos con lo que se minimiza el efecto de las mutaciones. En estos casos una mutación en la tercera posición del codón no cambia el aminoácido codificado denominándose mutación silenciosa. El codón AUG que codifica la metionina es el codón de inicio y hay tres codones que establecen la señal de terminación de la traducción (UAA, UAG, UGA). Las mutaciones que ocurren en estos codones dan lugar a la síntesis de proteínas anómalas.

1.2 Procedimiento evolutivo de las poblaciones virales.

Los virus sufren cambios evolutivos al igual que los seres vivos. Los genomas virales están sujetos a la mutación con la misma frecuencia común a todos los ácidos nucleicos, y cuando las condiciones favorecen a un mutante en particular, éste es seleccionado, dando origen a una nueva cepa que paulatinamente sustituye a la anterior. En algunos virus, la información genética es llevada en las moléculas ARN, pero la característica esencial de la herencia es la misma que en las moléculas de ADN, las cuales son transmitidas de generación a generación.

En el proceso de desarrollo, la información genética contenida en la secuencia de nucleótidos ADN, es transferida primeramente a la secuencia de nucleótidos de ARNm por un proceso simple de transcripción, uno a uno, de los nucleótidos en la molécula ADN. Por el mismo proceso, el tARN y el rARN son producidos. La información transferida al ARNm determina la secuencia de aminoácidos de la proteína que será sintetizada. Los nucleótidos de ARNm son leídos secuencialmente 3 cada vez. Cada triplete o codón es traducido en un aminoácido en la cadena de proteína mediante el código genético.

> "Cualquiera de las mutaciones que son reconocidas como cambios morfológicos o fisiológicos tiene que ser debido a algún cambio de las moléculas ADN. Existen cuatro tipos básicos de cambios en las moléculas ADN: reemplazo de un nucleótido por el otro, eliminación de nucleótidos, adición de nucleótidos, e inversión de nucleótidos. La adición, la eliminación y la inversión pueden ocurrir con uno o más nucleótidos como una unidad. Los remplazos de los nucleótidos pueden ser divididos en dos clases diferentes: la transición y la transversión. Transición es el reemplazo de una purina (adenina A o guanina G) por otra purina, o de una pirimidina (timina T o citosina C) por otra pirimidina. Los otros tipos de sustituciones de nucleótidos donde el cambio ocurre entre purinas y pirimidinas son llamados transversiones". {Nei, 1975 #62}.

1.3 Simulación de la evolución molecular.

La simulación de Monte Carlo ha sido utilizada comúnmente en los estudios filogenéticos para probar diferentes métodos de reconstrucción de árboles filogenéticos y, consecuentemente, sus aplicaciones para probar los modelos evolutivos pueden ser consideradas como una extensión natural de este uso.

En este trabajo, al igual que en el trabajo precedente, la simulación se basará en las cadenas de Markov (CM), todos los elementos de la CM serán definidos a través de una matriz de transición de m dimensiones, los elementos teóricos de las cadenas de Markov, no serán abordados, puesto que se encuentran detallados en la tesis que nos antecede, y el lector puede ahondar en el tema; tanto en esta tesis, como en la bibliografía de cualquier curso básico de Probabilidades y Estadística, de igual forma, no será abordado los elementos matemáticos de la estructura de datos Trie, ni del modelo de regresión lineal como herramienta estadística para el análisis relacional de los datos, fundamental en la simulación de la evolución molecular bajo la presión selectiva del sistema inmune (Elementos que forman parte de nuestra propuesta de solución), pues estos sustentos teóricos fueron igualmente esclarecidos con anterioridad, en su lugar, preferimos profundizar en la técnica de Markov Chain Monte Carlo, y esclarecer la relación entre esta técnica, y las matrices de probabilidad y probabilidad de transición, dependiendo del modelo evolutivo.

1.3.1 Técnica Markov Chain Monte Carlo.

Los métodos de Markov Chain Monte Carlo, o simplemente métodos MCMC, a menudo se aplican para resolver los problemas de integración y optimización en los espacios de grandes dimensiones. Estos métodos pertenecen a una clase de métodos estocásticos y se usan cada vez más en la inferencia bayesiana y en la máquina de aprendizaje. En la evolución molecular, las primeras aplicaciones de MCMC han aparecido por la primera vez en los años 90, cuando varios autores intentaron a desarrollar métodos para calcular las probabilidades posteriores de filogenias sobre las secuencias de ADN alineadas {Nielsen, 2005 #93}.

Uno de los objetivos básicos derivados de la teoría bayesiana general es calcular el valor esperado de una función *f* sobre un espacio paramétrico, con respecto a una distribución de probabilidad de alta dimensión $P(x_1, x_2,...,x_n)$, siendo x_i los valores de los parámetros o variables del modelo. La idea básica de MCMC consiste en, primero, estimar el valor esperado de *f* por:

$$E(f) = \sum_{x_1,x_2,...,x_n} f(x_1, x_2,..., x_n)P(x_1, x_2,..., x_n) \approx \frac{1}{T}\sum_{t=0}^{T} f(x_1^t, x_2^t,..., x_n^t), \qquad (1.1)$$

para cualquier $T > 0$ y $(x_1^t, x_2^t, \ldots, x_n^t)$ es generada como una muestra aleatoria, de acuerdo a su distribución $P(x_1, x_2 \ldots x_n)$. Segundo, la generación de las muestras se lleva a cabo mediante la construcción de una cadena de Markov, teniendo en cuenta P como la distribución estacionaria {Baldi, 2001 #1}.

Considerando un sistema $S = \{s_1, s_2, \ldots, s_n\}$ con n estados. Una cadena de Markov no es más que un conjunto $\{S^0, S^1, \ldots, S^t, \ldots\}$, donde cada elemento S^t representa un estado en un tiempo t determinado, es decir, en cualquier tiempo la cadena esta en un estado particular. Las variables S^t forman una cadena de Markov si y solo si para cualquier t, se tiene:

$$P(S^{t+1} | S^0, S^1, \ldots, S^t) = P(S^{t+1} | S^t) \qquad \textbf{(1.2)}$$

Intuitivamente, esto puede expresarse de otra manera, diciendo que el futuro depende solamente al presente. Una cadena de Markov es totalmente definida por la distribución inicial $P(S^0)$ y las probabilidades de $P^t = P(S^{t+1} | S^t)$. Aquí nos referiremos sólo a las cadenas de Markov estacionarias donde las probabilidades de transición son constantes, es decir independientes del tiempo. Luego, la matriz de transición de la cadena es definida por $T=(t_{ij})$, donde t_{ij} es la probabilidad de transición del estado s_i al estado s_j.

Una manera para construir una cadena apropiada de Markov es usar un método conocido como una instancia de los métodos MCMC, llamado Metrópolis. En este método, la idea de simulación de Monte Carlo es generar un conjunto de muestras dependientes, a partir de una distribución dada (*SP*, las cuales se aceptan o se rechazan en dependencia de como la probabilidad del estado es afectada. Precisamente, el método Metrópolis es definido usando dos familiares de distribuciones Q y R. De las cuales, $Q=(q_{ij})$ es la distribución de selección, donde q_{ij} es la probabilidad de seleccionar el nuevo estado s_j, dado el estado actual s_i. Mientras que $R=(r_{ij})$ es la distribución de aceptación, donde r_{ij} es la probabilidad de aceptar sj como un posible estado siguiente, conociendo el estado actual si. Obviamente, se debe tener $q_{ij} \geq 0$, $r_{ij} \geq 0$ y $\sum_i q_{ij} = 1$. En la mayoría de los casos prácticos, se puede suponer que Q es simétrica, es decir $q_{ij}=q_{ji}$. Sin embargo, esta hipótesis luego fue modificada por Hastings,

de forma tal que *Q* es asimétrica, por tanto el método Metrópolis se convierte en un nuevo método llamado Metropolis-Hasting {Yang, 2006 #70}.

A partir del estado s_i en el tiempo *t* ($S^t = s_i$), el algoritmo de Metrópolis procede de la siguiente forma:

1. Seleccionar aleatoriamente un nuevo estado sj de acuerdo a la distribución qij.
2. Aceptar s_j con la probabilidad r_{ij}. Esto es, $S^{t+1} = s_j$ con la probabilidad r_{ij} y $S^{t+1} = s_i$ con la probabilidad 1- r_{ij}.

En la versión más común del algoritmo de Metrópolis, la probabilidad de la aceptación se define por:

$$r_{ij} = \min\left(1, \frac{P(s_j)}{P(s_i)}\right) \tag{1.3}$$

Cuando se ignora la condición de que *Q* sea simétrica, es decir $q_{ij} \neq q_{ji}$, la probabilidad dada tiene la fórmula:

$$r_{ij} = \min\left\{1, \frac{P(s_j)}{P(s_i)} \cdot \frac{q_{ji}}{q_{ij}}\right\} \tag{1.4}$$

La distribución estacionaria *P* de una cadena de Markov y la probabilidad de transición $T(t_{ij})$ deben satisfacer la siguiente condición de balance:

$$P(S_i)\, t_{ji} = P(S_j)\, t_{ij} \qquad \forall i \neq j \tag{1.5}$$

En este caso, se dice que la distribución P es estable y las cadenas de Markov son reversibles, es decir, las tasas de transición de S_i a S_j es la misma que la de S_j a S_i {Nielsen, 2005 #93}. Esta ecuación implica que una cadena de Markov corre hacia delante en el tiempo es indistinguible probabilísticamente de la misma cadena de Markov cuando esta corre hacia atrás. Comúnmente, la cadena de Markov se supone que es equilibrada, con un vector de

distribución estacionario y es reversible en el tiempo. La reversibilidad es válida para un proceso en equilibrio cuando las ecuaciones de balance (1.5) mantengan válidas.

En la sección siguiente veremos con más detalle de cómo se aplica el método de MCMC en la simulación de la evolución molecular y, en particular, el uso de los modelos markovianos para el proceso evolutivo de las secuencias genómicas.

1.3.2 Simulación del proceso evolutivo mediante el modelo de Markov.

La caracterización de un modelo de Markov en los procesos evolutivos comienza con la definición de un alfabeto de residuos nucleotídicos. En el caso de las secuencias de ADN, el alfabeto consiste en el conjunto de 4 bases $S = \{A, C, G, T\}$.

La modelación del proceso de sustitución de nucleótidos en cualquier sitio particular de una secuencia de ADN, se describe por el método de cadena de Markov con los cuatro nucleótidos representando los estados de la secuencia. El aspecto principal de este método es que el mismo no tiene memoria, es decir: dado el presente, el futuro no depende del pasado. En otras palabras, la probabilidad con la cual una posición de la cadena salta a los otros estados depende solo del estado actual y no importa cómo se ha alcanzado este último. Esto es conocido como "la propiedad markoviana" {Yang, 2006 #70}.

Todos los sitios de una secuencia evolucionan independientemente de acuerdo a la misma cadena de Markov. La simulación de la evolución en los sitios se manifiesta en el evento de la mutación, en la cual el lugar que ocurre una mutación y el nucleótido mutado es determinado aleatoriamente {E. Gultepe, 2005 #55}. Un mecanismo que se usa para realizar este proceso aleatorio es la técnica de las cadenas de Markov descrita anteriormente, de forma que un nuevo nucleótido aleatorio generado s_j es aceptado como una sustitución del nucleótido original s_i ubicado en un sitio dado de la secuencia si se cumple que la probabilidad de (1.4) es mayor que un valor v aleatorio generado con distribución uniforme $U(0,1)$, de lo contrario se rechaza el nuevo nucleótido manteniendo el estado actual de la base que ocupa ese sitio. Este proceso se repite en toda la secuencia hasta que se genere una nueva que puede ser la mutación de la original o una idéntica a ella misma; en este último caso, no ha ocurrido ningún cambio en la evolución de la secuencia.

El modelo de evolución que utiliza un alfabeto extendido para codificar las moléculas ADN ha sido denominado por el Grupo de Bioinformática como "el modelo de las cinco bases" y lo

abreviamos como modelo FB por sus siglas en inglés (Five Bases), o también, como es tradición en la denominación de los modelos evolutivos, como SG09 atendiendo a los apellidos de sus autores (Sánchez y Grau) y el año de su revelación (2009). En la literatura, existen varios modelos de Markov diferentes para la evolución de las secuencias de ADN. Esto es porque los procesos evolutivos cambian entre los genomas y entre las regiones de un genoma. Estos modelos difieren en la forma de la parametrización de la matriz de tasas de sustitución y en la modelación de la variación de las tasas {Sánchez, 2009 #64}. Como casos particulares del modelo FB, los modelos de cuatro bases difieren en que estos últimos no trabajan con el alfabeto extendido, es decir, la base hipotética D en el modelo FB no se presenta.

Cuando los modelos de Markov son continuos en el tiempo, se considera que la matriz $Q=\{ q_{ij} \}$ representa la tasa de sustitución instantánea. Cada elemento q_{ij} de la matriz denota un cambio del nucleótido *i* al *j, con i, j = A, C, G, T, y* las probabilidades de transición entre los estados vienen dadas por la relación $P(t)=exp(Qt)$ que nos da la probabilidad de que cualquier nucleótido dado *i* se convierta al otro diferente *j* en un intervalo de tiempo pequeño $t>0$. Esta matriz se conoce como *matriz de probabilidades de transición* {Yang, 2006 #70}.

En los modelos de sustitución de nucleótidos, la forma estándar para calcular una función algebraica, como la exponencial, de una matriz *Q*, es diagonizarla. Suponiendo que *Q* pudiera escribirse en la forma:

$$Q = UDU^{-1}, \qquad (1.6)$$

donde *U* es una matriz no singular de tamaño 4x4, que viene dada por el modelo de Markov, U^{-1} es su inversa, y *D* es una matriz diagonal creada a partir del vector de frecuencias de distribución de las bases de las secuencias ADN, $D = diag\{432\lambda,\lambda,\lambda,\lambda 1\}$, el cual también se ha definido por el modelo de Markov. La ecuación (1.6) es conocida como la descomposición espectral de la matriz *Q*. De esta ecuación tenemos que:

$$Q^2 = (UDU^{-1}).(UDU^{-1}) = UD^2U^{-1} = U\,diag\{\lambda_1^2, \lambda_2^2, \lambda_3^2, \lambda_4^2\}\,U^1 \qquad (1.7)$$

Similarmente $Q^m = U\,diag\{\lambda_1^m, \lambda_2^m, \lambda_3^m, \lambda_4^m\}\,U^1$ para cualquier entero *m*. En general, cualquier función algebraica *f* aplicada a la matriz *Q* se puede calcular como

$$f(Q) = U\, diag\{f(\lambda_1), f(\lambda_2), f(\lambda_3), f(\lambda_4)\}U^{-1} \tag{1.8}$$

Siempre y cuando *f (Q)* exista. Interesa aquí particularmente la función exponencial aplicada a la matriz *Qt* para construir *P*(*t*), que en esencia, es la solución de una ecuación diferencial matricial (realmente un Problema de Cauchy que se detalla posteriormente), y su existencia se garantiza con ciertas condiciones aquí presentes, gracias a la convergencia de una serie de potencias de la matriz *Qt*. Por tanto, dada la ecuación (1.6) resulta:

$$P(t) = e^{Qt} = U\, diag\{\exp(\lambda_1 t), \exp(\lambda_2 t), \exp(\lambda_3 t), \exp(\lambda_4 t)\}\, U^{-1} \tag{1.9}$$

Los λ_i (i = 1, 2, 3, 4) son valores propios de la matriz *Q*. Las columnas de *U* y las filas de U^{-1} son vectores propios derechos e izquierdos respectivamente de *Q*. El lector puede consultar en un libro texto de álgebra lineal (ej: {Marcelo, 1974 #95}), como se calculan los valores propios y los vectores propios de una matriz. Lo mismo podría decirse con 5 o más bases.

Cuando *t* incrementa de 0 a ∞, los elementos en la diagonal $p_{jj}(t)$ de la matriz *P(t)* anterior decrecen de 1 a π_j, mientras que los elementos no diagonales $p_{ij}(t)$ incrementan de 0 a $j\pi$, con $p_{ij}(\infty)=\pi_j$, siempre conociendo que la distribución (π_A,π_C,π_G,π_T) es la estacionaria.

La mutación puede ocurrir en cada uno de los sitios de la secuencia ADN, teniendo en cuenta que en los mismos, la matriz de probabilidades de transición se calcula una sola vez. En caso de existir un ancestro de la secuencia actualmente procesada, la distribución de probabilidades de un nucleótido en un sitio dado se calcula de manera que la misma se ajusta con la del ancestro en el mismo sitio; de lo contrario, se toma, en la matriz de probabilidades de transición obtenida, la fila correspondiente a esa base y considerándola como su distribución de probabilidades. Un nuevo nucleótido es generado aleatoriamente y se toma como el de sustitución si la distribución de probabilidad correspondiente a él es menor que un valor aleatorio generado. A este mecanismo de generación de bases, que se ha conocido tanto en la simulación como en el campo matemático, se le llama *función de Roulette*.

1.3.3 Modelos Markovianos de la evolución molecular.

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, 2001 #1}. 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 #70}.

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. En la literatura, han sido diversos los modelos de Markov propuestos para la evolución de las secuencias, y se han desarrollado también, varias técnicas para estimar experimentalmente los parámetros de dichos modelos de Markov mediante el análisis de secuencias reales observadas.

Según Yang {Yang, 2006 #70}, de forma general, Los modelos evolutivos describen el modo y la probabilidad de que una secuencia de nucleótidos cambie a otra secuencia de nucleótidos homóloga a lo largo del tiempo. Estos modelos describen para cada uno de los sitios de la

matriz, la probabilidad de que se produzca el cambio de un nucleótido a otro a lo largo de las ramas de un árbol filogenético dado.

Los *modelos de evolución de nucleótidos* se definen matemáticamente mediante dos clases de parámetros que determinan el cambio:

1. Frecuencia de cada nucleótido. Parámetro que mide la frecuencia de cada nucleótido en la matriz de datos y puede tomar los valores siguientes:
   a) En los modelos evolutivos más sencillos se tiene una misma frecuencia para los cuatro nucleótidos sin tener en cuenta la frecuencia de aparición de los mismos en la matriz de datos. **($\pi_A = \pi_C = \pi_G = \pi_T = 0.25$)**.
   b) Los modelos más complejos asumen que las frecuencias asociadas a cada uno de los nucleótidos son diferentes, y son calculadas a partir de los datos ***($\pi_A \neq \pi_C \neq \pi_G \neq \pi_T$)***.
2. Tipos de sustituciones y sus correspondientes tasas de sustitución (***rate parameters***) las cuales se representan a partir de una matriz de tasas de sustitución. Las tasas de sustitución se representan con las tasas relativas de cambio de un nucleótido a otro para una posición de un tiempo t_0 a un tiempo t_1. Así, cada posición de la matriz tendrá una probabilidad asociada de cambio para cada unidad de tiempo (unidad de distancia evolutiva). Los modelos más sencillos asumen una misma tasa relativa para todas las sustituciones posibles, mientras que los más complicados asumen una tasa relativa diferente para cada tipo de sustitución. A partir de estas tasas relativas se calcula la tasa media de sustitución (μ).

