Práctica 5
Bioinformática

José Manuel Sánchez Aquilué `759267`

15 de abril de 2021

1. Introducción

En esta práctica se trabaja en base a todo lo aprendido en las anteriores con secuencias de SARS-CoV-2. Se divide en cuatro apartados. En el primero, solamente se menciona la procedencia de las secuencias. En el segundo, se documentará cómo se realizó el multialineamiento y se razonará la calidad de este. En tercer lugar, se realizará un análisis de posibles mutaciones en zonas críticas, atendiendo a criterios estadísticos. Asimismo se comprobará si existe presencia de la variante B.1.1.7 en nuestras muestras. Posteriormente, se construirá los árboles filogenéticos mediante distintos algoritmos. Serán dos programas los utilizados para la elaboración de la filogenia, RAxML y FastTree. Tras evaluar los árboles discutiremos sobre cual es el mejor método.

2. Obtención de información biológica

Atendiendo a las indicaciones del enunciado, desde moodle descargamos el fichero que contiene las secuencias de GISAID. En este caso en particular, trabajaremos con el grupo 3 de secuencias de detectadas en suelo catalán. Asimismo, durante esta sesión se trabajará con la secuencia de referencia, detectada en Wuhan, y con la variante B.1.1.

3. Multialineamiento

A partir de los resultados obtenidos en la práctica 3 podemos concluir que mafft es más óptimo que biopython, tanto en puntuación como en coste temporal. Ahora solo quedaría determinar qué combinación de parámetros da con el mejor multialineamiento. En primer lugar, con el fin de disminuir el tiempo de ejecución, conviene lanzar el programa en paralelo con el máximo número de hilos que soporte nuestra máquina (*–thread -1*). En segundo lugar, se ha de utilizar la heurística de seis-mers, que ya hemos comentado en otras sesiones, porque de lo contrarío el coste no es asumible en un tiempo razonable. Se recuerda que el algoritmo se cambia añadiendo el argumento *–6merpair*, así la

distancia se calcula en base al número de seis-mers compartidos.

En esta ocasión son tres ficheros los que contienen la secuencia a alinear. Nuestra manera de proceder será la siguiente: primero, alinearemos la cepa inglesa con la secuencia de Wuhan, esta será tomada como referencia. El resultado lo guardaremos en un fichero temporal, que será alineado con el resto de secuencias. Cuando termine el programa, el fichero temporal seré eliminado y tendremos un fichero *aligment.fasta* que contendrá el multialineamiento. Al igual que en la práctica 3 escribimos el argumento *–addfragments* y detrás las secuencias a alinear y la de referencia, en ese orden.

En lo que concierne a algoritmos, todos los empleados en la práctica 3 fueron puesto a prueba y se obtenía el mismo resultado o uno con la misma puntuación. De modo que se puede ejecutar cualquier de los vistos, en este caso nos decantamos por un refinamiento iterativo FFT-NS-i de 1000 ciclos. Que todos los métodos concedan un alineamiento similar no implica necesariamente que sea de calidad. Con el fin de determinar la precisión del algoritmo ejecutado, vamos a guiarnos por los criterios del enunciado. Primero de todo: la longitud. La secuencia de referencia; que a su vez es la más larga, empatada con otras; consta de 29903 bases. Mientras que el alineamiento, de 29906. Estaríamos hablando de una inserción de tres gaps en el alineamiento de 400 secuencias de una longitud de decenas de miles de bases, un aumento de lo más insignificante. Por otra parte, el coste temporal es de unos 25 segundos aproximadamente. Si lo comparamos con el de la práctica 3, nos podemos dar por satisfechos, es un precio de lo más razonable. En último lugar, podemos utilizar la puntuación como criterio a seguir. Contamos con un EBI de 0.9951 y una puntuación de 29356 (bajo los criterios de la práctica 3). Nuevamente, una calidad extraordinaria, si entramos en comparaciones con otros multialineamientos de prácticas anteriores.

