UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en

Ingeniería de Computadores

Trabajo Fin de Grado

Implementación en C++ de la herramienta MCTS-RNA para el diseño de secuencias de ARN

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en

Ingeniería de Computadores

Trabajo Fin de Grado

Implementación en C++ de la herramienta MCTS-RNA para el diseño de secuencias de ARN

Autor: Mario Bermejo Sánchez

Tutor: José María Granado Criado

Co-Tutor: Álvaro Rubio-Largo

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

En este proyecto se va a portar un código Python a C/C++ con el objetivo de aumentar su velocidad de ejecución, debido a que el problema que se está manejando, el problema del plegamiento inverso del ARN, tiene que hacer frente a un espacio de búsqueda que crece de forma exponencial respecto a la longitud de la secuencia que se introduzca, necesaria para llevar a cabo dicha ejecución. Esto nos lleva a unos tiempos de ejecución elevados para secuencias muy largas, los cuales se cree que se pueden reducir transformando dicho código a un lenguaje compilado, rápido y con más velocidad, a priori, de ejecución.

El código Python que ha sido usado se trata de un algoritmo de plegado inverso de ARN basado en los árboles de búsqueda de Monte Carlo (MCTS), de ahí el nombre del algoritmo: MCTS-RNA; este algoritmo, ha demostrado un buen desempeño a la hora de encontrar soluciones y de controlar con precisión el contenido de los pares de bases GC de las estructuras de ARN sobre otros algoritmos existentes.

Para llevar a cabo este objetivo, se tiene que realizar un proyecto nuevo en C/C++ en el que hay que rediseñar y adaptar gran parte del código debido a las diferencias existentes entre los lenguajes mencionados, a la vez de realizar las pertinentes pruebas para asegurar el correcto funcionamiento y confirmar, en el mejor de los casos, la mejoría en rendimiento.

ABSTRACT

In this project we will try to replicate, transform and redesign a Python code to C/C++ to try to increase its execution speed, because the problem that is being handled, the problem of inverse folding of the RNA, has to deal with a search space with an exponential size with respect to the length of the sequence that is inserted, necessary to carry out said execution. This leads to high execution times for very long sequences, which it is believed that can be reduced by transforming said code to a lower-level language, faster and with lower time in execution.

The Python code that has been used is a reverse folding algorithm based on Monte Carlo search trees (MCTS), hence the name of the algorithm: MCTS-RNA, this algorithm has shown a good ability to find solutions and to control with precision the GC base pair content of RNA structures over other existing algorithms.

To carry out this objective, a new C/C++ project has to be carried out. In this project, a large part of the code must be redesigned and adapted due to the differences between the aforementioned languages, as well as carrying out the pertinent tests to ensure correct operation and, in the best scenario, to confirm the improvement in performance.

1. INTRODUCCIÓN

1.1 Introducción al proyecto y antecedentes.

Este proyecto surge por la idea de querer mejorar la velocidad de ejecución del código Python llevado a cabo y desarrollado por los autores Xiufeng Yang , Kazuki Yoshizoe , Akito Taneda y Koji Tsuda, en el que realizan un algoritmo de plegado inverso de ARN mediante la búsqueda de árboles de Monte Carlo (Yang et al., 2017).

Dicho algoritmo, es un método de búsqueda aleatorio del mejor primero, el cual se ha usado y ha conseguido demostrar un rendimiento excelente en campos que tratan sobre biología computacional, el problema del juego *AlphaGo* (Ilustración 1) usando redes neuronales y árboles de búsqueda, y otros dominios de investigación.

Ilustración 1: Logo AlphaGo.[1]

Se lleva a cabo este algoritmo para intentar mejorar, satisfactoriamente, tanto las soluciones, como el tiempo de ejecución y la capacidad de computación de los algoritmos de plegado inverso de ARN (Ilustración 2) que se encontraban en la literatura hasta el momento. Dichos algoritmos eran *RNAifold*, *IncaRNAtion*, *MODENA* y *antaRNA* . Estos, diseñan secuencias de ARN para estructuras anidadas con control de contenido GC.

MCTS-RNA consigue encontrar más secuencias que coincidan con la estructura objetivo que *MODENA*, *ERD* y *antaRNA* realizando, en un tiempo límite de 10 minutos, un conjunto de extensas confrontaciones experimentales para estructuras

[1] https://www.seekpng.com/ipng/u2t4t4a9e6r5t4i1_alphago-logotype-black-highres-300ppi-copy-google-alphago/

anidadas y pseudoanidadas. Además, *MCTS-RNA* no solo encuentra más coincidencias que los otros algoritmos, sino que también encuentra secuencias coincidentes para algunas familias Rfam complejas en los que dichos algoritmos no han conseguido encontrar la secuencia coincidente dentro de estos 10 minutos límite.

Como se puede observar, estos son unos resultados bastante buenos y esperanzadores, los cuales demuestran la eficiencia de este algoritmo y de *MCTS* a la hora de realizar el plegamiento inverso de ARN, proponiendo una nueva forma de diseñar estos algoritmos para problemas combinatorios de biología computacional.

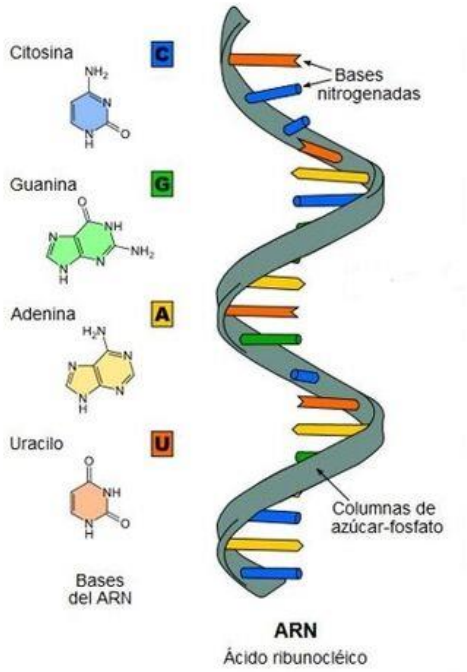

Ilustración 2: Molécula de ARN.[2]

Una de las funciones del ARN, y de los trabajos que se pueden llevar a cabo con él, es el desarrollo de vacunas. En estas fechas que tenemos la pandemia por COVID-19 bastante presente, podemos mencionar que las vacunas desarrolladas por Pfizer y Moderna contienen ARN mensajero (*ARNm*), al contrario de las vacunas más

[2] https://biologia.literaturamagica.net/acidos-nucleicos-adn-y-arn/

tradicionales, en las que se inyecta el patógeno de forma atenuada para que lo reconozca nuestro sistema inmune. El ARN mensajero lo usan nuestras células para fabricar proteínas, son las "instrucciones" para un tipo de proteína específica, es decir, las vacunas de tipo ARNm contienen las instrucciones para fabricar una proteína del virus, lo que nuestro sistema inmune reconoce, provocando una respuesta y memoria inmune sin necesidad de entrar en contacto con el virus. Gracias a ello, estas vacunas son seguras y eficaces, al no entrar en contacto con ningún componente patógeno. Los algoritmos, como el que vamos a tratar en este proyecto, nos ayudan con sus cálculos y son de gran importancia para temas como los mencionados anteriormente.

A lo largo de este trabajo, se llevará a cabo la conversión y sucesivas pruebas del código del algoritmo *MCTS-RNA* a C/C++, junto a todo el desarrollo y explicaciones necesarias.

1.2 Lenguaje y entorno elegidos.

Las principales razones de elegir C/C++ han sido su capacidad de ejecutar código más rápido que Python, al ser este un lenguaje compilado en vez de interpretado, cuyos logos podemos ver en las ilustraciones 3 y 4 respectivamente. Python al tener muchas más clases y funcionalidades ya implementadas, nos permite desarrollar código de manera mucho más abstracta que en C/C++. Debido a esto, muchas veces estamos más abstraídos de lo que está ocurriendo "internamente", por eso C/C++ es una buena opción para desarrollar algoritmia y planteamientos en el código más eficientes que en Python. Por último, otra de las razones esenciales era el conocer el funcionamiento y tener conocimientos sobre el propio C/C++ por mi propia parte, el autor de este TFG.

Ilustración 3: Logo Python.[3]

Ilustración 4: Logo C++.[4]

El entorno elegido ha sido Eclipse (Ilustración 5), siendo la razón principal de su elección la última mencionada anteriormente sobre C/C++, el conocimiento y manejo con ese entorno de desarrollo adquirido a lo largo de la carrera, junto con la capacidad de desenvolverse de manera correcta y el uso, más que aprendido, del modo *debug*, para poder localizar y solucionar todos los errores que pudieran ocurrir, a menudo difíciles de encontrar, durante el desarrollo del código.

Ilustración 5: Logo Eclipse.[5]

Como sistema operativo se ha usado Linux (Ilustración 6), más concretamente la distribución de Ubuntu 18.04 LTS (Ilustración 7). Se ha usado este sistema por la capacidad de integración con los lenguajes Python y C/C++, junto con el entorno Eclipse y el paquete ViennaRNA necesario para ejecutar y llevar a cabo los algoritmos.

[3] https://es.logodownload.org/python-logo/

[4] https://es.m.wikipedia.org/wiki/Archivo:ISO_C%2B%2B_Logo.svg

[5] https://www.freepng.es/png-x1cvhn/

Ilustración 6: Logo Linux.[6]

Ilustración 7: Logo Ubuntu.[7]

1.3 Organización del documento.

La organización de este documento es la siguiente:

- En primer lugar (capítulo INTRODUCCIÓN), nos encontramos con una explicación sobre el proyecto, por qué surge la idea y por qué queremos llevarla a cabo. Además de la razón sobre la selección del entorno y el lenguaje que hemos usado.
- En segundo lugar (capítulo OBJETIVOS), ponemos sobre el papel los objetivos que nos hemos propuesto y pretendemos alcanzar, junto a una breve planificación de cómo llevarlos a cabo.
- En tercer lugar (capítulo ANTECEDENTES / ESTADO DEL ARTE), tendremos información sobre el algoritmo y proyecto sobre el que partimos, poniendo sobre la mesa los datos sobre cómo es la implementación Python, qué es Montecarlo, ya que el algoritmo se basa en él; qué es el problema de plegamiento inverso de ARN, problema sobre el que giran los algoritmos, y, por último, una breve mención a otros algoritmos existentes sobre esta problemática junto con algunas ejecuciones y soluciones con resultados del algoritmo Python.
- En cuarto lugar (capítulo MATERIAL Y MÉTODO), se lleva a cabo la explicación de las herramientas y entorno que hemos elegido, el por qué, y una breve guía de instalación de las mismas. Posteriormente se explica todo el

[6] https://socialgeek.co/noticias/curso-introduccion-linux-gratis/attachment/logo-linux/

[7] https://logos-marcas.com/ubuntu-logo/

proceso seguido de traducción de código junto con algunas decisiones sobre el desarrollo que se han ido tomando. Así mismo, el último punto de este apartado consiste en cómo está organizado el código a nivel de ficheros y qué contiene cada uno.

- Hacia el final del documento, en quinto lugar (capítulos RESULTADOS Y DISCUSIÓN), estarán las ejecuciones y resultados obtenidos en las pruebas llevadas a cabo mediante el código desarrollado, junto con problemas que han surgido a lo largo del desarrollo; en último lugar (capítulo CONCLUSIONES), se presentan las conclusiones alcanzadas sobre el proyecto.

2. OBJETIVOS

El objetivo principal de este proyecto es conseguir elaborar una solución más rápida, eficiente y robusta del algoritmo MCTS-RNA mediante el lenguaje elegido C/C++, usando como base una implementación de dicho algoritmo en Python. De esta forma, se pretende, dejar un proyecto con buena base en el que se podría seguir trabajando en un futuro consiguiendo aún más optimizaciones y rendimiento.

Se llevará a cabo intentando respetar al máximo la solución e idea original, proporcionando cambios de diseño y soluciones distintas a problemas que podamos encontrar al cambiar de un lenguaje a otro, o cosas que, por el propio lenguaje, se pudieran realizar de una forma más agradable y eficiente.

Por último, se realizarán las comprobaciones y pruebas necesarias para observar la posible mejoría en eficiencia y tiempo del programa.

2.1 Planificación de tareas.

Para poder llevar a cabo los objetivos propuestos, la planificación e intencionalidad de llevar a cabo las tareas es la siguiente:

Primero, obtener y realizar un estudio sobre el código Python, intentando entender cómo se conforma y sus diferentes partes, es decir, las clases, cómo enlazan unos algoritmos con otros, cómo funciona la entrada y salida de datos, entender el flujo de datos, etc.

En segundo lugar, comenzar a plasmar las ideas en el código C/C++, partiendo desde escribir el método *main* junto con la entrada de datos y su tratamiento, a ir siguiendo el flujo de datos e ir programando y probando todos los métodos y clases según se van necesitando, por lo que se lleva a cabo un desarrollo de trabajo incremental. De esta manera se puede ir comprobando de “manera sencilla” si los datos que se van obteniendo a las entradas y salidas de los distintos métodos son comparables en ambos códigos, pudiendo así detectar fallas y errores en nuestro código.

Por último, una vez tengamos el programa completamente traducido, comprobar que todo funciona de la misma manera obteniendo las soluciones adecuadas junto con una comprobación de la mejora en rapidez y rendimiento.

Una vez todos estos pasos se hayan llevado a cabo, habremos llegado al final de este proyecto.

3. ANTECEDENTES / ESTADO DEL ARTE

3.1 Notación usada para las secuencias.

La notación que usamos para indicar la estructura de la molécula es la notación punto–paréntesis, con la que indicamos una base solitaria con un punto, o un par de bases con los paréntesis abierto-cerrado respectivamente. Una posible secuencia podría ser: "((((((….))))))" como se puede ver en la Ilustración 8.

Ilustración 8: Ejemplo notación punto-paréntesis y bucle horquilla.[8]

En estas secuencias nos podemos llegar a encontrar diferentes tipos de elementos estructurales (Ilustración 9), como pueden ser:

- *Bucle de horquilla (Hairpin Loop)*: Es un bucle no emparejado de ARN mensajero que se crea cuando una cadena de ARN se pliega y forma pares de bases con otra sección de la misma cadena.
- *Apilamiento de pares de bases (Stacked Pairs)*: Es una disposición común de las nucleobases que se encuentran en la estructura tridimensional de los ácidos

[8] https://eternagame.fandom.com/wiki/Dot-Bracket_Notation

nucleicos. Las bases (o pares de bases) son planas y estos planos se apilan a una distancia de contacto excluyendo el agua y maximizando las interacciones de *Van der Waals*. Las fuerzas de *Van der Waals* son interacciones electrostáticas de débil atracción y de corto alcance.

- *Bucles internos (Interior Loop)*: En el ARN se encuentran donde el ARN de doble cadena se separa debido a que no hay emparejamiento de bases entre los nucleótidos. Este tipo de bucles ocurren en mitad de un tramo de ARN de doble cadena, además, se pueden clasificar como simétricos o asimétricos.
- *Bucle de protuberancia (Bulge Loop)*: Ocurre cuando una cadena de ARN se ve interrumpida por uno o más nucleótidos desemparejados.
- *Bucle múltiple (Multiloop)*: A veces también se denomina bucle de unión o de ramificación múltiple, es una región en la que se juntan tres o más cadenas de bases. Estas cadenas pueden o no estar separadas por bases desemparejadas.
- *Bases desemparejadas (Unpaired bases)*: Como su nombre indica, son bases solitarias que están desemparejadas en la cadena.

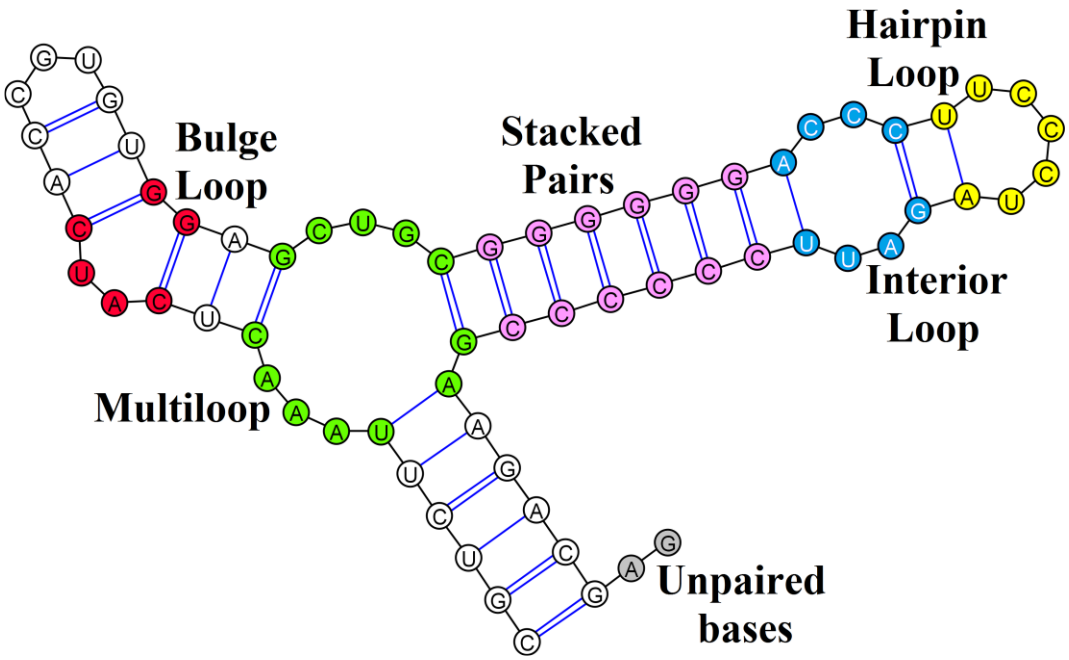

Ilustración 9: Representación de los diferentes tipos de estructuras de ARN.[9]

[9] Rubio-Largo et al., 2019

3.2 Implementación de Python.

Para poder ejecutar y probar esta implementación es necesario disponer del código y, una vez tengamos el equipo preparado, cosa que se explica a partir del capítulo 4 de este documento, podremos llevar a cabo esta tarea.

El código fuente original puede encontrarse en https://github.com/tsudalab/MCTS-RNA. Además, para facilitar su consulta, dicho código se muestra en el Anexo A.

Una vez nos encontramos con el código en nuestro equipo, simplemente abrimos una terminal y nos situaríamos en la misma carpeta del código. La ejecución se realiza de la siguiente manera:

```
python MCTS-RNA.py -s "Secuencia" -GC PorcentajeGC -d DesviacionGC  -pk Estructura
```

- Con -s indicamos la estructura secundaria ARN objetivo, tiene que indicarse entre comillas.
- Con -GC el porcentaje de pares GC objetivo en la secuencia ARN, siendo un valor entre 0 y 1.
- Con -d la desviación de la cantidad de pares de bases GC de la solución, que está en el rango 0 y 0.02. Cuanto más pequeño sea esta desviación, más precisa será la secuencia encontrada. El valor predeterminado de la desviación de contenido de GC es 0.01.
- Con -pk establecemos la estructura que se va a utilizar para la generación de la secuencia, usándose una estructura anidada para el valor 0 y una estructura pseudonudo para el valor 1.

Un ejemplo de ejecución podría ser el siguiente:

```
python MCTS-RNA.py -s "...  (((((..........)))))  ........  (((((((((.. ....)))))))) ((((.......)))))  ............ ((((((.. ( ((((.. ((.. ((. (((((. ((((.......)))).)))))  ...))  ...  .)))))))))))) "-GC 0,75 -d 0,01 -pk 0
```

Respecto al código, nos hemos encontrado con un código en principio bastante complicado de leer (ver el Anexo A), ya que no hay comentarios ni explicaciones de ningún tipo a lo largo del mismo, cosa que ha dificultado la comprensión y traducción.

Junto a esto, he encontrado un par de fallos de programación, el primero, en una condición if-else, lo que me estaba imposibilitando encontrar resultados iguales en ejecuciones idénticas en ambos proyectos. Este error se encuentra en la línea 1126 del código original actual subido al GitHub, en el método *calculate_GC_numbers*, donde se realiza:

```
if eposl[i]==’G’ or ‘C’:

    cunnt=cunnt+1
```

dando lugar a que siempre entra en esa condición al faltar la expresión eposl[i]==’C’ en el otro miembro de la expresión lógica.

El siguiente del que me di cuenta, en Python, en el primer árbol (MCTS), el método que lanza el árbol es UCTRNA. Este método dentro tiene una variable *k*, la cual se rellena de esta forma:

```
k=random.choice(state.GetPositions())
```

donde *GetPositions* simplemente devuelve un vector de enteros respecto a *untriedPositions*. Básicamente son los índices de cada posición no probada, en la primera iteración del bucle del árbol, todos.

Con una estructura de entrada de 83 posiciones sería:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82]

Y *k*, tomaría un valor aleatorio de ellos a través de la función *random.choice*.

Luego una vez dentro de MCTS, la primera vez que *k* es usada, es en este código:

```
if node.untriedPositions != []:
    if k > len(str_uindex)-1:
        if len(node.untriedbpp)==6:
        indice = sRNG.GetUint()%len(node.untriedPositions)
        seleccion = node.untriedPositions[indice]
        k = seleccion
    else:
        if len(node.untriedubpp)==4:
        indice = sRNG.GetUint()%len(node.untriedPositions)
        seleccion = node.untriedPositions[indice]
        k = seleccion
```

Y a partir de aquí, k va tomando un valor entero igualmente para cada iteración del bucle del árbol. Hasta aquí el funcionamiento es correcto.

Pero, en el segundo árbol, (MCTS_noGC), el método que lanza el árbol es UCTRNAnoGC, donde *k* tomaría todo el vector de posiciones:

```
k=state.GetPositions()
```

Posteriormente, en la primera ejecución, y solamente en la primera ejecución del árbol, llega al código anteriormente mencionado, ya que el árbol es bastante similar, y, al realizarse el *if-else: if k > len(str_uindex)-1*, se está comparando un entero, la longitud de un vector que indica las posiciones de los puntos en la secuencia de entrada, con el vector que ha tomado *k* anteriormente, cosa que resulta en que está entrando siempre en esa parte en el *if* en la primera iteración, cuando según el valor aleatorio podría entrar en el *else*.

La forma de solventar esto ha sido la de realizar exactamente la misma operación que en el primer árbol, cogiendo así un valor aleatorio del vector devuelto por *state.GetPositions().*

Una vez descubierto esto, se han corregido todos los errores en el código Python, y en el proyecto C/C++ se programó de la misma manera.

Todas las ejecuciones y pruebas se han realizado con estos errores corregidos, por lo que el código varía en esas dos condiciones expuestas respecto del original.

3.3 Árbol de búsqueda de Monte Carlo.

Como hemos estado mencionando a lo largo de este documento, el algoritmo desarrollado en Python, *MCTS-RNA*, usa, como su propio nombre indica, el árbol de búsqueda de Monte Carlo (*Monte Carlo Tree Search*), el cual se trata de un algoritmo de búsqueda heurístico implementado para la toma optima de decisiones. Gracias a esto combina la precisión de un árbol de búsqueda con la generalidad de las simulaciones aleatorias que se llevan a cabo; a su vez, MCTS va construyendo en memoria un árbol el cual mejora sucesivamente con la estimación de los valores de los movimientos más prometedores. Es especialmente usado en inteligencia artificial y procesos de tomas de decisiones que trabajan con juegos.

Una de sus menciones más destacables es su uso en el juego AlphaGO (*AlphaGo | DeepMind*, n.d.). Este es un programa desarrollado por Google DeepMind, y se encarga de jugar a GO, un juego de mesa. El uso de este algoritmo, ha dado unos muy buenos resultados en el juego.

El funcionamiento de los árboles de búsqueda de Monte Carlo consiste en prestar más atención a los nodos que nos resultan más prometedores, evitando tener que usar la fuerza bruta en todas las posibilidades, ya que esto no es práctico.

Vamos a tener iteraciones repetidas, a priori infinitas, en la práctica limitadas por el tiempo y los recursos que dispongamos de cálculo; estas iteraciones se dividen en 4 pasos: *selección, expansión, simulación* y *retropropagación* (Ilustración 10).

- ***Selección***: En este paso, se construye la ruta desde la raíz hasta el nodo hoja más prometedor, tomando una rama u otra dependiendo de la estrategia elegida de selección y la información que posea ese nodo en ese momento, como por ejemplo el número de visitas. Podríamos tener la estrategia UCT (*Upper Confidence bounds applied to Trees*), siendo una de las más usadas por su eficacia y simpleza. Con esta estrategia, calculamos para cada movimiento posible la combinación de la tasa de éxito del nodo y la relación entre las veces que se ha visitado el nodo padre y el número de veces que se ha visitado el nodo hijo. En AlphaGo por ejemplo, se sigue una política UCB (*Upper*

Confidence Bounds), con la que cuanto más se visita un nodo, menos posibilidades tiene de ser elegido.

- ***Expansión***: En este paso elegimos un nodo inexplorado de un nodo hoja para desarrollarlo, es decir, se añaden nodos al árbol. Igual que en el paso anterior, según el criterio que tengamos, podemos expandir siempre que se visite un nodo o cuando se alcanza un número de visitas mínimo. Por ejemplo, esto último ayudaría a ahorrar memoria, ya que se regula cuánto crece el árbol.
- ***Simulación***: Simplemente se simula una ejecución desde el nodo hoja alcanzado en las fases anteriores.
- ***Retropropagación***: En esta fase, el resultado de la simulación ejecutada anteriormente es propagado desde el nodo hoja hasta el nodo raíz, informando de los resultados obtenidos a cada nodo intermedio. Se podría incrementar por ejemplo el número de visitas junto con el resultado de la simulación.

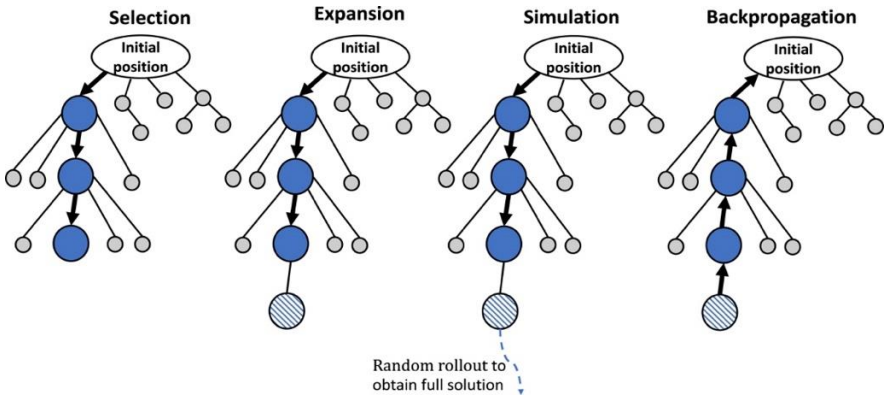

Ilustración 10: Árbol de Monte Carlo.[10]

3.4 Problema de plegamiento inverso del ARN.

Para explicar y poner en contexto este problema, vamos a comenzar exponiendo unas bases y un contexto.

[10] https://www.cambridge.org/core/journals/mrs-communications/article/monte-carlo-tree-search-for-materials-design-and-discovery/E8CFDE0EEC61B217A59BC42827205CBE

Con el término ácido nucleico, nos estamos refiriendo a un tipo de moléculas específicas y extensas en la célula formadas por cadenas de unidades de polímeros que se repiten. Los dos ácidos nucleicos más conocidos son el ADN (Ácido desoxirribonucleico) y el ARN (Ácido ribonucleico) (Ilustración 11), siendo este último nuestro ámbito de trabajo.

Básicamente, una de las funciones que desempeña es almacenar información en la célula, además de ser una proteína muy estable, con lo que, teniendo estas dos características en mente, nos encontramos con que son unas moléculas idóneas para transmitir información genética.

Por una parte, tenemos el ADN, el cual contiene la información genética hereditaria y se encuentra en las células, y una pequeña parte en las mitocondrias. Está formado por estructuras más simples conocidas como bases nitrogenadas, las cuales son: *Adenina, Guanina, Citosina y Timina*. Según el orden de estas bases, se compone nuestra genética.

Por otra parte, nos encontramos con el ARN, el cual nos da la capacidad de realizar la síntesis de proteínas. Este ayuda a que la información genética del ADN sea comprendida por las células. Está compuesto por una cadena simple, en contrapartida del ADN que son dos. Sus bases son: *Adenina, Guanina, Citosina y Uracilo*.

