Tarea 5 - Predicción de la estructura secundaria de ARN

Julián María Galindo Álvarez

Resumen

En este trabajo vamos a explorar la aplicación de la programación de conjuntos de respuesta al problema de predicción de la estructura secundaria de ARN, basándonos en el trabajo expuesto en "Exploring Life through Logic Programming: Logic Programming in Bioinformatics" [3]. Para ello, se realizará una introducción y descripción del problema para luego explorar el código propuesto por los autores y adaptar el código a la versión de clingo usada en clase, ejecutando algunos ejemplos e implementaciones propias. Además, buscando completar el trabajo, desarrollaremos también un script de python que usando clingo y la applet VARNA [2], generará imágenes de la estructura secundaria dada una secuencia de ARN de entrada.

1. Introducción

El ADN (ácido desoxirribonucleico) se caracteriza por una secuencia de nucleótidos de cuatro tipos: A, C, G y T (adenina, citosina, guanina y timina). Normalmente, las hebras de ADN tienen una alta tendencia a emparejarse: dos hebras pueden alinearse y los nucleótidos enfrentados, uno de cada hebra, pueden formar una unión relativamente estable. Algunos emparejamientos de nucleótidos son más favorables que otros. En concreto, dada una secuencia $s = \{s_1, \ldots, s_n\}$ con $s_i \in \{A, C, G, T\}$, su secuencia complementaria $\bar{s}$ se obtiene invirtiendo el orden de la secuencia y sustituyendo A por T, C por G y viceversa. Una cadena s y su cadena complementaria $\bar{s}$ forman enlaces entre cada par de nucleótidos correspondientes y juntas se pliegan en la famosa doble hélice.

La transcripción es el proceso de conversión de ADN en ARN (ácido ribonucleico), donde la doble hélice del ADN se separa localmente por efecto

de moléculas específicas (enzimas). Durante este proceso las dos hebras individuales quedan expuestas al entorno y pueden ser utilizadas para iniciar la transcripción. La nueva cadena de ARN está compuesta por cuatro tipos de nucleótidos: A, C, G y U, donde A, C y G son los mismos que en el ADN, mientras que U (uracilo) puede verse como una variante de T. La cadena de ARN es monocatenaria (formada por una sola cadena) y es menos estable y más corta que una secuencia de ADN aunque el ARN puede realizar varias tareas fundamentales dentro de la célula. Una vez finalizada la transcripción, la doble hélice de ADN se vuelve a formar y la nueva secuencia de ARN actúa como mensajera de la información extraída del ADN.

La cadena de ARN se pliega en el espacio según una serie de emparejamientos favorables entre pares de nucleótidos (emparejamiento de bases). En este caso, la presencia de una sola cadena permite formas más ricas con respecto a la hélice de ADN. La llamada **estructura secundaria del ARN** es el conjunto de sus emparejamientos de bases Figura 1. En particular, A-U y C-G son los emparejamientos de bases más favorecidos, aunque que también es posible encontrar pares U-G. La topología de los pares influye y estabiliza la forma funcional tridimensional de la cadena. La predicción de esta forma tridimensional es extremadamente importante, y este es el principal objetivo de este trabajo.

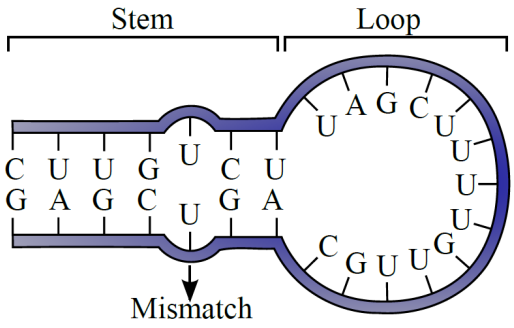

Figura 1: Ejemplo de la estructura secundaria de una cadena de ARN.

