Informática biomédica

Búsqueda de patrones

Rodrigo Santamaría

Patrones

- Un poco de biología
  - K-mers
  - DNA box
  - Motivos
- Cazando motivos esquivos
  - Mutaciones
  - Consenso

Un poco de biología

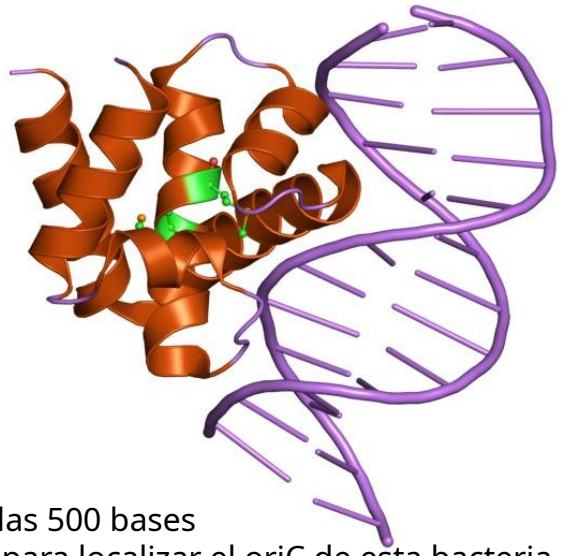
K-mero*

- Es una secuencia de k nucleótidos
  - Concepto muy utilizado en ensamblado, alineamiento y búsqueda de secuencias
- Una tarea común en bioinformática es buscar secuencias cortas (p. ej. k-meros) en secuencias largas (p. ej. genomas)
  - A veces de manera ‘inexacta’ o agregada, tan rápido como sea posible

* Del griego *méros* (parte), se podría traducir como ‘una parte o porción de longitud k’.
-*mer* es un sufijo muy utilizado en química para denotar partes de una molécula (p. ej. polímero)

DNA box

- La proteína **DnaA** es el factor de inicio de la replicación en bacterias
- **DnaA** 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’**



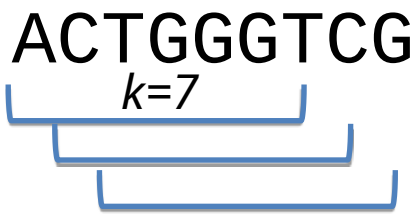
http://www.ebi.ac.uk/pdbe/entry/pdb/1j1v

**oriC* es una región abstracta, de una longitud arbitraria, generalmente entorno a las 500 bases
En el Ejercicio 4 de la Sesión 1 calculamos el sesgo mínimo de E coli, que usaremos para localizar el oriC de esta bacteria

** https://en.wikipedia.org/wiki/Prokaryotic_DNA_replication

Contando k-meros

- Es sencillo, simplemente implica recorrer una secuencia de longitud *n* y capturar cada subsecuencia de longitud *k*




```
ACTGGGT   +1
 CTGGGTC  +1
  TGGGTCG +1
    ...
```

¿Cuál es la complejidad O del método?

2 puntos

Ejercicio 1

- 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 1

- Ejemplo
  - **entrada**:
    - `seq=ACGTTGCATGTCGCATGATGCATGAGAGCT`
    - k=4
    - n=2
  - **salida**:
    - `{'GCAT': 3, 'ATGA': 2, 'TGCA': 2, 'CATG': 3}`
- Prueba
  - seq=OriC de *Vibrio cholerae*. Se puede encontrar aquí: http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/oric.fasta
  - k=9
  - n=3

Motivos

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

Usando `countKmers`, ¿qué 9-meros frecuentes podemos encontrar en la región *oriC* de *Vibrio cholerae*?

¡Eureka! Hay un patrón bastante claro: las secuencias `CTTGATCAT` y su 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: k-meros esquivos

- Si usamos `countKmers` en el oriC de *E coli*, no encontramos todas las ocurrencias del motivo esperado (TTATCCACA)*
- El código genético es **degenerado**
  - Existen mutaciones que pueden variar ligeramente el patrón de un motivo

TTATCCACA
TTATC**G**ACA
T**G**ATCCA**T**A

¿Hasta qué punto un patrón es igual pero degenerado, o directamente distinto?
¿Qué criterios estadísticos o biológicos marcamos para estas distinciones?
¿Cómo afecta el código degenerado a la búsqueda computacional (diseño de algoritmos, rendimiento, etc.)?
Todas estas son preguntas importantes que iremos discutiendo

*Ver diap. 4

Ejercicio 2

- Implementar la función `findMotif`:
  - **entrada**:
    - `motif` (motivo de ADN a buscar)
    - `seq` (cadena de ADN donde buscar)
    - `d` (nº de mutaciones permitidas respecto a `motif`, `menor que 4`)
  - **salida**: un diccionario `{pos,motif}` donde se almacena el patrón encontrado y su posición de inicio
- Ejemplo:
  - seq: `CGCCCGAATCCAGAACGCATTCCCCTGGCCTCCATTCTGGAACGGTACGGACGTCAATCAAAT`
  - motif: `ATTCTGGA`
  - d: `3`
  - salida: `{6: 'AATCCAGA', 7: 'ATCCAGAA', 33: 'ATTCTGGA'}`

1 punto

Ejercicio 2

- Prueba:
  - seq: `oriC` de *E coli*
    - Para calcularla, tomamos una sección del genoma de *E coli* de 500 bases, comenzando por la primera posición obtenida con la función `minSkew` (ejercicio 4 de la sesión 1)
  - motif: DNA box conocido por métodos experimentales
    - `TTATCCACA`
  - d: `1`

Secuencia consenso

- Secuencia más frecuente de entre varias
- Forma ‘comprimida’ de representar motivos

```
ATTCTGGA
AATCTCGA
ACACTGGG
--------
A*TCTGGA
31233232
```

En caso de que no haya ningún valor que destaque para una determinada posición, se suele utilizar alguna técnica de desempate o, más coherente, usar un carácter que indique que para esa posición no hay un consenso claro (p. ej. un asterisco)

Consenso y logos

- Se pueden usar mayúsculas y minúsculas para representar el grado de consenso, o técnicas más complejas (logos de secuencia)

```
ATTCTGGA
AATCTCGA
ACACTGGG
--------
A*tCTgGa
31233232
```

1 punto

Ejercicio 3

- Implementar la función `consensus`:
  - **entrada**:
    - `seqs` (lista de secuencias de ADN de la misma longitud)
  - **salida**: `str` con la cadena de consenso
    - En caso de empate, se desempata alfabéticamente
- Ejemplo:
  - seqs: `['ATTCTGGA', 'AATCCAGA', 'ATCCAGAA']`
  - salida: `ATTCAGGA`
- Prueba:
  - ADN mitocondrial de distintos primates*

* http://vis.usal.es/rodrigo/documentos/bioinfo/filogenia/mitDNAprimates.fasta

Patrones

- Un poco de biología
  - DNA box
  - K-mers
  - Motivos
- Cazando motivos esquivos

Cazando motivos esquivos

Motivos esquivos

- Para encontrar el motivo más frecuente debemos tener en cuenta cuenta motivos que no aparezcan en nuestra secuencia
  - ¡Puede haber motivos que ***no*** aparezcan en la secuencia pero por mutaciones sean los más comunes!
  - **Conclusión**: debemos tener en cuenta todos los motivos posibles para un tamaño de k-mer

¿Cuál es el 3-mer más frecuente en la secuencia `TTATTA` permitiendo hasta 1 mutación?