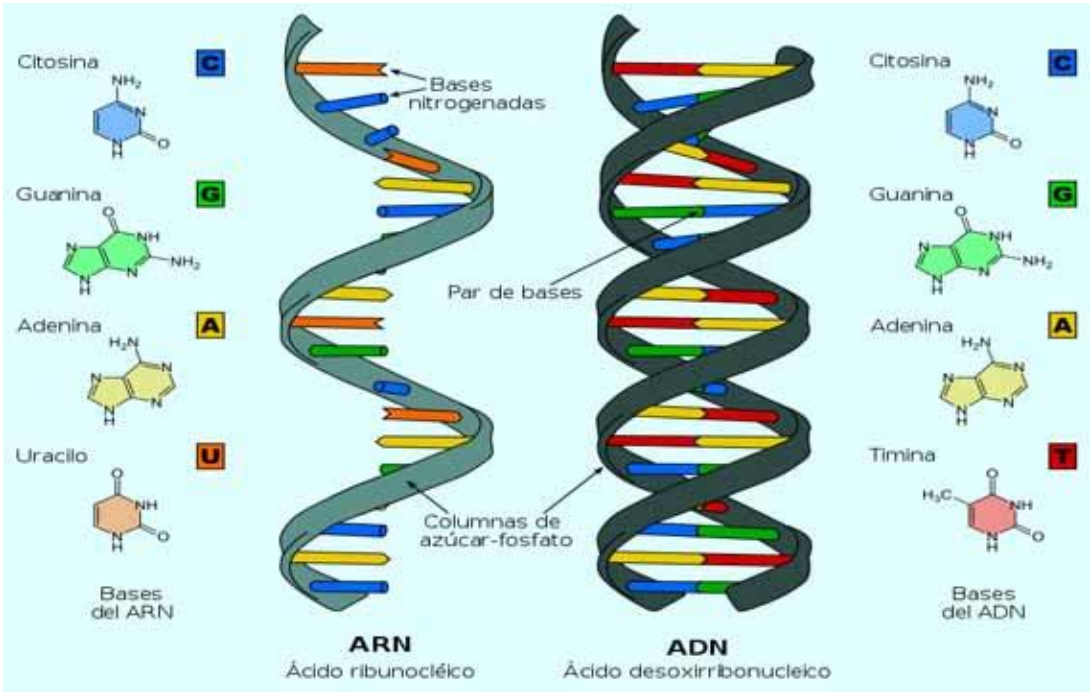

Ilustración 11: Moléculas de ARN y ADN.[11]

Al estar formado por una cadena simple, es posible que pueda dar lugar a la formación de enlaces con otras moléculas o permitir ciertas interacciones secundarias y terciarias consigo misma, que es lo que se conoce por plegamiento espacial.

Respecto al ARN, nos encontramos tres tipos de niveles estructurales: *la estructura primaria*, *la estructura secundaria* y *la estructura terciaria*.

- La *estructura primaria* es una secuencia de bases (AUGAUCUCGUAA…), es decir, una secuencia lineal con un tamaño dado por el número de nucleótidos que la forma, con un principio y fin. Una molécula de ARN sin plegar es simplemente una hélice abierta, siendo esta una situación inestable debido a que cierta parte de la molécula puede formar un enlace con bases de otra parte. Esto es lo que se llama plegamiento y nos deja con las ya mencionadas estructuras secundarias y terciarias.

[11] https://biologia.literaturamagica.net/acidos-nucleicos-adn-y-arn/

- La *estructura secundaria* es un paso intermedio del plegamiento, en el que se forman bucles y pares de bases. Puede ser representada en un plano, es decir, de forma bidimensional.
- La *estructura terciaria* sería el estado final, con interacciones entre ciertas partes de la molécula. Se trata de una representación tridimensional.

Como se puede apreciar con esta explicación, la estructura secundaria de ARN es una parte crucial sobre la que se construye la futura estructura terciaria.

Con el *problema de plegamiento inverso*, tendríamos una estructura secundaria *S* de longitud *l* de la que desconocemos su secuencia y tenemos que encontrar una secuencia de ARN con una energía libre mínima cuya estructura sea *S*. Esta secuencia estará formada por nuestro espacio de búsqueda de bases: A, C, G, U y será de una longitud *l*.

Con tener mínima energía libre nos estaríamos refiriendo al *principio de energía mínima* de la segunda ley de la termodinámica, la cual nos dice que, para un sistema cerrado con parámetros externos constantes y entropía, la energía interna disminuirá y su valor será cercano al mínimo en el equilibrio.

3.5 Trabajo relacionado.

En este apartado dejo enlaces hacía los otros algoritmos a los que se hacen referencia a lo largo de este documento y en la publicación de *MCTS-RNA* con los que se trabaja y se hacen comparaciones, para poder verlos y ampliar cualquier información que se quiera obtener de ellos.

- ***RNAiFold***: Es un algoritmo basado en programación de restricciones y desarrollado por J. A. García-Martín et al. 2013, 2015. Permite al usuario especificar uno de tres criterios de optimización diferentes: (i) MFE, (ii) energía libre o (iii) defecto del conjunto. Además, proporciona una amplia variedad de restricciones de diseño, como límites de pares de bases, motifs específicos, etc.

- ***Modena***: En Taneda, 2010, A. Taneda aplica el conocido algoritmo NSGA-II (*Fast Non-dominated Sorting Genetic Algorithm*) a este problema: MODENA (*Multi-Objective DEsign of Nucleic Acids*). Este enfoque multiobjetivo explora el conjunto aproximado de soluciones óptimas de Pareto no-dominadas en el espacio objetivo de dos funciones: (i) estabilidad y (ii) similitud de la estructura. En Taneda, 2012, el autor extiende MODENA con métodos de predicción de *pseudoknots* (p. ej. *IPknot o HotKnots*). La última versión de MODENA Taneda, 2015, acepta múltiples estructuras secundarias objetivo.

- ***antaRNA***: Resuelve el problema de plegamiento inverso de ARN dada una secuencia de estructura en notación punto-paréntesis, un valor objetivo de bases GC y una restricción de secuencia suplementaria. Para ello, en Kleinkauf et al., 2015 se aplica ACO (*Ant Colony Optimization*), un esquema de búsqueda local que se adapta automáticamente e imita la búsqueda adaptativa de alimento de las hormigas dentro de un terreno dado.

- ***IncaRNAtion***: En Reinharz et al., 2013 se plantea un algoritmo para el diseño de secuencias de ARN que se pliegan en estructuras secundarias con una distribución de nucleótidos predefinida. Usa un enfoque de muestreo global junto a técnicas de muestreo ponderado. Se demuestra que es un enfoque rápido, no tiene semillas (elimina el sesgo en la heurística de búsqueda local), y genera con éxito secuencias de alta calidad para cualquier contenido GC.

4. MATERIAL Y MÉTODO

4.1 Entorno Linux.

Linux es el nombre que obtienen un conjunto de sistemas operativos de tipo *Unix* con licencia *GNU GPL* (*General Public License of GNU*). Se basa en ser un sistema operativo muy ligero, y más tarde, según nos vaya haciendo falta, ir añadiendo todo lo necesario.

Se trata también de un sistema de código abierto, dentro de la categoría de software libre, lo que nos da la capacidad de modificarlo, usarlo o redistribuirlo, siempre y cuando sea bajo la licencia mencionada anteriormente con la que se avala Linux.

Mencionar que, a pesar que de forma coloquial nos refiramos a Linux como el sistema operativo, este en realidad es el nombre del núcleo. Posteriormente, el sistema operativo completo se forma con gran cantidad de componentes del proyecto *GNU*, como compiladores, componentes de terceros, o entornos de escritorio.

La historia de Linux surgió sobre el año 1960, teniendo su origen en *Unix*, creado por Dennis Ritchie y Ken Thompson. Posteriormente, Andrew Tanenbaum hizo un sistema operativo similar a *Unix*, llamado *Minix*, con la idea de usarlo en clase de forma docente para enseñar a sus alumnos. Como Tanenbaum no permitía que el sistema se modificara, debido a la posibilidad de introducir dificultades en el sistema para los alumnos, Linus Torvalds, decidió escribir su propio sistema, compatible con Unix.

Por otra parte, Richard Stallman, tenía su proyecto *GNU* desde hacía ya 10 años, el cual se basaba en un sistema básico casi completo a excepción del núcleo. Torvalds aprovechó el sistema *GNU* completándolo con su núcleo, que acabó llamándose Linux. El sistema al completo es lo que conocemos como *GNU/Linux*.

Una distribución Linux, en nuestro caso *Ubuntu*, está basada en el núcleo Linux pero incluyendo distintos paquetes de software ya preinstalados en el sistema; esa es la idea esencial de las distribuciones o también llamadas distros.

En este proyecto, como se ha mencionado en anteriores ocasiones, vamos a usar Ubuntu 18.04 LTS, una instalación limpia, sobre la que posteriormente instalaremos Eclipse y el paquete ViennaRNA.

En este apartado describiremos la instalación de Linux/Ubuntu de forma abreviada, debido a la cantidad de información y guías/tutoriales que existen para realizarla.

En primer lugar, tenemos que descargarnos la ISO del sistema operativo/distribución que deseemos, Ubuntu 18.04 LTS en nuestro caso, y, una vez la tengamos, crear un USB "Booteable" con la imagen ISO dentro. Esto no es más que un USB de arranque en el que llevamos el sistema operativo dentro y nos permitirá instalarlo en cualquier computadora. Existen gran cantidad de programas para llevar esto a cabo, en este caso, se ha usado el programa Rufus.

La imagen ISO podemos encontrarla en su página oficial: https://ubuntu.com/download/desktop , donde tendremos tanto las últimas versiones como un repositorio con algunas versiones anteriores, como podemos ver en la Ilustración 12.

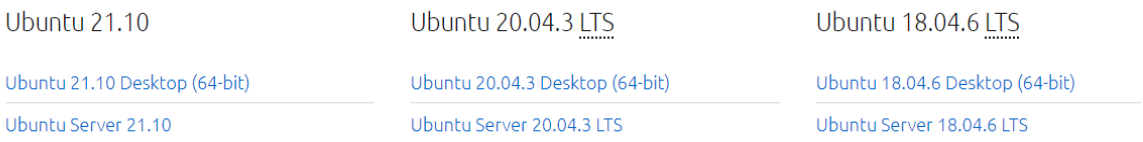

Ilustración 12: Imágenes ISO descargables de SO Ubuntu.

Rufus igualmente podemos encontrarlo en su página oficial: https://rufus.ie/es/ .

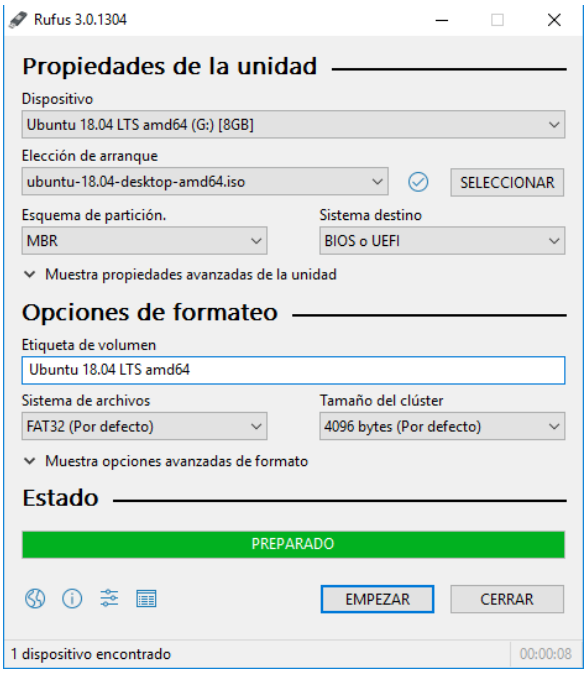

Ilustración 13: Programa Rufus.

Seleccionamos el dispositivo USB, la imagen ISO, le damos un nombre y con las demás opciones por defecto como se ven en la Ilustración 13, le damos a comenzar. Ya tendríamos el USB preparado.

Una vez tenemos esto, se procede con la instalación del sistema, introduciendo el USB y arrancando desde él, ya bien definiendo el orden de arranque en la BIOS, seleccionando que, si tenemos un USB arranque desde él, o bien seleccionar el arranque manualmente al inicio. Con esto entraríamos en el sistema Ubuntu y comenzaría el proceso de instalación (Ilustración 14).

Ilustración 14: Instalación Ubuntu 1.

Aquí se puede hacer una instalación sencilla simplemente siguiendo los pasos, o en ciertos puntos de la instalación, llegar a repartir por ejemplo el tamaño de las distintas particiones que han de realizarse, cosa recomendada si se tienen conocimientos sobre lo que se está haciendo, ya que podremos ajustar el tamaño del sistema a las preferencias que tengamos (Ilustración 15).

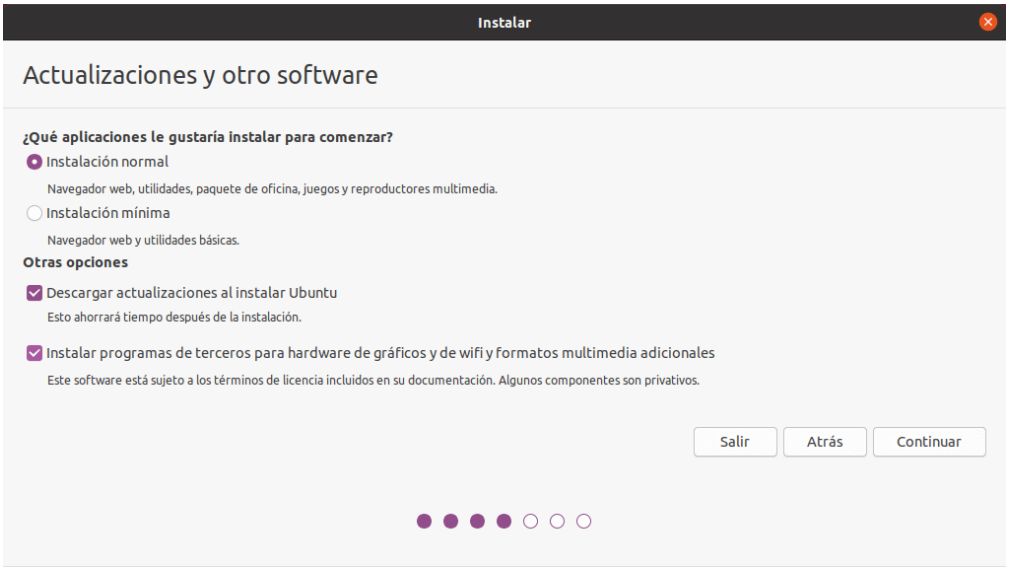

Ilustración 15: Instalación Ubuntu 2.

Seguimos los pasos de la instalación, y, una vez finalizada la misma, ya tendríamos Ubuntu en nuestro sistema preparado para los siguientes pasos.

4.2 Lenguaje C/C++.

El lenguaje de programación C cuenta con varias décadas a sus espaldas, concretamente nos tenemos que remontar a 1972 para ver su nacimiento, junto con el sistema operativo UNIX. Inicialmente nace como lenguaje de propósito general con el que podremos desarrollar desde aplicaciones hasta sistemas operativos y sirve como base para la creación de lenguajes más actuales como C++ y Java.

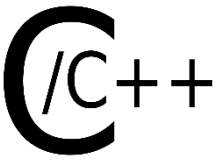

Ilustración 16: Logo C/C++.[12]

C/C++ (Ilustración 16) es apreciado por su eficiencia del código, ofrecer un control total sobre todo lo que sucede en el ordenador o que los programas producidos son bastantes potentes y rápidos, razones por las que hemos elegido este lenguaje para este proyecto.

Por otra parte, C++ es una "extensión" de C, con el que podremos programar con un mayor nivel de abstracción y manipular objetos. Fue creado a mediados de los años 80 y, al ser este una extensión de C, contiene los paradigmas de programación estructurada y programación orientada a objetos, lo que da como resultado lo que conocemos como lenguaje de programación multiparadigma.

[12] https://www.freepng.es/png-7nos4p/

Se puede seguir destacando lo mencionado anteriormente sobre C, este lenguaje sigue siendo rápido y potente, y al tratarse de un lenguaje compilado, para poder ejecutar el código es necesario compilar a bajo nivel, cosa que no ocurre en el caso de Python, al ser este un lenguaje interpretado, lo que nos disminuye notablemente el rendimiento de nuestros programas cuando tratamos operaciones muy complejas.

Para llevar a cabo el desarrollo de este proyecto no necesitamos instalar nada por esta parte, ya que tenemos compilador de base con Linux/Ubuntu, llamado GCC, capaz de compilar código C, C++ y de Objective-C.

Podemos comprobar la versión de nuestro compilador en Ubuntu, una vez tengamos instalado el sistema operativo, si abrimos la consola y escribimos: "**gcc --version**".

4.3 Entorno Eclipse.

Eclipse es el entorno que hemos elegido para desarrollar nuestro código. Es una plataforma de desarrollo de código abierto, disponible para varios lenguajes, entre ellos C/C++ o Java. No nació para dar lugar a un entorno para un lenguaje específico, sino que es un IDE genérico. Esta plataforma puede ser extendida mediante *plug-ins*, algunos de los cuales son extremadamente útiles según lo que estemos haciendo o queramos hacer; a su vez, nos ofrece herramientas para escribir, ejecutar, desplegar y depurar aplicaciones de forma sencilla.

La instalación es bastante sencilla. Visitamos su página oficial, de donde descargaremos el instalador, pudiendo elegir entre uno más completo, en el que nos dejara seleccionar la versión que queramos instalar, si para java, C/C++, PHP… o podemos seleccionar directamente el "Eclipse IDE for C/C++ Developers" para nuestro caso.

Simplemente descargamos, instalamos y ejecutamos.

La página oficial del instalador completo: https://www.eclipse.org/downloads/ (Ilustración 17).

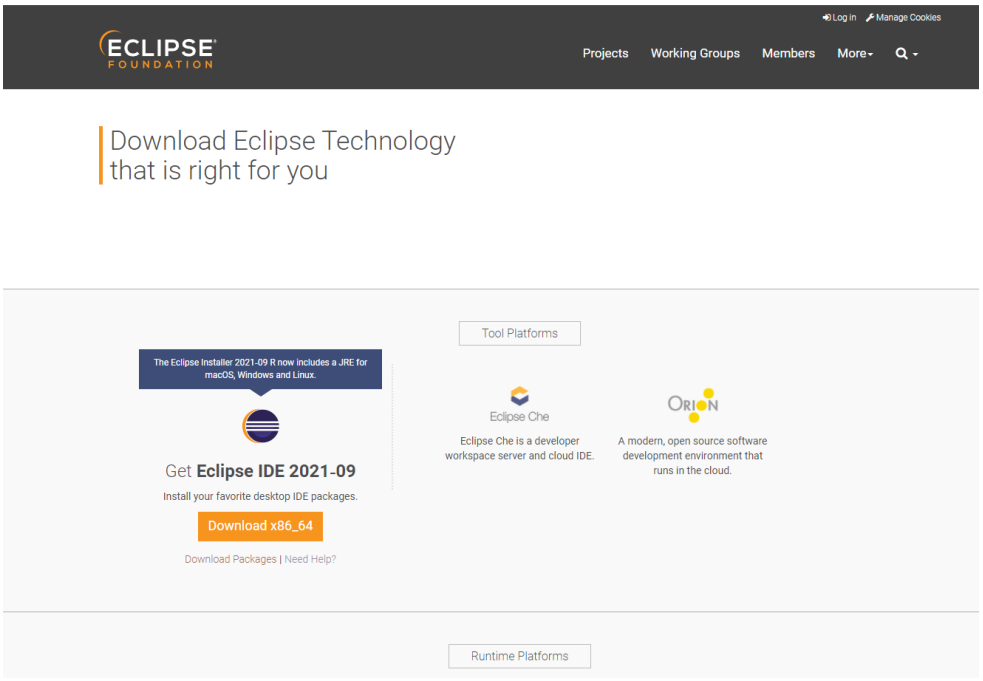

Ilustración 17: Página web de Eclipse.

El instalador es de la siguiente forma como se ve en la ilustración 18:

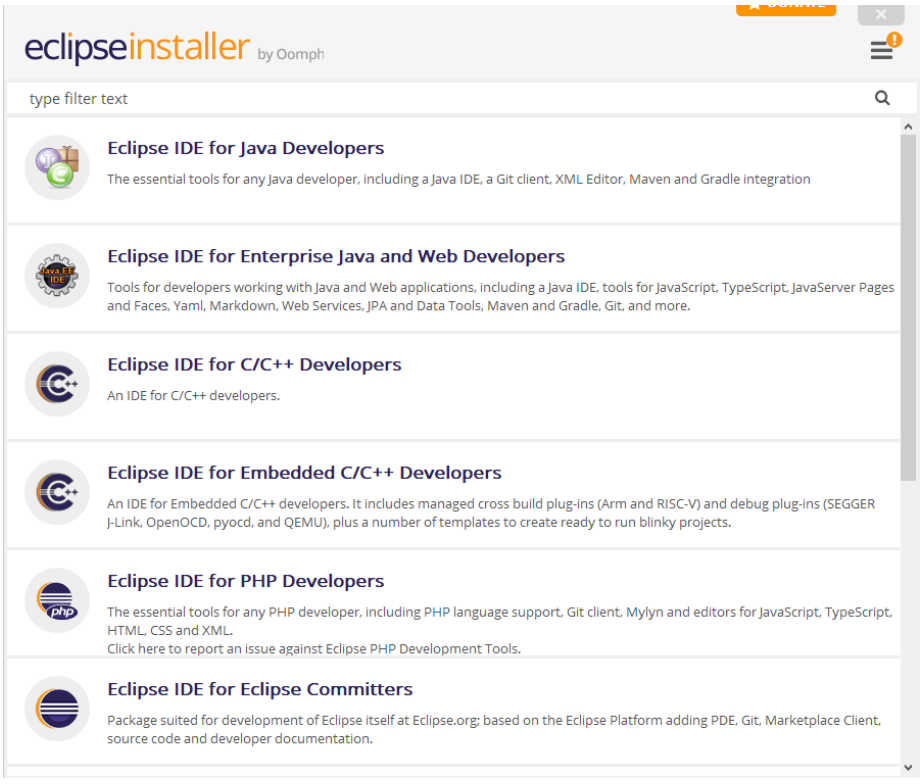

Ilustración 18: Instalador de Eclipse.

4.4 Paquete ViennaRNA.

ViennaRNA es la librería que usamos, tanto en Python como en C/C++, para el cálculo de ciertas operaciones con las secuencias de ARN. Tal y como se explica en su página de desarrollo, el paquete ViennaRNA está formado por una biblioteca de códigos C y varios programas independientes para la predicción y comparación de estructuras secundarias de ARN. Este paquete se distribuye libremente y los binarios están disponibles tanto para Linux, macOS o Windows.

La primera versión fue publicada por (Hofacker et al., 1994) en 1994. El paquete distribuía herramientas para calcular estructuras de energía libre mínima o funciones de partición de las moléculas de ARN.

La función más usada del paquete es la predicción de la estructura secundaria del ARN mediante la minimización de energía.

Algunos de los algoritmos que nos proporciona este paquete son: RNAfold, para calcular las estructuras de energía libre mínima y la función de partición de los ARN, RNA2Dfold, que calcula la estructura de MFE (*Minimum Free Energy*), la función de partición y las estructuras muestrales representativas de la secuencia introducida, o RNAheat que nos calcula el "calor" específico de una secuencia de ARN (curva de fusión).

En cuanto a la instalación de este paquete, es posible que haya algún inconveniente, pero si se han seguido las instrucciones hasta ahora, y tenemos una instalación de Ubuntu limpia, con los siguientes pasos la instalación debería realizarse de forma satisfactoria.

En primer lugar, desplegamos una consola y ejecutamos los siguientes comandos:

```
sudo apt-get update

sudo apt-get install build-essential libgsl23 libgslcblas0

wget https://www.tbi.univie.ac.at/RNA/download/ubuntu/ubuntu_18_04/viennarna_2.4.17-1_amd64.deb

sudo dpkg -i viennarna_2.4.17-1_amd64.deb
```

```
wget https://www.tbi.univie.ac.at/RNA/download/sourcecode/2_4_x/ViennaRNA-2.4.17.tar.gz

tar -zxvf ViennaRNA-2.4.17.tar.gz

cd ViennaRNA-2.4.17

./configure
```

La salida después de *./configure* no debe ser esta, ya que no se instalarán los módulos necesarios para Python2 y Python3, por tanto paramos y no hacemos *make* aún.

```
======================================
  ViennaRNA Package 2.4.17
======================================

Successfully configured with the following options:

Sub Packages
------------
 * Kinfold               : yes
 * RNAforester           : yes
 * Analyse{Dists,Seqs}   : no
 * RNAlocmin             : yes
 * Kinwalker             : no

Extra Libraries
---------------
 * Support Vector Machine   : yes
 * GNU Scientific Library   : no
 * GNU MPFR                 : no
 * JSON                     : yes

Features
--------
 * Boustrophedon          : yes
 * Use hash for NR Sampling : no
 * C11 features           : yes
 * TTY colors             : yes
 * Float Precision{PF}    : no
 * Deprecation Warnings   : no

Optimizations
-------------
 * Auto Vectorization     : yes
 * Explicit SIMD Extension : yes
 * Link Time Optimization  : yes
 * POSIX Threads          : yes
 * OpenMP                 : yes

Scripting Language Interfaces
-----------------------------
 * Perl 5                 : yes
 * Python 2               : no
 * Python 3               : no

Documentation
-------------
 * Reference Manual (PDF)  : yes
 * Reference Manual (HTML) : yes
 * Tutorial (PDF)          : yes
 * Tutorial (HTML)         : no

Unit Tests
----------
 * Executable Programs    : yes
 * C-Library              : no
 * Perl 5 Interface       : yes
 * Python 2 Interface     : no
 * Python 3 Interface     : no
```

```
MacOS X
-------
 * Universal Binary       : no
 * Installer              : no
 * SDK                    : custom

Install Directories
-------------------
 * Executables            : /usr/local/bin
 * Libraries              : /usr/local/lib
 * Header files           : /usr/local/include
 * Extra Data             : /usr/local/share
 * Man pages              : /usr/local/share/man
 * Documentation          : /usr/local/share/doc/ViennaRNA
   (HTML)                 : /usr/local/share/doc/ViennaRNA/html
   (PDF)                  : /usr/local/share/doc/ViennaRNA
 * Perl5 Interface        :
   (binaries)             : /usr/local/lib/x86_64-linux-gnu/perl/5.26.1
   (scripts)              : /usr/local/share/perl/5.26.1
 * Python2 Interface      : Not to be installed python2 executable missing
   (binaries)             :
   (scripts)              :
 * Python3 Interface      : Not to be installed Can't import distutils.sysconfig
   (binaries)             :
   (scripts)              :

You can run 'make', 'make check', and 'make install' now!
```

Por lo tanto, ejecutamos las siguientes instrucciones:

Instalamos Python 2.

```
Sudo apt-get install python
```

Instalamos NUMPY necesario en MCTS-RNA.

```
Sudo apt install python-numpy
```

Librerias para Python 2

```
sudo apt-get install python-dev
```

Librerias para Python 3

```
sudo apt-get install python3-dev
```

```
./configure
```

Ahora la salida de *./configure*, nos muestra que si van a ser instalados dichos módulos.

```
  ViennaRNA Package 2.4.17
------------------------------------------------------------------------

Successfully configured with the following options:

Sub Packages
------------------------------------------------------------------------
 * Kinfold               : yes
 * RNAforester           : yes
 * Analyse{Dists,Seqs}   : no
 * RNAlocmin             : yes
 * Kinwalker             : no

Extra Libraries
------------------------------------------------------------------------
 * Support Vector Machine   : yes
 * GNU Scientific Library   : no
```

```
 * GNU MPFR              : no
 * JSON                  : yes

Features
-------------------------------------------------------------------------
 * Boustrophedon         : yes
 * Use hash for NR Sampling : no
 * C11 features          : yes
 * TTY colors            : yes
 * Float Precision(PF}   : no
 * Deprecation Warnings  : no

Optimizations
-------------------------------------------------------------------------
 * Auto Vectorization    : yes
 * Explicit SIMD Extension : yes
 * Link Time Optimization : yes
 * POSIX Threads         : yes
 * OpenMP                : yes

Scripting Language Interfaces
-------------------------------------------------------------------------
 * Perl 5                : yes
 * Python 2              : yes
 * Python 3              : yes

Documentation
-------------------------------------------------------------------------
 * Reference Manual (PDF) : yes
 * Reference Manual (HTML) : yes
 * Tutorial (PDF)        : yes
 * Tutorial (HTML)       : no

Unit Tests
-------------------------------------------------------------------------
 * Executable Programs   : yes
 * C-Library             : no
 * Perl 5 Interface      : yes
 * Python 2 Interface    : yes
 * Python 3 Interface    : yes

MacOS X
-------------------------------------------------------------------------
 * Universal Binary      : no
 * Installer             : no
 * SDK                   : custom

Install Directories
-------------------------------------------------------------------------
 * Executables           : /usr/local/bin
 * Libraries             : /usr/local/lib
 * Header files          : /usr/local/include
 * Extra Data            : /usr/local/share
 * Man pages             : /usr/local/share/man
 * Documentation         : /usr/local/share/doc/ViennaRNA
   (HTML)                : /usr/local/share/doc/ViennaRNA/html
   (PDF)                 : /usr/local/share/doc/ViennaRNA
 * Perl5 Interface       :
   (binaries)            : /usr/local/lib/x86_64-linux-gnu/perl/5.26.1
   (scripts)             : /usr/local/share/perl/5.26.1
 * Python2 Interface     :
   (binaries)            : /usr/local/lib/python2.7/site-packages
   (scripts)             : /usr/local/lib/python2.7/site-packages
 * Python3 Interface     :
   (binaries)            : /usr/local/lib/python3.6/site-packages
   (scripts)             : /usr/local/lib/python3.6/site-packages

You can run 'make', 'make check', and 'make install' now!
```

Por lo tanto, procedemos con los últimos pasos, los cuales tardan bastante tiempo en ejecutarse.

```
make
sudo make install
```

Por último, hay que arreglar un problema con el directorio */usr/local/lib/python2.7/site-packages* , ya que por defecto se busca *en /usr/local/lib/python2.7/dist-packages* , dando esto errores al ejecutar el código.