2. Descripción del problema

La secuencia de ARN $s = \{s_1, \ldots, s_n\}$ es una cadena de longitud n de símbolos $\{A, C, G, U\}$. La secuencia puede plegarse en el espacio, de modo que los pares de bases pueden interactuar físicamente. En general, la evaluación energética de estas interacciones puede calcularse de forma precisa, independientemente de la disposición tridimensional global de la secuencia. Por lo tanto, es posible calcular la mejor estructura secundaria ignorando el plegado real en el espacio. Para ello, los autores definen una función P que es parcial (no todos los elementos del dominio tienen por qué tener imagen) e inyectiva que representa una posible estructura secundaria de la secuencia a través de posibles emparejamientos. Concretamente se define como:

$$\begin{aligned} P\colon \{1, \ldots, n\} &\longrightarrow \{1, \ldots, n\} \\ P(i) = j &\iff P(j) = i \\ P(i) = j &\implies (s_i, s_j) \in \{(A,U), (U,A), (C,G), (G,C), (U,G), (G,U)\}. \end{aligned} \tag{1}$$

Para simplificar, la función anterior se puede representar como el conjunto de pares $P = \{(i, j) | P(i) = j\}$ con las condiciones anteriores.

Con el fin de identificar la estructura secundaria más probable, se han estudiado varios modelos energéticos y se han propuesto métodos eficientes para su estimación. Los autores presentan dos aproximaciones sencillas de las funciones energéticas. La primera se basa en la aproximación más simple y maximiza el número de emparejamientos favorables (es decir, A-U, C-G y U-G). Una función de energía más refinada evalúa la frecuencia de los emparejamientos. Así, el problema de la predicción de la estructura secundaria del ARN es el de encontrar, entre todas las funciones P, la que minimiza una función de energía dada. Experimentamos con dos funciones de energía simples:

$$E_1 = -|P|. \tag{2.1}$$

$$E_2 = c_1(n - |P|) + c_2|AU - 0.35|P|| + c_3|CG - 0.53|P||, \tag{2.2}$$

donde AU y CG son el número de contactos de las bases A-U y C-G respectivamente; c_1, c_2, c_3 son pesos modificables y $|P|$ representa el cardinal de P.

La función de energía E_1 2.1 fue propuesta por Nussinov [7] y representa la aproximación más sencilla del problema. Maximiza el número de pares admisibles y, por tanto, busca el mayor empaquetamiento de la estructura.

La función de energía E_2 2.2 es una adaptación de los autores de lo expuesto en [1] y [5] y busca una distribución de tipos de pares lo más similar a una distribución probabilística típica obtenida a partir de observaciones experimentales. En concreto, esta función consta de tres términos que pretenden lo siguiente: el correspondiente a c_1 busca maximizar el número de emparejamientos en la estructura (misma idea que E_1). Los términos correspondientes a c_2 y c_3 buscan que las proporciones de los emparejamientos A-U y C-G se parezcan a las distribuciones obtenidas experimentalmente. En concreto, consultando [1], podemos ver que la probabilidad experimental de que un emparejamiento sea de tipo A-U es del 0.35 y 0.53 en el caso de C-G.

Notemos varios detalles en estas definiciones. En primer lugar, estas definiciones son en principio adimensionales (salvo que se impongan dimensiones de energía a los pesos) por lo que llamarlas "energía" es un abuso de nomenclatura, siendo mejor verlas como simples funciones de coste a minimizar, aunque puedan entenderse como proporcionales a la energía final del sistema. En segundo lugar, tal y como está definido el conjunto P, su cardinal $|P|$ corresponde al número de bases que tienen emparejamiento en la posible estructura secundaria P, esto es, el doble del número de emparejamientos. Por este motivo también hay que considerar de esta forma las variables AU y CG para que E_2 sea coherente.

3. Exploración del código ASP original

Veamos la implementación de los autores. Notemos que el código usado corresponde a una sintaxis distinta a la que usamos en la versión de *Clingo* vista en clase.

En primer lugar, la secuencia $S = \{s_1, \ldots, s_n\}$ se introducen a partir de n hechos de la forma `seq(i,s_i)`.

```
seq(1,a). seq(2,g). seq(3,u). seq(4,c). seq(5,c). seq(6,a).   %F1
```

Es decir, la secuencia del ejemplo se corresponde a la representada en la siguiente imagen:

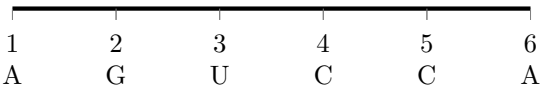

Figura 2: Secuencia de tamaño 6 del ejemplo del código.

```
%% domain predicates
sequence_index(X) :- seq(X,_).                                    %R1
sequence_base(B) :- seq(_,B).                                     %R2
```

