Workflow para asociar segmentos de una secuencia genómica a una familia de proteínas de interés.

Fátima Marín Nieto
Máster Universitario en Bioinformática y Bioestadística UOC-UB
Área1_Estadística y Bioinformática

Esteban Vegas Lozano
Alex Sánchez Pla

02 de enero de 2016

FICHA DEL TRABAJO FINAL

Título del trabajo:	Workflow para asociar segmentos de una secuencia genómica a una familia de proteínas de interés.
Nombre del autor:	Fátima Marín Nieto
Nombre del consultor/a:	Esteban Vegas Lozano
Nombre del PRA:	Alex Sánchez Pla
Fecha de entrega (mm/aaaa):	01/2017
Titulación:	Máster Universitario en Bioinformática y Bioestadística UOC-UB
Área del Trabajo Final:	Área1_Estadística y Bioinformática
Idioma del trabajo:	Castellano
Palabras clave:	workflow, homología, CAZy

Resumen del Trabajo (máximo 250 palabras):

CAZy (*Carbohydrate-Active Enzyme*) es una base de datos especializada en las enzimas que forman y degradan carbohidratos complejos y gliconconjugados. La identificación de estas enzimas, a partir de la secuencia genómica, continúa crecido a lo largo de los años permitiendo un mayor conocimiento de cómo las especies utilizan este tipo de compuestos y cómo los metabolizan. En este contexto, el objetivo del presente trabajo ha sido crear un *workflow* que permita seleccionar segmentos de un genoma microbiano que sean posibles candidatos de pertenecer a una familia funcional de proteínas descrita en CAZy. El *pipeline* para llevar a cabo este objetivo se ha diseñado en 5 fases principales: preparación de la base de datos CAZy, recuperación de secuencias fasta de proteínas de una familia funcional, análisis de homología por secuencia mediante alineamientos a pares y múltiples y búsqueda por patrón funcional. Para la realización del trabajo se han utilizado múltiples herramientas específicas para cada fase de análisis, como BLAST y HMMR3, y se han establecido los criterios óptimos que permitieran identificar las regiones de similitud en la secuencia genómica microbiana. El diseño del *pipeline*, y su posterior automatización, se ha desarrollado en una máquina virtual donde se instalaron localmente todos los programas necesarios y que trabaja con un sistema operativo Linux. Como producto final se ha obtenido el paquete de R cazypredict y una aplicación en Shiny diseñadas para ejecutarse en la máquina virtual que se creó específicamente para este fin.

Abstract (in English, 250 words or less):

CAZy (Carbohydrate-Active Enzyme) is a database focused on enzymes that build and breakdown complex carbohydrates and glycoconjugates. The identification of this enzymes, from genomic sequence, have been increasing in the past years contributing to a better knowledge of how species use and metabolize these compounds. In this context, the main aim of the present work is to create a workflow for selecting regions in bacterial genome sequences that could be candidates to belong to a functional protein family already described in CAZy. The pipeline designed to achieve this aim has been divided into 5 phases: setting up the CAZy database, retrieving fasta sequences of proteins belonging to the same functional family, performing sequence homology analysis by pairwise and multiple alignments and searching for functional patterns. In the workflow elaboration, several tools where used in each analysis step, like BLAST or HMMR3, and suitable criteria were stablished to accomplish the identification of the sequences with similarity to CAZy proteins in bacterial genomic sequence. The pipeline design, and subsequent automatization, have been developed on a virtual machine running in Linux OS, where it has been installed locally the required software The final products obtained were the R package cazypredict and a Shiny app designed to be executed on the virtual machine created specifically for that purpose.

Índice

Lista de figuras

Lista de tablas

1 Introducción

1.1 Contexto y justificación del trabajo

CAZy (*Carbohydrate-Active Enzyme*, http://www.cazy.org/) es una base de datos especializada en las enzimas que forman y degradan carbohidratos complejos y gliconconjugados (4; 9). Este tipo de enzimas están presentes en multitud de organismos y se encuentra en gran abundancia y diversidad en las bacterias. Las bacterias utilizan estas enzimas, por ejemplo, para degradar y metabolizar carbohidratos complejos de su entorno como fuente de energía, actuando en multitud de sustratos presentes en la naturaleza, y siendo, además, muy distintos estructuralmente. Por esta razón, la identificación de estas enzimas, a partir de la secuencia genómica, continúa crecido a lo largo de los años permitiendo un mayor conocimiento de cómo las especies utilizan este tipo de compuestos y cómo los metabolizan. Las enzimas en CAZy se clasifican en familias relacionadas estructuralmente y comparten similitud en su secuencia aminoacídica.

En este contexto, el trabajo tiene como objetivo poder relacionar segmentos de un genoma microbiano con una determinada familia funcional descrita en la base de datos CAZy. Para ello, se realizarán diversos análisis comparativos entre la secuencia genómica sin anotar y las secuencias proteicas de referencia que pertenezcan a una misma familia funcional. La finalidad de la comparativa será obtener descriptores de calidad que permitan identificar posibles candidatos a pertenecer a una familia concreta de interés. Los análisis a realizar se engloban en el área de la bioinformática, especialmente aplicada a la anotación de secuencias genómicas, como puede ser la homología por similitud entre secuencia o la búsqueda de patrones funcionales. Como resultado del trabajo se espera obtener un *workflow* donde se implemente todos los análisis y que genere una salida suficientemente comprensible, incluyendo varios criterios de bondad de asociación, para que el usuario final pueda identificar las regiones genómicas relacionadas con una familia CAZy.

1.2 Objetivos del Trabajo

El objetivo principal de este trabajo es el siguiente:

> **Crear un *workflow* que permita seleccionar segmentos de un genoma microbiano que sean posibles candidatos de pertenecer a una familia funcional de proteínas descrita en CAZy.**

Objetivos específicos:

Para conseguir el objetivo principal se plantean los siguientes objetivos específicos:

1- Seleccionar el tipo de análisis, herramientas y criterios que se utilizarán para identificar las regiones candidatas.

2- Diseñar el *pipeline* donde se vayan incorporando los análisis anteriormente seleccionados.

3- Automatizar el *pipeline* en un *workflow* para que cada grupo de análisis genere una salida con la información necesaria que permita la identificación de la región candidata.

4- Mejorar la visualización de los resultados mediante la incorporación del *workflow* en una aplicación web o similar.

1.3 Enfoque y método seguido

El principal objetivo de este trabajo es crear un *workflow* para asociar segmentos de la secuencia de nucleótidos sin anotar con la familia funcional de interés previamente seleccionada. Para llevar a cabo este objetivo se deben utilizar herramientas específicas para comparar estos dos tipos de secuencias y establecer los análisis y criterios que permitan identificar regiones de similitud en la secuencia genómica. Existen diferentes aproximaciones y múltiples herramientas para llevar a cabo este objetivo, por lo que el diseño final del *pipeline* y la creación del *worklfow* ha dependido en gran parte de la disponibilidad y facilidad de automatización de dichas herramientas, teniendo en cuenta que los resultados obtenidos en cada fase del análisis debe generar información adicional que proporcione al usuario los parámetros necesarios para reconocer e identificar la regiones candidatas.

El *pipeline* se ha diseñado en base a las siguientes etapas:

1) Preparación de los datos

La familia funcional microbiana se obtendrá de la base de datos CAZy (http://www.cazy.org/), especializada en enzimas con actividad de carbohidrato (9). Además, el *workflow* necesitará como datos de entrada una secuencia genómica de DNA sin anotaciones.

2) Análisis de homología por secuencia, alineamiento por pares

El primer análisis incluido en el *pipeline* ha sido la búsqueda de homología entre los dos tipos de secuencias mediante alineamientos locales a pares y utilizando un algoritmo como el BLAST (https://blast.ncbi.nlm.nih.gov/Blast.cgi) (1). Las herramientas incluidas en BLAST asocian diversos tipos de estadísticos a los alineamientos resultantes por lo que se fijarán criterios de filtraje que permitan restringir la selección a aquellos homólogos que presenten una mayor similitud a la familia funcional de interés. Se recuperarán las secuencias homólogas para incorporarlas a posteriores análisis y se añadirán otro tipo de análisis en el *pipeline*. Como se ha comentado, la inclusión final de un determinado análisis dependerá del rendimiento de las herramientas utilizadas, la información adicional que aporte para la identificación de la región funcional, la disponibilidad de las herramientas que se requieran para realizar el análisis y la facilidad de automatizarse en el *workflow* final.

2) Homología por secuencia, alineamiento múltiple

Los alineamientos múltiples permiten análisis más precisos, detectar homólogos más distantes y proporcionan mayor información estructural y funcional por lo que también se ha incluido este tipo de análisis en el *pipeline*. Se realizará el alineamiento múltiple de las proteínas pertenecientes a una misma funcional con herramientas específicas para ello, como Clustal o MUSCLE, y se construirán perfiles HMM (*Hidden Markov Models*) utilizando HMMR3. Posteriormente se buscaran los perfiles HMM en la secuencia genómica traducida a proteína. Alternativamente podría haberse escogido una estrategia similar como la utilización del programa PSI-BLAST basado en las matrices PSSMs (*position-specific weight matrices*) (2). Las regiones candidatas identificadas en esta fase pueden ser útiles para identificar motivos conservados en pasos posteriores del *pipeline*.

3) Homología por patrón funcional

Un tercer conjunto de análisis que nos permitirá refinar la identificación de las regiones se basa en la homología por patrón funcional, ya sea un motivo o dominio proteico.

El primer método utilizado ha sido la búsqueda de homología por dominios funcionales conocidos. Los dominios se definen como unidades estructurales independientes en las proteínas. La predicción de los dominios estructurales de la familia funcional en nuestra secuencia puede automatizarse utilizando HMMER si estos dominios son conocidos y están disponibles en bases de datos públicas como, por ejemplo, Pfam (http://pfam.xfam.org/) (7). Para ello se debe extraer los perfiles HMM descritos en las bases de datos que se correspondan con nuestras secuencias proteicas y alinearnos con las secuencias sin anotar previamente traducidas. Este tipo de análisis es similar al descrito anteriormente, en el que se utilizaban las proteínas de referencia para construir modelos HMM pero en este caso nos referimos a dominios estructurales ya descritos que se relacionan con una funcionalidad concreta.

Como alternativa a la primera aproximación, se ha realizado un análisis de homología por patrón funcional. Los patrones de secuencia definen pequeños segmentos conservados de las proteínas con una potencial funcionalidad. Se buscarán patrones conocidos, descritos en la base de datos PROSITE (http://prosite.expasy.org/) (13), en las proteínas de una misma familia funcional CAZy. Los motivos así identificados se escanearán en las regiones de la secuencia genómica de referencia candidatas a pertenecer a esa misma familia funcional.

4) Homología por similitud 3-D

Aunque inicialmente estaba previsto incluir un análisis por similitud de homología 3-D, finalmente se descartó este fase en el *pipeline* tal y como se comentó durante el seguimiento y en el apartado 4 de conclusiones.

En una primera se fase se diseñará el pipeline escogiendo e incorporando los análisis y herramientas previamente descritas y comprobando su funcionalidad utilizando un *dataset*, secuencias proteicas de referencia y secuencia genómica microbiana sin anotar, de prueba.

