

Informe Trabajo Extra → Cálculo de Primers Universales

1. Descripción y objetivos

La finalidad de este programa reside en la determinación de un cebador universal óptimo que consiga la máxima hibridación posible con todas las secuencias que se encuentren alineadas en un fichero de formato fasta. Este ejercicio resulta especialmente útil cuando las secuencias que se encuentran alineadas en el fichero tienen algún tipo de relación entre ellas. Es decir, que todas ellas pertenezcan a ortólogos de un mismo gen o que hagan referencia a las distintas cepas de un determinado virus.

De esta forma, el cebador otorgado por nuestro programa permitirá realizar con la mayor fiabilidad posible técnicas de laboratorio como la detección, secuenciación o amplificación por PCR cuando no se sepa a cuál de las secuencias incluidas en el fichero nos estemos enfrentando. Esto es un caso muy común en las pruebas de PCR cuando no sabemos la cepa que puede contener el paciente de estudio, por eso será necesaria la existencia de una sonda universal capaz de detectar cualquiera de ellas.

2. Restricciones

La elaboración de un buen primer debe atender principalmente a cuatro parámetros:

- a) **Complementariedad** → Es el principal requisito que debe cumplir un primer, el cual se encontrará por encima de todos los demás. Para que un cebador hibride con el molde necesitamos que sea lo más complementario posible al mismo. Este requisito en caso de tener un único gen de secuencia conocida no supone ningún problema, ya que nada te impide determinar la secuencia 100% complementaria al molde. Sin embargo en nuestro caso, dado que tenemos varias secuencias deberemos buscar la zona de las mismas que maximice el número de columnas conservadas y así asegurarnos de que la secuencia complementaria que diseñemos sea realmente complementaria a todas ellas.
- b) **Afinidad extremo 3'** → Es otro requisito fundamental muy relacionado con el anterior. Esta restricción consiste en que en el extremo 3' de nuestro cebador debe darse obligatoriamente un 100% de complementariedad con todas las secuencias, en caso de que esto no sea posible, nuestro programa no podrá ofrecer una solución.
- c) **Longitud** → Por norma general la longitud de un cebador debe encontrarse entre los 17 y 23 nucleótidos para asegurarnos su estabilidad durante su manipulación en el laboratorio.
- d) **Temperatura** → Además de lo dicho anteriormente, con el fin de evitar problemas como la desnaturalización de nuestro cebador o la amplificación inespecífica, necesitamos que nuestro primer tenga una temperatura de fusión asociada de en torno 60 grados. Esta temperatura se calcula mediante la fórmula de Wallace:

$$T_m = 2 (A+T) + 4 (G+C)$$

3. Procedimiento

El primer paso consiste en la lectura del fichero fasta proporcionado como parámetro de entrada en el programa. Dado que a priori no conocemos el número de secuencias contenidas en el archivo, haremos uso de una lista de String para su almacenamiento. Para ello, recorreremos línea a línea el documento identificando el comienzo de una nueva secuencia mediante el carácter “>”. Cada nueva secuencia será añadida a la lista para que esta pueda ser posteriormente convertida en una matriz de caracteres.

Tras la construcción de la matriz, comenzamos el proceso de búsqueda mediante la creación de dos vectores. El primero de ellos va a contener a una secuencia consenso formada por el nucleótido más frecuente de cada columna. El segundo, tendrá en cada posición el porcentaje de fiabilidad asociado al correspondiente nucleótido de la secuencia consenso. En caso de empate, el programa está diseñado para anteponer las letras G y C ante las A y T puesto que las primeras consiguen hibridar con una mayor fuerza respecto a las otras dos. Por supuesto en caso de empate nunca se escribiría un gap ya que perderíamos información. Un simple ejemplo sería:

```
SEC 1: 5' ACTA--CAAT 3'
SEC 2: 5' AGC-CC-CTT 3'
SEC 3: 5' GCT-AC-AAT 3'
SEC 4: 5' GCTA--CAAT 3'
SEC CONSENSO: 5' GCTA-CCAAT 3'
```

Una vez conseguidos los dos vectores mencionados anteriormente nos disponemos a recorrer el vector de fiabilidades en búsqueda de la zona que sea capaz de maximizar la suma de los porcentajes de sus posiciones y que a la vez cumpla con todos los requisitos descritos anteriormente. Esto se debe a que aunque dichas condiciones sean para el cebador y no para la secuencia molde, dado que el primer va a ser la secuencia inversa complementaria de la misma el cumplimiento en ella significará el igual cumplimiento en el cebador.