Las reglas `R1` y `R2` introducen los predicados referentes al dominio de la secuencia de entrada, lo que permite evitar variables no seguras en futuras cláusulas.

```
%%% Definition of the pairing function
0 {pairing(X,Y):sequence_index(Y)} 1 :- sequence_index(X).    %R3

%% the pairing is injective and symmetric
:-sequence_index(X1;X2;Y),X1<X2,pairing(X1,Y),pairing(X2,Y).  %R4
pairing(B,A):- sequence_index(A;B),pairing(A,B).              %R5
```

La regla `R3` define el predicado `pairing/2` que indica que dos bases (referenciadas a través de sus índices) son emparejadas. Esta línea genera entre 0 y 1 elementos `pairing(X,Y)` para cada índice `X` y posible índice `Y` (una base puede estar emparejada con sólo una base o con ninguna).

Por otro lado, las reglas `R4` y `R5` añaden las propiedades de inyección (dos bases distintas no pueden estar emparejadas con la misma base) y simetría (si la base X está emparejada con Y entonces Y está emparejada con X) de la función de emparejamiento a través de restricciones.

```
%% wrong associations
wrong(X,X):- sequence_base(X).                                    %R6
wrong(a,c). wrong(a,g). wrong(c,u).                               %R7
:-wrong(B1,B2),seq(X1,B1),seq(X2,B2),pairing(X1,X2).              %R8
```

Para eliminar los modelos en los que se dan emparejamientos que no son favorables, los autores definen a partir del predicado `wrong/2` que dos bases no dan lugar a emparejamientos: `R6` indica que dos bases iguales no pueden emparejarse. En `R7` se explicitan cuales son los emparejamientos no permitidos de bases distintas y finalmente `R8` es la restricción que elimina modelos donde existen emparejamientos no favorables.

```
%% Optional constraint: no pseudo-knots
:- sequence_index(X1;X2;X3;X4), X1<X3,X3<X2,X2<X4,          %R11
   pairing(X1,X2),pairing(X3,X4).
```

La restricción opcional `R11` elimina modelos en los que se dan pseudo-nudos (estructuras de tallo-bucle en las que la mitad de uno de los tallos está intercalado entre las dos mitades de otro tallo, donde por tallo nos referimos simplemente a una región de la secuencia).

```
%% Nussinov Energy E1
contacts(C):- C = #count{pairing(A,B)}.                    %R12
#maximize[contacts(C)=C].                                  %R13_1
```

La regla `R12` calcula el número de emparejamientos existentes. Notemos que así definido, el número de contactos es el doble que el del enlaces debido a la simetría. La regla `R13_1` corresponde a la maximización del número de emparejamientos existentes, correspondiente a la energía de Nussinov.

Para usar la definición de energía E_2, la regla `R13_1` se sustituiría por el siguiente código.

```
#maximize[energy(E)=E].                                    %R13_2
total(N) :- N=#count{ seq(X,Y)}.                           %R14
au(N) :- N=#count{pairing(A,B):seq(A,a):seq(B,u)}.         %R15
cg(N) :- N=#count{pairing(A,B):seq(A,c):seq(B,g)}.         %R16
energy(E) :- C1=1, C2=1, C3=1,                             %R17
         total(N), contacts(C), au(AU), cg(CG),
         E = C1 * (N-C/2) + C2 * #abs(100*AU - 35*C) +
            C3 * #abs(100*CG - 53*C).
```

La regla `R13_2` corresponde a la maximización de la función de energía E_2, definida en la regla `R17`. En esta regla se han considerado los pesos $c_1 = c_2 = c_3 = 1$. Además se han multiplicado los términos por 100 para transformar los números decimales de la fórmula a enteros.

3.1. Comentario sobre la implementación de E_2

En esta sección hemos comentado el código original planteado por los autores pero hemos observado varios problemas en la implementación de E_2. En primer lugar, aunque los autores comentan que el término `C1` también debe ir multiplicado por 100, en la implementación no es así. En segundo lugar, notemos que con esta implementación, `contacts(C)` coincide con $|P|$ (número de bases con emparejamientos), por lo que el término correspondiente a `C1` es precisamente el que no debería estar dividido entre 2, mientras que los correspondientes a `AU` y `CG` sí, ya que estos últimos se calculan sin repeticiones (es decir, número de emparejamientos y no número de bases emparejadas). Estos errores dan lugar a una función de coste mal planteada. En cualquier caso, el error más grave es que los autores maximizan esta función en lugar de minimizarla. Recordemos que está función esta planteada como un error entre lo esperado y lo obtenido (y es siempre positiva por definición con los pesos establecidos positivos) por lo que se espera reducir su valor a 0 lo máximo posible y por tanto, minimizarla.

