Trabajo Práctico Integrador

Análisis y Alineamiento de Secuencias

Proyecto: Bioinformática aplicada a un brote de brucelosis

Alumnos: García, Justo; Valle, Abril.

Docentes: Dra. Illeana Tossolini; Lic. Atilio Rausch.

Fecha: 31 de octubre de 2024.

Introducción

En este proyecto, formado por un grupo de investigación multidisciplinario (veterinarios, biotecnólogos, epidemiólogos y bioinformáticos), se tiene como objetivo caracterizar a la cepa de *Brucella suis* responsable de un brote de brucelosis porcina en el oeste de Entre Ríos.
A cada equipo de investigación del área bioinformática se le brindó un Set de aproximadamente 150-300 electroferogramas (obtenidos mediante secuenciación de Sanger, procedentes de clones individuales de una biblioteca construida con ADN fraccionado al azar de la bacteria en cuestión, aislada de muestras de cerdos infectados) y aproximadamente entre 55.000 a 70.000 secuencias cortas (generadas usando la tecnología Illumina, algunas *single-end* y otras *paired-end reads*). En el caso de nuestro equipo, trabajaremos con el **Set 3**.
Los investigadores principales necesitan que estas secuencias sean ensambladas y anotadas y, en todos los casos, deben finalizar con la construcción de un *scaffold* cuya secuencia consenso posee entre 26.000-30.000 pb.

Desarrollo:

1. Informe bibliográfico acerca de las bacterias del género Brucella,

Brucella suis (NCBI:txid29461) es una especie de α-proteobacteria perteneciente a la familia *Brucellaceae* (Schoch et al., 2020). Son bacterias gram negativas que sobreviven dentro de macrófagos y que causan una enfermedad zoonótica conocida como brucelosis, que afecta a mamíferos (principalmente a cerdos), incluyendo a humanos. Dicha enfermedad puede ser transmitida por contacto directo con animales infectados o productos derivados de estos.

En humanos, la brucelosis es una enfermedad febril sistémica y crónica que se caracteriza por la aparición de fiebre, dolor de cabeza, debilidad, sudoración, escalofríos, pérdida de peso y malestar general (*Brucelosis*, s. f.-a; *Brucelosis*, s. f.-b). En cerdos, provoca abortos, infertilidad y disminución de la productividad, lo que impacta directamente en la industria porcina. La enfermedad se encuentra presente en varias regiones del mundo, especialmente en aquellas zonas donde no hay controles rigurosos de la sanidad animal.

El genoma de *Brucella suis* (*Brucella Suis 1330 Genome Assembly ASM750v1*, s. f.) tiene un tamaño de 3.3 Mb y está compuesto por dos cromosomas circulares,los cuales contienen un total de 3,233 genes. Actualmente, hay 95 ensamblados de genomas disponibles para esta especie.

Cromosoma	RefSeq	Tamaño (pb)	GC%
I	NC_004310.3	2.107.794	57
II	NC_004311.2	1.207.381	57,5

Tabla 1: Información sobre cromosomas del genoma de *Brucella suis*.

Figura 1: Cultivo de Brucella suis.

2. Ensamblado de las muestras secuenciadas

a) Análisis de calidad y procesamiento para secuencias crudas.

En primer lugar, procesamos las secuencias obtenidas por el método Sanger para eliminar las regiones de baja calidad y aquellas correspondientes al vector.

Con este objetivo, utilizamos el programa **Pregap4** del paquete Staden, el cuál ejecutamos con el comando pregap4.

Primero, se configura la convención de nombres a utilizar en los electroferogramas para utilizar toda la información disponible en el ensamblado:
File → Load Naming Scheme → Browse → Cargo el archivo 'sanger_names_new.p4t' → OK.

Luego, selecciono y cargo los archivos a ensamblar:
Add Files → Carpeta 'set3' → Filtro por archivos de tipo SCF (*.scf) → Cargo todos los archivos.

A continuación configuramos los módulos que consideramos necesarios para nuestro procesamiento. Estos se aplicarán en forma secuencial a los electroferogramas procesados. Tal como se puede apreciar en la imagen a continuación se seleccionaron los siguientes módulos:

- **General configuration:** ‘No’ para que se utilice el nombre del archivo (si pusiera que sí, se utilizaría el nombre del electroferograma dado en el secuenciador automático).
- **Estimate Base Accuracies:** asigna un valor de calidad a cada base.
- **Initialize Experiment Files:** se inicializan “*Experiment Files”* que son archivos de texto en los cuales se guarda la información relativa a la secuencia, calidad de la base, corte de secuencia por calidad, etc.
- **Augment Experiment Files:** adiciona a los *Experiment Files* la información incorporada mediante el esquema de nombres y el tamaño del fragmento con el cual se construyó la biblioteca. Esto último se configuró de la siguiente manera: Click en botón Experiment File Line Type → Busco el campo ‘SI’ (*Size of Insert*) → coloco el valor “1000..3000” → Ok and Save.
- **Quality Clip**: enmascara las secuencias que considera de baja calidad. Parámetros por default.
- **Sequencing Vector Clip:** enmascara la secuencia correspondiente al vector, si es que las secuencias provienen de un plásmido.
- **Screen For Unclipped Vector:** busca dentro de las secuencias si existen secuencias de vector, para eliminarlas, si es que estas no se encuentran en los extremos.
- **Interactive Clipping:** muestra de forma interactiva los resultados finales de los módulos para cada electroferograma procesado con el visualizador de Staden **trev**.

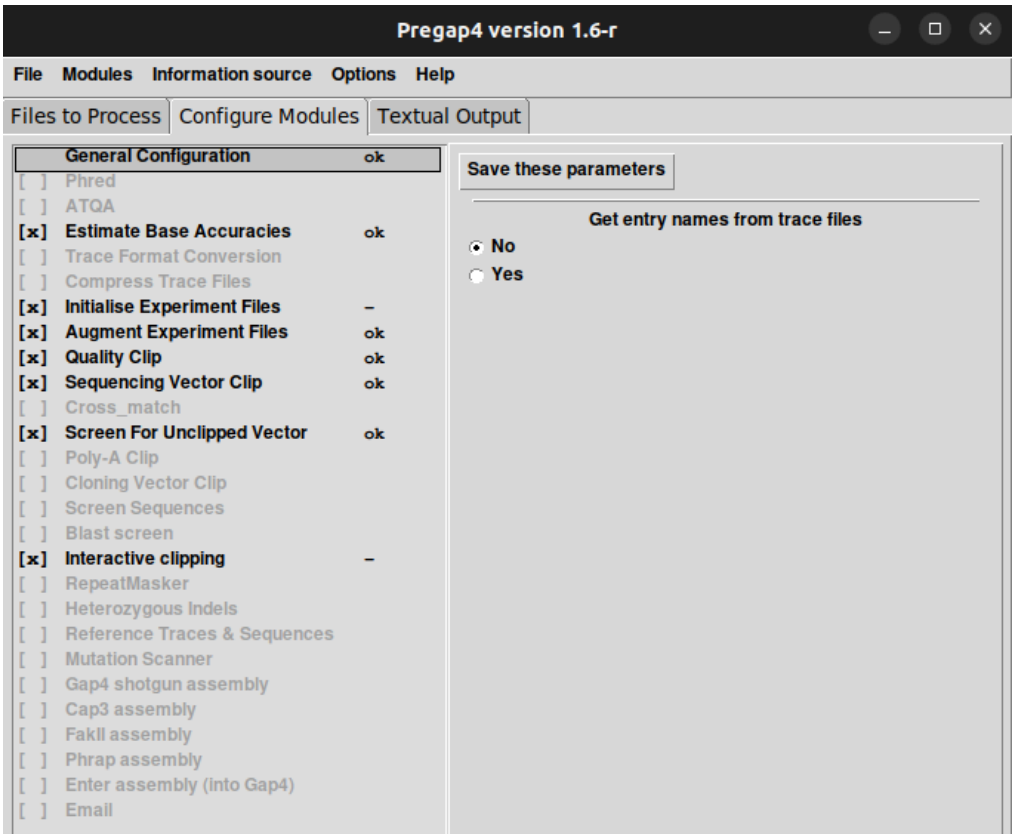


Figura 2: Configuración de módulos a correr por Pregap4

Habiendo configurado los módulos a aplicar a las secuencias seleccionadas, ejecutamos el programa y pudimos observar que los electroferogramas fueron procesados sin problemas-

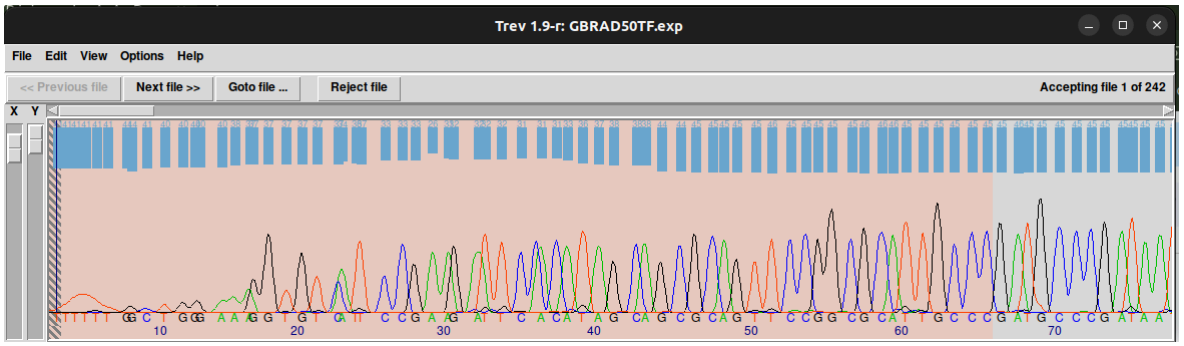


Figura 3: corte por calidad (líneas diagonales) y secuencia de vector al inicio de la secuencia.

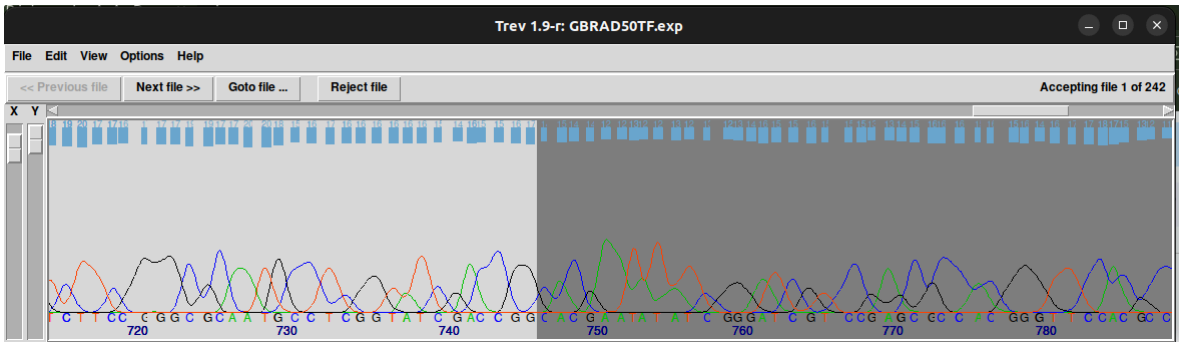


Figura 4: corte por calidad al final de la secuencia.

Los resultados de estos procesos producen:

- Un archivo .**exp** con el mismo nombre de cada electroferograma (“Experiment Files”)
- Una serie de archivos con listas de electroferogramas que pasaron los diferentes requerimientos de cada módulo: por ejemplo, **pregap.passed** (sí pasaron) y pregap.failed (no pasaron).

Para las **lecturas cortas** obtenidas por **Illumina**, analizamos la calidad de las mismas, pero no realizamos procesamiento adicionales. Para esto, utilizamos la herramienta **FastQC**, la cual proporciona un informe detallado sobre la calidad de las lecturas generadas por el secuenciador. Ejecutamos el comando:

```
fastqc *fq -o directorio_salida
```

Esto nos genera un archivo .html correspondiente a Set3.fq.

¿Qué tipo de biblioteca se secuenció? El tipo de biblioteca que se secuenció es una biblioteca construida con ADN genómico fraccionado al azar de la bacteria *Brucella suis*.

Basic Statistics

Measure	Value
Filename	Set3.fq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	65000
Total Bases	9.7 Mbp
Sequences flagged as poor quality	0
Sequence length	150
%GC	55

Figura 5: Módulo 'Basic Statics' de fastqc.

Podemos observar que se obtuvieron 65000 lecturas (reads) y que la longitud de las mismas es de 150 pb.
Además, se tiene un %GC de 55, el cual es un valor cercano al %GC reportado para el genoma de referencia (ver Tabla 1).