La solución es modificar la variable de entorno PYTHONPATH en el archivo *.bashrc* del usuario.

```
echo   "PYTHONPATH="/usr/local/lib/python2.7/site-packages/":"${PYTHONPATH}"
>> $HOME/.bashrc

echo "export PYTHONPATH" >> $HOME/.bashrc

exit
```

Si queremos probar que todo es correcto, instalamos git, y posteriormente descargamos y probamos el código.

```
git clone https://github.com/tsudalab/MCTS-RNA.git
cd MCTS-RNA
python                          MCTS-RNA.py                          -s
"...(((((.........)))))........(((((((.....)))))))(((((.......)))
)))............(((((..(((((..((..((.(((((.(((((.......)))))).)))))..
.))....))))))))))))" -GC 0.75 -d 0.01 -pk 0
```

4.5 Traducción de Python a C++

La conversión del código se ha intentado llevar a cabo de la forma en la que el resultado respecto al aspecto del código sea lo más parecido posible, pero intentando llevar nuestros objetivos a cabo, tratando de hacer los cambios estrictamente necesarios para poder pasar un código Python a C/C++.

Una vez que se tenía el código Python, se comenzó a leer y a intentar entender desde el principio todo lo que hacía. Debido a que es un código extenso que no está comentado, exceptuando alguna línea, esto dificultaba su comprensión; después de 2-3 lecturas, se podía observar cómo este código podía separarse en unas partes claramente diferenciadas, que eran parte del *main* junto con métodos que se usan en la carga de datos; parte de métodos de cálculo, siendo estos métodos extensos y

complejos, incluyendo el método más importante, el método de cálculo del árbol de búsqueda de Monte Carlo. Y, por último, la parte de las clases, en la que nos encontraríamos con dos, la clase *nodo*, la cual forma un nodo de nuestro árbol, y la clase *RNAestructura*, en la que vamos a almacenar todos los datos necesarios para la estructura de la molécula de ARN que va a ser guardada en cada nodo junto con sus métodos de simulación tratamiento de los datos propios.

Comenzando desde el principio, nuestro método *main*, en él, una vez lanzada la ejecución, obtenemos los parámetros de entrada, realizándoles unas pequeñas pruebas para comprobar que efectivamente se han introducido de manera correcta. Estas pruebas junto con el método de obtener los parámetros, en Python, no los encontramos, debido a que se realizan de forma automática gracias a la ayuda del lenguaje, con lo que se puede estar seguro de que se han introducido los datos correctamente. Una vez tenemos los datos de entrada/parámetros en nuestras variables, comienza el tratamiento de los datos junto con sus distintas comprobaciones.

Primero se realiza la separación de *puntos* y de *paréntesis*, estos, debido al lenguaje, se ha decidido separarlos en dos vectores, un *VectorPuntos* y un *VectorParentesis*, el vector de puntos sería un vector de enteros en el que se almacena en qué posición se encuentra el punto en nuestra cadena de caracteres punto-paréntesis, y lo mismo para vector paréntesis, pero en este caso se ha tomado la decisión de crear un *struct* con un par de valores enteros para indicar el par de apertura-cierre de los mismos.

Luego, formulamos una idea con posibles bases respecto a los pares de paréntesis, es decir, si tenemos *()*, tendríamos una "idea" de un posible par de bases *AU*, (otras posibles bases serían *CG, GC, UA*). Una vez tenemos estos pares de bases, de longitud igual a la longitud del número de pares de bases, pasaríamos a realizar lo mismo, pero con los puntos, siendo las bases posibles *A, G, C,* y *U*.

Posteriormente pasaríamos a realizar ciertas operaciones con los datos, como separar este vector de pares de paréntesis que teníamos, en un solo vector de enteros, ordenarlo, separar la idea de pares de bases en un vector único de tipo *string*…

Una vez realizadas estas operaciones, se llega al final de nuestro método *main* en el que lanzamos los métodos *UCTRNA* o *UCTRNAnoGC* respectivamente según el

parámetro *GC*, se encuentre este entre los valores 0 y 1 inclusives. Estos métodos lanzarán a su vez el algoritmo del árbol de Monte Carlo.

Una vez nos encontramos en estos métodos, en ellos creamos una nueva variable *state* de tipo RNAestructura, la clase que tenemos para almacenar todos los datos y realizar las operaciones referentes a la estructura de la molécula de ARN.

En esta clase como atributos tendremos: 4 vectores de tipo *string*, uno de ellos vacío, en el que iremos insertando bases, y los otros 3 ya llenos de bases tipo, con los que iremos trabajando extrayendo las bases de manera aleatoria en algoritmos posteriores de la clase. Además, tendremos un vector, en este caso, formado por un *struct* con 3 enteros y un *string*. Esta idea surge debido a que en el código Python, nos encontramos con que une los vectores de enteros que indican la posición de los puntos y de los pares apertura-cierre de paréntesis. Como esto en C/C++ no se puede hacer, la solución óptima encontrada fue esa, y con esto tendríamos en el *struct* un entero dedicado al punto, los otros dos a apertura y cierre de los paréntesis, y el *string* dedicado a la base, ya que también, a lo largo de la ejecución, en este vector en Python, se va cambiando, por ejemplo, la posición *X* del vector, que nos indicaría que un punto se encuentra en la posición *Y* por una base, teniendo así una mezcla de enteros, pares de enteros y bases; cosa que como mencionamos, en C/C++ no se puede. Así mismo, para diferenciar, todo está inicializado a -1, y el *string* a vacío, con lo que se han desarrollado uno métodos que indican si en esa posición de nuestro vector nos encontramos ante un punto, un par de paréntesis, o una base para cuando a lo largo de la ejecución tenemos que ir haciendo estas diferenciaciones para trabajar con una cosa u otra.

Otros métodos mencionables de esta clase, son los métodos de simulación, que de forma aleatoria y según unos parámetros, vamos eligiendo bases formando una estructura aleatoria cada vez; o, un método de clonación de todos los datos de la clase, a la clase del mismo tipo introducida por parámetro.

Volviendo a nuestro método *UCTRNA*, lanzaríamos el método *MCTS* que lleva a cabo la ejecución del árbol de Monte Carlo. Una vez saliésemos de este método, los resultados se mostrarían y posteriormente se acabaría la ejecución.

En *MCTS*, trabajamos con **nodos**, nuestra otra clase. En ella, guardamos un atributo tipo *RNAestructura*, es decir, tendremos una molécula de ARN con la que trabajar y simular por nodo, además de otros atributos básicos tipo nodos padre, nodos hijos, visitas, posiciones y búsquedas no probadas…

Algunos métodos de esta clase son los de añadir nodo, seleccionar un nodo, actualizar un nodo…

El grueso del método de *MCTS*, trabaja con un bucle controlado por tiempo, en el que mientras estemos por debajo del tiempo indicado, este bucle seguirá ejecutando mientras no se encuentre una estructura ARN deseada o no se llegue a los 10 minutos indicados desde que se lanza la ejecución. En este método llevamos a cabo todas las operaciones complejas junto con el uso de las operaciones, mencionadas en otras partes de este documento, relacionadas con el paquete *ViennaRNA.*

Tendremos otro método *MCTS*, como cabía esperar, llamado *MCTSnoGC*, si es el caso de que se ha lanzado una ejecución, como se mencionaba anteriormente, en la que el parámetro *GC* no se encuentra en 0 y 1 inclusives, lanzando *UCTSnoGC* y posteriormente este método.

Para finalizar este apartado, mencionar que se ha insertado tanto en el proyecto C/C++ como en Python un generador propio de números aleatorios con el fin de que ambos usen un generador propio diferente, por lo que, aun usando la misma semilla, los resultados eran diferentes.

Siguiendo los pasos de la página del blog de studywolf (*Using the Same Random Number Generator in C++ and Python | Studywolf*, n.d.) , se ha conseguido llevar la implementación de forma satisfactoria en ambos proyectos de un mismo generador números de aleatorios. Gracias a esto, ahora podemos ejecutar y realizar pruebas usando la misma semilla, obteniendo así, exactamente los mismos resultados, ayudándonos a cotejar soluciones, resultados de pruebas de velocidad y de número de ejecuciones, además de la ayuda a la hora de encontrar y corregir errores.

4.6 Metodología.

La metodología para el desarrollo del código C/C++ ha seguido el flujo de los datos de entrada. De esta forma, se podía ir creando el código y observando todas las operaciones que se iban realizando sobre ellos, además de ir realizando las pruebas pertinentes para asegurarnos de que todo iba según lo descrito, y que los resultados iban siendo los esperados y observados en las ejecuciones del código Python.

4.7 Organización del Código C++.

En este apartado vamos a llevar a cabo la explicación básica sobre cómo están organizados los ficheros de código del proyecto (Ilustración 19).

En primer lugar, tenemos el archivo **MCTS_RNA** donde se encuentra nuestro método *main;* en este archivo tenemos también la declaración de las variables principales usadas de forma global, como pueden ser un vectorPuntos, que no es más que un vector de enteros para almacenar la posición de los puntos en la variable *-s* de entrada con notación punto-paréntesis o variables para almacenar los parámetros de entrada. Posteriormente en el *main* tenemos el tratamiento de las variables de entrada, junto con la ejecución del propio algoritmo.

Aparte, tendríamos el par de ficheros **MetodosCarga.cpp/.h** en los que se encuentran todos los métodos usados en el *main* para la carga de datos y su tratamiento. Métodos como por ejemplo *ObtenerParametros* con el que leemos los parámetros introducidos en la ejecución y se les hace unas breves pruebas de comprobación de que están bien introducidos; otro método, por ejemplo, *SepararParentesis,* con el que cogemos los pares de paréntesis apertura-cierre de la secuencia *-s* introducida y los agrupa en un vector propio creado de *parParentesis*, que no es más que un *struct* de dos enteros que indican la posición apertura-cierre en la secuencia.

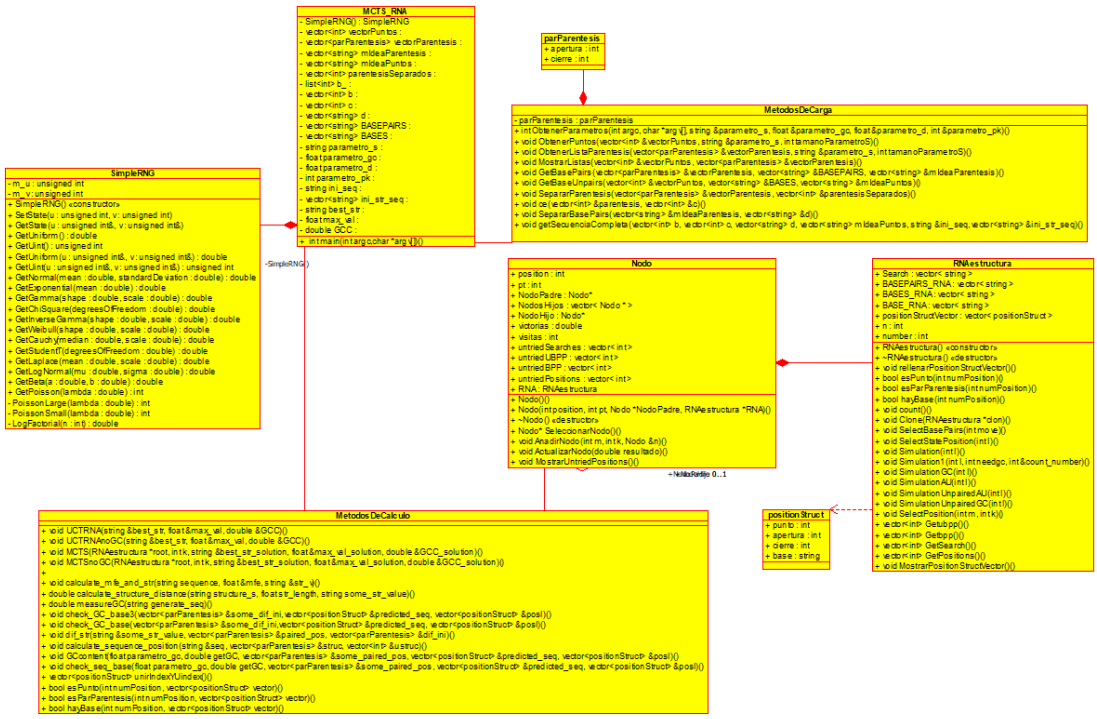

Ilustración 19: Diagrama de clases.

Por otra parte, tenemos las clases **Nodo.cpp/.h** y **RNAestructura.cpp/.h,** con las que tratamos de replicar los datos que necesitamos para un nodo de nuestro árbol de búsqueda de Monte Carlo, donde uno de sus atributos se va a encargar de almacenar un "RNAestructura", una idea de una molécula ARN la cuál va a ser desarrollada en cada ejecución del nodo. Esta última clase posee los distintos métodos para realizar las simulaciones, o por ejemplo, modificar la secuencia que estamos almacenando.

En último lugar, nos encontramos con **MetodosDeCalculo.cpp/.h**, los ficheros más extensos, ya que en ellos se encuentran los algoritmos más complejos de ejecución, como pueden ser el propio árbol de Monte Carlo, *MCTS*, y su variante *MCTSnoGC*, o métodos importantes como *calculate_mfe_and_str*, en el que usamos las funciones *fold* del paquete *ViennaRNA* para calcular la energía mínima libre (MFE) y su correspondiente estructura secundaria. Se puede consular la implementación completa realizada en C/C++ en el Anexo B.

5. RESULTADOS Y DISCUSIÓN

5.1 Experimentación.

Para llevar a cabo los experimentos, nos hemos apoyado en el uso de un nodo del clúster del grupo de investigación ARCO (arcohpc.unex.es). Concretamente, el nodo utilizado dispone de 64 GB de memoria *RAM* así como de cuatro procesadores *AMD Opteron(tm) Processor* 6174, cada uno de los cuales dispone de 12 *núcleos*, que trabajan a una frecuencia de 2.2 GHz. Dicho nodo tiene instalado Ubuntu 18.04.2 LTS y *gcc 8.4.0*. Gracias a esto tenía disponible 48 núcleos en los que poder lanzar sin problema, y con potencia suficiente, todas las pruebas pertinentes, ya que era necesaria esta potencia para poder ver el rendimiento real y potencial del proyecto.

Las pruebas que se van a realizar son dos, la primera teniendo en cuenta un límite de tiempo impuesto en los bucles de cada árbol de **600 segundos**, ya mencionado en otros apartados del documento. La segunda, modificando estos tiempos límite por defecto a **6000 segundos**, para comprobar si aquellas estructuras en las que no se obtiene una secuencia válida eran finalmente resueltas o no.

Para estas pruebas vamos a usar la familia de estructuras de *Rfam* [TANEDA11], que se trata de una base de datos que contiene información sobre las familias de ARN, siendo de acceso libre y originada en el *Wellcome Trust Sanger Institute*.

Vamos a usar 29 conjuntos de datos de entrada de esta familia, sus nombres y sus longitudes (entre paréntesis) son los siguientes: RF00001.121.ss (118), RF00002.2.ss (152), RF00003.94.ss (162), RF00004.126.ss (194), RF00005.1.ss (75), RF00006.1.ss (90), RF00007.20.ss (155), RF00008.11.ss (55), RF00009.115.ss (349), RF00010.253.ss (358), RF00011.18.ss (383), RF00012.15.ss (216), RF00013.139.ss (186), RF00014.2.ss (88), RF00015.101.ss (141), RF00016.15.ss (130), RF00017.90.ss (302), RF00018.2.ss (361), RF00019.115.ss (84), RF00020.107.ss (120), RF00021.10.ss (119), RF00022.1.ss (149), RF00024.16.ss (452), RF00025.12.ss (211), RF00026.1.ss (103), RF00027.7.ss (80), RF00028.1.ss (345), RF00029.107.ss (74), RF00030.30.ss (341).

Para cada conjunto de datos, lanzaremos 31 ejecuciones con semillas diferentes entre ellas, pero iguales entre versiones, para poder comparar resultados y llegar a unas conclusiones. Esto es posible gracias a que, como se mencionó anteriormente, ambos proyectos utilizan el mismo generador de números aleatorios para poder realizar estas ejecuciones obteniendo exactamente los mismos resultados y secuencias a lo largo de las mismas, centrándonos así en el tiempo, número de iteraciones, y victorias que obtenemos, es decir, si en el rango de tiempo impuesto se ha encontrado una solución.

En las primeras pruebas, para dar una aproximación del tiempo invertido en las ejecuciones, debido a que el límite es el tiempo y cada conjunto de datos de entrada máximo va a estar 10 minutos (600 segundos), esto nos eleva el tiempo de ejecución de cada versión a cerca de las 5h, al tener que ejecutar cada conjunto 31 veces. Posteriormente habría que lanzar el siguiente proyecto y repetir el mismo proceso. Ídem para la segunda prueba, pero, con 6000 segundos, llegando a estar unas 30h ejecutando en total.

Una vez se han obtenido todos los ficheros de salida de las ejecuciones, estos, mediante un script *bash* se han sacado todos los resultados pasándolos a hojas de cálculo posibilitando la interpretación y obtención de medias de forma sencilla, rellenando así las tablas del siguiente apartado.

5.2 Resultados.

Rfam	**Python**			**C/C++**		
Secuencia	Tiempo	Iter.	Vic.	Tiempo	Iter.	Vic.
RF00001.121.ss	462,5	543,2	14	437,3	572,4	14
RF00002.2.ss	600	437	0	600	464,1	0
RF00003.94.ss	600	340	0	600	364,4	0
RF00004.126.ss	199,6	79,1	29	188,4	80	30
RF00005.1.ss	0,5	3,2	31	0,16	3,2	31
RF00006.1.ss	85,94	213,9	31	75,6	213,9	31
RF00007.20.ss	107,1	65,5	30	97,6	66,5	30
RF00008.11.ss	0,6	2,5	31	0,03	2,5	31
RF00009.115.ss	600	70,7	0	600	76,3	0
RF00010.253.ss	600	56,3	0	600	60,8	0
RF00011.18.ss	600	51,2	0	600	55,6	0
RF00012.15.ss	552,4	173	4	550,3	184,8	7
RF00013.139.ss	33	15,7	31	31,2	15,7	31
RF00014.2.ss	0,02	1,4	31	0,01	1,4	31
RF00015.101.ss	578,5	487,5	2	577,2	523,6	2
RF00016.15.ss	600	677	0	600	735	0
RF00017.90.ss	9,08	1,78	31	6,5	1,78	31
RF00018.2.ss	600	64,8	0	600	70	0
RF00019.115.ss	2,2	6,3	31	1,6	6,3	31
RF00020.107.ss	600	642,6	0	600	690,3	0
RF00021.10.ss	1	2	31	0,12	2	31
RF00022.1.ss	202	144,3	27	194,9	148	28
RF00024.16.ss	600	35,1	0	600	38,5	0
RF00025.12.ss	600	206,4	0	600	221	0
RF00026.1.ss	74,1	121,7	31	51,13	121,7	31
RF00027.7.ss	1,3	11,5	31	0,8	11,5	31
RF00028.1.ss	600	71,5	0	600	77,3	0
RF00029.107.ss	75,93	270,4	31	65,26	270,4	31
RF00030.30.ss	600	72,3	0	600	77,9	0

Tabla 1: Resultados de la primera prueba

Observando los datos obtenidos en esta primera prueba, podemos ver una mejora general en el rendimiento del código, tanto en tiempo como en número de iteraciones del bucle del árbol de *Monte Carlo*.

Donde más notamos la mejoría es en el número de iteraciones del bucle, siendo esta la operación más costosa al tener que realizar todas las operaciones y recorrer el árbol.

En las ejecuciones que alcanzamos el tiempo límite, señaladas con un color más oscuro, es donde podemos observar dicha mejora de rendimiento en el número de iteraciones.

Por otra parte, tenemos las ejecuciones que no han llegado a alcanzar el tiempo límite, pero sus victorias son 31, es decir, las 31 ejecuciones de la secuencia han encontrado una solución. En este caso tenemos que el número de iteraciones va a ser exactamente igual, al ser ambos proyectos iguales y encontrar la solución en el mismo número de operaciones, la diferencia la marca el tiempo, donde podemos observar con de nuevo que la versión *C/C++* encuentra dichas soluciones en un tiempo menor.

Finalmente, el último caso que nos encontramos, sería no haber llegado al tiempo límite, y no haber encontrado las 31 soluciones en las 31 ejecuciones. En estos casos, por norma general, tenemos como la versión *C/C++* es mejor en cuanto al número de iteraciones y tiempo. Casos cómo RF00001.121.ss, que consiguen 14 victorias ambos, pero tienen ejecuciones que agotan el tiempo límite y otras que han hallado la solución en tiempo. Las soluciones que han agotado el tiempo es donde se marca esa diferencia de iteraciones que nos encontramos, aunque mínima en algunos casos, ya que de las 14 encontradas, esas sí tienen las mismas iteraciones, pero las otras 17, al ejecutar *C/C++* más veces el bucle, acaban realizando más iteraciones.

En RF00004.126.ss, por ejemplo, la versión *C/C++* llega incluso a encontrar una solución más, 29 frente a las 30 de *Python*. *Python*, en una de las ejecuciones, para ser más exactos la ejecución_22, se ha quedado en 242 iteraciones en el momento que ha acabado su tiempo, los 600 segundos, mientras en *C/C++* en 574 segundos ha llegado a 249 iteraciones, encontrando así la solución y acabando. Es decir, al ser *Python* más lento, perderíamos esa solución.

Otro cambio significativo hallado, es que, al ejecutar más iteraciones la versión *C/C++*, en los conjuntos de datos en los que se alcanza el tiempo límite sin encontrar una solución tienen ligeramente mejores soluciones intermedias finales.

Rfam	**Python**			**C/C++**		
Secuencia	Tiempo	Iter.	Vic.	Tiempo	Iter.	Vic.
RF00001.121.ss	1961,2	2392,1	28	1848,3	2453	28
RF00002.2.ss	6000	4379,4	0	6000	4672,7	0
RF00003.94.ss	6000	3402	0	6000	3646,5	0
RF00004.126.ss	202,5	80,5	31	191,8	80,5	31
RF00005.1.ss	0,5	3,2	31	0,16	3,2	31
RF00006.1.ss	83	213,9	31	76,1	213,9	31
RF00007.20.ss	111	70,4	31	104,4	70,4	31
RF00008.11.ss	0,6	2,5	31	0,12	2,5	31
RF00009.115.ss	6000	703,5	0	6000	753,6	0
RF00010.253.ss	6000	560,2	0	6000	598,7	0
RF00011.18.ss	6000	509,2	0	6000	545	0
RF00012.15.ss	3544	1114	20	3452,8	1161,5	20
RF00013.139.ss	33,9	15,7	31	31,3	15,7	31
RF00014.2.ss	0,1	1,4	31	0,06	1,4	31
RF00015.101.ss	4515	3817,2	16	4380,1	4002,2	18
RF00016.15.ss	6000	6842,3	0	6000	7428,3	0
RF00017.90.ss	7,3	1,78	31	6,4	1,78	31
RF00018.2.ss	6000	645,6	0	6000	690,8	0
RF00019.115.ss	2,3	6,3	31	1,51	6,3	31
RF00020.107.ss	6000	6438,8	0	6000	6919,3	0
RF00021.10.ss	1,13	2	31	0,22	2	31
RF00022.1.ss	270,9	192,2	31	252	192,2	31
RF00024.16.ss	6000	348,9	0	6000	373,5	0
RF00025.12.ss	5847,9	2009,9	2	5831	2155,7	2
RF00026.1.ss	55,9	121,7	31	50,2	121,7	31
RF00027.7.ss	1,51	11,5	31	0,9	11,5	31
RF00028.1.ss	6000	712,5	0	6000	762,2	0
RF00029.107.ss	71,5	270,4	31	64,1	270,4	31
RF00030.30.ss	6000	719,4	0	6000	766,2	0

Tabla 2: Resultados de la segunda prueba.

En esta segunda prueba vemos cómo los resultados obtenidos siguen la misma dinámica que en la anterior. Por ejemplo, podemos observar cómo para el conjunto RF00015.101.ss, debido a este extra de tiempo conseguimos encontrar 16 y 18 soluciones en *Python* y en *C/C++* respectivamente, cosa que en la primera prueba nos quedábamos en ambas ejecuciones en 2 soluciones. Además, gracias a que la versión *C/C++* tiene mejor rendimiento, llegamos a encontrar ese par de soluciones más.

Por último, en el caso del conjunto RF00025.12.ss, en el que, mientras en la primera prueba se agotaba el tiempo, en esta segunda hemos conseguido encontrar, al menos, un par de soluciones.

6. Conclusiones.

Podemos concluir este trabajo con unos resultados a priori satisfactorios ya que habiendo hecho la traducción a *C/C++* respetando el código original de *Python*, haciendo los cambios necesarios para adaptar el código a este lenguaje sin habernos puesto en el desarrollo de optimizaciones, hemos conseguido una mejora en los resultados, además de un código con igual funcionamiento y soluciones obtenidas respecto a las ejecuciones.

Con esto, tendríamos una buena base para, en un futuro, trabajar en ella intentando optimizar el código todo lo posible para llevarlo a su máximo potencial o implementar en él otro tipo de funcionalidades futuras.

De forma más personal, ha sido el proyecto más grande en el que me he involucrado y trabajado yo solo de forma autónoma, cosa que, como se nos ha ido enseñando a lo largo de la carrera, me ha servido aún más para aprender a desenvolverme, buscar, investigar y aprender por mi cuenta ayudándome a ser más resolutivo con mi trabajo.

Ha sido un proyecto en el que además de poner en práctica mis conocimientos adquiridos a lo largo de estos años, ha servido para mejorarlos o adquirir nuevos, ya que, por ejemplo, de *Python*, mis conocimientos eran “escasos”, pero a base de como mencionaba, buscar y mostrar interés en aprender, he llegado a encontrar ciertos errores en el código, o aprendido a plantear y pensar las cosas de otra manera. Llegando a añadir un mismo generador de números aleatorios en ambos proyectos para obtener iguales resultados, tocar y modificar el código *Python* para pruebas y ver como trabajaba, o simplemente aprender y poner en práctica cosas nuevas en *C/C++*.

