Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de un analizador de biosecuencias: Análisis de nudos y pseudonudos en ARN

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Guillem Pitarch Rodrigo

Tutor: Jose María Sempere Luna

2014-2015

Resumen

En el presente TFG se desarrollará un analizador de secuencias de carácter bioquímico (secuencias de ARN) para la búsqueda de posibles estructuras de nudos y pseudonudos ("knots and pseudonots"). El marco teórico de este trabajo se basa en la utilización de gramáticas formales (incontextuales y sensibles al contexto) y de los algoritmos de análisis eficientes para ellas. El analizador se desarrollará en JAVA y se pretende obtener una herramienta que trabaje en modo "stand-alone". Posteriormente, se evaluará su posible integración en un website para su ejecución en red.

Palabras clave: bioinformática, algorítmica, lenguajes formales, análisis estructural

A mi familia por su apoyo, a Rocío por su cariño,
y a todas las personas que a lo largo de estos
años han contribuido a hacer de mí quien soy ahora,
gracias.

Tabla de contenidos

Contenido

1. Introducción 5

2. Fundamentos bioquímicos..... 6

3. Fundamentos de lenguajes formales……..……..8

3.1 Gramática en forma normal…….……9
3.2 El análisis sintáctico y el algoritmo de Cocke-Younger-Kasami……...…10

4. Los pseudonudos, una aproximación matemática……12

4.1 Divisiones sobre la cadena de entrada……15
4.2 Bulbos……..16

5. Algunas gramáticas utilizadas…….....17

6. Aplicación: Interfaz y prestaciones……...19

7. Pruebas de ejecución……22

8. Conclusiones y posibles mejoras para la aplicación……23

9. Bibliografía……...……24

1. Introducción

Este proyecto se centra en el diseño y desarrollo de una herramienta para la investigación biomédica capaz de localizar estructuras formadas por las cadenas de aminoácidos conocidas como Ácido Ribonucleico (ARN).

Para conseguir este objetivo, utilizaremos algoritmos de procesado de cadenas de texto con gramáticas estocásticas incontextuales que proporcionarán las reglas con las que se desglosará y procesará cada secuencia de material genético hasta descubrir cómo están relacionados sus componentes. En base a esas relaciones, la aplicación localizará dos tipos de estructura básica: la horquilla (o tallo-bucle) y el pseudonudo.

El motivo que nos lleva a invertir tiempo en este trabajo es que, en el vasto campo de la biología molecular, es bien sabido que toda estructura orgánica está ligada a una función; por tanto, la existencia de estas formaciones naturales da a entender que pueden estar asociadas a la forma en la que se expresan los genes, pudiendo poner al descubierto la clave para avanzar en la investigación del tratamiento de enfermedades autosómicas (aquellas que se heredan) o la formación de proteínas dentro de nuestras propias células.

El problema al que nos enfrentamos radica en que la complejidad de dichas operaciones hace que su coste computacional sea sumamente elevado, ya que por la naturaleza del problema (que explicaremos más adelante) la mayoría de los programas se ven obligados a usar algoritmos estadísticos lentos (del orden de n^5) que hacen del trato de largas cadenas un proceso lento. Por tanto, el objetivo de este trabajo es manejar únicamente algoritmos de coste menor (n^3 por ejemplo), buscando alguna manera de superar los obstáculos impuestos por el propio tratado de las gramáticas incontextuales, ya que estas son, esencialmente, una aproximación a las gramáticas sensibles al contexto, de manera que no pueden proveernos de una solución perfecta.

Una vez implementado el análisis, presentamos la aplicación dentro de una interfaz desde la cual el usuario podrá cargar archivos de texto *fasta*, un tipo de formato usado para representar secuencias de ácidos nucleicos, aminoácidos u otras estructuras biológicas que no son el objetivo de este trabajo, para luego tratar cada entrada de dicho archivo individualmente con la gramática o gramáticas que el usuario desee.

Naturalmente, este es un trabajo en el que se podría seguir invirtiendo tiempo, dado que este tipo de herramientas siempre es susceptible de ser corregido, ampliado y mejorado.

2. Fundamentos bioquímicos

Todo ser vivo conocido responde a unas bases biológicas comunes conocidas. Una de esas bases es el conjunto de instrucciones que sirven para el desarrollo y funcionamiento de cada organismo o virus y que es, al mismo tiempo, responsable de la transmisión hereditaria: el material genético.

Si quisiéramos definirlo con un símil muy sencillo, podríamos decir que se trata del libro de instrucciones con el que puede crearse una versión completa del organismo en cuestión; dentro del material genético conocido como ADN (Ácido Desoxirribonucleico) se pueden encontrar las instrucciones biológicas a partir de las cuales las células generarán cada estructura del cuerpo codificadas como una serie de aminoácidos, conocidos como adenina (A), timina (T), citosina (C) y guanina (G). Dichos aminoácidos forman largas secuencias que van unidas con otra secuencia igual pero inversa; esto ocurre porque cada aminoácido se "empareja" con otro, concretamente la adenina con la guanina y la citosina con la timina. Así pues, el ADN queda conservado en el núcleo de la célula en la forma de una doble hélice:

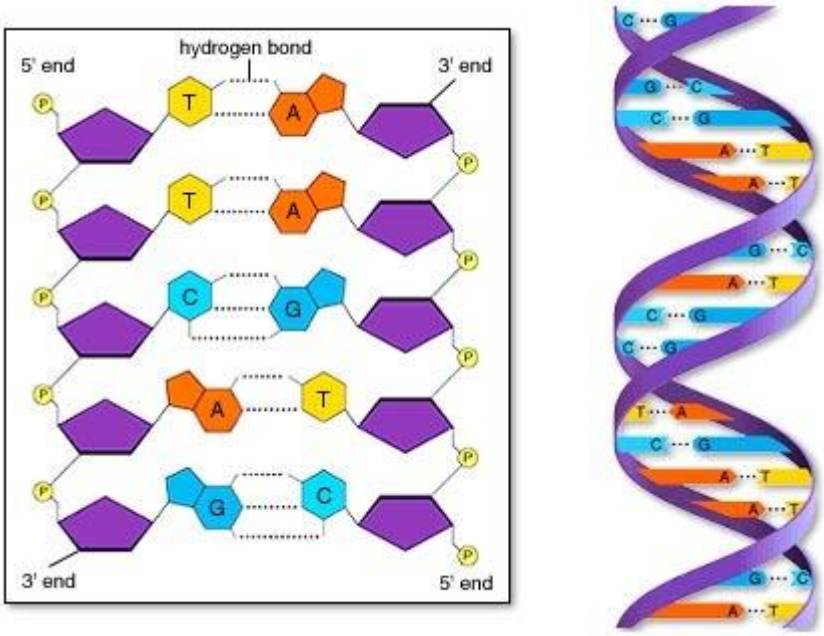

Figura 1: Ejemplo de pares de bases y ADN en forma de doble hélice

En ocasiones una célula necesitará parte de su código genético para efectuar algún proceso bioquímico en sus orgánulos (los órganos de la misma) y por tanto tendrá que leer dicho código sin dañar el ADN. Para hacerlo, se sintetiza una cadena auxiliar menos duradera conocida como ARN (Ácido Ribonucleico), que es un reflejo del fragmento del ADN que se quiere tratar, con la excepción de que el aminoácido timina es sustituido por uracilo (U). Esta cadena auxiliar será procesada y expresada por la enzima ARN-Transferasa, que creará estructuras de proteínas a partir de la misma cadena, de manera que el ADN permanecerá en el núcleo de la célula protegido en todo momento.

El problema reside en que, en ocasiones, esa estructura no se respeta y la cadena que forma el ARN puede replegarse y quedar sus aminoácidos emparejados con otros de la misma. Esa es la situación en la que decimos que encontramos una horquilla tal y como aparece en la figura 2:

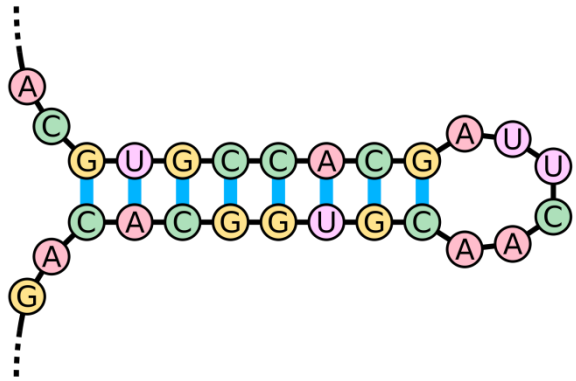

Figura 2: Ejemplo de una cadena de ARN plegada sobre si misma

Cuando esta estructura es doble, es decir, cuando encontramos dos horquillas entrelazadas, decimos que hemos encontrado un pseudonudo.

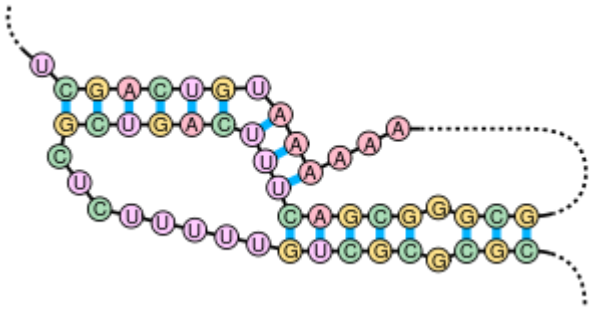

Figura 3: Ejemplo de pseudonudo de ARN

En la imagen de la Figura 3 hay dos fragmentos de material genético que están emparejados con aminoácidos de la misma cadena: el primero, a la izquierda y el segundo, a la derecha. Si leemos la cadena en orden, nos damos cuenta de que la segunda horquilla comienza antes de la segunda parte de la primera y termina después de dicha parte, quedando entonces entrelazadas.

3. Fundamentos de lenguajes formales

Definimos un alfabeto como un conjunto finito no vacío de elementos llamados símbolos. Una cadena o palabra sobre un alfabeto A es cualquier sucesión finita de elementos de A, de manera que si por ejemplo A = {x,y,z} las cadenas "xyyyyz" y "zz" cumplen dicha propiedad.

El conjunto de todas las palabras de un alfabeto A se denomina A^*, que incluye, además, la cadena vacía conocida como λ. Así pues aunque un alfabeto sea un conjunto finito sus palabras no tienen por qué serlo.

Llamamos lenguaje formal a un subconjunto de A^*, es decir, un número finito o infinito numerable de palabras donde se especifican una serie de reglas para la unir sus elementos. De esta forma, aunque la palabra "xyyzz" podría ser generada por el alfabeto definido anteriormente, no pertenecería al lenguaje con la regla "han de haber el mismo número de x que de z" pero sí al lenguaje "no puede haber una z antes de una y".

Una gramática incontextual se define como una tupla de cuatro elementos G=(N,T,P,S), donde N es un conjunto finito de símbolos auxiliares, T es el conjunto finito símbolos terminales del alfabeto, S es el símbolo inicial de escritura que cumple $S \in N$ (el símbolo inicial pertenece al conjunto de símbolos auxiliares) y P es un conjunto finito de producciones de la forma $A \rightarrow \alpha$ donde $A \in N$ (el lado izquierdo de las producciones es un carácter que pertenece al conjunto de símbolos auxiliares) y $\alpha \in (N \cup T)^*$ (el lado derecho de las producciones son concatenaciones de símbolos auxiliares o terminales).

Las producciones son las reglas que permiten transformar sucesivamente una cadena sustituyendo los símbolos auxiliares de la misma por los del lado derecho de la producción correspondiente. Para hacerlo más fácil de entender, supongamos la siguiente gramática:

N=(Y,A,B,C)
T =(a,b,c)
S=Y
P=(

Y→ aAa | bBb |cCc	Reglas 1, 2 y 3
A→ aAa | bBb	Reglas 4 y 5
B→ bBb | cCc	Reglas 6 y 7
C→ cCc | c	Reglas 8 y 9

)

Para generar una palabra iremos aplicando reglas sucesivamente en cada símbolo auxiliar que tengamos hasta quedarnos con una cadena de símbolos terminales:

Y → aAa → aaAaa → aabBbaa → aabcCcbaa → aabccCcbaa → aabccccbaa

Las reglas aplicadas han sido, en este orden: 1, 4, 5, 7, 8 y 9.