Ejercicio

- Usar la función `mutations*`:
  - **entrada**
    - `word`: palabra de la que buscamos todas la mutaciones
    - `letters`: posibles letras en `word` o en las mutaciones
    - `num_mismatches`: número de mutaciones puntuales
  - **salida**
    - vector con todas las posibles mutaciones de `word`
- Usar la función `mutationsEqualOrLess` que calcula todas las mutaciones con *hasta* `num_mismatches`

* http://vis.usal.es/rodrigo/documentos/bioinfo/muii/mutations.py

Solución

```
def mutations(word, letters, num_mismatches):
    import itertools
    for locs in itertools.combinations(range(len(word)), num_mismatches):
        this_word = [[char] for char in word]
        for loc in locs:
            orig_char = word[loc]
            this_word[loc] = [l for l in letters if l != orig_char]
        for poss in itertools.product(*this_word):
            yield ''.join(poss)
```

```
def mutationsEqualOrLess(word, num_mismatches, letters="ACGT"):
    matches=set()
    for dd in range(num_mismatches,-1,-1):
        matches.update(list(mutations(word, letters, dd)))
    return matches
```

http://vis.usal.es/rodrigo/documentos/bioinfo/muii/mutations.py

Paquetes

1. Añadir la ruta de nuestro paquete al path:

```
import sys
sys.path.append("/Users/rodri/Documents/docencia/sesionesPractica")
```

2. Importar archivo `sesion2methods.py` y utilizar cualquier función definida en él

```
import sesion2methods as s2
seq="aagcttcaccggcaagaagaaggtggtgatcgcccacggactcacatcctcattactattctgcctagcaa"
s2.frequentWordMMRC(seq, k=15,d=4)
```

2 puntos

Ejercicio 4

- Implementar la función `allMutations`:
  - **entrada**
    - seq: secuencia de la que buscamos todos los k-mers
    - k: tamaño de los k-mers
    - d: número de mutaciones puntuales
  - **salida**
    - **longitud del** vector con todos los k-mer posibles en seq, incluidos aquellos que no se encuentran explícitamente en seq, considerando hasta d mutaciones puntuales

Ejercicio 4

- Pista:
  - Utilizar la función `mutationsEqualOrLess` y una variable conjunto de tipo `set`
- Ejemplo:
  - seq: `CGCCCGAATCCAGAACGCATTCCCATATTTCGGGACCACTGG CCTCCACGGTACGGACGTCAATCAAAT`
  - k: 9
  - d: 2
  - salida: `20847`
- Prueba: oriC de *E coli**, con k=9 y d=3

*Obtenido como se indica en el ejercicio 2 de esta sesión, a partir del ejercicio 4 de la sesión 1

9 puntos

Ejercicio 5

- Implementar la función `countKmersV2`:
  - **entrada**:
    - seq (cadena donde buscamos)
    - d (nº de mutaciones permitidas)
    - k (tamaño del k-mero)
    - n (nº mínimo de veces que se encuentra el k-mero)
  - **salida**: diccionario con los k-meros (clave) y la frecuencia con la que ocurren (valor)
    - contando hasta d mutaciones
    - excluyendo los k-meros con menos de n ocurrencias
    - teniendo en cuenta k-meros que no están en la secuencia
    - teniendo en cuenta las ocurrencias del k-mero reverso complementario

Ejercicio 5

- Ejemplo:
  - seq: AACAAGCTGATAAACATTTAAAGAG
  - k: 5
  - d: 1
  - n: 4
  - salida: {'AAAAA': 4, 'CTTTT': 4, 'AACAG': 4,
    'TTTAA': 5, 'TAAAA': 5, 'TTAAT': 4, 'TTGAA': 4,
    'GTTAA': 4, 'ATTAA': 4, 'TTCAA': 4, 'TTAAC': 4,
    'TTATA': 4, 'TTAAA': 5, 'TTTTA': 5, 'TATAA': 4,
    'CTGTT': 4, 'AAAAG': 4, 'TTTTT': 4}
- Prueba:
  - OriC* de *E coli* con k=9 y d=1 (n=4)

*Obtenido como se indica en el ejercicio 2 de esta sesión, a partir del ejercicio 4 de la sesión 1

Buscando DNA boxes

- Ahora sí que encontramos las cuatro ocurrencias de TTATCCACA:

AATGATGATGACGTCAAAAGGATCCGGATAAAACATGGTGATTGCCTCGCATAACG
CGGTATGAAAATGGATTGAAGCCCGGGCCGTGGATTCTACTCAACTTTGTCGGCTT
GAGAAAGACCTGGGATCCTGGGTATTAAAAAGAAGATCTATTTATTTAGAGATCTG
TTCTATTGTGATCTCTTATTAGGATCGCACTGCCC**TGTGGATAA**CAAGGATCCGGC
TTTTAAGATCAACAACCTGGAAAGGATCATTAACTGTGAATGATCGGTGATCCTGG
ACCGTATAAGCTGGGATCAGAATGAGGGG**TTATACACA**ACTCAAAAACTGAACAAC
AGTTGTTC**TTTGGATAA**CTACCGGTTGATCCAAGCTTCCTGACAGAG**TTATCCACA**
GTAGATCGCACGATCTGTATACTTATTTGAGTAAATTAACCCACGATCCCAGCCAT
TCTTCTGCCGGATCTTCCGGAATGTCGTGATCAAGAATGTTGATCTTCAGTG

mutación sentido antisentido

Buscando DNA boxes

- Hemos tenido mucha suerte de que el motivo estuviera justo en los 500 nucleótidos que hemos elegido para definir *OriC*
- Hay muchas preguntas en el aire:
  - Hay otros motivos con 4 ocurrencias en *oriC*. ¿Son equivalentes al encontrado? ¿Para qué sirven?
  - ¿Por qué 9-mers y no 8-mers o 10-mers?
  - ¿Por qué una sola mutación, y no ninguna, o dos?
  - ¿Cuánto tarda tu algoritmo para una secuencia de 500 bases? ¿sería factible para buscar en todo un genoma?