BIBLIOGRAFÍA

Ácido nucleico | NHGRI. (n.d.). Retrieved December 8, 2021, from https://www.genome.gov/es/genetics-glossary/acido-nucleico

ADN y ARN concepto, diferencias y funciones | VIU. (n.d.). Retrieved December 8, 2021, from https://www.universidadviu.com/es/actualidad/nuestros-expertos/adn-y-arn-concepto-diferencias-y-funciones

Árbol de búsqueda Monte Carlo - FdIwiki ELP. (n.d.). Retrieved December 8, 2021, from https://wikis.fdi.ucm.es/ELP/%C3%81rbol_de_b%C3%BAsqueda_Monte_Carlo

Búsqueda de árboles de Montecarlo - programador clic. (n.d.). Retrieved December 8, 2021, from https://programmerclick.com/article/95711486601/

Download Ubuntu Desktop | Download | Ubuntu. (n.d.). Retrieved December 3, 2021, from https://ubuntu.com/download/desktop

El sistema operativo GNU y el movimiento del software libre. (n.d.). Retrieved December 3, 2021, from https://www.gnu.org/

Hofacker, I. L., Fontana, W., Stadler, P. F., Bonhoeffer, L. S., Tacker, M., & Schuster, P. (1994). Fast folding and comparison of RNA secondary structures. *Undefined*, *125*(2), 167–188. https://doi.org/10.1007/BF00818163

Linux.org. (n.d.). Retrieved December 3, 2021, from https://www.linux.org/

Lorenz, R., Bernhart, S. H., Höner zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P. F., & Hofacker, I. L. (2011). ViennaRNA Package 2.0. *Algorithms for Molecular Biology*, *6*(1). https://doi.org/10.1186/1748-7188-6-26

Monte Carlo Tree Search - beginners guide int8.io. (n.d.). Retrieved December 8, 2021, from https://int8.io/monte-carlo-tree-search-beginners-guide/

Monte Carlo Tree Search: An Introduction | by Benjamin Wang | Towards Data Science. (n.d.). Retrieved December 8, 2021, from https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168

Rufus - Cree unidades USB arrancables fácilmente. (n.d.). Retrieved December 3, 2021, from https://rufus.ie/es/

TBI - ViennaRNA Package 2. (n.d.). Retrieved December 8, 2021, from https://www.tbi.univie.ac.at/RNA/

Yang, X., Yoshizoe, K., Taneda, A., & Tsuda, K. (2017). RNA inverse folding using Monte Carlo tree search. *BMC Bioinformatics*, *18*(1), 1–12. https://doi.org/10.1186/S12859-017-1882-7/FIGURES/10

Using the same random number generator in C++ and Python | studywolf. (n.d.). Retrieved January 24, 2022, from https://studywolf.wordpress.com/2012/10/02/a-common-random-number-generator-for-c-and-python/

GitHub - tsudalab/MCTS-RNA: MCTS-RNA is a computational tool for solving RNA inverse folding problem with controlling the GC-content of the RNA sequence very precisely. (n.d.). Retrieved February 1, 2022, from https://github.com/tsudalab/MCTS-RNA

Enabling Open Innovation & Collaboration | The Eclipse Foundation. (n.d.). Retrieved February 1, 2022, from https://www.eclipse.org/

Qué es C++: Características y aplicaciones | OpenWebinars. (n.d.). Retrieved February 1, 2022, from https://openwebinars.net/blog/que-es-cpp/

Qué es C: Características y sintaxis | OpenWebinars. (n.d.). Retrieved February 1, 2022, from https://openwebinars.net/blog/que-es-c/

AlphaGo | DeepMind. (n.d.). Retrieved February 4, 2022, from https://deepmind.com/research/case-studies/alphago-the-story-so-far

Base stacking - Proteopedia, life in 3D. (n.d.). Retrieved February 5, 2022, from https://proteopedia.org/wiki/index.php/Base_stacking

hairpin loop (mRNA) | Learn Science at Scitable. (n.d.). Retrieved February 5, 2022, from https://www.nature.com/scitable/definition/hairpin-loop-mrna-314/

Multiloop | EteRNA Wiki | Fandom. (n.d.). Retrieved February 5, 2022, from https://eternagame.fandom.com/wiki/Multiloop

Stability of single-nucleotide bulge loops embedded in a GAAA RNA hairpin stem. (n.d.). Retrieved February 5, 2022, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3312567/

The Influence of Symmetric Internal Loops on the Flexibility of RNA - ScienceDirect. (n.d.). Retrieved February 5, 2022, from https://www.sciencedirect.com/science/article/abs/pii/S002228369690162X

Rubio-Largo, Á., Vanneschi, L., Castelli, M., & Vega-Rodríguez, M. A. (2019). Multiobjective Metaheuristic to Design RNA Sequences. *IEEE Transactions on Evolutionary Computation*, *23*(1), 156–169. https://doi.org/10.1109/TEVC.2018.2844116

García-Martín, J. A., Clote, P., & Dotu, I. (2013). RNAiFOLD: A constraint programming algorithm for RNA inverse folding and molecular design. Journal of Bioinformatics and Computational Biology, 11(2), 1350001. https://doi.org/10.1142/s0219720013500017

García-Martin, J. A., Dotu, I., & Clote, P. (2015). RNAiFold 2.0: a web server and software to design custom and Rfam-based RNA molecules. Nucleic Acids Research, 43(W1), W513-W521. https://doi.org/10.1093/nar/gkv460

Taneda, A. (2010). MODENA: a multi-objective RNA inverse folding. Advances and Applications in Bioinformatics and Chemistry, 1. https://doi.org/10.2147/aabc.s14335

Taneda, A. (2012). Multi-objective genetic algorithm for pseudoknotted RNA sequence design. *Frontiers in Genetics*, *3*(APR), 36. https://doi.org/10.3389/fgene.2012.00036

Taneda, A. (2015). Multi-objective optimization for RNA design with multiple target secondary structures. *BMC Bioinformatics*, *16*(1). https://doi.org/10.1186/s12859-015-0706-x

Kleinkauf, R., Houwaart, T., Backofen, R., & Mann, M. (2015). antaRNA--Multi-objective inverse folding of pseudoknot RNA using ant-colony optimization. *BMC Bioinformatics*, *16*(1). https://doi.org/10.1186/S12859-015-0815-6

Reinharz, V., Ponty, Y., & Waldispühl, J. (2013). A weighted sampling algorithm for the design of RNA sequences with targeted secondary structure and nucleotide

distribution. *Bioinformatics*, *29*(13), i308–i315. https://doi.org/10.1093/BIOINFORMATICS/BTT217

ANEXOS

ANEXO A

Código Python completo con generador de números aleatorios y las correcciones mencionadas implementadas.