La matriz de tasas más general de un modelo de Markov tiene solo el requisito de que los elementos no diagonales son positivos y la suma de cada fila es cero. Se han propuesto varios criterios paramétricos para simplificar la caracterización de la matriz de tasas en los modelos de la evolución de las secuencias de ADN. En esta sección se presenta un resumen de los modelos evolutivos de Markov más conocidos en la literatura y relacionados con este trabajo.

1.3.3.1 El modelo Jukes-Cantor (JC69)

El modelo paramétrico más simple de sustitución de nucleótidos fue propuesto por Jukes y Cantor en 1969. Su modelo asume que todos los nucleótidos tienen la misma tasa de sustitución, esto es, dado que la sustitución se ha producido, cualquier otro nucleótido tiene la

misma probabilidad de ser el nucleótido de sustitución. Por tanto, la matriz de tasas de este modelo puede ser parametrizada de la siguiente forma:

$$Q = \begin{bmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{bmatrix}, \qquad (1.10)$$

donde los nucleótidos son ordenados A, C, G y T. Todos los elementos de la diagonal son iguales a -3α pues que la suma de cada fila de *Q* debe ser igual a cero. Este modelo asume que la distribución estacionaria de las bases es uniforme siendo **($\pi_A = \pi_C = \pi_G = \pi_T = 1/4$)**, y que el modelo es reversible. Como se ha esbozado anteriormente, una cadena de Markov se dice que es reversible en el tiempo si y sólo si se satisface la siguiente ecuación de balance:

$$\pi_i q_{ij} = \pi_j q_{ji} \qquad \text{para todo } i \neq j, \qquad (1.11)$$

donde π_i es la proporción de tiempo que la cadena de Markov se queda en el estado *i*, $\pi_i q_{ij}$ es el flujo del estado *i* al *j*, mientras que $\pi_j q_{ji}$ es el flujo en la dirección opuesta. Esta ecuación significa que el flujo entre cualquier dos estados en la dirección opuesta, es el mismo que la directa {Yang, 2006 #70}. Usando las tasas de la matriz *Q* se puede obtener la probabilidad de sustitución de cada nucleótido en un tiempo **t >0** mediante la creación de una matriz simple de probabilidades de transición, que en este modelo resulta:

$$P(t) = e^{Qt} = \begin{bmatrix} p_0(t) & p_1(t) & p_1(t) & p_1(t) \\ p_1(t) & p_0(t) & p_1(t) & p_1(t) \\ p_1(t) & p_1(t) & p_0(t) & p_1(t) \\ p_1(t) & p_1(t) & p_1(t) & p_0(t) \end{bmatrix} \quad \text{con} \quad \begin{cases} p_0(t) = \frac{1}{4} + \frac{3}{4}e^{-4\lambda t} \\ p_1(t) = \frac{1}{4} - \frac{1}{4}e^{-4\lambda t} \end{cases} \qquad (1.12)$$

1.3.3.2 El modelo Kimura (K80).

Kimura introdujo un modelo un poco más complejo de la evolución de nucleótidos que permite las diferencias entre las tasas de transición y transversión. Como se ha dicho en la

sección 1.3.1 anterior, una transición no es más que una sustitución de una purina por otra purina, o de una pirimidina por otra pirimidina, mientras que una transversión es la sustitución de una purina por una pirimidina o viceversa. Los nucleótidos A y G son purinas, y los nucleótidos C y T son pirimidinas. Transiciones y tranversiones se difieran, ya que las purinas y pirimidinas tienen diferentes estructuras moleculares. La matriz de tasas de sustitución de este modelo es parametrizada como:

$$Q=\begin{bmatrix} -(\alpha+2\beta) & \alpha & \beta & \beta \\ \alpha & -(\alpha+2\beta) & \beta & \beta \\ \beta & \beta & -(\alpha+2\beta) & \alpha \\ \beta & \beta & \alpha & -(\alpha+2\beta) \end{bmatrix} \qquad (1.13)$$

El parámetro α controla la tasa de transiciones, mientras que el parámetro β controla la tasa de transversiones. Al igual que el modelo Jukes-Cantor, el modelo Kimura asume también una distribución de frecuencias uniforme de las bases y es un modelo reversible en el tiempo. Este modelo generaliza al modelo Jukes-Cantor (concuerda con él cuando α=β. La matriz de probabilidades de transición es:

$$P(t)=e^{Qt}=\begin{bmatrix} p_0(t) & p_1(t) & p_2(t) & p_2(t) \\ p_1(t) & p_0(t) & p_2(t) & p_2(t) \\ p_2(t) & p_2(t) & p_0(t) & p_1(t) \\ p_2(t) & p_2(t) & p_1(t) & p_0(t) \end{bmatrix} \qquad (1.14)$$

donde los tres elementos diferentes del modelo son:

$$\begin{cases} p_0(t)=\frac{1}{4}+\frac{1}{4}e^{-4\beta t}+\frac{1}{2}e^{-2(\alpha+\beta)t} \\ p_1(t)=\frac{1}{4}+\frac{1}{4}e^{-4\beta t}-\frac{1}{2}e^{-2(\alpha+\beta)t} \\ p_2(t)=\frac{1}{4}-\frac{1}{4}e^{-4\beta t} \end{cases} \qquad (1.15)$$

1.3.3.3 El modelo Felsenstein (F81)

Felsenstein introdujo una generalización alternativa del modelo Jukes-Cantor. Este modelo especifica que las tasas de sustitución son proporcionales a las distribuciones estacionarias de los nucleótidos de reemplazo. La matriz de tasas se parametriza de la siguiente forma:

$$Q = \begin{bmatrix} -\alpha(\pi_C + \pi_G + \pi_T) & \alpha\pi_C & \alpha\pi_G & \alpha\pi_T \\ \alpha\pi_A & -\alpha(\pi_A + \pi_G + \pi_T) & \alpha\pi_G & \alpha\pi_T \\ \alpha\pi_A & \alpha\pi_C & -\alpha(\pi_A + \pi_C + \pi_T) & \alpha\pi_T \\ \alpha\pi_A & \alpha\pi_C & \alpha\pi_G & -\alpha(\pi_A + \pi_C + \pi_G) \end{bmatrix} \quad (1.16)$$

Los parámetros π_A, π_C, π_G y π_T especifican la distribución estacionaria en la cual se supone que las frecuencias de las bases son diferentes. Dicho modelo asume la misma tasa de sustitución (α) para todos los nucleótidos y generaliza al modelo Jukes-Cantor en el cual se particulariza que **($\pi_A = \pi_C = \pi_G = \pi_T = 1/4$).** Al igual que Jukes-Cantor, este modelo es también reversible en el tiempo. La matriz de probabilidad de transición es:

$$P(t) = \begin{pmatrix} D1 & p1(t) & p2(t) & p3(t) \\ p0(t) & D2 & p2(t) & p3(t) \\ p0(t) & p1(t) & D3 & p3(t) \\ p0(t) & p1(t) & p2(t) & D4 \end{pmatrix} \quad (1.17)$$

donde los elementos del modelo son:

$$\begin{cases} p0(t) = -e^{-T\alpha}\left(-1 + e^{T\alpha}\right)(-1 + \Pi g + \Pi y) \\ p1(t) = -e^{-T\alpha}\left(-1 + e^{T\alpha}\right)(\Pi t - \Pi y) \\ p2(t) = \Pi g - e^{-T\alpha}\Pi g \\ p3(t) = \Pi t - e^{-T\alpha}\Pi t \\ D1 = e^{-T\alpha}\left(\Pi g + \Pi y - e^{T\alpha}(-1 + \Pi g + \Pi y)\right) \\ D2 = e^{-T\alpha}\left(1 + \Pi t - e^{T\alpha}\Pi t + \left(-1 + e^{T\alpha}\right)\Pi y\right) \\ D3 = -e^{-T\alpha}(-1 + \Pi g) + \Pi g \\ D4 = -e^{-T\alpha}(-1 + \Pi t) + \Pi t \end{cases} \quad (1.18)$$

1.3.3.4 El modelo Hasegawa-Kishino-Yano (HKY85).

Hasegawa introdujo un modelo que combina los aspectos de los dos modelos de Jukes-Cantor y Felsenstein. Este modelo es llamado HKY y se puede describir como modelo Felsenstein con un parámetro extra para caracterizar la diferencia entre las transiciones y transversiones. La matriz de tasas de sustitución se puede escribe como:

$$Q = \begin{bmatrix} -\alpha\pi_C - \beta(\pi_G + \pi_T) & \alpha\pi_C & \beta\pi_G & \beta\pi_T \\ \alpha\pi_A & -\alpha\pi_A - \beta(\pi_G + \pi_T) & \beta\pi_G & \beta\pi_T \\ \beta\pi_A & \beta\pi_C & -\alpha\pi_T - \beta(\pi_A + \pi_C) & \alpha\pi_T \\ \beta\pi_A & \beta\pi_C & \alpha\pi_G & -\alpha\pi_G - \beta(\pi_A + \pi_C) \end{bmatrix} \quad (1.19)$$

El modelo HKY generaliza al modelo Kimura cuando $\pi_A = \pi_C = \pi_G = \pi_T = 1/4$ y al modelo Felsenstein cuando $\alpha = \beta$. Como el modelo Felsenstein, el modelo HKY permite una distribución estacionaria no uniforme y es reversible en el tiempo.

$$P(t) = \begin{pmatrix} hx & ix & px & qx \\ jx & kx & px & qx \\ rx & sx & lx & mx \\ rx & sx & nx & ox \end{pmatrix}$$

(1.20)

$$\begin{cases} hx = T + ((T*(A+G))/(T+C))*\exp(-b*t) + (C/(T+C))*\exp(-((T+C)*a+(A+G)*b)*t) \\ ix = C + ((C*(A+G))/(T+C))*\exp(-b*t) - (C/(T+C))*\exp(-((T+C)*a+(A+G)*b)*t) \\ jx = T + ((T*(A+G))/(T+C))*\exp(-b*t) - (T/(T+C))*\exp(-((T+C)*a+(A+G)*b)*t) \\ kx = C + ((C*(A+G))/(T+C))*\exp(-b*t) + (T/(T+C))*\exp(-((T+C)*a+(A+G)*b)*t) \\ lx = A + ((A*(T+C))/(A+G))*\exp(-b*t) + (G/(A+G))*\exp(-((A+G)*a+(T+C)*b)*t) \\ mx = G + ((G*(T+C))/(A+G))*\exp(-b*t) - (G/(A+G))*\exp(-((A+G)*a+(T+C)*b)*t) \\ nx = A + ((A*(T+C))/(A+G))*\exp(-b*t) - (A/(A+G))*\exp(-((A+G)*a+(T+C)*b)*t) \\ ox = G + ((G*(T+C))/(A+G))*\exp(-b*t) + (A/(A+G))*\exp(-((A+G)*a+(T+C)*b)*t) \\ px = A * (1 - (\exp(-b*t))) \\ qx = G * (1 - (\exp(-b*t))) \\ rx = T * (1 - (\exp(-b*t))) \\ sx = C * (1 - (\exp(-b*t))) \end{cases}$$

1.3.3.5 El modelo Tamura-Nei (TN93).

Tamura y Nei propusieron un nuevo modelo, que se considera “ciencia constituida” por su actual uso en investigaciones teóricas y en notables productos de software de Bioinformática, el cual incluye los parámetros que distinguen las tasas de sustitución entre purinas y pirimidinas. Este modelo se conoce como una extensión del modelo HKY85 en que ya se

introducen dos parámetros α_1 *y* α_2 en lugar de un simple parámetro presente en el modelo KHY85 para las tasas transicionales. Formalmente, la matriz de tasas de sustitución *Q* para este modelo, es por tanto, muy similar a la del modelo KHY85, con las diferencias acentuadas. Por tanto como la matriz de probabilidades de transición *P* se especifica:

$$Q = \begin{bmatrix} -(\alpha_2\pi_G + \beta\pi_Y) & \beta\pi_C & \alpha_2\pi_G & \beta\pi_T \\ \beta\pi_A & -(\alpha_1\pi_T + \beta\pi_R) & \beta\pi_G & \alpha_1\pi_T \\ \alpha_2\pi_A & \beta\pi_C & -(\alpha_2\pi_A + \beta\pi_Y) & \beta\pi_T \\ \beta\pi_A & \alpha_1\pi_C & \beta\pi_G & -(\alpha_1\pi_C + \beta\pi_R) \end{bmatrix} \tag{1.21}$$

donde π_A, π_C, π_G y π_T son, como siempre, las frecuencias estacionarias de los nucleótidos A, C, G y T respectivamente, mientras que $\pi_R = \pi_A + \pi_G$ y $\pi_Y = \pi_C + \pi_T$ son las frecuencias de purinas y pirimidinas. Los parámetros α_1, α_2 y β son, respectivamente, la tasa de transiciones entre pirimidinas, entre purinas y la tasa de transversiones. Es en este modelo, en el que queda formalmente más claro, que a partir de esta matriz *Q*, es posible calcular la matriz de probabilidades de transición en el tiempo $P_{ij}(t)$ de la cadena, la cual se deriva de la siguiente ecuación diferencial:

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

A partir de la solución de esta ecuación con condición inicial *P(0)=I*, resulta la matriz de probabilidades de transición:

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

y sus elementos pueden ser calculados por medio de la descomposición espectral que se ha descrito anteriormente en la sección 1.3.2. Para el modelo TN93 se determinan analíticamente los valores propios de ***Q***; tenemos que los mismos son:

$$\lambda_1 = 0; \quad \lambda_2 = -\beta; \quad \lambda_3 = -(\pi_R \alpha_2 + \pi_Y \beta); \quad \lambda_4 = -(\pi_Y \alpha_1 + \pi_R \beta) \tag{1.24}$$

Las matrices de vectores propios serán:

$$U = \begin{bmatrix} 1 & 1/\pi_Y & 0 & \pi_C/\pi_Y \\ 1 & 1/\pi_Y & 0 & -\pi_T/\pi_Y \\ 1 & -1/\pi_R & \pi_G\pi_R & 0 \\ 1 & -1/\pi_R & -\pi_A\pi_R & 0 \end{bmatrix} \tag{1.25}$$

$$U^{-1} = \begin{bmatrix} \pi_T & \pi_C & \pi_A & \pi_G \\ \pi_T\pi_R & \pi_C\pi_R & \pi_A\pi_R & \pi_G\pi_R \\ 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 \end{bmatrix} \tag{1.26}$$

Sustituyendo $\boldsymbol{U}$ y $\boldsymbol{U^{-1}}$ en la ecuación (1.9) obtenemos a $\boldsymbol{P(t)}$ como:

$$P(t) = \begin{bmatrix} \pi_T + \frac{\pi_T\pi_R}{\pi_Y}e_2 + \frac{\pi_C}{\pi_Y}e_4 & \pi_C + \frac{\pi_C\pi_R}{\pi_Y}e_2 - \frac{\pi_C}{\pi_Y}e_4 & \pi_A(1-e_2) & \pi_G(1-e_2) \\ \pi_T + \frac{\pi_T\pi_R}{\pi_Y}e_2 - \frac{\pi_T}{\pi_Y}e_4 & \pi_C + \frac{\pi_C\pi_R}{\pi_Y}e_2 + \frac{\pi_T}{\pi_Y}e_4 & \pi_A(1-e_2) & \pi_G(1-e_2) \\ \pi_T(1-e_2) & \pi_C(1-e_2) & \pi_A + \frac{\pi_A\pi_Y}{\pi_R}e_2 + \frac{\pi_C}{\pi_R}e_3 & \pi_G + \frac{\pi_C\pi_R}{\pi_Y}e_2 - \frac{\pi_G}{\pi_R}e_3 \\ \pi_T(1-e_2) & \pi_C(1-e_2) & \pi_A + \frac{\pi_A\pi_Y}{\pi_R}e_2 - \frac{\pi_A}{\pi_R}e_3 & \pi_G + \frac{\pi_G\pi_R}{\pi_Y}e_2 + \frac{\pi_A}{\pi_R}e_3 \end{bmatrix} \tag{1.27}$$

donde $e_i = exp(\lambda_i t) = e^{(\lambda i t)}$. Cuando t incrementa de 0 a **1** los elementos $p_{ii}(t)$ de la diagonal principal de la matriz anterior decrementan de 1 a π_i, mientras que los elementos $p_{ij}(t)$ incrementan de 0 a π_{ij} ; por tanto $p_{ij}(\infty) = \pi_j$, además $(\pi_A, \pi_C, \pi_G, \pi_T)$ es la distribución estacionaria de la cadena que se construye a partir de P.

1.3.3.6 El modelo FB o Sánchez - Grau (SG09).

Este modelo es una extensión del modelo TN93 propuesta por Robersy Sánchez y Ricardo Grau, que da lugar al "modelo de cinco bases" {Sánchez, 2009 #64}. La adición de una nueva base (D) al conjunto de cuatro ya existentes (A, C, G y T) produce varias modificaciones al modelo TN93 antes expuesto. 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 modelo 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 $\alpha 1$ y $\alpha 2$ y se permite que las cinco bases posean distintas frecuencias. Dos nuevos parámetros γ y ND 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, 2009 #64}.

La matriz de las tasas de sustitución para este modelo es 5 x 5:

$$Q = \begin{bmatrix} * & \beta\pi_C & \alpha_R\pi_G & \beta\pi_T & \gamma\pi_D \\ \beta\pi_A & * & \beta\pi_G & \alpha_Y\pi_T & \gamma\pi_D \\ \alpha_R\pi_A & \beta\pi_C & * & \beta\pi_T & \gamma\pi_D \\ \beta\pi_A & \alpha_Y\pi_C & \beta\pi_G & * & \gamma\pi_D \\ \gamma\pi_A & \gamma\pi_C & \gamma\pi_G & \gamma\pi_T & * \end{bmatrix} \quad (1.28)$$

El símbolo '*' es usado para indicar que los elementos en la diagonal principal de la matriz de tasas de sustitución son definidos, de forma tal que la suma de cada fila es igual a 0. La solución formal, para el cálculo de la matriz P de probabilidad de transición (1.22) y (1.23), no

resultan útiles desde un punto de vista práctico. Para un propósito de aplicación, necesitamos encontrar funciones explícitas, para las probabilidades individuales de transición pij(t) que satisfagan la ecuación diferencial (1.22).