Una vez quedan eliminados todos los símbolos auxiliares, queda una palabra que pertenece al lenguaje generado por la gramática. Naturalmente, en cada paso haremos cambiar

únicamente uno de los símbolos auxiliares porque podemos encontrarnos con más de uno en la palabra al mismo tiempo.

El término "incontextual" se aplica a estas gramáticas porque en el lado izquierdo de sus producciones (los antecedentes) únicamente aparece un símbolo auxiliar. Existen gramáticas que pueden incluir más de uno de esos símbolos o incluso símbolos terminales; estas reciben el nombre de "contextuales" o sensibles al contexto, pero no son las que vamos a tratar en este trabajo.

Por otro lado, el término "estocástico" quiere decir que cada regla de producción tiene asignado un valor probabilístico (de 0 a 1), de manera que el sumatorio de todos los valores de las producciones de un mismo símbolo auxiliar sea igual a 1. Cuando una gramática tiene esta propiedad podemos conocer la probabilidad de que una palabra determinada sea generada por la misma.

Para poner un ejemplo imaginemos una versión estocástica de la gramática anterior, con las probabilidades de cada producción expresadas entre paréntesis a la derecha de las mismas:

N=(M,A,B,C)
T =(a,b,c)
S=Y
P=(

M→ aAa (0'6) | bBb (0'3) | cCc (0'1)	Reglas 1, 2 y 3
A→ aAa (0'4) | bBb (0'6)	Reglas 4 y 5
B→ bBb (0'4) | cCc (0'6)	Reglas 6 y 7
C→ cCc (0'5) | c (0'5)	Reglas 8 y 9

)

Así pues, para conocer la probabilidad de que la palabra "aabccccbaa" de dicho lenguaje sea producida hay que multiplicar el valor asociado a las reglas utilizadas para crearla; con la secuencia:

Regla 1→ Regla 4 → Regla 5 → Regla 7→ Regla 8 → Regla 9

Tenemos:

0'6*0'4*0'6*0'6*0'5*0'5 = 0'0216 = 2'16%

3.1 Gramática en forma normal

Decimos que una gramática está en una "forma" cuando todas sus reglas de producción siguen unas normas concretas y no tienen producciones fuera de ese marco. Una forma es "normal" o canónica cuando todo lenguaje generado por una clase de gramáticas puede ser descrito con sus normas sin que por ello cambie sus palabras.

Una forma normal muy conocida y sumamente relevante para este proyecto es la conocida como "Forma Normal de Chomsky", descrita por el lingüista Noam Chomsky, con la que se puede describir cualquier gramática incontextual y en la que todas las producciones siguen las estructuras:

A→BC
A→a

donde A,B,C ∈ N y $a \in T$

Si quisiéramos definir la gramática anterior dejándola en forma normal de Chomsky, quedaría de la siguiente manera:

N=(M,A,B,C, A', B', C', X, Y, Z)
T =(a,b,c)
S=Y
P=(

M→ A'X (0'6) | B'Y (0'3) | C'Z (0'1)	Reglas 1, 2 y 3
A→ A'X (0'4) | B'Y (0'6)	Reglas 4 y 5
B→ B'Y (0'4) | C'Z (0'6)	Reglas 6 y 7
C→ C'Z (0'5) | c (0'5)	Reglas 8 y 9
A'→ a (1'0)	Regla 10
B'→ b (1'0)	Regla 11
C'→ c (1'0)	Regla 12
X→ AA' (1'0)	Regla 13
Y→ BB' (1'0)	Regla 14
Z→ CC' (1'0)	Regla 15

)

Tal y como se puede ver, se trata de una gramática con más reglas de producción que su versión anterior y con algunos auxiliares extra, y por tanto más compleja en su descripción, pero más simple en su análisis (por eso las utilizamos) con exactamente el mismo alfabeto asociado e incluso las mismas probabilidades para cada palabra. Lo demostraremos con el siguiente ejemplo, calculando la probabilidad asociada a la palabra "aabcccccbaa" que ya vimos anteriormente:

M → A'X → aX → aAA' → aAa → aA'Xa → aaXa → aaAA'a → aaAaa → aaB'Yaa → aabYaa → aabBB'aa → aabBbaa → aabC'Zbaa → aabcZbaa → aabcCC'baa → aabcCcbaa → aabcC'Zcbaa → aabccZcbaa → aabccCC'cbaa → aabccCccbaa → aabcccccbaa

Las reglas aplicadas esta vez han sido: 1, 10, 13, 10, 4, 10, 13, 10, 5, 11, 14, 11, 7, 12, 15, 12, 8, 12, 15, 12 y 9.

Calculando su probabilidad asociada queda igual que en el caso anterior:
0'6*1*1*1*0'4*1*1*1*0'6*1*1*1*0'6*1*1*1*0'5*1*1*1*0'5 = 0'0216 = 2'16%

3.2 El análisis sintáctico y el algoritmo de Cocke-Younger-Kasami

La capacidad de generar palabras de las gramáticas no es lo que nos interesa en este trabajo, sino más bien los algoritmos que hacen el proceso inverso, es decir, determinar si una palabra dada pertenece o no a un lenguaje especificado por una gramática, y por lo tanto si puede ser creada usando sus reglas de derivación.

La mayoría de los algoritmos de análisis utilizan gramáticas en alguna forma normal y en este caso nos interesa la forma normal de Chomsky descrita anteriormente, que es la empleada en el algoritmo de Cocke-Younger-Kasami (CYK).