Para ello, haremos uso de un bucle for en el que para cada iteración recorreremos otro bucle de tipo while para simular el crecimiento desde esa posición de nuestro oligo. Para que dicho crecimiento siga adelante y que por tanto sigamos dentro del bucle while, no podemos encontrarnos con ningún gap en la secuencia consenso y los 5 primeros nucleótidos durante dicho crecimiento deben tener una fiabilidad de 1. Esto último se debe a la condición de necesitar el 100% de complementariedad en el extremo 3' del cebador.

Durante ese bucle while se aplicará una condición de tal forma que si el crecimiento ha conseguido llegar a los 17 nucleótidos con una temperatura superior a 55 °C y además superar la mejor marca hasta el momento, se guardará temporalmente como solución hasta que otra la mejore. En estos casos, el bucle while sigue analizando hasta que nos encontramos con un gap, hemos superado los 23 nucleótidos o la temperatura ha alcanzado los 60°C.

No debemos olvidar que la solución de este proceso hace referencia a la zona de la secuencia que mayor fiabilidad nos puede ofrecer. Por ello, necesitamos calcular la secuencia inversa complementaria a la que tenemos para poder bautizarla como el cebador universal resultante del programa.

Finalmente nuestro programa proporcionará como resultado:

- Secuencia consenso.
- Vector fiabilidades.
- Molde para diseñar el primer, junto con su posición dentro de la secuencia consenso.
- Cebador universal con su longitud y temperatura de fusión asociada.
- Hibridaciones del cebador en cada secuencia para comprobar la eficacia de nuestro resultado.

4. Exposición del código

Dada la extensión de nuestro programa mostraré las funciones que resulten más entendibles y significativas para complementar el procedimiento expuesto anteriormente. Para la construcción de nuestro programa he hecho uso de dos clases distintas. La primera de ellas denominada **Lectura.java** se encarga de recibir el fichero de entrada y convertirlo en una matriz de caracteres. Por otro lado la clase **Cebador.java** se centrará en el cálculo de nuestro cebador universal y proporcionar la salida del sistema.

Para la primera clase tenemos una única función cuyo código es el siguiente:

```
public char[][] leeFichero(String fichero) {
    List<String> listaRes = new ArrayList<>();
    String res_i = "";
    try {
        Scanner sc = new Scanner(new File(fichero));
        sc.nextLine();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            if (linea.startsWith(">")) {
                listaRes.add(res_i);
                res_i = "";
            } else {
                res_i += linea;
            }
        }
        listaRes.add(res_i);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    if (listaRes.size() < 2) {
        throw new RuntimeException("El fichero introducido no es correcto");
    }
    char[][] res = new char[listaRes.size()][listaRes.get(0).length()];
    for (int i = 0; i < listaRes.size(); i++) {
        if (i < listaRes.size() - 1 && listaRes.get(i).length() != listaRes.get(i+1).length()) {
            throw new RuntimeException("El fichero introducido tiene secuencias de distinta longitud");
        }
        for (int j = 0; j < listaRes.get(i).length(); j++) {
            res[i][j] = listaRes.get(i).charAt(j);
        }
    }
    return res;
}
```

Como podemos ver la función se divide en dos partes, en la primera de ellas nos encargamos de la lectura secuencial del fichero proporcionado como parámetro de entrada y revisamos que dicho documento contenga a más de dos secuencias, ya que en caso contrario no podremos llevar a cabo nuestro propósito.

En la segunda parte convertimos la lista inicial de secuencias en una matriz de caracteres, comprobando que todas ellas poseen la misma longitud y que por tanto se encuentran alineadas.

