Bioinformática avanzada: problemas y algoritmos

SESIÓN 2

Rodrigo Santamaría

2014

Nos sobran los motivos

BÚSQUEDA DE MOTIVOS

K-mero

- Secuencia de k nucleótidos, ***o***
- Todas las posibles secuencias de k nucleótidos en una determinada secuencia L
  - L-k+1 k-meros
- Es un concepto muy utilizado en ensamblado y alineamiento de secuencias

Ejercicio

- Implementar la función `countKmers`:
  - **entrada**:
    - `seq` (cadena original)
    - `k` (tamaño del k-mero)
    - `n` (nº de ocurrencias)
  - **salida**: diccionario que tenga como claves los k-meros y como valores el número de veces que aparece en `seq`. Sólo aparecerán en el diccionario los `k`-meros que aparezcan `n` o más veces

Ejercicio

- Ejemplo
  - **entrada**:
    - `ACGTTGCATGTCGCATGATGCATGAGAGCT`
    - `4`
    - `2`
  - **salida**:
    - `{'GCAT': 3, 'ATGA': 2, 'TGCA': 2, 'CATG': 3}`

Cronometrando

```
import time
t0=time.clock()
[...]                          #órdene(s) a cronometrar
print(time.clock()-t0)  #resultado en segundos
```

Los problemas bioinformáticos suelen ser cosa fina. Requieren mucha memoria y capacidad de cálculo. Por eso es muy importante monitorizar nuestro código, para detectar las zonas más lentas y optimizarlas.

¿Cuánto tarda en ejecutar tu código el ejercicio anterior? ¿Y si lo ejecutas sobre la cadena *oriC*?

Generalmente a los biólogos no les importa si un programa tarda un par de horas, están acostumbrados a ello en el laboratorio. !Para un informático, que algo tarde más de 5 minutos es una pérdida de tiempo!

Metiéndonos con el genoma

- La prueba de fuego para el rendimiento de un programa suele ser el genoma completo
- Aquí hay unos cuantos para hacer pruebas
  - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/genomas/

Metiéndonos con los ficheros

- Como os imaginaréis, no es operativo copiar y pegar toda la secuencia de un genoma en el programa
  - Recurrimos a la lectura de archivos

```
f=open("/Users/rodri/genomas/Vibrio_cholerae.txt", "r")
vc=f.readlines()    #lee las líneas del archivo a una lista
vc=vc[0]           #el fichero de genoma sólo tiene una línea
len(vc[0])          #(muy larga, claro)
#vc     #No se os ocurra mostrarla por pantalla, es enooorme!
f.close()       #Cuando salís de casa, cerrad con llave
```

¿Cuánto tarda tu código en encontrar los 10-meros que aparecen más de 10 veces en *Vibrio cholerae*?

Motivos

- Los motivos son k-meros que aparecen recurrentemente en algún aspecto biológico

¿Qué 9-meros frecuentes podemos encontrar en la región *oriC* de *Vibrio cholerae*?

¡Eureka! Hay un patrón bastante claro: las secuencias CTTGATCAT y su reversa complementaria ATGATCAAG aparecen 3 veces cada una.

Pero ojo, no podemos saltar rápidamente a la conclusión de que hay un motivo para todos los genomas bacterianos que caracterice orígenes de replicación. E incluso podría ser que el patrón encontrado para *V cholerae* sea sólo debido al azar y no tenga significancia estadística.

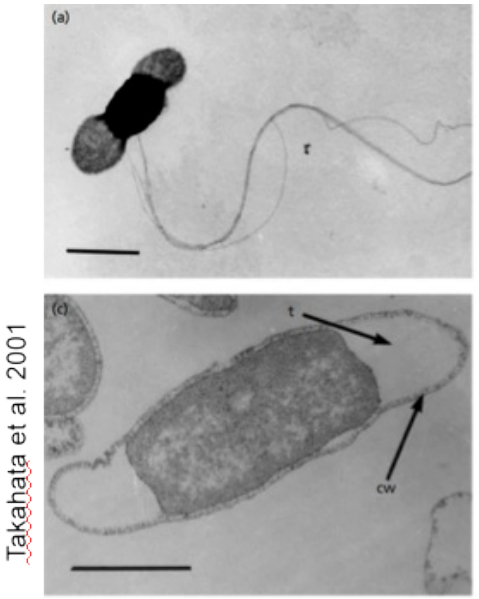
Deberemos comprobar estas dos cuestiones

Motivos

- Buscar 9-mers que aparezcan 3 o más veces en el *oriC* de *Thermotoga petrophila*
  - ¿Son los mismos que en *V Cholerae*?