El algoritmo CYK determina si una palabra puede ser generada por una gramática incontextual y cómo puede ser generada con una complejidad temporal $\Theta(n^3)$ en el peor de los casos. Para una gramática G=(N,T,P,S) y una palabra w formada por una serie de caracteres $w_0 w_1 w_2 \dots w_{n-1} w_n$ se definiría como sigue:

Para i=1 hasta n:
    $V_{ij} = \{A: A \to w_i \in P\}$
Fin Para
Para j=2 hasta n:
    Para i=1 hasta n-j+1:
        $V_{ij} = \emptyset$
        Para k=1 hasta j-1:
            $V_{ij} = V_{ij} \cup \{A: A \to \mathrm{BC} \in \mathrm{P}, \mathrm{B} \in V_{ik}, C \in V_{i+k,j-k}\}$
        Fin Para

    Fin Para
Fin Para
Si $S \in V_{1n}$ devolver Cierto (pertenece)
Si no devolver Falso (no pertenece)

Expresado brevemente, este algoritmo va formando una matriz donde cada celda de la misma indica qué símbolos auxiliares pudieron dar lugar a un subconjunto de elementos de la cadena a analizar; de manera que la fila inferior considerará cada elemento por separado, la siguiente por parejas anexas, la siguiente por grupos de 3 y así hasta llegar a la superior, que tendrá los auxiliares que pudieron producir la cadena completa; si ahí está el símbolo inicial de la gramática significa que esa palabra puede ser generada por la misma.

Con este algoritmo se puede saber, con un coste medianamente aceptable, si una palabra puede ser generada o no por un lenguaje y además la serie de reglas que ha sido empleada en el proceso, recorriendo la matriz V que fuimos generando durante todo el proceso.

Para un ejemplo de cómo fue implementado puede consultarse el anexo del proyecto.

4. Los pseudonudos, una aproximación matemática

Expuestas ya las herramientas formales a utilizar, vamos a tratar los pseudonudos desde un punto de vista matemático. Una cadena de aminoácidos puede representarse como una palabra *w* con *n* caracteres terminales A, G, C, T y U en caso del ARN. Dicha palabra puede ser generada por una gramática que incluya dichos terminales en su lista *T*, por tanto es posible afirmar que existe al menos un lenguaje incontextual que genera cualquier secuencia de aminoácidos.

Si diseñamos el lenguaje de manera que algunas reglas de derivación favorezcan a determinadas estructuras, localizarlas sería "tan sencillo" como recorrer el árbol de derivación y localizar los puntos en que dichas reglas fueron utilizadas. Para ello hacemos uso del algoritmo CYK.

Las estructuras que queremos localizar son horquillas y pseudonudos:

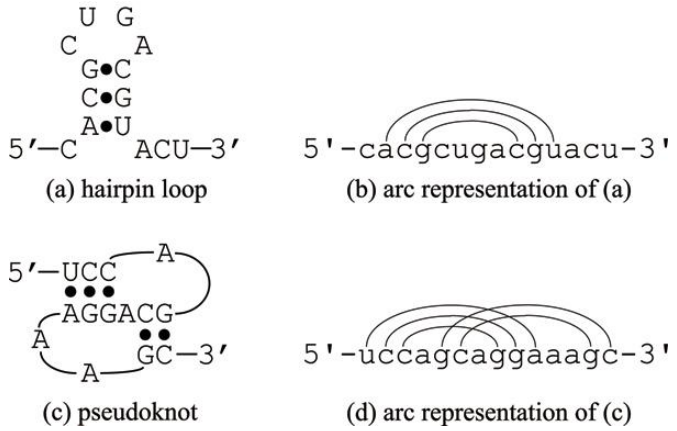

Figura 4: Ejemplo de horquilla (arriba) y pseudonudo (abajo) con ARN

Si estudiamos la estructura de la horquilla vemos que los aminoácidos están relacionados por parejas, lo que permite reconocer fácilmente esta situación con una regla del tipo A→[B], donde A,B ϵN y [ ] son dos símbolos terminales que representan cualquier combinación posible de aminoácidos emparejados.

Siempre que esta producción sea utilizada sabremos que los aminoácidos de la izquierda y la derecha están relacionados, y por tanto podremos asegurar que nos encontramos una horquilla tal y como sugiere la imagen superior derecha de la Figura 4.

El problema llega cuando intentamos diseñar una regla para localizar los pseudonudos en el árbol de derivación; si estudiamos la imagen inferior derecha de la Figura 4 podemos ver que el pseudonudo se puede definir como dos horquillas entrelazadas. Lamentablemente no se puede diseñar una regla (para una gramática incontextual) que controle al mismo tiempo la creación de ambas horquillas ya, que al definir una dejaremos los símbolos auxiliares que completarán el "interior" de la misma desconectados e imposibles de relacionar con los

auxiliares que están “fuera” de la horquilla, de manera que no podremos establecer esa relación. Podemos reconocer mejor esta situación con la gramática siguiente:

N=(M,L)
T =(a,g,c,u)
S=M
P=(
M→ aMg | gMa | cMu | uMc | MM | LM | a | g | c | u #Reglas 1 a 10
L→ aL | gL | cL | uL | a | g | c | u #Reglas 11 a 18
)

Cuando desarrollamos la cadena “uccagcaggaaagc” (vista en la Figura 4) pueden ocurrir dos cosas; o bien el árbol de derivación usa las reglas 1 a 4 (las que definen una horquilla) para encontrar la horquilla de la izquierda:

M → MM → uMaM → ucMgaM → uccMggaM

O bien usa esas mismas reglas de derivación para establecer la conexión entre los aminoácidos de la horquilla derecha:

M → MM → MgMc → MgcMgc

Se dé una situación o la otra, los símbolos auxiliares quedan de repartidos de manera que no se puede establecer esa relación y por tanto es imposible para una gramática incontextual detectar de forma segura la estructura de un pseudonudo.

Este es un problema bien conocido y que ya ha sido abordado por B. Knudsen y J. Hein [6]. No obstante, sí que existe una manera de encontrar esa relación: usar dos veces el algoritmo CYK en lugar de una: la primera ejecución encontrará las horquillas y en el caso de los pseudonudos nos dará independientemente cualquiera de las dos que lo forman. Para la segunda ejecución del CYK se eliminan previamente de la cadena todos los caracteres que ya han aparecido en una horquilla, sustituyéndolos por otro símbolo terminal (uno especial dedicado para esta función) que no pueda ser generado por las producciones A → [B] pero sí por las producciones A → α; ahora el segundo CYK encontrará las horquillas que pasó por alto la primera vez porque fueron “cortadas” por su pareja.

Terminados los dos CYK tendremos dos listas de horquillas y únicamente habrá que localizar aquellas de la primera que se crucen con alguna de la segunda, encontrando efectivamente un pseudonudo. Así pues, no existe la gramática perfecta para la búsqueda de pseudonudos por culpa de las limitaciones del sistema, pero sí puede recurrirse a este truco para conseguir el efecto deseado.