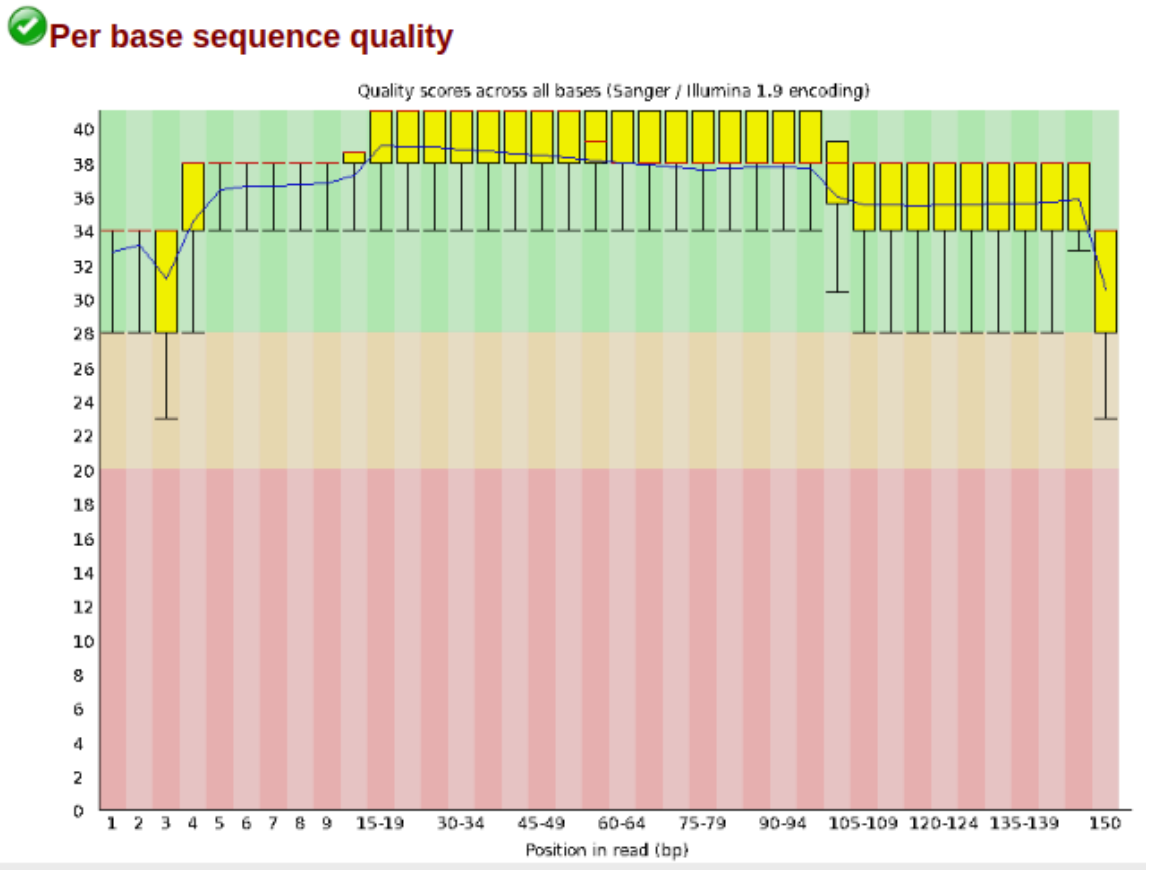


Figura 6: modulo 'Per base sequence quality'.

En la figura 6, observamos la calidad de las lecturas por posición en las secuencias. El

eje X representa la posición de las bases a lo largo de la secuencia y el eje Y representa la calidad de las bases según un Phred Score, donde un valor más alto representa mayor calidad. Podemos ver que, en general, se tiene una buena calidad ya que la mayor parte de las secuencias tienen valores de Phred mayores a 30, lo cual infiere una buena calidad de las mismas. Observamos también una disminución de calidad en el final de las lecturas, lo cual es común en las secuencias Illumina.

b) Ensamblado secuencias largas

Procedemos a ensamblar las secuencias largas con el programa Gap4 del paquete Staden, el cual abrimos con el comando `gap4`.

Creamos la base de datos desde el menú **File → New**.

Luego, para ensamblar las secuencias seleccionamos en el menú **Assembly → Normal Shotgun Assembly**, donde nos aparece una ventana donde seleccionamos los archivos a ensamblar (pregap.passed) y las opciones de ensamblado, las cuales dejamos por defecto.

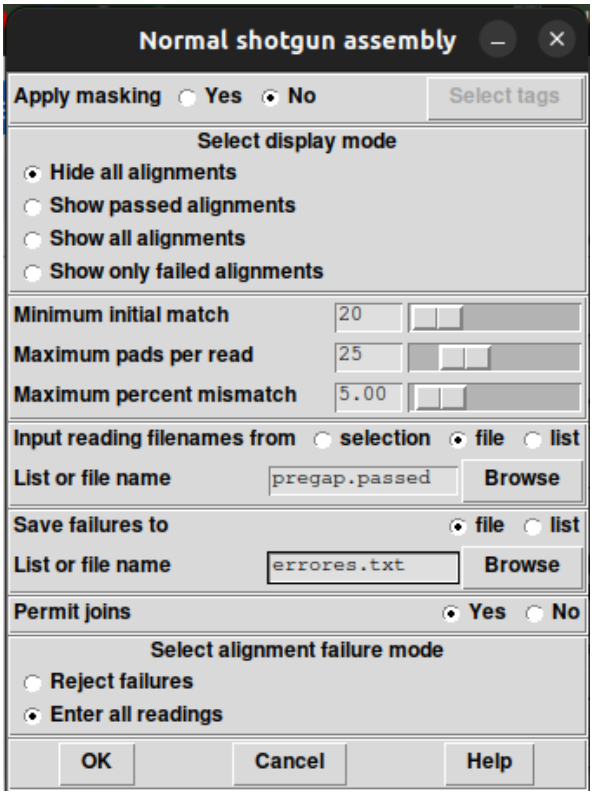


Figura 7: ventana Normal shotgun assembly con las respectivas configuraciones.

De la ventana principal de Gap4, obtenemos la siguiente información:

- **Secuencias procesadas:** 242
- **Número de secuencias ingresadas en la base de datos:** 242

Se abre la ventana **Contig Selector**, que muestra la representación gráfica de los contigs formados durante el ensamblado.

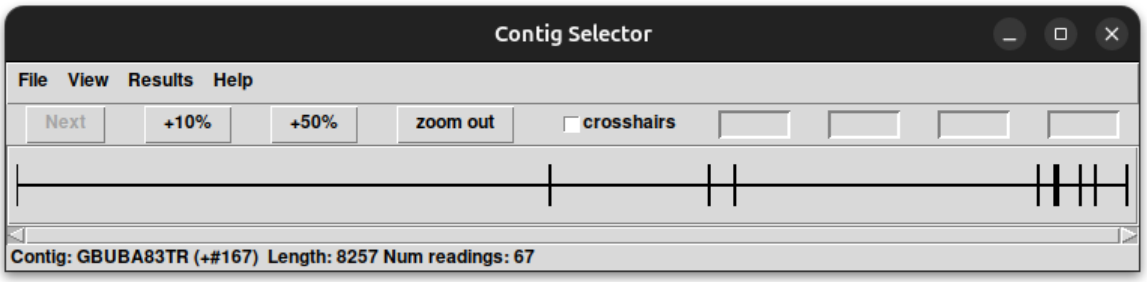


Figura 8: Contig Selector.

Visualizamos también la **Contig List**, que nos proporciona un resumen del largo de los contigs y el número de secuencias que forman parte de cada uno. Observamos que, a partir de 242 secuencias, se formaron **9 contigs**.

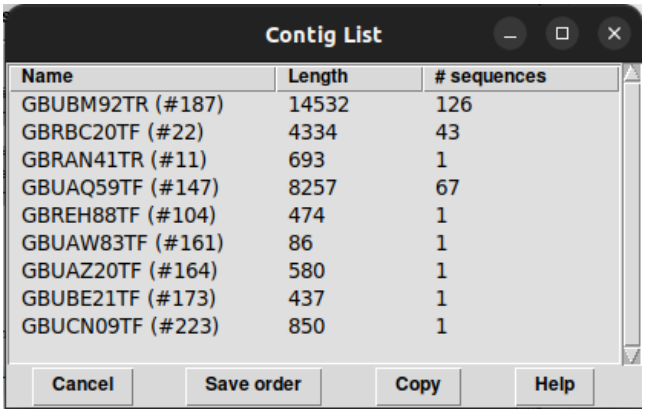


Figura 9: Contig List.

Ahora verificamos si existen superposiciones entre los contigs generados, de manera de poder unirlos.
Para determinar posibles alineamientos, hacemos desde el menú **View → Find internal joins.** Se despliega una pantalla de configuración de los parámetros de los alineamientos a considerar, los cuales dejamos por defecto.

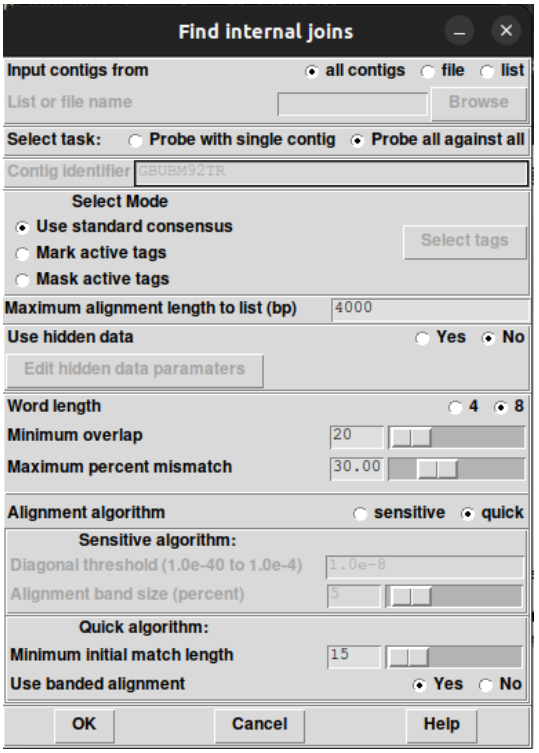


Figura 10: parámetros para la opción 'Find Internal Joins'.

Obtenemos los resultados de los alineamientos entre los contigs, representados mediante un dot-plot en el cual las barras muestran la superposición entre las diferentes secuencias.

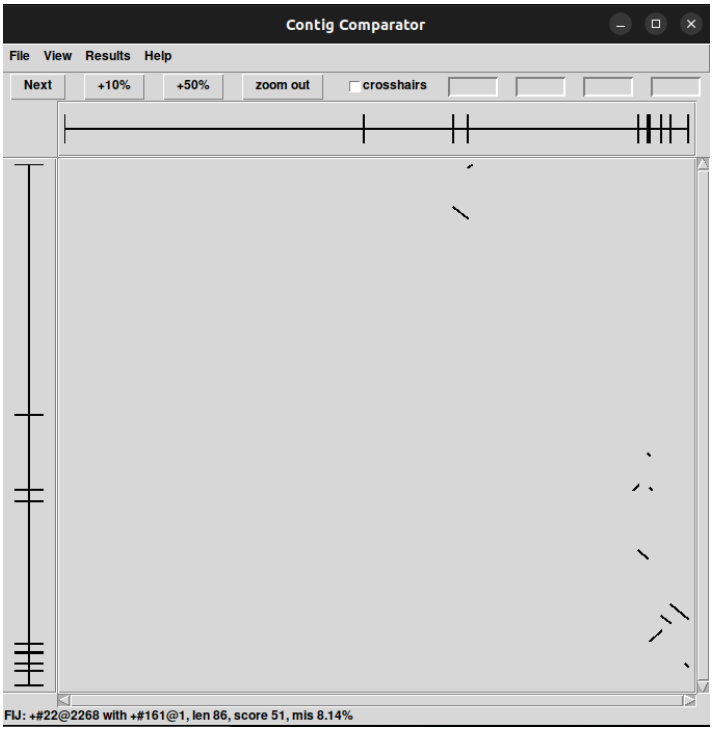


Figura 11: Dot-Plot

Para determinar si hacer un join, establecimos un criterio del 10%. Todas las superposiciones entre las secuencias presentaron un porcentaje menor al criterio, por lo cual hicimos join en todos los casos.

Figura 12: visualización de uno de los porcentajes de mismatch.

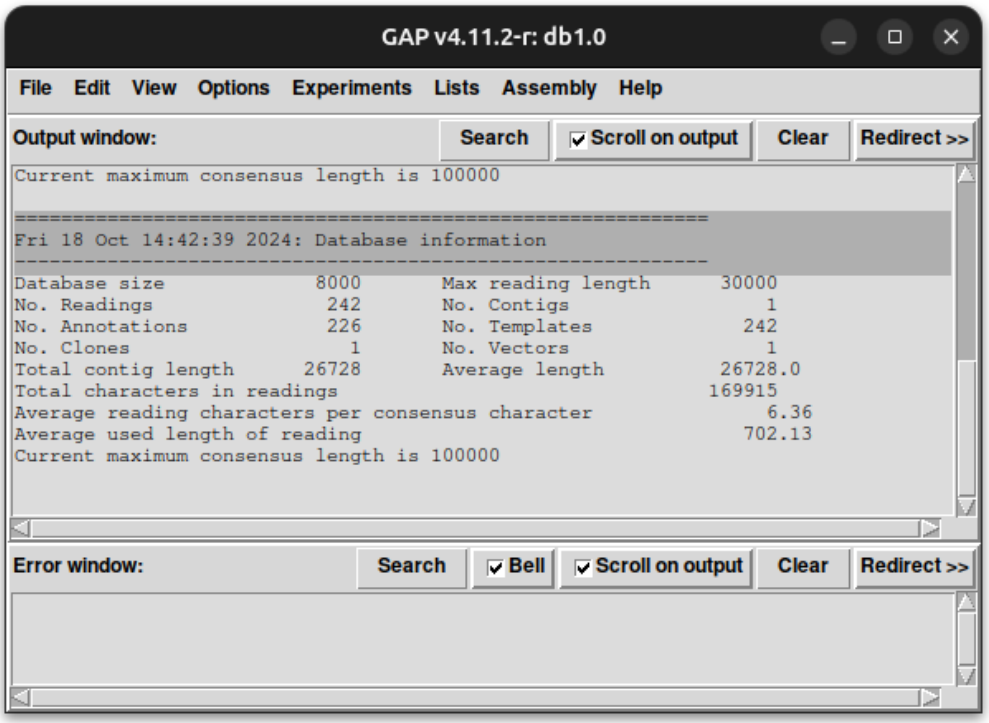


Figura 13: Resumen de la base de datos de Gap4 una vez obtenida la secuencia consenso.