Thermotoga petrophila es una bacteria que resiste temperaturas muy altas, gracias a su cubierta (la 'toga térmica')

Descubierta en un depósito de petróleo, a 70º C



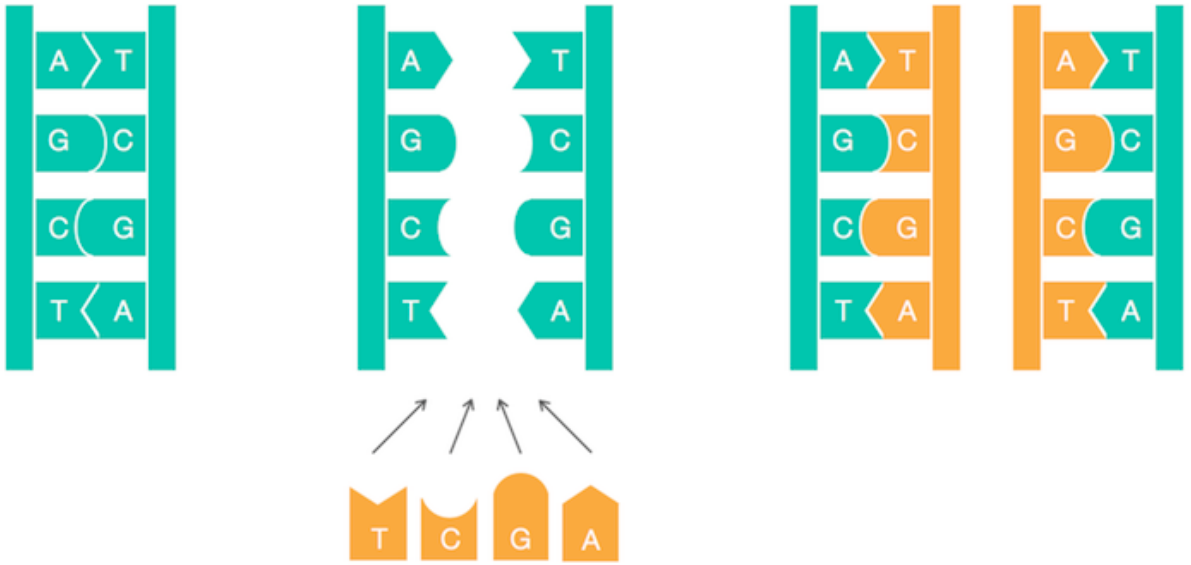
Takahata et al. 2001

En busca del oriC perdido

IDENTIFICANDO ORIC

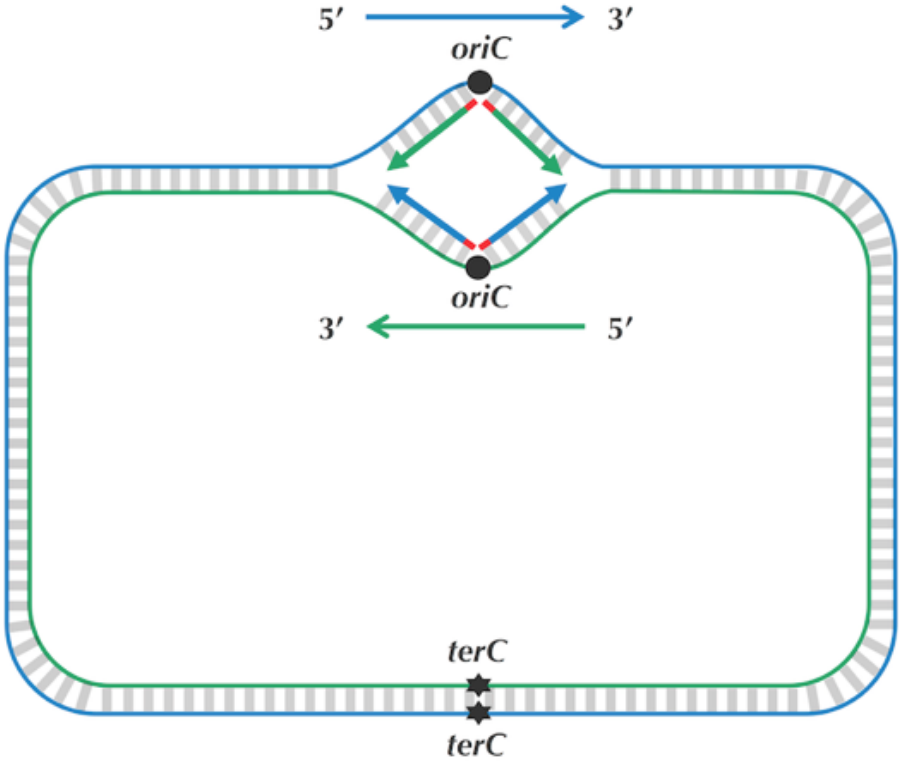