Creamos entonces una versión estocástica de CYK que guarda el árbol de derivación más probable para una cadena, lo que permite elegir la preferencia del algoritmo en el desarrollo de la palabra, forzando la aparición de las reglas que nos interesen; esto ocurre porque si una regla tiene una probabilidad de aparición mucho mayor que las demás el algoritmo elegirá el árbol de derivación que maximice las veces que se usa. Las gramáticas procesadas estarán en la siguiente forma:

A → x
A → [B]
A → BC

Donde A,B,C ∈ N, x ∈ $\{a, c, u, g, t\}$ y [ ] son cualquier combinación posible de pares de bases como símbolos auxiliares con una única producción a un símbolo terminal (A → x), por requisito del CYK.

La segunda regla es la más importante y a la que interesa dar un valor estocástico más elevado para encontrar las horquillas. Para que pueda ser procesada por el algoritmo CYK además tendrá que ser convertida a la Forma Normal de Chomsky de la siguiente manera (proceso del que se encarga el programa por su cuenta):

A → [B] (n)

A → [X (n)
X → B] (1.0)

4.1 Divisiones sobre la cadena de entrada

Cuando se llama por segunda vez a CYK existe la posibilidad de que se pierda alguna de las horquillas que buscamos, ya que en ocasiones el árbol de derivación encontrará otros enlaces, muchas veces fruto de la casualidad, que provocarán la misma situación que el problema descrito en el apartado anterior.

Para solucionarlo aplicamos dos técnicas heurísticas de análisis al algoritmo: descartar cualquier horquilla de menos de 3 aminoácidos, ya que puede ser considerada un error, y ofrecer al usuario la posibilidad de elegir hacer el segundo paso de forma independiente para cada horquilla encontrada en el anterior.

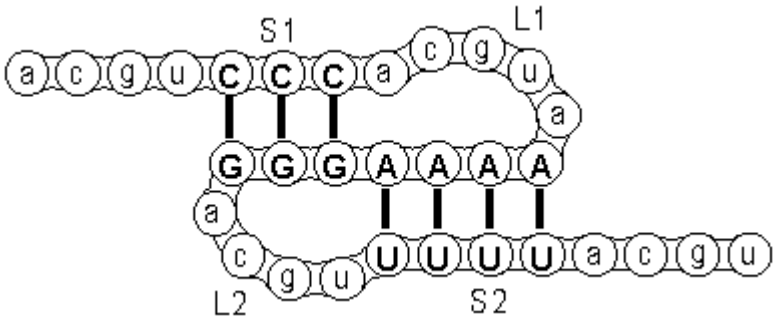

Figura 5: caso de pseudonudo

Eligiendo esta opción en lugar de aplicar CYK de nuevo sobre toda la cadena analizaríamos por separado el lado izquierdo y el derecho de cada horquilla conocida. Estos pasos extra suponen un tiempo de cálculo considerablemente mayor pero a cambio evitan en gran medida la aparición de falsas horquillas “cortando” la horquilla que queremos buscar.

Este sistema ha demostrado ser eficaz combinado con gramáticas simples; las gramáticas más complejas no suelen encontrarse con este problema y no es aconsejable sobrecargarlas con operaciones adicionales.

4.2 Bulbos

Dentro de una horquilla, un bulbo es una cadena de aminoácidos que se separa de uno de los lados emparejados formando una pequeña protuberancia.

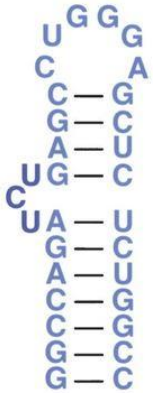

Figura 6: Caso de horquilla con bulbo

Con gramáticas incontextuales estos casos darán lugar a dos horquillas separadas aunque en realidad sean parte de la misma; para evitar ese problema el algoritmo considera como una única horquilla todas aquellas que tengan un extremo unido y no presenten otra horquilla en el espacio que las separa (el bulbo).

5. Algunas gramáticas utilizadas

Empleamos bastantes gramáticas para probar el programa pero nos quedamos con dos diseños de Robin D. Dowell y James WJ Anderson, siendo la primera una gramática ligera y la segunda una de las más pesadas, resultado de los algoritmos genéticos desarrollados en el artículo de Anderson [1].

G6 = (N,T,P,S)
N=(S,L,F,Cx,Gx,Ax,Ux)
T=(C,G,A,U)
S= S
P= (

S→L S | Cx F Gx | Gx F Cx |Ax F Ux | Ux F Ax | C | G | A | U
L→ Cx F Gx | Gx F Cx |Ax F Ux | Ux F Ax | C | G | A | U
F→ Cx F Gx | Gx F Cx |Ax F Ux | Ux F Ax | L S
Cx→C
Gx→G
Ax→A
Ux→U

)

G3 = (N,T,P,S)
N=(X,Y,Z,T,Cx,Gx,Ax,Ux,X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3,T1,T2,T3)
T=(C,G,A,U)
S= X
P= (