4. Detección de mutaciones y errores de secuenciación.

Los requisitos para lanzar el programa de cálculo de frecuencias y entropía son tener instalado python, pandas, biopython y numpy. Concretamente las versiones utilizadas fueron: de python la 3.7.4; de pandas la 1.0.5; de biopython la 1.78 y de numpy la 1.19.5. Sobre la instalación, con instroducir el siguiente comando se pueden obtener todos los paquetes mencionados.

```
pip3 install -Iv <nombre de biblioteca>==<version>
```

El script realizado comienza convirtiendo el multialineamiento en un Dataframe, de manera que podemos recorrerlo por columnas. Para cada columna, se calculan las frecuencias de símbolos. Se ha supuesto el índice de conservación lo determinan solo los cuatro nucleótidos más comunes en el DNA (Adenina, Citosina, Guanina y Tiamina). Por tanto no solo desestimamos los GAPs, sino que

Gen/Proteína	Número de alteraciones	Porcentaje de alteraciones
ORF1A	429	3,245 %
Spike	139	3,637 %
ORF3A	74	8,937 %
N	104	8,261 %

Figura 1: Análisis de variabilidad por zonas.

tampoco tenemos en cuenta cualquier combinación de nucleótidos; como pueden ser los enlaces débiles (W), enlaces fuertes (S), cualquier Nucleótido (N), la cetona (K), la pirimidina (Y) y un largo etcétera. Tras el filtrado únicamente debemos aplicar la fórmula $n_v(i)/n(i)$, donde $n_v(i)$ es el número de apariciones del nucleótido v en la columna i y $n(i)$ es la suma de todas las apariciones de la columna i.

Una vez que tenemos las frecuencias podemos sacar la entropía a partir de la siguiente formula.

$$C(i) = -\sum_{v=1}^{4} f_v(i) \ln f_v(i)$$

Cuando la frecuencia de v es cero, el producto $f_v(i) \ln f_v(i)$ no está definido. Ahora bien, si aplicamos la regla de l'hopital, concluimos que el límite cuando tiende a cero es cero. De manera que se puede redefinir la fórmula así.

$$C(i) = -\sum_{v=1}^{4} f_v(i) \ln f_v(i)[f_v(i) \neq 0]$$

Los resultados de cada columna se almacenan en un Dataframe y al terminar este se escribe en un csv, fichero que utilizaremos para hacer el análisis de variabilidad.

Como dice el enunciado, las zonas relevantes donde observar mutaciones son el gen ORF1A, la proteína Spike, el gen ORF3A y el gen N. Se buscará la cantidad de columnas cuyo índice de conservación sea distinto de 0, es decir, que al menos una secuencia sufre una alteración en esa parte. Finalmente sacaremos el porcentaje de posiciones que sufren alteraciones en el tramo que correspondiente a ese gen o proteína. Como podemos apreciar en la figura 1 existe un ligero porcentaje, pero significante, de mutaciones en zonas clave. El script también indica aquellas posiciones cuyo índice de conservación es bastante grande (con un umbral del 0.5). En el gen ORF3A la conservación es baja por lo general. Donde más alta está es en posiciones de mutación de la variante B.1.1.7, a continuación miraremos más en detalle esos casos.

Por otro lado, también nos interesaría conocer la presencia de cepa británica (B.1.1.7), ya que se ha demostrado que es mucho más contagiosa que otras y contiene una mayor carga viral [3]. Es por eso que debemos prestar atención al

Mutación	**porcentaje de secuencias**
C3267T	31,95 %
C5388A	32,1 %
T6954C	29,83 %
A23063T	34,39 %
C23271A	31,95 %
C23604A	34,71 %
C23709T	33,74 %
T24506G	32,2 %
G24914C	32,68 %
C27972T	32,51 %
G28048T	31,78 %
A28111G	30,97 %
28280 $GAT \rightarrow CTA$	33,52 %
C28977T	33,5 %

Figura 2: Análisis de mutaciones características de la cepa B.1.1.7.

porcentaje de secuencias que muestran las mutaciones propias de esta variante. Dado que contamos con un fichero que contiene la proporción de nucleótidos de cada posición, en este análisis solo deberemos consultar la fila indicada y el nucleótido al que muta. Cabe tener en cuenta que en las últimas posiciones, debido al multialineamiento, se ha producido un pequeño desplazamiento. Los resultados (Figura 2) indican que más o menos un tercio de las secuencias tienen mutaciones típicas de la variante B.1.1.7. El primer caso de variante británica en Cataluña se detecto en Enero[1]. Un informe del Ministerio de finales de Marzo la tacha de cepa dominante en Cataluña[2]. De modo que estos datos serán de contagios detectados entre Enero y Marzo.

Por último aclarar que los cálculos sobre los que se apoya este análisis los realiza el script *variability_analysis.py*.