Replicación

- Uno de los procesos más importantes de la célula
  - Necesario antes de dividirse, para dar una copia genética a cada célula hija



Origen de replicación (oriC)

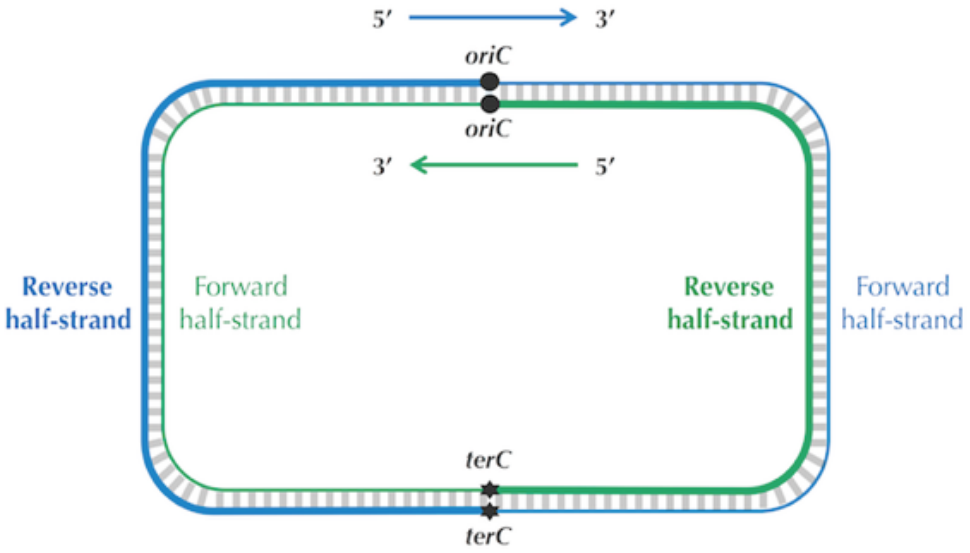
- Región donde comienza la replicación
  - Hasta terminar en otro punto (terC)