Si los valores propios p1; p2; p3; p4 y p5 de la matriz Qr que satisfacen (1.23) son distintos, entonces la matriz de los correspondientes vectores propios:

$$V = (\, v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \,)|$$

Tiene sus columnas linealmente independientes, es no singular y con inversa V-1. Bajo estas condiciones la matriz exponencial es diagonalizable, y la matriz de probabilidad de transición (1.23) puede ser expresada:

$$P(t) = VE(t)V^{-1}, \tag{1.29}$$

Entonces, $E(t) = \mathrm{diag}[e^{\rho_1 t}, e^{\rho_2 t}, e^{\rho_3 t}, e^{\rho_4 t}, e^{\rho_5 t}]$ es una matriz diagonal. Expandiendo (1.29) tenemos una solución explicita para las probabilidades de transición individuales pij(t):

$$P_{ij}(t) = \sum_{k=1}^{5} V_{ik} V_{kj}^{-1} \exp(\rho_k t), \tag{1.30}$$

Donde Vik es el elemento ik del eigenvector Vi, Vkj-1 = Wjk/det(V) son los elementos de la matriz inversa V-1, Wjk es el cofactor del elemento jk de V y det(V) es el determinante de V. Los valores propios de (1.28) son:

$$\begin{aligned}
&\rho_1 = 0, \quad \rho_2 = -\gamma, \quad \rho_3 = -\gamma + (\gamma - \beta)(1 - \pi_D), \\
&\rho_4 = -\gamma + (\gamma - \beta)\pi_Y + \pi_R(\gamma - \alpha_R), \\
&\rho_5 = -\gamma + (\gamma - \beta)\pi_R + \pi_Y(\gamma - \alpha_Y).
\end{aligned} \tag{1.31}$$

los correspondientes vectores propios:

$$v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad v_2 = \begin{pmatrix} \frac{\pi_D}{1-\pi_D} \\ \frac{\pi_D}{1-\pi_D} \\ \frac{\pi_D}{1-\pi_D} \\ \frac{\pi_D}{1-\pi_D} \\ 1 \end{pmatrix}, \quad v_3 = \begin{pmatrix} -\frac{\pi_Y}{\pi_R} \\ 1 \\ -\frac{\pi_Y}{\pi_R} \\ 1 \\ 0 \end{pmatrix}, \quad v_4 = \begin{pmatrix} -\frac{\pi_G}{\pi_A} \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_5 = \begin{pmatrix} 0 \\ -\frac{\pi_T}{\pi_C} \\ 0 \\ 1 \\ 0 \end{pmatrix}. \tag{1.32}$$

Los vectores columna de la matriz $V^{-1} = W_{jk}/\det(V) = (v_1^{-1}1 \;\; v2^{-1} \;\; v_3^{-1} \;\; v_4^{-1} \;\; v_5^{-1})$ son:

$$v_1^{-1} = \begin{pmatrix} \pi_A \\ \pi_C \\ \pi_G \\ \pi_T \\ \pi_D \end{pmatrix}, \quad v_2^{-1} = \begin{pmatrix} -\pi_A \\ -\pi_C \\ -\pi_G \\ -\pi_T \\ 1-\pi_D \end{pmatrix}, \quad v_3^{-1} = \begin{pmatrix} -\frac{\pi_A}{1-\pi_D} \\ \frac{\pi_C \pi_R}{\pi_Y(1-\pi_D)} \\ -\frac{\pi_G}{1-\pi_D} \\ \frac{\pi_T \pi_R}{\pi_Y(1-\pi_D)} \\ 0 \end{pmatrix},$$

$$v_4^{-1} = \begin{pmatrix} -\frac{\pi_A}{\pi_R} \\ 0 \\ 1-\frac{\pi_G}{\pi_R} \\ 0 \\ 0 \end{pmatrix}, \quad v_5^{-1} = \begin{pmatrix} 0 \\ -\frac{\pi_C}{\pi_Y} \\ 0 \\ 1-\frac{\pi_T}{\pi_Y} \\ 0 \end{pmatrix}, \tag{1.33}$$

Resolviendo (1.29) obtenemos nuestra matriz de probabilidad de transición P(t) que se muestra en el Anexo 1.

1.3.3.7 El modelo General Reversible en el Tiempo (GTR).

El modelo GTR sólo asume que el proceso de sustitución de nucleótidos es reversible. Los modelos que se han discutido anteriormente incluyendo JC69 (Jukes-Cantor 1969), K80 (Kimura 1980), F81 (Felsenstein 1981) , HKY85 (Hasegawa-Kishino-Yano 1985) y TN93 (Tamura y Nei 1993) todos son generalizadas con el modelo GTR, y también son reversibles en el tiempo. Incluso SG09 sigue esta filosofía de reversibilidad con la particularidad de que trata con 5 bases en lugar de 4 como todos los mencionados.

Otra condición equivalente para la reversibilidad es que la matriz de tasas de sustitución puede ser escrita como el producto de una matriz simétrica multiplicada por una matriz diagonal. Luego, los elementos en la diagonal especifican las frecuencias de equilibro. De hecho, la matriz de tasas de sustitución para el modelo general reversible en el tiempo es:

$$Q = \begin{bmatrix} * & a\pi_C & b\pi_G & c\pi_T \\ a\pi_A & * & d\pi_G & e\pi_T \\ b\pi_A & d\pi_C & * & f\pi_T \\ c\pi_A & e\pi_C & f\pi_G & * \end{bmatrix} =$$

$$= \begin{bmatrix} * & a & b & c \\ a & * & d & e \\ b & d & * & f \\ c & e & f & * \end{bmatrix} \begin{bmatrix} \pi_A & 0 & 0 & 0 \\ 0 & \pi_C & 0 & 0 \\ 0 & 0 & \pi_G & 0 \\ 0 & 0 & 0 & \pi_T \end{bmatrix} \tag{1.34}$$

Los elementos diagonales de *Q* son determinados por el requisito de que cada fila de *Q* tiene suma 0. Esta matriz incluye 10 parámetros: las tasas *a, b, c, d, e, f* y 4 parámetros de frecuencias π_A, π_C, π_G y π_T.

Si se especifican todos los parámetros de la matriz de tasas de sustitución *Q*, se deriva el modelo GTR. Sin embargo, es posible reducir el número de parámetros cuando los mismos son desconocidos y, se plantea la necesidad de estimarlos a partir de los datos. Esto se puede obtener más fácilmente introduciendo las restricciones que reflejan algunas simetrías aproximadas en el proceso de sustitución. Las restricciones consisten en la definición de los tipos de cambios de nucleótidos, los cuales se clasifican en dos grupos principales: las transiciones (Ts) y las tranversiones (Tv). Además, es posible distinguir entre sustituciones de purinas y pirimidinas. Una vez que se asumen estas restricciones, sólo quedan dos parámetros independientes: la razón *k* de tasas de Ts y Tv, y la razón γ de dos tipos de tasas de transición. Esto fue lo que definió el modelo TN93. En el caso que γ=1, o sea, las transiciones de purinas y pirimidinas tienen la misma tasa, este modelo se reduce al HKY85. Si las frecuencias de las

bases son uniformes, es decir π_i=1/4, entonces el modelo HKY85 se convierte al K80. Para k=1, el HKY85 se reduce al F81, y el K80 deriva al JC69. La jerarquía de los modelos de sustitución discutidos anteriormente se muestra en la *Figura (1.35)* siguiente.

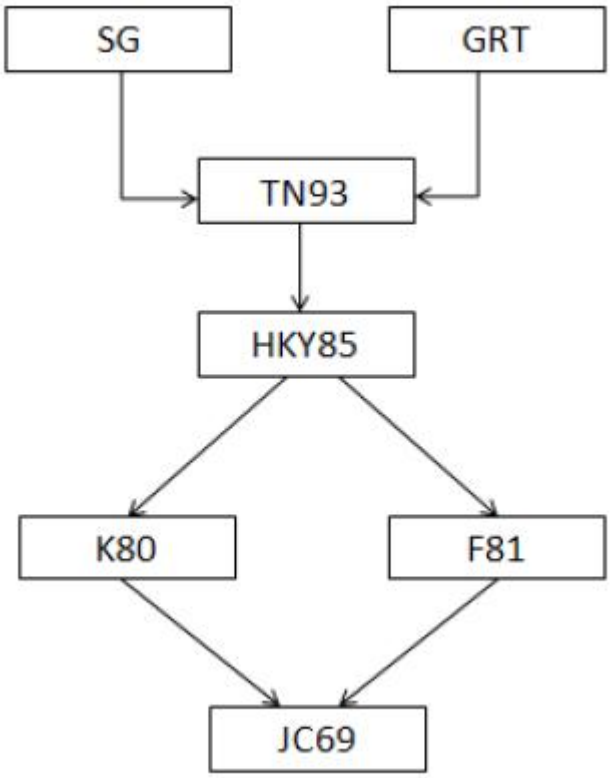


Figura (1.35). *Jerarquía de los modelos de sustitución de nucleótidos.*

1.4 Elementos Computacionales.

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 describe el modelo de programación en CUDA; Por otra parte los elementos teóricos de las bibliotecas utilizadas: (GSL y CURAND) y su utilización en los métodos numéricos, no son abordados, pues el lector puede esclarecer estos contenidos en la tesis precedente.

1.4.1 El lenguaje de programación C++

C++ es un lenguaje orientado a objetos derivado del C que fue diseñado a mediados de los años 80 por Bjarne Stroustrup. La intención de su creación fue extender el exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Posteriormente se añadieron facilidades de programación genérica, la cual se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos, POO). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma y no es un lenguaje orientado a objetos puro. En el diseño de la Librería Estándar C++, se ha usado ampliamente la dualidad de mezcla de un lenguaje tradicional con elementos de POO, lo que ha permitido un modelo muy avanzado de programación extraordinariamente flexible (programación genérica). Desde luego, **C++** es un lenguaje de programación extremadamente largo y complejo, sin embargo, ha experimentado un extraordinario éxito desde su creación. De hecho, muchos sistemas operativos, compiladores e intérpretes han sido escritos en **C++**. Una de las razones de su éxito es ser un lenguaje de propósito general que se adapta a múltiples situaciones. En nuestro caso, la elección de C/C++, no solo es por las facilidades y versatilidad que ofrece el lenguaje, sino también por su alto rendimiento con respecto a otros, su cercanía al hardware, y por su clara relación con el modelo de programación paralela en CUDA, que utiliza una variación del lenguaje C para codificar los algoritmos en GPU, y que para el uso de otro lenguaje debe ser por medio de wrappers.

Como se ha indicado, C++ fue diseñado para mejorar a un lenguaje puramente estructural C añadiendo el nuevo paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas de la computadora. El término de Programación Orientada a Objetos indica más una forma de diseño y una metodología de desarrollo de software, que un lenguaje de programación. En las secciones del capítulo siguiente veremos cómo se organizan los objetos usados y cómo se relacionan ellos en el desarrollo del modelo de la simulación.

1.4.2 Modelo de programación en CUDA.

El uso de las Unidades de Procesamiento Gráfico se ha popularizado y extendido en los últimos tiempos, en principio por el auge de los juegos en computadoras, en la llamada industria del entretenimiento. La utilización en estos de cada vez más recursos gráficos, requieren de una alta capacidad de procesamiento lo que ha llevado a un acelerado desarrollo de las GPUs. El gigante de las GPUs NVIDIA comprendió las ventajas de brindar facilidades para el uso de sus unidades de procesamiento gráfico en aplicaciones de propósito general, haciendo de estas completamente programables para su uso en aplicaciones científicas. Es así como el desarrollo de sus tarjetas gráficas ha ido más allá y ha facilitado el acceso a los programadores, a los recursos de dichas tarjetas a través de una interfaz.

CUDA son las siglas de ***C****ompute* ***U****nified* ***D****evice* ***A****rchitecture* (Arquitectura Unificada de Dispositivos de Cómputo) que hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo creadas por nVidia que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de NVIDIA.

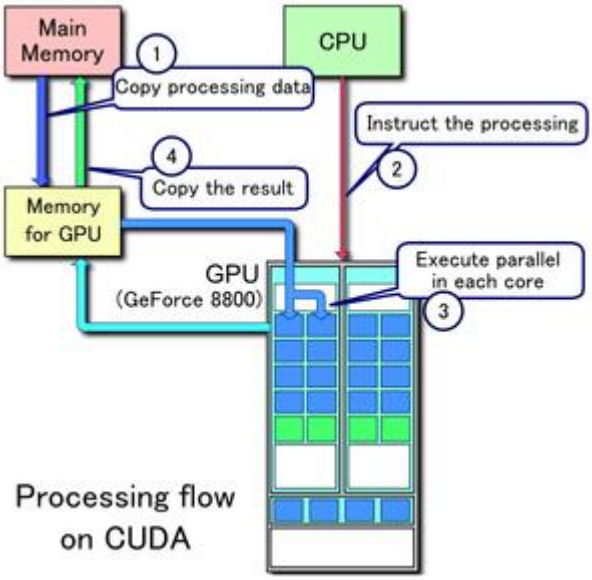
Por medio de wrappers se puede usar Python, Fortran y Java en vez de C/C++ y en el futuro también se añadirá FORTRAN, OpenGL y Direct3D.

Funciona en todas las GPU NVIDIA de la serie G8X en adelante. El diseño del modelo de programación en CUDA está pensado de forma tal que las aplicaciones creadas pueden, de forma transparente, escalar su paralelismo para así incrementar el número de núcleos computacionales {del Toro Melgarejo, 2012 #54}. La programación en CUDA está basada en tres abstracciones básicas; una jerarquía de grupos de hilos, de tipos de memoria, y barreras de sincronización. La estructura que conforma la jerarquía de hilos en este modelo está formada por tres unidades básicas: mallas, bloques e hilos. Múltiples bloques conforman las mallas; un bloque está formado por un grupo de hilos, la unidad constituyente más elemental. Una malla puede ser definida como un grupo de bloques que ejecuta cierta función llamada *kernel*.
CUDA intenta explotar las ventajas de las GPU frente a las unidades centrales de procesamiento (CPU) de propósito general utilizando el paralelismo que ofrecen los múltiples procesadores (conformados por varios núcleos) presentes en las interfaces de procesamiento

de video (tarjetas gráficas), que permiten la ejecución simultánea de un número elevado de hilos (*threads*). Si una aplicación está diseñada para emplear numerosos *threads* que realicen tareas independientes, una GPU podrá ofrecer un gran rendimiento que puede ser aprovechado en aplicaciones con fines docentes o investigaciones científicas para la Bioinformática, la Minería de Datos, Ciencias de los Materiales, la Defensa e Inteligencia, entre otros campos{del Toro Melgarejo, 2012 #54}.

Las funciones que se ejecutarán en la GPU paralelamente por varios hilos son llamadas *kernel*. Mediante una de las extensiones al lenguaje C creadas por NVIDIA, los *kernels*, se definen usando la declaración específica **__global__**. Uno de los rasgos principales que define a un *kernel* es la "configuración de ejecución", toda llamada a una función __global__ debe especificarla. Esta configuración define la dimensión de la malla de bloques y la cantidad de hilos por bloques con que será ejecutado el *kernel* en la GPU. Los hilos pertenecientes a un mismo bloque pueden trabajar en colectivo compartiendo información a través de una cantidad limitada de memoria compartida, accesible solo por los hilos del bloque. El trabajo de estos hilos puede ser sincronizado mediante directivas de bloqueo, lo que posibilita la coordinación entre estos; sin embargo, hilos agrupados en diferentes bloques no pueden comunicarse entre sí.

Este modelo permite a los *kernels* ejecutarse de manera eficiente en varios dispositivos sin recompilación y con diferentes capacidades paralelas en todos los bloques de una malla de forma secuencial si se tiene muy poca capacidad de paralelismo, o en paralelo si se tiene mucha capacidad de paralelismo, aunque por lo general se usa una combinación de ambos (del Toro Melgarejo et al., 2012).



Los hilos que se ejecutan en un dispositivo CUDA tienen acceso a múltiples espacios de memoria: global, local, compartida, textura y registros.

Cada hilo tiene un espacio de memoria local privado; a su vez, cada bloque de hilos posee memoria compartida visible solo a todos los hilos del bloque y con la misma duración de vida que el bloque. Además, todos los hilos ejecutando cierto *kernel* tienen acceso a la misma memoria global. Los espacios de memoria global, constante y de textura, son persistentes a través de los diferentes lanzamientos de *kernels* en la misma aplicación {Alba, 2013 #40}.

El diseño del modelo de programación en CUDA está pensado de forma tal que las aplicaciones creadas pueden, de forma transparente, escalar su paralelismo para así incrementar el número de núcleos computacionales {del Toro Melgarejo, 2012 #54}.

Las ventajas en el desempeño de las GPUs hacen a esta tecnología particularmente interesante para acelerar aplicaciones científicas en diversas ramas, como la generación de imágenes médicas y la electromagnética, entre otras.

1.5 Conclusiones parciales del capítulo

Se han abordado los fundamentos básicos de la evolución molecular y el mecanismo de simulación para la evolución de una población viral, en el cual el método de Markov Chain Monte Carlo juega un papel fundamental, resultando ser el núcleo de nuestra implementación del proceso de simulación de la evolución molecular, también se trataron los elementos teóricos de los nuevos modelos evolutivos que serán implementados. Finalmente, se exponen los elementos fundamentales de las herramientas computacionales utilizadas para la implementación de nuestros algoritmos, el lenguaje de programación C++, y el modelo de programación en CUDA, a través de sus principales características y ventajas.

2 Modelos e implementación del proceso de simulación de las poblaciones virales.

En el capítulo anterior, se introdujeron los elementos teóricos, y matemáticos fundamentales de la técnica Markov Chain Monte Carlo; Técnica de simulación que se ha aplicado extensiva y efectivamente en el área de Bioinformática, especialmente en el campo de la genética evolutiva.

La técnica de Markov puede, y es ampliamente utilizada, para modelar los procesos del tipo *estocástico*, como la divergencia genética a lo largo del tiempo. Estos procesos son conocidos como modelos estocásticos de Markov de primer orden, pues tienen la propiedad de que el estado presente sólo depende del inmediato anterior, por ejemplo, la probabilidad de mutación de un nucleótido A que cambia a otro T no se ve afectada por la historia previa que condujo hasta A. En estos modelos, las tasas de mutación entre diferentes estados discretos de la secuencia de ADN se describen mediante la matriz de transición markoviana $Q^{*} = \{q^{*}_{ij}\}$. La simulación utiliza esta matriz y procesa en un tiempo *T*, empezando a partir de un conjunto dado de secuencias semillas. Las secuencias generadas luego pueden ser pasadas como entradas de los métodos que estiman los parámetros del modelo de Markov para evaluar su exactitud en la reconstrucción de Q^{*}.

En este capítulo, se presenta el procedimiento basado en la técnica de Markov usado para la simulación de poblaciones virales. Se presenta también una descripción por pasos del proceso evolutivo de las poblaciones virales. Además, se exponen los modelos y el diseño general de la aplicación, los cuales constituyen un paso indispensable para construir exitosamente un sistema integrado, capaz de simular eficientemente el proceso evolutivo de las poblaciones virales.