Para hacerlo funcionar es necesario tener en el mismo directorio el archivo .o que se puede encontrar en la página web de la bibliografía *Using the same random number generator in C++ and Python*. Esto es mencionado al final del apartado 4.5 Traducción de Python a C++.

```
from subprocess import Popen, PIPE
from math import *
import random
import numpy as np
import RNA
from copy import deepcopy
from types import IntType, ListType, TupleType, StringTypes
import itertools
import time
import math
import argparse
import subprocess
import sys
from SimpleRNG import pySimpleRNG


class RNAstructure:
    def __init__(self):
        self.search = [0,0,0,0,0,0,0,0,0,0]
        self.basepairs=["AU", "CG", "GC", "UA","GU","UG"]
        self.bases=["A","C","G","U","AU", "CG", "GC", "UA","GU","UG"]
        self.base=["A","C","G","U"]
        self.position=str_uindex+str_index
        self.n= len(str_uindex+str_index)

    def count(self):
        self.number+=1

    def Clone(self):

        st = RNAstructure()
        st.search = self.search[:]
        st.basepairs = self.basepairs[:]
        st.position= self.position[:]
        return st

    def Rewards(self,k):
        #copy_unpairedposition=list(unpairedposition)
        #copy_bppused=list(bppused)
        if k > len(str_uindex)-1:
            posbasep=self.position[len(str_uindex):self.n]
```

```
            posbase=self.position[0:len(str_uindex)]

            e=list(itertools.chain(*posbasep))
            for i in range(len(a)):
                posbase.insert(b[i],e[c[i]])
            mutated_s= ''.join(map(str, posbase))
            mutated_str1=RNA.fold(mutated_s)
            mutated_str=mutated_str1[0]

            d=0.0
            g=0.0
            n=len(s)
            for i in range(len(s)):
                if mutated_str[i]!=s[i]:
                    d=d+1
            g=(n-d)/n
            if g==1.0:
                solution.append(mutated_s)
                return g
            else:
                return g

        if k <= len(str_uindex)-1:
            posbasep=self.position[len(str_uindex):self.n]
            posbase=self.position[0:len(str_uindex)]

            e=list(itertools.chain(*posbasep))
            for i in range(len(a)):
                posbase.insert(b[i],e[c[i]])
            mutated_s= ''.join(map(str, posbase))
            mutated_str1=RNA.fold(mutated_s)
            mutated_str=mutated_str1[0]
            d=0.0
            g=0.0
            n=len(s)
            for i in range(len(s)):
                if mutated_str[i]!=s[i]:
                    d=d+1
            g=(n-d)/n
            if g==1.0:
                solution.append(mutated_s)
                return g
            else:
                return g


    def SelectBasePairs(self,move):
        self.search[move]=self.basepairs[move]

    def Selectstateposition(self,l):
        if l > len(str_uindex)-1:
            self.position[l]=midea[l-len(str_uindex)]
        else:
            self.position[l]=copy_str_uindex[l]

    def Simulation(self,l):
        if l>len(str_uindex)-1:
          indice = sRNG.GetUint()%len(BASEPAIRS)
          seleccion=BASEPAIRS[indice]
            self.position[l]=seleccion
```

```
        else:
          indice = sRNG.GetUint()%len(bases)
          seleccion=bases[indice]
            self.position[l]=seleccion

    def simulation1(self, l,needgc,count_number):
        if l>len(str_uindex)-1 and count_number<=needgc:
          indice = sRNG.GetUint()%len(["CG","GC"])
          seleccion=["CG","GC"][indice]
            self.position[l]=seleccion
            count_number=count_number+1
        if l>len(str_uindex)-1 and count_number>needgc:
          indice = sRNG.GetUint()%len(["AU","UA"])
          seleccion=["AU","UA"][indice]
            self.position[l]=seleccion
        if l<=len(str_uindex)-1:
          indice = sRNG.GetUint()%len(["A","U"])
          seleccion=["A","U"][indice]
            self.position[l]=seleccion

    def simulationGC(self,l):
      indice = sRNG.GetUint()%len(["CG","GC"])
      seleccion=["CG","GC"][indice]
        self.position[l]=seleccion

    def simulationAU(self,l):
      indice = sRNG.GetUint()%len(["AU","UA"])
      seleccion = ["AU","UA"][indice]
        self.position[l]=seleccion

    def simulationunpairedAU(self,l):
      indice = sRNG.GetUint()%len(["A","U"])
      seleccion=["A","U"][indice]
        self.position[l]=seleccion

    def simulationunpairedGC(self,l):
      indice = sRNG.GetUint()%len(["G","C"])
      seleccion=["G","C"][indice]
        self.position[l]=seleccion

    def SelectPosition(self,m,k):
        self.position[k]=self.bases[m]

    def Getubpp(self):
        return [i for i in np.arange(0,4) if self.search[i] not in
["A","U","C","G"]]

    def Getbpp(self):
        return [i for i in np.arange(4,10) if self.search[i] not in
["AU", "CG", "GC", "UA","GU","UG"]]

    def GetSearch(self):
        return [i for i in range(len(self.search)) if self.search[i]
not in ["A","U","C","G","AU", "CG", "GC", "UA","GU","UG"]]

    def GetPositions(self):
        return[i    for    i    in    range(len(self.position))    if
self.position[i]   not   in   ["A","U","C","G","AU",   "CG",   "GC",
"UA","GU","UG"]]
```

```
class Node:
    def __init__(self, position = None, pt = None , parent = None,
state = None):
        self.position = position
        self.pt = pt
        self.parentNode = parent
        self.childNodes = []
        self.child=None
        self.wins = 0
        self.visits = 0
        self.untriedSearches = state.GetSearch()
        self.untriedubpp=state.Getubpp()
        self.untriedbpp=state.Getbpp()
        self.untriedPositions=state.GetPositions()

    def Selectnode(self):
        s = sorted(self.childNodes, key = lambda c: c.wins/c.visits +
0.1*sqrt(2*log(self.visits)/c.visits))[-1]
        return s

    def Addnode(self, m, k, s):

        n = Node(position = m, pt=k, parent = self, state = s)
        if k in self.untriedPositions:
            self.untriedPositions.remove(k)
        self.childNodes.append(n)
        self.child=n

        return n

    def Update(self, result):
        self.visits += 1
        self.wins += result



def MCTS(root, k, verbose = False):
    running_time=time.time()
    #out_time=running_time+60*10
    out_time=running_time+600*10
    rootnode = Node(state = root)
    state = root.Clone() # but this state is the state of the
initialization .  too important !!!

    contador = 1;

    while time.time()<=out_time:

      print
      print "Ejecucion MCTS numero: " + str(contador)
      contador = contador+1

        node = rootnode # important !    this node is different with
state / node is the tree node
        state = root.Clone() # but this state is the state of the
initialization .  too important !!!
        posi=[]
        posl=[]
        poslalpha=[]
```

```
        need=[]
        count_number=0
        count_number1=0
        pa=[]
        upa=[]

        while node.untriedubpp == [] or node.untriedbpp==[]:
            node = node.Selectnode()
            state.SelectPosition(node.position,node.pt)

        if node.untriedPositions != []:
            if k > len(str_uindex)-1:
                if len(node.untriedbpp)==6:
                indice = sRNG.GetUint()%len(node.untriedPositions)
                seleccion = node.untriedPositions[indice]
                    k = seleccion
            else:
                if len(node.untriedubpp)==4:
                indice = sRNG.GetUint()%len(node.untriedPositions)
                seleccion = node.untriedPositions[indice]
                    k = seleccion
            if node.untriedbpp != 6 or node.untriedubpp!=4:
                if node.child!=None:
                    k=node.child.pt

        if k > len(str_uindex)-1:
            if node.untriedbpp!=[]:
            indice = sRNG.GetUint()%len(node.untriedbpp)
            seleccion = node.untriedbpp[indice]
                m=seleccion
                node.untriedbpp.remove(m)
                state.SelectPosition(m,k)
                node=node.Addnode(m,k,state)
        else:
            if node.untriedubpp!=[]:
            indice = sRNG.GetUint()%len(node.untriedubpp)
            seleccion = node.untriedubpp[indice]
                m=seleccion
                node.untriedubpp.remove(m)
                state.SelectPosition(m,k)
                node=node.Addnode(m,k,state)

        posi=state.position
        goal=str_index+str_uindex

        for i in range(len(state.position)):
            if goal[i] not in posi:
                posl.append(goal[i])
            else:
                need.append(goal[i])
            if posi[i] not in goal:
                    poslalpha.append(posi[i])

        if len(poslalpha)<=len(str_index):
            eposl=list(itertools.chain(*poslalpha))
        else:
            eposl111=poslalpha[len(poslalpha)-
len(str_index):len(poslalpha)]
            eposl1=list(itertools.chain(*eposl111))
            eposl=poslalpha[0:len(poslalpha)-len(str_index)]+eposl1
```

```
need_GC=calculate_GC_numbers(eposl,defined_GC,need,poslalpha)
        y=state.GetPositions()

        for i in range(len(y)):
            if y[i]>len(str_uindex)-1:
                pa.append(y[i])
            else:
                upa.append(y[i])

        while pa !=[]:
          indice = sRNG.GetUint()%len(pa)
          seleccion = pa[indice]
            cpa=seleccion

            if count_number<=need_GC:

                state.simulationGC(cpa)
                count_number=count_number+1
                pa.remove(cpa)
            else:
                state.simulationAU(cpa)
                pa.remove(cpa)

        while upa!=[]:
          indice = sRNG.GetUint()%len(upa)
          seleccion = upa[indice]
            ucpa=seleccion

            if count_number<=need_GC:
                new_count=need_GC-count_number
                if count_number1<=new_count*2:
                    state.simulationunpairedGC(ucpa)
                    upa.remove(ucpa)
                    count_number1=count_number1+1
                else:
                    state.simulationunpairedAU(ucpa)
                    upa.remove(ucpa)
            else:
                state.simulationunpairedAU(ucpa)
                upa.remove(ucpa)

        posbasep=state.position[len(str_uindex):state.n]
        posbase=state.position[0:len(str_uindex)]

        e=list(itertools.chain(*posbasep))

        for i in range(len(a)):
            posbase.insert(b[i],e[c[i]])

        mutated_s= ''.join(map(str, posbase))

        ini_seq_pool=[]
        ini_str_pool=[]
        GC_pool=[]
        index_seq=0
        if defined_pseudo==1:
```

```
some_str_mfe,some_str_value=calculate__pseudo_mfe_and_str_pkiss(muta
ted_s)

some_str_distance=calculate_structure_distance_pKiss(str_index,len(s
tr_index),some_str_value)
        else:

some_str_mfe,some_str_value=calculate_mfe_and_str(mutated_s)#this  is
the nest structures

some_str_distance=calculate_structure_distance(s,len(s),some_str_val
ue)

        ini_seq_pool.append(mutated_s)
        ini_str_pool.append(some_str_distance)

        GCnum=measureGC(mutated_s)
        GC_pool.append(GCnum)

        #print
      #print "//////////////////Bucle 50/////////////////////: "
        #print

        for i in range(50):
          #print "Ejecucion bucle: " +str(i)
            paired_pos,dif_ini=dif_str(some_str_value)

mutated_seq=GCcontent(defined_GC,GCnum,paired_pos,posbase,posl)

mutated_seq1=check_GC_base3(dif_ini,mutated_seq,posl,defined_GC)
            mutated_seq2=''.join(map(str, mutated_seq1))

            GCnum=measureGC(mutated_seq2)
            GC_pool.append(GCnum)

            if defined_pseudo==1:
                mfe,kkk=pseudoknot_pkiss(mutated_seq2)

new_str_distance=calculate_structure_distance_pKiss(str_index+str_ui
ndex,len(str_index+str_uindex),kkk)
            else:
                kkk=RNA.fold(mutated_seq2)[0]

new_str_distance=calculate_structure_distance(s,len(s),kkk)

            some_str_value=kkk
            some_ini_seq=mutated_seq2
            ini_seq_pool.append(mutated_seq2)
            ini_str_pool.append(new_str_distance)
            index_seq=index_seq+1
            ggg=abs(defined_GC-GCnum)

            if ini_str_pool[index_seq]==1.0:
                break

        #print
      #print "//////////////////FIN DE BUCLE/////////////////////: "
      #print
        max_idx = np.argmax(ini_str_pool)
```

```
            GCnew=GC_pool[max_idx]

            max_val = ini_str_pool[max_idx]
            seq=ini_seq_pool[index_seq]
            ggg=abs(defined_GC-GCnum)
            gggg=abs(defined_GC-GCnew)

            if ini_str_pool[index_seq]==1.0 and ggg<=defined_gd:
                break

            if ini_str_pool[index_seq]==1.0:
                if ggg<=0.01:
                    re=1.0+1.0
                else:
                    re=1.0+0.0

            if max_val<1.0:
                if gggg<=0.01:
                    re=1.0+max_val
                else:
                    re=0.0+max_val

            while node != None:
                node.Update(re)
                node = node.parentNode

        print "SEQ: "
        print seq
        print "MAX_VAL: "
        print max_val
        print "GCNUM: "
        print GCnum

        return seq, max_val, GCnum


    def UCTRNA():
        one_search_start_time=time.time()
        time_out=one_search_start_time+60*10
        state = RNAstructure()
        print "search length:" + str(state.n) + "\n"

        aux = state.GetPositions()
        indice = sRNG.GetUint()%len(state.GetPositions())
        seleccion = aux[indice]
        k=seleccion
        print "k_UCTRNA: "
        print k

        m,goal,GCC = MCTS(root = state, k=k, verbose = False)
        print "Solution:" + str(m)

        if goal==1.0:
            finish_time=time.time()-one_search_start_time
        else:
            finish_time=0.0

        return goal,GCC,finish_time
```

```
def MCTSnoGC(root, itermax, k, verbose = False):

    running_time=time.time()
    out_time=running_time+60*10
    rootnode = Node(state = root)

    for i in range(itermax):
        if time.time() >= out_time:
            break
      print
      print "EJECUCION MCTS_NOGC NUMERO: " + str(i+1)

        node = rootnode # important !    this node is different with
state / node is the tree node
        state = root.Clone() # but this state is the state of the
initialization .  too important !!!
        posi=[]
        posl=[]

        while node.untriedubpp == [] or node.untriedbpp==[]:
            node = node.Selectnode()
            state.SelectPosition(node.position,node.pt)

        if node.untriedPositions != []:
            if k > len(str_uindex)-1:
                if len(node.untriedbpp)==6:
                indice = sRNG.GetUint()%len(node.untriedPositions)
                seleccion = node.untriedPositions[indice]
                    k = seleccion
            else:
                if len(node.untriedubpp)==4:
                indice = sRNG.GetUint()%len(node.untriedPositions)
                seleccion = node.untriedPositions[indice]
                    k = seleccion

        if k > len(str_uindex)-1:
            if node.untriedbpp!=[]:
            indice = sRNG.GetUint()%len(node.untriedbpp)
            seleccion = node.untriedbpp[indice]
                m=seleccion
                node.untriedbpp.remove(m)
                state.SelectPosition(m,k)
                node=node.Addnode(m,k,state)
        else:
            if node.untriedubpp!=[]:
            indice = sRNG.GetUint()%len(node.untriedubpp)
            seleccion = node.untriedubpp[indice]
                m=seleccion
                node.untriedubpp.remove(m)
                state.SelectPosition(m,k)
                node=node.Addnode(m,k,state)

        posi=state.position
        goal=str_uindex+str_index

        for i in range(len(state.position)):
            if goal[i] not in posi:
                posl.append(goal[i])

        while state.GetPositions() != []:
```

```
          indice = sRNG.GetUint() % len(state.GetPositions())
            state.Simulation(state.GetPositions()[indice])

        posbasep=state.position[len(str_uindex):state.n]
        posbase=state.position[0:len(str_uindex)]
        e=list(itertools.chain(*posbasep))

        for i in range(len(a)):
            posbase.insert(b[i],e[c[i]])

        mutated_s= ''.join(map(str, posbase))

        ini_seq_pool=[]
        ini_str_pool=[]
        GC_pool=[]
        index_seq=0
        if defined_pseudo==1:

some_str_mfe,some_str_value=calculate__pseudo_mfe_and_str(mutated_s)
# this is the pseudoknot structure
        else:

some_str_mfe,some_str_value=calculate_mfe_and_str(mutated_s)#this  is
the nest structures


some_str_distance=calculate_structure_distance(s,len(s),some_str_val
ue)

        ini_seq_pool.append(mutated_s)
        ini_str_pool.append(some_str_distance)
        GCnum=measureGC(mutated_s)
        GC_pool.append(GCnum)

        #print
      #print "//////////////////Bucle 50/////////////////////: "
        #print

        for i in range(50):
          #print "Ejecucion bucle: " +str(i)

            paired_pos,dif_ini=dif_str(some_str_value)
            mutated_seq=check_seq_base(paired_pos,posbase,posl)
            mutated_seq1=check_GC_base(dif_ini,mutated_seq,posl)
            mutated_seq2=''.join(map(str, mutated_seq1))
            GCnum=measureGC(mutated_seq2)
            GC_pool.append(GCnum)

            if defined_pseudo==1:
                kkk=pseudoknot(mutated_seq2)[0]
            else:
                kkk=RNA.fold(mutated_seq2)[0]


new_str_distance=calculate_structure_distance(s,len(s),kkk)

            some_str_value=kkk
            some_ini_seq=mutated_seq2
            ini_seq_pool.append(mutated_seq2)
            ini_str_pool.append(new_str_distance)
```

```
            index_seq=index_seq+1
            if ini_str_pool[index_seq]==1.0:
                break

        #print
      #print "//////////////////FIN DE BUCLE//////////////////////: "
        #print

        max_idx = np.argmax(ini_str_pool)
        GCnew=GC_pool[max_idx]
        max_val = ini_str_pool[max_idx]
        seq=ini_seq_pool[index_seq]

        if ini_str_pool[index_seq]==1.0:
            break
        if max_val<1.0:
            re=max_val

        while node != None:
            node.Update(re)
            node = node.parentNode

    return seq,ini_str_pool[index_seq], GCnew



def UCTRNAnoGC():
    one_search_start_time=time.time()
    time_out=one_search_start_time+60*10
    state = RNAstructure()
    print "search length:" + str(state.n) + "\n"
    aux = state.GetPositions()
    indice = sRNG.GetUint()%len(state.GetPositions())
    seleccion = aux[indice]
    k=seleccion
    print "k_UCTRNAnoGC: "
    print k

    m,goal,GC= MCTSnoGC(root = state, itermax = 100000, k=k, verbose
= False)

    if goal==1.0:
        finish_time=time.time()-one_search_start_time
    else:
        finish_time=0.0

    print "solution:"+ str(m)
    print "running time:" + str(finish_time)
    print "GC-content:"+str(GC)
    print "structure distance:" + str(goal)


def       calculate_structure_distance(structure_s,       str_length
,some_str_value):
    sdt=0.0
    sd=0.0
    for i in range(len(structure_s)):
        if some_str_value[i]!=s[i]:
            sd=sd+1
        sdt=(str_length-sd)/str_length
```

```
    return sdt


def    calculate_structure_distance_pKiss(structure_s,    str_length
,some_str_value):
    paired_str=str_index
    unpaired_str=str_uindex

struc,ustruc=calculate__pseudo_sequence_position_pKiss(some_str_valu
e)
    structure_s_new=struc+ustruc
    sdt=0.0
    sd=0.0
    for i in range(len(str_index)):
        if paired_str[i] not in struc:
            sd=sd+1
    for i in range(len(str_uindex)):
        if unpaired_str[i] not in ustruc:
            sd=sd+1
    sdt=(len(structure_s)-sd)/len(structure_s)
    return sdt


def intialization(structure_s):

    BASEPAIRS = ["AU", "CG", "GC", "UA", "GU", "UG"]
    basepro=[0.2,0.3,0.3,0.2,0.1,0.1]
    CGbases=["CG","GC"]
    AUbases=["AU","UA"]
    GUbases=["GU","UG"]
    CGbases=["CG","GC"]

    indice = sRNG.GetUint()%len(CGbases)
    seleccion = CGbases[indice]
    CGbases1=seleccion
    indice = sRNG.GetUint()%len(AUbases)
    seleccion = AUbases[indice]
    AUbases1=seleccion
    indice = sRNG.GetUint()%len(GUbases)
    seleccion = GUbases[indice]
    GUbases1=seleccion
    j=[CGbases1,AUbases1,GUbases1]
    bases="AGCU"

    return



def pick_with_probility(some_list, probabilities):
    x = random.uniform(0, 1)
    cumulative_probability = 0.0
    for item, item_probability in zip(some_list, probabilities):
        cumulative_probability += item_probability
        if x < cumulative_probability: break
    return item


def calculate_sequence_position(seq):
    stack = []
    struc = []
```

```
    ustruc=[]
    for i in xrange(len(seq)):
        if seq[i] == '(':
            stack.append(i)
        if seq[i] == ')':
            struc.append((stack.pop(), i))
        elif seq[i]=='.':
            ustruc.append(i)
    return struc,ustruc


def getinput():
    return input ("percentage of GC : ").lower


def calculate_a(some_str_index):
    a = list(itertools.chain(*some_str_index))
    return a


def calculate_b(some_a):
    b=sorted(some_a)
    return b


def calculate_c(some_a):
    c=sorted(range(len(some_a)),key=lambda x: a[x])
    return c

def calculate_c_1(some_a):
    c=sorted(range(len(some_a)),key=lambda x: some_a[x])
    return c


def getbasepairs(some_str_index):
    midea=[]
    for i in range(len(some_str_index)):
      indice = sRNG.GetUint() % len(BASEPAIRS)
      seleccion = BASEPAIRS[indice]
        midea.append(seleccion)
    return midea


def getunbases(some_str_uindex):
    some_copy_str_uindex=list(some_str_uindex)
    for i in range(len(some_str_uindex)):
      indice = sRNG.GetUint() % len(bases)
      seleccion = bases[indice]
        some_copy_str_uindex[i]=seleccion
    return some_copy_str_uindex


def getwholesequence(some_b,some_c,some_d,some_copy_str_uindex):
    for i in range(len(some_c)):
        some_copy_str_uindex.insert(some_b[i],some_d[some_c[i]])
    wholesequence = ''.join(map(str, some_copy_str_uindex))
    return wholesequence,some_copy_str_uindex


def calculate_d(some_midea):
```

```
    d = list(itertools.chain(*some_midea))
    return d


def calculate_mfe_and_str(sequence):
    rnafold= RNA.fold(sequence)
    mfe=rnafold[1]
    str_v=rnafold[0]
    return mfe,str_v


def error_diagnosis():
    for i in range(len(paired)):
        if paired[i]:
            pass
    return


def identical_position(input_str,predicted_str):## calculate the some
position between initial and mutated
    modified_seq=[]
    modified_pos=[]
    for i in range(len(input_str)):
        if predicted_str[i]==input_str[i]:
            modified_pos.append(i)
            #modified_seq[i]=initial_seq[i]

    return modified_pos


def find_dif_pos_ini():
    dif_pos_ini=[]
    for i in range(len(str_index)):
        if str_index[i] not in paired[i]:
            dif_pos_ini.append(str_index[i])
    return

def
find_dif_str_position_between_target_and_predicted(target_seq,predic
ted_seq):
    break_pairs=["AA","CC","GG","AG","CU","UC","UU","GA"]
    #break_pairs=["UU"]
    comp=["GC","CG"]
    save_paired_pos=[]
    save_unpaired_pos=[]
    ori_paired_pos=[]
    ori_unpaired_pos=[]
    paired_dif_pos=[]

    paired,unpaired=calculate_sequence_position(predicted_seq)

    for i in range(len(paired)):
        if paired[i] not in str_index:
            save_paired_pos.append(paired[i])
            paired_dif_pos.append(random.choice(break_pairs))
        else:
            ori_paired_pos.append(paired[i])


    for i in range(len(unpaired)):
```

```
        if unpaired[i] not in str_uindex:
            save_unpaired_pos.append(unpaired[i])
        else:
            ori_unpaired_pos.append(unpaired[i])

    dif_pos_ini=[]
    dif_pos_base=[]

    for i in range(len(str_index)):
        if str_index[i] not in paired:
            dif_pos_ini.append(str_index[i])
            dif_pos_base.append(random.choice(comp))



    return save_paired_pos,dif_pos_ini, paired_dif_pos,dif_pos_base


def dif_str(predicted_seq):

    save_paired_pos=[]
    save_unpaired_pos=[]
    ori_paired_pos=[]
    ori_unpaired_pos=[]
    paired_dif_pos=[]
    paired,unpaired=calculate_sequence_position(predicted_seq)

    for i in range(len(paired)):
        if paired[i] not in str_index:
            save_paired_pos.append(paired[i])

    dif_pos_ini=[]
    dif_pos_base=[]

    for i in range(len(str_index)):
        if str_index[i] not in paired:
            dif_pos_ini.append(str_index[i])

    return save_paired_pos,dif_pos_ini



def assign_to_paired(some_save_paired_pos, predicted_seq):

    break_pairs=["AA","CC","GG","AG","CU","UC","UU","GA"]
    break_pairs=["AA","UU","AC"]

    paired,unpaired=calculate_sequence_position(predicted_seq)   #get
the position of the new structure


    some_midea=getbasepairs(some_save_paired_pos)
    some_copy_str_uindex=getunbases(unpaired)

    a1=calculate_a(some_save_paired_pos)
    b1=calculate_a(a1)
    c1=calculate_c(a1)
    d1=calculate_(some_midea)
    muta_seq=getwholesequence(a1,c1,d1,some_copy_str_uindex)
```

```
    return muta_seq

def assign_to_unpaired():

    return

def CGmonitor():

    return

def   pair_replace(initial_seq,save_paired_pos,some_paired):#   this
function used to replace basepairs

    to_modify = initial_seq
    a1=calculate_a(save_paired_pos) #connect the paired position into
one sequence
    b1=calculate_b(a1)              #sorted the index of the position
descend order
    c1=calculate_c(a1)               #calculate the original index of
the paired positon
    d1=calculate_d(some_paired)     #connect the base paires into one
sequence

    replacements=[]
    indexes=b1


    for i in range(len(c1)):
        replacements.append(d1[c1[i]])
        to_modify[indexes[i]] = replacements[i]

    return to_modify


def GC_pairreplace(ini_seq, dif_ini_GC,some_paired):
    GC_to_modify = initial_seq
    a1=calculate_a(dif_ini_GC) #connect the paired position into one
sequence
    b1=calculate_b(a1)              #sorted the index of the position
descend order
    c1=calculate_c(a1)               #calculate the original index of
the paired positon
    d1=calculate_d(some_paired)


    return

def check_seq_base(some_paired_pos, predicted_seq,posl):
    check_even=[]
    check_odd=[]
    A_change=["G","C"]
    C_change=["A","U"]
    U_change=["U","C"]
    G_change=["A","G"]
    a1=calculate_a(some_paired_pos)
    even=a1[::2]
    odd=a1[1::2]
    new_even=[]
    new_odd=[]
```

```
    for i in range(len(even)):
        if even[i] not in posl:
            new_even.append(even[i])
            new_odd.append(odd[i])
        if odd[i] not in posl:
            new_even.append(even[i])
            new_odd.append(odd[i])

    for i in range(len(new_odd)):
        check_even.append(predicted_seq[new_even[i]])
        check_odd.append(predicted_seq[new_odd[i]])

    for i in range(len(new_odd)):
        if predicted_seq[new_odd[i]]=="A":
          indice = sRNG.GetUint()%len(A_change)
          seleccion = A_change[indice]
            predicted_seq[new_even[i]]=seleccion

        elif predicted_seq[new_odd[i]]=="U":
          indice = sRNG.GetUint()%len(U_change)
          seleccion = U_change[indice]
            predicted_seq[new_even[i]]=seleccion

        elif predicted_seq[new_odd[i]]=="C":
          indice = sRNG.GetUint()%len(C_change)
          seleccion = C_change[indice]
            predicted_seq[new_even[i]]=seleccion

        elif predicted_seq[new_odd[i]]=="G":
          indice = sRNG.GetUint()%len(G_change)
          seleccion = G_change[indice]
            predicted_seq[new_even[i]]=seleccion


    return predicted_seq

def check_GC_base(some_dif_ini,predicted_seq,posl):## assign GC or CG
to predicted sequence
    GC=["G","C"]
    new_dif_ini=[]
    for i in range(len(some_dif_ini)):
        if some_dif_ini[i] not in posl:
            new_dif_ini.append(some_dif_ini[i])
    a1=calculate_a(new_dif_ini)

    even=a1[::2]
    odd=a1[1::2]
    for i in range(len(odd)):
      indice = sRNG.GetUint()%len(GC)
      seleccion = GC[indice]
        predicted_seq[odd[i]]=seleccion
        if predicted_seq[odd[i]]=="G":
            predicted_seq[even[i]]="C"
        if predicted_seq[odd[i]]=="C":
            predicted_seq[even[i]]="G"

    return predicted_seq
```

```
def update(paired_pos,dif_ini,posl,predicted_seq):
    a1=calculate_a(paired_pos)
    a2=calculate_a(str_index)
    even=a1[::2]
    odd=a1[1::2]
    new_paired_pos=[]
    updated_weaken_pairs_position=[]
    check_odd=[]
    check_even=[]

    for i in range(len(even)):
        if even[i] and odd[i] not in posl:
            new_paired_pos.append(paired_pos[i])

    a_new=calculate_a(new_paired_pos)
    new_even=a_new[::2]
    new_odd=a_new[1::2]

    for i in range(len(new_even)):
        if new_even[i] and new_odd[i] not in a2:
            updated_weaken_pairs_position.append(new_paired_pos[i])

    a_final=calculate_a(updated_weaken_pairs_position)
    final_even=a_final[::2]
    final_odd=a_final[1::2]
    A_change=["G","C"]
    C_change=["A","U"]
    U_change=["U","C"]
    G_change=["A","G"]

    for i in range(len(final_even)):
        if predicted_seq[final_even[i]]=="A":
          indice = sRNG.GetUint()%len(A_change)
          seleccion = A_change[indice]
            predicted_seq[final_odd[i]]=seleccion
        if predicted_seq[final_even[i]]=="U":
          indice = sRNG.GetUint()%len(U_change)
          seleccion = U_change[indice]
            predicted_seq[final_odd[i]]=seleccion
        if predicted_seq[final_even[i]]=="C":
          indice = sRNG.GetUint()%len(C_change)
          seleccion = C_change[indice]
            predicted_seq[final_odd[i]]=seleccion
        if predicted_seq[final_even[i]]=="G":
          indice = sRNG.GetUint()%len(G_change)
          seleccion = G_change[indice]
            predicted_seq[final_odd[i]]=seleccion

    GC=["G","C"]
    AU=["A","U"]
    new_dif_ini=[]
    for i in range(len(dif_ini)):
        if dif_ini[i] not in posl:
            new_dif_ini.append(dif_ini[i])
    a_ini=calculate_a(new_dif_ini)

    even_ini=a_ini[::2]
    odd_ini=a_ini[1::2]

    for i in range(len(even_ini)):
```

```
        if predicted_seq[odd_ini[i]]=="G":
            predicted_seq[even_ini[i]]="C"
        if predicted_seq[odd_ini[i]]=="C":
            predicted_seq[even_ini[i]]="G"

        if predicted_seq[odd_ini[i]]=="A":
            predicted_seq[even_ini[i]]="U"
        if predicted_seq[odd_ini[i]]=="U":
            predicted_seq[even_ini[i]]="A"

    return predicted_seq


def    check_GC_base3(some_dif_ini,predicted_seq,posl,defined_GC):##
assign GC or CG to predicted sequence
    GC=["G","C"]
    AU=["A","U"]
    newgc=measureGC(predicted_seq)
    new_dif_ini=[]
    for i in range(len(some_dif_ini)):
        if some_dif_ini[i] not in posl:
            new_dif_ini.append(some_dif_ini[i])
    a1=calculate_a(new_dif_ini)

    even=a1[::2]
    odd=a1[1::2]

    for i in range(len(odd)):
        if defined_GC>=newgc:
          indice = sRNG.GetUint()%len(GC)
          seleccion = GC[indice]
            predicted_seq[odd[i]]=seleccion
            if predicted_seq[odd[i]]=="G":
                predicted_seq[even[i]]="C"
            if predicted_seq[odd[i]]=="C":
                predicted_seq[even[i]]="G"
        else:
          indice = sRNG.GetUint()%len(AU)
          seleccion = AU[indice]
            predicted_seq[odd[i]]=seleccion
            if predicted_seq[odd[i]]=="A":
                predicted_seq[even[i]]="U"
            if predicted_seq[odd[i]]=="U":
                predicted_seq[even[i]]="A"

    return predicted_seq


def  obtain_initial_sequence(input_structure_s):##obtain  some  good
initial sequence over 0.8
    ini_seq_pool=[]
    ini_str_pool=[]


some_str_index,some_str_uindex=calculate_sequence_position(input_str
ucture_s)
    some_midea=getbasepairs(some_str_index)####   this   is   global
varable
    some_copy_str_uindex=getunbases(some_str_uindex)# unpaired bases
## this is global varable
```

```
    some_a=calculate_a(some_str_index)
    some_b=calculate_b(some_a)
    some_c=calculate_c(some_a)
    some_d=calculate_d(some_midea)
    some_ini_seq,some_ini_str_seq=getwholesequence(some_b,some_c
,some_d , some_copy_str_uindex)
    some_str_mfe,some_str_value=calculate_mfe_and_str(some_ini_seq)

some_str_distance=calculate_structure_distance(input_structure_s,len
(input_structure_s),some_str_value)
    ini_seq_pool.append(some_ini_seq)
    ini_str_pool.append(some_str_distance)
    print some_str_value



    for i in range(10):

        paired_pos,dif_ini=dif_str(some_str_value)
        mutated_seq=check_seq_base(paired_pos,some_ini_str_seq)
        mutated_seq1=check_GC_base(dif_ini,mutated_seq)
        mutated_seq2=''.join(map(str, mutated_seq1))

        kkk=RNA.fold(mutated_seq2)[0]
        some_str_value=kkk
        some_ini_seq=mutated_seq2
        new_str_distance=calculate_structure_distance(s,len(s),kkk)
        ini_seq_pool.append(mutated_seq2)
        ini_str_pool.append(new_str_distance)

    max_idx = np.argmax(ini_str_pool)
    max_val = ini_str_pool[max_idx]

    seq=ini_seq_pool[max_idx]

    return seq,max_val,mutated_seq1


def GCcontent(defined_GC,getGC,some_paired_pos, predicted_seq,posl):
    check_even=[]
    check_odd=[]
    A_change=["G","C"]
    C_change=["A","U"]
    U_change=["U","C"]
    G_change=["A","G"]
    a1=calculate_a(some_paired_pos)
    even=a1[::2]
    odd=a1[1::2]
    new_even=[]
    new_odd=[]

    for i in range(len(even)):
        if even[i] not in posl:
            new_even.append(even[i])
            new_odd.append(odd[i])
        if odd[i] not in posl:
            new_even.append(even[i])
            new_odd.append(odd[i])

    for i in range(len(new_odd)):
```

```
        check_even.append(predicted_seq[new_even[i]])
        check_odd.append(predicted_seq[new_odd[i]])

    for i in range(len(new_odd)):
        if getGC<defined_GC:
            if predicted_seq[new_odd[i]]=="A":
            indice = sRNG.GetUint()%len(A_change)
            seleccion = A_change[indice]
                predicted_seq[new_even[i]]=seleccion
            elif predicted_seq[new_odd[i]]=="U":
                predicted_seq[new_even[i]]="C"
            elif predicted_seq[new_odd[i]]=="C":
                predicted_seq[new_even[i]]="C"
            elif predicted_seq[new_odd[i]]=="G":
                predicted_seq[new_even[i]]="G"
        else:
            if predicted_seq[new_odd[i]]=="A":
                predicted_seq[new_even[i]]="A"
            elif predicted_seq[new_odd[i]]=="U":
                predicted_seq[new_even[i]]="U"
            elif predicted_seq[new_odd[i]]=="C":
            indice = sRNG.GetUint()%len(C_change)
            seleccion = C_change[indice]
                predicted_seq[new_even[i]]=seleccion
            elif predicted_seq[new_odd[i]]=="G":
                predicted_seq[new_even[i]]="A"

    return predicted_seq


def                    GCcontent1(defined_GC,getGC,some_paired_pos,
predicted_seq,posl):

    check_even=[]
    check_odd=[]
    A_change=["G","C"]
    C_change=["A","U"]
    U_change=["U","C"]
    G_change=["A","G"]
    a1=calculate_a(some_paired_pos)
    even=a1[::2]
    odd=a1[1::2]
    new_even=[]
    new_odd=[]

    for i in range(len(even)):
        if even[i] not in posl:
            new_even.append(even[i])
            new_odd.append(odd[i])
        if odd[i] not in posl:
            new_even.append(even[i])
            new_odd.append(odd[i])

    for i in range(len(new_odd)):
        check_even.append(predicted_seq[new_even[i]])
        check_odd.append(predicted_seq[new_odd[i]])

    for i in range(len(new_odd)):

        if predicted_seq[new_odd[i]]=="A":
```

```
            predicted_seq[new_even[i]]="A"
        if predicted_seq[new_odd[i]]=="U":
            if predicted_seq[new_even[i]]=="G":
                predicted_seq[new_even[i]]="C"
            else:
                predicted_seq[new_even[i]]="U"
        if predicted_seq[new_odd[i]]=="C":
            predicted_seq[new_even[i]]="C"
        if predicted_seq[new_odd[i]]=="G":
            if predicted_seq[new_even[i]]=="U":
                predicted_seq[new_even[i]]="A"
            else:
                predicted_seq[new_even[i]]="G"

    return predicted_seq

def measureGC(generate_seq):
    n=len(generate_seq)
    cont=0.0
    indexGC=[]
    indexnotGC=[]
    getGC=0.0
    for i in range(len(generate_seq)):
        if generate_seq[i]=="C":
            cont=cont+1
            indexGC.append(i)
        if generate_seq[i]=="G":
            cont=cont+1
            indexGC.append(i)

        else:
            indexnotGC.append(i)

    getGC=cont/n
    return getGC

def  calculate_GC_numbers(eposl,defined_GC,need,poslalpha):
    cunnt=0.0
    for i in range(len(eposl)):
        if eposl[i]=='G' or eposl[i]=='C':
            cunnt=cunnt+1

    needGC=len(s)*defined_GC-cunnt

    if needGC>0:
        real=round(needGC/2,0)
    else:
        real=0

    return real



def error_check(defined_GC1,defined_gd1,s1,d1):
    if defined_GC1>1.0 or defined_GC1<0.0:
        print "Error,please input a right range in [0, 1.0]"

def calculate__pseudo_sequence_position(seq):
    stack = []
    struc = []
```

```
    pseu=[]
    pseu1=[]
    ustruc=[]
    for i in xrange(len(seq)):
        if seq[i] == '(':
            stack.append(i)
        if seq[i] == ')':
            struc.append((stack.pop(), i))
        if seq[i]=='.':
            ustruc.append(i)
        if seq[i]=='[':
            pseu.append(i)
        if seq[i]==']':
            pseu1.append((pseu.pop(),i))

    return struc,ustruc,pseu1

def calculate__pseudo_mfe_and_str(sequence):
    rnafold= pseudoknot(sequence)
    mfe=rnafold[1]
    str_v=rnafold[0]
    return mfe,str_v

def pseudoknot(se):

    cmd = ["RNAPKplex","-e","-8.10"]
    #tmpdir = mkdtemp()

    p = Popen(cmd, stdin = PIPE, stdout = PIPE)
    print >> p.stdin, se
    p.stdin.close()
    t = p.stdout.readlines()[-1].strip().split(None, 1)
    p.stdout.close()
    return t


def calculate__pseudo_sequence_position_pKiss(seq):
    stack = []
    struc = []
    pseu=[]
    pseu1=[]
    ustruc=[]
    pseularg=[]
    pseularg1=[]
    pseupk=[]
    pseupk1=[]
    for i in xrange(len(seq)):
        if seq[i] == '(':
            stack.append(i)
        if seq[i] == ')':
            struc.append((stack.pop(), i))
        if seq[i]=='.':
            ustruc.append(i)
        if seq[i]=='[':
            pseu.append(i)
        if seq[i]==']':
            pseu1.append((pseu.pop(),i))
        if seq[i]=='{':
            pseularg.append(i)
        if seq[i]=='}':
```

```
            pseularg1.append((pseularg.pop(),i))
        if seq[i]=='<':
            pseupk.append(i)
        if seq[i]=='>':
            pseupk1.append((pseupk.pop(),i))

    return struc+pseu1+pseularg1+pseupk1,ustruc

def calculate__pseudo_mfe_and_str_RNApKplex(sequence):
    rnafold= pseudoknot_RNApKplex(sequence)
    mfe=rnafold[1]
    str_v=rnafold[0]
    return mfe,str_v

def calculate__pseudo_mfe_and_str_pkiss(sequence):
    mfe,str_v= pseudoknot_pkiss(sequence)
    return mfe,str_v

def checkpKiss():

      #pKiss_output   =   subprocess.Popen(["which",   "pKiss_mfe"],
stdout=subprocess.PIPE).communicate()[0].strip()
      pKiss_output   =   subprocess.Popen(["which",   "pKiss_mfe"],
stdout=subprocess.PIPE, shell=True).communicate()[0].strip()
      if len(pKiss_output) > 0 and pKiss_output.find("found") == -1
and pKiss_output.find(" no ") == -1:
            return True
      else:
            print "Please install pKiss"
            print   "Download   from   http://bibiserv2.cebitec.uni-
bielefeld.de/pkiss"
            exit(0)


def pseudoknot_RNApKplex(se):

    cmd = ["RNAPKplex","-e","-8.10"]
    p = Popen(cmd, stdin = PIPE, stdout = PIPE)
    print >> p.stdin, se
    p.stdin.close()
    t = p.stdout.readlines()[-1].strip().split(None, 1)
    p.stdout.close()
    return t



def checkRNAfold():

      RNAfold_output   =   subprocess.Popen(["which",   "RNAfold"],
stdout=subprocess.PIPE).communicate()[0].strip()
      if len(RNAfold_output) > 0 and RNAfold_output.find("found") ==
-1 and RNAfold_output.find(" no ") == -1:
            return True
      else:
            print "Please install RNAfold"
            print "Download from http://www.tbi.univie.ac.at/"
            exit(0)

def pseudoknot_pkiss(se):
```

```
    cmd = ["pKiss_mfe",se]
    p = Popen(cmd, stdin = PIPE, stdout = PIPE)
    t = p.stdout.read().split("\n");
    if (len(t) > 1):
        mfe = "".join(t[1].split(" ")[1])
        structure= "".join(t[1].split(" ")[3])
    p.stdout.close()
    return mfe, structure




if __name__ == "__main__":

    sRNG = pySimpleRNG()
    #CON -sem VAMOS A INTRODUCIR SEMILLAS PARA EL SCRIPT

    #sRNG.SetState(1234,1234) # set the seed

    BASEPAIRS = ["AU", "CG", "GC", "UA"]
    bases="AGCU"

    parser = argparse.ArgumentParser()
    #parser.add_argument('-f',                          dest='action',
action='store_const',const=None,help="monte carlo tree search for RNA
inverse folding")
    parser.add_argument('-s',help="input       the       dot-branket
representation of the RNA secondary structure")
    parser.add_argument('-GC',default=0.5,help="input the target GC
content,the default GC-content is 0.5")
    parser.add_argument('-d',default=0.01,help="input the GC content
error range [0, 0.02],the default GC-content error is 0.01 for nested
structures, 0.02 for pseudoknot structures")
    parser.add_argument('-pk',default=0, help="this is for handling
pseduoknot structures, you can use different pseduoknot prediction
software, value=1 means choose pKiss, value=0 means choose RNAfold")

    parser.add_argument('-sem',default=0,   help="semilla   para   el
script")

    parsed_args = parser.parse_args()
    s=getattr(parsed_args, 's')
    defined_GC=float(getattr(parsed_args, 'GC'))
    defined_gd=float(getattr(parsed_args,'d'))
    defined_pseudo=float(getattr(parsed_args,'pk'))

    sem=float(getattr(parsed_args,'sem'))

    sRNG.SetState(sem, sem)
    print "Semilla manual: " + str(sem)

    if defined_pseudo==1:
        checkpKiss()

str_index1,str_uindex1=calculate__pseudo_sequence_position_pKiss(s)
        str_index=str_index1
        str_uindex=str_uindex1
```

```
    else:
        checkRNAfold() #checkea si se encuentra ViennaPackage
instalado
        str_index,str_uindex=calculate_sequence_position(s)

    print "STR_INDEX(pareja de parentesis): "
    print str_index
    print "STR_UINDEX(puntos): "
    print str_uindex
    print "-----------------------------------------------------"

    midea=getbasepairs(str_index)#### this is global varable

    print "MIDEA: "
    print midea

    copy_str_uindex=getunbases(str_uindex)# unpaired bases ## this is
global varable

    print "COPY_STR_UINDEX: "
    print copy_str_uindex
    print "-----------------------------------------------------"

    a= calculate_a(str_index)

    print "CALCULATE_A A: "
    print a

    b= calculate_b(a)

    print "CALCULATE_B B: "
    print b

    c=calculate_c(a)

    print "CALCULATE_C C: "
    print c

    d=calculate_d(midea)

    print "CALCULATE_D D: "
    print d

    ini_seq,ini_str_seq=getwholesequence(b,c ,d , copy_str_uindex)

    print "INI_SEQ: "
    print ini_seq
    print "INI_STR_SEQ: "
    print ini_str_seq


    if defined_GC<=1.0 and defined_GC>=0.0:
        best_str,GC,run_time=UCTRNA()


        if best_str==1.0:
            print "running time:"+str(int(math.ceil(run_time)))
            print "GC content:"+str(GC)
            print "GC distance:"+str(abs(GC-defined_GC))
            print "structure distance:" +str(best_str)
```

```
        else:
            print "running time:"+str(int(math.ceil(run_time)))
            print "GC content:"+str(GC)
            print "GC distance:"+str(abs(GC-defined_GC))
            print "structure distance:" +str(best_str)

    else:
        UCTRNAnoGC()
```

ANEXO B

Código C/C++ completo.

Clase MCTS_RNA.cpp, main.

```
extern "C"
{
#include <ViennaRNA/fold.h>
#include <ViennaRNA/utils/basic.h>
}
#include <time.h>
#include "MetodosCarga.h"
#include "MetodosDeCalculo.h"
#include "Nodo.h"
#include "SimpleRNG.h"
using namespace std;

vector<int> vectorPuntos;//str_uindex
vector<parParentesis> vectorParentesis;//str_index
vector<string> mIdeaParentesis;
vector<string> mIdeaPuntos;

vector<int> parentesisSeparados;//  -(a) //Indices de los pares de
parentesis, por separado
list<int> b_;
vector<int> b;    //Vector parentesis separados, ordenado de menor a
mayor.
vector<int> c;    //Ordenacion
vector<string> d; //Bases de los pares de parentesis, separadas en
caracteres individuales.

vector<string> BASEPAIRS = {"AU", "CG", "GC", "UA"};
vector<string> BASES = {"A", "G", "C", "U"};

string parametro_s;     //"input the dot-branket representation of the
RNA secondary structure"
float parametro_gc = 0.5; //"input the target GC content,the default
GC-content is 0.5"
float parametro_d = 0.01; //"input the GC content error range [0,
0.02],the default GC-content error is 0.01 for nested structures, 0.02
for pseudoknot structures"
int parametro_pk = 0; //"this is for handling pseduoknot structures,
you can use different pseduoknot prediction software, value=1 means
choose pKiss, value=0 means choose RNAfold"
                                    //todo uso 0 de momento en esta
ejecucion, no contemplo corchetes.

string ini_seq;
vector<string> ini_str_seq;

string best_str = "";
float max_val = 0;
double GCC = 0;

SimpleRNG sRNG;

//argc- numero de parametros
//argv- parametros( el 0 es el nombre del ejecutable(./xyz) )
```

```
 int main(int argc,char *argv[]) {

       int aleatorio = time(NULL);
       //srand (aleatorio);
       //cout<< " Semilla: " << aleatorio << endl;

       int semillaParametro;
       if(argc >= 10){
             semillaParametro = atoi(argv[9]);
             cout<<"Semilla  por  parametro:  "<<  semillaParametro
<<endl;
       }

       //sRNG.SetState(987654,987654);

       sRNG.SetState(semillaParametro, semillaParametro);

       cout.precision(12);

      if (!ObtenerParametros(argc, argv, parametro_s, parametro_gc,
parametro_d, parametro_pk)) {
            printf("Ejecución incorrecta\nEl formato correcto es %s"
                        " [-s] target RNA secondary structure"
                        " [-GC] target GC-content of the RNA sequence"
                        " [-d] GC-content deviation of the solution"
                        " [-pk] nested structure -pk 0 and pseudoknot
structure -pk 1"
                        "\n", argv[0]);
            return 0;
      }

      int tamanoParametroS = parametro_s.length();   //Tamaño  de  la
cadena de entrada
    cout  <<   "Tamaño  parametro  S(secuencia  punto-parentesis):  "<<
tamanoParametroS << endl;

      //Obtengo en que posicion se encuentran los puntos en el
parametro_s (str_uindex)
      ObtenerPuntos(vectorPuntos, parametro_s, tamanoParametroS);
      //Obtengo el par correspondiente de parentesis de apertura y
cierre            (str_index)
      ObtenerListaParentesis(vectorParentesis,          parametro_s,
tamanoParametroS);
      //Muestro las listas obtenidas
      MostrarListas(vectorPuntos, vectorParentesis);

      //mIdeaParentesis-(midea)
      GetBasePairs(vectorParentesis, BASEPAIRS, mIdeaParentesis);

      printf("mIdeaParentesis: ");
    for (string elemento : mIdeaParentesis) { cout << elemento << "
"; } cout << endl;

      //mIdeaPuntos-(copy_str_uindex)
      GetBaseUnpairs(vectorPuntos, BASES, mIdeaPuntos);

      printf("mIdeaPuntos: ");
    for (string elemento : mIdeaPuntos) { cout << elemento << "  "; }
cout << endl;
```

```
//A = vectorParentesis
      SepararParentesis(vectorParentesis, parentesisSeparados);
      printf("A -> Parentesis Separados: ");
      for (int elemento : parentesisSeparados) { cout << elemento <<
"  "; } cout << endl;

//B = b
      //list<int>                    b_(parentesisSeparados.begin(),
parentesisSeparados.end());  //esta linea si no se hace con el copy,
el debugger no ejecuta, no entiendo el por qué
      copy(                              parentesisSeparados.begin(),
parentesisSeparados.end(),back_inserter( b_ ) );
      b_.sort();
      copy(b_.begin(), b_.end(), back_inserter(b));
      printf("B -> Parentesis Ordenados: ");
    for (int elemento : b) { cout << elemento << "  "; } cout << endl;

//C = c
    //todo parentesis separados es variable global, se puede elminar
el paso a la funcion
      ce(parentesisSeparados, c);
      printf("C -> Ordenacion : ");
    for (int elemento : c) { cout << elemento << "  "; } cout << endl;

//D = d Separar los pares de bases
      SepararBasePairs(mIdeaParentesis, d);
      printf("D -> Separacion de BasePairs(mIdeaParentesis): ");
    for (string elemento : d) { cout << elemento << "  "; } cout <<
endl;


//getWholeSequence
      getSecuenciaCompleta(b,    c,    d,    mIdeaPuntos,    ini_seq,
ini_str_seq);
      printf("getWholeSequence -> ini_str_seq: ");
    for (string elemento : ini_str_seq) { cout << elemento << "  "; }
cout << endl;
    //for(auto ele : ini_str_seq){cout << ele << "  ";}cout << endl;

      printf("getWholeSequence -> ini_seq: ");
    cout << ini_seq << endl;

//UCTRNA
    cout<< "Parametro_GC " << parametro_gc << endl;
    time_t tiempoInicial = time(NULL);

    if(parametro_gc>=0 && parametro_gc<=1){
      UCTRNA(best_str, max_val, GCC);
    }else{
      UCTRNAnoGC(best_str, max_val, GCC);
    }

    time_t tiempoSalida= time(NULL);

    cout   <<   "Tiempo   empleado:   "   <<   difftime(tiempoSalida,
tiempoInicial) << " segundos\n" << endl;
      cout << " Exito. Fin de Ejecucion " << endl;

      return 0;
}
```