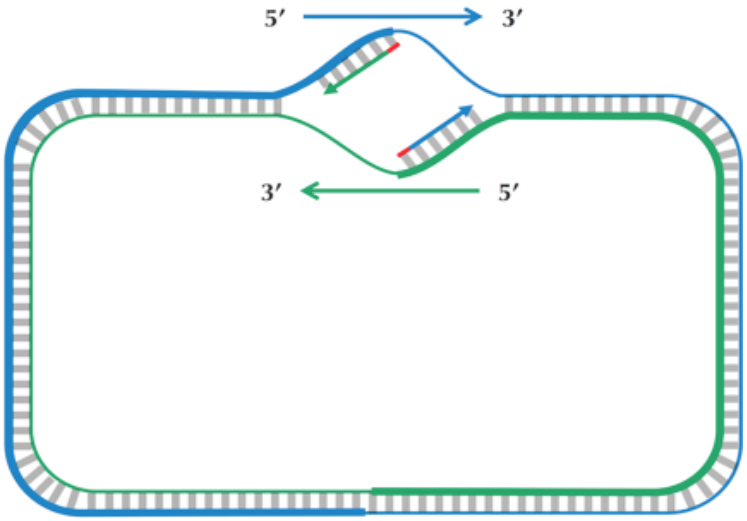


Cuatro DNA polimerasas (rojo) replicando el genoma desde oriC

Asimetría en la replicación

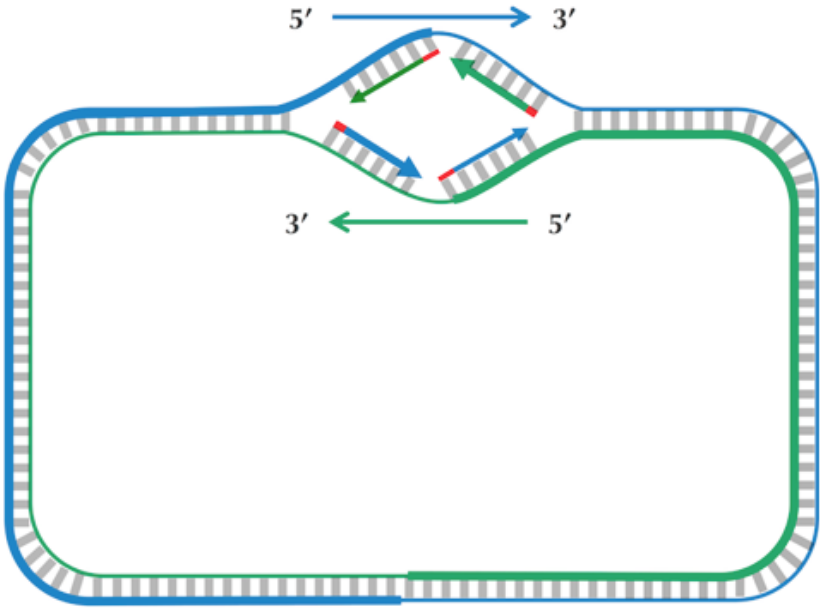
- *DNA polimerasa*: enzima que va 'añadiendo nucleótidos' complementarios a una cadena simple de ADN
  - Sólo lo hace en el sentido 3' → 5'

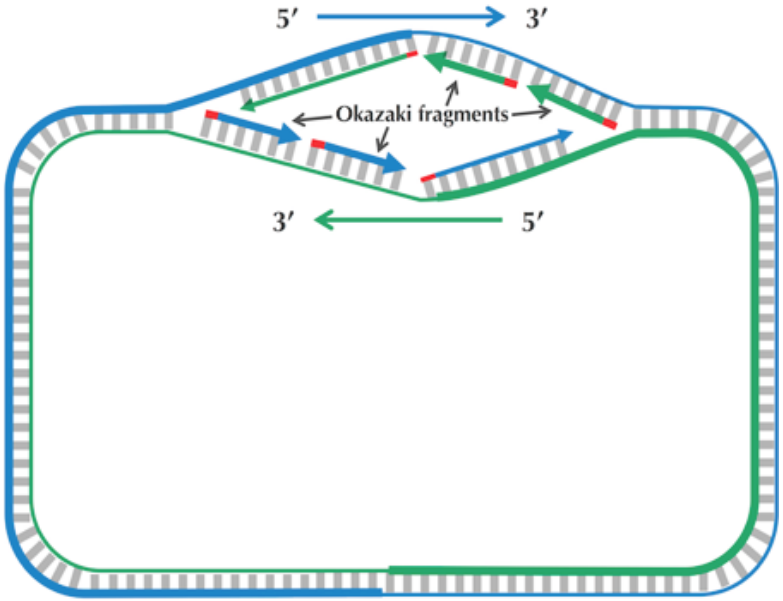




Asimetría en la replicación (II)