4. Adaptación del código a `CLINGO` 5.4

En esta sección voy a adaptar el código propuesto por los autores a la sintaxis de clingo en su versión 5.4. Además, dado lo visto en la Subsección 3.1, realizaré una nueva propuesta para la función de coste E_2. El código de clingo lo he dividido en cuatro archivos:

- **rna_ss_input.lp**: archivo con los hechos que describen la secuencia de entrada, separado del código principal para independizar los distintos procesos.
- **rna_ss_prediction_base.lp**: adaptación del código original hasta el cálculo de contactos. Importa el archivo anterior.
- **rna_ss_prediction_E1.lp**: adaptación del código original correspondiente a E_1. Importa el archivo rna_ss_prediction_base.lp.

- **rna_ss_prediction_E2.lp**: adaptación del código original correspondiente a E_1. Importa el archivo rna_ss_prediction_base.lp.

Dado un archivo de input **rna_ss_input.lp**, los archivos a ejecutar dependiendo de la función de coste que se quiera utilizar para la predicción son **rna_ss_prediction_E1.lp** y **rna_ss_prediction_E2.lp**. Veamos ahora las distintas adaptaciones.

4.1. rna_ss_prediction_base.lp

En primer lugar, dado que los autores consideran el permitir pseudo-nudos como elemento opcional, añadimos una constante `allow_pseudo_knots` para poder controlar el uso de la regla o no mediante parámetros por consola. Incluimos además el archivo **rna_ss_input.lp** para importar la secuencia de entrada.

```
#const allow_pseudo_knots=false.
pseudo_knots(allow_pseudo_knots).

#include "input/rna_ss_input.lp".
```

La siguiente modificación se debe a que esta versión de clingo, cuando existen predicados de la forma `p(X1;X2)` en el cuerpo de una regla, se genera una regla en el grounding del programa para cada elemento del predicado y este no es el comportamiento pensado por los autores en su código. Por este motivo, debemos cambiar este tipo de predicados por `p(X1), p(X2)`.

Para que la restricción de los pseudo-nudos sea sensible a la variable de control, añadimos el predicado `pseudo_knots(false)` en la regla `R11`.

```
:- sequence_index(X1), sequence_index(X2),                    %R11
   sequence_index(X3), sequence_index(X4),
   X1<X3,X3<X2,X2<X4,
   pairing(X1,X2),pairing(X3,X4),
   pseudo_knots(false).
```

Por último, debemos adaptar la sintaxis de la función de agregado `#count`:

```
contacts(C):- C = #count{A,B : pairing(A,B)}.
```

4.2. rna_ss_prediction_E1.lp

En este caso realizamos las siguientes modificaciones:

- Definimos el predicado `energy1/1` que se define como el negativo del número de contactos.
- Minimizamos el valor `energy1/1`. Este cambio no mejora nada con respecto a la implementación original, de hecho empeora el rendimiento al añadir más hechos a la base de conocimiento. En cualquier caso prefiero esta representación para recalcar qué función de coste estamos minimizando.
- Adaptamos la sintaxis de la función de agregado `#count`.

```
#include "rna_ss_prediction_base.lp".

%% minimize energy function
#minimize{E: energy1(E)}.                                    %R13_1

%% Nussinov Energy E1
energy1(-C) :- contacts(C).                                  %R13_1_1
```

4.3. rna_ss_prediction_E2.lp

Para este caso, voy a proponer una función ligeramente distinta a la de los autores o más bien, matizarla. Consultando [1] podemos ver que empíricamente se dan las siguientes probabilidades de que un emparejamiento sea de un determinado tipo:

$$P_{CG} = 0.53, \qquad P_{AU} = 0.35, \qquad P_{GU} = 0.12.$$

Notemos que tal y como hacen los autores, no es necesario añadir restricciones para las tres probabilidades en la función de coste ya que fijadas dos de ellas, la tercera es complementaria. La función propuesta considera sólo el peso ajustable c_1 que indica la proporción de bases que deben emparejarse. El resto de pesos no son necesarios al pretender ajustar una distribución experimental que depende implícitamente de este nuevo c_1.