MétodosCarga.h

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <list>
#include <vector>
#include <algorithm>
#include "SimpleRNG.h"
using namespace std;

#ifndef METODOSCARGA_H_
#define METODOSCARGA_H_

extern SimpleRNG sRNG;

struct parParentesis {
      int apertura = -1;
      int cierre = -1;
};

//todo -> revisar explicaciones

//Obtiene los parametros de entrada y realiza unas comprobaciones
int ObtenerParametros(int argc, char *argv[], string &parametro_s,
float &parametro_gc, float &parametro_d, int &parametro_pk);

//Del parametro de entrada S, obtiene los indices de donde se situan
los puntos
void ObtenerPuntos(vector<int> &vectorPuntos, string &parametro_s,
int tamanoParametroS);

//Del parametro de entrada S, obtiene los indices de donde se situan
los pares de parentesis apertura-cierre
void ObtenerListaParentesis(vector<parParentesis> &vectorParentesis,
string &parametro_s, int tamanoParametroS);

//Muestra las listas de puntos y parentesis obtenidas al inicio
void MostrarListas(vector<int> &vectorPuntos, vector<parParentesis>
&vectorParentesis);

//Obtiene una lista de BASEPAIRS aleatorios de longitud la lista de
parentesis
void      GetBasePairs(vector<parParentesis>      &vectorParentesis,
vector<string> &BASEPAIRS, vector<string> &mIdeaParentesis);

//Obtiene una lista de BASES aleatoriosde longitud la lista de puntos
void GetBaseUnpairs(vector<int> &vectorPuntos, vector<string> &BASES,
vector<string> &mIdeaPuntos);

//Coge los pares de parentesis apertura-cierre, y los separa del
struct metiendo los valores por separado en una lista de enteros
void    SepararParentesis(vector<parParentesis>    &vectorParentesis,
vector<int> &parentesisSeparados);

void ce(vector<int> &parentesis, vector<int> &c);
```

```
//Bases de los pares de parentesis, separadas en caracteres
individuales.
void SepararBasePairs(vector<string> &mIdeaParentesis, vector<string>
&d);

//getWholeSequence
void getSecuenciaCompleta(vector<int> b, vector<int> c,
vector<string> d, vector<string> mIdeaPuntos, string
&ini_seq,vector<string> &ini_str_seq);

#endif /* METODOSCARGA_H_ */
```

MétodosCarga.cpp

```
#include "MetodosCarga.h"

//Comprobacion de parametros, tienen que estar todos con el formato
correcto. No se comprueba el "valor" puesto.
//Si uno de los parametros intermedios no esta, falla, no se hace esa
comprobacion. Tienen que estar en orden o bien faltar
//parametros por el final y se escogen los valores default.
int ObtenerParametros(int argc, char *argv[], string &parametro_s,
float &parametro_gc, float &parametro_d, int &parametro_pk) {
      int i;
      //nombre, -s + valor, -GC + valor, -d + valor, -pk + valor. (9
parametros)
      if (argc<2) //Al menos tiene que estar el parametro -s, los
otros tienen valores default
            return 0;

      for (i=1; i<argc; i = i+2) {
            switch(i)
            {
                  case 1:
                        if (strcmp(argv[i], "-s")!=0){
                          printf("Parametro argv[%d]: %s incorrecto
\n", i, argv[i]); return 0;
                        }
                        parametro_s = argv[2];
                        break;

                  case 3:
                        if (strcmp(argv[i], "-GC")!=0){
                          printf("Parametro argv[%d]: %s incorrecto
\n", i, argv[i]); return 0;
                        }
                        if(argv[4] != NULL)
                              parametro_gc = atof(argv[4]);
                        break;

                  case 5:
                        if (strcmp(argv[i], "-d")!=0){
                          printf("Parametro argv[%d]: %s incorrecto
\n", i, argv[i]); return 0;
                        }
                        if(argv[6] != NULL)
                              parametro_d = atof(argv[6]);
                        break;

                  case 7:
                        if (strcmp(argv[i], "-pk")!=0){
                          printf("Parametro argv[%d]: %s incorrecto
\n", i, argv[i]); return 0;
                        }
                        if(argv[8] != NULL)
                              parametro_pk = atoi(argv[8]);
                        break;
            }
      }
      return 1;//lectura correcta
}
```

```
void ObtenerPuntos(vector<int> &vectorPuntos, string &parametro_s,
int tamanoParametroS){
      for(int i = 0; i < tamanoParametroS; i++){
            if (parametro_s[i] == '.'){
                  vectorPuntos.push_back(i);
            }
      }
}

void ObtenerListaParentesis(vector<parParentesis> &vectorParentesis,
string &parametro_s, int tamanoParametroS){
      list<int> stack;
      parParentesis par;
      int ultimo;

      for(int i = 0; i < tamanoParametroS; i++){
            if (parametro_s[i] == '('){
                  stack.push_back(i);
            }else if(parametro_s[i] == ')'){
                  ultimo = stack.back();
                  par.apertura = ultimo;
                  stack.pop_back();
                  par.cierre = i;
                  vectorParentesis.push_back(par);
            }
      }
}

void MostrarListas(vector<int> &vectorPuntos, vector<parParentesis>
&vectorParentesis){
      printf("Puntos: ");
    for (int punto : vectorPuntos) {
        cout << punto << "  ";
    }
      cout << endl;

      printf("Pares de Parentesis: ");
    for (parParentesis par : vectorParentesis) {
        cout << par.apertura << "/" << par.cierre << "  ";
    }
      cout << endl;
}

void      GetBasePairs(vector<parParentesis>      &vectorParentesis,
vector<string> &BASEPAIRS, vector<string> &mIdeaParentesis){
      int aleatorio;

      for(unsigned i=0; i<vectorParentesis.size(); i++){
            aleatorio = sRNG.GetUint() % BASEPAIRS.size();
            mIdeaParentesis.push_back(BASEPAIRS[aleatorio]);
      }

}

void GetBaseUnpairs(vector<int> &vectorPuntos, vector<string> &BASES,
vector<string> &mIdeaPuntos){
      int aleatorio;

      for(unsigned i=0; i<vectorPuntos.size(); i++){
            aleatorio = sRNG.GetUint() % BASES.size();
```

```
            mIdeaPuntos.push_back(BASES[aleatorio]);
      }
}

void    SepararParentesis(vector<parParentesis>    &vectorParentesis,
vector<int> &parentesisSeparados){
    for (parParentesis par : vectorParentesis) {
      parentesisSeparados.push_back(par.apertura);
      parentesisSeparados.push_back(par.cierre);
    }
}

void ce(vector<int> &parentesisSeparados, vector<int> &c){
      //Parentesis separados es variable global, puedo no pasarla por
parametro.
      vector<int>
VCopiaParentesisSeparados(parentesisSeparados.begin(),
parentesisSeparados.end());
      int minElementIndex;
      int                        maxElementIndex                        =
max_element(VCopiaParentesisSeparados.begin(),
VCopiaParentesisSeparados.end())                                          -
VCopiaParentesisSeparados.begin();

      if(parentesisSeparados.size() % 2 == 0){ //si es par
            for( unsigned i=0; i<parentesisSeparados.size(); i++){
      //Bucle de 0 al tamaño del vector aunque siempre acabe en el
break la ejecución.
                  minElementIndex                                         =
min_element(VCopiaParentesisSeparados.begin(),
VCopiaParentesisSeparados.end())                                          -
VCopiaParentesisSeparados.begin();
                  c.push_back(minElementIndex);
                  VCopiaParentesisSeparados[minElementIndex]        =
VCopiaParentesisSeparados[maxElementIndex]+1;
                  if(minElementIndex == maxElementIndex){   //Ya    he
ordenado todos los valores y he llegado al último
                        break;
                  }
            }
      }else{      //si es impar
            printf("SECUENCIA  INCORRECTA,  YA  QUE  EL  NUMERO  DE
PARENTESIS NO PUEDE SER IMPAR\n");
      }
}

void SepararBasePairs(vector<string> &mIdeaParentesis, vector<string>
&d){
      string el;

    for (string elemento : mIdeaParentesis) {
      for (unsigned i=0; i<elemento.length(); ++i)
        {
            el = elemento.at(i);    //Extraigo   cada   caracter   del
string y lo introduzco por separado en d
          d.push_back(el);
        }
    }
}
```

```
void    getSecuenciaCompleta(vector<int>    b,    vector<int>    c,
vector<string>     d,     vector<string>     mIdeaPuntos,     string
&ini_seq,vector<string> &ini_str_seq){
      int num;
      string base;
      vector<string>::iterator it1 = mIdeaPuntos.begin();

      for(unsigned i=0; i< c.size(); i++){
            it1 = mIdeaPuntos.begin() + b[i];
            num = c[i];
            base = d[num];
            mIdeaPuntos.insert(it1, base);
      }

      //Secuencia completa en un vector de strings
      ini_str_seq.insert(ini_str_seq.begin(),    mIdeaPuntos.begin(),
mIdeaPuntos.end());

      //Secuencia completa en un string
      for(const auto &str : ini_str_seq) {
            ini_seq += str;
      }
}
```

Nodo.h

```
#ifndef NODO_H_
#define NODO_H_

#include <vector>
#include <algorithm>
#include "RNAestructura.h"
#include "SimpleRNG.h"

extern int global;

extern SimpleRNG sRNG;

class Nodo {
public:

	int position;
	int pt;
	Nodo *NodoPadre;
	vector<Nodo*> NodosHijos;
	Nodo *NodoHijo;
	double victorias;
	int visitas;
	vector<int> untriedSearches;
	vector<int> untriedUBPP;
	vector<int> untriedBPP;
	vector<int> untriedPositions;
	RNAestructura RNA;

	Nodo();
	Nodo(int position, int pt, Nodo *NodoPadre, RNAestructura *RNA);
	//state = RNAestructura
	virtual ~Nodo();

	//Devuelve el ultimo nodo, despues de hacer una ordenacion.
	Nodo* SeleccionarNodo();

	//Añade el nodo indicado por parameto al vector de nodos hijos.
	void AnadirNodo(int m, int k, Nodo &n);

	//Actualiza las victorias y visitas del nodo.
	void ActualizarNodo(double resultado);

	//Muestra el vector UntriedPositions.
	void MostrarUntriedPositions();

};

#endif /* NODO_H_ */
```

Nodo.cpp

```
#include "Nodo.h"
#include "math.h"

Nodo::Nodo() {
this->position = -1;
this->pt = -1;
this->NodoPadre = NULL;
this->NodosHijos;
this->NodoHijo = NULL;
this->victorias = 0;
this->visitas = 0;
this->untriedSearches;
this->untriedUBPP;
this->untriedBPP;
this->untriedPositions;
this->RNA;
}

Nodo::Nodo(int position, int pt, Nodo *NodoPadre, RNAestructura *RNA)
{
this->position = position;
this->pt = pt;
this->NodoPadre = NodoPadre;
this->NodoHijo = NULL;
this->victorias = 0;
this->visitas = 0;
this->RNA = *RNA;
this->untriedSearches = RNA->GetSearch();
this->untriedUBPP = RNA->Getubpp();
this->untriedBPP = RNA->Getbpp();
this->untriedPositions = RNA->GetPositions();
}

Nodo::~Nodo() {

}

//todo mirar funciones lambda en c++
Nodo* Nodo::SeleccionarNodo(){

      struct parNodo {
            double valor = 0;
            int index = 0;
      };

      //cout<<endl;
      vector<Nodo*> NodosHijos_copia = this->NodosHijos;
      vector<Nodo*> prueba;
      //vector<double> enterosPrueba;
      vector<parNodo> parPrueba;

      //cout<<endl;
      //cout << "Ordenacion prueba:"<<endl;
      for(unsigned i=0; i<NodosHijos.size(); i++){
            //cout<<"[position,   pt]:   "   <<   this->NodosHijos[i]-
>position << " / " << this->NodosHijos[i]->pt <<endl;
            //cout<<"[victorias, visitas]: " << this->NodosHijos[i]-
>victorias << " / " << this->NodosHijos[i]->visitas<<endl;
```

```
            double sol = ( (this->NodosHijos[i]->victorias/this-
>NodosHijos[i]->visitas)+(0.1*sqrt(2*log(this->visitas)/this-
>NodosHijos[i]->visitas)) );
            //enterosPrueba.push_back(sol);
            parNodo pp;
            pp.valor = sol;
            pp.index = i;
            parPrueba.push_back(pp);
            //cout << "Ecuación: "<< pp.valor << endl;
      }

      //cout << "Vector double prueba - PRE sort:"<<endl;
      //for(unsigned i=0; i<parPrueba.size(); i++){
      //    cout << "Resultados / I: "<< parPrueba[i].valor<< " / "<<
parPrueba[i].index << endl;
      //}
      //cout<<endl;

      sort(parPrueba.begin(), parPrueba.end(), [ ]( const auto& lhs,
const auto& rhs )
      {
            return lhs.valor < rhs.valor;
      });

      //cout << "Vector double prueba - POST sort:"<<endl;
      //for(unsigned i=0; i<parPrueba.size(); i++){
      //    cout << "Resultados / I: "<< parPrueba[i].valor<< " / "<<
parPrueba[i].index << endl;
      //}
      //cout<<endl;

      //int                    maxElementIndex                    =
std::max_element(enterosPrueba.begin(),enterosPrueba.end())       -
enterosPrueba.begin();
      //cout << "maxElementIndex: "<< maxElementIndex << endl;
/*
      for(unsigned i=maxElementIndex; i<enterosPrueba.size(); i++){
            if(enterosPrueba[maxElementIndex] == enterosPrueba[i]){
                  maxElementIndex = i;
                  cout << "NUEVO maxElementIndex: "<< maxElementIndex
<< endl;
            }
      }


      cout<<endl;
      cout<<"Seleccionar Nodo: "<<endl;
      cout<<"[position, pt]: " << NodosHijos_copia[maxElementIndex]-
>position << " / " << NodosHijos_copia[maxElementIndex]->pt <<endl;
      cout<<"[victorias,          visitas]:          "          <<
NodosHijos_copia[maxElementIndex]->victorias   <<   "   /   "   <<
NodosHijos_copia[maxElementIndex]->visitas<<endl;

      return NodosHijos_copia[maxElementIndex];
      */
      //cout<<endl;
      //cout<<"Seleccionar Nodo: "<<endl;
      //cout<<"[position,            pt]:            "            <<
NodosHijos_copia[parPrueba.back().index]->position  <<  "  /  "  <<
NodosHijos_copia[parPrueba.back().index]->pt <<endl;
```

```
      //cout<<"[victorias,                visitas]:                "
<<NodosHijos_copia[parPrueba.back().index]->victorias  <<  "  /  "  <<
NodosHijos_copia[parPrueba.back().index]->visitas<<endl;
      return NodosHijos_copia[parPrueba.back().index];
}

void Nodo::AnadirNodo(int m, int k, Nodo &n){
    if          (find(this->untriedPositions.begin(),          this-
>untriedPositions.end(), k) != this->untriedPositions.end()) {
          vector<int>::iterator new_end;
          //new_end  =  remove(this->untriedPositions.begin(),  this-
>untriedPositions.end(), k);
          this->untriedPositions.erase(remove(this-
>untriedPositions.begin(),  this->untriedPositions.end(),  k),  this-
>untriedPositions.end());
    }
    this->NodosHijos.emplace_back(&n);
    this->NodoHijo = &n;

      //cout<<"Añadir Nodo: "<<endl;
      //cout<<"[position, pt]: " << n.position << " / " << n.pt <<endl;
      //cout<<"[victorias, visitas]: " << n.victorias << " / " <<
n.visitas <<endl;
}

void Nodo::ActualizarNodo(double resultado){
      this->visitas = this->visitas + 1;
      this->victorias = this->victorias + resultado;
}

void Nodo::MostrarUntriedPositions(){
      cout<< "UntriedPositions: "<< endl;
      for(int elemento : this->untriedPositions){
            cout << elemento << " " <<endl;
      }
}
```

RNAestructura.h

```
#ifndef RNAESTRUCTURA_H_
#define RNAESTRUCTURA_H_

#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <list>
#include <vector>
#include <algorithm>
#include "MetodosCarga.h"
#include "SimpleRNG.h"
using namespace std;

extern vector<string> BASEPAIRS;
extern vector<string> BASES;

extern vector<int> vectorPuntos;//uindex
extern vector<parParentesis> vectorParentesis;//index
extern vector<string> mIdeaParentesis;
extern vector<string> mIdeaPuntos;

extern vector<int> parentesisSeparados;  //a
extern vector<int> b;
extern vector<int> c;
extern vector<string> d;

struct positionStruct {
      int punto = -1;
      int apertura = -1;
      int cierre = -1;
      string base = "";
};

extern SimpleRNG sRNG;

class RNAestructura {
public:

      vector<string> Search;
      vector<string> BASEPAIRS_RNA;
      vector<string> BASES_RNA;
      vector<string> BASE_RNA;
      //CAMBIO-> QUITO VECTOR POSITION Y METO EL "STRING" BASE EN LA
ESTRUCTURA PARA TRABAJAR MEJOR.
      //std::vector<string> position;      //vector en el que guardo
los caracteres que va cambiando de la "positionStructVector"
      vector<positionStruct> positionStructVector;    //en python une
puntos con lista parentesis en una misma variable.(este vector guarda
eso en forma

                  //de struct de 3 variables)
      int n;
      int number;


      RNAestructura();
```

```
	virtual ~RNAestructura();

	//Relleno positionStructVector con la union de la posicion de
los puntos y el parParentesis
	void rellenarPositionStructVector();

	//Devuelve true si la posicion indicada por parametro del
vector, se trata de un punto. Es decir que punto!=-1
	bool esPunto(int numPosition);

	//Devuelve true si la posicion indicada por parametro del
vector, se trata de un par de ParParentesis. Es decir que apertura!=-
1 y cierre !=-1
	bool esParParentesis(int numPosition);

	//Devuelve true si en la posicion indicada existe una base, es
decir que base != "". Que no este vacio.
	bool hayBase(int numPosition);

	//Suma 1 a number
	void count();

	//Clona los datos de la clase a la estructura que se pasa por
parametro
	void Clone(RNAestructura *clon);

	//¿no se usa en el codigo de python?
	void Rewards(int k);

	//Copia de Basepairs_RNA a Search la posicion indicada por
parametro.
	void SelectBasePairs(int move);

	void SelectStatePosition(int l);

	void Simulation(int l);

	void Simulation1(int l, int needgc, int &count_number);

	void SimulationGC(int l);

	void SimulationAU(int l);

	void SimulationUnpairedAU(int l);

	void SimulationUnpairedGC(int l);

	void SelectPosition(int m, int k);

	vector<int> Getubpp();

	vector<int> Getbpp();

	vector<int> GetSearch();

	vector<int> GetPositions();

	void MostrarPositionStructVector();
};
#endif /* RNAESTRUCTURA_H_ */
```

RNAestructura.cpp

```
#include "RNAestructura.h"

RNAestructura::RNAestructura() {

	this->Search = {"", "", "", "", "", "", "", "", "", ""};
	this->BASEPAIRS_RNA = {"AU", "CG", "GC", "UA", "GU", "UG"};
	this->BASES_RNA = {"A", "C", "G", "U", "AU", "CG", "GC", "UA",
"GU", "UG"};
	this->BASE_RNA = {"A", "C", "G", "U"};
	//this->position;	//vector en el que guardo los caracteres
que va cambiando de la "positionStructVector"
	//this->positionStructVector;	//en python une puntos con
lista parentesis en una misma variable.(este vector guarda eso en
forma de struct de 3 variables)
	this->n = vectorPuntos.size() + vectorParentesis.size();
	this->number = 0;

	rellenarPositionStructVector();

    //MostrarPositionStructVector();
}

RNAestructura::~RNAestructura() {

}

//Relleno positionStructVector con la union de la posicion de los
puntos y el parParentesis
void RNAestructura::rellenarPositionStructVector(){
	positionStruct posStruct;
	for(unsigned i=0; i<vectorPuntos.size(); i++){
		posStruct.punto = vectorPuntos[i];
		this->positionStructVector.push_back(posStruct);
		posStruct.punto = -1;
	}

    for (parParentesis par : vectorParentesis) {
        posStruct.apertura = par.apertura;
        posStruct.cierre = par.cierre;
            this->positionStructVector.push_back(posStruct);

        posStruct.apertura = -1;
        posStruct.cierre = -1;
    }
}

bool RNAestructura::esPunto(int numPosition){
	if(this->positionStructVector[numPosition].punto != -1){
		return true;
	}
	return false;
}

bool RNAestructura::esParParentesis(int numPosition){
	if( (this->positionStructVector[numPosition].apertura != -1) &&
(this->positionStructVector[numPosition].cierre != -1) ){
		return true;
	}
```

```
	return false;
}

bool RNAestructura::hayBase(int numPosition){
	if(this->positionStructVector[numPosition].base != ""){
		return true;
	}
	return false;
}

void RNAestructura::count(){
	this->number++;
}

//Clona los datos de la clase a la estructura que se pasa por parametro
void RNAestructura::Clone(RNAestructura *clon){
	clon->Search = this->Search;
	clon->BASEPAIRS_RNA = this->BASEPAIRS_RNA;
	//clon->position = this->position;
	clon->positionStructVector = this->positionStructVector;
}

//TODO ¿en python no entra en rewards en las ejecuciones??
void RNAestructura::Rewards(int k){
	//std::vector<parParentesis>	a_(vectorParentesis.begin(),
vectorParentesis.end());
	//std::vector<int> b_(b.begin(), b.end());
	//std::vector<int> c_(c.begin(), c.end());
	//std::vector<string> d_(d.begin(), d.end());

	vector<string> posBaseP;
	/*
	struct posBaseStruct {
		int entero = -1;
		string cadena = "";
	};*/
	vector<string> posBase;

	//Esto es comun a ambos if
	int contador = 0;
	for(unsigned i=vectorPuntos.size(); i < this->n; i++){
		//posBaseP[contador] = this->position[i];
		contador++;
	}
	contador=0;
	for(unsigned i=0; i < vectorPuntos.size(); i++){
		//posBase[contador] = this->position[i];
		contador++;
	}
	contador=0;
	//e=list(itertools.chain(*posbasep)) en python hace esto, ¿crea
una lista de la variable posbasep?
	                                                //yo ya tengo la
lista e=posBaseP

	//for(int i=0; i<a_.size(); i++){
		//posBase[i] =
	//}
	/*
	if(k > puntos.size()-1){	}
```

```
	if(k <= puntos.size()-1){	}
	*/

}//

void RNAestructura::SelectBasePairs(int move){
	this->Search[move] = this->BASEPAIRS_RNA[move];
}

void RNAestructura::SelectStatePosition(int l){
	if(l > vectorPuntos.size()-1){
		this->positionStructVector[l].base = mIdeaParentesis[l -
vectorPuntos.size()];
	}else{
		this->positionStructVector[l].base = mIdeaPuntos[l];
	}
}

void RNAestructura::Simulation(int l){
	int aleatorio;

	if(l > vectorPuntos.size()-1){
		aleatorio = sRNG.GetUint() % BASEPAIRS.size();
		this->positionStructVector[l].base                    =
BASEPAIRS[aleatorio];
	}else{
		aleatorio = sRNG.GetUint() % BASES.size();
		this->positionStructVector[l].base = BASES[aleatorio];
	}
}

void    RNAestructura::Simulation1(int    l,    int    needgc,    int
&count_number){
	int aleatorio;
	vector<string> BASES1 = {"CG", "GC"};
	vector<string> BASES2 = {"AU", "UA"};
	vector<string> BASES3 = {"A", "U"};

	if(l > (vectorPuntos.size()-1) && (count_number<=needgc) ){
		aleatorio = sRNG.GetUint() % BASES1.size();
		this->positionStructVector[l].base =  BASES1[aleatorio];
		count_number++;
	}
	if(l > (vectorPuntos.size()-1) && (count_number>needgc) ){
		aleatorio = sRNG.GetUint() % BASES2.size();
		this->positionStructVector[l].base =  BASES2[aleatorio];
	}
	if(l <= vectorPuntos.size()-1){
		aleatorio = sRNG.GetUint() % BASES3.size();
		this->positionStructVector[l].base =  BASES3[aleatorio];
	}

}

void RNAestructura::SimulationGC(int l){
	int aleatorio;
	vector<string> BASES = {"CG", "GC"};
	aleatorio = sRNG.GetUint() % BASES.size();
	this->positionStructVector[l].base = BASES[aleatorio];
}
```

```
void RNAestructura::SimulationAU(int l){
      int aleatorio;
      vector<string> BASES = {"AU", "UA"};
      aleatorio = sRNG.GetUint() % BASES.size();
      this->positionStructVector[l].base = BASES[aleatorio];
}

void RNAestructura::SimulationUnpairedAU(int l){
      int aleatorio;
      vector<string> BASES = {"A", "U"};
      aleatorio = sRNG.GetUint() % BASES.size();
      this->positionStructVector[l].base = BASES[aleatorio];
}

void RNAestructura::SimulationUnpairedGC(int l){
      int aleatorio;
      vector<string> BASES = {"G", "C"};
      aleatorio = sRNG.GetUint() % BASES.size();
      this->positionStructVector[l].base = BASES[aleatorio];
}

void RNAestructura::SelectPosition(int m, int k){
      this->positionStructVector[k].base = this->BASES_RNA[m];
      this->positionStructVector[k].punto = -1;
      this->positionStructVector[k].apertura = -1;
      this->positionStructVector[k].cierre = -1;
}

//todo habria que probar que el retorno no esté vacio donde se llame
a las funciones.
vector<int> RNAestructura::Getubpp(){
      //std::vector<string> uBASES = {"A", "U", "C", "G"};
      vector<int> retorno;

            //for de 0 a 4
            for(unsigned j=0; j<4; j++){
                if        (find(this->BASE_RNA.begin(),        this-
>BASE_RNA.end(), this->Search[j]) != this->BASE_RNA.end()) {

                }else{
                    retorno.push_back(j); //Devuelvo  las  posiciones
en la que se ha encontrado.
                }
            }
            return retorno;
}

vector<int> RNAestructura::Getbpp(){
      //std::vector<string> pBASES = {"AU", "CG", "GC", "UA", "GU",
"UG"};
      vector<int> retorno;

      //for de 4 a 10
            for(unsigned j=4; j<10; j++){
                if      (find(this->BASEPAIRS_RNA.begin(),      this-
>BASEPAIRS_RNA.end(), this->Search[j]) != this->BASEPAIRS_RNA.end())
{

                }else{
```

```
                  retorno.push_back(j);   //Devuelvo la posicion en la
que se ha encontrado.
                }
            }
            return retorno;
}

vector<int> RNAestructura::GetSearch(){
      //std::vector<string> BASES = {"A", "U", "C", "G","AU", "CG",
"GC", "UA", "GU", "UG"};
      vector<int> retorno;

            for(unsigned j=0; j<this->Search.size(); j++){
                if        (find(this->BASES_RNA.begin(),        this-
>BASES_RNA.end(), this->Search[j]) != this->BASES_RNA.end()) {

                }else{
                  retorno.push_back(j);   //Devuelvo la posicion en la
que se ha encontrado
                }
            }
            return retorno;
}

//Probado-Funcionamiento correcto
vector<int> RNAestructura::GetPositions(){
      //std::vector<string> BASES = {"A", "U", "C", "G","AU", "CG",
"GC", "UA", "GU", "UG"};
      vector<int> retorno;

      for(unsigned j=0; j<this->positionStructVector.size(); j++){
          if  (find(this->BASES_RNA.begin(),  this->BASES_RNA.end(),
this->positionStructVector[j].base)  !=  this->BASES_RNA.end())  {//si
está el elemento...