2.1 Procedimientos de simulación basados en la técnica de Markov

Los métodos clásicos de simulación de la evolución de secuencias requieren como entradas un árbol filogenético con la longitud de una rama específica, una secuencia de raíz y un modelo de Markov de la evolución de secuencias. La simulación es realizada por cada rama, empezando por la raíz y terminando en las puntas de las hojas. Para cada rama, se construye

una matriz de mutación *P(t)* (*denominada matriz de tasas de sustitución, cuyo calculo para el modelo evolutivo con que se desee trabajar fue visto con anterioridad*), a partir de del modelo de Markov propuesto, y es escalada de manera tal, que el número medio de sustituciones coincide con la longitud de la rama. La matriz *P(t)* luego se aplica en la parte superior de la rama para crear una secuencia en la parte inferior {Nielsen, 2005 #93}. Una ventaja de nuestro simulador es que no requiere como entrada un árbol filogenético, sino que lo genera como parte de la simulación de acuerdo a un proceso que se junta con el proceso de simulación de mutación de secuencias, reflejando así sus dependencias esperadas en la vida real.

2.1.1 Metropolis-Hasting

Metropolis-Hasting, es conocido como el método más popular de la familia de los métodos Markov Chain Monte Carlo, y comúnmente muy usado para la evolución de secuencias (Andrieu, 2003). Este método consiste en generar un nuevo valor candidato x^* dado el valor actual x, de acuerdo a la probabilidad de distribución propuesta $q(x^* \mid x)$. La cadena de Markov se mueve hacia x^* con la probabilidad de aceptación

$$p(x, x^*) = \min\left\{1, \frac{p(x^*)q(x \mid x^*)}{p(x)q(x^* \mid x)}\right\}$$

En la cual *p(x)* es la distribución invariante definida en *x*, o se queda en este último. El pseudocódigo de este algoritmo se muestra a continuación:

1. Inicializar $x^{(0)}$
2. Para i=0 hasta N-1:
   - Muestrear $u \sim U(0,1)$
   - Muestrear $x^* \sim q(x^* \mid x^{(i)})$
   - Si $u < p(x^{(i)}, x^*) = \min\left\{1, \frac{p(x^*)q(x^{(i)} \mid x^*)}{p(x^{(i)})q(x^* \mid x^{(i)})}\right\}$

     $x^{(i+1)} = x^*$

     si no

     $x^{(i+1)} = x^{(i)}$

Este algoritmo es el más general de los métodos de MCMC. La mayoría de los otros algoritmos prácticos son interpretados como sus casos especiales o extensiones, que difieren en la especificación de las opciones de la probabilidad de distribución propuesta $q(x^* \mid x)$.

En resumen, la matriz P(t) se calcula para cada mutación que se vaya a generar de la secuencia ancestral, acorde al modelo evolutivo que se desea. Luego que se tiene la misma podemos efectuar la técnica de **MCMC** sobre la secuencia ancestral obteniéndose las mutaciones. Según {Minh, 2010 #60} una vez que el valor candidato $\boldsymbol{x}^*$ es aceptado, esto es equivalente a decir que ha ocurrido una mutación del nucleótido en el sitio dado. Este cambio da como consecuencia que tanto la probabilidad de distribución como la probabilidad de mutación del nucleótido se van a determinar para el nuevo estado de acuerdo a la probabilidad previamente definida del nuevo nucleótido. En el caso de que no haya cambio ninguno, o sea, que la cadena de Markov se mantenga en el estado actual con el mismo valor de nucleótido en el sitio dado, entonces no hay necesidad de cambiar las probabilidades de distribución y mutación de este último. Se puede apreciar que cuando se aplica este proceso sobre una secuencia de ADN se trabaja con cada uno de los nucleótidos, de manera tal que la nueva secuencia generada tiene la misma longitud y se comporta como una mutación de su ancestro, formándose a partir de los componentes ya modificados y evidentemente posee propiedades biológicas diferentes {Minh, 2010 #60}.

Algoritmo 2.1 Algoritmo MCMC para la generación de nuevas mutaciones

Entrada: Secuencia ancestral $\boldsymbol{A}$ de longitud $\boldsymbol{N}$.
Salida: Secuencia descendiente $\boldsymbol{D}$.

1: Elegir la secuencia ancestral $\boldsymbol{A}$;
2: **for** $\boldsymbol{i = 0}$ hasta $\boldsymbol{N - 1}$ **do**
3: $\quad \boldsymbol{x^i = A_i}$;
4: $\quad$ Muestrear $\boldsymbol{u \sim U(0,1)}$;
5: $\quad$ Muestrear $\boldsymbol{x^* \sim P(x^* | x^i)}$;
6: $\quad$ **if** $\boldsymbol{u < min(1, \dfrac{f(x^i)P(x^*, x^i)}{f(x^*)P(x^i, x^*)})}$ **then**
7: $\quad\quad \boldsymbol{D_i = x^*}$;
8: $\quad$ **else**
9: $\quad\quad \boldsymbol{D_i = x^i}$;
10: $\quad$ **end if**
11: **end for**
12: Retornar la secuencia descendiente $\boldsymbol{D}$;

2.2 Descripción del proceso evolutivo de las poblaciones virales

Este proceso se simula partiendo de un conjunto de secuencias ADN correspondientes a los genes del virus que se desea simular su evolución molecular, captados en Internet, en nuestro caso, de las bases de datos gratuitas del NCBI. Cada uno de estas secuencias es considerada como un ancestro, que pudiera generar un número de descendientes llamados mutaciones, a través del proceso de simulación que en el trabajo se describe. La forma de seleccionar los primeros ancestros se lleva a cabo aleatoriamente, teniendo en cuenta que la distribución de probabilidades de las secuencias en la población inicial se toma como una medida para realizar la selección, de modo que un ancestro se selecciona si el mismo tiene la distribución de probabilidad mayor que un número aleatorio generado. El número de ancestros seleccionados por generación será fijo y, para cada uno de ellos la cantidad de descendientes a generar también será determinada, de manera tal que los diferentes ancestros tienen la misma cantidad de mutaciones.

La evolución de las poblaciones virales es un proceso bastante complejo que pudiera ser hecha bajo las restricciones evolutivas, para realizar la selección de secuencias que satisfagan cierta condición; proceso que será utilizado, en nuestro caso, para simular la evolución molecular de las poblaciones en secuencias genéticas, con la presión selectiva del sistema inmune, inducida por la vacunación. En las secciones siguientes, veremos resumidamente cómo se realiza la generación de las secuencias mutantes, resumiendo y citando los elementos fundamentales y de mayor importancia tratados en la tesis anterior, tanto en el caso de simulación sin restricciones como con la presencia de la presión selectiva del sistema inmune.

2.2.1 Generación de las mutaciones sin selección.

En este caso, las secuencias mutantes se generan libremente para formar nuevas poblaciones a partir de los ancestros, sin analizar condición evolutiva alguna. Todas las secuencias generadas de esta manera se incluyen completamente en la población del virus, claro, siempre que esta no exista con anterioridad, y se clasifican según sean sus propiedades estructurales. Concretamente, este proceso se realiza mediante los pasos algorítmicos descritos a continuación:

- Paso 1: Especificar la cantidad de descendientes que pudiera tener cada ancestro.

- Paso 2: A partir de la secuencia ancestral seleccionada, generar una secuencia mutante aplicando la técnica de Markov.
- Paso 3: Comprobar que no existan codones de parada en la mutación obtenida, en caso de existir realizar las operaciones necesarias para eliminar los mismos y obtener la secuencia mutante sin codones paradas.
- Paso 4: Comparar la secuencia mutante con el ancestro y con el resto de las otras secuencias; en caso de que la misma sea igual que uno de ellos, se aumentará el contador de este en 1, de lo contrario se agregará la mutación a la población como una nueva clase del virus y se aumentará el número de clases en 1. En los dos casos la cantidad total de secuencias de la población se incrementa en una unidad.
- Paso 5: El ancestro actual se traslada a la secuencia recién creada y el algoritmo vuelve al paso 2 en caso que no se haya acabado la cantidad de descendientes generados del primer ancestro; de lo contrario se mueve al paso siguiente
- Paso 6: Si el número de posibles ancestros no se ha rebasado, seleccionar el nuevo ancestro entre las secuencias de la población inicial disponible, de forma independiente de la selección de los otros ancestros y volvemos al paso 1; de otro modo se mueve al paso siguiente.
- Paso 7: Terminar el algoritmo.

Algoritmo para encontrar los codones de parada:

Algoritmo 2.2 Algoritmo para encontrar los codones de parada

Entrada: Secuencia mutante M de longitud n.
Salida: Secuencia mutante sin codones de parada.

```
1: Calcular el total de codones t = n/3;
2: for i = 1 hasta t do
3:    Calcular el codón C_i de M;
4:    if C_i = codón de parada then
5:       C_i* = Mutar(C_i) {Algoritmo Metroplis-Hastings};
6:       if C_i* = codón de parada then
7:          Modificar(C_i*) {Modifica la última posición del codón por T};
8:          C_i = C_i*;
9:       else
10:         C_i = C_i*;
11:      end if
12:      Ubicar C_i dentro de M;
13:   end if
14: end for
```

Este procedimiento se realiza para cada generación del virus, y se repite tantas veces como sea necesario, acorde al número de generaciones especificado. Una vez que se ha terminado de generar las mutaciones de una generación, se actualizará la distribución de probabilidades de la población actual. De tal manera, los ancestros seleccionados en una generación pueden incluir a las secuencias que fueron creadas en las generaciones anteriores.

Al finalizar este proceso, se obtiene una nueva población más amplia formada por varias generaciones que incluyen todas las secuencias mutantes, sin tener en cuenta ninguna condición evolutiva. En la siguiente sección, se mostrará cómo es el proceso de generación de las mutaciones cuando se aplica la presión selectiva sobre la población.

2.2.2 Generación de las mutaciones bajo la presión selectiva.

De forma general el proceso que a continuación se describe, genera las secuencias de ADN mutantes mediante los mismos pasos descritos en el algoritmo anterior, excepto que entre el paso 2 y el 3 se aplica el mecanismo de selección.

Este proceso de simular la evolución molecular de poblaciones de virus en secuencias genéticas, con selección, resulta mucho más realista que sin presión selectiva, pues en la naturaleza, el proceso evolutivo no escapa a los mecanismos selectivos, y en el caso que nos compete, la selección de nuevas clases o secuencias virales, que son captadas por la presión selectiva que ofrece el sistema inmune, resultado de la vacunación.

Cada nueva secuencia generada a partir de un ancestro dado, se analizará, de forma tal que la misma pudiera satisfacer o no la condición de resistencia contra la vacuna. En el caso de que la mutación sea reconocida por la vacuna, entonces será eliminada definitivamente y, por supuesto, no sigue generando descendientes. De lo contrario, si la nueva secuencia no es reconocida por la vacuna, se agregará a la población, tal y como se ha descrito en el paso 3 anterior. La simulación computacional realiza la selección, mediante el análisis de regresión lineal. Este método estadístico se aplica en este caso de la forma siguiente:

- Transformar la secuencia de ADN generada a la secuencia de aminoácidos correspondiente, usando el código genético. Luego, convirtiéndose esta última en un vector de valores reales mediante las energías de los aminoácidos.
- Sobre este vector de energías, se crea un nuevo vector de medias, teniendo en cuenta el tamaño de la ventana de desplazamiento.

- Realizar el mismo proceso con la secuencia de vacuna para obtener otro vector de medias.
- Aplicar el modelo de regresión lineal simple sobre los vectores obtenidos, dado que la variable dependiente (denotada por *y*) corresponde a la media de la secuencia mutante, y la variable independiente (denotada por *x*) es correspondiente a la media de la secuencia vacuna.

Al encontrar el modelo de regresión adecuado, podemos determinar los residuos del mismo, y luego estudentizarlos para buscar los *outliers.* Estos últimos se usan para confirmar el hecho de que la mutación sea reconocida o no por la vacuna. Si existen por lo menos 4 *outliers,* decimos que la vacuna no reconoce la secuencia mutante, y ésta se agrega a la población; de lo contrario, la misma es eliminada.

- Paso 1. Especificar la cantidad de descendientes que pudiera tener el ancestro.
- Paso 2. Escoger aleatoriamente a partir de la distribución de frecuencia de las secuencias un ancestro, luego generar una mutación del mismo aplicando la estrategia MCMC.
- Paso 3. Comprobar que no existan codones de parada en la mutación obtenida, en caso de existir realizar las operaciones necesarias para eliminar los mismos y obtener la secuencia mutante sin *stop codons*.
- Paso 4. Transformar la secuencia de ADN mutante y la ancestral (o la vacuna si esta existe) en las secuencias de aminoácidos correspondientes, luego convertirlas en los vectores de energías para calcular después los vectores de medias conociendo el tamaño del desplazamiento de la ventana.
- Paso 5. Aplicar la técnica del modelo de regresión lineal para encontrar los residuos. Al estudentizar estos últimos, se obtendrán los residuos estudentizados que serán usados para encontrar los posibles *outliers*. La presencia de cuatro o más *outliers* es el criterio tomado para aceptar en la población la nueva variante mutacional. En el caso de aceptarlo, se continúa con el paso siguiente. De lo contrario se elimina la variante y se vuelve al paso 2.
- Paso 6. Comparar la secuencia mutante generada con el ancestro y con el resto de las otras secuencias; en caso de que la misma sea igual que uno de ellos, se aumentará el

contador de este en 1, de lo contrario se agregará la mutación a la población como una nueva clase del virus y se aumentará el número de clases en 1. En los dos casos la cantidad total de secuencias de la población se incrementa en una unidad.

- Paso 7. El ancestro actual se traslada a la secuencia recién creada y el algoritmo vuelve al paso 2 en caso que no se haya acabado la cantidad de descendientes generados del primer ancestro; de lo contrario se mueve al paso siguiente.
- Paso 8. Si el número de posibles ancestros no se ha rebasado, seleccionar el nuevo ancestro entre las secuencias de la población inicial disponible, de forma independiente de la selección de los otros ancestros y volver al paso 1; de otro modo se mueve al paso siguiente.
- Paso 9. Terminar el algoritmo.

El algoritmo antes planteado, describe la evolución de poblaciones virales con la aplicación de la restricción selectiva, resultando que finalmente contaremos, solo los individuos resistentes, aquellas secuencias virales que pudieran sobrevivir y adaptarse a los cambios frecuentes del ambiente.

Algoritmo 2.3 Clasificar una secuencia a partir de una vacuna.

Entrada: secuencia mutante **M**, secuencia vacuna **V**.
Salida: Acepta o no a **M** como posible mutación.

```
1: R = standard_resid(V,M);
2: V_R = aminoacids_means(V);
3:
4: if (student_resid(V_R, R)) then
5:    Aceptar la secuencia
6: else
7:    No aceptar la secuencia obtenida como una mutación
8: end if
```

En el algoritmo (2.3) tenemos el pseudo-código que describe los pasos mediante los cuales se clasifica una secuencia, en el mismo se utilizan varias funciones auxiliares, una de ellas es **aminoacids_means** la cual recibe una secuencia de ADN y retorna un vector de valores reales, calculados utilizando las energías de los aminoácidos que conformaban la secuencia. La función **standard_resid** cuya tarea es recibir secuencias de ADN y obtener los residuos estandarizados relativos al modelo de regresión lineal, que se obtiene a partir de los vectores de medias de los aminoácidos de cada una de las secuencias, el algoritmo que obtiene estos

residuos es el (2.4), luego que se tienen los mismos, entonces utilizando la función **student_resid** se estudentizan y se chequea si existen o no valores *outliers*, para luego proceder a realizar la clasificación de la secuencia; el algoritmo (2.5) contiene los pasos a seguir durante este último proceso de clasificación.

Algoritmo 2.4 Para hallar los residuos estándares de un modelo de regresión lineal

Entrada: secuencia mutante M, secuencia vacuna V.
Salida: Vector de residuos estandarizados $\boldsymbol{R}$.

```
1: X = aminoacids_means(V);
2: Y = aminoacids_means(M);
3: Declarar e inicializar en 0 a sumX, sumY, sumXX, sumYY, sumXY;
4:
5: for i = 1 hasta n − 1 do
6:    sumX += X_i;
7:    sumY += Y_i ;
8:    sumXX += (X_i)^2;
9:    sumYY += (Y_i)^2;
10:   sumXY += X_i * Y_i;
11: end for
12:
13: yBar = sumY/n;
14: xBar = sumX/n;
15: Syy = sumYY-sumY*sumY/n;
16: Sxy = sumXY-sumX*sumY/n;
17:
18: slope = Sxy/Sxx;
19: intercept = yBar - xBar*slope;
20: r = Sxy*sqrt(Syy*Sxx);
21:
22: for i = 1 hasta n − 1 do
23:    yobs = slope*X_i + intercept;
24:    R_i = Y_i - yobs;
25: end for
26: return R_i;
```

Algoritmo 2.5 Estudentiza un conjunto de residuos estandarizados y clasifica

Entrada: Vector de residuos estandarizados $\boldsymbol{R}$, vector de medias $\boldsymbol{V}$ realtivo a la vacuna.
Salida: **true** o **false**.

1: Declarar matriz $\boldsymbol{M}$ de $\boldsymbol{n}$x**2** dimensiones;
2: **for** $\boldsymbol{i = 1}$ hasta $\boldsymbol{n - 1}$ **do**
3: $\quad \boldsymbol{M_{i,0} = 1}$;
4: $\quad \boldsymbol{M_{i,1} = X_i}$;
5: **end for**
6:
7: Declarar matriz $\boldsymbol{P}$ de **2**x**2** dimensiones;
8: $\boldsymbol{P = (M^T)M}$;
9: $\boldsymbol{P^{-1} = \text{invert}(P)}$;
10:
11: Declarar matriz $\boldsymbol{H}$ de $\boldsymbol{n}$x$\boldsymbol{n}$ dimensiones;
12: $\boldsymbol{H = X(P^{-1})X^T}$;
13:
14: $\boldsymbol{sum = 0}$;
15: **for** $\boldsymbol{i = 1}$ hasta $\boldsymbol{n - 1}$ **do**
16: $\quad \boldsymbol{sum+ = (R_i)^2}$;
17: **end for**
18: $\boldsymbol{var\_est = \sqrt{\frac{sum}{n - 2}}}$;
19:
20: $\boldsymbol{c = 0}$;
21: **for** $\boldsymbol{i = 1}$ hasta $\boldsymbol{n - 1}$ **do**
22: $\quad \boldsymbol{rs = \frac{R_i}{(var\_est)\sqrt{1 - H_{i,i}}}}$;
23:
24: $\quad$ **if** $\boldsymbol{(|rs| > 2)}$ **then**
25: $\quad\quad \boldsymbol{c++}$;
26: $\quad$ **end if**
27: **end for**
28:
29: **if** $\boldsymbol{(c >= 4)}$ **then**
30: $\quad$ **return true**;
31: **else**
32: $\quad$ **return false**;
33: **end if**

El resultado obtenido mediante este proceso difiere del que no tiene en cuenta la restricción selectiva en el hecho de que el tiempo de ejecución crece y el tamaño de la población final es reducido (Minh, 2010). En la próxima sección, se expondrá la implementación de estos algoritmos con más detalle.