X→X Y |Y X | Y Y | X X | T T | Cx X1 Gx | Gx X1 Cx | Ax X1 Ux | Ux X1 Ax | Cx Y1 Gx | Gx Y1 Cx | Ax Y1 Ux | Ux Y1 Ax | Cx Z1 Gx | Gx Z1 Cx | Ax Z1 Ux | Ux Z1 Ax | Cx T1 Gx | Gx T1 Cx | Ax T1 Ux | Ux T1 Ax |
Y→C | G | A | U
Z→ X X | C | G | A | U | Cx T1 Gx | Gx T1 Cx | Ax T1 Ux | Ux T1 Ax |
T→ Z T | Y T | Cx X1 Gx | Gx X1 Cx | Ax X1 Ux | Ux X1 Ax | Cx Z1 Gx | Gx Z1 Cx | Ax Z1 Ux | Ux Z1 Ax
Cx→ C
Gx→ G
Ax→ A
Ux→ U
X1→ Cx X2 Gx | Gx X2 Cx | Ax X2 Ux | Ux X2 Ax
X2→ Cx X3 Gx | Gx X3 Cx | Ax X3 Ux | Ux X3 Ax
X3→ Cx X3 Gx | Gx X3 Cx | Ax X3 Ux | Ux X3 Ax | Cx X Gx | Gx X Cx | Ax X Ux | Ux X Ax
Y1→ Cx Y2 Gx | Gx Y2 Cx | Ax Y2 Ux | Ux Y2 Ax
Y2→ Cx Y3 Gx | Gx Y3 Cx | Ax Y3 Ux | Ux Y3 Ax
Y3→ Cx Y3 Gx | Gx Y3 Cx | Ax Y3 Ux | Ux Y3 Ax | Cx Y Gx | Gx Y Cx | Ax Y Ux | Ux Y Ax
Z1→ Cx Z2 Gx | Gx Z2 Cx | Ax Z2 Ux | Ux Z2 Ax
Z2→ Cx Z3 Gx | Gx Z3 Cx | Ax Z3 Ux | Ux Z3 Ax
Z3→ Cx Z3 Gx | Gx Z3 Cx | Ax Z3 Ux | Ux Z3 Ax | Cx Z Gx | Gx Z Cx | Ax Z Ux | Ux Z Ax
T1→ Cx T2 Gx | Gx T2 Cx | Ax T2 Ux | Ux T2 Ax
T2→ Cx T3 Gx | Gx T3 Cx | Ax T3 Ux | Ux T3 Ax

T3→ Cx T3 Gx | Gx T3 Cx | Ax T3 Ux | Ux T3 Ax | Cx T Gx | Gx T Cx | Ax T Ux | Ux T Ax
)

La segunda gramática fue modificada respecto a cómo aparece en el artículo de James WJ Anderson para conseguir que las horquillas necesitaran una longitud mínima de 3 aminoácidos, gracias a la inserción de los auxiliares Xn, Yn, Zn y Tn. Esta mejora redujo el número de errores encontrados y mejoró su tasa de éxito, aunque a cambio de incrementar el tiempo de ejecución.

Otro problema que encontramos es que los creadores de las gramáticas originales no proporcionan los valores estocásticos de cada producción, por lo que nos ha tocado elegir unos valores arbitrarios siguiendo el patrón de puntuar más altas las producciones A → [B], lo que dio buenos resultados, pero no obstante una gramática con valores mejor elegidos tendría que ser más eficiente.

6. Aplicación: Interfaz y prestaciones

La aplicación ha sido desarrollada usando Java y posee una interfaz para facilitar su control al usuario. Se accede a la misma a través de una ventana de selección desde la que también se puede acceder a otras herramientas de análisis de biosecuencias. Una vez seleccionado tenemos la siguiente ventana:

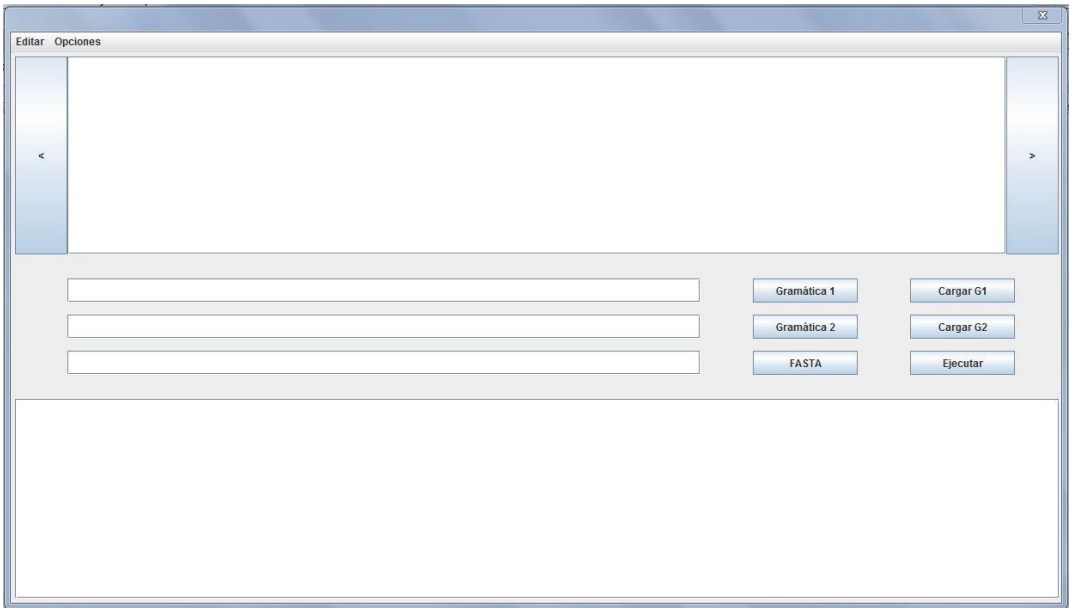

Figura 7: Interfaz abierta sin modificar

Con los botones “Gramática 1”, “Gramática 2” y “FASTA” se abre una ventana de selección para leer las gramáticas y los archivos fasta. En el caso de las gramáticas es necesario pulsar el botón de Cargar para que el programa las asimile.