          }else{  //Si no está el elemento (Lo que buscamos)
            retorno.push_back(j);  //Devuelvo la posicion en la que
se ha encontrado
          }
      }
      return retorno;
}

void RNAestructura::MostrarPositionStructVector(){

      printf("POSITION_STRUCT_VECTOR:  ");
    for (positionStruct p : this->positionStructVector) {
        cout << p.punto << "/" << p.apertura << "/" << p.cierre <<
"/" << p.base << "  ";
    }
      cout << endl;
}
```

MetodosDeCalculo.h

```
using namespace std;
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <cmath>
#include "RNAestructura.h"
#include "Nodo.h"
#include "SimpleRNG.h"

extern "C"
{
#include <ViennaRNA/fold.h>
#include <ViennaRNA/utils/basic.h>
}


#ifndef METODOSDECALCULO_H_
#define METODOSDECALCULO_H_

extern vector<int> parentesisSeparados;// -(a)
extern vector<int> b;
extern vector<int> c;
extern vector<string> d;

extern string parametro_s;
extern float parametro_gc;
extern float parametro_d;
extern int parametro_pk;

extern vector<int> vectorPuntos;//str_uindex
extern vector<parParentesis> vectorParentesis;//str_index

extern SimpleRNG sRNG;

void UCTRNA(string &best_str, float &max_val, double &GCC);

void UCTRNAnoGC(string &best_str, float &max_val, double &GCC);

void MCTS(RNAestructura *root, int k, string &best_str_solution, float
&max_val_solution, double &GCC_solution);

void MCTSnoGC(RNAestructura *root, int k, string &best_str_solution,
float &max_val_solution, double &GCC_solution);

int calculate_GC_numbers(vector<string> eposl, float parametro_gc,
vector<positionStruct> need, vector<positionStruct> poslalpha);

void calculate_mfe_and_str(string sequence, float &mfe, string
&str_v);

double calculate_structure_distance(string structure_s, float
str_length, string some_str_value);

double measureGC(string generate_seq);
```

```
void                              check_GC_base3(vector<parParentesis>
&some_dif_ini,vector<positionStruct>                  &predicted_seq,
vector<positionStruct> &posl);

void                               check_GC_base(vector<parParentesis>
&some_dif_ini,vector<positionStruct>                  &predicted_seq,
vector<positionStruct> &posl);

void    dif_str(string    &some_str_value,    vector<parParentesis>
&paired_pos, vector<parParentesis> &dif_ini);

void calculate_sequence_position(string &seq, vector<parParentesis>
&struc, vector<int> &ustruc);

void      GCcontent(float      parametro_gc,      double      getGC,
vector<parParentesis>    &some_paired_pos,    vector<positionStruct>
&predicted_seq, vector<positionStruct> &posl);

void     check_seq_base(float     parametro_gc,     double     getGC,
vector<parParentesis>    &some_paired_pos,    vector<positionStruct>
&predicted_seq, vector<positionStruct> &posl);


vector<positionStruct> unirIndexYUindex();
bool esPunto(int numPosition, vector<positionStruct> vector);
bool esParParentesis(int numPosition, vector<positionStruct> vector);
bool hayBase(int numPosition, vector<positionStruct> vector);

#endif /* METODOSDECALCULO_H_ */
```

MetodosDeCalculo.cpp

```
#include "MetodosDeCalculo.h"

void UCTRNA(string &best_str, float &max_val, double &GCC){
      RNAestructura *state = new RNAestructura;

      cout<<"Search length:  "<< state->n <<endl;

      int aleatorio, k;
      vector<int> positions_ = state->GetPositions();
      aleatorio       =       sRNG.GetUint()       %       state-
>positionStructVector.size();
      k = positions_[aleatorio];

      MCTS(state, k, best_str, max_val, GCC);

      cout<<"Fin ejecucion MCTS - Soluciones: "<<endl;
      cout<<"best_str: "<<best_str<<endl;
      cout<<"GCC content: "<<GCC<<endl;
      cout<<"GCC distance(diferencia entre contenido GC propuesto y
alcanzado): "<< abs(GCC-parametro_gc) <<endl;
      cout<<"max_val(Similitud): "<<max_val<<endl;

      delete state;
}

void UCTRNAnoGC(string &best_str, float &max_val, double &GCC){
      RNAestructura *state = new RNAestructura;
      int aleatorio, k;
      vector<int> positions_ = state->GetPositions();
      aleatorio       =       sRNG.GetUint()       %       state-
>positionStructVector.size();
      k = positions_[aleatorio];

      MCTSnoGC(state, k, best_str, max_val, GCC);

      cout<<"Fin ejecucion MCTSnoGC - Soluciones: "<<endl;
      cout<<"best_str: "<<best_str<<endl;
      cout<<"GCC content: "<<GCC<<endl;
      cout<<"GCC distance(diferencia entre contenido GC propuesto y
alcanzado): "<< abs(GCC-parametro_gc) <<endl;
      cout<<"max_val(Similitud): "<<max_val<<endl;

      delete state;
}

void MCTS(RNAestructura *root, int k, string &best_str_solution, float
&max_val_solution, double &GCC_solution){
    time_t tiempoActual = time(NULL);
    //time_t tiempoSalida= tiempoActual+(60*10);
    time_t tiempoSalida= tiempoActual+(600*10);
    cout<<endl;
    cout << "DENTRO DE MCTS" << endl;

      Nodo *rootNode = new Nodo(-1, -1, NULL, root), *Node, *Node2;
      RNAestructura *state = new RNAestructura();
      root->Clone(state);

      int aleatorio = -1;
```

```
      int    new_count=0,    need_GC=0,    index_seq=0,    contador=0,
count_number=0, count_number1=0;
      //he cambiado de float a double
      double  some_str_distance,  new_str_distance,  max_idx,  GCnew,
GCnum;
      double ggg=0, gggg=0;
      float some_str_mfe=0, notUsable;
      //double
      double re=0, max_val = 0;
      string some_str_value="", kkk, some_ini_seq, seq;
      vector<string> eposl;
      vector<string> eposl111;
      vector<int> y;
      vector<parParentesis> paired_pos;
      vector<parParentesis> dif_ini;

      bool puntoBool = false;
      bool parParentesis = false;

      while(tiempoActual <= tiempoSalida){
      //while(contador < MAX_IT){
            cout << endl;
          tiempoActual = time(NULL);
            contador=contador+1;
          cout << "EJECUCION DE MCTS NUMERO: "<< contador << endl;
          cout << "Tiempo actual: "<< tiempoActual << " Tiempo salida:
"<< tiempoSalida << endl;
            //cout << endl;

            Node = rootNode;
            root->Clone(state);
            count_number = 0;
            count_number1 = 0;

            re=0, max_val = 0;

            paired_pos.clear();
            vector<positionStruct> posi;
            vector<positionStruct> posl;
            vector<positionStruct> poslalpha;
            vector<positionStruct> need;
            vector<int> pa;
            vector<int> upa;

            while(Node->untriedUBPP.empty()         ||         Node-
>untriedBPP.empty()){
                  Node = Node->SeleccionarNodo();
                  state->SelectPosition(Node->position, Node->pt);
            }

            if( !Node->untriedPositions.empty() ){
                  if(k > (vectorPuntos.size()-1) ){
                        if(Node->untriedBPP.size() == 6){
                              aleatorio  =  sRNG.GetUint()  %  Node-
>untriedPositions.size();
                              k = Node->untriedPositions[aleatorio];
                        }
                  }else{
                        if(Node->untriedUBPP.size() == 4){
```

```
                              aleatorio = sRNG.GetUint() % Node-
>untriedPositions.size();
                              k = Node->untriedPositions[aleatorio];
                        }
                  }
                  if(Node->untriedBPP.size() != 6 || Node-
>untriedUBPP.size() != 4){
                        if(Node->NodoHijo != NULL){
                              k = Node->NodoHijo->pt;
                        }
                  }
            }

            if(k > vectorPuntos.size()-1){
                  if( !Node->untriedBPP.empty() ){
                        aleatorio = sRNG.GetUint() % Node-
>untriedBPP.size();
                        int m = Node->untriedBPP[aleatorio];
                        Node->untriedBPP.erase(remove(Node-
>untriedBPP.begin(),      Node->untriedBPP.end(),      m),      Node-
>untriedBPP.end());
                        state->SelectPosition(m, k);
                        Node2 = new Nodo(m,k,Node, state);
                        Node->AnadirNodo(m, k, *Node2);
                        Node = Node2;
                  }
            }else{
                  if( !Node->untriedUBPP.empty() ){
                        aleatorio = sRNG.GetUint() % Node-
>untriedUBPP.size();
                        int m = Node->untriedUBPP[aleatorio];
                        Node->untriedUBPP.erase(remove(Node-
>untriedUBPP.begin(),      Node->untriedUBPP.end(),      m),      Node-
>untriedUBPP.end());
                        state->SelectPosition(m, k);
                        Node2 = new Nodo(m,k,Node, state);
                        Node->AnadirNodo(m, k, *Node2);
                        Node = Node2;
                  }
            }

            posi = state->positionStructVector;
            vector<positionStruct> goal = unirIndexYUindex();

            puntoBool = false;
            parParentesis = false;

            for(unsigned i=0; i<state->positionStructVector.size();
i++){
                  if(esPunto(i, goal)){  //Si es != -1, es una
posicion valida de punto. En la busqueda, goal[i] no valdrá -1 nunca.
                        int id = goal[i].punto;
                        auto pred = [id](const positionStruct & item)
{
                            return item.punto == id;
                        };
                         puntoBool = find_if(begin(posi), end(posi),
pred) != end(posi);
                         if(!puntoBool){
                               posl.push_back(goal[i]);
```

```
                        }else{
                              need.push_back(goal[i]);
                        }
                        puntoBool=false;
                  }else{
                        if(esParParentesis(i, goal)){
                              //Si no encuentro la apertura, no
contraré su par de cierre. Solo compruebo apertura.
                              int id = goal[i].apertura;
                              auto pred = [id](const positionStruct &
item) {
                                  return item.apertura == id;
                              };
                               parParentesis = find_if(begin(posi),
end(posi), pred) != end(posi);
                               if(!parParentesis){
                                     posl.push_back(goal[i]);
                               }else{
                                     need.push_back(goal[i]);
                               }
                               parParentesis=false;
                        }else{
                              cout << "Fallo en MCTS, no es ni punto
ni parParentesis" << endl;
                        }
                  }
                  if(hayBase(i, posi)){
                        poslalpha.push_back(posi[i]);    //Meto    la
estructura que contiene una base, ergo "no estaria" en goal, al ser
goal simplemente

//los enteros de las posiciones donde se encuentran los puntos y los
pares de parentesis.
                  }
      //Con poslalpha solo quiero quedarme las bases
            }//for


            eposl.clear();
            eposl111.clear();
            if( poslalpha.size() <= vectorParentesis.size() ){
                  //eposl- va a ser una lista de las bases
                  for(auto elemento : poslalpha){
                      string s = elemento.base;
                      char arr[s.length()];
                      strcpy(arr,s.c_str());
                      for(unsigned i = 0; i < s.length(); i++){
                        eposl.push_back(string(1,arr[i]));
                      }
                  }
            }else{
                  for(unsigned      i      =      poslalpha.size()-
vectorParentesis.size(); i<poslalpha.size(); i++){
                      string s = poslalpha[i].base;
                      char arr[s.length()];
                      strcpy(arr,s.c_str());
                      for(unsigned i = 0; i < s.length(); i++){
                        eposl111.push_back(string(1,arr[i]));
                      }
                  }
```

```
                //eposl1=list(itertools.chain(*eposl111))      esta
linea en python seria eposl111, ya que ya es una lista con las bases

                for(unsigned       i=0;       i<(poslalpha.size()-
vectorParentesis.size()); i++){
                    string s = poslalpha[i].base;
                    char arr[s.length()];
                    strcpy(arr,s.c_str());
                    for(unsigned i = 0; i < s.length(); i++){
                      eposl.push_back(string(1,arr[i]));
                    }
                }
                //sumar eposl111 a eposl
                for(string elemento : eposl111){
                      eposl.push_back(elemento);
                }
          }

          need_GC = calculate_GC_numbers(eposl, parametro_gc, need,
poslalpha);
          y=state->GetPositions();

          for(unsigned i=0; i<y.size(); i++){
                if(y[i]>vectorPuntos.size()-1){
                      pa.push_back(y[i]);
                }else{
                      upa.push_back(y[i]);
                }
          }
          //cout << endl;
          while(!pa.empty()){
                aleatorio = sRNG.GetUint() % pa.size();
                int cpa = pa[aleatorio];

                if(count_number<=need_GC){
                      state->SimulationGC(cpa);
                      count_number = count_number+1;
                      pa.erase(remove(pa.begin(),  pa.end(),  cpa),
pa.end());
                }else{
                      state->SimulationAU(cpa);
                      pa.erase(remove(pa.begin(),  pa.end(),  cpa),
pa.end());
                }
          }

          while(!upa.empty()){
                aleatorio = sRNG.GetUint() % upa.size();
                int ucpa = upa[aleatorio];

                if(count_number<=need_GC){
                      new_count=need_GC-count_number;
                      if(count_number1 <= (new_count*2)){
                            state->SimulationUnpairedGC(ucpa);
                            upa.erase(remove(upa.begin(),
upa.end(), ucpa), upa.end());
                            count_number1 = count_number1+1;
                      }else{
                            state->SimulationUnpairedAU(ucpa);
```

```
                              upa.erase(remove(upa.begin(),
upa.end(), ucpa), upa.end());
                        }
                  }else{
                        state->SimulationUnpairedAU(ucpa);
                        upa.erase(remove(upa.begin(),      upa.end(),
ucpa), upa.end());
                  }
            }

            vector<positionStruct> posbasep;
            vector<positionStruct> posbase;

            for(int i=vectorPuntos.size(); i<state->n; i++){
                  posbasep.push_back(state-
>positionStructVector[i]);
            }

            for(unsigned i=0; i<vectorPuntos.size(); i++){
                  posbase.push_back(state->positionStructVector[i]);
            }
            //e=list(itertools.chain(*posbasep)) posbasep lo tengo ya
como un vector, no hace falta transoformarlo a lista

          vector<string> e;
          for(positionStruct elemento : posbasep){
            for(unsigned i=0; i<elemento.base.size(); i++) {
                  string str(1, elemento.base[i]);
                  e.push_back(str);
            }
          }

            //base en "e"(posbasep) en la posicion c[i], lo inserto
en posbase en la posicion b[i]
            //Probado y corregido
            for(unsigned i=0; i<parentesisSeparados.size(); i++){
                  auto it = posbase.begin() + b[i];
                  positionStruct aux;
                  aux.base = e[c[i]];
                  posbase.insert(it, aux);
            }

            string mutated_s="";
            for(unsigned i=0; i<posbase.size(); i++){
                  mutated_s = mutated_s + posbase[i].base;
            }
            //cout << " mutated_s: "  << mutated_s <<endl;
            vector<string> ini_seq_pool;
            vector<double> ini_str_pool;
            vector<double> GC_pool;
            index_seq=0;
            //TODO si introduzco corchetes, tengo que  escribir esta
parte de codigo, si pk=1. De momento trabajo con pk=0
            if(parametro_pk == 1){

            }else{
                  calculate_mfe_and_str(mutated_s,      some_str_mfe,
some_str_value);
```

```
      some_str_distance=calculate_structure_distance(parametro_s,
parametro_s.size(), some_str_value);
            }
            ini_seq_pool.push_back(mutated_s);
            ini_str_pool.push_back(some_str_distance);
            GCnum=measureGC(mutated_s);
            GC_pool.push_back(GCnum);

            //cout<<endl;
            //cout<<                          "//////////////////Bucle
50//////////////////////: "<<endl;
            //cout<<endl;
            for(int i=0; i<50; i++){
                  //cout << "Ejecucion bucle: " << i << endl;

                  paired_pos.clear();

                  //devuelve paired_pos y dif_ini
                  dif_str(some_str_value, paired_pos, dif_ini);
                  //devuelve posbase cambiado(mutated_seq)
                  GCcontent(parametro_gc, GCnum, paired_pos, posbase,
posl);
                  //devuelve posbase modificado
                  check_GC_base3(dif_ini, posbase, posl);

                  string mutated_seq="";
                  for(positionStruct p : posbase){
                        mutated_seq = mutated_seq+p.base;
                  }
                  GCnum = measureGC(mutated_seq);

                  GC_pool.push_back(GCnum);

                  //TODO si introduzco corchetes, tengo que  escribir
esta parte de codigo, si pk=1. De momento trabajo con pk=0
                  if(parametro_pk == 1){

                  }else{
                        calculate_mfe_and_str(mutated_seq, notUsable,
kkk);

      new_str_distance=calculate_structure_distance(parametro_s,
parametro_s.size(), kkk);
                  }
                  some_str_value=kkk;
                  some_ini_seq=mutated_seq;
                  ini_seq_pool.push_back(mutated_seq);
                  ini_str_pool.push_back(new_str_distance);
                  index_seq = index_seq+1;
                  ggg=abs(parametro_gc-GCnum);

                  if(ini_str_pool[index_seq] == 1){
                        break;
                  }
            }
            //cout<<endl;
            //cout<<            "//////////////////FIN            DE
BUCLE//////////////////////: "<<endl;
            //cout<<endl;
```

```
            int                     maxElementIndex                  =
std::max_element(ini_str_pool.begin(),ini_str_pool.end())          -
ini_str_pool.begin();

            GCnew=GC_pool[maxElementIndex];
            max_val=ini_str_pool[maxElementIndex];
            seq=ini_seq_pool[index_seq];
            ggg=abs(parametro_gc-GCnum);
            gggg=abs(parametro_gc-GCnew);

            best_str_solution=seq;
            max_val_solution=max_val;
            GCC_solution=GCnum;

            if((ini_str_pool[index_seq] == 1) && (ggg<=parametro_d)){
                  break;
            }

            if(ini_str_pool[index_seq] == 1){
                  if(ggg<=0.01){
                        re=1+1;
                  }else{
                        re=1+0;
                  }
            }

            if(max_val<1){
                  if(gggg<=0.01){
                        re=1+max_val;
                  }else{
                        re=0+max_val;
                  }
            }

            bool salir = false;
            int j=0;
            //si nodopadre es null, estoy en el ultimo nodo(osea, el
primero del arbol)
            while(!salir){
                  if(Node->NodoPadre == NULL){
                        salir = true;
                  }
                j++;
                //cout << "---------WHILE--------- "<< j << endl;
                  Node->ActualizarNodo(re);
                  Node = Node->NodoPadre;
            }

      }//primer while - tiempo

      //devolver seq, max_val y GCnum
}//MCTS()

void MCTSnoGC(RNAestructura *root, int k, string &best_str_solution,
float &max_val_solution, double &GCC_solution){
    time_t tiempoActual = time(NULL);
    time_t tiempoSalida= tiempoActual+(60*10);

    //Numero de iteraciones del bucle principal
```

```
    int itermax = 100000;

    cout << "DENTRO DE MCTSnoGC" << endl;

      Nodo *rootNode = new Nodo(-1, -1, NULL, root), *Node, *Node2;
      RNAestructura *state = new RNAestructura();
      root->Clone(state);

      int aleatorio = -1;

      vector<string> eposl;
      vector<string> eposl111;
      vector<int> y;
      vector<parParentesis> paired_pos;
      vector<parParentesis> dif_ini;

      int index_seq=0, contador=1;
      string some_str_value="", kkk, some_ini_seq, seq;
      double some_str_distance, re, new_str_distance, max_idx, GCnew,
max_val, GCnum, GCnum2;
      float notUsable;
      bool puntoBool = false;
      bool parParentesis = false;

      for(int z=0; z<itermax; z++){
            cout << endl;
          tiempoActual = time(NULL);
          cout << "EJECUCION DE MCTS_NO_GC NUMERO: "<< contador <<
endl;
          cout << "Tiempo actual: "<< tiempoActual << " Tiempo salida:
"<< tiempoSalida << endl;
            contador=contador+1;

            if(tiempoActual >= tiempoSalida){
                  break;
            }

            Node = rootNode;
            root->Clone(state);
            vector<positionStruct> posi;
            vector<positionStruct> posl;

            while(Node->untriedUBPP.empty()          ||          Node-
>untriedBPP.empty()){
                  Node = Node->SeleccionarNodo();
                  state->SelectPosition(Node->position, Node->pt);
            }

            if( !Node->untriedPositions.empty() ){
                  if(k > (vectorPuntos.size()-1) ){
                        if(Node->untriedBPP.size() == 6){
                              aleatorio  =  sRNG.GetUint()  %  Node-
>untriedPositions.size();
                              k = Node->untriedPositions[aleatorio];
                        }
                  }else{
                        if(Node->untriedUBPP.size() == 4){
                              aleatorio  =  sRNG.GetUint()  %  Node-
>untriedPositions.size();
                              k = Node->untriedPositions[aleatorio];
```

```
                        }
                  }
            }

            if(k > vectorPuntos.size()-1){
                  if( !Node->untriedBPP.empty() ){
                        aleatorio    =    sRNG.GetUint()    %    Node-
>untriedBPP.size();
                        int m = Node->untriedBPP[aleatorio];
                        Node->untriedBPP.erase(remove(Node-
>untriedBPP.begin(),      Node->untriedBPP.end(),      m),      Node-
>untriedBPP.end());
                        state->SelectPosition(m, k);
                        Node2 = new Nodo(m,k,Node, state);
                        Node->AnadirNodo(m, k, *Node2);
                        Node = Node2;
                  }
            }else{
                  if( !Node->untriedUBPP.empty() ){
                        aleatorio    =    sRNG.GetUint()    %    Node-
>untriedUBPP.size();
                        int m = Node->untriedUBPP[aleatorio];
                        Node->untriedUBPP.erase(remove(Node-
>untriedUBPP.begin(),     Node->untriedUBPP.end(),     m),     Node-
>untriedUBPP.end());
                        state->SelectPosition(m, k);
                        Node2 = new Nodo(m,k,Node, state);
                        Node->AnadirNodo(m, k, *Node2);
                        Node = Node2;
                  }
            }

            posi = state->positionStructVector;
            vector<positionStruct> goal = unirIndexYUindex();

            puntoBool = false;
            parParentesis = false;

            for(unsigned  i=0;  i<state->positionStructVector.size();
i++){

                  if(esPunto(i, goal)){
                        int id = goal[i].punto;
                        auto pred = [id](const positionStruct & item)
{
                            return item.punto == id;
                        };
                         puntoBool = find_if(begin(posi), end(posi),
pred) != end(posi);
                         if(!puntoBool){
                               posl.push_back(goal[i]);
                         }
                         puntoBool=false;
                  }else{
                        if(esParParentesis(i, goal)){
                              //Si  no  encuentro  la  apertura,  no
contraré su par de cierre. Solo compruebo apertura.
                              int id = goal[i].apertura;
                              auto pred = [id](const positionStruct &
item) {
```

```
                                    return item.apertura == id;
                                };
                                parParentesis  =  find_if(begin(posi),
end(posi), pred) != end(posi);
                                if(!parParentesis){
                                      posl.push_back(goal[i]);
                                }
                                parParentesis=false;
                          }else{
                                cout << "Fallo en MCTS, no es ni punto
ni parParentesis" << endl;
                          }
                    }
              }//for

              while (!state->GetPositions().empty()){
                    aleatorio     =     sRNG.GetUint()     %     state-
>GetPositions().size();
                    state->Simulation(state-
>GetPositions()[aleatorio]);
              }

              vector<positionStruct> posbasep;
              vector<positionStruct> posbase;

              for(unsigned i=vectorPuntos.size(); i<state->n; i++){
                    posbasep.push_back(state-
>positionStructVector[i]);
              }

              for(unsigned i=0; i<vectorPuntos.size(); i++){
                    posbase.push_back(state->positionStructVector[i]);
              }
              //e=list(itertools.chain(*posbasep)) posbasep lo tengo ya
como un vector, no hace falta transoformarlo a lista

            vector<string> e;
            for(positionStruct elemento : posbasep){
              for(unsigned i=0; i<elemento.base.size(); i++) {
                    string str(1, elemento.base[i]);
                    e.push_back(str);
              }
            }

              //base en "e"(posbasep) en la posicion c[i], lo inserto
en posbase en la posicion b[i]
              //Probado y corregido
              for(unsigned i=0; i<parentesisSeparados.size(); i++){
                    auto it = posbase.begin() + b[i];
                    positionStruct aux;
                    aux.base = e[c[i]];
                    posbase.insert(it, aux);
              }

              string mutated_s="";
              for(unsigned i=0; i<posbase.size(); i++){
                    mutated_s = mutated_s + posbase[i].base;
              }

              vector<string> ini_seq_pool;
```

```
            vector<double> ini_str_pool;
            vector<double> GC_pool;
            index_seq=0;
            //TODO si introduzco corchetes, tengo que  escribir esta
parte de codigo, si pk=1. De momento trabajo con pk=0
            if(parametro_pk == 1){

            }else{
                  float notUsable;
                  calculate_mfe_and_str(mutated_s, notUsable, kkk);
            }

            some_str_value = kkk;

      some_str_distance=calculate_structure_distance(parametro_s,
parametro_s.size(), some_str_value);
            ini_seq_pool.push_back(mutated_s);
            ini_str_pool.push_back(some_str_distance);
            GCnum=measureGC(mutated_s);
            GC_pool.push_back(GCnum);

            //cout<<endl;
            //cout<<                          "/////////////////Bucle
50//////////////////////: "<<endl;
            //cout<<endl;

            for(int i=0; i<50; i++){
                  //cout << "Ejecucion bucle: " << i << endl;

                  paired_pos.clear();

                  //devuelve paired_pos y dif_ini
                  dif_str(some_str_value, paired_pos, dif_ini);
                  //devuelve posbase cambiado(mutated_seq)
                  check_seq_base(parametro_gc,   GCnum,   paired_pos,
posbase, posl);
                  //devuelve posbase modificado
                  check_GC_base(dif_ini, posbase, posl);

                  string mutated_seq="";
                  for(positionStruct p : posbase){
                        mutated_seq = mutated_seq+p.base;
                  }
                  GCnum2 = measureGC(mutated_seq);
                  GC_pool.push_back(GCnum2);

                  //TODO si introduzco corchetes, tengo que  escribir
esta parte de codigo, si pk=1. De momento trabajo con pk=0
                  if(parametro_pk == 1){

                  }else{
                        calculate_mfe_and_str(mutated_seq, notUsable,
kkk);
                  }

      new_str_distance=calculate_structure_distance(parametro_s,
parametro_s.size(), kkk);
                  some_str_value=kkk;
                  some_ini_seq=mutated_seq;
                  ini_seq_pool.push_back(mutated_seq);
```

```
                ini_str_pool.push_back(new_str_distance);
                index_seq = index_seq+1;
                if(ini_str_pool[index_seq] == 1){
                      break;
                }
                //cout<<endl;
          }//for <50

          //cout<<endl;
          //cout<<            "//////////////////FIN            DE
BUCLE//////////////////////: "<<endl;
          //cout<<endl;

          int                    maxElementIndex                    =
std::max_element(ini_str_pool.begin(),ini_str_pool.end())          -
ini_str_pool.begin();

          GCnew=GC_pool[maxElementIndex];
          max_val=ini_str_pool[maxElementIndex];
          seq=ini_seq_pool[index_seq];

          best_str_solution=seq;
          max_val_solution=max_val;
          GCC_solution=GCnew;

          if(ini_str_pool[index_seq] == 1){
                break;
          }

          if(max_val<1){
                re=max_val;
          }

          bool salir = false;
          int j=0;
          //si nodopadre es null, estoy en el ultimo nodo(osea, el
primero del arbol)
          while(!salir){
                if(Node->NodoPadre == NULL){
                      salir = true;
                }
              j++;
              //cout << "---------WHILE--------- "<< j << endl;
                Node->ActualizarNodo(re);
                Node = Node->NodoPadre;
          }