Analizamos con seqkit la secuencia consenso obtenida:

```
file              format  type  num_seqs  sum_len  min_len  avg_len  max_len
cons_final.fasta  FASTA   DNA          1   26,430   26,430   26,430   26,430
```

Llegado a este punto, y gracias a la complementariedad de herramientas, nos dimos cuenta que teníamos discrepancias en los resultados obtenidos. Por un lado, la base de datos de gap4 nos indicaba que nuestra secuencia consenso tenía un largo de 26728 pb y por el

otro, seqkit, que nuestra secuencia consenso tenía un largo de 26430. Al consultar con el docente, nos dimos cuenta que al exportar dicha secuencia estábamos quitando los pads. Como luego debíamos hacer un ensamblado híbrido, no tenía sentido llevar a cabo este proceso.
Por ello, procedimos a exportar nuevamente la secuencia consenso ajustando este parámetro.

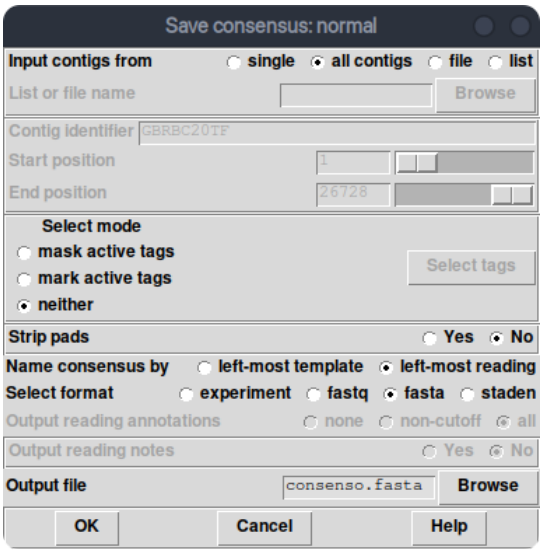


Figura 14: guardado de la secuencia consenso con los pads.

Llevando a cabo esta modificación, pudimos resolver estas diferencias. A continuación se muestra el resumen de seqkit para la secuencia consenso.

```
file            format  type     num_seqs  sum_len  min_len  avg_len  max_len
consenso.fasta  FASTA   Protein         1   26,728   26,728   26,728   26,728
```

Para calcular la cobertura de la secuencia consenso obtenida del ensamblado, utilizamos la siguiente fórmula:

$$C = \frac{l\,x\,n}{L}$$

Donde:

- C: cobertura
- L: longitud del contig
- l: longitud de las lecturas
- n: número de lecturas

Reemplazando a partir del resumen de la figura 13:

$$C = \frac{702.13\ x\ 242}{26728} = 6.357$$

c) Ensamblado híbrido

Llevamos a cabo un ensamblado híbrido utilizando el contig del paso anterior y las secuencias cortas producidas por Illumina. Como se muestra en el resumen siguiente, estas son 65.000 lecturas cortas de un largo de 150 bases.

```
file     format  type  num_seqs    sum_len  min_len  avg_len  max_len
Set3.fq  FASTQ   DNA     65,000  9,750,000      150      150      150
```

Para el ensamblado híbrido, utilizamos el programa *SPAdes,* el cual ejecutamos con el siguiente comando:

```
spades.py --s1 Set3.fq --trusted-contigs contig.fasta --careful -o
spades
```

Como usamos el resultado del ensamblado por Sanger, le indicamos al programa la opción `–trusted-contigs contig.fasta` y, como el tipo de biblioteca que se usó para Illumina es *single pair no unpaired* y de la hebra R1 indicamos esto con —s1 para poder pasarle correctamente los archivos de entrada con las *reads*.

```
file            format  type    num_seqs  sum_len  min_len  avg_len  max_len
before_rr.fasta  FASTA   DNA           1   30,000   30,000   30,000   30,000
contigs.fasta    FASTA   DNA           1   30,000   30,000   30,000   30,000
scaffolds.fasta  FASTA   DNA           1   30,000   30,000   30,000   30,000
```

Obtuvimos un scaffold, con una cobertura de 158. Este valor lo extrajimos del identificador del scaffold:

```
>NODE_1_length_30000_cov_158.511780
```

En resumen:

Ensamblado	Longitud (pb)	Cobertura
Con secuencias largas (Sanger)	26,430	6.36x
Híbrido (Illumina+Sanger)	30,000	158x

Tabla 2: Resumen de los datos obtenidos de los ensamblados.

Al comparar los resultados obtenidos con cada ensamblado, observamos que con el ensamblado híbrido se obtuvo un contig de mayor longitud en comparación con el ensamblado obtenido con Sanger. Además, tiene una cobertura significativamente mayor. Por lo tanto, el ensamblado híbrido es más confiable.

Llevamos a cabo el alineamiento del scaffold contra el genoma de referencia de *Brucella suis.* Utilizamos Quast con el siguiente comando:

```
quast.py contigs.fasta -r suis.fna -o quast_results
```

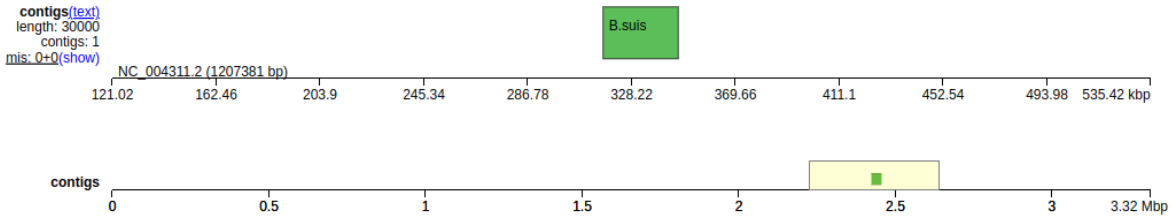


Figura 15: visualización del archivo 'report.html'

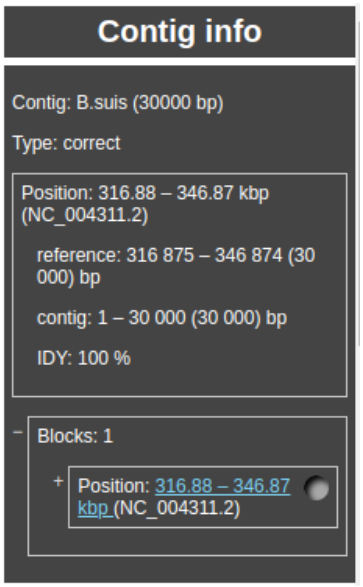


Figura 16: Contig info

A partir del análisis con Quast, observamos que la secuencia scaffold se alinea con el genoma de referencia en la posición 316.88-346.78 kbp del mismo, sin sufrir reordenamientos ni alteraciones.

3- Reportes

Ensamblado	%GC
Con secuencias largas (Sanger)	55.95
Híbrido (Illumina+Sanger)	55.57

Tabla 3: Reportes de Ensamblado y %GC.

El %GC de los contigs obtenidos mediante ensamblado son menores que el %GC de los cromosomas del genoma de referencia (Ver tabla 1) .
El contenido G+C en un organismo procariota es importante porque nos puede dar una idea de dónde se localizan promotores y exones. Un %GC alto puede estar indicando una mayor estabilidad del ADN mientras que un %GC bajo puede estar relacionado con tasas de mutación más altas y menor estabilidad (Šmarda et al., 2014).

4- Predicción de genes y Anotación del ensamblado

a) Predicción de genes

Habiendo encontrado la secuencia consenso nos encontramos en condiciones de realizar la predicción de genes. Siguiendo las recomendaciones decidimos llevarlo a cabo por dos caminos:

1. Un método *ab initio*
2. Un *pipeline* integrador (predicción + anotación)

Método ab initio

En primer lugar decidimos utilizar Glimmer (Delcher, 2006) para el primer caso, debido a que tenemos un genoma de referencia que nos permite entrenar los modelos ocultos de markov (HMM) que implementa este software. Por otro lado, lo elegimos también por la capacidad de ejecutarlo localmente y pudiendo ajustar los diferentes parámetros de él.

Como dijimos esta herramienta realiza la producción a través de HMM. Lógicamente, debemos entrenar estos modelos para luego predecir los orfs de nuestra secuencia ensamblada. Con este fin descargamos la secuencia en formato FASTA del genoma de referencia previamente referenciado (GCF_000007505.1).
A partir del genoma procedemos a realizar los pasos para extraer el set de entrenamiento utilizando las herramientas contenidas en el paquete.

```
tigr-glimmer long-orfs -n -t 1.15 suis.fna bsuis.longorfs
```

Con este comando, obtuvimos los diferentes orfs con información de su inicio, fin, hebra y marco de lectura. A partir de esta tabla buscamos la secuencia de los orfs en el genoma de referencia con el siguiente comando:

```
tigr-glimmer extract -t suis.fna bsuis.longorfs > bsuis.train
```

Dicho comando extrae las secuencias de los ORFs a partir del genoma completo en suis.fna y las guarda en un archivo FASTA llamado bsuis.train.

Teniendo ya el set de entrenamiento tal como lo necesita Glimmer, podemos pasar a construir el HMM.

```
tigr-glimmer build-icm -r bsuis.icm < bsuis.train
```

Finalizada la ejecución del comando anterior tenemos en el archivo bsuis.icm el modelo oculto de Markov ya entrenado y listo para usar en la predicción de ORFs de nuestro ensamblado híbrido. Para ello, ejecutamos el siguiente comando:

```
tigr-glimmer glimmer3 -o50 -g110 -t30 contigs.fasta bsuis.icm bsuis
```

Parámetros:

- **-o**: Establece la longitud máxima de superposición en n. Se permiten superposiciones de esta cantidad o menos de bases entre genes.
- **-g**: Establece la longitud mínima del gen en n nucleótidos.
- **-t**: establece el puntaje umbral en el score del algoritmo para que sea considerado un potencial gen (el puntaje de las regiones tienen que ser igual o mayor que el umbral). Este puntaje es el de la columna etiquetada como “InFrm” en el archivo .detail, no el puntaje decimal en la columna etiquetada como “Raw”.

Los archivos finales obtenidos para la predicción son los siguientes:

- *bsuis.predict* → Coordenadas de los genes predichos.
- *bsuis.detail* → Parámetros y resultados detallados.

Luego de la predicción podemos observar que se creó la carpeta bsuis tal como lo solicitamos en el comando, donde podemos encontrar diferente información sobre los orfs predichos. Por ejemplo, en `bsuis.predict` podemos encontrar información sobre los ids de los 27 ORF predichos, su inicio, fin, hebra, marco de lectura y el score de la predicción.

Llegado a este punto procedimos a utilizar el script provisto por la cátedra para la práctica referente a predicción y anotación, el cuál permite la extracción de la secuencia de los orfs predichos de nuestra secuencia ensamblada a partir del archivo predict descrito previamente.

Ejecutamos este script de Python y obtuvimos dos archivos multifasta, uno para los orfs en la hebra positiva y otro para los de la hebra negativa. Revisando estos archivos nos dimos cuenta que otra vez nos topamos con el problema reportado en la práctica. Cuando lo discutimos en ese espacio pudimos ver que el problema era el último ORF, el cuál está en la hebra negativa pero tiene el inicio y el fin al revés. La versión de glimmer utilizada es la 3.02, distribuida a través de bioconda.

```
08:03:32  |predYanotAyAS|justo@net glimmer ±|main ✗|→ conda list | grep glimm
glimmer                   3.02                 h87f3376_6    bioconda
```

Figura 17: versión de Glimmer.

Tras discutirlo con la cátedra decidimos ignorar este orf y eliminarlo de la lista de predichos. Volvimos a ejecutar el script de Python y al observar las estadísticas de los multifastas con seqkit pudimos apreciar que ya no teníamos ninguna secuencia de largo 0.

A continuación se presentan los dos resultados de stats para los orfs antes y después de eliminar el último orf. En el primero de ellos podemos observar que hay uno de largo 0.

```
file               format  type  num_seqs  sum_len  min_len  avg_len  max_len
orfs_negativos.fasta  FASTA   DNA       8    8,694        0  1,086.8   1,593
orfs_positivos.fasta  FASTA   DNA      19   15,321      132    806.4   1,563
```

Figura 18: resultados antes de eliminar el último orf.

```
file               format  type  num_seqs  sum_len  min_len  avg_len  max_len
orfs_negativos.fasta  FASTA   DNA      7      8,694        900    1,242  1,593
   orfs_positivos.fasta  FASTA   DNA      19   15,321  132    806.4  1,563
```

Figura 19: resultados después de eliminar el último orf.