2.3 Implementación de los algoritmos de simulación usando el paradigma de programación orientada a objetos.

Esta sección presenta una descripción del diseño general de las aplicaciones resultantes de este trabajo, en las cuales se implemntan los métodos descritos en las secciones anteriores que forman parte del procedimiento de la simulación. Se introducen además, los diagramas de clases que representan las entidades modeladas para representar la solución computacional del problema.

2.3.1 Propuesta de solución.

La simulación de la evolución de las poblaciones virales puede ser hecha bajo restricciones evolutivas para generar solo las secuencias que satisfagan cierta condición, o pudiera ser realizada de manera que las mismas se generan libremente sin limitación alguna. A continuación se muestra el esquema general del método para la simulación de la evolución de poblaciones virales presentado en la tesis anterior {López González, 2014 #92}. Este esquema incluye tanto la simulación con presión selectiva (“Clasificar secuencias”) que sin presión selectiva (“No clasificar secuencias”).

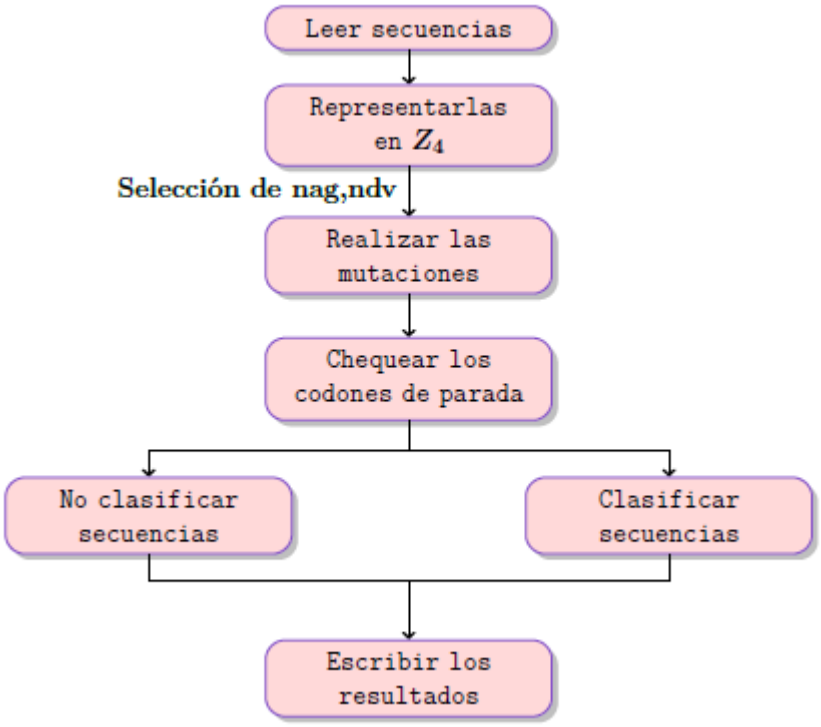


Figura 2.1: Proceso de simulación de poblaciones virales.

La solución propuesta en este trabajo son aplicaciones de consola que reciben como parámetros toda la información de entrada de datos y de salida de la solución, esa información es generalmente nombres de archivos. Se obtienen dos aplicaciones, una que simula la evolución molecular sin presión selectiva del sistema inmune, y otra que lo hace con selección (presión selectiva), tanto en la versión secuencial como en la paralela, para cada uno de los modelos evolutivos que se implementan: JC69, K80, F81, HKY85.

Los requisitos funcionales del sistema están relacionados con que sea capaz de simular la evolución molecular de un conjunto de secuencias de ADN genómico, atendiendo a ciertos parámetros: número de ancestros por generación (nag), número de descendientes por ancestro (ndv), y el número deseado de generaciones, y retorne como resultado un archivo con las secuencias resultantes del proceso evolutivo viral.

Como requisitos no funcionales se especifican los siguientes:

Usabilidad: Facilidad de uso por personas sin experiencia previa.

Rendimiento: Dada la complejidad del proceso de simulación de la evolución molecular de poblaciones virales, y el gran tamaño que casi siempre tienen las bases de casos de secuencias de ADN viral se espera una disminución del tiempo de ejecución al usar programación paralala y CUDA.

Cada aplicación toma como entradas un conjunto de secuencias de ADN, así como un archivo resultante del proceso ASR (Reconstrucción de Secuencias Ancestrales), generado por el software ASR_MPI, a partir de las cuales, se desea aplicar el algoritmo de simulación, teniendo en cuenta los parámetros del modelo seleccionado, y el vector de las tasas de sustitución por sitio. Una vez que se hayan establecido los datos anteriores, es necesario especificar los parámetros de la simulación: número de ancestros por generación, número de descendientes por ancestro, y número de generaciones.

2.3.2 Simulación de las poblaciones en el caso sin restricción.

En el diagrama de la ***Figura 2.2***, se muestra el modelo de simulación en el caso sin restricciones, que muestra las clases utilizadas y las relaciones existentes entre ellas. Observando el esquema, se puede ver que el sistema se constituye por 3 bloques principales:

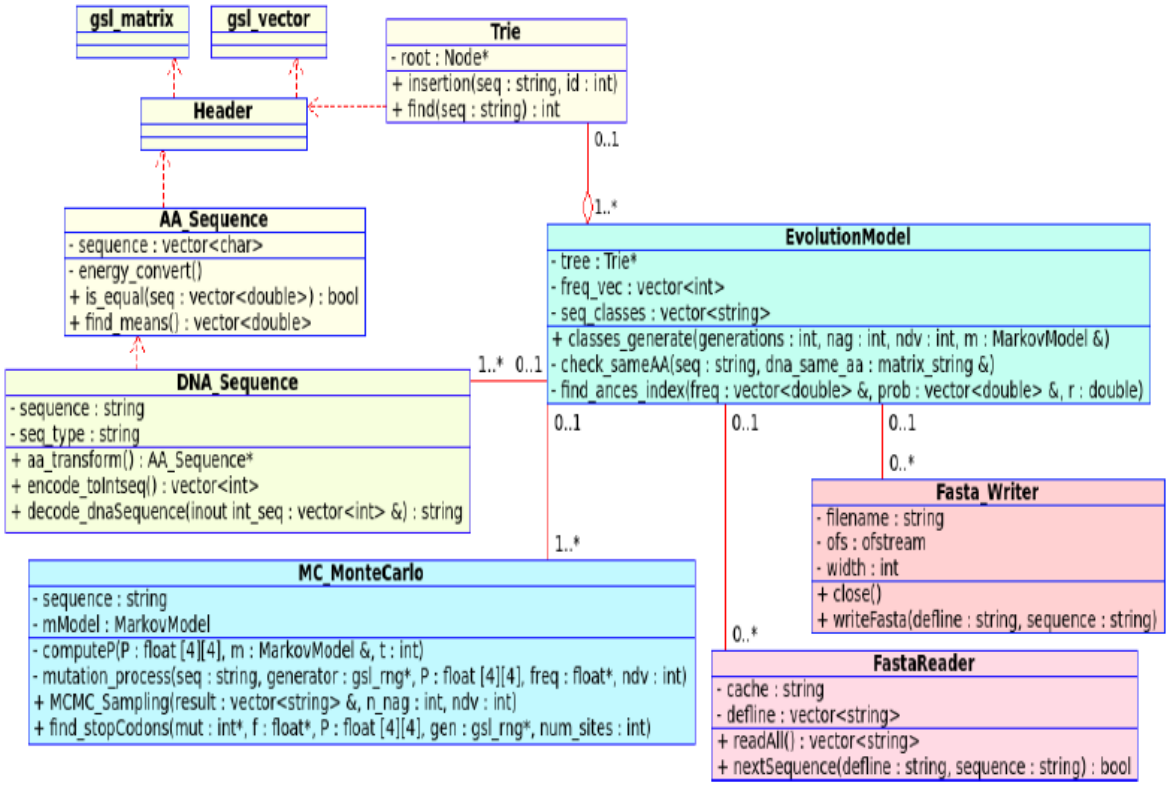


Figura 2.2: Diagrama de clases sin presión selectiva.

El primer grupo está compuesto por las clases consideradas como básicas que se definen en el modelo. Para realizar las tareas simples del procesamiento de las secuencias de ADN contamos con las clases **AA_Sequence** y **DNA_Sequence**. También tenemos aquí la clase **Header**, la cual tiene como objetivo servir de encabezado común para todas las demás entidades del modelo; en la misma se encuentran almacenados los principales archivos cabeceras necesarios en la aplicación, dentro de estos archivos tenemos la inclusión de los concernientes a la *Gnu Scientific Library* (GSL), que son utilizados en la aplicación para complementar los algoritmos implementados en otras clases facilitando el trabajo algebraico necesario. Para mayor información sobre esta biblioteca se puede consultar la dirección www.gnu.org/software/gsl/manual.

El segundo grupo lo conforman aquellas clases que llevan a cabo la tarea principal de la aplicación que es obtener las mutaciones, el mismo está compuesto por: **MC_MonteCarlo** y **EvolutionModel**. La función de **EvolutionModel** es recibir del programa principal (*main*) los parámetros de la aplicación, entre ellos: el número de generaciones, total de ancestros por cada

generación, la base de casos, entre otros. La misma procesa estos parámetros, obtiene aleatoriamente un ancestro y le envía la secuencia ancestral a **MC_MonteCarlo** para que obtenga las mutaciones; luego que ya se tienen las mutaciones esta se encarga de realizar todo el procedimiento de almacenamiento de los resultados en los archivos de salida y también actualiza el conjunto de secuencias disponibles, además del vector de frecuencias de aparición; en síntesis, su tarea es preparar lo necesario para la mutación y procesar el resultado obtenido al mutar.

Una de las funcionalidades más importantes de esta clase es clasificar las mutaciones de acuerdo a las proteínas que estas representan, de forma tal que si existen 2 secuencias distintas que sintetizan una misma proteína entonces dichas mutaciones forman parte del conjunto representado por la proteína codificada. Por ejemplo, supongamos que nuestras secuencias están conformadas por un solo aminoácido; es decir, tienen solamente 3 bases, entonces podemos tener como mutaciones a UUA y UUG representando ambas al mismo aminoácido y por tanto a la misma proteína, en este caso entonces utilizando un vector se almacena una de las 2 mutaciones como representante de la proteína, la otra se inserta en la estructura a continuación de la anterior, y así ocurrirá con las demás mutaciones obtenidas que se ajusten a una misma codificación. Por otro lado, la clase **MC_MonteCarlo** es la encargada de mutar aplicando el algoritmo (2.1) una secuencia ancestral, tantas veces como el valor especificado por número de descendientes por virus; esta se encarga de calcular en cada mutación la matriz de probabilidades de transición. También el procedimiento de encontrar los codones de parada de una mutación descrito en el algoritmo (2.2) se lleva a cabo en **MC_MonteCarlo**. La tarea de la misma termina cuando ya se han procesado todos los descendientes del ancestro, retornando los mismos a **EvolutionModel** para ser procesados.

El tercer bloque del sistema es responsable de realizar los procesos de leer y escribir datos desde y hacia los ficheros. Para ello, se han usado las clases *FastaReader* y *FastaWriter* que heredan de las clases superiores *FileReader* y *FileWriter*, respectivamente. La primera es usada para la lectura, mientras que la segunda es para la escritura. Ambas clases trabajan con los ficheros del formato Fasta, los cuales se caracterizan por la estructura de almacenamiento de las secuencias de ADN, especificando un encabezamiento identificador de la secuencia, y luego la secuencia alineada. La herencia de las clases superiores es útil en el caso de realizar

tareas simples con los ficheros como, por ejemplo, el chequeo de espacios en blanco, recuperación de las líneas, cerrar ficheros, etc. Las clases *FastaReader* y *FastaWriter* forman parte de la librería BOOM (Bioinformatics Object-Oriented Modules).

2.3.3 Simulación de las poblaciones bajo presión selectiva.

El procedimiento de selección de las secuencias mutantes, que satisfagan cierta condición evolutiva, se introduce al modelo de simulación cuando es necesario chequear la resistencia de las secuencias genómicas capaces de evadir la acción del sistema inmune, inducida por la vacunación. Para definir la secuencia de ADN viral como una vacuna, se escoge, entre otras variantes mutacionales, aquella que al ser comparada con el ancestro en cuanto a las características físico-químicas, mediante el análisis de regresión lineal, no presenten más de cuatro *outliers*. La misma es usada para seleccionar, entre las secuencias descendientes, aquellas que son probabilísticamente *resistentes a la vacuna*, de modo que las mismas se añaden como nuevos individuos a la población del virus.

El esquema de la ***Figura 2.3*** muestra el diseño de clases para el modelo de simulación bajo la presión selectiva de las secuencias mutantes. Al igual que en el caso presentado anteriormente, este modelo se construye mediante el diseño estructural de algoritmos dividido en tres bloques de diferentes funcionamientos, más un bloque adicional que contiene las clases relativas al análisis de regresión lineal. Este nuevo bloque se compone por las clases siguientes **Regression**, **LinRegressor** y **LinearFunc**. De estas nuevas 3 clases las 2 últimas se reimplementaron tomando como base 2 clases de la biblioteca BOOM mencionada anteriormente. La clase **Regression** tiene como funcionalidades principales encontrar el modelo de regresión lineal a partir de 2 conjuntos de valores; es decir, construir la recta de regresión y obtener los valores aproximados de ***Y***, luego la clase se encarga de encontrar los residuos del modelo, estandarizarlos y estudentizarlos para luego encontrar los *outliers* y realizar el procedimiento de clasificación. Para realizar todas las tareas anteriores dicha clase utiliza a las restantes clases del bloque. Al igual que en el esquema anterior, se utilizaron coloraciones distintas para la representación gráfica de cada uno de los grupos de entidades. Esta forma de dividir trabajos, basada en el uso de clases y objetos, solamente la podría tener un lenguaje orientado a objetos. El mejor y más poderoso de los que se ha usado, el C++, nos

ha facilitado muy eficientemente la programación y en el uso de las bibliotecas estándares disponibles del lenguaje.

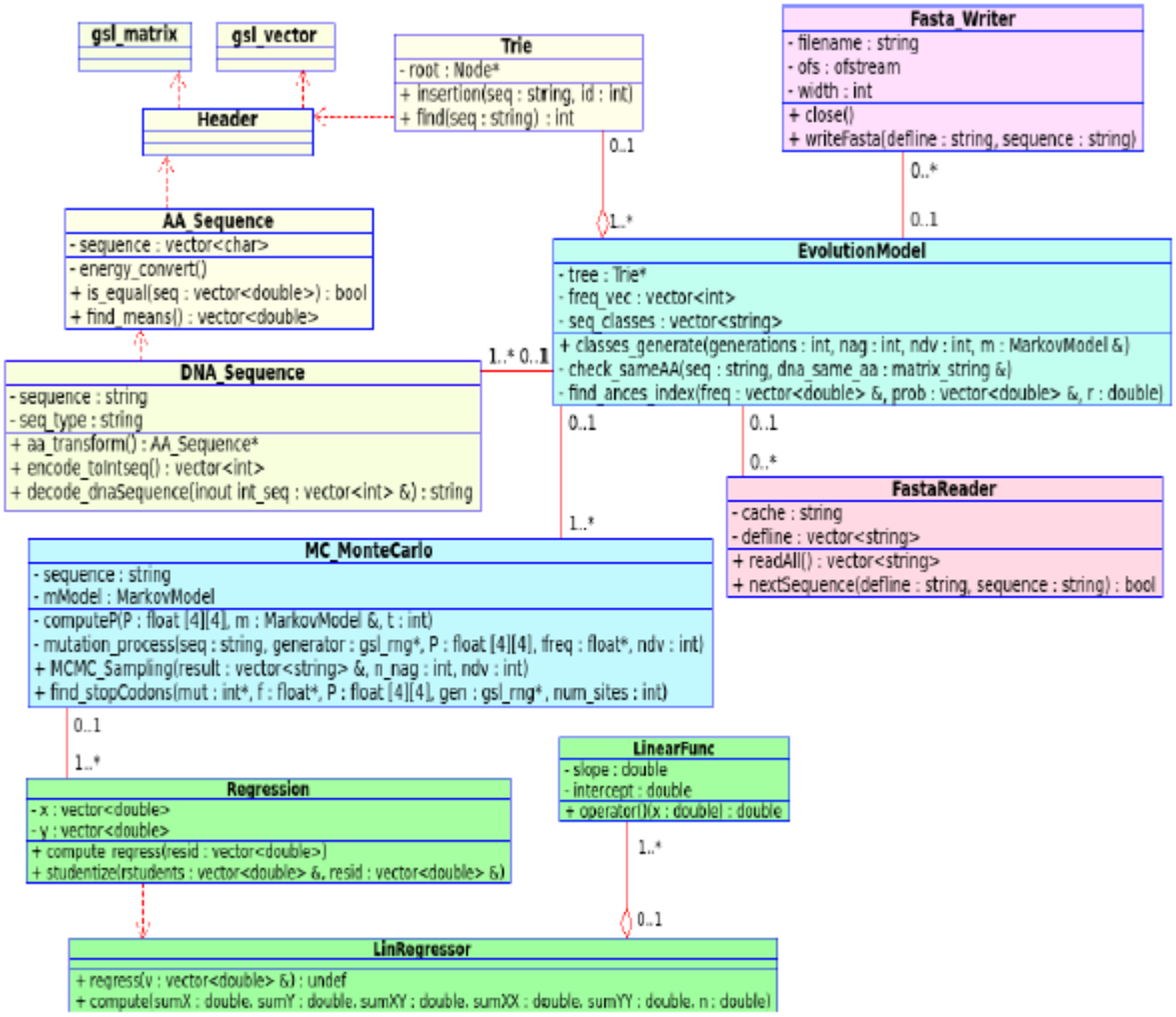


Figura 2.3: Diagrama de clases con presión selectiva.

2.4 Implementación paralela utilizando CUDA de la simulación de las poblaciones virales

El proceso de generar mutaciones de una secuencia genética puede ser en ocasiones muy complejo computacionalmente, y por tanto una aplicación que lleve a cabo el mismo tendría un elevado tiempo de ejecución. En la sección anterior se ha expuesto la forma propuesta en el trabajo anterior, para desarrollar una aplicación que lleve a cabo el mecanismo evolutivo de forma secuencial, a continuación, y dada la importancia, y al hecho de que nuestro trabajo no pretende cambiar la forma propuesta de paralelización del trabajo que nos antecede, sino,

extenderlo a nuevos modelos evolutivos, y reutilizar en la mayor medida el código anterior, se expondrá la propuesta paralela en CUDA hecha por Omar Enrique López {López González, 2014 #92}, tomando en cuenta que nuestra implementación no cambia las clases, ni su estructura, sino los elementos internos, que son necesarios acorde a cada nuevo modelo evolutivo que se desea implementar. Además, al utilizar el C++ para desarrollar la aplicación se garantiza la total compatibilidad existente entre este y CUDA, siendo por lo tanto mucho menor el trabajo a realizar durante la paralelización del procedimiento pudiéndose reutilizar gran parte del código empleado en la versión secuencial.