La siguiente clase es un poco más compleja y por ello solo mostraré su método principal. Éste hace uso de funciones auxiliares cuyo propósito viene descrito mediante su propio nombre.

```
public void compute(char[][] sec) {
    char[] secuenciaConsenso = new char[sec[0].length];
    double[] fiabilidad = new double[sec[0].length];
    calculaSecConsensoYFiabilidad(sec, secuenciaConsenso, fiabilidad);
    int[] posInPosFinTemp = calculaPosMaxFiabilidad(secuenciaConsenso, fiabilidad);
    int posInicial = posInPosFinTemp[0];
    int posFinal = posInPosFinTemp[1];
    int temperatura = posInPosFinTemp[2];
}
```

```

String molde = String.valueOf(secuenciaConsenso).substring(posInicial, posFinal);
cebador = calculaInvCom(molde);
String hibridaciones = calculaHibridaciones(sec, posInicial, posFinal, molde);
String StringFiabilidades = "";
for (double v : fiabilidad) {
    StringFiabilidades += " " + v;
}
resultadoCompleto = "Secuencia consenso: 5'- " + String.valueOf(secuenciaConsenso) + " - 3' \n"
    + "Fiabilidades: " + StringFiabilidades + "\n"
    + "Molde Secuencia Consenso: 5'- " + molde + " - 3'" + " Posición: " + posInicial + " - " + posFinal + "\n"
    + "Cebador universal: 5'- " + cebador + " -3'"
    + " Longitud: " + (posFinal - posInicial) + " Temperatura: " + temperatura + "°C" + "\n"
    + "Hibridaciones del cebador en cada secuencia: \n"
    + hibridaciones;
}

```

En él podemos ver que se sigue el procedimiento anteriormente descrito. Comenzamos con la declaración de los vectores de la secuencia consenso y la fiabilidad, buscamos la zona de máxima complementariedad, determinamos el molde y finalmente calculamos la secuencia inversa complementaria para obtener el cebador.

Tras todo ello construimos un String con toda la información almacenada que proporcionaremos como resultado en pantalla y será grabado en el fichero de salida.

Para profundizar en el algoritmo de búsqueda procedo a mostrar el trozo de código correspondiente a los bucles mencionados en el anterior apartado.

```

for (int i = 0; i < fiabilidad.length; i++) {
    gap = false;
    hibridacionTresPrimaCebador = true;
    valorZonaActual = 0;
    tempActual = 0;
    j = 0;
    while (!gap && hibridacionTresPrimaCebador && j < 23 && tempActual < 60 && i < fiabilidad.length - 23) {
        if (secuenciaConsenso[i + j] == '-') {
            gap = true;
        }
        if (j < 6 && fiabilidad[i + j] < 1) {
            hibridacionTresPrimaCebador = false;
        }
        tempActual += temperaturas.get(secuenciaConsenso[i + j]);
        valorZonaActual += fiabilidad[i + j];
        j++;
        if (!gap && hibridacionTresPrimaCebador && valorZonaActual / j > maxAlcanzado && j > 17 && tempActual > 55) {
            maxAlcanzado = valorZonaActual / j;
            res[0] = i;
            res[1] = i + j;
            res[2] = tempActual;
        }
    }
}
if (res[1] == 0) {
    throw new RuntimeException("No es posible la construcción de un primer universal");
}

```

Donde lo único que me queda por comentar es que en caso de no haber alcanzado en ningún momento las condiciones mínimas necesarias para el diseño del primer, se lanzará una excepción informando de dicha situación.

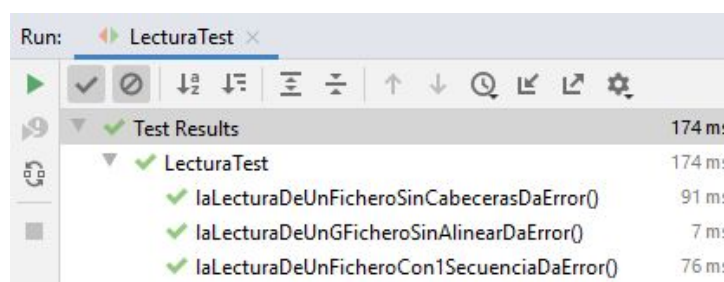
5. Test

Para este programa dada la complejidad del resultado he realizado dos clases Test que realizan comprobaciones bastantes sencillas. La mayoría de las pruebas están destinadas a la detección de errores:

- **LecturaTest** → Como su nombre indica se centra en ejecutar casos de prueba para la clase Lectura que se encarga del análisis del fichero de entrada. En este caso he diseñado tres pruebas consistentes en la entrada de un fichero sin cabeceras, otro con una única secuencia y finalmente un tercero donde las secuencias no se encuentran alineadas. Por tanto, la respuesta que nos debe dar el programa en los tres casos es el lanzamiento de una excepción.

```
public class LecturaTest {  
    @Test  
    public void laLecturaDeUnFicheroCon1SecuenciaDaError(){  
        Lectura l = new Lectura();  
        assertThrows(RuntimeException.class, () -> l.leeFichero("Fichero1secuencia.fasta"));  
    }  
    @Test  
    public void laLecturaDeUnFicheroSinCabecerasDaError(){  
        Lectura l = new Lectura();  
        assertThrows(RuntimeException.class, () -> l.leeFichero("FicheroSinCabeceras.fasta"));  
    }  
    @Test  
    public void laLecturaDeUnGFicheroSinAlinearDaError(){  
        Lectura l = new Lectura();  
        assertThrows(RuntimeException.class, () -> l.leeFichero("FicheroSinAlinear.fasta"));  
    }  
}
```

Si las probamos vemos como efectivamente obtenemos lo esperado:



Run: LecturaTest	
Test Results 174 ms	
✓ LecturaTest	174 ms
✓ laLecturaDeUnFicheroSinCabecerasDaError()	91 ms
✓ laLecturaDeUnGFicheroSinAlinearDaError()	7 ms
✓ laLecturaDeUnFicheroCon1SecuenciaDaError()	76 ms

- **CebadorTest** → En este caso probaremos el funcionamiento de la clase cebador. Para ello he diseñado dos pruebas donde una de ellas está destinada al fracaso y la otra al claro éxito del programa.

La que ha sido diseñada para que el programa falle consiste en tres secuencias estratégicamente montadas para que la secuencia consenso impida al algoritmo implementado obtener una solución.

Por el otro lado, la otra consiste en dos secuencias idénticas llenas de gaps menos 20 nucleótidos seguidos cuya temperatura se encuentra entre 55 y 60 grados, lo cual define claramente la única solución posible.


```

public class CebadorTest {

    @Test
    public void elCebadorDeUnFicheroConMuchosGapsDaError() {
        Lectura l = new Lectura();
        char[][] secuencia = l.leeFichero("FicheroMuchosGaps.fasta");
        Cebador c = new Cebador();
        assertThrows(RuntimeException.class, () -> c.compute(secuencia));
    }

    @Test
    public void elCebadorDeUnFicheroConTodoGapsMenos20NucleotidosDaLaComplementariaDeEllosMismos() {
        Lectura l = new Lectura();
        char[][] secuencia = l.leeFichero("TodoGapsMenos20NucleótidosAlineados.fasta");
        Cebador c = new Cebador();
        c.compute(secuencia);
        assertEquals( expected: "ACGCGTCATATATCTGCGT", c.getCebador());
    }
}

```

Si ejecutamos la clase vemos como ambas pruebas obtienen el resultado esperado:

Run: CebadorTest	
Test Results	151 ms
CebadorTest	151 ms
elCebadorDeUnFicheroConMuchosGapsDaError()	125 ms
elCebadorDeUnFicheroConTodoGapsMenos20NucleotidosDaLaComplementariaDeEllosMismos()	26 ms

6. Ejemplos

Para la ejecución de nuestro programa, aparte de los ejemplos ejecutables en el entorno he construido un fichero jar que permite su uso a través de un terminal introduciendo como parámetro el fichero de entrada.

Además de los ficheros fasta contenidos en el proyecto, se adjuntará a este informe los documentos en formato pdf correspondientes a las alineaciones de las secuencias de entrada para cada ejemplo.

a. Ejecutables en el entorno

- Ejemplo 1:

```

"C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\i
Secuencia consenso: 5'- -----C--GCCCTCACACCCGTGCTGCCACGGAACCTGCGTTTAGGGGTGTCACCCCT
Fiabilidades: 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.8 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6
Molde Secuencia Consenso: 5'- TGGCAGGTCAGCCTGCAC - 3' Posición: 1110 - 1128
Cebador universal: 5'- GTGCAGGCTGACCTGCCA -3' Longitud: 18 Temperatura: 60°C
Hibridaciones del cebador en cada secuencia:
Sec 1: 5'- TGGCAGGTCAGCCTGCAT -3'
Sec 2: 5'- TGGCAGGTCAGCCTGCAC -3'
Sec 3: 5'- TGGCAGGTCAGTCTGCAC -3'
Sec 4: 5'- TGGCAGGTCAGCCTGCAC -3'
Sec 5: 5'- TGGCAGGTCAGCCTGCAC -3'