Podemos ver también que se obtuvieron un total de 26 orfs.

Pipeline integrador: prokka

Como se mencionó anteriormente y como lo solicitaba la consigna, llevamos a cabo el proceso de predicción con dos herramientas, la ab initio descrita en la sección anterior y un pipeline integrador que además realiza la anotación de los orfs.
Utilizamos el siguiente comando:

```
prokka contigs.fasta --evalue 1e-05 --rawproduct
```

```
file               format  type  num_seqs  sum_len  min_len  avg_len  max_len
PROKKA_10252024.faa  FASTA  Protein  26      8,139    50        313     530
```

Prokka utiliza el software Prodigal 2.6 para la predicción de genes y, para la anotación, compara las secuencias contra varias bases de datos (como UniProt y RefSeq), para poder así asignar funciones a los genes.

Para la predicción de genes, es más confiable integrar varios métodos y quedarnos con una sola tabla de genes predichos.
Por simplicidad, a partir de aquí continuamos con la predicción y anotación generada con Prokka.

b) Visualización del ensamblado

Para visualizar el ensamblado junto a su anotación, utilizamos el visualizador de genomas ‘Artemis’.
Visualizamos el archivo .gff (General Feature Format), el cual contiene:

- La posición de las características anotadas en el genoma (genes, ARNs, CDS, etc.)
- Información sobre las proteínas hipotéticas y otros elementos anotados.

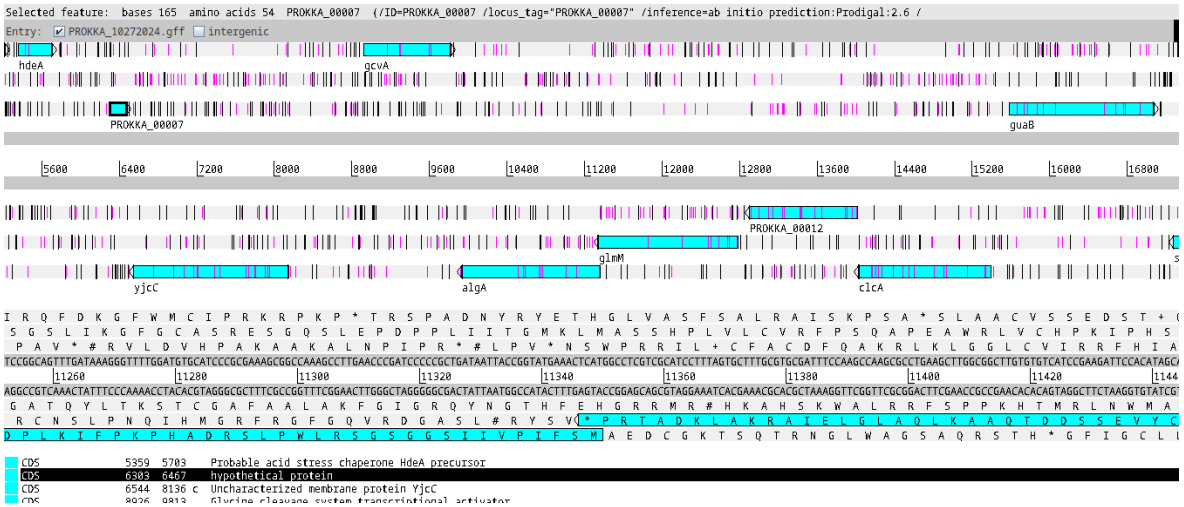


Figura 20: visualización del ensamblado junto a su anotación en Artemis.

Modificamos la representación, para representar con colores rosa los codones de inicio y negro los de stop.

c) Comparación contra bases de datos

Continuando con los resultados obtenidos de la ejecución de prokka, decidimos realizar un script en Python que nos permita refinar y completar la anotación de los genes predichos comparando las proteínas predichas contra bases de datos. Este se encuentra adjuntado en el Anexo o en nuestro repositorio de github (*justog220/TIF-AyA: Trabajo integrador final para Análisis y Alineamiento de Secuencias*, s. f.). Este integra tres herramientas para su anotación:

- BLAST (remoto, a partir del módulo disponible a través de Bioython). Se decidió limitar el número de hits a 10 por los tiempos de ejecución que representaba buscar un número mayor para un gran número de proteínas.
- Pfam_scan (Zielezinski, 2022/2024), para analizar una proteína query contra PFAM a través de modelos ocultos de markov. El script utilizado utiliza HMMER internamente, tal como se lo hizo en la práctica.
- DeepLocPro (Jaimomar99, 2023/2024; Moreno et al., 2024), que se presenta como una extensión a DeepLoc (Thumuluri et al., 2022) para predecir la/s localización/es subcelular/es en células procariotas (su antecesor aplica a eucariotas). Este utiliza redes neuronales y se distribuye como modelo de PyTorch.

Todas estas herramientas fueron integradas en el script de Python mencionado previamente. En el se lanza la ejecución de los diferentes algoritmos y se parsean sus salidas de modo que se obtenga un archivo Tsv con los datos de la proteína predichos por Prokka, los datos de pfam (inicio, fin, número de acceso, nombre, etc) y la localización predicha. Por otro lado, se guardan en una carpeta `blast_outputs` los resultados de cada ejecución de BLAST en `.xml`.

Habiendo implementado este script se procedió a ejecutarlo, tal como se muestra en la imagen a continuación.

```
11:56:25 - End running pfam scan for PROKKA_00002
11:56:25 - Start running deeploc for PROKKA_00002
11:56:35 - End running deeploc for PROKKA_00002
11:56:35 - End processing PROKKA_00002
11:56:35 - Start processing PROKKA_00003
11:56:35 - Start running Blast for PROKKA_00003
11:58:44 - End running Blast for PROKKA_00003
11:58:45 - Write Blast results for PROKKA_00003 at blast_output/PROKKA_00003.xml
11:58:45 - Start running pfam scan for PROKKA_00003
11:58:45 - End running pfam scan for PROKKA_00003
11:58:45 - Start running deeploc for PROKKA_00003
11:58:53 - End running deeploc for PROKKA_00003
11:58:53 - End processing PROKKA_00003
11:58:53 - Start processing PROKKA_00004
11:58:53 - Start running Blast for PROKKA_00004
Procesando PROKKA_00004:  12%|                                        | 3/26
```

Figura 21: Ejecución de nuestro script.

Además, debido a la dificultad de interpretar los archivos XML devueltos por BLAST realizamos un script que nos permita parsearlo, lo disponibilizamos en el anexo y en nuestro repositorio. Este nos permite extraer la información de los hits de estas salidas y guardarlas en un txt. Junto a esta entrega se adjuntan los archivos de texto plano tal como se solicita en el enunciado.

En suma, como seguía siendo un texto muy largo para llevar a cabo una anotación manual en los tiempos dispuestos, adaptamos el código provisto por la cátedra para utilizarlo en nuestro script y disponer así de una salida más clara, aunque perdiendo información de BLAST que podría ser relevante. Este genera dos salidas, una más extensa y otra en la cuál procesamos los datos aún más para disponer de un resumen para la anotación de los genes predichos.

```
File: blast_parseado.txt

PROKKA_00001.xml
Program: BLASTP 2.16.0+
     db: nr
  Query: Query_8108655 (length=50)
         PROKKA_00001 hypothetical protein
   Hits: No hits found
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
PROKKA_00002.xml
Program: BLASTP 2.16.0+
     db: nr
  Query: Query_698087 (length=319)
         PROKKA_00002 Glutamate decarboxylase alpha
   Hits: ----  -----  ----------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ----------------------------------------------------------
            0      1  gb|AEK55785.1|  glutamate decarboxylase beta [Brucella ...
            1      1  ref|WP_006073452.1|  MULTISPECIES: glutamate decarboxyl...
            2      1  ref|WP_105985131.1|  MULTISPECIES: glutamate decarboxyl...
            3      1  gb|ACU49468.1|  glutamate decarboxylase beta [Brucella ...
            4      1  gb|AEW15625.1|  glutamate decarboxylase [Brucella canis...
            5      1  ref|WP_041594741.1|  glutamate decarboxylase [Brucella ...
            6      1  ref|WP_008509903.1|  glutamate decarboxylase [Brucella ...
            7      1  ref|WP_153528505.1|  MULTISPECIES: glutamate decarboxyl...
            8      1  ref|WP_070998094.1|  glutamate decarboxylase [Brucella ...
            9      1  ref|WP_138144872.1|  glutamate decarboxylase [Brucella ...
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Figura 22: Archivo de texto generado por nuestro script para parsear salidas XML.