$$E_2 = |c_1 N - |P|| + |AU_B - 0.35|P|| + |CG_B - 0.53|P||.$$

Debemos tener en cuenta el siguiente matiz: dada la definición de P, recordemos que $|P|$ indica el número de contactos (número de bases con emparejamiento). Para que esta función sea coherente, tanto AU_B como CG_B deben indicar el número de bases con emparejamiento de ese tipo. En la implementación original, sin embargo, se calcula el número de emparejamientos de ese tipo, cuyo valor es la mitad. Por lo que equivalentemente:

$$E_2 = |c_1 N - |P|| + \left|AU - 0.35\frac{|P|}{2}\right| + \left|CG - 0.53\frac{|P|}{2}\right|,$$

donde AU y CG indican el número de emparejamientos de cada tipo, por lo que deben compararse con el número de emparejamientos totales $|P|/2$.

En la siguiente implementación vamos a dar un conjunto de respuesta para el parámetro c_1 definiendo antes un grid de posibles valores de búsqueda (pondremos por defecto el conjunto de 0.4 a 1 con paso 0.1). Este grid es modificable para afinar el proceso. De esta manera el parámetro c_1 es aprendido por el modelo para ajustarse lo mejor posible a la distribución de emparejamientos.

```
#include "rna_ss_prediction_base.lp".

%% minimize energy function
#minimize{E : energy2(E)}.                                    %R13_2

%% length of the sequence
total(N) :- N=#count{X,Y : seq(X,Y)}.                         %R14

%% number of AU and CG pairings
au(N) :- N=#count{A,B:pairing(A,B),seq(A,a),seq(B,u)}.        %R15
cg(N) :- N=#count{A,B:pairing(A,B),seq(A,c),seq(B,g)}.        %R16
```

```
%% search grid for parameter c1
c(40;50;60;70;80;100).                                          %R17
1{c1(C) : c(C)}1.                                               %R18

%% cost function
energy2(E) :- total(N), contacts(C), P = C/2,                   %R19
              au(AU), cg(CG), c1(C1),
              E =  |C1*N-100*C|
                  + |100*AU - 35*P|
                  + |100*CG - 53*P|.
```

5. Predictor con generador de imágenes

Para poder visualizar las estructuras predichas por el programa, he desarrollado un script de python simple que llama al programa clingo para realizar la predicción y posteriormente al applet VARNA [2] para generar imágenes correspondientes a las predicciones (requiere de la instalación de java). Todo el código, incluido el de clingo puede encontrarse en el siguiente repositorio.

https://github.com/julgalalv/RNA-secondary-structure-prediction

En el archivo README.md del repositorio pueden encontrase instrucciones para ejecutar el script. En la siguiente sección veremos algunos ejemplos.

Cabe mencionar que de todos los modelos que minimizan las funciones (estructuras compatibles) vamos a representar el primero que devuelve clingo, aunque lo habitual será que existan otras estructuras que minimicen las funciones de coste.

Como observación, debemos comentar que dependiendo de la función de coste usada, este problema tiene una complejidad de tiempo entre $O(n^2)$ y $O(n^4)$, por lo que si se quiere probar el programa, no recomiendo usar cadenas mayores de 24 bases (longitud de la cadena más larga que veremos en las siguientes secciones).

6. Comparación y evaluación.

En esta sección vamos a visualizar las distintas estructuras de los cuatro ejemplos disponibles en el artículo de referencia [3]. Vamos a comparar la estructura generada usando las funciones E_1 y mi propuesta de $E2$ junto a la implementación original de E_2 (indicada en las imágenes con el sufijo $E2\_0$).

Para evaluar los modelos vamos a fijarnos en la distribución empírica de pares que vimos en [1]:

$$P_{CG} = 0.53, \qquad P_{AU} = 0.35, \qquad P_{GU} = 0.12.$$

Consideraremos que el mejor modelo será aquel que mejor se ajuste a esta distribución. Además en el caso de E_2 indicaremos el mejor parámetro c_1 obtenido.

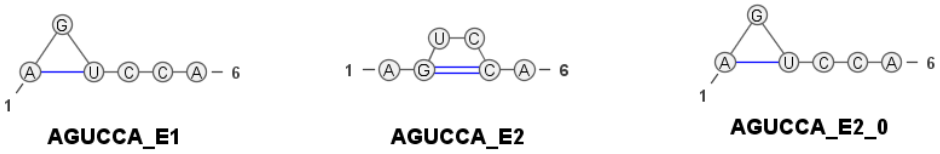

Figura 3: Ejemplo 1.