2.4.1 Estrategias de solución paralela.

Luego de ejecutar la versión secuencial con distintos parámetros de entradas ha sido posible analizar exhaustivamente el tiempo de ejecución de la aplicación. El principal objetivo de este análisis es detectar los momentos del proceso que consumen la mayor parte del tiempo de ejecución. Los procedimientos de mayor consumo temporal que se identificaron fueron la mutación y la clasificación de las secuencias de acuerdo a las proteínas que estas representaban.

Esta última funcionalidad es la que mayor tiempo consumía y de una generación a otra aumentaba el tiempo que esta tarda en realizarse, debido a que se van añadiendo las nuevas mutaciones a la base lo cual implica un mayor número de comparaciones. En la Fig. 2.4 se muestra el flujo de trabajo general llevado a cabo de forma paralela. Las estrategias seguidas para paralelizar estas funciones se exponen a continuación.

Paralelización de las mutaciones

Las mutaciones como se ha mencionado previamente en el trabajo se producen en cada uno de los sitios de las secuencias, teniendo como base inicial de un sitio los nucleótidos que componen el ancestro que se esté mutando. Estos sitios mutan de forma independiente, es decir que las mutaciones que ocurran en uno de ellos no influyen de ninguna manera en las que pueden tener lugar en los otros. Esta independencia es una característica que facilita directamente el proceso de paralelización, pues la misma cumple con las características necesarias para la aplicación de la técnica *SIMD (Single Instructions Multiple Data)*. Dentro de la computación, *SIMD* es una técnica para conseguir paralelismo a nivel de datos. Los repertorios de este tipo consisten en instrucciones que aplican una misma operación sobre un

conjunto más o menos grande de datos. Es una organización en donde una única unidad de control común despacha las instrucciones a diferentes unidades de procesamiento. Todas estas reciben la misma instrucción, pero operan sobre diferentes conjuntos de datos. Es decir, la misma instrucción es ejecutada de manera síncrona por todas las unidades de procesamiento {Grama, 2003 #58}. Aquí, como tenemos que el proceso de mutar un sitio es invariable, y para llevarlo a cabo solo nos interesa la base nitrogenada que posea inicialmente el ancestro y el sitio en el que nos encontremos, entonces programando dicho procedimiento como una operación y aplicándola a más de un sitio durante una ejecución obtenemos la paralelización deseada.

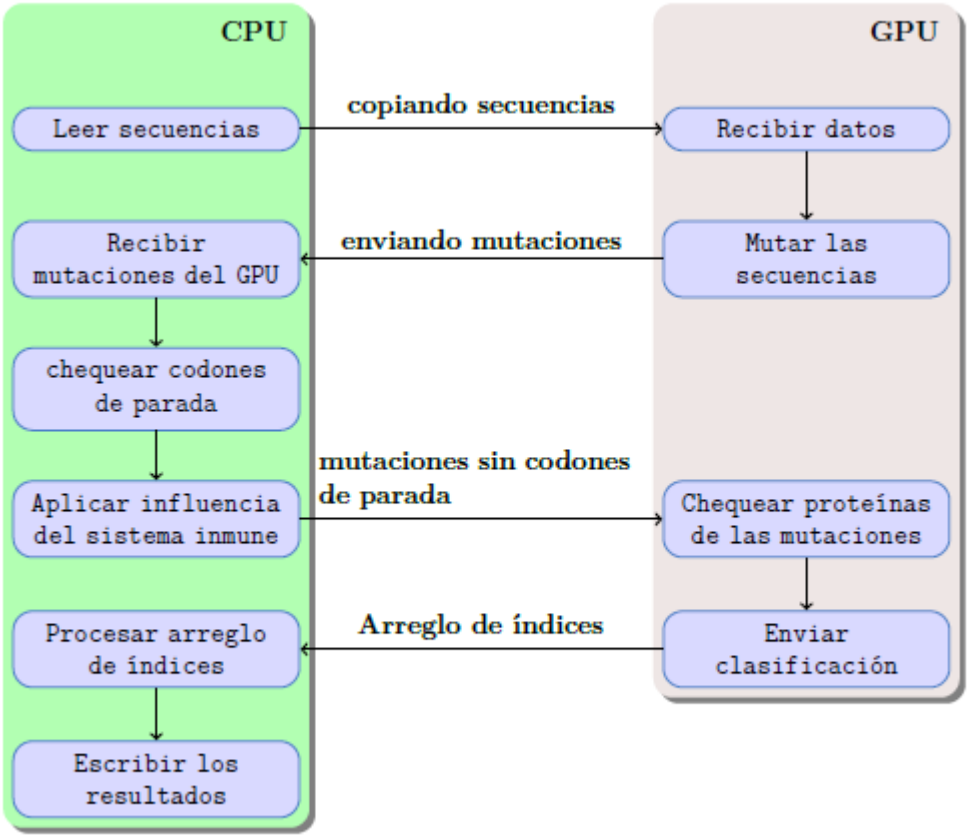


Figura 2.4: Esquema general para la paralelización de la evolución de poblaciones virales.

En la figura 2.5 tenemos un esquema que muestra gráficamente la idea que se ha seguido para lograr el paralelismo a la hora de obtener las mutaciones.

La implementación desarrollada en CUDA de esta funcionalidad se hizo en un kernel ubicado en la clase **MC_MonteCarlo**. Las secuencias se modelaron como arreglos de tipo *char* linealizados, pues al solo contar con 4 bases se desaprovecharía demasiado espacio si para un sitio se utilizara el tipo de dato *short*10 para almacenar la codificación de dicho nucleótido; esta variante posee un menor consumo en memoria, pero el procesamiento de un sitio necesita

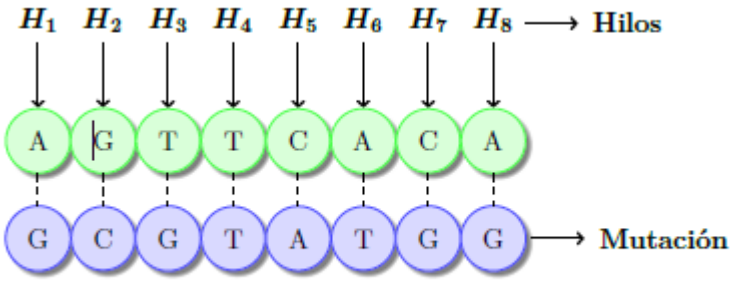


Figura 2.5: Proceso de mutación paralelizado por sitio

de más operaciones debido a que es necesario realizar conversiones de las bases a su respectiva codificación numérica antes de mutar las mismas, y después de obtener la mutación el resultado debe ser codificado como un caracter. Si analizamos el algoritmo 2.1 podemos ver que las mutaciones se llevan a cabo de forma tal que se obtiene una mutación en cada iteración del algoritmo; por tanto, si necesitamos obtener 10 descendientes, el mismo debe ejecutarse 10 ocasiones. En la implementación paralela el algoritmo recibe todos los ancestros de una generación y obtiene todas las mutaciones de estos, dicha tarea se logra en un solo paso.

El pseudo-código 2.6 contiene el procedimiento de las mutaciones paralelizado. Este algoritmo utiliza otras funciones definidas anteriormente como, por ejemplo, la función mutación, la cual se corresponde con el pseudo-código 2.1 explicado en la sección 2.1. Otros métodos como el de codificar las bases no se exponen de forma explícita, pues su tarea es bastante trivial, debido a que no es más que llevar de las bases A, C, G, T a los números 0, 1, 2, 3 y viceversa. Los valores aleatorios ***u, v*** utilizados en el algoritmo se generan dentro del kernel a partir de una distribución uniforme, utilizando la biblioteca CURAND mencionada en 1.7.

La sincronización de los hilos se logra utilizando **_syncthreads()**, la misma pertenece al conjunto de funciones propias de CUDA. Dicha sincronización es necesaria ya que el encargado en cada bloque del cálculo de la matriz ***P*** es el hilo con índice **0**, por tanto los

demás hilos del bloque deben esperar a que este coloque dicha matriz en memoria compartida para luego comenzar las mutaciones. Esta forma de obtener la matriz de probabilidades de transición, a pesar de contener divergencia entre los hilos de un bloque, conduce a un menor consumo temporal gracias a la rapidez con la que se producen los accesos a la memoria compartida por parte de los hilos.

Algoritmo 2.6 Algoritmo paralelo para realizar las mutaciones

Entrada: arreglo de ancestros **A**, arreglo para almacenar las mutaciones **M**, número de ancestros **nag**, número de descendiente por cada ancestro **ndv**, número de sitios **N**, frecuencias de aparición **F**, **alpha1**, **alpha2**, **beta**.

Salida: Mutaciones almacenadas en **M**.

```
1:  Declarar en memoria compartida arreglo P con 16 posiciones;
2:
3:  tid ← id_hilo + id_Bloque * size_Bloque;
4:  Declarar u,v;
5:
6:  while (tid < N) do
7:    for (i = 0 hasta nag) do
8:      pos ← i*nag+tid;
9:      c ← A[pos];
10:     old ← convert_base(c, 0);
11:     j ← 0;
12:     Declarar r;
13:
14:     repeat
15:       if (id_hilo = 0) then
16:         computeP(P, beta, alpha1, alpha2, F, pos);
17:       end if
18:
19:       Sincronizar_Hilos;
20:       u ← U(0, 1);
21:       v ← U(0, 1);
22:       Declarar new;
23:
24:       mutation(P, old, new, u, v, F);
25:       r ← convert_base(new, 1);
26:       M[i*N*ndv+N*j + tid] = r;
27:       old ← new;
28:
29:       j++;
30:     until (j < ndv)
31:   end for
32:
33:   tid ← tid + size_Malla * size_Bloque;
34:
35: end while
```

Clasificación de las mutaciones mediante sus proteínas

Previamente, mencionábamos el proceso de clasificación que se realizaba con las mutaciones obtenidas en una generación. Este se llevaba a cabo con el objetivo de detectar cuando 2 secuencias distintas sintetizaban una misma proteína. En la versión secuencial esto se hace comparando cada secuencia obtenida con cada una de las que ya se encuentran almacenadas.

En dicha tarea se tienen en cuenta tanto las mutaciones que han sido generadas como las secuencias iniciales. Si las mutaciones se están obteniendo sin presión selectiva la base de casos crece de una generación a otra mediante miles de nuevas mutaciones y además el algoritmo que logra esta funcionalidad tiene una complejidad polinomial $\boldsymbol{O(n^2)}$, por tanto la ejecución del mismo en la CPU consume una cantidad de tiempo considerable, siendo CUDA una alternativa de solución para reducir el tiempo de ejecución.

Para aplicar el paralelismo a este procedimiento de clasificación hemos utilizado un arreglo de índices (***I***), el cual tendrá una posición por cada una de las secuencias obtenidas en una generación; es decir, que su longitud será igual a ***nag _ ndv***, este es inicializado con $\boldsymbol{-100}$ en todas sus posiciones.

En los sitios de este arreglo se tendrá $\boldsymbol{-100}$ en caso de que la correspondiente secuencia no sintetice una proteína obtenida anteriormente, un valor del intervalo $\boldsymbol{[0, nag - 1]}$ si la misma produce una proteína igual que algunas de las correspondientes a los ancestros o el mismo estará en el intervalo $\boldsymbol{[nag, nag + nag\ \_\ ndv - 1]}$ si la proteína se corresponde con algunas de las mutaciones que se generaron junto con ella en la actual generación.

En el algoritmo se asigna un hilo a cada una de las secuencias; es decir, de forma diferente a como se paralelizaron las mutaciones, en las que un hilo era responsable de la mutación de un sitio. Luego que el hilo sabe que mutación le corresponde entonces este realiza la comparación entre su respectiva secuencia y todos los ancestros, si la misma no es clasificada por ningún ancestro se procede a compararla con cada una de las mutaciones que se obtuvieron de forma posterior a ella, esto último se hace buscando las que son clasificadas por ella.

Suponiendo que estamos analizando la mutación ***Mj*** , en caso de que la clasificación mediante los ancestros sea positiva para el ancestro ***Ai*** se asigna ***i*** a la posición ***j*** del arreglo que se tiene compartido, si la clasificación se lograra para algunas de las secuencias ***Mk*** con $\boldsymbol{k > j}$, entonces lo que hace el algoritmo es que en la posición correspondiente a la mutación ***k*** asigna ***j***. Notemos que durante el algoritmo si una posición cumple que $\boldsymbol{Ii = -100}$ cuando va a ser

procesada, entonces dicho hilo pasa a procesar otra secuencia pues la actual ha sido clasificada. Como se hace uso de un arreglo en memoria compartida y el mismo es escrito y leído por los hilos de forma simultánea, entonces este constituye una sección crítica, por tanto se hace uso de un semáforo para realizar tanto las lecturas como las escrituras de forma atómica. El algoritmo 2.7 contiene el pseudo-código que realiza el procedimiento descrito anteriormente.

Los arreglos que se muestran como parámetros de entrada se encuentran linealizados, en la implementación de la aplicación la conversión de los mismos a aminoácidos se realiza dentro del algoritmo pero esto no se ha mostrado así por cuestiones relativas al diseño del documento. No obstante, la idea central del método gira alrededor de los arreglos de aminoácidos.

Algoritmo 2.7 Algoritmo para clasificar las mutaciones

Entrada: aminoácidos de las mutaciones $\boldsymbol{M}$, aminoácidos de los ancestros $\boldsymbol{A}$, índices $\boldsymbol{I}$, num_sitios, nag, ndv

Salida: Arreglo de índices ($\boldsymbol{I}$) con valores de las clasificaciones.

```
1: codons ← num_sites/3;
2: tot_seqs ← nag * ndv;
3: tid ← id_hilo + id_Bloque * size_Bloque;
4: while (tid < tot_seqs) do
5:   if (I[tid] = -100) then
6:     flag ← false; i ← 0;
7:     while (!flag && i < nag) do
8:       j ← 0;
9:       while (j < codons) do
10:        if (A[i * codons + j] != S[tid * codons + j]) then
11:          break;
12:        end if
13:        j ← j + 1;
14:      end while
15:      if (j = codons) then
16:        flag ← true;
17:        if (I[tid] = -100) then
18:          I[tid] ← i;
19:        end if
20:      end if
21:      i ← i + 1;
22:    end while
23:    if (!flag && I[tid] = -100) then
24:      i ← tid + 1;
25:      while (i < tot_seqs) do
26:        j ← 0;
27:        while (j<codons) do
28:          if (S[tid * codons + j] != S[i * codons + j]) then
29:            break;
30:          end if
31:          j ← j + 1;
32:        end while
33:        if (j = codons && I[i] = -100) then
34:          I[i] ← nag + tid;
35:        end if
36:        i ← i + 1;
37:      end while
38:    end if
39:  end if
40:
41:  tid ← tid + size_Malla * size_Bloque;
42:
43: end while
```

2.4.2 Presión selectiva y paralelismo.

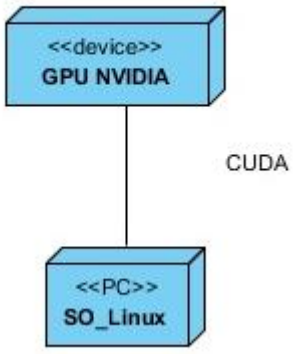
El proceso de presión selectiva explicado en la sección 2.2.2 a pesar de ser costoso no se desarrolló en paralelo, debido a las características que posee el mismo. Una de las razones principales por la que se desarrolló esta funcionalidad de forma secuencial es la alta linealidad que encontramos dentro del procedimiento, como se explicó anteriormente si obtenemos una secuencia que no es reconocida por la vacuna con la cual se ha hecho el análisis de regresión lineal, entonces se añade la nueva secuencia a la población, se toma como ancestro y por tanto también como vacuna. Por lo expuesto podemos apreciar que no es correcto tomar un ancestro como vacuna común para todos sus descendientes, pues para saber con quién analizar la secuencia $\boldsymbol{Sk}$ debemos haber analizado todas las secuencias $\boldsymbol{S0, S1, \ldots, Sk-1}$, lo cual dificulta en gran medida la paralelización. Una posible solución a este inconveniente es tener un hilo por cada ancestro, de forma tal que este realice todos los chequeos necesarios entre el ancestro correspondiente y sus respectivas mutaciones, logrando de esta manera aplicar paralelismo al problema, pero como en cada generación de los experimentos realizados el número de ancestros se encuentra entre **2** y **5**, siendo este último entonces el máximo número de hilos posibles, esto implica que no se aprovechen las bondades que brindan las
GPU en cuanto a capacidad de cálculo, ya que estas son capaces de lanzar miles de hilos en una ejecución.

Otra de las ventajas que ofrece una versión secuencial que realice la simulación de la influencia del sistema inmune en las mutaciones es la disponibilidad de la GSL. Se expuso en la sub-sección 2.2.2 que para llevar a cabo el cálculo de los residuos estudentizados de un modelo de regresión lineal, se deben utilizar elementos algebraicos como la inversa de una matriz y su traspuesta. Esta biblioteca mencionada con anterioridad nos brinda varios mecanismos del álgebra lineal, a partir de los cuales es posible lograr una correcta implementación de las funcionalidades deseadas. Por el contrario, en caso de desarrollar una alternativa paralela para esta tarea se tendría que implementar utilizando CUDA todas las funciones necesarias para lograr una correcta clasificación, haciéndose entonces mucho más engorroso el desarrollo de la aplicación.

Por tanto, la etapa "Aplicar influencia del sistema inmune" mostrada en la Fig. 2.4, no se implementará de forma paralela utilizando CUDA. Esta característica es la que distingue a la variante que simula el proceso de selección respecto a su homóloga sin selección.

2.5 Despliegue de la aplicación y pruebas

El sistema que implementa este trabajo, se ejecutará en una PC con sistema operativo Linux, dada la gran versatilidad de este sistema operativo, no obstante, el código desarrollado no es dependiente de la plataforma donde fue escrito, con solo recompilarlo con las herramientas adecuadas puede ser utilizado en cualquier SO. Como requisito, la máquina donde se desee ejecutar la aplicación debe poseer una GPU NVIDIA que soporte la tecnología CUDA, ya sea integrada, o una tarjeta de procesamiento gráfico. El soporte para la API CUDA fue incluido en las GPUs de NVIDIA a partir de la GeForce 8800 GTX, el diagrama de despliegue es el siguiente:



Resulta necesario, para comprobar el correcto funcionamiento de nuestra aplicación, y garantizar que no sea susceptible a los errores que un usuario puede tener, realizar un conjunto de pruebas de caja negra, a continuación se muestran los casos de prueba fundamentales para la aplicación, y su diseño:

Nombre de la sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo Central**
SC 1: Simular evolución.	EC 1.1: Flujo Norma	El usuario realiza la evolución acorde a los parámetros deseados.	1. El usuario ejecuta el programa con los parámetros deseados. 2. El sistema simula la evolución molecular de las secuencias genéticas virales, y muestra resultados.
	EC 1.2: Flujo Alternativo Parámetro(s) incorrecto(s)	El usuario introduce parámetro(s) incorrecto(s) o no especifica alguno, y el sistema muestra un aviso de error.	1. El encargado introduce sus parámetros. 2. El sistema verifica que se especifican todos, y la valides de cada uno de ellos, y al encontrar algún problema, se lanza un mensaje de error.