```
File: blast_resume.txt

PROKKA_00001.xml
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
PROKKA_00002.xml
    |_glutamate decarboxylase beta [Brucella pinnipedialis B2/94]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [Brucella]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [unclassified Brucella]
    |   |_E-value: 0.0
    |_glutamate decarboxylase beta [Brucella microti CCM 4915]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [Brucella canis HSK A52141]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [Brucella microti]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [Brucella inopinata]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [unclassified Brucella]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [Brucella inopinata]
    |   |_E-value: 0.0
    |_glutamate decarboxylase [Brucella sp. 10RB9215]
    |   |_E-value: 0.0
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
PROKKA_00003.xml
    |_glutamate decarboxylase beta [Brucella canis HSK A52141]
```

Figura 23: Archivo generado como resumen de los resultados de BLAST.

A continuación, se esquematiza el pipeline propuesto para la anotación. En este caso no se contempla el parse de las salidas de BLAST, sólo su almacenamiento como `.xml`.

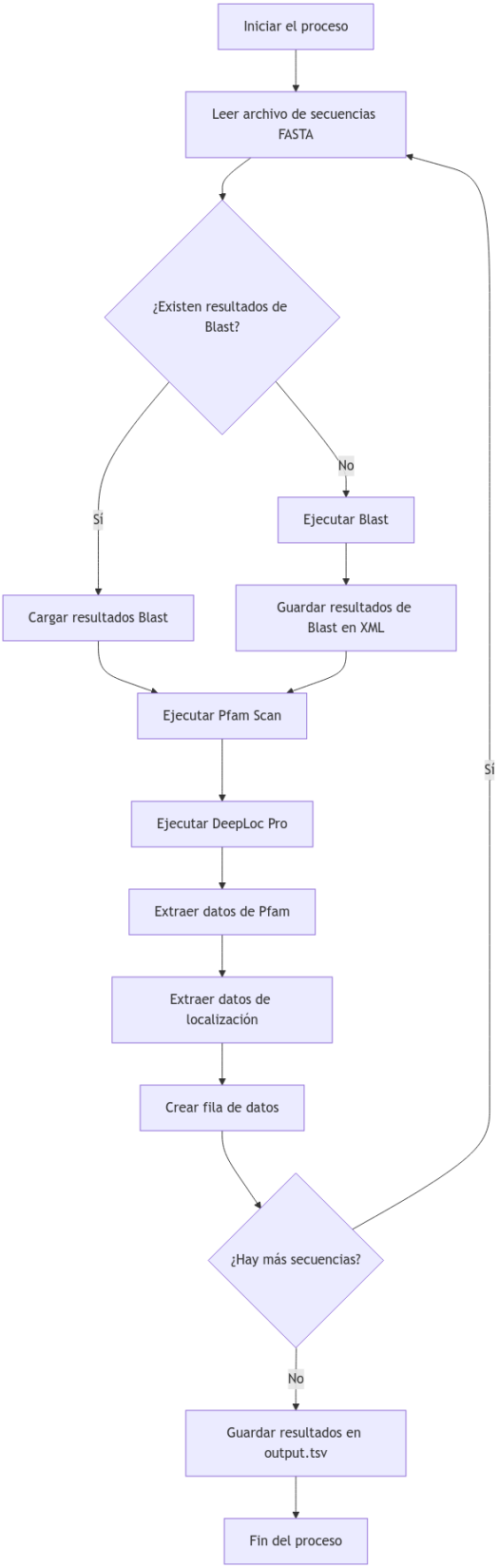


Figura 24: Diagrama de flujo de datos para el script de anotación propuesto.

d) Resumen

Generar una tabla con el nombre de cada gen y la función predicha (o indicios de una posible función).
➡ ¿Cambió la anotación funcional respecto a la obtenida al ejecutar el pipeline?

A continuación se decidió presentar el tsv obtenido en formato de tabla gráfica.

id	Función Prokka	Función Blast	Largo	Pfam acc	Pfam name	Pfam type	Pfam start	Pfam end	Location
PROKKA_00001	hypothetical protein	non result	50						Cytoplasmic
PROKKA_00002	Glutamate decarboxylase alpha	glutamate decarboxylase beta	319	PF00282.24	Pyridoxal_deC	Domain	8	316	Cytoplasmic
PROKKA_00003	Glutamate decarboxylase beta	glutamate decarboxylase beta	136						Cytoplasmic
PROKKA_00004	Glutamate/gamma-aminobutyrate antiporter	Glutamate/gamma-aminobutyrate antiporter	426	PF13520.11	AA_permease_2	Family	67	425	Cytoplasmic Membrane
PROKKA_00005	Glutaminase 1	glutaminase A	317	PF04960.20	Glutaminase	Domain	1	286	Cytoplasmic
PROKKA_00006	Probable acid stress chaperone HdeA precursor	acid-activated periplasmic chaperone HdeA	114	PF06411.16	HdeA	Domain	2	91	Periplasmic
PROKKA_00007	hypothetical protein	hypothetical protein	54						Cytoplasmic Membrane
PROKKA_00008	Uncharacterized membrane protein YjcC	EAL domain-containing protein	530	PF12792.12	CSS-motif	Domain	1	206	Cytoplasmic Membrane
PROKKA_00009	Glycine cleavage system transcriptional activator	LysR family transcriptional regulator	295	PF00126.32	HTH_1	Domain	1	59	Cytoplasmic
PROKKA_00010	Alginate biosynthesis protein AlgA	mannose-1-phosphate guanylyltransferase/mannose-6-phosphate isomerase	471	PF00483.28	NTP_transferase	Family	2	238	Cytoplasmic

PROKKA_00011	Phosphoglucosamine mutase	phosphomannomutase	477	PF02878.21	PGM_PMM_I	Domain	2	133	Cytoplasmic
PROKKA_00012	hypothetical protein	P1 family peptidase	368	PF03576.19	Peptidase_S58	Family	2	300	Periplasmic
PROKKA_00013	H(+)/Cl(-) exchange transporter ClcA	chloride channel protein	451	PF00654.25	Voltage_CLC	Family	2	346	Cytoplasmic Membrane
PROKKA_00014	Inosine-5'-monophosphate dehydrogenase	IMP dehydrogenase	497	PF00478.30	IMPDH	Domain	1	345	Cytoplasmic
PROKKA_00015	SoxAX cytochrome complex subunit A precursor	c-type cytochrome	357	PF21342.2	SoxA-TsdA_cyt-c	Domain	13	87	Cytoplasmic Membrane
PROKKA_00016	Hydrogen peroxide-inducible genes activator	hydrogen peroxide-inducible genes activator	317	PF00126.32	HTH_1	Domain	2	59	Cytoplasmic
PROKKA_00017	Catalase	Catalase	499	PF00199.24	Catalase	Domain	1	382	Periplasmic
PROKKA_00018	hypothetical protein	predicted protein	57						Cytoplasmic Membrane
PROKKA_00019	Disulfide bond formation protein B	disulfide bond formation protein B	208	PF02600.21	DsbB	Family	5	87	Cytoplasmic Membrane
PROKKA_00020	Ribosomal RNA small subunit methyltransferase B	RsmB/NOP family class I SAM-dependent RNA methyltransferase	429	PF22458.1	RsmF-B_ferredox	Domain	2	67	Cytoplasmic
PROKKA_00021	hypothetical protein	phenylacetic acid degradation protein	191	PF03061.27	4HBT	Domain	1	77	Cytoplasmic
PROKKA_00022	5'-methylthioadenosine/S-adenosylhomocysteine nucleosidase	5'-methylthioadenosine/S-adenosylhomocysteine nucleosidase	210	PF01048.25	PNP_UDP_1	Domain	48	207	Cytoplasmic
PROKKA_00023	GMP synthase [glutamine-hyd	glutamine-hydrolyzing GMP	520	PF00117.33	GATase	Domai	1	188	Cytoplasmic

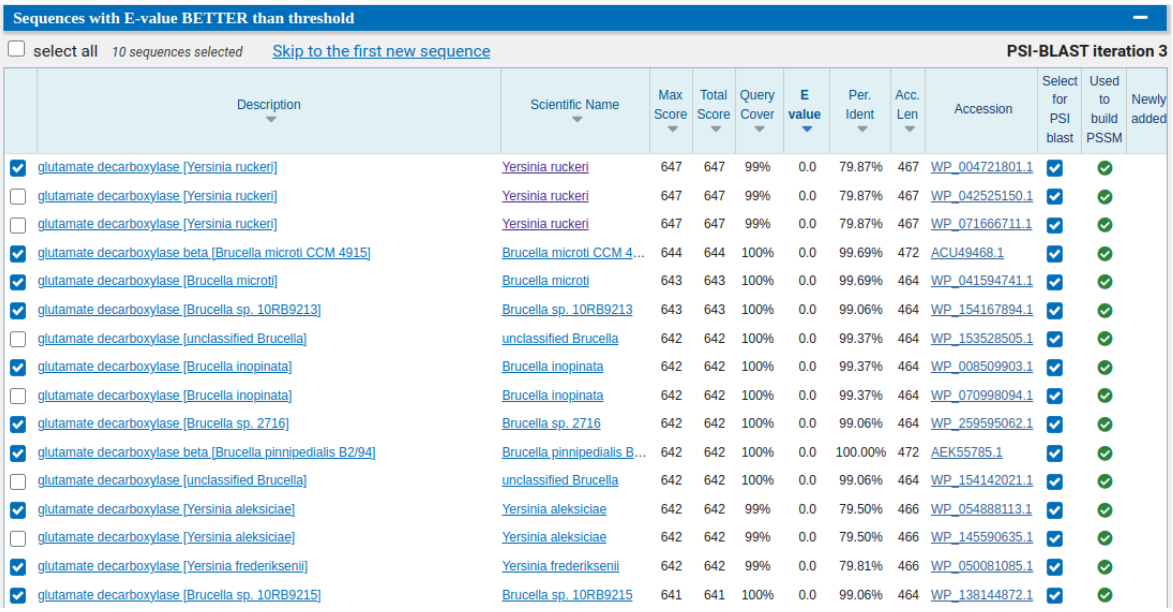
	rolyzing]	synthase				n			
PROKKA_00024	Prophage CP4-57 integrase	hypothetical protein C064_02129	418	PF13356.11	Arm-DNA-bind_3	Domain	1	82	Cytoplasmic
PROKKA_00025	hypothetical protein	AlpA family transcriptional regulator	76	PF05930.17	Phage_AlpA	Family	7	51	Cytoplasmic
PROKKA_00026	hypothetical protein	replication initiator protein A	352	PF10134.14	RPA	Family	1	228	Cytoplasmic

Podemos observar que, a grandes rasgos, las anotaciones obtenidas con Prokka y con BLAST son parecidas, salvo algunos casos como por ejemplo PROKKA_00001 que en Prokka figura como proteína hipotética y, en BLAST, directamente no figuran resultados, o como PROKKA_00025 el cual figura como proteína hipotética en la anotación obtenida con el pipeline pero, con BLAST, tiene asociada la función de ser un regulador transcripcional de la familia AlpA.

5- Estudio de la evolución de *Brucella suis* mediante análisis filogenético

a) PSI-Blast

Seleccionamos la proteína con función predicha **PROKKA_00002** y, luego de 3 iteraciones con 500 secuencias en PSI-BLAST, decidimos tomar especies por fuera de los 10 primeros hits ya que éstos no presentaban gran variedad de especies. Para esto, elegimos otras secuencias que también mostraron buena similitud y query cover, además del género *Brucella*:



Description	Scientific Name	Max Score	Total Score	Query Cover	E value	Per. Ident	Acc. Len	Accession	Select for PSI blast	Used to build PSSM	Newly added
glutamate decarboxylase [Yersinia ruckeri]	Yersinia ruckeri	647	647	99%	0.0	79.87%	467	WP_004721801.1	✓	✓	
glutamate decarboxylase [Yersinia ruckeri]	Yersinia ruckeri	647	647	99%	0.0	79.87%	467	WP_042525150.1	✓	✓	
glutamate decarboxylase [Yersinia ruckeri]	Yersinia ruckeri	647	647	99%	0.0	79.87%	467	WP_071666711.1	✓	✓	
glutamate decarboxylase beta [Brucella microti CCM 4915]	Brucella microti CCM 4...	644	644	100%	0.0	99.69%	472	ACU49468.1	✓	✓	
glutamate decarboxylase [Brucella microti]	Brucella microti	643	643	100%	0.0	99.69%	464	WP_041594741.1	✓	✓	
glutamate decarboxylase [Brucella sp. 10RB9213]	Brucella sp. 10RB9213	643	643	100%	0.0	99.06%	464	WP_154167894.1	✓	✓	
glutamate decarboxylase [unclassified Brucella]	unclassified Brucella	642	642	100%	0.0	99.37%	464	WP_153528505.1	✓	✓	
glutamate decarboxylase [Brucella inopinata]	Brucella inopinata	642	642	100%	0.0	99.37%	464	WP_008509903.1	✓	✓	
glutamate decarboxylase [Brucella inopinata]	Brucella inopinata	642	642	100%	0.0	99.37%	464	WP_070998094.1	✓	✓	
glutamate decarboxylase [Brucella sp. 2716]	Brucella sp. 2716	642	642	100%	0.0	99.06%	464	WP_259595062.1	✓	✓	
glutamate decarboxylase beta [Brucella pinnipedialis B2/94]	Brucella pinnipedialis B...	642	642	100%	0.0	100.00%	472	AEK55785.1	✓	✓	
glutamate decarboxylase [unclassified Brucella]	unclassified Brucella	642	642	100%	0.0	99.06%	464	WP_154142021.1	✓	✓	
glutamate decarboxylase [Yersinia aleksiciae]	Yersinia aleksiciae	642	642	99%	0.0	79.50%	466	WP_054888113.1	✓	✓	
glutamate decarboxylase [Yersinia aleksiciae]	Yersinia aleksiciae	642	642	99%	0.0	79.50%	466	WP_145590635.1	✓	✓	
glutamate decarboxylase [Yersinia frederiksenii]	Yersinia frederiksenii	642	642	99%	0.0	79.81%	466	WP_050081085.1	✓	✓	
glutamate decarboxylase [Brucella sp. 10RB9215]	Brucella sp. 10RB9215	641	641	100%	0.0	99.06%	464	WP_138144872.1	✓	✓	

Figura 25: selección de secuencias de diferentes especies

Descargamos estas 10 secuencias completas en formato FASTA y le concatenamos nuestra proteína que usamos como query.

PSI-BLAST utiliza los resultados de BLASTP para construir una matriz de puntuación de posición específica (PSSM, Position-Specific Scoring Matrix) y, a continuación, localizar secuencias con un parentesco remoto. PSI-BLAST es más sensible que BLASTp y su objetivo es encontrar proteínas distantemente relacionadas, por lo que es mejor para este tipo de análisis.

b) Alineamiento Múltiple

Luego, llevamos a cabo un alineamiento múltiple entre todas estas secuencias (incorporando también la proteína en cuestión), utilizando ClustalX.

Antes de hacer el alineamiento, modificamos el multifasta de forma tal que el identificador de cada secuencia comienza con el nombre del organismo del cual proviene, para una mejor visualización. Para ello desarrollamos un script en Python que realice dicha tarea, disponible en nuestro repositorio de GitHub y en el anexo.

Figura 26: visualización del principio del alineamiento llevado a cabo con ClustalX.

Podemos observar que, arriba del alineamiento, hay una línea que marca las posiciones fuertemente conservadas. Allí, encontramos tres caracteres:

- **‘*’** indica posiciones que tienen un residuo único totalmente conservado.
- **‘:’** indica una conservación fuerte en función de propiedades similares.
- **‘.’** indica una conservación débil (algunos residuos comparten propiedades comunes, pero hay más variabilidad).

En la Figura 26 se observan varias regiones conservadas a lo largo del alineamiento y también patrones divergentes con las secuencias pertenecientes a la especie Yersinia.

Figura 27: visualización del alineamiento llegando a sus posiciones finales.

También, en el resto del alineamiento, encontramos columnas sin ninguno de los símbolos mencionados anteriormente, lo que significa que dichas posiciones son las más variables en el alineamiento.
En la Figura 27 también se puede observar que, a partir de cierta posición, la secuencia correspondiente a nuestra proteína (PROKKA_00002_*Glutamate_decarboxylase_alhpa*) presenta una línea de gaps que se mantiene hasta el final del alineamiento, lo cual puede

reflejar eventos de inserción o deleción evolutiva en esta especie o simplemente un error en la predicción llevada a cabo por el pipeline.

c) Búsqueda de dominios conservados

Ahora, investigamos la proteína elegida para el MSA en InterPro, en búsqueda de dominios conservados. Obtenemos el siguiente resultado.

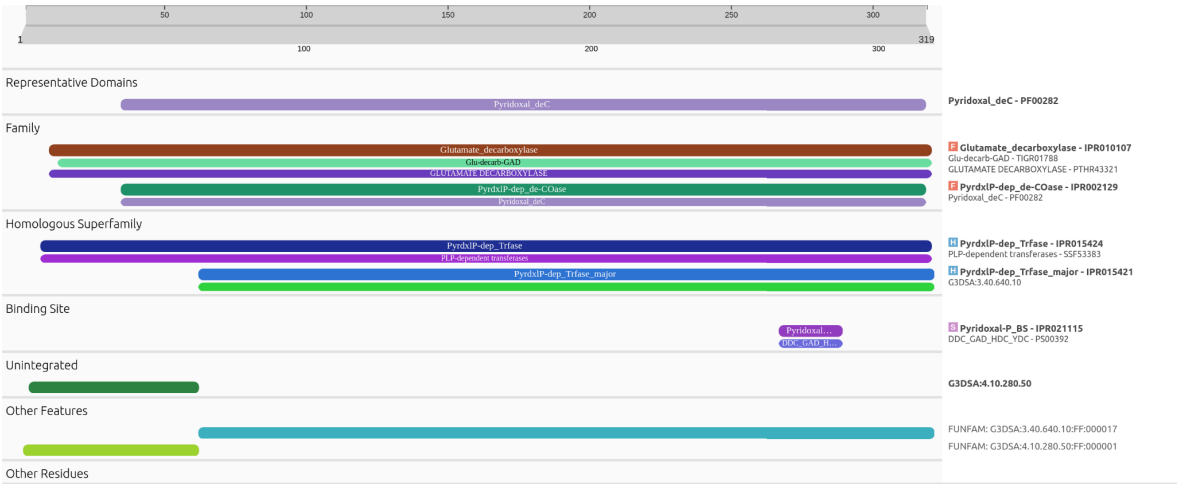


Figura 28: resultados obtenidos de InterPro.

Observamos que hay términos GO (*Gene Ontology*) asociados a la proteína: *Biological Process,* el cual indica en qué procesos celulares participa la proteína, y *Molecular Function,* el cual describe la actividad bioquímica de la proteína (qué tipo de reacción/interacción realiza).

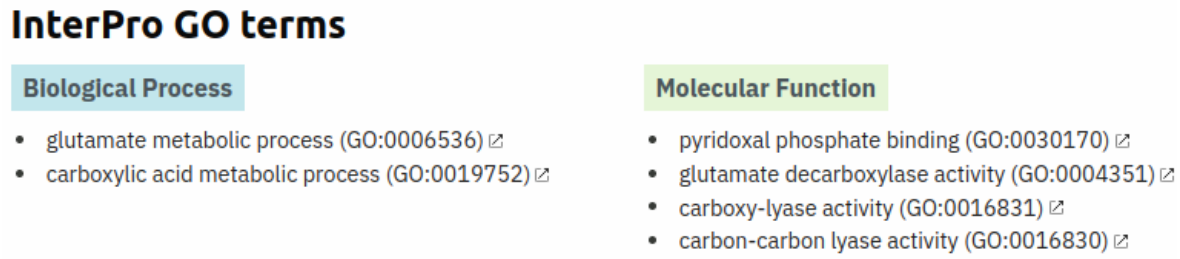


Figura 29: términos GO asociados a la proteína

Vemos también que la proteína tiene un dominio representativo, el cual se encuentra entre las posiciones 37-316.

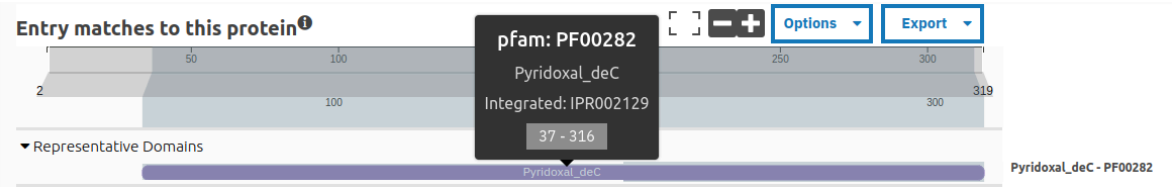


Figura 30: dominio representativo de la proteína.

Al observar la región del dominio representativo de la proteína en el alineamiento múltiple, vemos que corresponde a la región donde se concentran más residuos conservados.
Generamos un logo para representar dicha región utilizando el servidor WebLogo, donde cargamos el MSA generado en ClustalX y en la opción "*Logo Rage"* indicamos el rango de la región conservada que, como en nuestro caso dicha región supera los 30 aa, acotamos el rango a 37-67 para facilitar la visualización.

Figura 31: logo para la región correspondiente al dominio representativo de la proteína.

En este gráfico podemos observar cuán conservados están estos aminoácidos en las diferentes posiciones.

d) Árbol filogenético

Procedemos a construir el árbol filogenético para poder visualizar la distancia evolutiva entre *Brucella suis* y otros organismos, lo cual nos aportará datos cruciales para comprender la evolución de esta bacteria y sus implicancias en la virulencia y la adaptación al hospedador porcino.
Para esto, utilizaremos un árbol basado en distancia, particularmente el método de *Neighbor-joining* (el cual no es enraizado), ya que éste no asume la existencia de un reloj evolutivo, lo cual nos pareció más confiable.
Tuvimos que acortar los nombres del multifasta original, ya que Figtree acorta los nombres a 10 caracteres y, debido a esto, varios nombres nos quedaban iguales. Por lo tanto, la asignación de nombres quedó de la siguiente manera:

Nombre original	Nombre Árbol filogenético
PROKKA_00002_Glutamate_decarboxylase_alpha	P_00002_GI

Yersinia_ruckeri WP_004721801.1 glutamate decarboxylase	Y_ruckeri
Brucella_microti_CCM_4915 ACU49468.1 glutamate decarboxylase beta	B_mic_CCM_
Brucella_microti WP_041594741.1 glutamate decarboxylase	B_mic_WP_0
Brucella_sp._10RB9213 WP_154167894.1 glutamate decarboxylase	B_sp._13
Brucella_inopinata WP_008509903.1 glutamate decarboxylase	B_inopinat
Brucella_sp._2716 WP_259595062.1 glutamate decarboxylase	B_sp._2716
Brucella_pinnipedialis_B2/94 AEK55785.1 glutamate decarboxylase beta	B_pinniped
Yersinia_aleksiciae WP_054888113.1 glutamate decarboxylase	Y_aleksici
Yersinia_frederiksenii WP_050081085.1 glutamate decarboxylase	Y_frederik
Brucella_sp._10RB9215 WP_138144872.1 glutamate decarboxylase	B_sp._15

Procedemos a preparar los archivos, realizando nuevamente el alineamiento múltiple de nuestro multifasta pero ahora utilizando la herramienta Emma del paquete EMBOSS, ya que necesitamos que el alineamiento esté en formato .msa. Utilizamos el siguiente comando:

```
emma -sequence mi_fasta.fasta -dendoutfile mi_fasta.dend -outseq
mi_fasta.msa
```

Esto me crea 2 archivos de formatos .dend y .msa, donde este último, a diferencia del archivo .fasta, contiene gaps.

Luego, procedemos a obtener la matriz de distancias, mediante el comando `fprotdist`, para el cual utilizaremos el algoritmo JTT ya que fue el mismo que utilizamos durante la clase Práctica de Filogenia.

`fprotdist` utiliza uno de cinco algoritmos para calcular las distancias:

- **PAM:** Utiliza una matriz PAM 001. La matriz PAM es una matriz de sustitución obtenida empíricamente. (`-method d`).
- **JTT:** Nombrado por sus creadores, Jones, Taylor y Thornton, se basa en el mismo concepto que el método PAM, solo que la matriz de sustitución fue creada con un set de datos mucho más grande (`-method j`).
- **PBM:** Las matrices de este modelo derivan de la base de datos BLOCKS que contiene secuencias de dominios conservados (`-method h`).
- **Kimura Formula**: Una aproximación precalculada de la matriz PAM, lo que ofrece un cálculo más veloz sacrificando especificidad (`-method k`).
- **Similarity Table**: Una proyección de distancias en la que se asume que los aminoácidos varían según un caso particular de la fórmula de Kimura (`-method s`).

Figura 32: algoritmos para fprotdist.

Construimos la matriz de distancia:

```
fprotdist -method j -sequence mi_fasta.msa -outfile mi_fasta.dist
```

Ahora, con las distancias ya calculadas, procedemos a agrupar:

```
fneighbor -datafile mi_fasta.dist -outfile mi_fasta.tree -outtreefile
mi_fasta-NJ.treefile
```

Obtenemos 2 archivos: mi_fasta.tree (que contiene un árbol filogenético de Neighbor-Joining en formato Newick) y mi_fasta.treefile (el cual contiene información más detallada sobre el árbol).
Visualizamos el árbol con `figtree`:

```
figtree mi_fasta.treefile
```

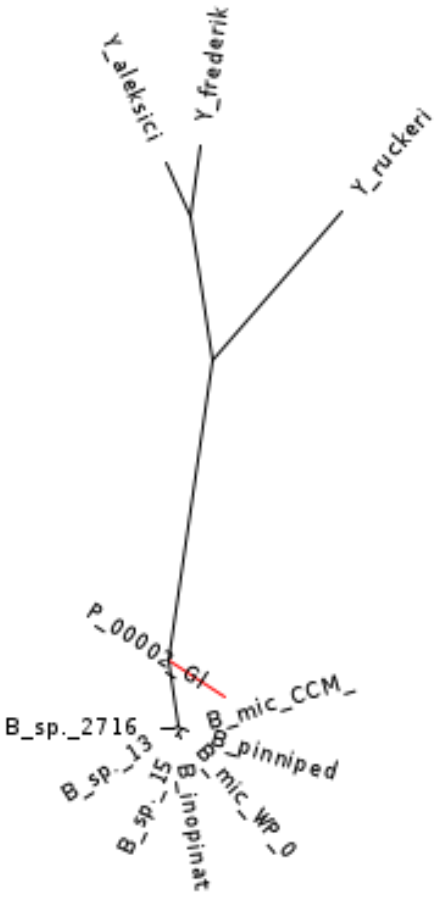


Figura 33: Árbol filogenético

En base a nuestro árbol filogenético, observamos una separación muy clara entre el grupo correspondiente a Yersinia y el grupo de Brucella, lo cual era esperado ya que corresponden a especies distintas. Sin embargo, nuestra proteína predicha por Prokka (para nuestro contig de B. Suis) quedaba igualmente diferenciada del clado de Brucella. Por ello, decidimos volver al MSA y revisarlo.

Como se puede observar en la Figura 27, observamos que nuestra proteína era más corta y estábamos obteniendo una gran cantidad de gaps al final. Tomamos la decisión de filtrar esta región y volver a alinear y construir el árbol. A esto lo hicimos mediante el visualizador de alineamiento Jalview:

```
jalview mi_fasta.msa
```

Habiendo realizado esto, volvimos a repetir los pasos hasta obtener los archivos .tree y .treefile y obtuvimos el árbol que se observa a continuación, en el cuál puede verse la proteína predicha y anotada por PROKKA (P_000002_GI) dentro del clado de las Brucellas.

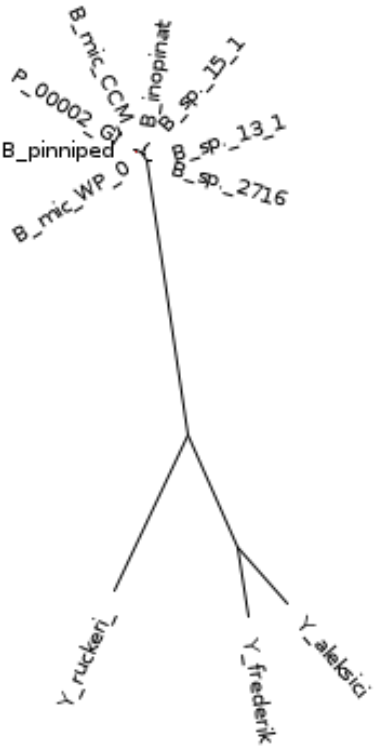


Figura 34: Árbol filogenético en base al MSA filtrado.

Luego, como las ramas del clado de *Brucellas* no se lograban diferenciar bien, decidimos realizar otro árbol mediante el método UPGMA, el cual sí es enraizado, para ver si lograbamos un mejor resultado y visualización.

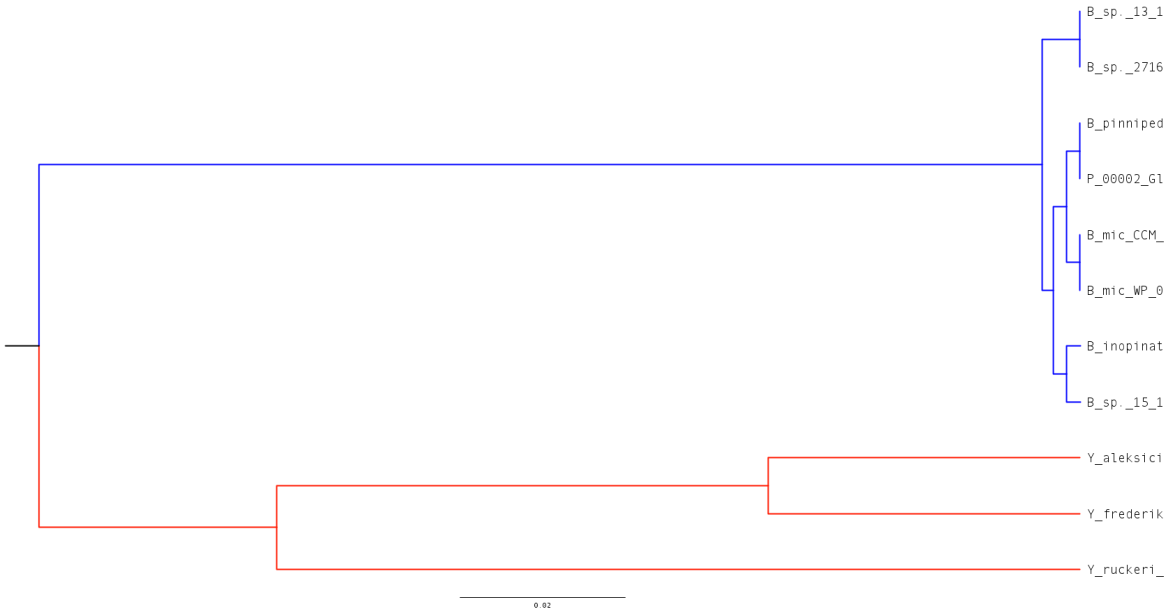


Figura 35: Árbol UPGMA.

Finalmente, ya pudiendo observar con exactitud las ramas en la Figura 35, agregamos al análisis anterior que nuestra proteína de interés P_000002_GI está agrupada con la proteína B_pinniped, lo cual sugiere que comparte una relación evolutiva más cercana con esta secuencia que con las demás.

6- Diseño de *primers* para el desarrollo de un *kit* diagnóstico por PCR

Llegado a este punto, en el enunciado se nos plantea que se busca desarrollar un *kit* diagnóstico basado en PCR para detectar *Brucella suis.* Nuestra tarea es diseñar *primers* específicos para amplificar los genes anotados mediante PCR. Estos *primers* serán utilizados para desarrollar el *kit* diagnóstico.

a) Scripting en Perl

Se nos solicita que desarrollemos un script en **Perl** que realice la tarea del diseño para todos los genes predichos anteriormente. En el enunciado se debían cumplir ciertos requerimientos, vamos a ir analizando los distintos requerimientos y en el anexo adjuntamos el código entero y lo disponibilizamos a través de nuestro repositorio de GitHub.

- ☑ Usar como archivos de entrada (ingresados por consola): la secuencia fasta resultante del ensamblado y el archivo GFF con los genes predichos. Tener en cuenta si el GFF es versión 3 para indicarle al módulo `Bio::Tools::GFF`.

En primer lugar, hacemos las importaciones necesarias para nuestro script.