- El sentido 5'→3' se replica 'hacia atrás' cuando la cadena se ha abierto un poco
  - Trocito a trocito (fragmentos de Oyazaki)





Buscando OriC

- La cadena 'inversa' tiene que esperar un poco hasta que puede replicarse
- Es decir, pasa más tiempo como cadena 'simple' → es más fácil que sufra mutaciones

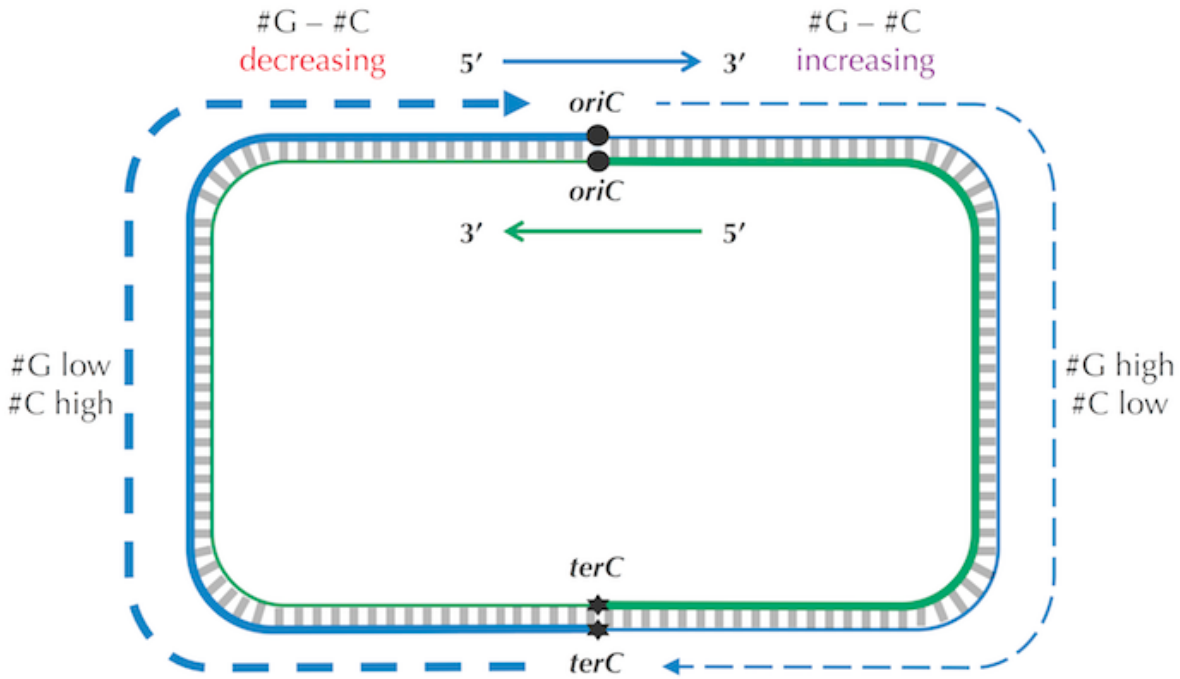
	C	G	A	T
Entire strand	427419	413241	491488	491363
Reverse half-strand	219518	201634	243963	246641
Forward half-strand	207901	211607	247525	244722
Difference	**+11617**	**-9973**	-3562	-1919

de nucleótidos en *Thermotoga petrophila*

¿sabrías calcular estos números?

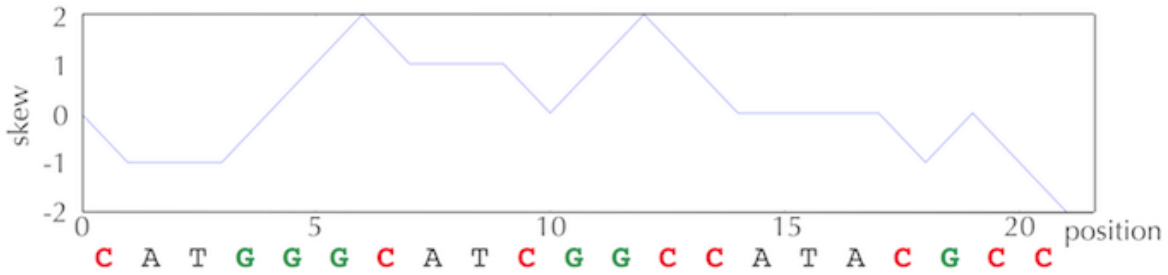
Buscando OriC (II)