Sequence: AGUCCA				
	P_{CG}	P_{AU}	P_{GU}	Best c_1
E_1	0.00	1.00	0.00	-
E_2	1.00	0.00	0.00	0.40
Original E_2	0.00	1.00	0.00	-

Tabla 1: Resultados correspondientes al Ejemplo 1 (Figura 3).

En el caso del ejemplo de la Tabla 1 en el que la cadena es muy corta podemos ver que la única configuración que prioriza el emparejamiento C-G (el más común) es el de la función E_2.

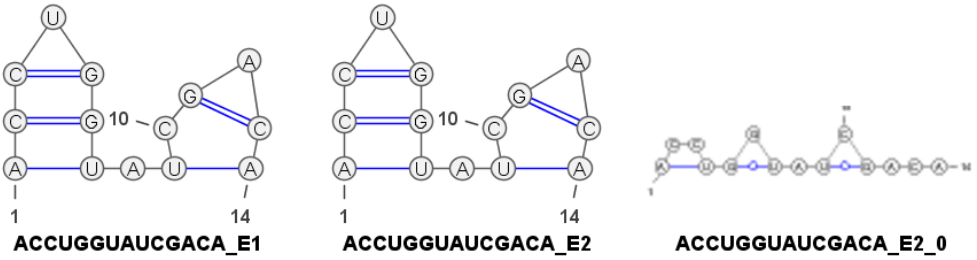

Figura 4: Ejemplo 2.

Sequence: ACCUGGUAUCGACA				
	P_{CG}	P_{AU}	P_{GU}	Best c_1
E_1	0.60	0.40	0.00	-
E_2	0.60	0.40	0.00	0.70
Original E_2	0.00	0.33	0.67	-

Tabla 2: Resultados correspondientes al Ejemplo 2 (Figura 4).

En el caso del ejemplo de la Tabla 2 se observa que tanto el uso de E_1 como E_2 dan exactamente la misma distribución y cercana a la esperada, mientras que la implementación original de E_2 se aleja bastante.

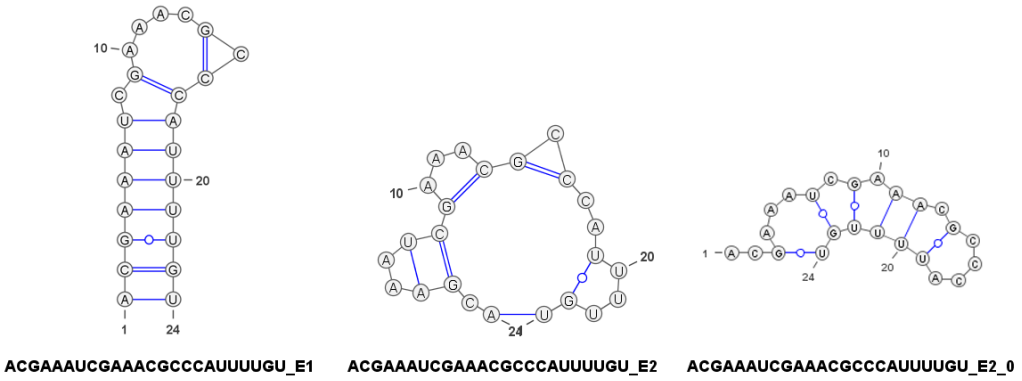

Figura 5: Ejemplo 3.

Sequence: ACGAAAUCGAAACGCCCAUUUUGU				
	P_{CG}	P_{AU}	P_{GU}	Best c_1
E_1	0.33	0.56	0.11	-
E_2	0.50	0.33	0.17	0.5
Original E_2	0.00	0.33	0.12	-

Tabla 3: Resultados correspondientes al Ejemplo 3 (Figura 5).

En el caso del ejemplo de la Tabla 3 se observa como E_2 es el modelo que mejor se ajusta a la distribución esperada.