```
use Bio::Tools::GFF;
use Bio::SeqIO::fasta;
use Bio::Seq;
```

Luego, chequeamos que el script haya sido ejecutado con todos los argumentos necesarios (la ruta al archivo fasta y la ruta al archivo GFF). En caso de no cumplirse se finaliza la ejecución informando la forma de uso correcta del script. Caso contrario, continúa con la ejecución normal almacenando en variables ambos argumentos.

```
if (scalar @ARGV != 2)
{
        die "Uso: perl disenio_primers.pl ruta_fasta ruta_gff\n"
}

my $fasta_route = $ARGV[0];
my $gff_route = $ARGV[1];
```

Como necesitamos tener constancia de la versión del archivo GFF, lo visualizamos como texto plano y vimos que estaba en la primer línea como ##gff-version 3. Por ello procedimos a cargarla y procesarla para separarla y quedarnos con el último elemento.

```
open(GFFFILE, $gff_route) or die "No se pudo abrir el archivo GFF.\n";

my $primer_linea = <GFFFILE>;
chomp($primer_linea);
my @fields = split(' ', $primer_linea);
my $version = @fields[-1];
```

Ya teniendo toda la información necesaria para cargar el archivo GFF y el de secuencia procedimos a realizar dicha acción con BioPerl. Además abrimos un archivo fasta de salida para uno de los requerimientos posteriores.

```
my $gffio = Bio::Tools::GFF->new(-file => $gff_route,
                                 -gff_version =>$version);

my $seqio = Bio::SeqIO->new(-format => 'fasta',
                            -file => $fasta_route);

my $seqout= Bio::SeqIO->new(-format => 'Fasta',
                            -file => '>primers.fa');
```

Llegado a este punto leímos la documentación de BioPerl para Bio::Tools:GFF (*MetaCPAN-Bio::Tools::GFF*, s. f.)

- ☑ Generar los *primers forward* (fw) y *reverse* (rv) de 20 nucleótidos cada uno, para cada gen, usando la secuencia genómica y las coordenadas de cada gen que se encuentran en el archivo GFF. Los *primers* deben "pegar" a 5 nucleótidos del inicio y fin de cada gen para amplificar su secuencia completa. No olvidar tener en cuenta la hebra en la que está codificado cada gen.

```
while($feature = $gffio->next_feature())
{
      my $inicio = $feature->start;
      my $fin = $feature->end;
      my $seqstr;
      my $seq_fw;
      my $seq_rv;
      if ($feature->strand > 0)
      {
          $seqstr = $seq->trunc($inicio-5, $fin+5)->seq;
          $seq_fw = $seq->trunc($inicio-5, $inicio+14)->seq;
          $seq_fw =~ tr/ACGT/TGCA/;
          $seq_rv = $seq->trunc($fin-14, $fin+5)->seq;
          $seq_rv = reverse $seq_rv;
      }
      else
```

```
        {
            $seqstr = $seq->trunc($inicio-5, $fin+5)->seq;
            $seqstr =~ tr/ACGT/TGCA/;
            $seqstr = reverse $seqstr;
            $seq_fw = $seq->trunc($fin-15, $fin+4)->seq;
            $seq_fw = reverse $seq_fw;
            $seq_rv = $seq->trunc($inicio-5, $inicio+14)->seq;
            $seq_rv =~ tr/ACGT/TGCA/;
        }
}
```

Dentro de nuestro ciclo para cada feature almacenada en el GFF fuimos extrayendo de la secuencia de entrada, donde codificamos un condicional para extraer los primers de la secuencia y transformarlos según la hebra en la cuál se codifica el gen.

- ☑ Generar como salida un archivo separado por tabulaciones que tenga la siguiente información en cada columna: Nombre del gen, secuencia *primer* fw, Tm fw, Ta fw, secuencia *primer* rv, Tm rv, Ta rv, Tamaño producto.
    - Donde: Tm es la temperatura de Melting calculada con la fórmula:
    $$Tm = 4(G + C) + 2(A + T)$$
    - Ta es la temperatura de Annealing
    $$Ta = Tm - 5$$
    - Tamaño producto es la longitud en pb de la secuencia amplificada al usar cada par de *primers*.

Para cumplir con este requerimiento lo primero que identificamos como importante es la necesidad de conocer el número de ocurrencias de cada base. Esto lo llevamos a cabo con el siguiente código:

```
my @bases = ('A', 'C', 'G', 'T');
my %conteos_fw;
my %conteos_rv;
foreach my $base (@bases){
      my $base_min = lc($base);
      $conteos_fw{$base} = length( $seq_fw =~ s/[^\Q$base\E]//rg );
      $conteos_rv{$base} = length( $seq_rv =~ s/[^\Q$base\E]//rg );
}
```

Quisimos realizar este procedimiento con los contenidos de la práctica llevada a cabo en clase pero nos encontramos con que el patrón regular nos estaba devolviendo un arreglo y no podíamos manejarlo dentro del ciclo, con la base variable, como queríamos. Por ello buscamos en StackOverflow y encontramos una respuesta que daba solución a nuestro problema (ikegami, 2015).

```
my $ruta_salida = 'salida.tsv';
```

```
open(SALIDA, '>', $ruta_salida) or die "No se pudo abrir el archivo de salida\n";
print SALIDA
"nombre_gen\tsecuencia_fw\tTm_fw\tTa_fw\tsecuencia_rv\tTm_rv\tTa_rv\ttamanio_producto\n"
;

while($feature = $gffio->next_feature())
{
  my @ids = $feature->get_tag_values('ID');
  my $gen = $ids[0];

  my $Tm_fw = 4 * ($conteos_fw{'G'} + $conteos_fw{'C'})
              + 2 * ($conteos_fw{'A'} + $conteos_fw{'T'});

  my $Ta_fw = $Tm_fw - 5;

  my $Tm_rv = 4 * ($conteos_rv{'G'} + $conteos_rv{'C'})
              + 2 * ($conteos_rv{'A'} + $conteos_rv{'T'});

my $Ta_rv = $Tm_rv - 5;

my $tamanio = $feature->length;
```

Finalmente, guardamos en el archivo de salida Tsv que definimos los datos separados por tabulaciones.

```
print SALIDA $gen . "\t" . $seq_fw . "\t" . $Tm_fw . "\t" . $Ta_fw .
"\t" . $seq_rv . "\t" . $Tm_rv . "\t" . $Ta_rv . "\t" . $tamanio . "\n";
```

- ☑ Generar un archivo de salida multifasta que contenga las secuencias de los productos de PCR (uno por cada gen) y que el identificador incluya el nombre del gen y el tamaño del producto amplificado.

Generamos un flujo de salida en formato fasta, luego para cada característica en el GFF fuimos generando el header con la información establecida por el requerimiento y escribimos la secuencia en el archivo de salida.

```
my $seqout= Bio::SeqIO->new( -format => 'Fasta',
                             -file => '>primers.fa');

while($feature = $gffio->next_feature())
{
  my $header = $gen . "-" . $tamanio . "bp";
  my $new_seq = Bio::Seq->new(-seq => $seqstr, -id => $header);

  $seqout->write_seq($new_seq)
}

$seqout->close();
```

Habiendo corrido este script obtuvimos la salida en formato TSV y el multifasta de los amplicones, tal como se solicitaba en la consigna. Luego,pasamos a hacer la elección de un gen para el kit de diagnóstico. Para ello, realizamos una búsqueda bibliográfica de la aplicación de PCR para la detección de Brucella Suis, donde vimos que en la mayoría de ellos se utilizaba PCR múltiple, para amplificar distintos productos y así detectarla (Bricker & Halling, 1994; Fretin et al., 2008; López-Goñi et al., 2011).

La complejidad de ello, escapaba a los objetivos de este trabajo ya que se busca un único gen. Por ello es que continuamos con la búsqueda y decidimos basarnos en un paper que propone una herramienta para la obtención de primers para qPCR (Thornton & Basu, 2011). En él se propone que un primer óptimo debe tener Tms entre 59 y 68, y amplificar un producto de entre 80 y 150 pares de bases. En suma, medimos el porcentaje de GC que era un parámetro no contemplado en el script y que es de relevancia en el diseño de primers, establecimos como rango aceptable entre 40 y 60.

Todas estas reglas condicionales las aplicamos en una hoja de cálculo[1] obteniendo la visualización que se muestra en la tabla siguiente, para una mejor presentación se sustituyó el ID de Prokka por los últimos dos dígitos y no se incluyó la secuencia de ambos primer, sólo sus atributos

id	%GC_fw	Tm_fw	%GC_rv	Tm_rv	tamanio_producto
01	50	60	70	68	163
02	30	52	65	66	970
03	55	62	35	54	421
04	75	70	60	64	1291
05	35	54	45	58	964
06	40	56	30	52	355
07	15	46	40	56	175
08	40	56	35	54	1603
09	35	54	60	64	898
10	55	62	45	58	1426
11	55	62	60	64	1444
12	50	60	50	60	1117
13	45	58	50	60	1366
14	45	58	60	64	1504
15	45	58	45	58	1084
16	40	56	25	50	964
17	55	62	40	56	1510
18	50	60	55	62	184

[1] Primers

19	30	52	55	62	637
20	60	64	45	58	1300
21	55	62	45	58	586
22	45	58	60	64	643
23	60	64	45	58	1573
24	45	58	65	66	1267
25	35	54	55	62	241
26	55	62	55	62	1069

Cuando tenemos en cuenta todas nuestras reglas consideradas, el candidato más viable es el gen PROKKA_00018, excediendo el límite pero estando dentro de los rangos que son considerados aceptables por el artículo referenciado.

Bibliografía

Bio::Tools::GFF. (s. f.). MetaCPAN. Recuperado 28 de octubre de 2024, de https://metacpan.org/pod/Bio::Tools::GFF

Bricker, B. J., & Halling, S. M. (1994). Differentiation of Brucella abortus bv. 1, 2, and 4, Brucella melitensis, Brucella ovis, and Brucella suis bv. 1 by PCR. *Journal of Clinical Microbiology*, *32*(11), 2660-2666. https://doi.org/10.1128/jcm.32.11.2660-2666.1994

Brucella suis 1330 genome assembly ASM750v1. (s. f.). NCBI. Recuperado 15 de octubre de 2024, de https://www.ncbi.nlm.nih.gov/data-hub/assembly/GCF_000007505.1/

Brucelosis. (s. f.-a). Recuperado 15 de octubre de 2024, de https://www.who.int/es/news-room/fact-sheets/detail/brucellosis

Brucelosis: MedlinePlus enciclopedia médica. (s. f.-b). Recuperado 15 de octubre de 2024, de https://medlineplus.gov/spanish/ency/article/000597.htm

Delcher, A. L. (2006). *Glimmer Release Notes* (Versión 3.02) [Software]. University of Maryland Center for Bioinformatics & Computational Biology.

Fretin, D., Whatmore, A. M., Al Dahouk, S., Neubauer, H., Garin-Bastuji, B., Albert, D., Van Hessche, M., Ménart, M., Godfroid, J., Walravens, K., & Wattiau, P. (2008). *Brucella suis* identification and biovar typing by real-time PCR. *Veterinary Microbiology*, *131*(3), 376-385. https://doi.org/10.1016/j.vetmic.2008.04.003

ikegami. (2015, diciembre 23). *Answer to «Is there a better way to count occurrence of char in a string?»* [Post]. Stack Overflow. https://stackoverflow.com/a/34442251

Jaimomar99. (2024). *Jaimomar99/deeplocpro* [Python]. https://github.com/Jaimomar99/deeplocpro (Obra original publicada en 2023)

Justog220/TIF-AyA: Trabajo integrador final para Análisis y Alineamiento de Secuencias. (s. f.). Recuperado 28 de octubre de 2024, de https://github.com/justog220/TIF-AyA

López-Goñi, I., García-Yoldi, D., Marín, C. M., de Miguel, M. J., Barquero-Calvo, E., Guzmán-Verri, C., Albert, D., & Garin-Bastuji, B. (2011). New Bruce-ladder multiplex PCR assay for the biovar typing of *Brucella suis* and the discrimination of *Brucella suis* and *Brucella canis*. *Veterinary Microbiology*, *154*(1), 152-155.

https://doi.org/10.1016/j.vetmic.2011.06.035

Moreno, J., Nielsen, H., Winther, O., & Teufel, F. (2024). *Predicting the subcellular location of prokaryotic proteins with DeepLocPro*. https://doi.org/10.1101/2024.01.04.574157

Schoch, C. L., Ciufo, S., Domrachev, M., Hotton, C. L., Kannan, S., Khovanskaya, R., Leipe, D., Mcveigh, R., O'Neill, K., Robbertse, B., Sharma, S., Soussov, V., Sullivan, J. P., Sun, L., Turner, S., & Karsch-Mizrachi, I. (2020). NCBI Taxonomy: A comprehensive update on curation, resources and tools. *Database*, *2020*, baaa062. https://doi.org/10.1093/database/baaa062

Šmarda, P., Bureš, P., Horová, L., Leitch, I. J., Mucina, L., Pacini, E., Tichý, L., Grulich, V., & Rotreklová, O. (2014). Ecological and evolutionary significance of genomic GC content diversity in monocots. *Proceedings of the National Academy of Sciences*, *111*(39), E4096-E4102. https://doi.org/10.1073/pnas.1321152111

Thornton, B., & Basu, C. (2011). Real-time PCR (qPCR) primer design using free online software. *Biochemistry and Molecular Biology Education*, *39*(2), 145-154. https://doi.org/10.1002/bmb.20461

Thumuluri, V., Almagro Armenteros, J. J., Johansen, A. R., Nielsen, H., & Winther, O. (2022). DeepLoc 2.0: Multi-label subcellular localization prediction using protein language models. *Nucleic Acids Research*, *50*(W1), W228-W234. https://doi.org/10.1093/nar/gkac278

Zielezinski, A. (2024). *Aziele/pfam_scan* [Python]. https://github.com/aziele/pfam_scan (Obra original publicada en 2022)

Anexo

Script de anotación