- ¿Cómo podemos usar esta estadística para intentar localizar *OriC*?
  - Mediante la diferencia entre G y C a través del ADN



Ejercicio

- Implementar la función `skew`:
  - **entrada**: `seq` (cadena de ADN)
  - **salida**: lista numérica que comienza en 0 y va variando en +1 cuando encontremos una G y en -1 cuando encontremos una C



`CATGGGCATCGGCCATACGCC` → `0 -1 -1 -1 0 1 2 1 1 1 0 1 2 1 0 0 0 0 -1 0 -1 -2`

Ejercicio

- Implementar la función `minSkew`:
  - **entrada**: `seq` (cadena de ADN)
  - **salida**: posiciones de seq donde el skew tiene un valor mínimo

- Prueba:
  - entrada:
    - `TAAAGACTGCCGAGAGGCCAACACGAGTGCTAGAACGAGGGGCGTAAACGCGGGTCCGAT`
  - salida: `11,24`

Dibujando

```
import matplotlib.pyplot as plt

plt.plot(lista, "titulo")  #datos a dibujar y título
plt.xlabel("etiquetaX")    #etiquetas en los ejes X e Y
plt.ylabel("etiquetaY")
plt.show()                 #hasta este punto no se dibuja
```

Dibujando

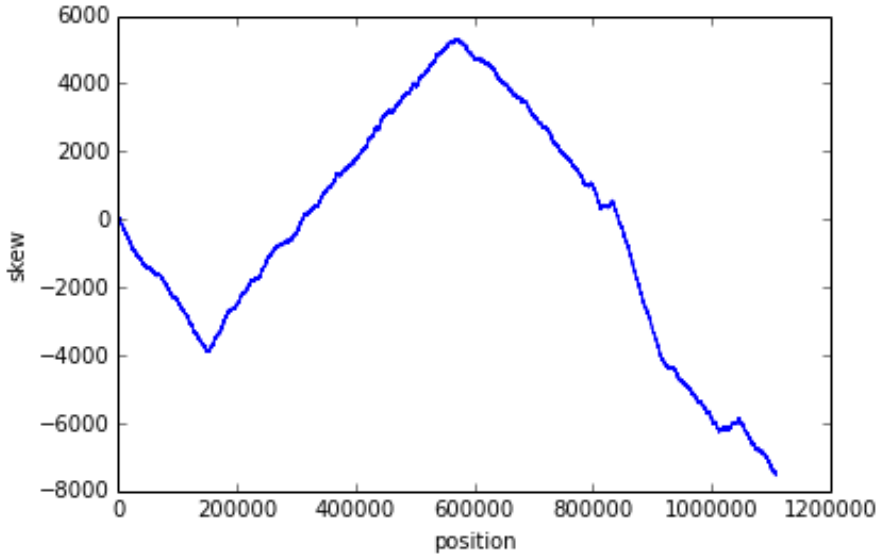
```
import matplotlib.pyplot as plt

plt.plot(lista, "titulo")  #datos a dibujar y título
plt.xlabel("etiquetaX")    #etiquetas en los ejes X e Y
plt.ylabel("etiquetaY")
plt.show()                 #hasta este punto no se dibuja
```