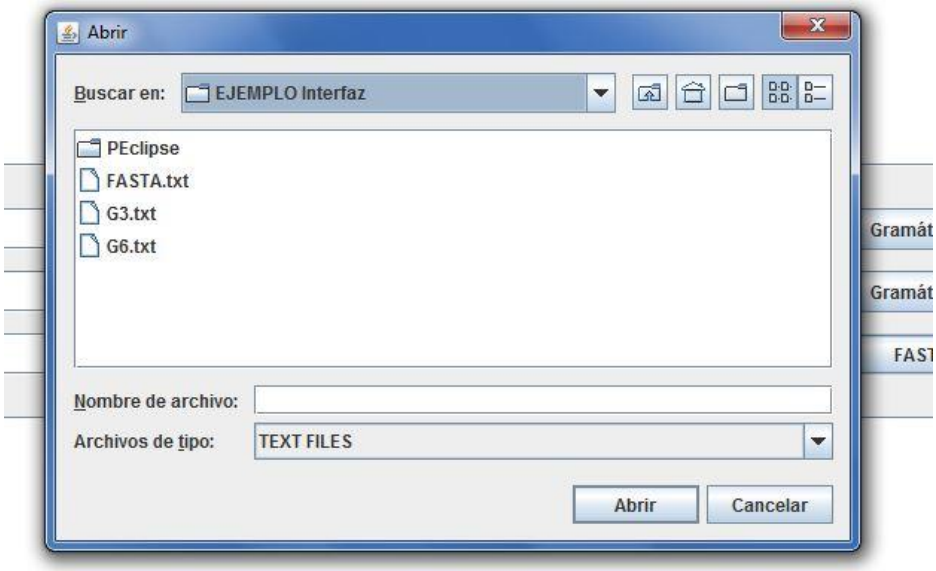

Figura 8: Menú de selección de archivos

Una vez elegido un fichero fasta el programa separará cada una de sus cadenas de entrada y las mostrará en la zona superior de la interfaz, de manera que el usuario tenga siempre a la vista solo un caso. Al presionar el botón "Ejecutar" se tratará la cadena seleccionada y se mostrará el resultado en la parte inferior de la interfaz, junto a una breve descripción de las condiciones de ejecución (número de gramáticas empleadas y algoritmo utilizado).

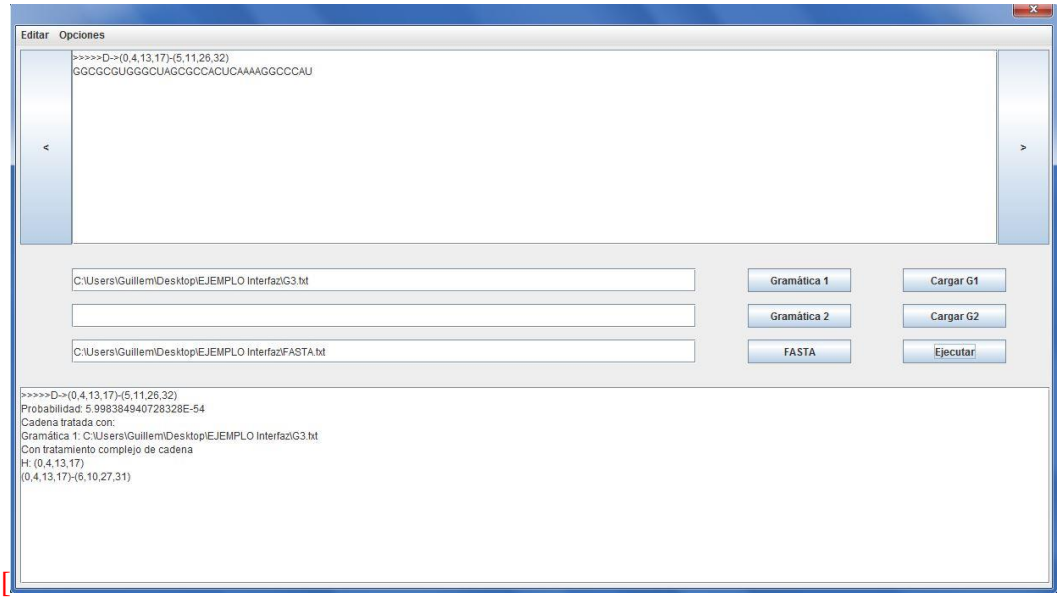

Figura 9: Interfaz con fasta cargado tras una ejecución

En la barra de herramientas el usuario podrá acceder a las opciones de ejecución. El menú "Opciones" permite elegir el número de gramáticas (en el caso de elegir dos, la segunda se usa en la segunda invocación a CYK) y la complejidad del algoritmo (simple o dividiendo la cadena).

Encontramos también en la misma barra el menú de edición, que permite modificar en tiempo de ejecución cualquiera de las dos gramáticas cargadas por el programa:

Figura 10: Pantalla de edición de gramática

El formato seguido, que también han de respetar los .txt de donde sean leídas, comienza con un entero N que indica el número de símbolos auxiliares de la gramática, seguido por N líneas dentro de las cuales se especifican las reglas de un auxiliar a la izquierda de la "->". A la derecha, se enumeran las transiciones que siguen el patrón:

A → x
A → [B]
A → BC

Los elementos del lado derecho de cada producción han de estar separados por comas; luego va el valor estocástico, separado por dos puntos, y finalmente la siguiente producción va separada por un punto y coma.

L->Cx,F,Gx:0.2;Gx,F,Cx:0.2;Ax,F,Ux:0.2;Ux,F,Ax:0.2;C:0.04;G:0.04;A:0.04;U:0.04

7. Pruebas de ejecución

Uno de los principales problemas a la hora de poner a prueba la efectividad de la aplicación fue la falta de material disponible. Dado que no encontramos ninguna base de datos con ejemplos de cadenas pseudoanudadas tuvimos que recurrir a cadenas sueltas encontradas en varios artículos.

Los resultados obtenidos son satisfactorios, en tanto en cuanto conseguimos encontrar las estructuras secundarias buscadas, aunque hay que tener en cuenta que la efectividad del método depende en gran medida de lo bien acotada que esté la gramática y el objetivo de este trabajo no es encontrar ese diseño sino proporcionar la herramienta.

Pese a todo con algo tan "tosco" como la gramática G3, definida con anterioridad, la aplicación ya muestra buenos resultados incluso sin aplicar algunas de sus técnicas heurísticas adicionales (el uso de dos gramáticas o la división de la cadena). A continuación puede verse un ejemplo donde las horquillas están definidas como conjuntos de 4 elementos (n,m,m',n') donde n y m indican los aminoácidos inicial y final del primer lado del solapamiento y m' y n' lo propio para el segundo lado del solapamiento, de manera que el aminoácido en la posición n está unido al de n', el de n+1 con el de n'-1 y así sucesivamente hasta m, que está unido a m'.