Informática biomédica

Motivos

Rodrigo Santamaría

Motivos

- Motivos
- Puntuaciones
- Búsqueda avariciosa
- Búsqueda aleatoria

Motivos

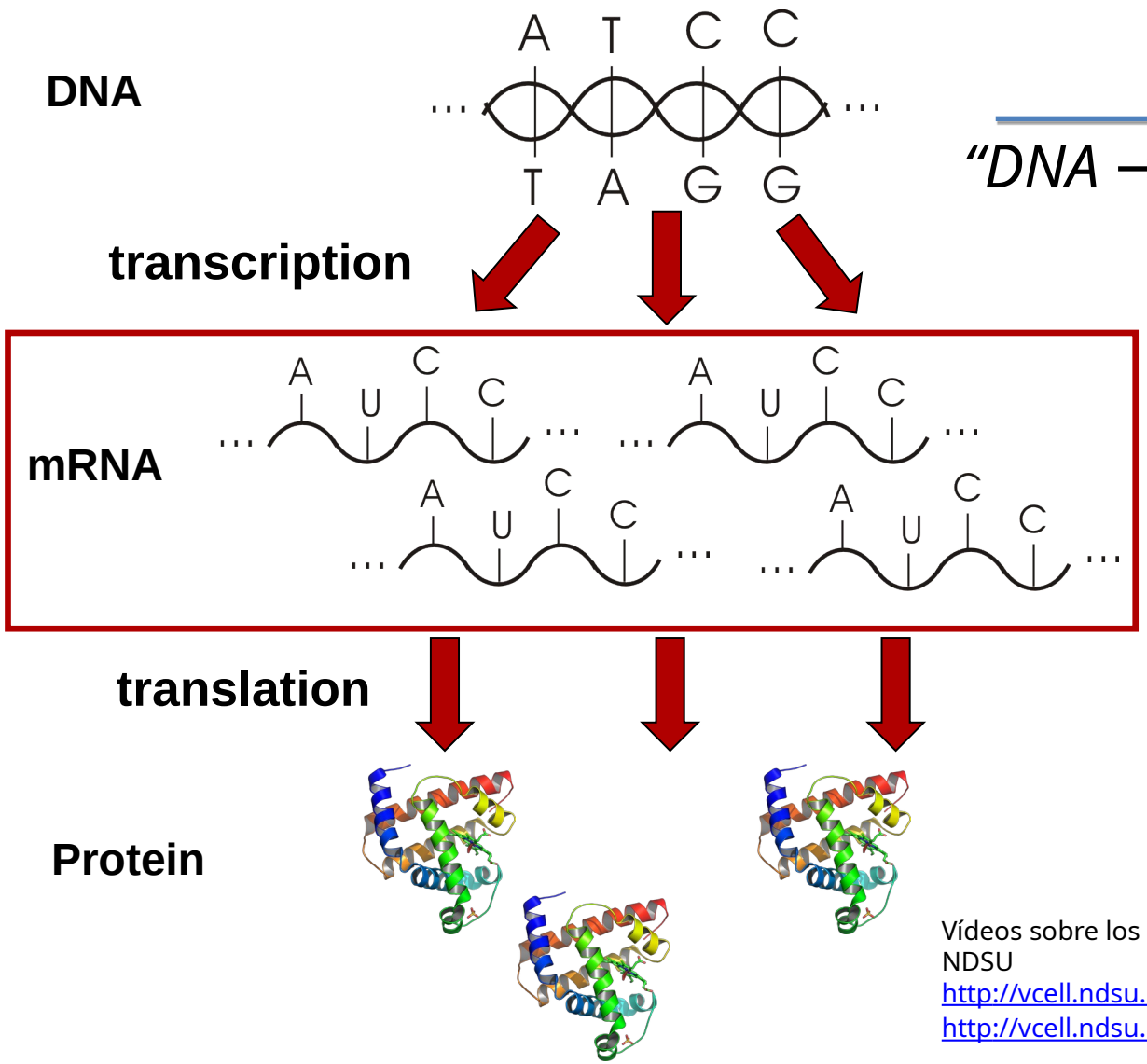
Expresión génica y reloj circadiano

Regulación genómica y motivos

Reloj circadiano

- ‘Reloj’ biológico que controla nuestra agenda
  - Razón por la que sufrimos jet-lag
  - Experimentos con sujetos en total oscuridad durante 24h muestran que los ciclos se mantienen
- Muy importante en plantas
  - Para coordinarse con la salida/puesta del sol
  - Dependiendo del momento del día determinado por su reloj, sus genes se **expresarán** diferente

Dogma central de la biología molecular



expresión

"DNA → mRNA → proteínas"

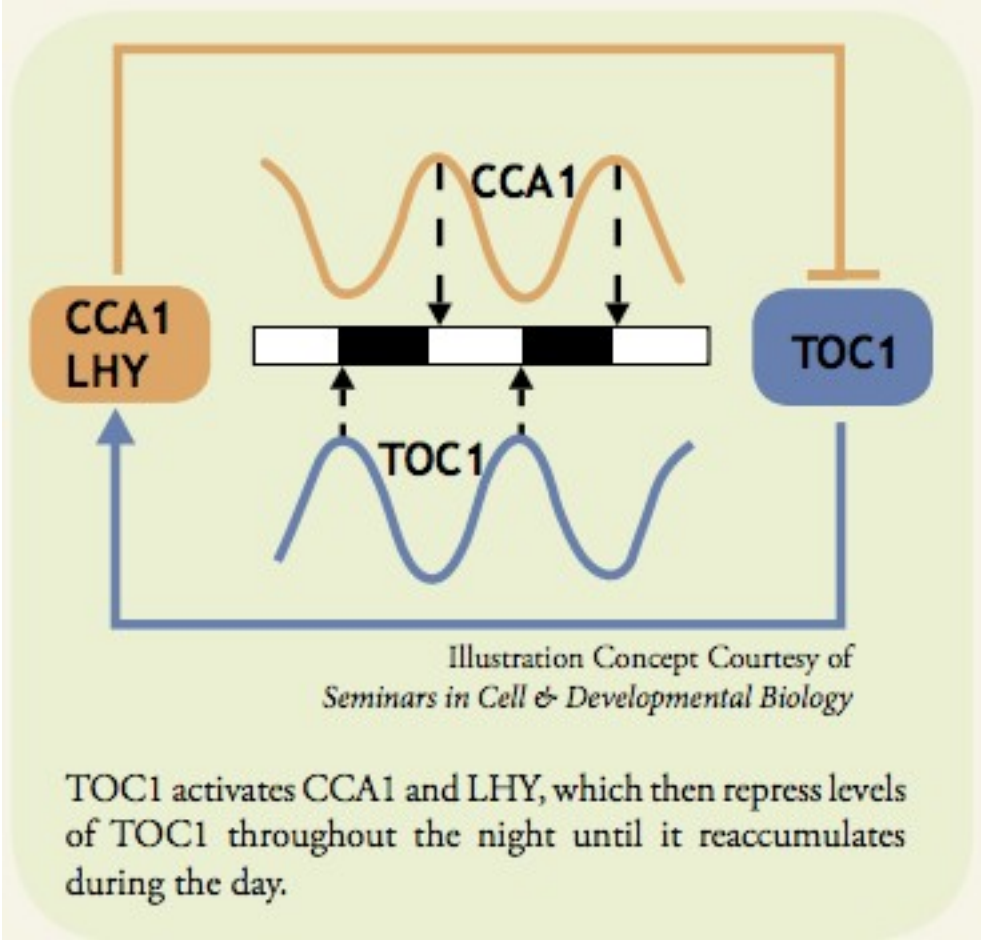
Vídeos sobre los procesos de transcripción y traducción de la NDSU

http://vcell.ndsu.nodak.edu/animations/transcription

http://vcell.ndsu.nodak.edu/animations/translation

Genes reguladores

- En plantas, **LHY, CCA1 y TOC1** son los genes que regulan el reloj circadiano
  - TOC1 activa CCA1/LHY, que luego reprimen los niveles de TOC1
  - Al reprimirse los niveles de TOC1, bajan también los de CCA1/LHY, hasta que TOC1 se vuelve a recuperar por la noche y los reactiva para un nuevo día
- Esto se conoce como un **bucle de realimentación negativo** y es muy común



https://sqonline.ucsd.edu/2013/06/growth-spurts-circadian-secrets-and-an-algal-cure-for-malaria/