Un punto importante en la consecución de los objetivos es la automatización del *pipeline* en un *workflow*. El *pipeline* incorpora diversos análisis que requieren de diversas herramientas y consultas a base de datos. Por esa razón, el trabajo se llevará a cabo en una máquina virtual con sistema operativo Linux donde se hayan instalado localmente el mayor número posible de programas (como BLAST o HMMER) y bases de datos, aunque estas últimas necesitarían de una actualización constante. Para realizar los scripts se utilizará R, por su compatibilidad con Shiny, aunque se podrán incorporar otros lenguajes cuando sea necesario, como Phyton o instrucciones de sistema a través del *command-line*. La incorporación del *workflow* en una aplicación web permitirá una mejor visualización de los resultados. Este objetivo secundario en la realización del proyecto, puede llevarse a cabo mediante la utilización de la herramienta Shiny que está específicamente diseñada para crear aplicaciones web en R y es de fácil uso.

1.4 Planificación del Trabajo

A continuación se detallan las táreas organizadas por objetivos tal y como se previó en la planificación inicial del trabajo:

Objetivo 1- Seleccionar el tipo de análisis, herramientas y criterios que se utilizarán para identificar las regiones candidatas.

Aunque parte de las tareas derivadas de este primer objetivo ya se han realizado para elaborar el plan de trabajo (tipo de análisis a realizar), quedarán pendientes las siguientes tareas:

1.1- Preseleccionar herramientas específicas y disponibles para cada tipo de análisis previsto.

1.2- Establecer posibles criterios de calidad para cada uno de los *outputs* generados.

1.3- Crear una máquina virtual donde se instalen las herramientas necesarias para la ejecución del proyecto.

Objetivo 2- Diseñar el *pipeline* donde se vayan incorporando los análisis seleccionados.

Las tareas necesarias para alcanzar este objetivo se organizan en función de los análisis descritos en el apartado 3. Para cada análisis se testarán las herramientas previamente seleccionadas y se fijarán criterios de filtrado y calidad para los descriptores que se generen en cada uno de ellos, evaluando en cada momento su utilidad.

2.1- Análisis de homología por secuencia, alineamiento a pares.

2.2- Análisis de homología por secuencia, alineamiento múltiple.

2.3- Análisis de homología por patrón funcional.

2.4- Análisis de similitud por similitud 3-D.

2.5- Comprobar con un *dataset* de prueba (familia funcional y secuencia genómica no anotada) el rendimiento del *pipeline*. Esta última tarea se superpone a las anteriores ya que se necesitará del *dataset* para testar las diferentes herramientas.

Objetivo 3- Automatizar el *pipeline* en un *workflow* para que cada grupo de análisis genere una salida con la información necesaria que permita la identificación de la región candidata.

3.1- Automatizar la selección de la familia funcional y la recuperación de las secuencias proteicas en formato fasta (*input*).

3.2- Automatizar el *pipeline* propuesto para cada análisis mediante scripts *scripts* que no requieran de modificación cada vez que se ejecuten (*output*).

Las tareas se dividen en función de las salidas que previsiblemente generará el *workflow*:

3.2.1- *Output* 1: alineamiento a pares.

3.2.2- *Output* 2: alineamiento múltiple.

3.2.3- *Output* 3: homología por patrón funcional.

3.2.4- *Output* 4: modelado 3-D.

Objetivo 4- Mejorar la visualización de los resultados mediante la Incorporación del *workflow* en una aplicación web o similar.

4.2- Crear una salida resumen de todos los *outputs* generados en el *workflow.*

4.1- Crear una aplicación en Shiny donde se incorporen los scripts creados para la automatización del *workflow*.

A continuación en la FIGURA 1 se muestra la planificación temporal prevista en el plan de trabajo:

Hitos y tareas	Inicio	Fin	Duración
PEC1- Plan de trabajo	14/10/2016		
Preparar herramientas trabajo	**17/10/2016**	**20/10/2016**	**4 días**
Búsqueda herramientas y criterios calidad	17/10/2016	19/10/2016	
Creación de la máquina virtual	19/10/2016	20/10/2016	
Diseñar pipeline	**21/10/2016**	**10/11/2016**	**21 días**
Alineamiento a pares	21/10/2016	25/10/2016	
Alineamiento múltiple	26/10/2016	30/10/2016	
Homología por patrón funcional	30/10/2016	05/11/2016	
Similitud 3-D	04/11/2016	10/11/2016	
PEC2 - Desarrollo del trabajo - Fase 1	31/10/2016		
Automatizar pipeline en un workflow	**11/11/2016**	**25/11/2016**	**15 días**
Input	11/11/2016	13/11/2016	
Output	14/11/2016	25/11/2016	
Mejorar visualización de los resultados	**26/11/2016**	**05/12/2016**	**10 días**
Crear una aplicación en shiny	26/11/2016	05/12/2016	
PEC3 - Desarrollo del trabajo - fase 2	28/11/2016		
Redacción de la memoria final	05/12/2016	26/12/2016	22 días
Memoria del trabajo final	26/12/2016		
Elaborar la presentación y formulario autoevaluación	27/12/2016	01/01/2017	**5 días**
Presentación y autoevaluación del trabajo	02/01/2017		
Preparar la presentación y defensa del trabajo	09/01/2017	17/01/2017	**8 días**
Defensa pública	17/01/2017		

FIGURA 1. Planificación temporal de las tareas a realizar.

1.5 Breve sumario de productos obtenidos

- Paquete cazypredict de R que incluye las cuatro funciones necesarias para la ejecución del *pipeline* diseñado. Permite identificar regiones candidatas de una secuencia genómica a pertenecer a una determinada familia funcional
- Aplicación Shiny para una mejor interacción con el paquete cazypredict
- Máquina virtual con los herramientas y programas necesarios para ejecutar el *worklfow* instalados localmente.

1.6 Breve descripción de los otros capítulos de la memoria

En el apartado 2 se describe la creación del *pipeline* y su automatización con funciones de R. Las fases principales del *pipeline* descritas son:

- La preparación de la base de datos CAZy y recuperación de secuencias fasta de las proteínas pertenecientes a una misma familia funcional
- Análisis por homología de secuencia mediante alineamiento a pares y múltiple
- Búsqueda por patrón funcional, mediante perfiles de dominios Pfam o patrones descritos en PROSITE

En el apartado 3 se realiza una comprobación del *workflow* con el paquete cazypredict y la aplicación Shiny.

En los anexos se incluye la documentación necesaria utilizada para la instalación de los diversos programas, un script auxiliar para el proyecto realizado y los scripts de las cuatro funciones de R y de la aplicación Shiny.

2 Creación y automatización del *pipeline*

2.1 Creación de la máquina virtual

Se ha creado una máquina virtual (VM, *virtual machine*) con VirtualBox 4.3.40 de Oracle (https://www.virtualbox.org/) donde se ha instalado un sistema basado en Linux 64 bits (Ubuntu 16.04 LTS) que ha servido para el desarrollo del *pipeline* y su posterior automatización. La máquina virtual dispone de 20 GB de disco duro creado dinámicamente y se ha configurado con 6 GB de memoria, aunque este último parámetro puedo modificarse según las características del ordenador huésped. El tamaño del disco duro es limitado por lo que se recomienda utilizar una carpeta compartida con el sistema huésped donde se almacenen los archivos de gran tamaño. Todo el proyecto se ha desarrollado en la máquina virtual y contiene todos los archivos y programas necesarios para ejecutar el *workflow*. El pipeline se ha desarrollado íntegramente desde el terminal de Linux mediante el *command-line* y la automatización se ha realizado en el entorno de R. El nombre de usuario y la contraseña para utilizar la VM es "uoc" en ambos casos. Debido al tamaño actual del disco duro de la VM, y la imposibilidad de entregarse por el medio habitual de la UOC, se enviará un correo al consultor y al responsable con un el link de Google Drive donde se podrá descargar la máquina virtual una vez entregada la memoria.

Para instalar esta máquina virtual en otro ordenador los pasos a seguir serán los siguientes:

1. Entramos en VirtualBox y seleccionamos el botón de "Nueva" en la parte superior izquierda

FIGURA 2. Creación de una nueva máquina virtual.

2. Introducimos un nombre para la máquina virtual, seleccionamos tipo Linux y versión Ubuntu (64 bits). Asignamos la memoria que deseemos en función de nuestro ordenador huésped. En unidad de disco duro seleccionamos "Usar un archivo de disco duro virtual existente" y en el icono de la carpeta adyacente seleccionamos la localización donde se encuentra la copia de la VM (TFM.vdi) que se habrá descargado previamente del link de Google Drive.

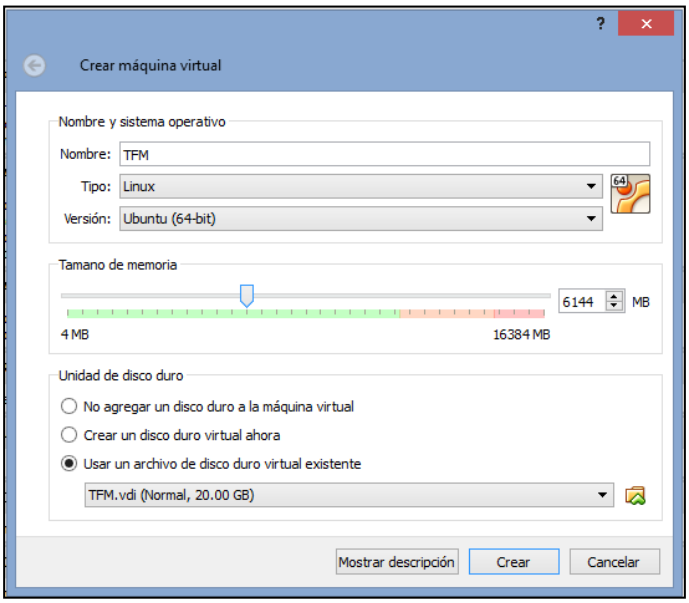


FIGURA 3. Ajustes para la creación de la nueva máquina virtual.

Se han instalado localmente todas aquellas herramientas (software libre o de código abierto) que se han considerado imprescindibles para la ejecución del trabajo. Además de herramientas básicas de uso general del sistema y de uso general en Bionformática, a continuación se describen los programas más destacados que se han seleccionado para su utilización en el *workflow*. En el ANEXO I (apartado 6.1) se indican los programas instalados y las instrucciones de instalación correspondientes.

- **cazy-parser**: herramienta basada en Python para descargarse la base de datos CAZy (8).
- **EDirect** (*Entrez Direct*): conjunto de herramientas que proporciona acceso a la plataforma del NCBI y todas sus bases de datos (publicaciones, secuencias…) desde un terminal de UNIX (https://www.ncbi.nlm.nih.gov/books/NBK179288/).
- **EMBOSS** (*The European Molecular Biology Open Software Suite*): una compilación de distintas herramientas de código abierto específicamente diseñadas para análisis relacionados con la biología molecular (http://emboss.sourceforge.net/) (10).
- **BLAST** (*Basic Local Alignment Search Tool*): BLAST se utiliza como herramienta para encontrar regiones de similitud entre secuencias biológicas. El programa pude comparar secuencias proteicas o nucleotídicas con una base de datos secuencias (pública o creada por el usuario) y calcula estadísticos asociados (https://blast.ncbi.nlm.nih.gov/Blast.cgi).
- **Bedtools / Samtools**: conjunto de utilidades para realizar una gran variedad de tareas relacionadas con el análisis genómico y que permiten manipular archivos en formato FASTA,

BED, SAM, BAM o GFF/GTF entre otros. Alguna de las utilidades que se incluyen son la recuperación de secuencias fasta a partir de un archivo tipo bed o creación de un índice en archivos fasta (http://bedtools.readthedocs.io/en/latest/, http://samtools.sourceforge.net/.