Id del escenario	**Escenario**	**Variable X** *(nag, ndv, ng, sequence_file, ACR_file, result_file)*	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 1	Flujo Alternativo: Parámetro(s) incorrecto(s)	Incorrecta	Se espera que el sistema identifique que se ha introducido parámetro(s) incorrecto(s) o que falta alguno.	Al realizar esta prueba se obtiene un mensaje de error especificando el correcto uso del programa, o el parámetro erróneo, y su rango de valores permitidos
		Incorrectas		

Todas las pruebas de caja negra realizadas sobre el software, funcionaron correctamente, obteniéndose en todos los casos de prueba planteados, la respuesta esperada. A continuación se exponen las conclusiones parciales de este capítulo.

2.6 Conclusiones parciales del capítulo

Se ha desarrollado una herramienta para simular la evolución de secuencias de ADN a partir de una población inicial de secuencias virales, acorde a un modelo evolutivo dado, basada en la técnica de simulación de Monte Carlo.

Los algoritmos se implementan usando el paradigma de programación orientada a objetos. Las versiones paralelas se desarrollaron mediante el uso de CUDA, con el esquema general de paralelización propuesto en la tesis anterior. El estudio desarrollado permitirá simular la evolución de las poblaciones virales, tanto para el caso sin restricciones como para el caso en el que se aplica presión selectiva de las secuencias mutantes, para diferentes modelos evolutivos Markovianos.

A tal fin se desarrollaron 2 aplicaciones (una con presión selectiva y otra sin ella), tanto de forma secuencial, como paralela (utilizando CUDA), para cada uno de los modelos evolutivos que se quieren implementar. En el capítulo siguiente se realizarán simulaciones, con bases de datos reales, para cada uno de los modelos evolutivos implementados, a fin de realizar comparaciones, tanto entre los modelos evolutivos, como entre las versiones paralelas y secuenciales de cada uno de ellos.

3 Resultados y discusión.

En este capítulo se realizarán comparaciones, no solo entre los tiempos de ejecución de las versiones secuenciales *vs* paralelas, de los diferentes modelos evolutivos implementados, sino también entre dichos modelos evolutivos. Para ello se realizaron simulaciones del proceso de evolución molecular de poblaciones virales del virus H3N2, y el virus H1N1, tomadas de la base de datos del GenBank en el NCBI (http://www.ncbi.nlm.nih.gov/).

3.1 Características del Software y Hardware utilizados.

La aplicación se desarrolló utilizando el *IDE* ***Netbeans* 7.4**, sobre el sistema operativo ***Debian* 7.3**. Los experimentos se realizaron sobre dos computadoras con 3 GB de memoria RAM y un microprocesador **Intel Core 2 Quad** modelo **Q8200**. Cada una de ellas cuenta con una tarjeta de video cuyas especificaciones se muestran en la tabla 3.1. Como se observa, la primera de las 2 tarjetas posee una mayor capacidad de cómputo puesto que contiene el cuádruple de núcleos CUDA que la segunda.

Procesador	Arquitectura	Memoria	Núcleos
GeForce GT 630	GK107	2048 Mb	192
GeForce GT 610	GF119	1024 Mb	48

Tabla 3.1: Especificaciones de las tarjetas gráficas.

3.2 Comparaciones entre las versiones secuenciales y paralelas.

Las pruebas realizadas entre las versiones secuenciales y paralelas de cada uno de los modelos evolutivos que se implementó en este trabajo y el modelo evolutivo del trabajo anterior, tienen como objetivos verificar las ventajas que nos proporciona CUDA en cuanto al tiempo de ejecución, mostrar si la aplicación paralela posee algún tipo de escalabilidad.

Los parámetros de las pruebas se tomaron de forma tal que el número de ancestros por generación (nag) varía su valor en el intervalo entero **[2, 5]**, la cantidad de descendientes por ancestro (ndv) debe tomar uno de los valores **{1000, 1500, 2000, 2500, 3000}** y el número de generaciones(ngen) se establece en dependencia de la base de secuencias que se estén

procesando (en los experimentos realizados se toma el valor 1 y 2); estos parámetros se tomaron de acuerdo a lo expuesto en (Minh, 2010).

Se comienza la comparación entre las versiones secuenciales y paralelas de los modelos evolutivos implementados, y del modelo TN93 implementado en la tesis anterior, haciendo un análisis descriptivo de los datos. Es examinado el comportamiento de los tiempos de ejecución de las cuatro versiones de cada modelo evolutivo (secuencial, secuencial con selección, paralelo y paralelo con selección) de forma general, a través de medidas de tendencia central y de variabilidad.

A continuación se presenta el resultado de este estadístico descriptivo en el SPSS:

Estadísticos

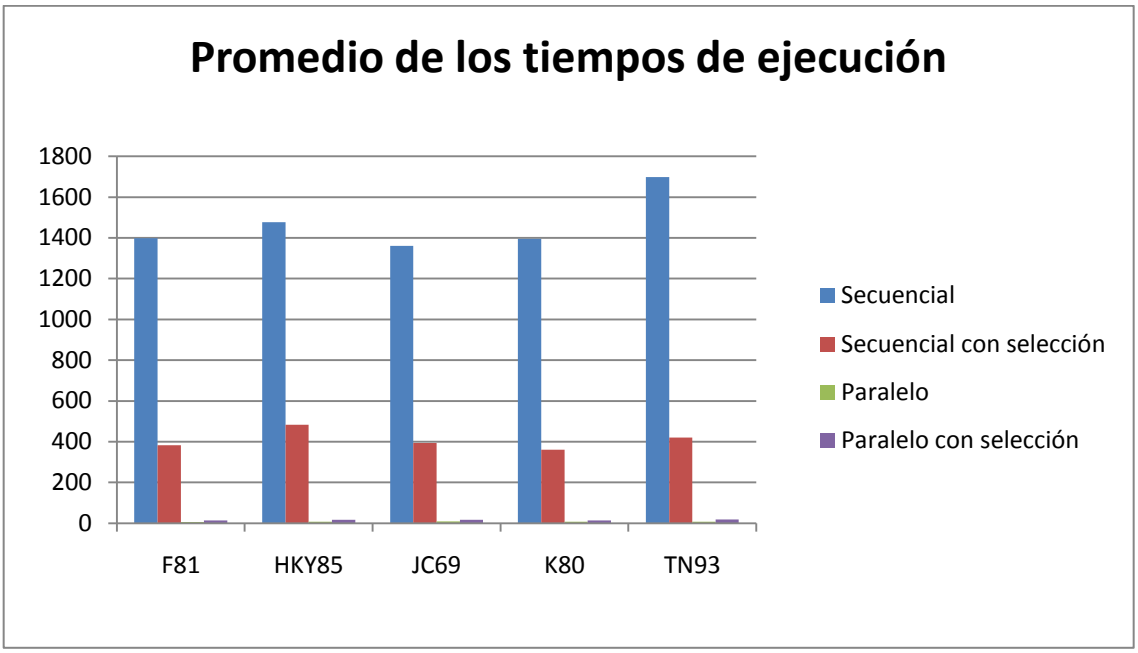
		Secuencial	secuencial_seleccion	paralelo	paralelo_seleccion
N	Válidos	15	15	15	15
	Perdidos	0	0	0	0
Media		1466,119333	407,984667	7,410000	15,773333
Mediana		1236,870000	26,000000	7,290000	20,020000
Moda		2,5400	5,5300	,2700[a]	,9300[a]
Desv. típ.		1331,1882639	579,1332973	5,3893652	11,7230814
Varianza		1772062,194	335395,376	29,045	137,431
Mínimo		2,3300	3,9000	,2700	,9300
Máximo		3648,1200	1418,6300	14,6400	30,8700

a. Existen varias modas. Se mostrará el menor de los valores.

Como promedio la versión de software que mayor tiempo de ejecución consume es la secuencial, cuya media es de 1466,119333 segundos para las simulaciones realizadas, y la de menor tiempo con una media de 7,41 segundos es la versión paralela.

La versión secuencial sin selección, alcanza las mayores medidas de dispersión, con valor de 1331,1882639 evidenciando el notable incremento del tiempo de simulación a medida que la base de casos de secuencias aumenta de tamaño, así como el número de generaciones.

Se efectúa a continuación un análisis más detallado por modelo evolutivo, del promedio del tiempo de ejecución en cada una de las versiones, a través de un gráfico de barras.



La diferencia entre las versiones secuenciales, y las paralelas es notable en el gráfico anterior, por otra parte, es necesario destacar que el grafico anterior muestra el valor promedio de cada versión (por modelo evolutivo), por lo que no puede plasmar, una tendencia detectada, y descrita desde la tesis anterior, que describe como disminuye la diferencia entre los tiempos de ejecución de las versiones secuenciales y paralelas, para el caso de simulación de la evolución molecular de las poblaciones virales con presión selectiva del sistema inmune, debido al número de mutaciones que genera cada versión, siendo en la secuencial varias veces menor el número de mutaciones obtenidas, que en la versión paralela, a razón de la naturaleza estocástica de los procedimientos empleados. Esto se puede ver en {López González, 2014 #92}, pues se muestra para un solo modelo evolutivo el proceso de evolución para diferentes parámetros de simulación.

A continuación se muestra el resultado de aplicar la prueba de Mann Whitney sobre estos casos descritos con anterioridad, para poder comparar resultados de la versión secuencial de la simulación bajo presión selectiva contra su versión paralela respecto a los tiempos de ejecución.

Para las pruebas realizadas (las pruebas paramétricas), se utilizan las dos hipótesis expuestas a continuación:

Hipótesis Fundamental (H0): No existen diferencias significativas entre los grupos que se comparan.

Hipótesis Alternativa (H1): Existen diferencias significativas entre los grupos que se comparan.

Cuando la significación obtenida en una prueba determinada es mayor que 0,05, se acepta la Hipótesis Fundamental y se rechaza la Hipótesis Alternativa y cuando la significación obtenida es menor que 0,05, se acepta la Hipótesis Alternativa y se rechaza la Hipótesis Fundamental.

Rangos

	version	N	Rango promedio	Suma de rangos
tiempo_ejecucion	1	13	16,62	216,00
	2	15	12,67	190,00
	Total	28		

Estadísticos de contraste[a]

	tiempo_ejecucion
U de Mann-Whitney	70,000
W de Wilcoxon	190,000
Z	-1,267
Sig. asintót. (bilateral)	,205
Sig. exacta [2*(Sig. unilateral)]	,217[b]

a. Variable de agrupación: versión

b. No corregidos para los empates.

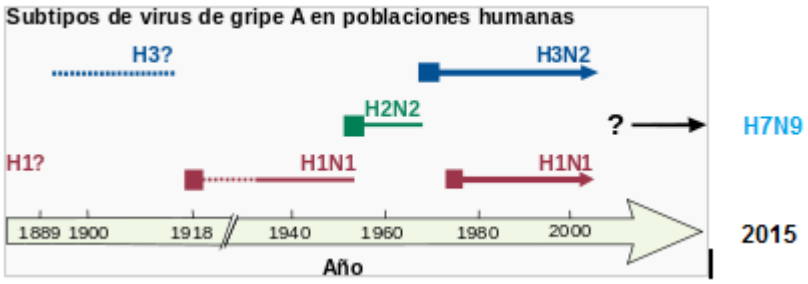
Como se puede observar en los resultados de las comparaciones, la significación es mayor que 0.05 por lo que se acepta la hipótesis de igualdad, concluyendo que no existen diferencias significativas entre los tiempos de ejecución para la simulación del proceso evolutivo de secuencias virales, bajo la presión selectiva del sistema inmune, entre las versiones secuenciales y paralelas, para el caso de los resultados obtenidos en la simulación con nuestra base de casos de secuencias del virus H1N1, para los parámetros 1 generación, 4 ancestros por generación y 2000 descendientes por ancestro.

3.3 Comparaciones entre modelos.

Las comparaciones realizadas entre los diferentes modelos evolutivos, se realizaron con el software IBM SPSS Statistics, también se utilizó el software **BLAST** (**Basic Local Alignment Search Tool**), el programa es capaz de comparar una secuencia problema (también denominada en la literatura secuencia *query*) contra una gran cantidad de secuencias que se encuentren en una base de datos, y calcula la significación de sus resultados, por lo que nos provee de un parámetro para juzgar los resultados que se obtienen. Cuando una nueva secuencia es obtenida, se usa el BLAST para compararla con otras secuencias que han sido previamente caracterizadas, para así poder inferir su función. El BLAST es la herramienta más usada para la anotación y predicción funcional de genes o secuencias proteicas.

BLAST es desarrollado por los Institutos Nacionales de Salud del gobierno de EE. UU, por lo que es de dominio público y puede usarse gratuitamente desde el servidor del Centro Nacional para la Información Biotecnológica (NCBI) (http://www.ncbi.nlm.nih.gov/BLAST/).

Con el BLAST se procesaron los resultados de simular la evolución molecular del virus **H3N2**, para verificar en qué medida, partiendo de este virus, las nuevas secuencias obtenidas se relacionan con el virus **H7N9-HA-Human**.



Al igual que en el epígrafe anterior, para todas las pruebas realizadas en este epígrafe (las pruebas paramétricas), se utilizan las dos hipótesis:

Hipótesis Fundamental (H0): No existen diferencias significativas entre los grupos que se comparan.

Hipótesis Alternativa (H1): Existen diferencias significativas entre los grupos que se comparan.

Que establecen lo siguiente: (Cuando la significación obtenida en una prueba determinada es mayor que 0,05, se acepta la Hipótesis Fundamental y se rechaza la Hipótesis Alternativa y cuando la significación obtenida es menor que 0,05, se acepta la Hipótesis Alternativa y se rechaza la Hipótesis Fundamental).

Primeramente se considera la comparación del tiempo de ejecución entre los diferentes modelos evolutivos, para ello es necesario comenzar con una comparación de grupos independientes (modelo evolutivo: F81, HKY85, JC69, K80 y TN93) en cada una de las cuatro versiones de simulación (secuencial, secuencial con presión del sistema inmune, paralela, y paralela con presión del sistema inmune). Para este propósito se aplica la **Prueba de Kruskal-Wallis**, con la que se obtiene la siguiente tabla.

Estadísticos de contraste[a,b]

	secuencial	secuencial_seleccion	paralelo	paralelo_seleccion
Chi-cuadrado	,509	,660	,567	,467
gl	4	4	4	4
Sig. asintót.	,973	,956	,967	,977

a. Prueba de Kruskal-Wallis

b. Variable de agrupación: modelo

Como la significación es mayor que 0,05 en todas las versiones de simulación, se acepta la hipótesis fundamental **H0**, significa que no existen diferencias significativas en los tiempos de ejecución de las versiones de software entre los modelos evolutivos, acorde a los resultados obtenidos de la simulación realizada con la base de casos de secuencias virales H2N2, es decir, a los efectos del tiempo de ejecución, es prácticamente igual hacer una simulación con la versión secuencial sin selección del modelo TN93, que hacerla con la versión secuencial sin selección del modelo JC69, por citar un ejemplo; eludiendo la diferencia en cuanto a complejidad de los dos modelos evolutivos, ahora bien, es necesario destacar que la base de casos de secuencias virales utilizada para obtener los resultados de esta simulación fue la base H2N2, una base de casos relativamente pequeña, que cuenta con tan solo 198 secuencias cada una con 1765 sitios, cabe preguntarse si ocurrirá lo mismo, para iguales parámetros de simulación, con una base de casos mucho más grande, por ejemplo la base del virus H1N1 de 2451 secuencias y 1803 sitios por secuencia.

A los tiempos de ejecución de esta simulación con la nueva base de casos (H1N1) e iguales parámetros de simulación (ngen, nag, ndv), se le aplicó la **Prueba de Kruskal-Wallis** para comparar nuevamente los grupos independientes (modelo evolutivo: F81, HKY85, JC69, K80 y TN93) en cada una de las cuatro versiones de simulación (secuencial, secuencial con presión del sistema inmune, paralela, y paralela con presión del sistema inmune), obteniendo los siguientes resultados:

Estadísticos de contraste[a,b]

	secuencial	secuencial_seleccion	paralelo	paralelo_seleccion
Chi-cuadrado	3,857	5,891	3,273	4,179
gl	4	4	4	4
Sig. asintót.	,426	,207	,513	,382

a. Prueba de Kruskal-Wallis

b. Variable de agrupación: modelo

Como la significación es mayor que 0,05 en todas las versiones de simulación, se acepta la hipóteis fundamental **H0**, significando igualmente que no existen diferencias significativas entre los tiempos de ejecución de las versiones de software, de los diferentes modelos evolutivos, acorde a los resultados obtenidos en simulación realizada para la base de casos de secuencias virales H1N1, por otra parte, si miramos las significaciones anteriores, y las comparamos con las obtenidas para la prueba de simulación del H2N2 (**,973** | **,956** | **,967** | **,977**), se puede observar que pese a continuar siendo mayores de 0.05 disminuyeron drásticamente su valor, para los casos de simulación con presión selectiva del sistema inmune, corroborando nuevamente la influencia de la naturaleza estocástica de los procedimientos empleados en el proceso de selección.

