Ingeniería de la Salud
Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Práctica 4. Plegado de cadenas de ARN.

Arrabalí Cañete, Carmen Lucía

---

1. Introducción y objetivos

El algoritmo de Nussinov es un algoritmo de predicción de la estructura del ácido nucleico utilizado en biología computacional para predecir el plegamiento de una molécula de ARN que hace uso de principios de programación dinámica.

Fue uno de los primeros algoritmos desarrollados para la predicción de estructuras de ARN, ya que la combinatoria de ARN puede ser bastante costosa y complicada.

1.1. Contexto Biológico del problema

La función principal del ARN es, básicamente, servir como medio entre la información que aporta el ADN y la proteína a la que le llega toda esta información,para realizar la síntesis proteica, además, gracias a la acción del ARN la síntesis proteica se produce en su justa medida. Esta función se considera un proceso de traducción entre el ADN y la proteína.

1.2. ARN y Nussinov

Se puede definir el ARN como una secuencia S, que está formada por cuatro bases nitrogenadas, $S \in \{C, G, A, U\}$. Estas cuatro bases vienen en pares y se combinan de la siguiente manera $A \leftrightarrow U$ y $G \leftrightarrow C$.

Estos son los pares de bases mas fuertes o estables, llamados pares de bases de Watson-Crick, pero también se tiene el acoplamiento de pares menos estables, conocidos como pares de bases de Wobble, los cuales son $G \leftrightarrow U$.

Por lo tanto, la predicción de la estructura secundaria del ARN sería que dada una molécula de ARN $S = s_1, s_2, ..., s_n$, donde cada nucleótido $s_i \in \{C, G, A, U\}$, para $1 \leqslant i \leqslant n$, se tiene que encontrar un conjunto de pares de bases compatibles $P \subset \{(i, j) | 1 \leqslant i < j \leqslant n\}$ de cardinalidad máxima, respetando cualquier restricción estructural que pueda existir, en este caso se tiene que tener en cuenta la compatibilidad de los pares de bases que se explicó con anterioridad.

También hay que tener en cuenta las restricciones estructurales:

- **Sin Pseudoknots**: P no se cruza. Para cualquier $(i, j), (i', j') \in P$, no puede ocurrir que $i < i' < j < j'$.
- **Tamaño del bucle o Loop Size**: Para cualquier $(i, j) \in P$, $j - i > L$, donde L es el tamaño mínimo del bucle del tallo (también conocido como horquilla o hairpin-loop).

1.3. Ecuación de Bellman

$$N_{i,j} = \begin{cases} 0 & j - i \leqslant L \\ \max\left(N_{i,j-1}, \max\limits_{\substack{i \leqslant k < j-L \\ \text{pair}(s_k, s_j)}} (N_{i,k-1} + N_{k+1,j-1} + 1) \right) & j - i > L \end{cases}$$

Figura 1: Ecuación de Bellman

2. Configuración del equipo

Se ha realizado la implementación en un equipo con un sistema operativo Windows 10 Home 64 bits, con un procesador Intel(R) Core (TM) i7-6500U CPU @ 2.50GHz 2.59 GHz (4 CPUs) y un disco SSD de 480GB y 12GB de RAM. La versión Java corresponde a la número 17.

3. Implemetación

En el primer método de la implementación, *protected String _run* de la clase *Nussinov* (ver apartado 3.1) recibe como parámetros un String con la cadena de ARN y el tamaño mínimo de plegado, en la realidad suele estar entre 4 y 8, pero el algoritmo puede aceptar cualquier valor. Se encarga de implementar la ecuación de Bellman (véase figura 1) y de devolver la secuencia de ARN plegada, es decir, se encarga de calcular la matriz de decisiones D y hacer una llamada al segundo método *private String reconstructSolution* (ver apartado 3.2) de la misma clase, el cual se encarga de, recibiendo como parámetros la matriz de decisiones D y los índices i y j desde los que se debe reconstruir la solución, devolver la secuencia de ARN como un String haciendo uso de la notación Dot-Backet.

En el primer bucle, línea 7 del primer método (ver apartado 3.1) se encarga de comprobar si los nucleótidos que se estudian son compatibles entre sí en función de las restricciones comentadas anteriormente.

Los siguientes dos bucles, línea 13, se encargan del caso base de la ecuación de Bellman (véase figura 1), cuando $M(i, j) = 0$ si $j \leqslant i + minloopSize$.