Ejercicio

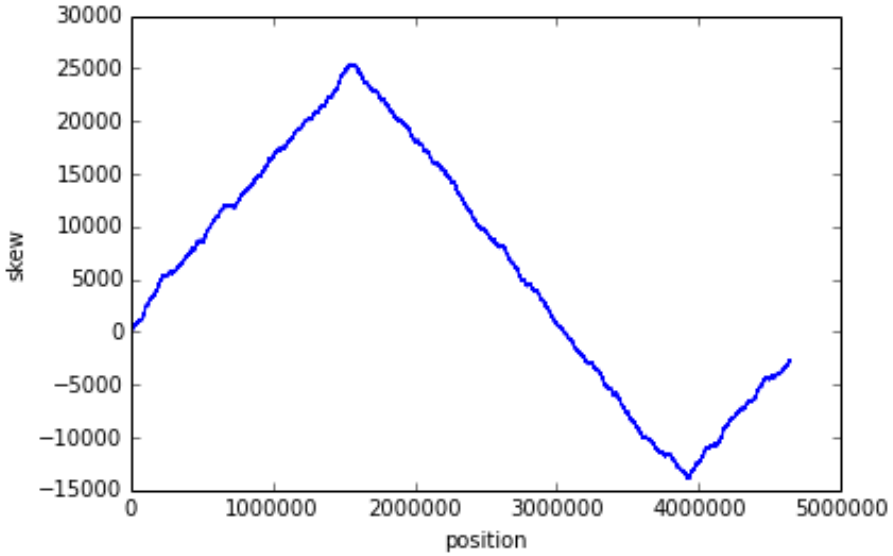
- Dibujar el skew y determinar el mínimo skew para *E coli*, *V cholerae* y *T petrophila*

Resultados



Vibrio cholerae

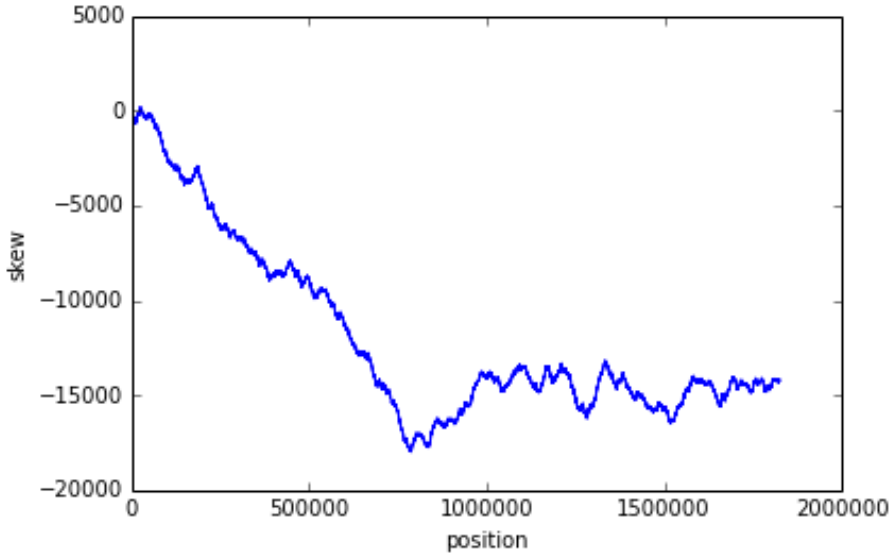
```
[1108189, 1108190, 1108192,
1108207, 1108208, 1108212, 1108213]
```



Escherichia coli

```
[3923620, 3923621, 3923622, 3923623]
```

Resultados



Thermotoga petrophila

[787199, 787200, 787201, 787202, 787203, 787204, 787205, 787206, 787207, 787209, 787210]

K-meros esquivos

- Gracias al skew podemos localizar donde está el origen de replicación de E. coli
  - entorno a la posición 3923620
- ¿Hay algún 9-mer que aparezca al menos 3 veces en las 500 bases siguientes a 3923620?

La proteína **DnaA** es el factor de inicio de la replicación, y se une a determinadas regiones en oriC, conocidas como **DnaA boxes**. Por ejemplo, en *E. coli* hay 4 DnaA boxes que contienen el 9-mer 5'-TTATCCACA-3'

Si es que ya no te puedes fiar de nadie…

CÓDIGO DEGENERADO

K-meros esquivos

- El código genético es *degenerado*
  - Existen mutaciones que pueden variar ligeramente un motivo

ATTCTGGA
ATTCTCGA
ATACTGGG

Ejercicio

- Implementar la función `findPattern`:
  - **entrada**:
    - `pattern` (motivo de ADN a buscar)
    - `seq` (cadena de ADN donde buscar)
  - **salida**: un diccionario `{pos,pattern}` donde se almacena el patrón encontrado y su posición de inicio