Factores de transcripción

- LHC/CCA1/TOC1 son genes reguladores porque las proteínas que expresan son **factores de transcripción**
  - Se unen a **motivos reguladores** (o lugares de unión de factores de transcripción)
    - Secuencias cortas de DNA cerca del inicio de transcripción de un gen
    - De esta manera, promueven la activación del gen

Motivos

- [Steve Kay, 2000] encontró que 46 de 500 genes con un comportamiento circadiano de una planta, *Arabidopsis thaliana*, compartían el motivo AAAATATCT

- **Ejercicio**: *¿Cuál es el número esperado de ocurrencias de un 9-mer en 500 secuencias, cada una de longitud 1000?*

En cada secuencia, puede aparecer en 1000-9+1=992 posiciones
En cada una de esas posiciones, la posibilidad es de 1/(4e9)=0.0000038147
Luego 992*0.0000038147*500=1.89

Motivos esquivos

- Desgraciadamente, lo normal es que los motivos contengan mutaciones

1	T	C	G	G	G	G	g	T	T	T	t	t
2	c	C	G	G	t	G	A	c	T	T	a	C
3	a	C	G	G	G	G	A	T	T	T	t	C
4	T	t	G	G	G	G	A	c	T	T	t	t
5	a	a	G	G	G	G	A	c	T	T	C	C
6	T	t	G	G	G	G	A	c	T	T	C	C
7	T	C	G	G	G	G	A	T	T	c	a	t
8	T	C	G	G	G	G	A	T	T	c	C	t
9	T	a	G	G	G	G	A	a	c	T	a	C
10	T	C	G	G	G	t	A	T	a	a	C	C

Por ejemplo, estas son todas la variaciones del motivo **TCGGGGATTTCC**, regulador de la respuesta inmune en *Drosophila melanogaster* (mosca).
En este motivo se une el factor de transcripción NF-xB*

**NF-xB es un actor clave en muchos procesos celulares, especialmente en la reacción inmunológica frente a una infección. Su regulación incorrecta está ligada al desarrollo de cáncer, choques sépticos y enfermedades autoinmunes.*

Motivos esquivos

- Implantamos un 15-mer en estas 10 secuencias :

aagcttcaccggcAAAAAAAAGGGGGGGatcgcccacggactcacatcctcattactattctgcctagcaaactcaaa
ctacgaacgcactcacagtcgcatcataatcctctctcaaggacttcaaactctactccAAAAAAAAGGGGGGGgatg
acttctagcaagcctcgctAAAAAAAAGGGGGGGcccactattaacctactgggagaactctctgtgctagtaaccac
gttctcctgatcaaatAAAAAAAAGGGGGGGacaggactcaacatactagtcacagccctatactccctctacatatt
taccacaacacaatggggctcactcacccaccacattaacaacataaaacccAAAAAAAAGGGGGGGaacaccctcat
gttcatacacctatcccccattctcctAAAAAAAAGGGGGGGcgacatcattaccgggttttcctcttgtaaatatag
tttaaccaaaacatcagattgtgaatctgaAAAAAAAAGGGGGGGacccctttatttaccgagaaagctcacaagaact
gctaactcaAAAAAAAAGGGGGGGcaacatggctttctcaacttttaaaggataacagctatccattggtcttaggcc
ccaaaaattttggtgcaactccaaataaaagtaataaccatgcacactactataacAAAAAAAAGGGGGGGttcccta
attcccccatccttaccaccctcgttaaccctaacaaaaaaaactcataccccattatgtaAAAAAAAAGGGGGGGa

- Si lo convertimos en un motivo degenerado con d=4:

aagcttcaccggcAAgAAgAAGGtGGtGatcgcccacggactcacatcctcattactattctgcctagcaaactcaaa
ctacgaacgcactcacagtcgcatcataatcctctctcaaggacttcaaactctactccAAAgAAtaGGGGcGGgatg
acttctagcaagcctcgctAccAAAAAGGGtGGacccactattaacctactgggagaactctctgtgctagtaaccac
gttctcctgatcaaatAAAgAAgAaGGGtGGacaggactcaacatactagtcacagccctatactccctctacatatt
taccacaacacaatggggctcactcacccaccacattaacaacataaaacccAAAAcgAAtGGGGGaaacaccctcat
gttcatacacctatcccccattctcctAAAtAAtAGGGGcGacgacatcattaccgggttttcctcttgtaaatatag
tttaaccaaaacatcagattgtgaatctgacAAAAAgtGcGGGGGacccctttatttaccgagaaagctcacaagaact
gctaactcaAAAAAAAtaGGGaaGcaacatggctttctcaacttttaaaggataacagctatccattggtcttaggcc
ccaaaaattttggtgcaactccaaataaaagtaataaccatgcacactactataacAAAAAAtAcGGtaGGttcccta
attcccccatccttaccaccctcgttaaccctaacaaaaaaaactcataccccattatgtaAtAcAgAAaGGGGGGa

Puntuando motivos

Motivos

```
T C G G G G g T T T t t
c C G G t G A c T T a C
a C G G G G A T T T t C
T t G G G G A c T T t t
a a G G G G A c T T C C
T t G G G G A c T T C C
T C G G G G A T T c a t
T C G G G G A T T c C t
T a G G G G A a c T a C
T C G G G t A T a a C C
```

score(Motivos) **3+4+0+0+1+1+1+5+2+3+6+4 = 30**

profile(Motivos)

```
A: .2 .2  0  0  0  0 .9 .1 .1 .1 .3  0
C: .1 .6  0  0  0  0  0 .4 .1 .2 .4 .6
G:  0  0  1  1 .9 .9 .1  0  0  0  0  0
T: .7 .2  0  0 .1 .1  0 .5 .8 .7 .3. 4
```

consensus(Motivos) **T C G G G G A T T T C C**

1 punto

Ejercicio 6

- Implementar la función `score`:
  - **entrada**:
    - `Dna` (conjunto de t cadenas de longitud n)
  - **salida**: suma de todos los nucleótidos que no coinciden con el nucleótido mayoritario de su columna
- Ejemplo:
  - `Dna`: utilizar los doce motivos de NF-xB descritos anteriormente
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/nfkbMotifs.txt
  - salida: `30`
- Prueba
  - `Dna`:
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/dataset_40_9.txt