Por último, en la línea 21, los tres bucles anidados se encargan principalmente de la segunda parte de la ecuación de Bellman (véase figura 1), que trata del caso general cuando $M(i, j) = max(M(i, j-1), max(M(i, k-1) + M(k+1, j-1) + 1$ donde $i \leqslant k < j - minloopSize$ y s_k y s_j son complementarios)) si $j > i + minloopSize$

3.1. Método *protected String _run* de la clase *Nussinov*

```
protected String _run(String rnaSeq, int minLoopSize) {
  int n = rnaSeq.length();      // number of nucleotides in the sequence
  int[][] M = new int[n][n+1];    // to store costs
  int[][] D = new int[n][n];      // to store decisions
  boolean[][] B = new boolean[n][n];  // to precompute which base pairs match

  for (int i=0; i<n; i++) {
    for (int j=i+1; j<n; j++) {
      B[i][j] = isCompatible(rnaSeq.charAt(i), rnaSeq.charAt(j));
    }
  }

  for (int i = 0; i < n; i++) {
    M[i][i] = 0;         // M(i,i-1) = 0
    for (int j = i, k = 0; (k <= minLoopSize) && (j<n); j++, k++) {
      M[i][j+1] = 0;       // M(i,j) = 0;
      D[i][j] = -1;      // -1 => unpaired
    }
  }

  for (int j = 0; j < n; j++) {
    for (int i = 0; i < j; i++) {
      int newJ = j+1;
      M[i][newJ] = M[i][newJ-1];
      int bestK = -1;
      for (int k = i; k < newJ-minLoopSize-1; k++) {
        int newK = k+1;
        if (B[k][j] && M[i][newJ] < M[i][newK-1] + M[k+1][newJ-1] + 1) {
          M[i][j+1] = M[i][newK-1] + M[k+1][newJ-1] + 1 ;
          bestK = k;
        }
      }
      D[i][j] = bestK;
    }
  }
  // Hay un trozo de codigo adicional que se encarga de imprimir por pantalla
  // la matriz de decisiones.

  // Reconstruction of the optimal solution
  return reconstructSolution(D, 0, n-1);
}
```

3.2. Método *private String reconstructSolution* de la clase *Nussinov*

```java
private String reconstructSolution (int[][] D, int i, int j) {
  // TODO
  // returns the folding in dot-bracket notation.
  // use a recursive approach for simplicity
  String result = "";
  if (j>i-1) {
    int k = D[i][j];
    if (k>-1) {
      return reconstructSolution(D,i,k-1) +"(" + reconstructSolution(D,k+1,j-1) +")";
    }
    else {
      return reconstructSolution(D, i, j-1) + ".";
    }
  }else if(j == i){
    result = ".";
  }
  return result;
}

```

4. Resultados y Conclusiones

Para poder analizar correctamente los resultados, el método main recibe en los argumentos de entrada, varios modos de ejecución entre los que se encuentra $-r$ <*init_length*> <*doubling*> <*num_seq*> <*minlop*> el cual genera un archivo con secuencias de ARN aleatorias con distintas longitudes. Se generan *num_seq* secuencias de cada longitud, que empezará con *init_length* y se irá doblando tantas veces como indique el parámetro *doubling*. Por último, se pasa como parámetro la separación mínima para plegar nucleótidos.

Al utilizar ese modo de ejecución, se genera un archivo con los diferentes tiempos de ejecución, el cual se llama un archivo *folding.txt*.

Por otro lado, entre los recursos aportados en el Campus Virtual se tiene un archivo llamado *analyze.R* que contiene un script el cual se encarga de generar una gráfica haciendo uso del archivo .txt generado anteriormente con el modo de ejecución $-$r.

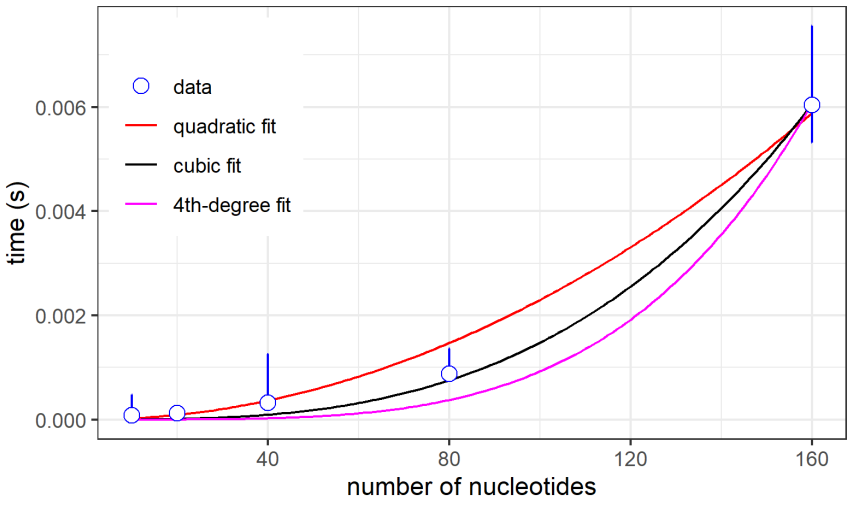

Figura 2: Resultado de la ejecución del algoritmo con los parámetros utilizados.

Para poder generar un archivo con unos datos asequibles para el algoritmo y para el equipo que se utiliza, se han dado los siguientes valores:

- <**init_length**>: 10
- <**num_seq**>: 20
- <**doubling**>: 5
- <**minlop**>: 6

Como se puede observar en la figura 2, cuanto mayor sea el número de nucleótidos de la cadena con la que se quiere obtener el plegamiento, el tiempo que tarda el algoritmo en obtener el resultado también es mayor, tanto de forma cuadrática, como cúbica, como, incluso, de cuarto grado, hasta llegar al límite del número máximo de nucleótidos que se pasan.