5. Construcción de una filogenia.

5.1. RAxML

Se ha optado por utilizar la versión en local de RAxML en una maquina con linux. En concreto el algoritmo de pthreads, con el fin de poder lanzar los programas en paralelo y reducir el coste temporal. Como alternativa existen versiones en paralelo con mpi o mixtas.

Todos los métodos empleados hacen uso del modelo GTR (General Time Reversible), un modelo de Markov de sustitución [7]. Dentro de GTR podemos distinguir entre dos variantes: CAT y GAMMA. CAT es una aproximación computacional del modelo general, carece de una fundamentación matemática

pero es bastante más rápido que GAMMA. En árboles pequeños, los resultados obtenidos por ambos modelos es parecido. GAMMA es más lento y con arboles descomunales (unos 10.000 taxones) puede fallar.

Para cada aproximación, probaremos distintos algoritmos: hill-climbing rápido[6], hill-climbing con Bootstrap, hill-climbing sin heurística de poda y el análisis rápido con Bootstrap. El análisis rápido con Bootstrap es con diferencia bastante más lento que hill-climbing, de hecho, con la aproximación de GAMMA estuvo muchas horas en ejecución y no devolvió ningún resultado. Asimismo se ejecutó con la opción de calcular el árbol final con una estimación de los sitios variables (CATI y GAMMAI), con el algoritmo por defecto. En último lugar, quedó por probar la corrección de sesgo ASC. Estos experimentos no pudieron hacerse debido a que la corrección ASC no admite la presencia de GAPS u otro tipo de caracteres que no sean los cuatro nucleótidos principales[5].

En lo relativo a la ejecución de los algoritmos, se implementó un script (*raxmlHPC.sh*) que lanza el programa RAxML con todas las alternativas nombradas anteriormente. La aproximación a utilizar se indica con el parámetro -m, mientras que el algoritmo con -f. Ya que instalamos una versión en paralelo, lo ejecutamos con 4 thread (-T 4). Todos los algoritmos necesitan una semilla para las inferencias con parsimonia, como semilla se ha seleccionado el número 161, aunque serviría cualquier otro número. Los algoritmos con Bootstrap también requieren de una semilla; a parte de un número de iteraciones, que en este caso hemos elegido 50. En Bootstrap se ha escogido el valor 77. Naturalmente, si fuésemos a utilizar este script más de una vez, en vez de tener semillas constantes, emplearíamos variables, como puede ser la hora de ejecución. Por último, el argumento -k obliga a sacar los árboles con la longitud de sus ramas.

Como era de esperar, en la práctica, GAMMA es más lento que CAT y paradójicamente no aporta mejores soluciones, de hecho, es al revés (Figura 3). Por otra parte, descubrimos que más tiempo de ejecución no implican mejores resultados. En secuencias tan similares, como en este caso, una aproximación muy compleja como hill-climbing con Bootstrap con GAMMA, devuelve árboles de peor calidad que los generados por otros métodos. Sin embargo, ese mismo algoritmo con la aproximación CAT devuelve el mejor resultado. Vemos que la heurística de poda no interfiere mucho en la puntuación, pero desprendernos de ella ralentiza bastante el proceso. Por último, el método de análisis rápido no merece la pena, porque, a pesar de mejorar ligeramente la puntuación, el tiempo de ejecución incrementa con creces. Mientras que la estimación de los sitios variables (CATI y GAMMAI) sí que merece la pena, ya que con unos minutos más obtienes una de las mejores puntuaciones.

5.2. FastTree

En esta ocasión volvemos a decantarnos por una ejecución en local desde linux. En lo que concierne a la instalación, desde la web de fastTree se descarga

Algoritmo	Tiempo	Puntuación
CAT hill-climbing	5 min 20 segundos	-55992.111375
CAT hill-climbing con Bootstrap	25 min 33 segundos	-49883.460905
CAT hill-climbing sin heurística de poda	11 min	-55992.111375
CAT análisis rápido con Bootstrap	1 h 6 minutos	-55974.011339
GAMMA hill-climbing	8 min 20 segundos	-56013.297556
GAMMA hill-climbing con Bootstrap	53 min 10 segundos	-64992.941444
GAMMA hill-climbing sin heurística de poda	12 min 43 segundos	-56013.303522
CATI	9 min 3 segundos	-55720.546957
GAMMAI	12 min 47 segundos	-55720.061704