Regla de sucesión de Laplace

- Los ceros en estadística pueden ser un problema
  - P. ej. cualquier secuencia que comience por G tendría una Pr=0 en el caso anterior
- Solución: Regla de sucesión de Laplace*
  - Sumar 1 a cada evento antes de calcular la probabilidad

```
T A A C
G T C T
A C T A
A G G T
```

Cuenta

```
A: 2 1 1 1
C: 0 1 1 1
G: 1 1 1 0
T: 1 1 1 2
```

```
A: 2+1 1+1 1+1 1+1
C: 0+1 1+1 1+1 1+1
G: 1+1 1+1 1+1 0+1
T: 1+1 1+1 1+1 2+1
```

Probabilidad

```
A: 2/4 1/4 1/4 1/4
C: 0   1/4 1/4 1/4
G: 1/4 1/4 1/4 0
T: 1/4 1/4 1/4 2/4
```

```
A: 3/8 2/8 2/8 2/8
C: 1/8 2/8 2/8 2/8
G: 2/8 2/8 2/8 1/8
T: 2/8 2/8 2/8 3/8
```

*La regla de Cromwell dice *"no debemos usar 0 o 1 como probabilidades salvo que hablemos de predicados lógicos verdaderos o falsos"*. De hecho, el matemático francés Laplace lo llevó a rajatabla, calculando la probabilidad de que el sol no saliera mañana como 1/1826251, basándose en que había salido durante los últimos 5000 años. Aunque esta afirmación pueda resultar extrema, la aproximación de Laplace tiene un papel muy importante en la estadística

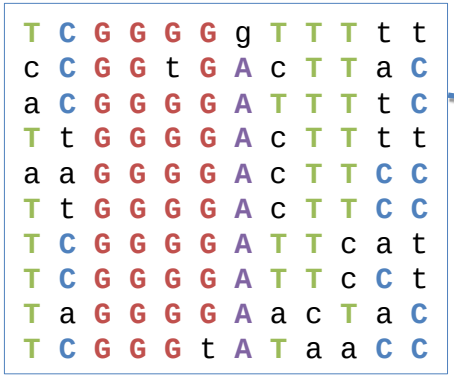
1 punto

Ejercicio 7

- Implementar la función `profile`:
  - **entrada**:
    - `Dna` (conjunto de t cadenas de longitud n)
    - Laplace (booleano para aplicar la regla de sucesión o no)
  - **salida**: perfil de las cadenas. Es decir, un diccionario con la frecuencia de cada nucleótido para cada posición, modificado según la regla de sucesión de si Laplace=True (por defecto).
- Ejemplo:
  - Dna: `["TAAC", "GTCT", "ACTA", "AGGT"]`
  - Laplace=`True`
  - `{'A': [0.375, 0.25, 0.25, 0.25], 'C': [0.125, 0.25, 0.25, 0.25], 'G': [0.25, 0.25, 0.25, 0.125], 'T': [0.25, 0.25, 0.25, 0.375]}`
- Prueba:
  - `Dna`: utilizar los doce motivos de NF-xB descritos anteriormente
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/nfkbMotifs.txt
  - Laplace=True

Probabilidad de un motivo

- Podemos calcular la probabilidad *Pr* de que un determinado perfil 'produzca' un motivo:




```
A: .2 .2  0  0  0  0 .9 .1 .1 .1 .3  0
C: .1 .6  0  0  0  0  0 .4 .1 .2 .4 .6
G:  0  0  1  1 .9 .9 .1  0  0  0  0  0
T: .7 .2  0  0 .1 .1  0 .5 .8 .7 .3. 4
```

Pr(ACGGGGATTACC|*Perfil*) = .2*.6* 1* 1*.9*.9*.9*.5*.8*.1*.4*.6

= 0.000939808

1 punto

Ejercicio 8

- Implementar la función `Pr`:
  - **entrada**:
    - `perfil` (matriz con probabilidades para cada nucleótido y posición)
    - `motivo` (secuencia del mismo tamaño que las columnas del perfil)
  - **salida**: número real con probabilidad de que el motivo aparezca en ese perfil
- Ejemplo:
  - perfil: utilizar los doce motivos de NF-xB descritos anteriormente para generar el perfil, sin aplicar la sucesión de Laplace
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/nfkbMotifs.txt
  - motivo=ACGGGGATTACC
  - salida: `0.000839808`
- Prueba:
  - `perfil`: el mismo que en el ejemplo
  - `motivo`: TCGGGGATTTCC

Ejercicio 9

2 puntos

- Implementar la función `mostProbableKmer`:
  - **entrada**:
    - `perfil` (matriz con probabilidades para cada nucleótido y posición)
    - `texto` (secuencia en la que buscar el k-mer más probable según el perfil)
    - k (tamaño del k-mer y del perfil)
  - **salida**: k-mer más probable según el perfil, entre los k-mer posibles de texto
- Ejemplo:
  - perfil de ["TTATATCGG", "GTCTACACA", "ACTAGGAGC", "AGGTTTATA"]
  - texto: TTATGTTTGGAAACTCTATCACCGACTGCTAGCA
  - k=9
  - Salida: ATGTTTGGA (tiene probabilidad de $2.896*10^{-5}$)
- Prueba:
  - perfil: utilizar los doce motivos de NF-xB descritos anteriormente para generar el perfil
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/nfkbMotifs.txt
  - texto=`GGTACGGGGATTACCT`
  - k: `12`

Motivos

- Motivos
- Búsqueda de motivos
- Búsqueda avariciosa
- Búsqueda aleatoria

Búsqueda de Motivos

Definición del problema

Fuerza bruta

Definición del problema (score)

- **Entrada**:
  - `Dna`: colección de `t` cadenas
  - `k`: tamaño del motivo
- **Salida**:
  - Una colección `motivos` de k-mers, uno de cada cadena en `Dna`, *que minimicen* `score(motivos)` entre todas las posibles combinaciones

Búsqueda de motivos (score)

Fuerza bruta?

- Probar todas las combinaciones de motivos buscando el menor `score`
  - `n-k+1` posibles motivos en cada secuencia
  - $(n-k+1)^t$ combinaciones en `t` secuencias
  - `score(motivos)` está en el orden $k \cdot t$
  - Asumiendo n>>k, estaríamos hablando de
    - $O(n^t \cdot k \cdot t)$
    - Muy lento!