- **Clustal Omega:** programa de alineamiento múltiple (http://www.ebi.ac.uk/Tools/msa/clustalo/) que utiliza nuevas métodos de construcción progresiva, perfil-perfil HMM y "*seeded guide trees*" para generar alineamientos entre tres o más secuencias.
- **MUSCLE:** programa ampliamente utilizado para el alineamiento múltiple de 3 o más secuencias de DNA o proteína. Esta herramienta realiza comparaciones múltiples de secuencia por *Log-Expectation* (http://www.ebi.ac.uk/Tools/msa/muscle/).
- **HMMER3**: programa utilizado para la búsqueda de secuencias homólogas que implementa en sus métodos modelos probabilísticos denominados perfiles HMM (*hidden Markov models*) (http://hmmer.org/). Esta herramienta puede utilizarse tanto en los alineamientos múltiples como en la homología por patrón funcional.
- **ps-scan:** programa en Perl utilizado para escanear uno o diversos patrones o perfiles de PROSITE en una o diversas secuencias de Swiss-Prot o en formato fasta. Requiere la compilación de dos programos externos incluidos en el paquete PFTOOLS: "pfscan" y "psa2msa". Es la versión local de la herramienta ScanProsite (http://prosite.expasy.org/scanprosite/).

2.2 Preparación de la base de datos

Para diseñar el *pipeline* se ha tenido en cuenta, en primer lugar, el tipo de datos con los que vamos a trabajar. La familia funcional microbiana se obtendrá de la base de datos CAZy, especializada en enzimas con actividad de carbohidrato (9). De esta base de datos se debe escoger el tipo de bacteria y familia funcional de interés para la cual se obtendrán los números de entrada (o *accession number*) de las proteínas que forman parte de dicha familia. Los números de entrada servirán para recuperar las secuencias proteicas en formato fasta de una base de datos pública y se utilizarán como referencia en el *workflow*. Además, el *workflow* necesitará como datos de entrada una secuencia genómica microbiana que consistirá en secuencias de nucleótidos de DNA sin anotaciones como *reads*, *contigs* o *scaffolds*.

El primero paso fue, por tanto, recuperar la base de datos CAZy. La web que alberga la base de datos CAZy (http://www.cazy.org/) es exclusivamente de consulta y no permite descargarse las familias ni los números de acceso de las proteínas asociadas. Aunque existen otras plataformas que han desarrollado herramientas para la recuperación de los datos CAZy, algunas actualmente están inactivas (http:/mothra.ornl.gov/cgi-bin/cat/cat.cgi) (11), se necesita permiso explícito de su autor para su utilización (http://research.ahv.dk/cazy) o, en el caso de dbCAN (http://csbl.bmb.uga.edu

/dbCAN/) (16), la base de datos disponible es incompleta, no en registros sino en la información que proporciona y que permita su posterior automatización.

En GITHUB se encontró un programa CAZy-parser, https://github.com/rodrigovrgs/cazy-parser (8), que permite la descarga y posterior recuperación de las secuencias en formato fasta. Los detalles de la instalación se especifican en el ANEXO I (apartado 7.1). Debido a que la recuperación de las secuencias en formato fasta es muy poco flexible, únicamente se pueden descargar por nombre de familia y no por descripción, por ejemplo, de la proteína o por tipo de organismo, se ha utilizado únicamente el CAZy-parser para la descarga de la base de datos, proceso sumamente lento, que requiere de múltiples horas, con el siguiente comando:

```
$ create cazy_db
```

Esta función utiliza un script basado en Pyhton, create_cazy_db.py, con el código necesario para procesar la estructura HTML y extraer la información.

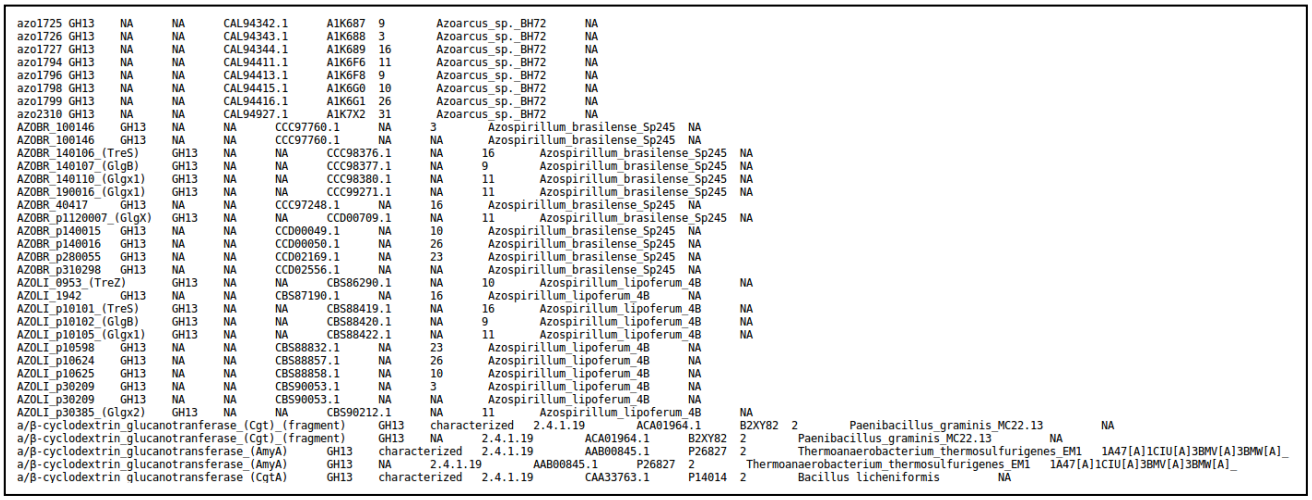
A partir de la descarga realizada, se ha manipulado a través del *command-line* la base de datos para obtener un formato tabular que permitiera posteriormente acceder fácilmente a la base de datos. El script necesario para la modificación de la base de datos se detalla en ANEXO II (apartado 7.2). Se ha creado una copia local de la base de datos CAZy descargada originalmente (CAZy_DB_02-11-2016.csv) y procesada (CAZy_DB_02-11-2016_parsed.txt) en el directorio "/home/uoc/Desktop/Data/cazy_db/" de la máquina virtual. En la base de datos procesada, además, se han eliminado dos columnas (domain y organism_code) que, aunque están incluidas en la descarga, no se encuentran en la web de CAZy.

La información disponible en la base de datos se describe en la siguiente tabla:

TABLA 1. Campos incluidos en la base de datos CAZy.

1.	protein_name	name of the protein	
2.	family	including the class or category of the module and the family number:	
		GH	Glycoside Hydrolase (ex: GH7)
		GT	Glycosyltransferase (ex: GT7)
		PL	Polysaccharide Lyase (ex: PL5)
		CE	Carbohydrate Esterase (ex: CE2)
		AA	Auxiliary Activity (ex: AA10)
3.	tag	characterized status	
4.	ec	stands for enzyme comission number	
5.	genbank	Id or accession number for GenBank database	
6.	uniprot	code for Uniprot database	
7.	subfamily	subgroups found within a family	
8.	organism	protein belongs to this organism	
9.	pdb	code for Protein Data Bank (experimentally-determined structures)	

A continuación se muestra un extracto de la base de datos donde se puede apreciar los distintos campos anteriormente descritos:



```
azo1725 GH13    NA      NA      CAL94342.1      A1K687  9       Azoarcus_sp._BH72       NA
azo1726 GH13    NA      NA      CAL94343.1      A1K688  3       Azoarcus_sp._BH72       NA
azo1727 GH13    NA      NA      CAL94344.1      A1K689  16      Azoarcus_sp._BH72       NA
azo1794 GH13    NA      NA      CAL94411.1      A1K6F6  11      Azoarcus_sp._BH72       NA
azo1796 GH13    NA      NA      CAL94413.1      A1K6F8  9       Azoarcus_sp._BH72       NA
azo1798 GH13    NA      NA      CAL94415.1      A1K6G0  10      Azoarcus_sp._BH72       NA
azo1799 GH13    NA      NA      CAL94416.1      A1K6G1  26      Azoarcus_sp._BH72       NA
azo2310 GH13    NA      NA      CAL94927.1      A1K7X2  31      Azoarcus_sp._BH72       NA
AZOBR_100146    GH13    NA      NA      CCC97760.1      NA      3       Azospirillum_brasilense_Sp245   NA
AZOBR_100146    GH13    NA      NA      CCC97760.1      NA      NA      Azospirillum_brasilense_Sp245   NA
AZOBR_140106_(TreS)     GH13    NA      NA      CCC98376.1      NA      16      Azospirillum_brasilense_Sp245   NA
AZOBR_140107_(GlgB)     GH13    NA      NA      CCC98377.1      NA      9       Azospirillum_brasilense_Sp245   NA
AZOBR_140110_(Glgx1)    GH13    NA      NA      CCC98380.1      NA      11      Azospirillum_brasilense_Sp245   NA
AZOBR_190016_(Glgx1)    GH13    NA      NA      CCC99271.1      NA      11      Azospirillum_brasilense_Sp245   NA
AZOBR_40417     GH13    NA      NA      CCC97248.1      NA      16      Azospirillum_brasilense_Sp245   NA
AZOBR_p1120007_(GlgX)   GH13    NA      NA      CCD00709.1      NA      11      Azospirillum_brasilense_Sp245   NA
AZOBR_p140015   GH13    NA      NA      CCD00049.1      NA      10      Azospirillum_brasilense_Sp245   NA
AZOBR_p140016   GH13    NA      NA      CCD00050.1      NA      26      Azospirillum_brasilense_Sp245   NA
AZOBR_p280055   GH13    NA      NA      CCD02169.1      NA      23      Azospirillum_brasilense_Sp245   NA
AZOBR_p310298   GH13    NA      NA      CCD02556.1      NA      NA      Azospirillum_brasilense_Sp245   NA
AZOLI_0953_(TreZ)       GH13    NA      NA      CBS86290.1      NA      10      Azospirillum_lipoferum_4B       NA
AZOLI_1942      GH13    NA      NA      CBS87190.1      NA      16      Azospirillum_lipoferum_4B       NA
AZOLI_p10101_(TreS)     GH13    NA      NA      CBS88419.1      NA      16      Azospirillum_lipoferum_4B       NA
AZOLI_p10102_(GlgB)     GH13    NA      NA      CBS88420.1      NA      9       Azospirillum_lipoferum_4B       NA
AZOLI_p10105_(Glgx1)    GH13    NA      NA      CBS88422.1      NA      11      Azospirillum_lipoferum_4B       NA
AZOLI_p10598    GH13    NA      NA      CBS88832.1      NA      23      Azospirillum_lipoferum_4B       NA
AZOLI_p10624    GH13    NA      NA      CBS88857.1      NA      26      Azospirillum_lipoferum_4B       NA
AZOLI_p10625    GH13    NA      NA      CBS88858.1      NA      10      Azospirillum_lipoferum_4B       NA
AZOLI_p30209    GH13    NA      NA      CBS90053.1      NA      3       Azospirillum_lipoferum_4B       NA
AZOLI_p30209    GH13    NA      NA      CBS90053.1      NA      NA      Azospirillum_lipoferum_4B       NA
AZOLI_p30385_(Glgx2)    GH13    NA      NA      CBS90212.1      NA      11      Azospirillum_lipoferum_4B       NA
a/β-cyclodextrin_glucanotranferase_(Cgt)_(fragment)     GH13    characterized   2.4.1.19        ACA01964.1      B2XY82  2       Paenibacillus_graminis_MC22.13          NA
a/β-cyclodextrin_glucanotranferase_(Cgt)_(fragment)     GH13    NA      2.4.1.19        ACA01964.1      B2XY82  2       Paenibacillus_graminis_MC22.13          NA
a/β-cyclodextrin_glucanotransferase_(AmyA)      GH13    characterized   2.4.1.19        AAB00845.1      P26827  2       Thermoanaerobacterium_thermosulfurigenes_EM1    1A47[A]1CIU[A]3BMV[A]3BMW[A]_
a/β-cyclodextrin_glucanotransferase_(AmyA)      GH13    NA      2.4.1.19        AAB00845.1      P26827  2       Thermoanaerobacterium_thermosulfurigenes_EM1    1A47[A]1CIU[A]3BMV[A]3BMW[A]_
a/β-cyclodextrin_glucanotransferase_(CqtA)      GH13    characterized   2.4.1.19        CAA33763.1      P14014  2       Bacillus licheniformis          NA
```

FIGURA 4. Extracto de la familia GH13 de base de datos Cazy. En los últimos registros se pueden observar proteínas caracterizadas (columna 3, *characterized*) y con código PDB.