- Prueba:
  - seq: `CGCCCGAATCCAGAACGCATTCCCCTGGCCTCCATTCTGGAACGGTACGGACGTCAATCAAAT`
  - pattern: `ATTCTGGA`
  - salida: `{33: 'ATTCTGGA'}`

Ejercicio

- Modificar la función `findPattern`:
  - **entrada**:
    - `pattern` (motivo de ADN a buscar)
    - `seq` (cadena de ADN donde buscar)
    - **`mismatch`** (nº de mutaciones permitidas respecto a `pattern`)
  - **salida**: un diccionario `{pos,pattern}` donde se almacena el patrón encontrado y su posición de inicio

- Prueba:
  - seq: `CGCCCGAATCCAGAACGCATTCCCCTGGCCTCCATTCTGGAACGGTACGGACGTCAATCAAAT`
  - pattern: `ATTCTGGA`
  - d: `3`
  - salida: `{6: 'AATCCAGA', 7: 'ATCCAGAA', 33: 'ATTCTGGA'}`

Secuencia consenso

- Secuencia más frecuente de entre varias

```
ATTCTGGA
ATTCTCGA
ATACTGGG
--------
ATTCTGGA
33233232
```

Ejercicio

- Implementar la función `consensus`:
  - **entrada**:
    - `seqs` (lista de secuencias de ADN de la misma longitud)
  - **salida**: `str` con la cadena de consenso

- Prueba:
  - seqs: `['ATTCTGGA', 'AATCCAGA', 'ATCCAGAA']`
  - salida: `ATTCAGGA`

Ejercicio

- Implementar la función `findConsensus`:
  - **entrada**:
    - `pattern` (patrón a buscar)
    - `seq` (secuencia de ADN donde buscar)
    - `d` (mutaciones permitidas)
  - **salida**: `str` con la cadena de consenso para las coincidencias encontradas
- Es simplemente combinar las dos funciones anteriores
- Prueba:
  - seq: `CGCCCGAATCCAGAACGCATTCCCCTGGCCTCCATTCTGGAACGGTACGGACGTCAATCAAAT`
  - pattern: `ATTCTGGA`
  - d: `3`
  - salida: `ATTCAGGA`

Ejercicio

- Implementar la función `countKmersMM`:
  - **entrada**:
    - `seq` (cadena original)
    - `d` (nº de mutaciones permitidas)
    - `k` (tamaño del k-mero)
  - **salida**: diccionario que tenga como claves los `k`-meros y como valores el número de veces que aparece en `seq`, sin alterar o como mucho con `d` mutaciones.

Ejercicio

- Prueba:
  - seq: `ACGTTGCATGTCGCATGATGCATGAGAGCT`
  - d: `1`
  - k: `4`
  - salida:
```
{'TGAT': 2, 'ATGC': 5, 'GTCG': 2, 'AGAG': 2,
'ACGT': 2, 'GTTG': 3, 'CATG': 4, 'TGTC': 1, 'TCGC': 2,
'CGTT': 1, 'GAGC': 2, 'ATGA': 4, 'TGAG': 3, 'TGCA': 3,
'GAGA': 2, 'ATGT': 5, 'GCAT': 3, 'TTGC': 3, 'GATG': 5,
'AGCT': 1, 'CGCA': 3}
```
- Probar cómo cambia la salida si `d=0`

Ejercicio

- Modificar ligeramente la función anterior para hacer la función `mostFrequentKmers`:
  - **entrada**:
    - `seq` (cadena donde buscamos)
    - `d` (nº de mutaciones permitidas)
    - `k` (tamaño del k-mero)
  - **salida**: lista con los `k`-mers más frecuentes en `seq`, contando hasta `d` mutaciones

Reflexión

- ¿Encontrados todos los k-mers posibles?
  - ¡No! Sólo aquellos que ya se encuentren en la secuencia
  - Necesitamos pensar en términos de todos los posibles k-mers
    - Problemas de combinatoria (estadística)
    - Problemas de rendimiento (computación)