Motivos

- Motivos
- Búsqueda de motivos
- Búsqueda avariciosa
- Búsqueda aleatoria

Búsqueda avariciosa

Matrices de probabilidad

Sustitución de Laplace

Búsqueda avariciosa de motivos

- Los **algoritmos avariciosos** son procedimientos iterativos que escogen la mejor alternativa entre un conjunto de soluciones *aleatorias*
  - Normalmente, no encuentran la mejor solución
    - Pero son muy rápidos en encontrar una solución *aproximada*
- Es una solución muy usada en bioinformática

Búsqueda avariciosa de motivos

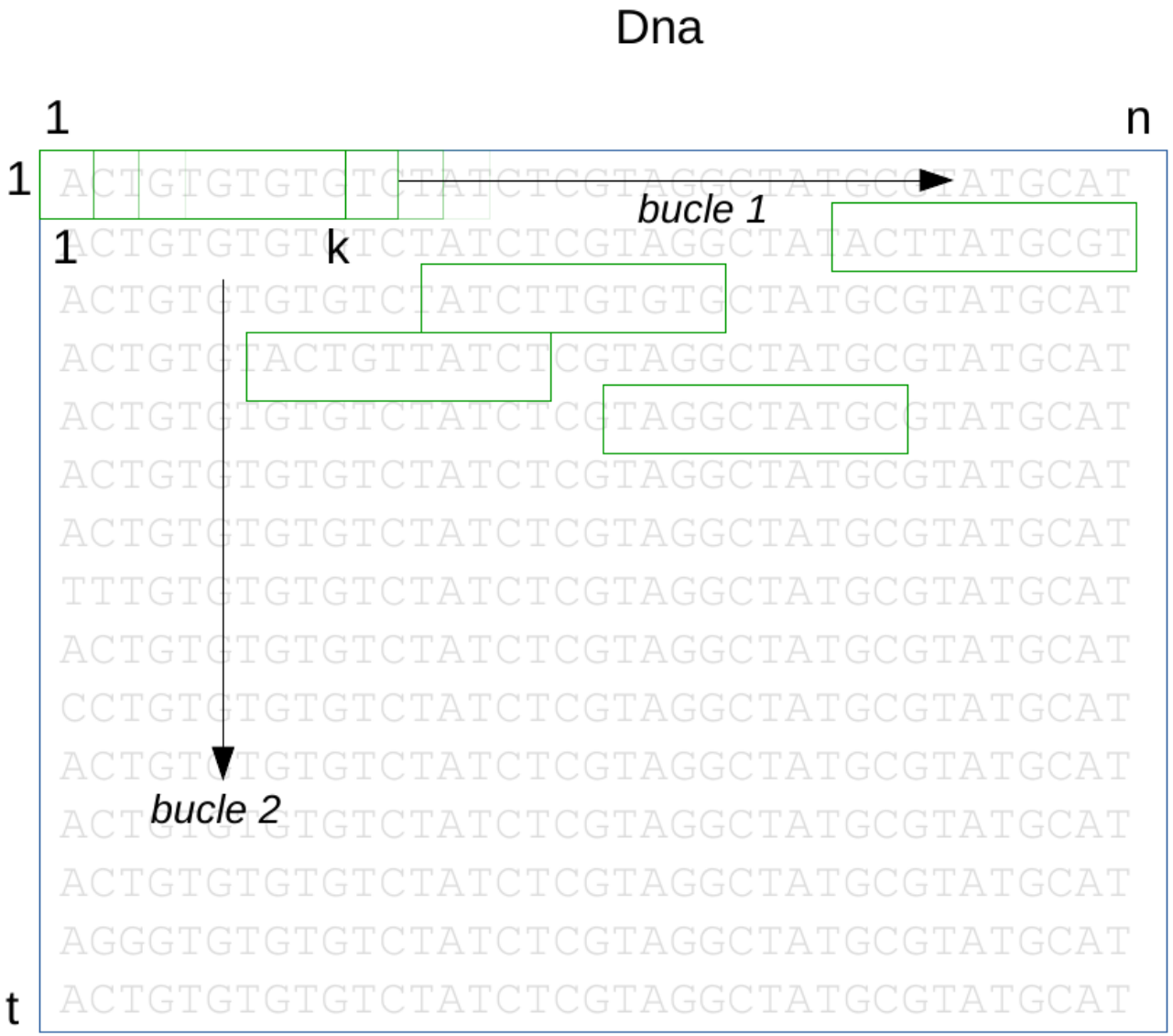
```
GreedyMotifSearch(Dna, k, t)
    bestMotifs <- matriz con los primeros k-mers de Dna
    para cada k-mer motif en la primera cadena de Dna
        motif_1 <- motif
        para i=2 hasta t
            p <- profile(motif_1,...,motif_i-1)
            motif_i <- mostProbableKmerGivenProfile(p,Dna[i],k)
        motifs <-(motif_1,...,motif_t)
        si score(motifs) < score(bestMotifs)
            bestMotifs <- motifs
  return bestMotifs
```

Búsqueda avariciosa

- El pseudocódigo anterior básicamente
  - Elige como k-mers más homogéneos en Dna unos al azar
    - p. ej. el primero de cada secuencia en Dna
  - Toma la primera secuencia de Dna
    - Para cada k-mer **M** en dicha secuencia
      - Para cada secuencia **s** de Dna
        - » Construye un perfil P_{ant} con las secuencias *anteriores*
        - » Determina qué k-mer $\mathbf{M_s}$ es más probable en **s** a partir de P_{ant}
        - » Añade $\mathbf{M_s}$ al conjunto de k-mers parecidos a **M**
      - Si el conjunto de k-mers parecidos a **M** es más homogéneo que el conjunto de k-mers que teníamos hasta ahora, lo sustituye

Una buena explicación paso a paso del algoritmo, con ejemplos, puede encontrarse aquí: http://www.mrgraeme.co.uk/greedy-motif-search/

Búsqueda avariciosa de motivos



"para cada motivo en la cadena 1"…

"...haz crecer 'motifs' añadiendo el motivo más probable de los restantes"

5 puntos

Ejercicio 10

- Implementar la función `greedyMotifSearch`:
  - **entrada**:
    - dna (t secuencias en las que buscar un motivo)
    - k (tamaño del motivo a buscar)
  - **salida**: t k-mers, uno de cada secuencia, que corresponden al motivo más probable, usando la regla de sucesión de Laplace para el perfil
- Ejemplo:
  - dna: `['GGCGTTCAGGCA', 'AAGAATCAGTCA', 'CAAGGAGTTCGC', 'CACGTCAATCAC', 'CAATAATATTCG']`
  - k: 3
  - salida: `['TTC', 'ATC', 'TTC', 'ATC', 'TTC']`
- Prueba:
  - dna: http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/ seqs4mut.txt
  - k=15

Analizando la busqueda avariciosa

- Obtenemos un resultado muy rápido
- No es perfecto, pero es bastante preciso (si usamos la sucesión de Laplace):

```
AAAAAAAAGGGGGGG
tAAAAAAAaGGatGG  (consenso con Laplace)
ggggcAAAtataaGa  (consenso sin Laplace)
```

- No obstante, ¿podríamos tener un resultado aún más preciso?