    }//primer while - tiempo

    //devolver seq, max_val y GCnum
}

int calculate_GC_numbers(vector<string> eposl, float parametro_gc,
vector<positionStruct> need, vector<positionStruct> poslalpha){
    int cunnt=0;
    int real = 0;
    for(unsigned i=0; i<eposl.size(); i++){
          if(eposl[i]=="G" || eposl[i]=="C"){
                cunnt=cunnt+1;
          }
```

```
      }//for

      int needGC = (parametro_s.size()*parametro_gc) - cunnt;

      if(needGC > 0){
            real = ceil((float)needGC/2);
      }else{
            real = 0;
      }
      return real;
}

void calculate_mfe_and_str(string sequence, float &mfe, string
&str_v){

    char seq[sequence.size()];
    strcpy(seq,sequence.c_str());

    /* allocate memory for MFE structure (length + 1) */
    char  *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq)
+ 1));

    /* predict Minmum Free Energy and corresponding secondary
structure */
    mfe = vrna_fold(seq, structure);
    string copia(structure);
    str_v = copia;

    /* print sequence, structure and MFE */
    //todo
    //cout<<"sequence(mutated_s): "<< sequence <<" MFE(some_str_mfe):
"<<mfe<<" str_v(structure)(some_str_value): "<<str_v<<endl;

    /* cleanup memory */
    free(structure);
}

//structure_s = parametro_s ---Secuencia original
double   calculate_structure_distance(string   structure_s,   float
str_length, string some_str_value){
      double sdt=0;
      int sd=0;

      for(unsigned i=0; i<structure_s.size(); i++){
            if(some_str_value[i]!=structure_s[i]){
                  sd=sd+1;
            }
      }

      sdt=(str_length-(double)sd)/str_length;

      return sdt;
}

double measureGC(string generate_seq){
      int n=generate_seq.size();
      int cont=0;
      vector<int> indexGC;
      vector<int> indexnotGC;
      double getGC=0;
```

```
	for(unsigned i=0; i<generate_seq.size(); i++){
		if(generate_seq[i]=='C'){
			cont=cont+1;
			indexGC.push_back(i);
		}
		if(generate_seq[i]=='G'){
			cont=cont+1;
			indexGC.push_back(i);
		}
		if(generate_seq[i]!='C' && generate_seq[i]!='G'){
			indexnotGC.push_back(i);
		}
	}
	getGC=(double)cont/n;
	return getGC;
}


void	dif_str(string	&some_str_value,	vector<parParentesis>
&save_paired_pos, vector<parParentesis> &dif_pos_ini){
	vector<parParentesis> paired;
	vector<int> unpaired;
	bool seEncuentra = false;

	calculate_sequence_position(some_str_value, paired, unpaired);

	for(unsigned i=0; i<paired.size(); i++){
		for(parParentesis par : vectorParentesis){
			if(	(par.apertura	==	paired[i].apertura)	&&
(par.cierre == paired[i].cierre)){
				seEncuentra = true;
				break;
			}
		}
		if(!seEncuentra){
			save_paired_pos.push_back(paired[i]);
		}else{
			seEncuentra = false;
		}
	}

	dif_pos_ini.clear();

	seEncuentra = false;
	for(unsigned i=0; i<vectorParentesis.size(); i++){
		for(parParentesis par : paired){
			if( (par.apertura == vectorParentesis[i].apertura)
&& (par.cierre == vectorParentesis[i].cierre)){
				seEncuentra = true;
				break;
			}
		}
		if(!seEncuentra){
			dif_pos_ini.push_back(vectorParentesis[i]);
		}else{
			seEncuentra = false;
		}
	}
}
```

```
void calculate_sequence_position(string &seq, vector<parParentesis>
&struc, vector<int> &ustruc){
	vector<int> stack;
	int ultimo;
	parParentesis par;

	for(unsigned i=0; i<seq.size(); i++){
		if(seq[i] == '('){ stack.push_back(i);}
		if(seq[i] == ')'){
			ultimo = stack.back();
			par.apertura = ultimo;
			stack.pop_back();
			par.cierre = i;
			struc.push_back(par);
		}else if(seq[i] == '.'){ ustruc.push_back(i); }
	}
}

void	GCcontent(float	parametro_gc,	double	getGC,
vector<parParentesis>	&some_paired_pos,	vector<positionStruct>
&predicted_seq, vector<positionStruct> &posl){
	vector<string> check_even;
	vector<string> check_odd;
	vector<string> A_change = {"G", "C"};
	vector<string> C_change = {"A", "U"};
	vector<string> U_change = {"U", "C"};
	vector<string> G_change = {"A", "G"};
	vector<int> parentesisSeparados;
	vector<int> Par;
	vector<int> imPar;
	vector<int> nuevo_Par;
	vector<int> nuevo_imPar;
	bool seEncuentraPar = false;
	bool seEncuentraImpar = false;
	int aleatorio;

	SepararParentesis(some_paired_pos, parentesisSeparados);

	//Coger pares e impares
	for(unsigned i=0; i<parentesisSeparados.size(); i++){
		if (i%2 == 0){
			Par.push_back(parentesisSeparados[i]);
	    }else{
			imPar.push_back(parentesisSeparados[i]);
	    }
	}

	if(!posl.empty()){

	for(unsigned i=0; i<Par.size(); i++){

		for(unsigned j=0; j<posl.size(); j++){
			if(Par[i] == posl[j].punto){
				seEncuentraPar = true;
			}
			if(imPar[i] == posl[j].punto){
				seEncuentraImpar = true;
			}
		}
```

```
            if(!seEncuentraPar){
                  nuevo_Par.push_back(Par[i]);
                  nuevo_imPar.push_back(imPar[i]);
            }else{
                  seEncuentraPar = false;
            }
            if(!seEncuentraImpar){
                  nuevo_Par.push_back(Par[i]);
                  nuevo_imPar.push_back(imPar[i]);
            }else{
                  seEncuentraImpar = false;
            }
      }//for len(par)
      }//posl.empty

      for(unsigned i=0; i<nuevo_imPar.size(); i++){
            check_even.push_back(predicted_seq[nuevo_Par[i]].base);
            check_odd.push_back(predicted_seq[nuevo_imPar[i]].base);
      }

      for (unsigned i = 0; i < nuevo_imPar.size(); i++) {
            if (getGC < parametro_gc) {
                  if (predicted_seq[nuevo_imPar[i]].base == "A") {
                        aleatorio = sRNG.GetUint() % A_change.size();
                        predicted_seq[nuevo_Par[i]].base            =
A_change[aleatorio];
                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"U") {
                        predicted_seq[nuevo_Par[i]].base = "C";
                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"C") {
                        predicted_seq[nuevo_Par[i]].base = "C";
                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"G") {
                        predicted_seq[nuevo_Par[i]].base = "G";
                  }
            } else {
                  if (predicted_seq[nuevo_imPar[i]].base == "A") {
                        predicted_seq[nuevo_Par[i]].base = "A";
                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"U") {
                        predicted_seq[nuevo_Par[i]].base = "U";
                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"C") {
                        aleatorio = sRNG.GetUint() % C_change.size();
                        predicted_seq[nuevo_Par[i]].base            =
C_change[aleatorio];
                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"G") {
                        predicted_seq[nuevo_Par[i]].base = "A";
                  }
            }
      }
}

void     check_seq_base(float     parametro_gc,     double     getGC,
vector<parParentesis>    &some_paired_pos,    vector<positionStruct>
&predicted_seq, vector<positionStruct> &posl){
      vector<string> check_even;
```

```
vector<string> check_odd;
vector<string> A_change = {"G", "C"};
vector<string> C_change = {"A", "U"};
vector<string> U_change = {"U", "C"};
vector<string> G_change = {"A", "G"};
vector<int> parentesisSeparados;
vector<int> Par;
vector<int> imPar;
vector<int> nuevo_Par;
vector<int> nuevo_imPar;
bool seEncuentraPar = false;
bool seEncuentraImpar = false;
int aleatorio;

SepararParentesis(some_paired_pos, parentesisSeparados);

//Coger pares e impares
for(unsigned i=0; i<parentesisSeparados.size(); i++){
    if (i%2 == 0){
            Par.push_back(parentesisSeparados[i]);
    }else{
            imPar.push_back(parentesisSeparados[i]);
    }
}

if(!posl.empty()){

for(unsigned i=0; i<Par.size(); i++){

      for(unsigned j=0; j<posl.size(); j++){
            if(Par[i] == posl[j].punto){
                  seEncuentraPar = true;
            }
            if(imPar[i] == posl[j].punto){
                  seEncuentraImpar = true;
            }
      }

      if(!seEncuentraPar){
            nuevo_Par.push_back(Par[i]);
            nuevo_imPar.push_back(imPar[i]);
      }else{
            seEncuentraPar = false;
      }
      if(!seEncuentraImpar){
            nuevo_Par.push_back(Par[i]);
            nuevo_imPar.push_back(imPar[i]);
      }else{
            seEncuentraImpar = false;
      }
}//for len(par)
}//posl.empty


for(unsigned i=0; i<nuevo_imPar.size(); i++){
      check_even.push_back(predicted_seq[nuevo_Par[i]].base);
      check_odd.push_back(predicted_seq[nuevo_imPar[i]].base);
}

for (unsigned i = 0; i < nuevo_imPar.size(); i++) {
```

```
            if (getGC < parametro_gc) {
                  if (predicted_seq[nuevo_imPar[i]].base == "A") {
                        aleatorio = sRNG.GetUint() % A_change.size();
                        predicted_seq[nuevo_Par[i]].base           =
A_change[aleatorio];

                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"U") {
                        aleatorio = sRNG.GetUint() % U_change.size();
                        predicted_seq[nuevo_Par[i]].base           =
U_change[aleatorio];

                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"C") {
                        aleatorio = sRNG.GetUint() % C_change.size();
                        predicted_seq[nuevo_Par[i]].base           =
C_change[aleatorio];

                  } else if (predicted_seq[nuevo_imPar[i]].base ==
"G") {
                        aleatorio = sRNG.GetUint() % G_change.size();
                        predicted_seq[nuevo_Par[i]].base           =
G_change[aleatorio];
                  }
            }
      }

}


void                              check_GC_base3(vector<parParentesis>
&some_dif_ini,vector<positionStruct>                  &predicted_seq,
vector<positionStruct> &posl){
      vector<string> GC = {"G", "C"};
      vector<string> AU = {"A", "U"};

      string aux="";
      for(positionStruct p : predicted_seq){
            aux = aux+p.base;
      }
      double newGC = measureGC(aux);
      vector<parParentesis> new_dif_ini;
      bool parParentesis = false;
      vector<int> parentesisSeparados_;//a1
      vector<int> Par;
      vector<int> imPar;
      int aleatorio;

      for(unsigned i=0; i<some_dif_ini.size(); i++){
            //Si  no  encuentro  la  apertura,  no  contraré  su  par  de
cierre. Solo compruebo apertura.
            int id = some_dif_ini[i].apertura;
            auto pred = [id](const positionStruct & item) {
                return item.apertura == id;
            };
             parParentesis = find_if(begin(posl), end(posl), pred) !=
end(posl);
             if(!parParentesis){
                   new_dif_ini.push_back(some_dif_ini[i]);
             }
```

```
	}

	SepararParentesis(new_dif_ini, parentesisSeparados_);

	//Coger pares e impares
	for(unsigned i=0; i<parentesisSeparados_.size(); i++){
	    if (i%2 == 0){
	            Par.push_back(parentesisSeparados_[i]);
	    }else{
	            imPar.push_back(parentesisSeparados_[i]);
	    }
	}

	for (unsigned i = 0; i < imPar.size(); i++) {
	      if (parametro_gc>=newGC) {
	            aleatorio = sRNG.GetUint() % GC.size();
	            predicted_seq[imPar[i]].base = GC[aleatorio];
	            if (predicted_seq[imPar[i]].base == "G") {
	                  predicted_seq[Par[i]].base = "C";
	            } else if (predicted_seq[imPar[i]].base == "C") {
	                  predicted_seq[Par[i]].base = "G";
	            }
	      } else {
	            aleatorio = sRNG.GetUint() % AU.size();
	            predicted_seq[imPar[i]].base = AU[aleatorio];
	            if (predicted_seq[imPar[i]].base == "A") {
	                  predicted_seq[Par[i]].base = "U";
	            } else if (predicted_seq[imPar[i]].base == "U") {
	                  predicted_seq[Par[i]].base = "A";
	            }
	      }
	}

}

void                              check_GC_base(vector<parParentesis>
&some_dif_ini,vector<positionStruct>                 &predicted_seq,
vector<positionStruct> &posl){
	vector<string> GC = {"G", "C"};

	vector<parParentesis> new_dif_ini;
	bool parParentesis = false;
	vector<int> parentesisSeparados;//a1
	vector<int> Par;
	vector<int> imPar;
	int aleatorio;

	for(unsigned i=0; i<some_dif_ini.size(); i++){
	      //Si  no encuentro  la  apertura,  no  contraré  su  par  de
cierre. Solo compruebo apertura.
	      int id = some_dif_ini[i].apertura;
	      auto pred = [id](const positionStruct & item) {
	          return item.apertura == id;
	      };
	       parParentesis = find_if(begin(posl), end(posl), pred) !=
end(posl);
	       if(!parParentesis){
	             new_dif_ini.push_back(some_dif_ini[i]);
	       }
	}
```

```
      SepararParentesis(new_dif_ini, parentesisSeparados);
      //Coger pares e impares
      for(unsigned i=0; i<parentesisSeparados.size(); i++){
          if (i%2 == 0){
                  Par.push_back(parentesisSeparados[i]);
          }else{
                  imPar.push_back(parentesisSeparados[i]);
          }
      }

      for (unsigned i = 0; i < imPar.size(); i++) {
                  aleatorio = sRNG.GetUint() % GC.size();
                  predicted_seq[imPar[i]].base = GC[aleatorio];
                  if (predicted_seq[imPar[i]].base == "G") {
                        predicted_seq[Par[i]].base = "C";
                  } else if (predicted_seq[imPar[i]].base == "C") {
                        predicted_seq[Par[i]].base = "G";
                  }
      }
}


vector<positionStruct> unirIndexYUindex(){
      positionStruct posStruct;
      vector<positionStruct> Union;

    for (parParentesis par : vectorParentesis) {
        posStruct.apertura = par.apertura;
        posStruct.cierre = par.cierre;
            Union.push_back(posStruct);

        posStruct.apertura = -1;
        posStruct.cierre = -1;
    }

      for(unsigned i=0; i<vectorPuntos.size(); i++){
            posStruct.punto = vectorPuntos[i];
            Union.push_back(posStruct);

            posStruct.punto = -1;
      }

    return Union;
}

bool esPunto(int numPosition, vector<positionStruct> vector){
      if(vector[numPosition].punto != -1){
            return true;
      }
      return false;
}

bool esParParentesis(int numPosition, vector<positionStruct> vector){
      if(      (vector[numPosition].apertura      !=      -1)      &&
(vector[numPosition].cierre != -1) ){
            return true;
      }
      return false;
}
```

```
bool hayBase(int numPosition, vector<positionStruct> vector){
      if(vector[numPosition].base != ""){
            return true;
      }
      return false;
}
```

SimpleRNG.h, para la generación de números aleatorios.

```
#ifndef SIMPLERNG_H_
#define SIMPLERNG_H_

// A simple random number generator based on George Marsaglia's MWC
(Multiply With Carry) generator.
// This is not intended to take the place of the library's primary
generator, Mersenne Twister.
// Its primary benefit is that it is simple to extract its state.

class SimpleRNG
{
public:

    SimpleRNG();

    // Seed the random number generator
    void SetState(unsigned int u, unsigned int v);

    // Extract the internal state of the generator
    void GetState(unsigned int& u, unsigned int& v);

    // A uniform random sample from the open interval (0, 1)
    double GetUniform();

    // A uniform random sample from the set of unsigned integers
    unsigned int GetUint();

    // This stateless version makes it more convenient to get a uniform
    // random value and transfer the state in and out in one operation.
    double GetUniform(unsigned int& u, unsigned int& v);

    // This stateless version makes it more convenient to get a random
unsigned integer
    // and transfer the state in and out in one operation.
    unsigned int GetUint(unsigned int& u, unsigned int& v);

    // Normal (Gaussian) random sample
    double GetNormal(double mean, double standardDeviation);

    // Exponential random sample
    double GetExponential(double mean);

      // Gamma random sample
    double GetGamma(double shape, double scale);

      // Chi-square sample
    double GetChiSquare(double degreesOfFreedom);

      // Inverse-gamma sample
    double GetInverseGamma(double shape, double scale);

      // Weibull sample
    double GetWeibull(double shape, double scale);

      // Cauchy sample
    double GetCauchy(double median, double scale);

      // Student-t sample
```

```
    double GetStudentT(double degreesOfFreedom);

    // The Laplace distribution is also known as the double
exponential distribution.
    double GetLaplace(double mean, double scale);

      // Log-normal sample
    double GetLogNormal(double mu, double sigma);

      // Beta sample
    double GetBeta(double a, double b);

      // Poisson sample
      int GetPoisson(double lambda);

private:
    unsigned int m_u, m_v;
      int PoissonLarge(double lambda);
      int PoissonSmall(double lambda);
      double LogFactorial(int n);
};

#endif /* SIMPLERNG_H_ */
```

SimpleRNG.cpp

```
#include <cmath>
#include <stdexcept>
#include <sstream>
#include <iostream>
#include "SimpleRNG.h"

#define PI 3.14159265358979323846264338327795

SimpleRNG::SimpleRNG()
{
    // These values are not magical, just the default values Marsaglia
used.
    // Any unit should work.
    m_u = 521288629;
    m_v = 362436069;
}

void SimpleRNG::SetState(unsigned int u, unsigned int v)
{
    m_u = u;
    m_v = v;
}

void SimpleRNG::GetState(unsigned int& u, unsigned int& v)
{
    u = m_u;
    v = m_v;
}

double SimpleRNG::GetUniform(unsigned int& u, unsigned int& v)
{
    // 0 <= u <= 2^32
    unsigned int z = GetUint(u, v);
    // The magic number is 1/(2^32 + 1) and so result is positive and
less than 1.
    return z*2.328306435996595e-10;
}

unsigned int SimpleRNG::GetUint(unsigned int& u, unsigned int& v)
{
    v = 36969*(v & 65535) + (v >> 16);
    u = 18000*(u & 65535) + (u >> 16);
    return (v << 16) + u;
}

double SimpleRNG::GetUniform()
{
    return GetUniform(m_u, m_v);
}

unsigned int SimpleRNG::GetUint()
{
    return GetUint(m_u, m_v);
}

// Get normal (Gaussian) random sample with specified mean and
standard deviation
```

```
double    SimpleRNG::GetNormal(double    mean    =    0.0,    double
standardDeviation = 1.0)
{
    if (standardDeviation <= 0.0)
    {
            std::stringstream os;
        os << "Standard deviation must be positive." << "\n"
           << "Received standard deviation " << standardDeviation;
        throw std::invalid_argument( os.str() );
    }
      // Use Box-Muller algorithm
    double u1 = GetUniform();
    double u2 = GetUniform();
    double r = sqrt( -2.0*log(u1) );
    double theta = 2.0*PI*u2;
    return mean + standardDeviation*r*sin(theta);
}

// Get exponential random sample with specified mean
double SimpleRNG::GetExponential(double mean = 1.0)
{
    if (mean <= 0.0)
    {
            std::stringstream os;
        os << "Exponential mean must be positive." << "\n"
           << "Received standard deviation " << mean;
        throw std::invalid_argument( os.str() );
    }
    return -mean*log( GetUniform() );
}

double SimpleRNG::GetWeibull(double shape, double scale)
{
    if (shape <= 0.0 || scale <= 0.0)
    {
            std::stringstream os;
        os << "Shape and scale parameters must be positive." << "\n"
           << "Received shape " << shape << " and scale " << scale;
        throw std::invalid_argument( os.str() );
    }
    return scale * pow(-log(GetUniform()), 1.0 / shape);
}

double SimpleRNG::GetCauchy(double median, double scale)
{
    if (scale <= 0)
    {
            std::stringstream os;
            os << "Shape parameter must be positive." << "\n"
           << "Received shape parameter " << scale;
        throw std::invalid_argument( os.str() );
    }

    double p = GetUniform();

    // Apply inverse of the Cauchy distribution function to a uniform
    return median + scale*tan(PI*(p - 0.5));
}
```

```
// The Laplace distribution is also known as the double exponential
distribution.
double SimpleRNG::GetLaplace(double mean, double scale)
{
    double u = GetUniform();
    return (u < 0.5) ?
        mean + scale*log(2.0*u) :
        mean - scale*log(2*(1-u));
}

double SimpleRNG::GetLogNormal(double mu, double sigma)
{
    return exp(GetNormal(mu, sigma));
}

int SimpleRNG::GetPoisson(double lambda)
{
    return    (lambda    <    30.0)    ?    PoissonSmall(lambda)    :
PoissonLarge(lambda);
}

int SimpleRNG::PoissonSmall(double lambda)
{
    // Algorithm due to Donald Knuth, 1969.
    double p = 1.0, L = exp(-lambda);
      int k = 0;
      do
      {
            k++;
            p *= GetUniform();
      }
      while (p > L);
    return k - 1;
}

int SimpleRNG::PoissonLarge(double lambda)
{
      // "Rejection method PA" from "The Computer Generation of
Poisson Random Variables" by A. C. Atkinson
      // Journal of the Royal Statistical Society Series C (Applied
Statistics) Vol. 28, No. 1. (1979)
      // The article is on pages 29-35. The algorithm given here is
on page 32.

      double c = 0.767 - 3.36/lambda;
      double beta = PI/sqrt(3.0*lambda);
      double alpha = beta*lambda;
      double k = log(c) - lambda - log(beta);

      for(;;)
      {
            double u = GetUniform();
            double x = (alpha - log((1.0 - u)/u))/beta;
            int n = (int) floor(x + 0.5);
            if (n < 0)
                  continue;
            double v = GetUniform();
            double y = alpha - beta*x;
        double temp = 1.0 + exp(y);
            double lhs = y + log(v/(temp*temp));
```

```
            double rhs = k + n*log(lambda) - LogFactorial(n);
            if (lhs <= rhs)
                  return n;
      }
}

double SimpleRNG::LogFactorial(int n)
{

    if (n < 0)
    {
        std::stringstream os;
        os << "Invalid input argument (" << n
        << "); may not be negative";
        throw std::invalid_argument( os.str() );

    }
    else if (n > 254)
    {
        double x = n + 1;
        return (x - 0.5)*log(x) - x + 0.5*log(2*PI) + 1.0/(12.0*x);
    }
    else
    {
        double lf[] =
        {
            0.000000000000000,
            0.000000000000000,
            0.693147180559945,
            1.791759469228055,
            3.178053830347946,
            4.787491742782046,
            6.579251212010101,
            8.525161361065415,
            10.604602902745251,
            12.801827480081469,
            15.104412573075516,
            17.502307845873887,
            19.987214495661885,
            22.552163853123421,
            25.191221182738683,
            27.899271383840894,
            30.671860106080675,
            33.505073450136891,
            36.395445208033053,
            39.339884187199495,
            42.335616460753485,
            45.380138898476908,
            48.471181351835227,
            51.606675567764377,
            54.784729398112319,
            58.003605222980518,
            61.261701761002001,
            64.557538627006323,
            67.889743137181526,
            71.257038967168000,
            74.658236348830158,
            78.092223553315307,
            81.557959456115029,
            85.054467017581516,
```

```
88.580827542197682,
92.136175603687079,
95.719694542143202,
99.330612454787428,
102.968198614513810,
106.631760260643450,
110.320639714757390,
114.034211781461690,
117.771881399745060,
121.533081515438640,
125.317271149356880,
129.123933639127240,
132.952575035616290,
136.802722637326350,
140.673923648234250,
144.565743946344900,
148.477766951773020,
152.409592584497350,
156.360836303078800,
160.331128216630930,
164.320112263195170,
168.327445448427650,
172.352797139162820,
176.395848406997370,
180.456291417543780,
184.533828861449510,
188.628173423671600,
192.739047287844900,
196.866181672889980,
201.009316399281570,
205.168199482641200,
209.342586752536820,
213.532241494563270,
217.736934113954250,
221.956441819130360,
226.190548323727570,
230.439043565776930,
234.701723442818260,
238.978389561834350,
243.268849002982730,
247.572914096186910,
251.890402209723190,
256.221135550009480,
260.564940971863220,
264.921649798552780,
269.291097651019810,
273.673124285693690,
278.067573440366120,
282.474292687630400,
286.893133295426990,
291.323950094270290,
295.766601350760600,
300.220948647014100,
304.686856765668720,
309.164193580146900,
313.652829949878990,
318.152639620209300,
322.663499126726210,
327.185287703775200,
331.717887196928470,
```

```
336.261181979198450,
340.815058870798960,
345.379407062266860,
349.954118040770250,
354.539085519440790,
359.134205369575340,
363.739375555563470,
368.354496072404690,
372.979468885689020,
377.614197873918670,
382.258588773060010,
386.912549123217560,
391.575988217329610,
396.248817051791490,
400.930948278915760,
405.622296161144900,
410.322776526937280,
415.032306728249580,
419.750805599544780,
424.478193418257090,
429.214391866651570,
433.959323995014870,
438.712914186121170,
443.475088120918940,
448.245772745384610,
453.024896238496130,
457.812387981278110,
462.608178526874890,
467.412199571608080,
472.224383926980520,
477.044665492585580,
481.872979229887900,
486.709261136839360,
491.553448223298010,
496.405478487217580,
501.265290891579240,
506.132825342034830,
511.008022665236070,
515.890824587822520,
520.781173716044240,
525.679013515995050,
530.584288294433580,
535.496943180169520,
540.416924105997740,
545.344177791154950,
550.278651724285620,
555.220294146894960,
560.169054037273100,
565.124881094874350,
570.087725725134190,
575.057539024710200,
580.034272767130800,
585.017879388839220,
590.008311975617860,
595.005524249382010,
600.009470555327430,
605.020105849423770,
610.037385686238740,
615.061266207084940,
620.091704128477430,
```

```
625.128656730891070,
630.172081847810200,
635.221937855059760,
640.278183660408100,
645.340778693435030,
650.409682895655240,
655.484856710889060,
660.566261075873510,
665.653857411105950,
670.747607611912710,
675.847474039736880,
680.953419513637530,
686.065407301994010,
691.183401114410800,
696.307365093814040,
701.437263808737160,
706.573062245787470,
711.714725802289990,
716.862220279103440,
722.015511873601330,
727.174567172815840,
732.339353146739310,
737.509837141777440,
742.685986874351220,
747.867770424643370,
753.055156230484160,
758.248113081374300,
763.446610112640200,
768.650616799717000,
773.860102952558460,
779.075038710167410,
784.295394535245690,
789.521141208958970,
794.752249825813460,
799.988691788643450,
805.230438803703120,
810.477462875863580,
815.729736303910160,
820.987231675937890,
826.249921864842800,
831.517780023906310,
836.790779582469900,
842.068894241700490,
847.352097970438420,
852.640365001133090,
857.933669825857460,
863.231987192405430,
868.535292100464630,
873.843559797865740,
879.156765776907600,
884.474885770751830,
889.797895749890240,
895.125771918679900,
900.458490711945270,
905.796028791646340,
911.138363043611210,
916.485470574328820,
921.837328707804890,
927.193914982476710,
932.555207148186240,
```

```
            937.921183163208070,
            943.291821191335660,
            948.667099599019820,
            954.046996952560450,
            959.431492015349480,
            964.820563745165940,
            970.214191291518320,
            975.612353993036210,
            981.015031374908400,
            986.422203146368590,
            991.833849198223450,
            997.249949600427840,
            1002.670484599700300,
            1008.095434617181700,
            1013.524780246136200,
            1018.958502249690200,
            1024.396581558613400,
            1029.838999269135500,
            1035.285736640801600,
            1040.736775094367400,
            1046.192096209724900,
            1051.651681723869200,
            1057.115513528895000,
            1062.583573670030100,
            1068.055844343701400,
            1073.532307895632800,
            1079.012946818975000,
            1084.497743752465600,
            1089.986681478622400,
            1095.479742921962700,
            1100.976911147256000,
            1106.478169357800900,
            1111.983500893733000,
            1117.492889230361000,
            1123.006317976526100,
            1128.523770872990800,
            1134.045231790853000,
            1139.570684729984800,
            1145.100113817496100,
            1150.633503306223700,
            1156.170837573242400,
        };
        return lf[n];
    }
}
```