2.3 Recuperación de las secuencias fasta

Una vez extraída la base de datos CAZy se puede acceder a cualquiera de sus categorías, obtener el número de acceso de las proteínas de interés y recuperar las secuencias en formato fasta de la base de datos pública de referencia NCBI (https://www.ncbi.nlm.nih.gov/protein). Las secuencias fasta de las proteínas de una misma familia funcional se utilizarán en análisis posteriores.

Para diseñar y configurar las diferentes fases del *pipeline* descritas a continuación y buscar candidatos a pertenecer a una familia funcional se han utilizado los siguientes *datasets* de ejemplo:

- Como secuencia genómica de referencia se han utilizado los *scaffolds* de *Paenibacillus barcinonensis*.
- Como familia funcional se ha utilizado "*glucuronoyl*"y "*xylosidase*" como término de búsqueda.

Para obtener las secuencias fasta se buscó en la base de datos local CAZy aquellas proteínas que coincidieran con los términos anteriormente descritos, mediante el comando "grep", y se recuperaron solamente los registros únicos. A continuación se ejemplifica el comando utilizado:

```
$ grep "glucuronoyl" CAZy_DB_02-11-2016_parsed.txt | awk '{print $7}' | sort | uniq >
list_acc_ glucuronoyl.txt
```

```
uoc@tfm:~$ grep "glucuronoyl" /home/uoc/Desktop/Data/cazy_db/CAZy_DB_02-11-2016_parsed.txt | awk '{print $2,$3,$7,$10}' | sort | uniq
4-O-methyl-glucuronoyl_methylesterase CE15 XP_003026289.1 Schizophyllum_commune_H4-8
4-O-methyl-glucuronoyl_methylesterase_(Cip2) CE15 AAP57749.1 Trichoderma_reesei_QM6A
4-O-methyl-glucuronoyl_methylesterase_(Ge2;StGE2;MYCTH_55568) CE15 AEO60464.1 Myceliophthora_thermophila_ATCC_42464_(Spoth2)
4-O-methyl-glucuronoyl_methylesterase_(GE) CE15 AIY68500.1 Cerrena_unicolor
4-O-methyl-glucuronoyl_methylesterase_/_glucuronoyl_cinnamylesterase_(PaGE1;Pa_0_910;PODANSg148) CE15 CAP60908.1 Podospora_anserina_S_mat+_(Podan2)
4-O-methyl-glucuronoyl_methylesterase_(PcGE1;PcGCE) CE15 AFM93784.1 Phanerochaete_carnosa
glucuronoyl_esterase CE15 AOT21131.1 Acremonium_alcalophilum_ATCC_90507
```

FIGURA 5. Registros correspondientes a la familia funcional *glucuronoyl*.

En el caso anterior, familia *glucoronyl*, se recuperan 7 números de acceso de acceso. La misma búsqueda en la web CAZy reporta 11 registros. Cuatro registros están duplicados (mismo organismo, mismo nombre de proteína) ya que se han clasificado como la familia funcional CE15 y también como CBM1, categoría perteneciente a módulos asociados a carbohidratos (CBM, *Carbohydrate-Binding Modules*).

La herramienta utilizada únicamente permite obtener los registros asociados a familias funcionales correspondientes a módulos catalíticos (enzimas): *Glycoside Hydrolases*, *GlycosylTransferases*, *Polysaccharide Lyases*, *Carbohydrate Esterases* y *Auxiliary Activities*. Se ha intentado buscar y utilizar otras herramientas para recuperar la categoría CBM pero, como se ha comentado anteriormente, muchas de ellas están obsoletas ya que dependen de la estructura HTML de la web CAZy y ésta va cambiando regularmente. Para poder recuperar de manera exhaustiva la información de la web se debería crear un script personalizado y complejo para procesar HTML. Aunque lo óptimo sería disponer de esta información en nuestra base de datos, se ha considerado que puede ejecutarse el *workflow* con la misma eficencia ya que los CBM siempre están asociados a un módulo catalítico, por ello la duplicidad de registros (organismo, proteína) cuando se busca en la web. El número de acceso para ambos registros, módulo catalítico y módulo asociado, previsiblemente será el mismo aunque la búsqueda en nuestra base de datos no se podrá hacer por código de CBM (ej. CBM1, CMB35...).

Otro inconveniente de la base de datos creada, y que de nuevo depende de la estructura HTML en que se ha organizado la web, es que no se distingue entre organismos eucariotas y bacterias. Aunque si se puede realizar una búsqueda por organismo específico (género o especie) no permite filtrar de manera general por eucariotas y bacterias.

Una vez se han obtenido los números de acceso se puede utilizar la herramienta efetch, de las utilidades Edirect, para recuperar las secuencias fasta de la base de datos del NCBI. Como input utilizamos el listado de números de acceso creados en el paso anterior:

```
$ efetch -db protein -id $(paste -s -d ',' list_acc_ glucuronoyl.txt) -format fasta >
glucuronoyl_prot_seqs.fa
```

Las secuencias fasta así recuperadas son las que se utilizarán como referencia para buscar candidatos a pertenecer a una determinada familia funcional en la secuencia genómica.

2.3.1 Automatización con la función searchcazy

Para automatizar la búsqueda y recuperación de familias funcionales se ha creado la función correspondiente en R *searchcazy*.

```
searchcazy (pattern, organism, type = c("PROT", "FAM"))
```

Los argumentos de dicha función son los siguientes:

- pattern: texto a buscar en la base de datos, puede ser el nombre de la proteína o el ID de la familia.
- organism: texto correspondiente al organismo (género o especie) en el que se desea realizar la búsqueda. Este argumento es opcional.
- type: el tipo de búsqueda a realizar. Los posibles tipos son los siguientes:
    - PROT: para buscar por nombre de proteína
    - FAM: para buscar por ID de familia

La función *searchcazy* permite obtiene los números de acceso relativos a los parámetros introducidos por el usuario, que se guardarán en el archivo list_acc.txt, y recuperar las correspondientes secuencias fasta del NCBI que se guardarán en el archivo list_acc.fa. Los archivos creados se guardarán siempre en el directorio donde se ejecutarán todos los análisis y que previamente se habrá creado para esta finalidad, en este caso "/home/uoc/Desktop /CAZy_pipeline/". Como *output* de la función en la consola se visualizará el número de secuencias que se han recuperado. El script completo de la función se muestra en el ANEXO III (apartado 7.3).

En una fase final del proyecto se ha detectado un error en la función cuando se intenta recupera más de, aproximadamente, 10000 secuencias fasta. Este error puede subsanarse sustituyendo el comando de sistema efetch, invocado desde R, por un script en bash que incluya este comando en un *loop* "while". No obstante, debido al tiempo ajustado de la entrega y los desvíos en la planificación que se comentan en el apartado de conclusiones, no se ha podido introducir esta mejora en los productos finales.

2.4 Homología por secuencia, alineamiento a pares

La búsqueda por similitud de secuencia es la primera fase, y posiblemente una de las más informativas, para identificar homólogos. De hecho, las familias de CAZy se definen en parte por su similitud de secuencia aminoacídica con, al menos, un miembro caracterizado bioquímicamente. Por tanto, el primer análisis ha sido la búsqueda de homología de secuencia entre las proteínas de la misma familia funcional y la secuencia genómica de *Paenibacillus* mediante alineamientos locales a pares y utilizando un algoritmo de BLAST (https://blast.ncbi.nlm.nih.gov/Blast.cgi) específico para proteínas.

2.4.1 Traducción

Para realizar el alineamiento se ha tenido en cuenta la diferente naturaleza de las secuencias, existiendo dos posibilidades para aplicar el algoritmo:

- traducir previamente la secuencia nucleotídica a proteína y utilizar blastp
- escogerá una herramienta específica de BLAST que incorpore este proceso, como tblastn.

En previsión a que posteriormente se necesitaría igualmente la secuencia proteica derivada de la secuencia de DNA genómico de referencia se decidió optar por traducir inicialmente esta secuencia genómica y tenerla ya disponible, si fuera necesario, en pasos posteriores del *workflow*. La secuencia genómica se traducirá a las seis distintas pautas (3 pautas para la secuencia *forward* y 3 pautas para la secuencia *reverse*) utilizando el programa "transeq" incluido en el paquete de EMBOSS (10). Esta herramienta devuelve seis secuencias proteicas en formato fasta, una por pauta, para cada una de las secuencias nucleotídicas que especifiquemos en el *input* y, que en este caso, serán los *scaffolds* de *Paenibacillus barcinonensis* que se encuentran en el archivo sga-scaffolds_prot.fa. A continuación se ejemplifica el comando utilizado:

```
$ transeq -frame 6 -sequence sga-scaffolds.fa -outseq sga-scaffolds_prot.fa
```

El programa añade una codificación del "_1" al "_6" al ID de la secuencia fasta para que las pautas sean fácilmente identificables.

De entre las distintas opciones que incorpora "transeq" existe la posibilidad de escoger el código genético en el que se basará la traducción. Como en este caso el genoma de referencia es una bacteria también realizaremos la traducción utilizando el código genético correspondiente (- table 11).

```
$ transeq -frame 6 -table 11 -sequence sga-scaffolds.fa -outseq sga-scaffolds_prot11.fa
```

De la misma manera que en el código estándar, la iniciación más eficiente se realiza con el codón AUG y, por tanto, no existen diferencias respecto la tabla estándar aunque se han documentado inicio de traducción alternativos con los codones GUG y UUG y existe también un caso descrito de incio CUG para una proteina de E.coli . Además de los inicios con codones NUG comentados, las bacterias pueden utilizar el codón AUU aunque en muy pocos casos. Se puede consultar una descripción exhaustiva de los distintos códigos genéticos en el siguiente link:
https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi

Para diseñar el pipeline utilizaremos las opciones por defecto de "transeq" aunque en pasos posteriores compararemos el *output* con el obtenido con el código genético específico para bacterias que, previsiblemente, reportará los mismos resultados.