Motivos

- Motivos
- Búsqueda de motivos
- Búsqueda avariciosa
- Búsqueda aleatoria

Búsqueda aleatoria

Algoritmos de Monte Carlo

Muestreo de Gibbs

Algoritmos aleatorios

- Dos tipos
  - Algoritmos de Las Vegas
    - Solución siempre correcta
    - Aleatoriedad en el uso de recursos para conseguirla
  - Algoritmos de Monte Carlo*
    - Tiempo de ejecución determinista
    - **Solución incorrecta con un error muy bajo**
    - La mayoría de algoritmos aleatorios son de este tipo
- En ambos casos, son algoritmos **muy rápidos**

*El nombre de los dos tipos de algoritmos aleatorios viene dado por ciudades con una larga tradición de casinos

Ejercicio

- Implementar la función `motifs`:
  - **entrada**:
    - `profile` (un perfil 4 x k)
    - dna (conjunto de t cadenas)
  - **salida**: t k-mers, uno de cada secuencia, que corresponden al motivo más probable, usando el perfil dado
- Ejemplo:
  - dna: `['TTACCTTAAC', 'GATGTCTGTC', 'ACGGCGTTAG', 'CCCTAACGAG', 'CGTCAGAGGT']`
  - profile: `{'A':[0.8, 0 ,0, 0.2], 'C':[0, 0.8, 0.2, 0], 'G':[0.2, 0.2, 0.6, 0], 'T':[0, 0.2, 0, 0.8]}`
  - salida: `['ACCT', 'ATGT', 'GCGT', 'ACGA', 'AGGT']`

Búsqueda aleatoria de motivos

- Ejecución iterativa de `motifs`
  - La primera ejecución con un conjunto aleatorio de motivos
  - Ejecuciones progresivas de `motifs` sobre los resultados de la ejecución anterior, mientras mejoremos el `score`

Biblioteca `random` en python

```
import random
random.seed()
ri=random.randint(minValue,maxValue)
```

Búsqueda avariciosa de motivos

```
RandomizedMotifSearch(Dna, k)
   motivos <- selección aleatoria de k-mers en
   cada cadena de Dna
   mejores <- motivos
   para siempre
      perfil <- profile(motivos)
      motivos <- motifs(perfil, Dna)
      si score(motivos) < score(mejores)
       mejores <- motivos
      si no
        return mejores
```

Ejercicio

- Implementar la función `randomizedMotifSearch`:
  - **entrada**:
    - `profile` (un perfil 4 x k)
    - dna (conjunto de t cadenas)
  - **salida**: t k-mers, uno de cada secuencia, que corresponden al motivo más probable, según este método aleatorio
- Ejemplo:
  - dna: `['TTACCTTAAC', 'GATGTCTGTC', 'ACGGCGTTAG', 'CCCTAACGAG', 'CGTCAGAGGT']`
  - k: `4`
  - salida: `?`
    - Ejecuta el método varias veces, imprimiendo el score del resultado

Búsqueda aleatoria de motivos

- Esto es un desastre!
  - Distintas ejecuciones dan distintos resultados
    - Porque cada vez usamos un conjunto inicial aleatorio distinto
- **Prueba**: Ejecutar el algoritmo 1000+ veces, seleccionando el resultado con mejor `score`, sobre http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/seqs4mut.txt
  - ¿Cuál es el mejor resultado y su score?

Búsqueda aleatoria de motivos

- El mejor resultado se alcanza siempre con 100.000 iteraciones, aunque normalmente con 1000 es suficiente

Método	Tiempo (k=15)	Error
medianString	~horas	Exacto
greedySearch	~segundos	Alto
greedySearch (+Laplace)	~segundos	Bajo
randomizedSearch	~minutos	Más bajo

¿Por qué funciona si es aleatorio?

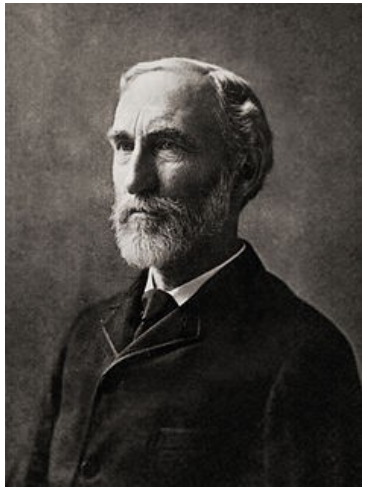
- La probabilidad de que, por azar, capturemos todos los motivos implantados en todas las secuencias es prácticamente nula
  - Pero no la de capturar al menos uno
    - ¿Cuál es esa probabilidad?
    - ¿En qué factor tenemos que multiplicar esa probabilidad para estar seguros de capturarlo?
    - ¿Es suficiente con capturar uno de ellos en `randomizedMotifSearch`?

* La probabilidad de cazarlos todos sería $(1/(n-k+1))^t$
En el caso del AAAAAAAAGGGGGGG es $(1/(82-15+1))^{10} \rightarrow 4.7*10^{-19}$
Sin embargo para cazar uno será $1/(n-k+1) \rightarrow 1.5\%$
* Con un factor 100 nos valdría seguramente
* En principio sí... Aunque podríamos descartarlo si hay algún mínimo local!

Muestreo de Gibbs*

- El muestreo aleatorio altera todos los k-mers en cada ejecución
  - Puede descartar información valiosa
- El muestreo de Gibbs altera sólo un k-mer en cada paso, en vez de todos.

*El muestreo de Gibbs debe su nombre al físico Josiah Willard Gibbs (1839-1903) en honor a sus trabajos teóricos sobre la optimización, aunque realmente los que nombraron así al algoritmo fueron sus autores, los hermanos Stuart y Donald Geman en 1984



Búsqueda avariciosa de motivos

```
GibbsSampler(Dna, k, t, N)
   motivos <- selección aleatoria de k-mers en cada
   cadena de Dna
   mejores <- motivos
   para j <- 1 hasta N
     i <-  random(t)
     perfil <- profile(motivos[-i])
     motivos[i] <-
      mostProbableKmerGivenProfile(Dna[i], k, perfil)
     si score(motivos) < score(mejores)
       mejores <- motivos
    return mejores
```

6 puntos

Ejercicio 11

- Implementar la función `gibbsSampler`:
  - **entrada**:
    - dna (conjunto de t cadenas)
    - k (tamaño del motivo a buscar)
    - `N` (número de iteraciones)
  - **salida**: t k-mers, uno de cada secuencia, que corresponden al motivo más probable, según el muestreo de Gibbs
    - **NOTA**: ejemplo y prueba usan la secuencia consenso de los t k-mers
- Ejemplo:
  - dna: http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/secuencias.txt
  - k: `7`
  - N: `100`
  - Ejecuta 50 veces `gibbsSampler` con estos parámetros y toma el resultado con score más bajo (debería tardar pocos minutos) → Salida (consenso): `GATTACA`

Ejercicio 11

- Prueba:
  - dna:
    http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/secuencias2.txt
  - k: `8`
  - N: `100`
  - Ejecuta 50 veces `gibbsSampler` con estos parámetros y toma el resultado con score más bajo.