```

El resultado se ha guardado en el fichero CebadorUniversalEjemplo1.fasta.txt

- Ejemplo 2:

```
C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\
Secuencia consenso: 5'- -----C--CC--T-----A--ACATAC-
Fiabilidades: 0.8333333333333334 0.8333333333333334 0.8333333333333334 0.8333333333333334 0.8333333333333334 0.8333333333333334 0.
Molde Secuencia Consenso: 5'- GGCTTCTTCTACAACCCC - 3' Posición: 290 - 308
Cebador universal: 5'- GGGGTTGTAGAAGAAGCC -3' Longitud: 18 Temperatura: 56°C
Hibridaciones del cebador en cada secuencia:
Sec 1: 5'- GGCTTCTTCTACTCAGCC -3'
Sec 2: 5'- GGCTTCTTCTACTCAGCC -3'
Sec 3: 5'- GGCTTCTTCTACAACCCC -3'
Sec 4: 5'- GGCTTCTTCTACAACCCC -3'
Sec 5: 5'- GGCTTCTTCTACAACCCC -3'
Sec 6: 5'- GGCTTCTTTACAACCCC -3'
```

El resultado se ha guardado en el fichero CebadorUniversalEjemplo2.fasta.txt

- Ejemplo 3:

```
C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition\
Secuencia consenso: 5'- -----TGGAG-GCATTTCGTCGCCAGCGGTCTGACTTTCGCGGCGCCGCCCGGC
Fiabilidades: 0.6666666666666666 0.6666666666666666 0.6666666666666666 0.6666666666666666 0.6666666666666666 0.6666666666666666
Molde Secuencia Consenso: 5'- CATCACATTGGAATGCAGAT - 3' Posición: 610 - 630
Cebador universal: 5'- ATCTGCATTCCAATGTGATG -3' Longitud: 20 Temperatura: 56°C
Hibridaciones del cebador en cada secuencia:
Sec 1: 5'- CATCACCTGGCATGCAGAT -3'
Sec 2: 5'- CATCACATTGGAATGCAGAT -3'
Sec 3: 5'- CATCACATTGGAATGCAGAT -3'
```

El resultado se ha guardado en el fichero CebadorUniversalEjemplo3.fasta.txt

b. Ejecutados por consola mediante el uso del archivo jar

- Ejemplo 4:

```
C:\Users\Usuario\IdeaProjects\PAB-PrimerUniversal\out\artifacts\PAB_PrimerUniversal_jar>java -jar PAB-PrimerUniversal.jar "Ejemplo4.fasta"
```


[illegible][illegible]

Molde Secuencia Consenso: 5'- TTTGAGGTGCGTGTTTGTGC - 3' Posición: 998 - 1018
Cebador universal: 5'- GCACAAACACGCACCTCAAA -3' Longitud: 20 Temperatura: 60°C
Hibridaciones del cebador en cada secuencia:
Sec 1: 5'- TTTGAGGTGCGTGTGTGTGC -3'
Sec 2: 5'- TTTGAGGTTTCGTGTTTGTGC -3'
Sec 3: 5'- TTTGAGGTGCGTGTTTGTGC -3'
Sec 4: 5'- TTTGAGGTGCGTGTTTGTGC -3'

El resultado se ha guardado en el fichero CebadorUniversalEjemplo4.fasta.txt

Donde podemos ver que para la versión de consola he decidido quitar los colores de las letras ya que no todas disponen de esa opción y que en dicho caso el resultado se muestra ilegible.

Por último, decir que tras cada una de estas ejecuciones tal y como se muestra al final de cada captura, el resultado es grabado en un fichero con extensión txt que también proporcionaré adjuntos al informe.



- CebadorUniversalEjemplo1.fasta.txt
- CebadorUniversalEjemplo2.fasta.txt
- CebadorUniversalEjemplo3.fasta.txt
- CebadorUniversalEjemplo4.fasta.txt