2.4.2 BLAST

El análisis de homología de secuencia por alineamiento local a pares se ha realizado con la herramienta BLAST+ (3) instalada localmente en la máquina virtual. Las secuencias proteicas, traducidas a partir de la secuencia genómica, son las que se utilizan para crear la base de datos de BLAST que se utilizará para la comparativa. Utilizamos el comando "makeblastdb".

```
$ makeblastdb -in sga-scaffolds_prot.fa -dbtype "prot" -parse_seqids
```

Una vez creada la base de datos de referencia, ejecutamos BLAST, en concreto blastp, para el cuál ajustamos un *evalue* de 0.01. Este valor describe, según la definición de BLAST, el número de hits que se podrían esperar al azar cuando se realiza la búsqueda en una base de determinado tamaño. Cuanto menor es este valor mayor es la significación del alineamiento. No obstante, hay que tener presente que el cálculo de este valor toma en cuenta la longitud de la secuencia *input* por lo que alineamientos muy cortos pueden tener e*values* relativamente altos. A continuación se describe el comando utilizado:

```
$ blastp -query glucuronoyl_fam_prot.fa -task "blastp" -db sga-scaffolds_prot.fa -evalue
0.01 -outfmt "6 std qlen qcovhsp" –out glucuronoyl_fam_prot_sga_blastp
```

Se ha utilizado para el BLAST una salida en formato tabla estándar (-outftm "6 std") que incluye los campos 1 a 12 de la TABLA 2. Además, se han incluido dos campos adicionales correspondientes a la longitud de la proteína y al *coverage* (-outfmt "6 std qlen qcovhsp"), número 13 y 14, que nos facilitarán el filtrado de los datos.

TABLA 2. Campos incluidos en el *output* de BLAST.

1.	qseqid	query (e.g., gene) sequence id
2.	sseqid	subject (e.g., reference genome) sequence id
3.	pident	percentage of identical matches
4.	length	alignment length
5.	mismatch	number of mismatches
6.	gapopen	number of gap openings
7.	qstart	start of alignment in query
8.	qend	end of alignment in query
9.	sstart	start of alignment in subject
10.	send	end of alignment in subject
11.	evalue	expect value
12.	bitscore	bit score
13.	qlen	query sequence length
14.	qcovhsp	query coverage per HSP

La tabla anterior servirá de referencia para identificar los diferentes campos en el *output* que genere el BLAST. Con los parámetros anteriormente indicados se obtuvo un único hit para *glucuronoyl* y más de 2000 para *xylosidase*. Aunque se han testado parámetros más laxos (*evalue* de 0.1), el número de

hits aumenta muy ligeramente en el caso de *glucuronoyl* pero sin aportar información adicional útil ya que tanto la identidad como el *coverage* del alineamiento son muy bajos y, por tanto, se considera que no son alineamientos de calidad. Como criterio general se utilizará un *evalue* de 0.01.

El único alineamiento para *glucuronoyl* no es muy consistente ya que presenta una identidad menor al 30% y un *coverage* del 31%, tal y como se muestra en la FIGURA 6.

```
uoc@tfm:~$ more /home/uoc/Desktop/Pipeline/Blast/glucuronoyl_fam_prot_sga_blastp
AIY68500.1      unplaced-47_4   26.667  150     106     1       72      217     5752    5901    0.005   39.3    474     31
```

FIGURA 6. Output de BLAST para la familia *glucuronoyl*.

Es de destacar que esta familia funcional únicamente consta de 7 proteínas y todas ellas pertenecen a hongos, mientras que nuestro genoma candidato es bacteriano. Por tanto, es posible que no se hallen homólogos para esta familia en *Paenibacillus barcinonensis*. Si se repitiera el proceso con otro término parecido como *glucuronyl*, que si se encuentra en otras especies de bacterias, obtendríamos un mayor número de *hits*, 49, y algunos de ellos con una porcentaje de identidad bastante aceptable, alrededor del 50%, y con un *coverage* en algunos casos elevado, >80%, como se muestra a continuación:

```
uoc@tfm:~$ blastp -query /home/uoc/Desktop/Pipeline/glucuronyl_fam_prot.fa -task "blastp" -db /home/uoc/Desktop/Data/scaffolds/sga-scaffolds_prot.fa -evalue 0.01 -outfmt "6 std qlen qcovhsp"
AAD37770.1      unplaced-198_6  33.607  122     76      2       5       126     17807   17923   1.94e-10        62.0    329     37
AAD37770.1      unplaced-198_6  27.778  126     74      4       5       125     18111   18224   5.06e-05        45.1    329     37
AAD37770.1      unplaced-198_5  36.264  91      52      3       6       94      2116    2202    2.45e-06        49.3    329     27
AAD37770.1      unplaced-198_5  28.723  94      65      2       7       100     3248    3339    5.86e-05        45.1    329     29
AAD37770.1      unplaced-342_4  24.576  118     77      4       7       124     922     1027    7.68e-04        41.2    329     36
AAD37770.1      unplaced-183_5  33.333  51      34      0       4       54      940     990     0.001   40.8    329     16
AAK33600.1      unplaced-81_1   41.463  369     212     4       32      399     7185    7550    2.88e-90        298     399     92
AAK33600.1      unplaced-31_1   31.707  369     236     6       32      394     4709    5067    1.54e-46        171     399     91
AAK99096.1      unplaced-81_1   40.053  377     222     4       21      396     7177    7550    1.24e-81        273     396     95
AAK99096.1      unplaced-31_1   31.367  373     240     6       29      395     4709    5071    2.49e-43        161     396     93
AAL43286.1      unplaced-186_1  28.571  154     93      7       162     300     13263   13414   7.93e-07        50.8    349     40
AAL43286.1      unplaced-186_1  31.818  88      56      2       201     284     11325   11412   0.002   40.0    349     24
AAL43286.1      unplaced-323_3  26.241  141     86      5       163     287     19266   19404   1.45e-06        50.1    349     36
AAL43286.1      unplaced-186_2  31.731  104     61      3       178     272     12497   12599   6.50e-05        45.1    349     27
AAL43286.1      unplaced-186_2  30.405  148     81      6       178     304     12879   13025   3.88e-04        42.4    349     36
AAL43286.1      unplaced-214_2  32.168  143     79      7       200     328     4113    4251    1.02e-04        44.3    349     37
AAL43286.1      unplaced-432_2  30.303  132     70      6       162     279     9548    9671    0.001   41.2    349     34
AAX20106.1      unplaced-198_6  32.787  122     77      2       5       126     17807   17923   5.33e-10        60.8    329     37
AAX20106.1      unplaced-198_6  29.032  93      61      2       5       97      18111   18198   1.46e-04        43.5    329     28
AAX20106.1      unplaced-198_5  36.264  91      52      3       6       94      2116    2202    2.52e-06        49.3    329     27
AAX20106.1      unplaced-198_5  28.723  94      65      2       7       100     3248    3339    6.57e-05        44.7    329     29
AAX20106.1      unplaced-342_4  24.576  118     77      4       7       124     922     1027    8.98e-04        41.2    329     36
AAX20106.1      unplaced-183_5  33.333  51      34      0       4       54      940     990     0.001   40.8    329     16
ABG82476.1      unplaced-81_1   39.633  381     226     4       17      396     7173    7550    2.95e-91        300     396     96
ABG82476.1      unplaced-31_1   35.083  362     221     6       41      395     4717    5071    2.28e-53        191     396     90
ABM47015.1      unplaced-238_3  42.188  64      37      0       285     348     28116   28179   2.95e-08        56.2    407     16
ACM43487.1      unplaced-81_1   48.936  188     95      1       2       189     7221    7407    1.50e-56        191     189     99
ACM43487.1      unplaced-31_1   39.267  191     112     2       3       189     4742    4932    7.29e-31        118     189     99
ACU05029.1      unplaced-81_1   33.333  330     197     9       72      387     7221    7541    5.42e-44        164     402     79
ACU05029.1      unplaced-31_1   33.939  330     201     6       70      387     4739    5063    5.30e-41        155     402     79
AFQ98272.1      unplaced-485_3  27.632  228     140     6       161     376     343     557     2.55e-19        90.1    377     57
AFQ98272.1      unplaced-214_5  23.429  175     125     4       161     330     13773   13943   1.80e-12        68.9    377     45
AFQ98272.1      unplaced-301_6  25.275  182     117     5       161     332     2241    2413    5.96e-12        67.4    377     46
AFQ98272.1      unplaced-113_3  24.116  311     203     11      59      360     14008   14294   6.25e-09        57.8    377     80
BAA18336.1      unplaced-214_2  35.200  125     72      4       185     302     4091    4213    4.23e-11        64.3    354     33
BAA18336.1      unplaced-323_3  50.000  46      23      0       265     310     19372   19417   5.34e-06        48.5    354     13
BAA18336.1      unplaced-187_1  26.207  145     90      5       191     323     4068    4207    1.64e-04        43.5    354     38
BAA18336.1      unplaced-198_3  24.779  113     79      1       247     353     881     993     4.77e-04        42.4    354     30
BAA18336.1      unplaced-198_6  44.186  43      23      1       265     306     16287   16329   0.005   38.9    354     12
BAA18336.1      unplaced-432_2  39.623  53      32      0       265     317     9648    9700    0.008   38.1    354     15
BAA84216.1      unplaced-81_1   58.667  375     152     3       2       376     7183    7554    2.74e-143       449     377     99
BAA84216.1      unplaced-31_1   40.263  380     209     8       2       375     4707    5074    6.87e-61        212     377     99
BAG31948.1      unplaced-238_3  26.016  123     82      4       324     446     28120   28233   0.004   40.0    453     27
BAV13188.1      unplaced-214_5  45.070  355     193     1       8       360     13651   14005   1.76e-108       348     362     98
BAV13188.1      unplaced-301_6  44.179  335     184     2       23      357     2136    2467    1.10e-92        303     362     93
BAV13188.1      unplaced-485_3  23.885  381     225     14      6       357     209     553     7.04e-21        94.4    362     97
BAV13188.1      unplaced-113_3  24.211  285     183     11      2       275     13986   14248   1.09e-14        75.9    362     76
CAD47548.1      unplaced-81_1   40.682  381     222     4       19      398     7173    7550    2.01e-88        292     398     95
CAD47548.1      unplaced-31_1   32.000  375     243     5       29      397     4703    5071    6.62e-47        172     398     93
```

FIGURA 7. Output de BLAST para la familia *glucuronyl*.