Figura 3: Resultados de RAxML

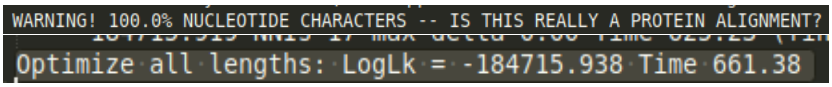

Figura 4: Aviso y puntuación del algoritmo JTT.

directamente el ejecutable, de manera que solo es necesario darle permisos para poder empezar a utilizar el programa. Nuevamente se ha elegido la versión multithreaded, por las mismas razones de rendimiento expuestas en el subapartado anterior. En RAxML el programa se apoyaba en pthreads, aquí en OpenMP.

Volvemos a hacer distinción entre la aproximación CAT y GAMMA. Ahora, además del algoritmo de GTR, tenemos el modelo de Jukes-Cantor. También existen algoritmos como Jones-Taylor-Thornton (JTT), Whelan & Goldman (WAG) y Le and Gascuel (LG), aunque según la documentación[4] estos son exclusivos para cadenas de aminoácidos. Si probamos a ejecutar el alineamiento con esos algoritmos nos avisará de que las secuencias son de nucleótidos y nos dará una puntuación nefasta (Figura 4). Si insistimos, e indicamos a través del argumento -nt que son nucleótidos, el algoritmo que ejecuta es Jukes-Cantor. Por tanto, no se ha tenido en cuenta esos algoritmos exclusivos para proteínas. Ahora bien, como en un principio sí que se entraron a valorar, dejaremos en la entrega el output de la ejecución y los árboles generados, tanto la representación visual como la estructura en newick.

Sobre la ejecución decir que, si se añade el flag -gamma se ejecuta la aproximación homónima, mientras que si no se coloca nada se lanza con CAT. Algo similar ocurre con el argumento -gtr, cuya presencia indica que se ha de hacer uso del algoritmo con el mismo nombre, y su ausencia, del algoritmo JC. Al ser cadenas de nucleótidos no puede faltar -nt.

Los resultados obtenidos son coherentes (Figura 5). Cuanto más complejo es el método, más tiempo de ejecución, pero mejor puntuación.

Algoritmo	Tiempo	Puntuación
CAT JC	2 min 45 segundos	-66937.165
CAT GTR	6 min 57 segundos	-63330.967
GAMMA JC	3 min 5 segundos	-58596.959
GAMMA GTR	6 min 54 segundos	-57567.419

Figura 5: Resultados de FastTree

5.3. Conclusiones

Al tratarse de un multialineamiento con pocas secuencias, RAxML devuelve una buena puntuación en un tiempo razonable, entre 5 y 25 minutos. Mientras que FastTree con su método más sofisticado tarda más y da peores resultados que RAxML por defecto. Es por eso que aconsejo emplear la aproximación de CATI de RAxML y dejar FastTree para árboles con un exorbitante número de secuencias.

Como se puede comprobar, la calidad de los árboles se ha evaluado únicamente en base a criterios de puntuación y tiempo. En la entrega se dejan las visualizaciones y se advierte que, al tratarse de árboles con tantas ramas, es muy complejo llegar a alguna descripción a partir de las imágenes. En último lugar, aclarar que las representaciones fueron generadas con la herramienta oline iTOL, utilizada en la práctica anterior.

Referencias

[1] Diario ara. Catalunya confirma el primer cas de la variant britànica del coronavirus. urlhttps://www.ara.cat/societat/verges-confirma-britanica-covid-19-catalunya$_{12}$543558.*html*, 2021.

[2] Ministerio de Sanidad. Actualización de la situación epidemiológica de las variantes de sars-cov-2 de importancia en salud pública en españa. 2021.

[3] Azucena Martín. La variante británica del coronavirus no es más mortal que el resto, según los datos. urlhttps://hipertextual.com/2021/04/cepa-britanica-coronavirus, 2021.

[4] Morgan N. Price. *FastTree Manual*. Adam Arkin's group, 2010.

[5] Alexandros Stamatakis. *The RAxML v8.2.X Manual*. Heidelberg Institute for Theoretical Studies, 2016.

[6] Blagojevic F. Nikolopoulos Stamatakis, A. Exploring new search algorithms and hardware for phylogenetics: Raxml meets the ibm cell. 2007.

[7] Yang Z. *Computational molecular evolution*. Oxford University Press, 2006.