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

Bioinformática

Informe Práctica 2. Árbol filogenético y mínima parsimonia.

Arrabalí Cañete, Carmen Lucía

---

1. Introducción

Esta práctica consiste en calcular la mínima parsimonia con la que se permite estimar una hipótesis con el menor número de cambios evolutivos según un árbol binario filogenético, el cual es una forma de representar las relaciones evoltivas de diferentes especies.

2. Implementación

Para la implementación de este algoritmo se propone crear dos funciones. Una de ellas *computeParsimonyScore* que recibe un árbol filogenético y devuelve el valor de la parsimonia, el cual se calcula haciendo uso de una función auxiliar llamada *computeScoreRec*, que dado un árbol filogenético, calcula de forma recursiva la matriz de puntuación s del nodo raíz.

Adicionalmente a esos dos métodos, se ha de crear la interfaz *BinaryTree<E>* que crea el elemento raíz, el subárbol izquierdo y el subárbol derecho y, también, comprueba si el árbol inicial está vacío.

En el proyecto también se incluye la Clase *PhylogeneticTree* que se encarga de tener todas las características de los árboles filogenéticos.

2.1. Variables necesarias para la implementación de las funciones

```
1  public static char alphabet[] = {'A','C', 'G', 'T'};
2  public static int delta[][] = {{0, 1, 1, 1}, {1, 0, 1, 1}, {1, 1, 0, 1}, {1, 1, 1, 0}};
3  public static int m = 0;
4  public static char label[] = new char[m];
```

Donde:

- *alphabet*[] incluye las letras necesarias de los nucleótidos que componen nuestro organismo
- *delta*[][] es una matriz de tamaño *nxn* que almacena el coste de cambiar un nucleótido por otro. Se tomará como coste 0 si no se cambian los nucleótidos y de coste 1 si se cambia.
- *m* será la longitud de la etiqueta que lleve cada nodo con la información correspondiente, la cual se pasa a Char en la siguiente línea.

2.2. Función *computeParsimonyScore*

```
public static int computeParsimonyScore(PhylogeneticTree t) {
  int parsimony = 0;
  int min = 0;
  int s [][] = computeScoreRec(t);
  for(int i = 0; i <= m-1; i++) {
    min = s[i][0];
    for(int j = 1; j <= alphabet.length-1; j++) {
      if(s[i][j] < min) {
        min = s[i][j];
      }
    } parsimony += min;
  }
  return parsimony;
}
```

Donde se tiene que en un principio tanto la parsimonia como el valor mínimo son cero y, además, una matriz s vacía que llama al método auxiliar *computeScoreRec*. Se recorre la matriz con un bulce *for* con la condición de que la variable i sea menor o igual que la etiqueta menos uno que, en el caso de cumplir las condiciones, el mínimo ahora sería la matriz $min = s[i][0]$.

Después, dentro de ese bucle *for*, hay otro bucle *for* que se encarga de recorrer la misma matriz de nuevo, pero esta vez, la condición es que sea menor o igual que la longitud total del alfabeto menos uno, que daría como resultado la matriz $min = s[i][j]$ en el caso de que $s[i][j]$ sea menor que el mínimo anterior. La variable j se inicializa en 1 mientras que la i se inicializa en 0.

Se actualiza el valor de la parsimonia añadiéndole el mínimo y se devuelve valor de la misma.

2.3. Función *computeScoreRec*

```
public static int[][] computeScoreRec(PhylogeneticTree t) {
  int s [][] = new int[m][alphabet.length];

  if(t.isLeaf()) {
    label = t.root().toCharArray();
    for(int i = 0; i <= m-1; i++) {
      for(int j = 0; j <= alphabet.length-1; j++) {
        if(label[i] == alphabet[j]) {
          s[i][j] = 0; //No cambia el nucleotido
        } else {
          s[i][j] = Integer.MAX_VALUE - 1; // si se cambia el nucleotido
        }
      }
    }
  } else{
    int sl [][] = computeScoreRec(t.left());
    int sr [][] = computeScoreRec(t.right());
    for(int i = 0; i <= m-1; i++) {
      for(int j = 0; j <= alphabet.length-1; j++) {
        int minLeft = Integer.MAX_VALUE - 1;
        int minRight = Integer.MAX_VALUE - 1;
        for(int k = 0; k <= alphabet.length - 1; k++) {
          if(sl[i][k] + delta[j][k] < minLeft) {
            minLeft = sl[i][k] + delta[j][k];
          }
          if(sr[i][k] + delta[j][k] < minRight) {
            minRight = sr[i][k] + delta[j][k];
```

```
28              }
29            }
30            s[i][j] = minLeft + minRight;
31          }
32        }
33      }
34      return s;
35    }
```

Esta es una función auxiliar que se encarga de calcular, de manera recursiva y dado un árbol filogenético, la matriz de puntuación *s* del nodo raíz.

Lo primero que comprueba es si es un nodo hoja o no, en el caso de que si lo sea, la etiqueta se asocia al nodo raíz y luego va iterando por la matriz hasta *m-1* y hasta *alphabet.length-1* que en el caso de que la posición *i* de la etiqueta sea igual que la posición *j* del alfabeto, el elemento de *s* en las posiciones *i* y *j* no cambia su nucleótido, en cambio, si no es igual en posición, si que cambia y el elemento seleccionado sería infinito, definido en el lenguaje como Integer.MAXVALUE-1.

Por otro lado, en el caso de que no sea hoja, por lo tanto es nodo interno, comprueba si hay mínimos.

Devuelve en todos los casos la matriz de puntuación del nodo raíz.