Continuando con el ejemplo anterior, podemos realizar una comparativa entre los resultados obtenidos a partir de la secuencia genómica traducida a proteína y el alineamiento con *blastp* (FIGURA 8) y la utilización de *tblatsn*, que permite utilizar como *input* la secuencia de DNA (FIGURA 9). Para poder comparar los resultados utilizaremos los hits de mayor calidad, aquellos que presenten una identidad mayor al 40% y un *coverage* mayor del 80%:

```
$ blastp -query glucuronyl_fam_prot.fa -task "blastp" -db sga-scaffolds_prot.fa -outfmt "6
std qlen qcovhsp" | awk '{if ($3>=40 && $14>=80) {print $0}}'
```

AAK33600.1	unplaced-81_1	41.463	369	212	4	32	399	7185	7550	2.88e-90	298	399	92
AAK99096.1	unplaced-81_1	40.053	377	222	4	21	396	7177	7550	1.24e-81	273	396	95
ACM43487.1	unplaced-81_1	48.936	188	95	1	2	189	7221	7407	1.50e-56	191	189	99
BAA84216.1	unplaced-81_1	58.667	375	152	3	2	376	7183	7554	2.74e-143	449	377	99
BAA84216.1	unplaced-31_1	40.263	380	209	8	2	375	4707	5074	6.87e-61	212	377	99
BAV13188.1	unplaced-214_5	45.070	355	193	1	8	360	13651	14005	1.76e-108	348	362	98
BAV13188.1	unplaced-301_6	44.179	335	184	2	23	357	2136	2467	1.10e-92	303	362	93
CAD47548.1	unplaced-81_1	40.682	381	222	4	19	398	7173	7550	2.01e-88	292	398	95

FIGURA 8. Output de blastp para la familia *glucuronyl*. Se ha utilizado como *database* el genoma bacteriano traducido a proteina con transeq de EMBOSS. Los hits se han filtrado para una identidad >40% y coverage > 80%.

```
$ tblastn -query glucuronyl_fam_prot.fa -db sga-scaffolds.fa -outfmt "6 std qlen qcovhsp" |
awk '{if ($3>=40 && $14>=80) {print $0}}'
```

AAK33600.1	unplaced-81	40.209	383	225	4	18	399	21511	22650	3.43e-86	284	399	96
AAK99096.1	unplaced-81	40.053	377	222	4	21	396	21529	22650	3.46e-82	272	396	95
ACM43487.1	unplaced-81	48.936	188	95	1	2	189	21661	22221	3.19e-50	171	189	99
BAA84216.1	unplaced-81	58.667	375	152	3	2	376	21547	22662	1.31e-128	405	377	99
BAV13188.1	unplaced-214	45.070	355	193	1	8	360	24229	23165	5.44e-99	319	362	98
BAV13188.1	unplaced-301	43.953	339	187	2	23	361	1598	591	1.15e-89	292	362	94
CAD47548.1	unplaced-81	40.470	383	224	4	17	398	21511	22650	2.94e-85	281	398	96

FIGURA 9. Output de blastn para la familia glucuronyl. Se ha utilizado como *database* los *scaffolds* con la secuencia de DNA del genoma bacteriano. Los hits se han filtrado para una identidad >40% y coverage > 80%.

Como se observa en los outputs anteriores los resultados son prácticamente idénticos. En el caso de *blastp* se obtiene un *hit* adicional que corresponde a un alineamiento con un 40.263% de identidad. Es posible que debido a las ligeras diferencias en porcentaje de identidad que hay en algunos casos entre las dos herramientas, este alineamiento presente un porcentaje de identidad un poco inferior al 40% en *tblastn* y, por tanto, no haya pasado el filtro. La posición de inicio y final en la proteína de la familia funcional es idéntica o prácticamente idéntica en ambos casos y la posición en la secuencia de referencia de nuestro genoma corresponde a la posición nucleotídica en el caso de *tblastn* que es la misma que en *blastp* pero multiplicada por 3. Es interesante destacar que distintas proteínas de la familia se alinean en la misma región de nuestra secuencia genómica de referencia indicando que éstas son claras regiones candidatas. Por tanto, se puede concluir que ambas herramientas son válidas, aunque en este caso se ha utilizado la primera. Estos resultados nos sirven, además, para validar la estrategia utilizada para crear el *pipeline*.

Se podrían establecer algunos criterios mínimos para algunos parámetros como la identidad y el *coverage* del alineamiento respecto a la secuencia proteica de la familia funcional. Sin embargo, establecer criterios generales puede ser complejo ya que cada caso es muy particular. En los ejemplos anteriores, BLAST retorna un único hit con *evalue* menor a 0.01 para el *dataset* de *glucuronoyl*. Sin embargo, en el caso de las proteínas *xylosidase*, de las cual disponemos un total de 285 para realizar la comparativa, obtenemos más de 2000 hits utilizando los mismos parámetros. El elevado número de *hits* exige, en este caso, establecer criterios de calidad para restringir las posibles regiones candidatas.

Automatizar con criterios generales la salida de este tipo de análisis, puede conllevar, tal como se desprende de los ejemplos anteriores, una pérdida o exceso información. Estos criterios deben ser establecidos en función de las características de las proteínas pertenecientes a la familia funcional y del genoma en el que se vaya a realizar la predicción. Existen diversas posibilidades encaminadas a intentar solventar esta situación:

1- Establecer criterios más laxos y dejar a manos del usuario final la decisión de restringir el *input* inicial de proteínas pertenecientes a una familia funcional determinada, es decir, concretar no únicamente la descripción de la proteína sino también el organismo, por ejemplo.
2- Establecer criterios muy restrictivos a expensas de perder información. Este podría ocurrir, por ejemplo, en aquellas familias funcionales para las cuáles haya pocos representantes o presenten poca similitud entre especies.
3- Restringir que BLAST retorne como máximo un número determinado de hits para cada una de las proteínas que hemos establecido en el *input*. De esta manera, en aquellos casos en que exista un gran número hits, únicamente se retornarán aquellos que presenten mayor significancia estadística, es decir, los alineamientos mayor calidad.

Finalmente, se ha escogido una combinación de las distintas posibilidades como la estrategia más adecuada para diseñar *pipeline*. En la automatización de la búsqueda de las proteínas pertenecientes a una determinada familias funcional, descrita en el apartado anterior, ya se ha incluido la posibilidad de que el usuario pueda restringir el *input* inicial concretando también el organismo (género o especie). Como criterio de calidad se utilizará un *evalue* por defecto de 0.01 ya que por debajo de este valor los alineamientos que se obtienen son de muy baja calidad y no permiten establecer claros candidatos. Sin embargo, será el usuario quien fijará finalmente este parámetro así como los de identidad y *coverage*. Para aquellos casos en que exista un gran número de proteínas pertenecientes a una misma familia funcional también se dará la posibilidad de limitar el número de *hits* para cada proteína perteneciente a la familia funcional a un *hit* por cada una de ellas. Esta limitación se obtiene configurando las opciones -max_target_seqs 1 y -max_hsps 1 de blastp:

```
$ blastp -query xylosidase_fam_prot.fa -task "blastp" -db sga-scaffolds_prot.fa -evalue
0.01 -outfmt "6 std qlen qcovhsp" -max_target_seqs 1 -max_hsps 1
```

Si aplicamos el comando anterior a la familia de *xylosidase*, el número de *hits* disminuye de 2212 a 267 y un gran porcentaje presenta más del 90% de *coverage* y más del 50% de identidad. Este *output* es mucho más útil y claro para identificar regiones candidatas a pertenecer una determina familia funcional. Cabe destacar, además, que diferentes proteínas de la familia se alinean en una misma región candidata de la secuencia genómica traducida a proteína, resultando en 27 regiones únicas. A continuación, como ejemplo, se muestra una de estas regiones:

AJK31203.1	unplaced-537_1	25.950	763	435	25	60	763	3718	4409	1.54e-49	189	777	91
AET31459.1	unplaced-537_1	26.602	718	396	27	98	762	3729	4368	1.28e-38	155	803	83
ABQ45227.1	unplaced-537_1	27.554	744	418	20	76	760	3716	4397	1.27e-65	238	774	89
ABQ45228.1	unplaced-537_1	27.614	746	415	20	76	760	3716	4397	2.29e-68	246	774	89
CAB89357.1	unplaced-537_1	28.533	750	407	24	71	759	3716	4397	2.02e-64	234	773	89
CUT08919.1	unplaced-537_1	28.838	749	425	25	51	744	3698	4393	1.01e-67	244	774	90
ADD92015.1	unplaced-537_1	28.964	801	459	23	24	750	3637	4401	3.92e-87	302	776	94
ADM89626.1	unplaced-537_1	29.055	709	377	26	74	715	3724	4373	8.58e-61	223	765	84
AAK38482.1	unplaced-537_1	29.078	705	389	20	113	767	3751	4394	2.27e-68	246	777	84
ADD17009.1	unplaced-537_1	29.692	714	366	27	74	715	3724	4373	2.40e-63	231	765	84
AAA80156.1	unplaced-537_1	30.330	455	249	14	58	492	3630	4036	2.97e-41	162	654	67
AMP82915.1	unplaced-537_1	31.495	689	352	24	4	599	3629	4290	5.71e-72	252	607	98
AAC99628.1	unplaced-537_1	32.620	794	444	18	15	748	3629	4391	1.63e-106	362	861	85
AIO06740.1	unplaced-537_1	33.292	802	391	17	26	714	3629	4399	2.86e-126	415	724	95
CAP07659.1	unplaced-537_1	34.110	730	394	26	53	746	3716	4394	2.09e-96	329	761	91
AAO42605.1	unplaced-537_1	35.192	807	425	18	15	759	3629	4399	4.30e-130	429	791	94
AJY53618.1	unplaced-537_1	36.957	782	416	17	10	735	3637	4397	5.30e-133	437	801	91
AMK07510.1	unplaced-537_1	37.218	798	435	13	3	760	3628	4399	1.21e-143	468	789	96
AMK07469.1	unplaced-537_1	38.143	797	429	14	3	760	3628	4399	3.21e-145	472	789	96
ACM61424.1	unplaced-537_1	38.471	785	424	12	2	745	3636	4402	8.90e-171	546	771	96
AEJ44817.1	unplaced-537_1	38.662	807	404	15	1	750	3629	4401	4.49e-166	532	779	96
AFY97406.1	unplaced-537_1	39.070	796	416	14	1	748	3627	4401	3.41e-173	553	775	97
AAD35170.1	unplaced-537_1	40.075	801	404	16	1	751	3627	4401	7.76e-175	558	778	97
AFM44649.1	unplaced-537_1	40.455	791	415	11	10	762	3629	4401	0.0	581	789	95
ACK42133.1	unplaced-537_1	43.019	795	408	15	4	759	3635	4423	0.0	587	762	99

FIGURA 10. *Output* ara la región candidata correspondiente a la secuencia unplaced_537_1 identificada con BLAST. El número de hits se ha restringido a 1 hit por cada miembro de la familia funcional con los parámetros- "max_target_seqs 1" y "-max_hsps 1".

En el caso anterior no se ha utilizado ningún filtro por *coverage* o identidad y, no obstante, es posible identificar claras regiones candidatas limitando el número de *hits*.

Como se ha comentado en el apartado de traducción, existe la posibilidad de utilizar la secuencia genómica traducida a proteína utilizando el código genético específico para bacterias, que no presenta diferencias relevantes respecto al estándar. Utilizaremos el ejemplo anterior de *xylosidases* para comparar los resultados de BLAST entre las dos traducciones. A continuación se muestran los comandos para realizar el BLAST con la secuencia traducida utilizando la tabla de código genético 11.

```
$ makeblastdb -in sga-scaffolds_prot11.fa -dbtype "prot" -parse_seqids

$ blastp -query xylosidase_fam_prot.fa -task "blastp" -db sga-scaffolds_prot11.fa -evalue
0.01 -outfmt "6 std qlen qcovhsp" -max_target_seqs 1 -max_hsps 1 >
xylosidase_fam_prot11_sga_blastp_uniq

$ wc -l xylosidase_fam_prot11_sga_blastp_uniq

267 xylosidase_fam_prot11_sga_blastp_uniq
```

En el *output* del BLAST anterior también se obtienen 267 *hits*. Estos alineamientos coinciden exactamente con los obtenidos anteriormente, para la región ejemplificad en el caso anterior, a, unplaced_537_1, y, por tanto, el *output* es exactamente mismo que el que muestra la FIGURA 10.