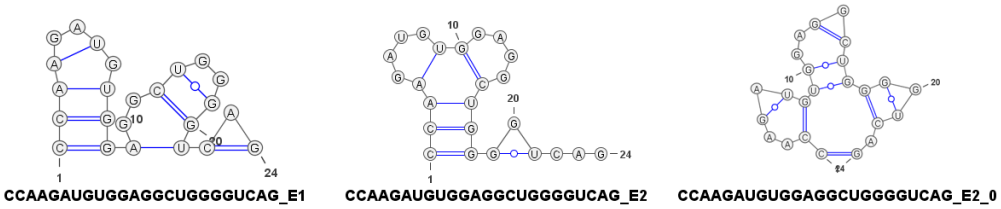

Figura 6: Ejemplo 4.

Sequence: CCAAGAUGUGGAGGCUGGGGUCAG				
	P_{CG}	P_{AU}	P_{GU}	Best c_1
E_1	0.50	0.38	0.12	-
E_2	0.50	0.33	0.17	0.5
Original E_2	0.50	0.00	0.5	-

Tabla 4: Resultados correspondientes al Ejemplo 4 (Figura 6).

Por último, en el caso del ejemplo de la Tabla 4 podemos ver que tanto el uso de E_1 como E_2 dan aproximaciones muy buenas de la distribución esperada.

En general, tras esta evaluación podemos considerar el uso de la función modificada E_2 como el mejor modelo de los tres, siendo claro que la implementación original de E_2 presenta errores.

7. Ejemplos de uso y conclusiones

El ARN está siendo considerado cada vez más importante en muchos procesos biológicos. Por ejemplo, en los últimos diez años se ha determinado que el ARN desempeña funciones en la inmunidad y el desarrollo [6] . Se están encontrando nuevos tipos de secuencias de ARN funcionales, denominadas ARN no codificantes (ARNn), mediante cribado experimental y computacional y por métodos tradicionales de biología molecular. Dada la estrecha relación entre la estructura macromolecular y la función, para comprender a fondo el mecanismo de acción de una secuencia de ARN es necesario entender la estructura del ARN por tanto su estructura secundaria.

Los principales métodos de predicción de la estructura secundaria del ARN pueden clasificarse en análisis comparativo de secuencias y algoritmos de plegado con esquemas de puntuación termodinámicos, estadísticos o probabilísticos [8] y ha sido abordada por un gran número de propuestas, presentando diferentes algoritmos y varias funciones de energía, desde el estudio de la tendencia a formar hélices [9] a métodos populares que consideran el número máximo de pares admisibles en la secuencia. Se han propuesto otras funciones energéticas como resultado de análisis probabilísticos, como la distribución media de pares de bases encontrada en estructuras secundarias reales de ARN, que se ha utilizado como objetivo de la predicción de la estructura secundaria, como hemos visto en este trabajo con el caso de E_2.

Por último, mencionar que aunque se ha añadido la regla que permite pseudo-nudos, en el trabajo no los hemos considerado ya que estos dan estructuras bastante más complejas que las que pretendemos abordar en este trabajo. En relación con esto último, en [4] pueden encontrarse propuestas clásicas de funciones de energía que consideran pseudo-nudos.

Referencias

[1] Maryam Bavarian and Veronica Dahl. Constraint based methods for biological sequence analysis. *J. Univers. Comput. Sci.*, 12(11):1500–1520, 2006.

[2] Kévin Darty, Alain Denise, and Yann Ponty. Varna: Interactive drawing and editing of the rna secondary structure. *Bioinformatics*, 25(15):1974, 2009.

[3] Andrea Formisano and Enrico Pontelli. Exploring life through logic programming: Logic programming in bioinformatics.

[4] Rune B Lyngsø and Christian NS Pedersen. Rna pseudoknot prediction in energy-based models. *Journal of computational biology*, 7(3-4):409–427, 2000.

[5] John S McCaskill. The equilibrium partition function and base pair binding probabilities for rna secondary structure. *Biopolymers: Original Research on Biomolecules*, 29(6-7):1105–1119, 1990.

[6] Gunter Meister and Thomas Tuschl. Mechanisms of gene silencing by double-stranded rna. *Nature*, 431(7006):343–349, 2004.

[7] Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.

[8] Elena Rivas. The four ingredients of single-sequence rna secondary structure prediction. a unifying perspective. *RNA biology*, 10(7):1185–1196, 2013.

[9] Ignacio Tinoco, Olke C Uhlenbeck, and Mark D Levine. Estimation of secondary structure in ribonucleic acids. *Nature*, 230(5293):362–367, 1971.