```
from Bio import Blast
from Bio import SeqIO
from io import StringIO
from tqdm import tqdm
import pandas as pd
import subprocess
import tempfile
import os
from datetime import datetime

Blast.email = os.getenv("BLAST_EMAIL")

def run_deeplocpro(fasta_record):

    with tempfile.NamedTemporaryFile(delete=False, mode="w") as
temp_fasta:
        SeqIO.write(fasta_record, temp_fasta, "fasta")
        temp_fasta = temp_fasta.name


    script_dir = os.path.dirname(os.path.abspath(__file__))

    with tempfile.TemporaryDirectory() as temp_dir:
        output_path = temp_dir

        cmd = f"deeplocpro -f {temp_fasta} -o {temp_dir}"
        subprocess.run(cmd,
                       capture_output=True,
                       text=False,
                       shell=True)

        temp_file = subprocess.check_output(f"ls -t {temp_dir} | head -n
1", shell=True).decode('utf-8').strip()
        with open(temp_dir + "/" + temp_file, "r") as oarchi:
            output = oarchi.read()

    return parse_deeplocpro(output)

def parse_deeplocpro(deeploc_outp):
    output_io = StringIO(deeploc_outp)
```

```
    df = pd.read_csv(output_io)

    return df[["ACC", "Localization"]]

def run_pfam_scan(fasta_record):

    with tempfile.NamedTemporaryFile(delete=False, mode="w") as
temp_fasta:
        SeqIO.write(fasta_record, temp_fasta, "fasta")
        temp_fasta_path = temp_fasta.name


        # Obtener la ruta del directorio donde está este script
(anotacion.py)
    script_dir = os.path.dirname(os.path.abspath(__file__))


    pfam_scan_path = os.path.join(script_dir, "pfam_scan",
"pfam_scan.py")
    pfam_db_path = os.path.join(script_dir, "pfam_scan", "pfamdb")

    cmd = f"python {pfam_scan_path} {temp_fasta_path} {pfam_db_path}"
    result = subprocess.run(cmd,
                    capture_output=True,
                    text=True,
                    shell=True)

    output = result.stdout

    output_io = StringIO(output)

    df = pd.read_csv(output_io)

    columnas_interes = [
        "hmm_acc",
        "hmm_name",
        "type",
        "hmm_start",
        "hmm_end"
    ]

    rename_columnas_interes = [
        "pfam_acc",
        "pfam_name",
        "pfam_type",
```

```
        "pfam_start",
        "pfam_end"
    ]

    rename_dict = dict(zip(columnas_interes, rename_columnas_interes))

    df.rename(columns=rename_dict, inplace=True)

    columnas_interes = rename_columnas_interes
    del(rename_columnas_interes)

    return df[columnas_interes]

def current_time():
    return datetime.now().strftime("%H:%M:%S")

df = pd.DataFrame({
    "id": [],
    "funcion_prokka": [],
    "largo": pd.Series(dtype="int"),
    "pfam_acc": [],
    "pfam_name": [],
    "pfam_type": [],
    "pfam_start": pd.Series(dtype="int"),
    "pfam_end": pd.Series(dtype="int"),
#    "interpro_scan": [],
    "location": [],
})

dbs = ["nr", "swissprot", "refseq_protein"]

fasta_file = "../data/prokka/PROKKA_10252024/PROKKA_10252024.faa"

blast_output = "blast_output"
os.makedirs(blast_output, exist_ok=True)

records = list(SeqIO.parse(fasta_file, "fasta"))
bar_format = "{l_bar}{bar}| {n_fmt}/{total_fmt}"

with tqdm(records, desc="Procesando secuencias", bar_format=bar_format,
dynamic_ncols=True) as pbar:
    for record in pbar:
        id = record.id
        pbar.set_description(f"Procesando {id}")
        tqdm.write(f"{current_time()} - Start processing {id}")
```

```
        nombre = record.description
        nombre = " ".join(nombre.split(sep=" ")[1:])

        largo = int(len(record.seq))

        ruta_salida_blast = f"{blast_output}/{record.id}.xml"

        if id + ".xml" not in os.listdir(blast_output):
            tqdm.write(f"{current_time()} - Start running Blast for
{id}")
            result_stream = Blast.qblast("blastp", "nr", format(record,
"fasta"), hitlist_size=10)
            tqdm.write(f"{current_time()} - End running Blast for {id}")

            with open(ruta_salida_blast, "wb") as out_stream:
                out_stream.write(result_stream.read())

            tqdm.write(f"{current_time()} - Write Blast results for {id}
at {ruta_salida_blast}")

            result_stream.close()

        else:
            tqdm.write(f"{current_time()} - Blast results for {id}
already available at {ruta_salida_blast}")


        tqdm.write(f"{current_time()} - Start running pfam scan for
{id}")
        pfam_info = run_pfam_scan(record)
        tqdm.write(f"{current_time()} - End running pfam scan for {id}")

        tqdm.write(f"{current_time()} - Start running deeploc for {id}")
        location_info = run_deeplocpro(record)
        tqdm.write(f"{current_time()} - End running deeploc for {id}")

        new_row = {
            "id": id,
            "funcion_prokka": nombre,
            "largo": largo,
            "pfam_acc": pfam_info["pfam_acc"].iloc[0] if not
pfam_info.empty else None,
            "pfam_name": pfam_info["pfam_name"].iloc[0] if not
pfam_info.empty else None,
            "pfam_type": pfam_info["pfam_type"].iloc[0] if not
pfam_info.empty else None,
```

```
            "pfam_start": pfam_info["pfam_start"].iloc[0] if not
pfam_info.empty else None,
            "pfam_end": pfam_info["pfam_end"].iloc[0] if not
pfam_info.empty else None,
            "location": location_info["Localization"].iloc[0]
        }

        df = df._append(new_row, ignore_index=True)

        tqdm.write(f"{current_time()} - End processing {id}")



df.to_csv("output.tsv", sep="\t", index=False)
```

Script para la refactorización de archivos FASTA de PSI-BLAST

```
import sys

def parse_fasta(input_file):
    output_file = input_file.replace(".", "_parseado.")

    print(output_file)
    with open(input_file, "r") as infile, open(output_file, "w") as
outfile:
        for line in infile:
            if line.startswith(">"):

                especie = line.split("[")[-1].split("]")[0]

                especie = especie.replace(" ", "_")

                line = line.split("[")[0]

                nuevo_header = f">{especie} {line[1:]}\n"
                outfile.write(nuevo_header)
            else:
                outfile.write(line)
    print(f"Archivo procesado guardado como: {output_file}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Uso: python script.py <archivo_multifasta.fasta>")
    else:
        parse_fasta(sys.argv[1])
```

Script para el diseño de primers

```
#/usr/bin/perl -w
use strict;
use Bio::Tools::GFF;
use Bio::SeqIO::fasta;
use Bio::Seq;


#Manejo argumentos
if (scalar @ARGV != 2)
{
      die "Uso: perl disenio_primers.pl ruta_fasta ruta_gff\n"
}

my $fasta_route = $ARGV[0];
my $gff_route = $ARGV[1];

#-----Leo version GFF----
open(GFFFILE, $gff_route) or die "No se pudo abrir el archivo GFF.\n";

my $primer_linea = <GFFFILE>;
chomp($primer_linea);
my @fields = split(' ', $primer_linea);
my $version = @fields[-1];
#------------------------

my $ruta_salida = 'salida.tsv';
open(SALIDA, '>', $ruta_salida) or die "No se pudo abrir el archivo de
salida\n";
print SALIDA
"nombre_gen\tsecuencia_fw\tTm_fw\tTa_fw\tsecuencia_rv\tTm_rv\tTa_rv\ttam
anio_producto\n";

my $gffio = Bio::Tools::GFF->new(-file => $gff_route, -gff_version
=>$version);
my $seqio = Bio::SeqIO->new(-format => 'fasta', -file => $fasta_route);
my $seqout= Bio::SeqIO->new( -format => 'Fasta', -file =>
'>productos_pcr.fa');
my $seq = $seqio->next_seq();

my $feature;
my @bases = ('A', 'C', 'G', 'T');

use Data::Dumper;

while($feature = $gffio->next_feature())
```

```
{
      my @ids = $feature->get_tag_values('ID');
      my $gen = $ids[0];
      # print "Feature for ",$feature->get_tag_values('ID')," starts
",$feature->start," ends ", $feature->end," strand ",$feature->strand,"
length ",$feature->length,"\n";
      my $inicio = $feature->start;
      my $fin = $feature->end;
      my $seqstr;
      my $seq_fw;
      my $seq_rv;
      if ($feature->strand > 0)
      {
      $seqstr = $seq->trunc($inicio-5, $fin+5)->seq;
      $seq_fw = $seq->trunc($inicio-5, $inicio+14)->seq;
      $seq_fw =~ tr/ACGT/TGCA/;
      $seq_rv = $seq->trunc($fin-14, $fin+5)->seq;
      $seq_rv = reverse $seq_rv;
      }
      else
      {
      $seqstr = $seq->trunc($inicio-5, $fin+5)->seq;
      $seqstr =~ tr/ACGT/TGCA/;
      $seqstr = reverse $seqstr;
      $seq_fw = $seq->trunc($fin-15, $fin+4)->seq;
      $seq_fw = reverse $seq_fw;
      $seq_rv = $seq->trunc($inicio-5, $inicio+14)->seq;
      $seq_rv =~ tr/ACGT/TGCA/;
      }

      my %conteos_fw;
      my %conteos_rv;
      foreach my $base (@bases){
      my $base_min = lc($base);
      $conteos_fw{$base} = length( $seq_fw =~ s/[^\Q$base\E]//rg ); #
Teníamos problemas con como lo habiamos visto en clase y lo solucionamos
con:
https://stackoverflow.com/questions/34437248/is-there-a-better-way-to-co
unt-occurrence-of-char-in-a-string
      $conteos_rv{$base} = length( $seq_rv =~ s/[^\Q$base\E]//rg );
      }

      # print Dumper(\%conteos_fw);
      # print Dumper(\%conteos_rv);

      my $Tm_fw = 4 * ($conteos_fw{'G'} + $conteos_fw{'C'}) + 2 *
```

```
($conteos_fw{'A'} + $conteos_fw{'T'});
      my $Ta_fw = $Tm_fw - 5;
      my $Tm_rv = 4 * ($conteos_rv{'G'} + $conteos_rv{'C'}) + 2 *
($conteos_rv{'A'} + $conteos_rv{'T'});
      my $Ta_rv = $Tm_rv - 5;

      my $tamanio = $feature->length + 10;
      print SALIDA $gen . "\t" . $seq_fw . "\t" . $Tm_fw . "\t" . $Ta_fw
. "\t" . $seq_rv . "\t" . $Tm_rv . "\t" . $Ta_rv . "\t" . $tamanio .
"\n";

      my $header = $gen . "-" . $tamanio . "bp";

      my $new_seq = Bio::Seq->new(-seq => $seqstr,
                                  -id => $header);

      $seqout->write_seq($new_seq)
}

$gffio->close();
$seqio->close();
$seqout->close();
close(SALIDA);
```