En el último ejemplo se puede observar como diferentes miembros de una misma familia funcional pueden alinearse en la misma región de la secuencia genómica. Cuando esto sucede, se obtienen múltiples *hits* con coordenadas solapantes y con valores para los diferentes parámetros muy similares. Para facilitar la identificación de regiones únicas, que incluyan todas las regiones solapantes, y recuperar las secuencias fasta de dichas regiones para posteriores análisis se creará, en primer lugar, un archivo con formato bed que recopile todas las coordenadas del *output* de BLAST.

```
awk '{print $2,$9,$10,$2":"$9"-"$10}' xylosidase_fam_prot_sga_blastp_uniq | tr ' ' '\t' |
sort -k1,1 -k2,2n > xylosidase_fam_prot_sga_blastp_uniq.bed
```

Con el comando merge de bedtools podemos fusionar las coordenadas que sean solapantes de la siguiente manera:

```
bedtools merge -i xylosidase_fam_prot_sga_blastp_uniq.bed >
xylosidase_fam_prot_sga_blastp_uniq_merge.bed
```

De esta manera, como ejemplo, para la región que se había mostrado en la FIGURA 10, obtendríamos una única coordenada unplaced_537_1:3627-4423.

Las secuencias fasta para cada una de las regiones candidatas únicas puede recuperarse mediante el comando getfasta de bedtools:

```
Bedtools getfasta -fi sga scaffolds_prot.fa  -bed xylosidase_fam_prot_sga_blastp_uniq_merge
.bed -fo xylosidase_fam_prot_sga_blastp_uniq.fa
```

2.4.3 Automatización con la función blastcazy

Tanto la traducción como el alineamiento por homología con el algoritmo BLAST se han automatizado mediante la función *blastcazy* cuyo script completo se indica en el ANEXO III (apartado 7.3).

```
blastcazy (filename, evalue = 0.01, identity = 0, coverage = 0, hits = c("uniq",
"multiple"), output = c("dense","full"))
```

Los argumentos de la función son los siguientes:

- filename: se debe indicar la ruta y el archivo en formato fasta que contiene la secuencia genómica en la cuál se buscarán regiones candidatas.
- evalue: número indicando el umbral para el *evalue* (por defecto es 0.01)
- identity: número indicando el valor mínimo de identidad por el que se filtraran los resultados (por defecto es 0).
- coverage: número indicando el valor mínimo de *coverage* por el que se filtraran los resultados (por defecto es 0).

- hits: tipo de filtrado por número de *hits*. Existen dos opciones:
    - "multiple": si limitación en el número de *hits* obtenidos.
    - "uniq": limitar a un *hit* por cada proteína de la familia funcional.
- output: tipo de output que retornará la función:
    - "dense": únicamente se retornarán las coordenadas de las regiones candidatas
    - "extense": se retornará una tabla con más información sobre el resultado de BLAST.

En la función se incluyen 3 argumentos (evalue, identity y coverage) que permitirán al usuario personalizar el análisis según su criterio. Aunque es recomendable utilizar un *evalue* de 0.01, o menor, para obtener *hits* de calidad, se ha incluido este argumento en la función para permitir al usuario ajustar este parámetro según su criterio. Los otros dos parámetros, identidad y *coverage*, tienen un valor por defecto de 0 en el que no se aplicaría ningún tipo de filtro.

Configurar el argumento de hits a "uniq" permite utilizar el filtro por *coverage* o identidad por defecto, y obtener unos resultados de suficiente calidad y no excesivamente extensos para identificar claramente las regiones candidatas a pertenecer a la familia funcional.

Aunque formalmente quizás sería más correcto haber incluido la opción de escoger entre la tabla estándar y la específica para bacterias, al no presentar diferencias relevantes se ha decidido no incorporar esta opción para simplificar la función.

Esta primera función es importante porque genera diversos archivos intermedios que se necesitarán en los pasos posteriores, como la traducción de la secuencia genómica de referencia a proteína. Además, la función ejecuta un script en bash que se utiliza para la creación de archivos bed y la recuperación de las secuencias fasta de las regiones candidatas que serán útiles en la fase de búsqueda por patrón funcional.

Además se puede escoger entre dos tipos de formato de *output*, "dense" y "full". Si se escoge la opción "dense" únicamente se mostrará el ID de la secuencia y las coordenadas de las regiones únicas, no solapantes, identificadas. Si escoge la opción "full" se mostrarán todos los hits obtenidos por BLAST y los campos que se indican en la TABLA 2.

2.5 Homología por secuencia, alineamiento múltiple

La siguiente fase en el *pipeline* ha sido diseñar el análisis por alineamientos múltiples. Aunque el alineamiento local a pares nos permita obtener homólogos próximos asociados significativamente a las proteínas de referencia, los alineamientos múltiples permiten análisis más precisos, detectar homólogos más distantes y proporcionan mayor información estructural y funcional. Este tipo de análisis es de mayor relevancia cuando disponemos de diversas proteínas de referencia de una misma familia funcional, como es en nuestro caso.

A partir del alineamiento múltiple de las secuencias proteicas pertenecientes a una determinada familia funcional se construirán perfiles HMM (*Hidden Markov Models*) (14), utilizando el algoritmo implementado en HMMER (http://hmmer.org/) (5). Estos perfiles pueden ser escaneados posteriormente en las secuencias sin anotar previamente traducida permitiendo la detección de homólogos con baja similitud de secuencia.

El primer paso ha sido realizar un alineamiento múltiple de las secuencias proteicas pertenecientes a la misma familia funcional, para el que se han testado dos programas: MUSCLE (6) y Clustal Omega (12). Inicialmente se había considerado utilizar ClustalW (15) pero Clustal Omega es una versión más reciente y óptima de Clustal, que mejora respecto ClustalW en la precisión del alineamiento y en la escalabilidad cuando se utilizan gran número de secuencias. A continuación se muestran los comandos utilizados para generar los archivos del alineamiento múltiple que se utilizarán posteriormente para crear un perfil HMM :

```
$ clustalo -i glucuronoyl_fam_prot.fa -o glucuronoyl_fam_prot_align_clustalo.fa
$ muscle in glucuronoyl_fam_prot.fa out glucuronoyl_fam_prot_align_muscle.fa
```

También creamos los alineamientos en formato msf, de más fácil interpretación que el formato fasta, para comparar los dos métodos. Utilizaremos como ejemplo la familia *glucuronoyl* ya que es la que presenta menos representantes para facilitar la visualización de los alineamientos:

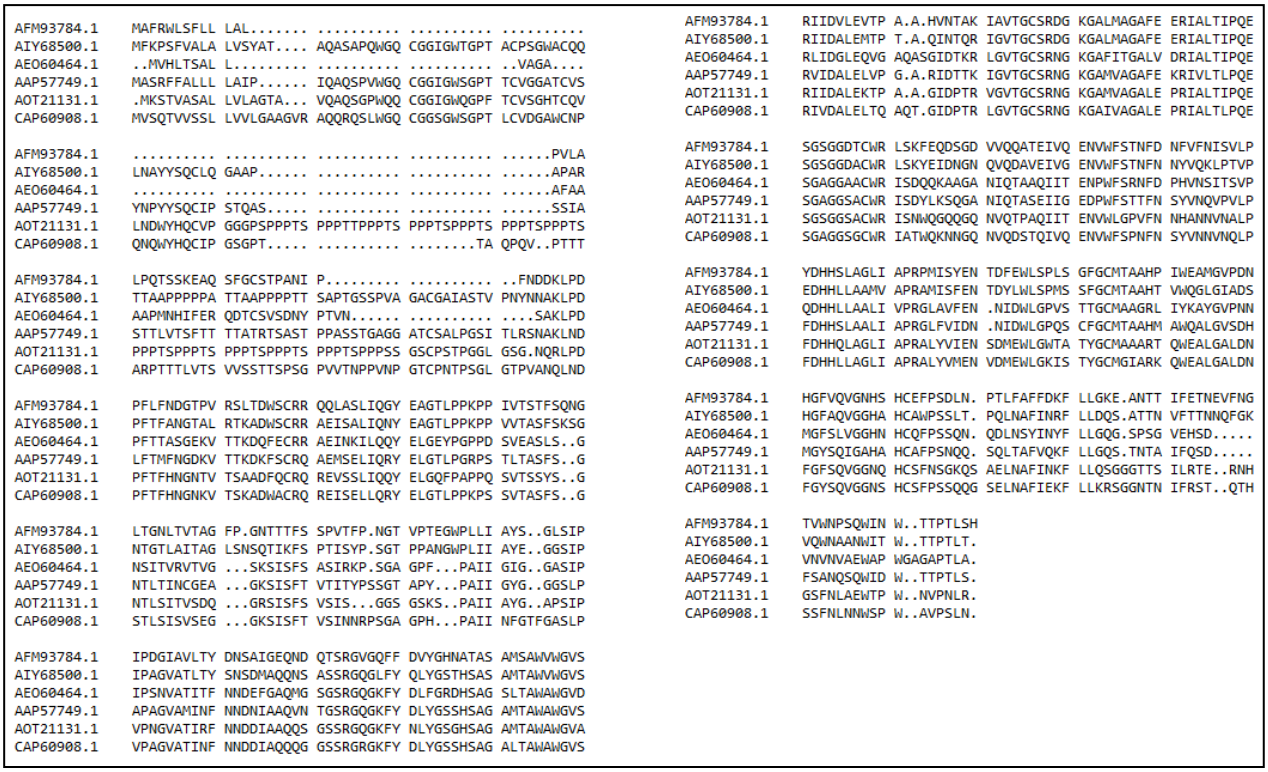


FIGURA 11. . Alineamiento de las proteínas de la familia glucuronoyl con Clustal Omega.

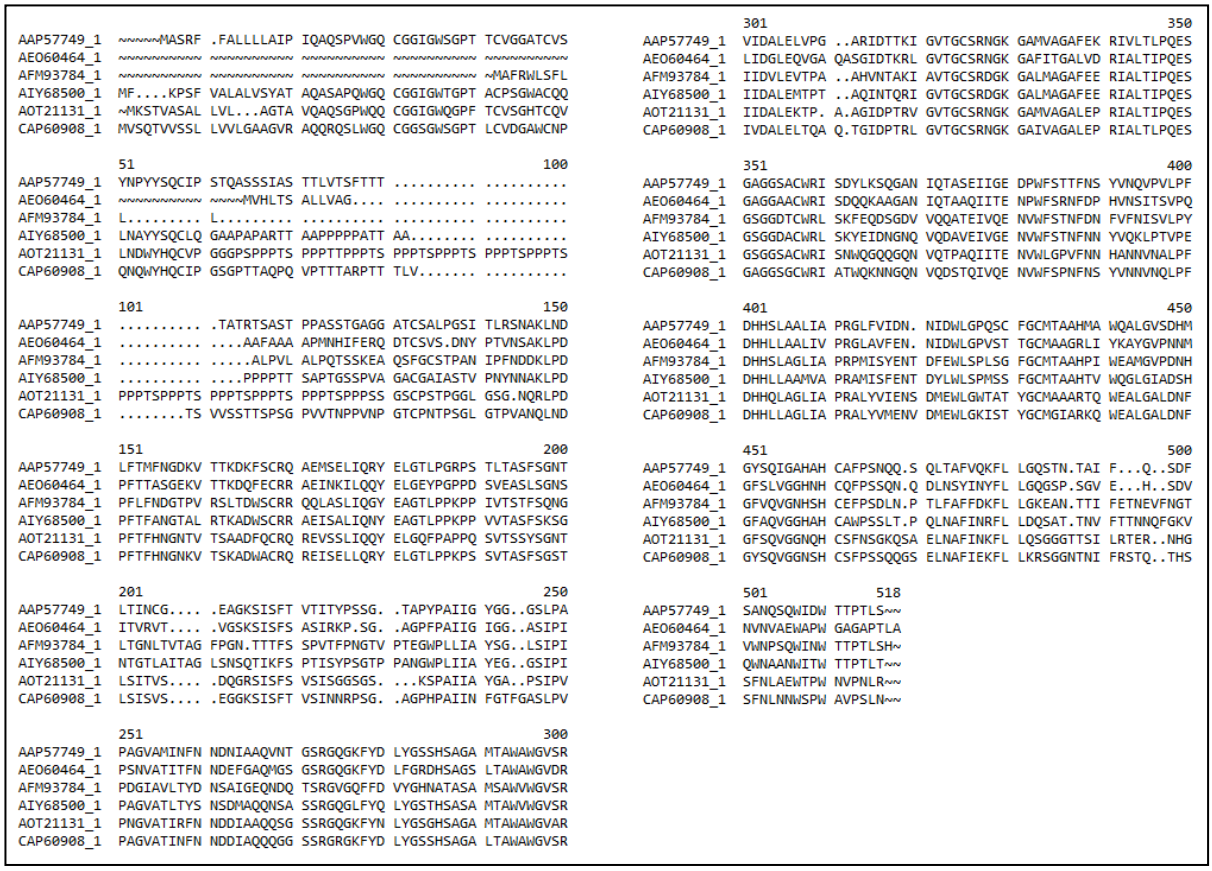


FIGURA 12. Alineamiento de las proteínas de la familia glucuronoyl con MUSCLE.