Resumen

Método	Tiempo (k=15)	Error (ejemplo)
medianString	~horas	`- (exacto)`
greedySearch	~segundos	`gcccttgttaGaaa`
greedySearch (+Laplace)	~segundos	`tAAAAAAAaGGatGG`
randomizedSearch	~minutos (150.28s con 10.000 its.)	`tAAAAAAAaGGaGGG`
gibbsSampling	~minutos (137.16s con N=1.000 y 200 its.)	`AAAAAAAAGGaGGGG`

Más problemas

- Distribución de nucleótidos de fondo
  - Si algún tipo de nucleótido tiene más frecuencia que el resto, el motivo con mínimo score puede estar compuesto de él, y no ser interesante

taaaaGTCGa
acGCTGaaaa
aaaaGCCTat
aCCCGaataa
agaaaaGGCG

En este ejemplo, A tiene una frecuencia muy alta y enmascara el motivo GCCG (score 4) con el motivo AAAA (score 1)

  - Solución: entropía relativa*

*No vamos a entrar en detalle. Una buena explicación está disponible en:
Pevzner and Compeau, *Bioinformatics Algorithms: An Active Learning Approach*, página 125

Más problemas

- Motivos representados por otros alfabetos
  - Con frecuencia, un motivo se representa mejor con alfabetos híbridos, por ejemplo:

```
W: A o T
S: G o C
K: G o T
Y: C o T
```

CSKWYWWATKWATYYK representa el motivo CSRE en la levadura.
Este motivo engloba a 2^{11} motivos distintos del alfabeto estándar, que por separado serían muy débiles para ser encontrados por los algoritmos tratados aquí

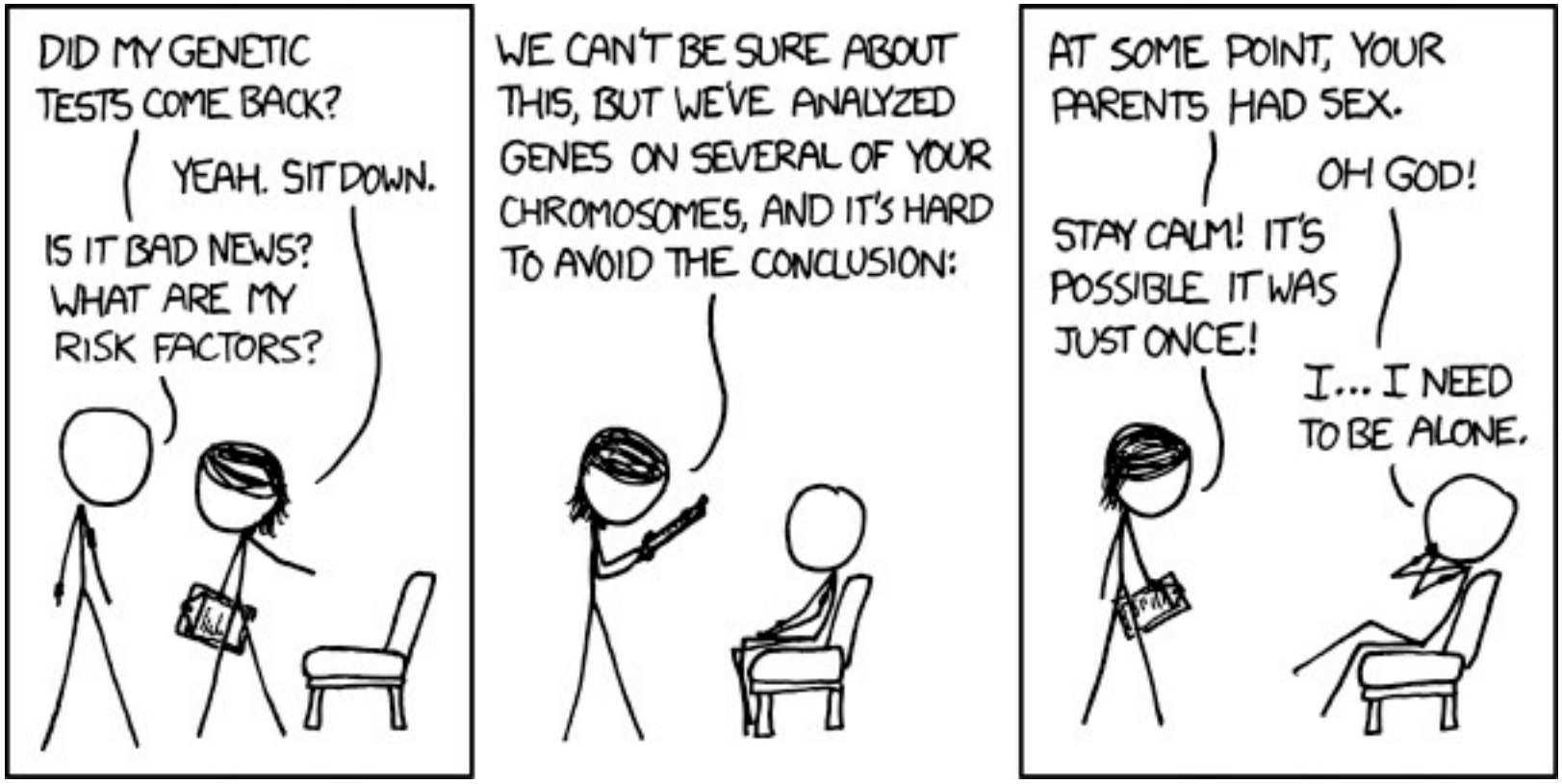
Tuberculosis

- La tuberculosis (**TB**) es una enfermedad infecciosa causada por la bacteria *Mycobacterium tuberculosis* (**MTB**)
- Las variantes resistentes a los antibióticos de esta bacteria están emergiendo, gracias a que estas MTB pueden vivir años en estado de hibernación (en el caso de bacterias se habla de **esporulación**, pues forman esporas durmientes que pueden sobrevivir a condiciones muy duras), sin oxígeno (**hipoxia**)
- Una aproximación es identificar qué genes regulan la detección de hipoxia y comienzan el proceso de esporulación
- Hace 10 años un grupo de biólogos encontró el **regulador de supervivencia durmiente** (**DosR**), un factor de transcripción que regula los genes cuya expresión cambia dramáticamente bajo hipoxia. Se han encontrado al menos 25 genes regulados por DoSR

Ejercicio*

- Con la batería de métodos para búsqueda de motivos que hemos desarrollado, buscar el motivo al que enlaza DosR en los 250 nucleótidos arriba de estos 10 genes que varían sensiblemente bajo hipoxia:
  - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/DosR.txt
- Para ello, podemos utilizar distintos métodos y parámetros
  - Especialmente, k entre 8 y 12, aunque podemos hacer pruebas con k=20

nviar la solución encontrada (código y un pequeño informe de un par de folios) al correo del profesor



http://xkcd.com/830/