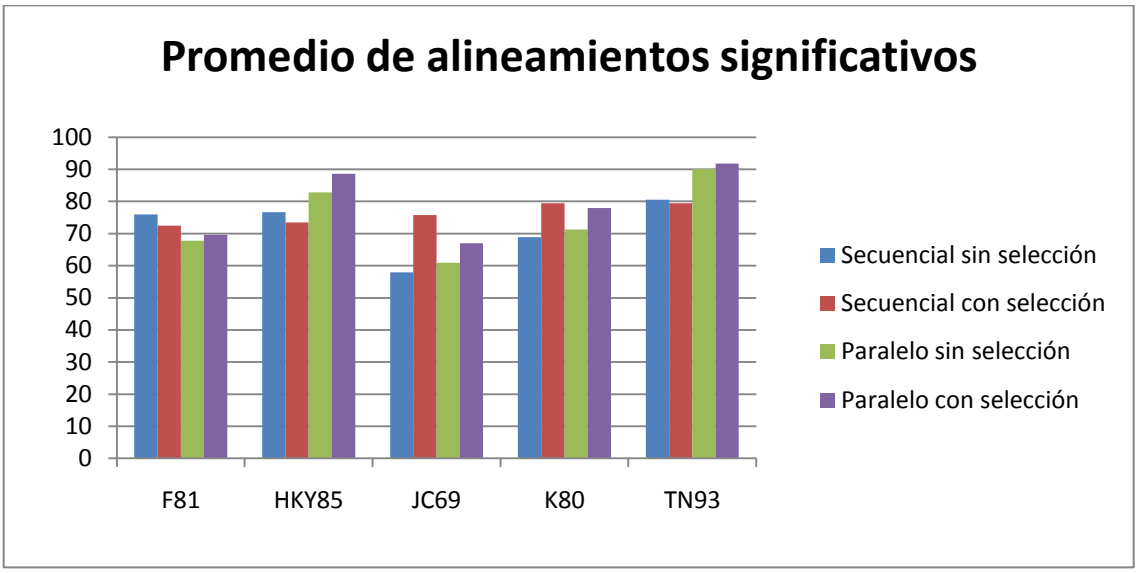
Como anteriormente ha quedado demostrado que no resultan significativas las diferencias en tiempos de ejecución para los distintos modelos evolutivos implementados, a continuación nos centraremos en la comparación de las significaciones de las secuencias generadas por cada modelo evolutivo; se evaluarán en BLAST los resultados de la simulación de la evolución

molecular de secuencias genómicas del virus H2N2 (sin presión selectiva del sistema inmune, y con ella).

De cada una de las simulaciones se toma el porciento del número de secuencias con alineamientos significativos (sas) con el virus H7N9 con respecto al total de secuencias generadas en la simulación (tsg) como criterio de compración:

x% = 100 * (tsg – sas) / tsg

A continuación se muestra un gráfico de barras, con estos resultados porcentuales, de cada una de las versiones, en los diferentes modelos evolutivos implementados:



Como se puede apreciar, de modo general el porciento de secuencias que producen un alineamiento significativo, está estrechamente ligado a la complejidad del modelo evolutivo, asi el modelo TN93, que es el más complejo de todos ellos es el que claramente produce los mejores resultados, y el JC69 que es el más simple, produce los alineamientos significativos más modestos.

Para obtener mayor información de estos datos, se realiza un análisis descriptivo del comportamiento de los resultados de las cuatro versiones de cada modelo evolutivo (secuencial, secuencial con selección, paralelo y paralelo con selección) de forma general, a través de medidas de tendencia central y de variabilidad.

A continuación se presenta el resultado de este estadístico descriptivo en el SPSS:

Estadísticos

		secuencial	secuencial_seleccion	paralelo	paralelo_seleccion
N	Válidos	5	5	5	5
	Perdidos	0	0	0	0
Media		72,01279932	76,13502402	74,57517720	79,01107008
Mediana		75,96153850	75,80794700	71,28712870	77,92207790
Moda		57,952381[a]	72,483222[a]	60,891089[a]	67,021277[a]
Desv. típ.		8,910508783	3,284002219	11,757765843	11,083545450
Varianza		79,397	10,785	138,245	122,845
Mínimo		57,952381	72,483222	60,891089	67,021277
Máximo		80,582524	79,487180	90,109890	91,849462

a. Existen varias modas. Se mostrará el menor de los valores.

Como promedio la versión de software que mejores resultados arroja en el BLAST es la paralela con selección, cuya media es de 79,01107008 % de secuencias que producen alineamientos significativos (en este caso, las secuencias producidas por la simulación de la evolución molecular del virus H2N2, y que producen alineamientos significativos para bases de datos del NCBI, del virus H7N9 entre los años 2013, y la fecha actual), la segunda mejor versión es la secuencial con selección, cuyas secuencias alinean significativamente en un 76,13502402%, este comportamiento no es de extrañar, dado el hecho de que nuestra simulación con selección elimina en su proceso de evolución las secuencias que no ofrecen resistencia al sistema inmune, realizando una simulación mucho más realista que sin selección, y adaptándose mejor al flujo evolutivo de los virus en la vida real. Por último, con medias de 74,57517720 y 72,01279932 se encuentran la versión paralela y por último la secuencial, ambas sin selección.

3.4 Conclusiones parciales del capítulo.

En las secciones anteriores se han expuesto de forma explícita los resultados de las pruebas realizadas a los diferentes modelos evolutivos implementados, en sus versiones secuenciales y paralelas, con presencia o no de la presión selectiva del sistema inmune. Además se realizaron comparaciones, no solo entre las versiones secuenciales y paralelas, sino también entre los

diferentes modelos evolutivos implementados, utilizando la herramienta SPSS. Como es de esperar el resultado de esta discusión confirma que las aplicaciones secuenciales consumen mucho más tiempo que las paralelas. Excepto en aquellos casos en que la versión paralela con presión selectiva genera un gran número de secuencias (debido a que está sujeta a elementos estocásticos, es decir, determinado tanto por las acciones predecibles del proceso como por elementos aleatorios) con respecto a la versión secuencial y esto provoca un mayor tiempo de ejecución en el proceso de selección, concluyendo que la versión paralela no representa para estos casos ganancia respecto a la versión secuencial, esto se debe a que la simulación de la evolución molecular de las poblaciones virales con presión selectiva del sistema inmune. En cuanto a las comparaciones realizadas entre los modelos evolutivos implementados, se evidenció que no existen diferencias significativas entre los tiempos de ejecución de estos modelos, no sucede lo mismo en cuanto a la significación de las secuencias obtenidas, para el caso de simulación del H2N2, y su comparación mediante el software BLAST con bases de datos de secuencias del virus H7N9, se evidenció que la calidad de los resultados es directamente proporcional a la complejidad del modelo evolutivo empleado, también se evidenció que las simulaciones con presión selectiva del sistema inmune, arrojaron mejores resultados que su contraparte sin selección.

Conclusiones

- Los fundamentos teóricos, que describen el proceso de simulación de la evolución genética, en especial la simulación de la evolución molecular de poblaciones virales, según los modelos evolutivos markovianos, y los métodos Markov Chain Monte Carlo, permitieron desarrollar diferentes algoritmos paralelos para la implementación de este proceso de simulación en los modelos JC69, K80, F81, y HKY85, los cuales pueden ser utilizados independientemente a través de la aplicación de consola correspondiente.
- La paralelización en CUDA del proceso de simulación de la evolución molecular de secuencias genéticas virales, sin la influencia del sistema inmune, mejora significativamente el tiempo de ejecución de la aplicación, para los diferentes modelos evolutivos implementados, respecto a sus versiones secuenciales.
- En el proceso de simulación de la evolución de poblaciones virales, con presión selectiva del sistema inmune, existen bases de casos de secuencias, en las cuales la simulación utilizando paralelismo no presenta mejoras significativas con respecto a la versión secuencial, debido a que este proceso está sujeto a elementos estocásticos, es decir, está determinado tanto por las acciones predecibles del proceso como por elementos aleatorios.
- En la simulación realizada con bases de casos de los virus H1N1, y H2N2, se evidenció que no existen diferencias significativas en cuanto a tiempos de ejecución se refiere, entre las versiones desarrolladas, de los diferentes modelos evolutivos implementados.
- El análisis de los resultados obtenidos, con el software BLAST, destacó que la calidad de los resultados es directamente proporcional a la complejidad del modelo de sustitución de nucleótidos empleado, coronando al modelo TN93, como el mejor de todos los implementados.
- Los resultados de la simulación de la evolución molecular de secuencias genéticas virales en presencia de la presión selectiva del sistema inmune, con modelos markovianos, arroja en promedio mejores resultados que su contraparte sin selección.

Recomendaciones

- Agregar a la aplicación nuevos modelos evolutivos, en especial el modelo SG09, cuyos sustentos teóricos ya fueron abordados en este trabajo, quedando solo pendiente su implementación.
- Investigar posibles técnicas de optimización aplicables a la solución propuesta en la paralelización de los diferentes algoritmos abordados, con el fin de mejorar la escalabilidad de la aplicación en las diferentes tarjetas gráficas.

Referencias bibliográficas

2012. *NVIDIA CUDA C Programming Guide 5.0* [Online]. Available: https://developer.nvidia.com. [Accessed 14-11-2014].

ACOSTA, F. A., SEGURA, O. M. & OSPINA, A. E. Modelos de Programación en Paralelo.

ALBA, R. F. 2013. *Reconstrucción de secuencias genéticas utilizando cuda,*. Universidad Central Marta Abreu de Las Villas

ALBERTS, B., JOHNSON, A., LEWIS, J., RAFF, M., ROBERTS, K. & WALTER, P. 2010. *Biologia molecular da célula,* Artmed.

ALEJO, A. C. 2012. *Programación pararlela usando cuda: aplicación en la bioinfórmatica,*. Universidad Central "Marta Abreu" de Las Villas .

BERTSEKAS, D. P. & TSITSIKLIS, J. N. 2000. Introduction to Probability.

CAI, J. J., SMITH, D. K., XIA, X. & YUEN, K.-Y. 2006. *Evolutionary Bioinformatics.,* 2.

CROCHEMORE, M., HANCART, C. & LECROQ, T. 2001. *Algorithms on Strings.* Cambridge

University Press.

DEL TORO MELGAREJO, L. F., LÍO, D. G., BRITO, D. I. T. & ALEJO, A. C. (eds.) 2012. *Acercamiento a la programación paralela utilizando cuda, Technical report*: uclv.

DOOLITTLE, W. F. 2000. Nuevo árbol de la vida. *Investigación y Ciencia,* 283**,** 26-32.

E. GULTEPE, M. 2005. Monte carlo simulation and statistical analysis of genetic information coding. *Science Direct*.

GALASSI, M., DAVIES, J., THEILER, J., GOUGH, B., ALKEN, P. & ROSSI, F. 2013. GNU Scientific Library. Reference Manual.

GAMERMAN, D. & LOPES, H. F. 2006. *Markov chain Monte Carlo: stochastic simulation for Bayesian inference*, CRC Press.

GENTLE, J. E., HÄRDLE, W. & MORI, Y. 2004. Handbook of computational statistics.

GEYER, C. J. 1992. Practical markov chain monte carlo. *Statistical Science***,** 473-483.

GILKS, W. R. 2005. *Markov chain monte carlo*, Wiley Online Library.

GRAMA, A. & GUPTA, A. 2003. Introduction to parallel computing. Addison-Wesley.

GUIL, N., UJALDÓN, M. & DE COMPUTADORES, D. A. 2008. La gpu como arquitectura emergente para supercomputación. *XIX Jornadas de Paralelismo de Castellon*.

J.CAI, J., K.SMITH, D., XIA, X. & YUEN, K.-Y. 2006. MBEToolbox 2.0: An enhanced version of a MATLAB toolbox for Molecular Biology and Evolution. *Evol Bioinform Online.*

LAGUNA-SÁNCHEZ, G. A., OLGUıN-CARBAJAL, M. & BARRÓN-FERNÁNDEZ, R. 2011. Introducción a la programación de códigos paralelos con CUDA y su ejecución en un GPU multi-hilos. *ContactoS,* 80**,** 65-69.

LONDOÑO-GONZÁLEZ, C. A., TORO-ZAPATA, H. D. & TRUJILLO-SALAZAR, C. A. 2014. Modelo de simulación para la infección por VIH y su interacción con la respuesta inmune citotóxica. *Rev. salud pública,* 16**,** 114-127.

LÓPEZ GONZÁLEZ, O. E. 2014. *Aplicación de técnicas paralelas utilizando CUDA, al proceso de simulación de poblaciones en secuencias genéticas.* Ciencia de la Computación, Universidad Central de Las Villas (UCLV).

LORENZO, M. G., ANDRÉS, D. M., MESA, R. S. & VIERNES, H. L. M. M. J. Curso de programación paralela en procesadores gráficos PROFESORES.

LUQUE CABRERA, J. & HERRAEZ SÁNCHEZ, A. 2001. Biología Molecular e Ingeniería Genética. Conceptos, Técnicas y Aplicaciones en Ciencias de la Salud. Ediciones Harcourt. Madrid, España.

MARCELO, M. V. V. 1974. *Algebra lineal*, Pueblo y Educación.

MINH, T. 2010. *Simulación de la evolución de poblaciones del virus de la influenza a/h1n1,*. Universidad Central "Marta Abreu" de Las Villas.

MOREIRA, J. E., SÁNCHEZ, R., DEL TORO MELGAREJO, L., DEL CARMEN CHÁVEZ, M. & GRAU, R. 2011. Parallel implementation of basic algorithms used in the phylogenetic analysis of sequences of the influenza a virus. h1n1. Poster in CICI 2011, International Convention, Informatics 2011.

NEI, M. (ed.) 1975. *Molecular population genetics and evolution*: North-Holland publishing company - Amsterdam, Oxford.

NIELSEN, R., ED . 2005. Statistical Methods in Molecular Evolution. Statistics for Biology and Health. *Springer*.

NVIDIA, C. 2008. Programming guide.

NYLANDER, J. A., WILGENBUSCH, J. C., WARREN, D. L. & SWOFFORD, D. L. 2008. AWTY (are we there yet?): a system for graphical exploration of MCMC convergence in Bayesian phylogenetics. *Bioinformatics,* 24**,** 581-583.

RAFTERY, A. E. & LEWIS, S. M. 1996. Implementing mcmc. *Markov chain Monte Carlo in practice.* Springer.

SÁNCHEZ, R. & GRAU, R. 2009. An algebraic hypothesis about the primeval genetic code architecture. *Mathematical Biosciense*s.

SÁNCHEZ, R., GRAU, R., DEL TORO MELGAREJO, L. F., BROCHE, J. E. M., CHÁVEZ CÁRDENAS, M. D. C. & MIN, T. N. T. 2011. Stocasthic simulation of influenza virus population. Poster in CICI 2011, International Convention, Informatics 2011.

SELANDER, R. K., CLARK, A. G. & WHITTAM, T. S. 1991. *Evolution at the molecular level*, Sinauer Associates.

SPIEGEL, M. R., SCHILLER, J. & SRINIVASAN, R. A. 2009. Probability and Statistics.

SUCHARD, M. A. & RAMBAUT, A. 2009. Many-core algorithms for statistical phylogenetics. *Journal Bioinformatics* 25.

TAMURA, K. & NEI, M. 1993. Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. *Journal Bioinformatics* 10.

WALSH, B. 2004. Markov Chain Monte Carlo and Gibbs Sampling. EEB 581.

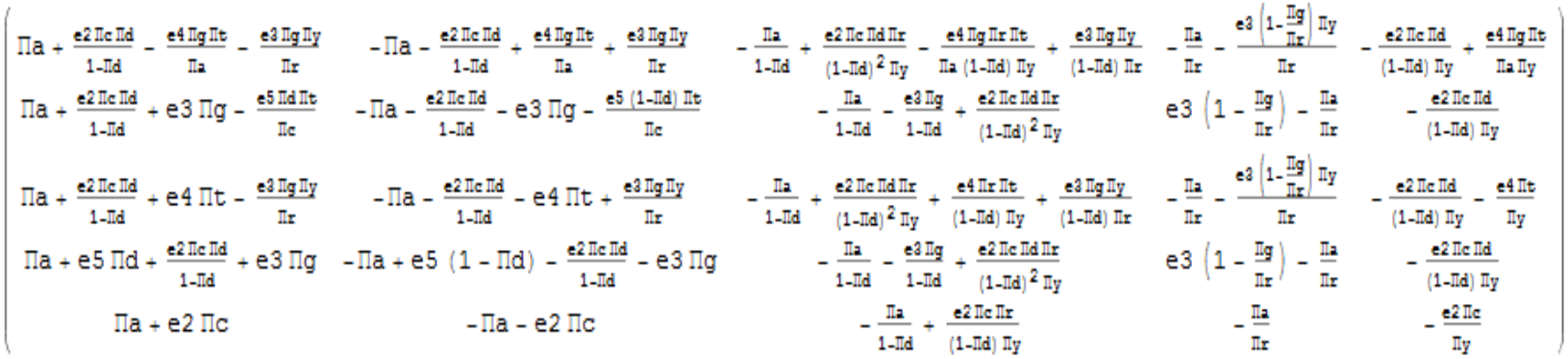
YANG, Z. 2006. Computational Molecular Evolution. *Oxford University Press.*

ZAHA, A., BUNSELMEYER FERREIRA, H. & PASSAGLIA, L. M. 2014. *Biologia Molecular Básica-5,* Artmed Editora.

Anexos

Anexo 1

Matriz de probabilidad para el modelo SG09



[volver]

Anexo 2

Manual de usuario de la aplicación

Como se indicó anteriormente, la aplicación desarrollada es una aplicación de consola, y su forma de ejecutarla es similar a la de un comando clásico estilo UNIX, desde línea de comandos y tomando como parámetros las entradas que necesita. Un ejemplo de ejecución podría ser de la siguiente forma:

$./sequence_mutation_sellection_cuda FASTA_H3N2.fas asr_h3n2.txt 1 2 100

En este caso se está ejecutando el proceso de simulación de una población del virus H3N2, en la versión paralela de nuestro simulador, y con presión selectiva del sistema inmune; Para comprender lo anterior, a continuación se explica el formato general de ejecución de la aplicación.

El formato general de la ejecución de la aplicación para las versiones paralelas es:

Sin selección:

$./sequence_mutation_cuda <secfile> <ASR_f> <ngen> <nag> <ndv> <dir>

Con selección:

$./ sequence_mutation_sellection_cuda <secfile> <ASR_f> <ngen> <nag> <ndv> <dir>

Para las versiones secuenciales el formato general de la ejecución de la aplicación es el siguiente:

Sin selección:

$./sequence_mutation <secfile> <ASR_f> <ngen> <nag> <ndv>

Con selección:

$./ sequence_mutation_selection <secfile> <ASR_f> <ngen> <nag> <ndv>

Donde **<secfile>** y **<ASR_f**> que indican respectivamente el camino al fichero que contiene las secuencias genómicas virales y su archivo ASR generado por el software ASR_MPI, estos caminos pueden ser absolutos o relativos.

<ngen> : Representa el número de generaciones, ngen es un entero mayor e igual que 1, que se establecerá acorde a la base de secuencias que se esté procesando (Para pequeñas bases puede seleccionarse un mayor número de generaciones, en el caso de grandes bases, es crítico el tiempo de ejecución).

<nag> : Representa el número de ancestros por generación, entero cuyo valor es recomendable que se encuentre en el intervalo **[2, 5].**

<ndv> : Representa la cantidad de descendientes por ancestro, es un entero que acorde a lo expuesto en (Minh, 2010) debe tomar uno de los valores **{1000, 1500, 2000, 2500, 3000}**.

<dir> indican el camino absolutos donde se desea guardar las secuencias resultantes del proceso de simulación, en caso de la versión secuencial, que no cuenta en su formato general de ejecución con este parámetro, las secuencias resultantes serán guardadas en la dirección en que se encuentre el archivo ejecutable (**sequence_mutation ó sequence_mutation_selection**).