>GGCGCGUGGGCUAGCGCCACUCAAAAGGCCCAU
Un pseudonudo entre las horquillas (6,10,27,31) y (0,4,13,17)

> GCGGCCAGCUCCAGGCCGCCAAACAAUAUGGAGCAC
Un pseudonudo entre las horquillas (0,5,13,18) y (7,12,28,33)

>GGGCUGUUUUUCUCGCUGACUUUCAGCCCCAAACAAAAAAGUCAGCA
Dos horquillas: (14,18,22,26) y (2,10,42,39)

>GGCAGCGAUCUAGAGUUCGACGCUGCAUGAACUCG
Un pseudonudo entre las horquillas (12,17,28,33) y (1,6,20,25)

> CAGACCAGCCACCGAAUGCCUGGCUUACAUUCCAGCCUGU
Un pseudonudo entre las horquillas (6,10,20,24) y (13,17,27,31)

>UAGCAGCCUAACUUAAGUUGUUAGGUAACUUAACGUUAGA
Dos horquillas: (19,22,25,28) y (7,15,34,32)

>GAAGCUUCUGCCGGUACCGUUGUGGCGAUAGCAGAAGCAACACAACGGUU
Un pseudonudo entre las horquillas (15,23,40,48) y (3,10,30,37)

8. Conclusiones y posibles mejoras para la aplicación

Aunque es posible encontrar estructuras como los pseudonudos usando gramáticas incontextuales, es cierto que los problemas mencionados anteriormente lo convierten en una tarea más compleja, lo que se traduce en un mayor tiempo de ejecución si queremos aumentar las probabilidades de éxito.

Una gramática diseñada con unos valores estocásticos mejor elegidos sin lugar a dudas obtendría mejores resultados, aunque de conseguir un algoritmo de clasificación rápido para gramáticas sensibles al contexto se podría mejorar aún más el programa. Si nos quedamos con las incontextuales entonces podríamos recurrir a métodos de entrenamiento de gramáticas, a fin de refinar la creación de los árboles de derivación; además de la inclusión de nuevas técnicas heurísticas. Este último punto es delicado, ya que si tenemos en mente un programa rápido hay que elegir con mucho cuidado qué operaciones extra son incluidas.

Sería posible hacer una versión on-line de esta aplicación aunque terminamos por descartarlo por falta de tiempo, no obstante tal y como comentamos en la introducción aún se podría invertir tiempo en mejorar la aplicación.

9. Bibliografía

Trabajos de particular interés en el desarrollo de este programa:

1. JAMES WJ. ANDERSON, PAULA TATARU, JOE STAINES, JOTUN HEIN, RUNE LYNGSO (2012) "Evolving stochastic context-free grammars for RNA secondary structure prediction" en *BMC Bioinformatics*, University of Oxford
2. MICHAEL BROWN, CHARLES WILSON (1995) "RNA Pseudoknot Modeling Using Intersections of Stochastic Context Free Grammars with Applications to Database Search". University of California.
3. ANNE CONDON, BETH DAVY, BAHARAK RASTEGARI, SHELLY ZHAO, FINBARR TARRAN (2004) "Classifying RNA pseudoknotted structures". University of British Columbia.
4. ROBIN D. DOWELL, SEAN R. EDDY (2004) "Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction" en *BMC Bioinformatics*. University of Colorado.
5. MICHAEL R. GAREY, DAVID S. JOHNSON (1999) *Computers and Intractability a guide to the Theory of NP-Completeness*. New Jersey: W. H. Freeman
6. B. KNUDSEN, J. HEIN (1999) "RNA secondary structure prediction using stochastic context-free grammars and evolutionary history". University of Aarhus.
7. DAVID L. NELSON, MICHAEL M. COX (2002) *Principles of Biochemistry*. Stanford: W. H. Freeman
8. ELENA RIVAS, SEAN R. EDDY (1999) "A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots". University of Oxford.
9. ELENA RIVAS, SEAN R. EDDY (1999) "The language of RNA: a formal grammar that includes pseudoknots". University of Oxford.
10. DAVID W. STAPLE, SAMUEL E.BUTCHER (2005) "Pseudoknots: RNA Structures with Diverse Functions".
11. STEVEN VAN VAERENBERG, LUIS VIELVA, "Una introducción a la predicción de la estructura secundaria del RNA mediante métodos estocásticos"
12. ERIC WESTHOF, LUC JAEGGER (1992) RNA Pseudoknots

Anexo

Para implementar en Java el CYK estocástico y construir el árbol de derivación creamos una clase "Tree" con los siguientes elementos:

```
class Tree{
   String phrase; #El auxiliar que dio lugar a esta producción
   int startPhrase, endPhrase; #Índices del inicio y fin de la palabra
   String word; #Terminal, si nos encontramos en una hoja del árbol
   Tree left, right; #Lado izquierdo y derecho respectivamente en el árbol
   double prob; #Probabilidad estocástica asociada hasta el momento
        public Tree(String phr,int str, int nd, String wrd, Tree lft, Tree rgh, double prb){
                this.phrase=phr;
                this.startPhrase=str;
                this.endPhrase=nd;
                this.word=wrd;
                this.left=lft;
                this.right=rgh;
                this.prob=prb;
        }
}
```

E implementamos CYK de la siguiente manera (algoritmo):

Inputs: sentence (una cadena de caracteres)
grammar (la gramática utilizada)

```
N = length(sentence);
FOR(i=0 hasta N-1){
        Word = sentence[ i ];
        FOR(cada regla "POS→Word[prob]" en la gramática )
                P[POS,i,i] = new Tree(POS,i,iword,null,null,prob);
}
 FOR(L=1 hasta N-1){
        FOR(i=0 hasta N-L-1){
                j = i+L
                FOR(cada auxiliar M){
                        P[M,i,j] = new Tree(M,i,j,null,null,null,0.0);
                        FOR(k=i hasta j-1){
                                FOR(cada regla "M→Y,Z (prob)" en la gramática){
                                NuevaProb = P[Y,i,k].prob * P[Z,k+1, j].prob * prob;
                                SI(NuevaProb > P[M,i,j].prob){
                                        P[M,i,j].left = P[Y,i,k];
                                        P[M,i,j].right = P[Z,k+1,j];
                                        P[M,I,j].prob = NuevaProb;
                                }//Fin del IF
                                }//Fin del tercer FOR
                        }//Fin del segundo FOR
                }//Fin del primer FOR
        }//Fin del FOR que recorre la palabra
        Return P;
}
```

El árbol de derivación más probable, además de la probabilidad asociada a la producción de la cadena, quedan en P[S,0,N-1], donde S representa la posición del símbolo inicial del alfabeto, generalmente 0.