De los alineamientos anteriores se puede concluir que las proteínas clasificadas en la misma familia funcional, en este caso proteínas denominadas *glucuronoyl* y que tienen el mismo tamaño, 520 aa, presentan una gran homología. La homología es especialmente elevada en determinadas regiones centrales donde las secuencias se alinean perfectamente. Sin embargo, los dos métodos utilizados presentan ligeras diferencias alineando y delimitando estas regiones homólogas por lo que se incluirán ambos en posteriores pasos para poder evaluar su utilidad en la creación de perfiles HMM.

A partir de los alineamientos múltiples construimos un perfil HMM que, posteriormente, buscaremos en la secuencia genómica traducida a proteína con el programa HMMER3 (http://hmmer.org/) (5). Para crear los perfiles utilizamos el comando hmmbuild.

```
$ hmmbuild glucuronoyl_fam_prot_align_muscle.hmm glucuronoyl_fam_prot_align_muscle.fa
```

Los perfiles HMM convierten un alineamiento múltiple en un sistema de puntuación específico por posición que es adecuado para buscar en una base de datos secuencias homólogas remotas.

Realizamos la búsqueda en la base de datos que hemos creado anteriormente con el genoma traducido:

```
$ hmmsearch -domtblout sga-scaffolds_prot_glucuronoyl_hmm3_dom.tbl glucuronoyl_fam_prot_ali
gn_muscle.hmm sga-scaffolds_prot.fa
```

Los resultados se guardarán en archivo en formato tabular, delimitado por espacios, resumiendo el *output* obtenido por dominio, es decir, un registro para cada dominio homólogo que se ha encontrado en nuestra secuencia genómica, traducida a proteína, de referencia.

En el caso de la familia *glucuronoyl* no obtenemos ningún resultado positivo. Posiblemente, al tratarse de una familia con pocos representantes y todos en ellos pertenecientes a hongos, no existe ninguna región candidata en el genoma bacteriano que estamos analizando como ya se deducía de los resultados de BLAST. Sin embargo con *xylosidase* obtenemos diferentes regiones candidatas, por lo que utilizaremos este ejemplo para realizar la comparativa entre MUSCLE y Clustal Omega.

Clustal Omega fue mucho más rápido en el proceso, dos minutos en comparación con los siete minutos que tardó MUSCLE en realizar los alineamientos. Se construyeron los perfiles HMM y se buscaron en nuestra secuencia genómica de referencia obteniendo los siguientes resultados:

#						--- full sequence ---					--- this domain ---					hmm coord		ali coord		env coord		
# target name	accession	tlen	query name	accession	qlen	E-value	score	bias	#	of	c-Evalue	i-Evalue	score	bias	from	to	from	to	from	to	acc
unplaced-249_4	-	1777	xylosidase_fam_prot_align_muscle	-	680	3.4e-166	552.6	0.0	1	1	1.3e-168	4.3e-166	552.2	0.0	48	667	5	663	1	685	0.94
unplaced-537_1	-	14428	xylosidase_fam_prot_align_muscle	-	680	1.5e-164	547.2	0.0	1	1	7.1e-167	2.4e-164	546.5	0.0	14	665	3606	4398	3553	4435	0.92
unplaced-468_1	-	33336	xylosidase_fam_prot_align_muscle	-	680	1.3e-147	491.2	0.0	1	1	6.1e-150	2.1e-147	490.5	0.0	27	665	6741	7450	6716	7489	0.91
unplaced-41_3	-	22500	xylosidase_fam_prot_align_muscle	-	680	3.8e-144	479.7	6.4	1	1	1.8e-146	6.1e-144	479.0	6.4	15	676	15651	16455	15601	16475	0.90
unplaced-81_2	-	11204	xylosidase_fam_prot_align_muscle	-	680	4.7e-142	472.8	0.0	1	1	2.6e-144	8.6e-142	471.9	0.0	32	665	4169	4867	4156	4894	0.91
unplaced-373_4	-	1911	xylosidase_fam_prot_align_muscle	-	680	2.3e-78	262.3	9.8	1	2	2.1e-39	7.2e-37	125.2	5.9	335	665	478	864	455	902	0.76
unplaced-373_4	-	1911	xylosidase_fam_prot_align_muscle	-	680	2.3e-78	262.3	9.8	2	2	1.6e-43	5.4e-41	138.8	0.1	30	352	977	1296	921	1317	0.77
unplaced-253_1	-	30848	xylosidase_fam_prot_align_muscle	-	680	3.3e-65	218.8	21.4	1	1	1.4e-65	4.8e-63	211.7	21.4	79	676	23068	23941	23026	23945	0.84
unplaced-109_4	-	1132	xylosidase_fam_prot_align_muscle	-	680	1.8e-25	87.5	0.0	1	1	9.2e-28	3.1e-25	86.7	0.0	27	487	491	1029	487	1067	0.68
unplaced-349_4	-	973	xylosidase_fam_prot_align_muscle	-	680	5e-25	86.1	1.8	1	1	2.1e-27	7e-25	85.6	1.8	41	433	312	592	290	624	0.83
unplaced-72_3	-	3692	xylosidase_fam_prot_align_muscle	-	680	2.7e-14	50.6	0.0	1	2	2	6.9e+02	-3.6	0.0	82	157	1147	1230	1089	1245	0.71
unplaced-72_3	-	3692	xylosidase_fam_prot_align_muscle	-	680	2.7e-14	50.6	0.0	2	2	1.7e-16	5.7e-14	49.5	0.0	34	321	2097	2438	2034	2476	0.71

FIGURA 13. Output de HMMR3 para MUSCLE. Regiones candidatas identificades a partir de la búsqueda de perfiles HMM construidos con el alineamiento múltiple de las proteínas xylosidase utilizando MUSCLE.

#						--- full sequence ---					--- this domain ---					hmm coord		ali coord		env coord		
# target name	accession	tlen	query name	accession	qlen	E-value	score	bias	#	of	c-Evalue	i-Evalue	score	bias	from	to	from	to	from	to	acc
unplaced-360_5	-	29379	xylosidase_fam_prot_align_clustalo	-	434	2.1e-49	166.5	2.4	1	1	1.5e-51	4.1e-49	165.5	2.4	21	358	8259	8782	8180	8783	0.72
unplaced-113_3	-	18251	xylosidase_fam_prot_align_clustalo	-	434	4.9e-44	148.8	1.4	1	1	2.9e-46	8.1e-44	148.0	1.4	23	357	17721	18233	17670	18236	0.72
unplaced-41_3	-	22500	xylosidase_fam_prot_align_clustalo	-	434	1.4e-42	144.0	0.7	1	1	8.3e-45	2.3e-42	143.3	0.7	25	405	15684	16450	15622	16564	0.70
unplaced-249_4	-	1777	xylosidase_fam_prot_align_clustalo	-	434	3.2e-41	139.5	0.0	1	1	1.6e-43	4.3e-41	139.1	0.0	27	405	10	667	3	687	0.77
unplaced-537_1	-	14428	xylosidase_fam_prot_align_clustalo	-	434	1.3e-36	124.3	0.0	1	1	9.5e-39	2.6e-36	123.3	0.0	22	402	3721	4396	3635	4478	0.75
unplaced-468_1	-	33336	xylosidase_fam_prot_align_clustalo	-	434	3.4e-32	109.7	0.0	1	1	2.2e-34	6e-32	108.9	0.0	29	376	6815	7393	6737	7458	0.70
unplaced-96_2	-	22410	xylosidase_fam_prot_align_clustalo	-	434	1.1e-29	101.5	2.2	1	2	0.00012	0.033	10.9	0.0	370	433	14694	14937	14665	14938	0.70
unplaced-96_2	-	22410	xylosidase_fam_prot_align_clustalo	-	434	1.1e-29	101.5	2.2	2	2	3.3e-28	9.2e-26	88.5	1.2	19	359	15206	15725	15140	15726	0.70
unplaced-1_4	-	3439	xylosidase_fam_prot_align_clustalo	-	434	1.5e-23	81.3	5.6	1	1	7.9e-26	2.2e-23	80.7	5.6	24	356	2959	3432	2902	3438	0.71
unplaced-253_1	-	30848	xylosidase_fam_prot_align_clustalo	-	434	1.5e-12	45.0	2.9	1	2	1.7	4.9e+02	-2.8	0.2	58	132	11279	11367	11271	11404	0.65
unplaced-253_1	-	30848	xylosidase_fam_prot_align_clustalo	-	434	1.5e-12	45.0	2.9	2	2	5.3e-15	1.5e-12	45.0	2.9	43	401	23063	23923	22994	24051	0.61
unplaced-95_2	-	11648	xylosidase_fam_prot_align_clustalo	-	434	4e-10	37.0	0.3	1	1	2.4e-12	6.7e-10	36.3	0.3	20	224	63	375	23	519	0.74
unplaced-349_4	-	973	xylosidase_fam_prot_align_clustalo	-	434	9e-10	35.9	0.0	1	2	1.4e-11	3.9e-09	33.8	0.0	379	434	310	530	225	530	0.69
unplaced-349_4	-	973	xylosidase_fam_prot_align_clustalo	-	434	9e-10	35.9	0.0	2	2	0.085	24	1.5	0.0	143	201	514	593	383	628	0.53
unplaced-360_4	-	29378	xylosidase_fam_prot_align_clustalo	-	434	2.8e-07	27.6	0.1	1	1	1.7e-09	4.8e-07	26.9	0.1	377	419	5873	6003	5777	6131	0.74

FIGURA 14. *Output* de HMMR3 para Clustal Omega. Regiones candidatas identificades a partir de la búsqueda de perfiles HMM construidos con el alineamiento múltiple de las proteínas xylosidase utilizando Clustal Omega.

Los cuatro dominios más significativos identificados a partir del alineamiento con MUSCLE también se encuentran entre los resultados obtenidos a partir del alineamiento de Clustal Omega, en estos casos ambos métodos delimitan regiones similares. Sin embargo, existen dos dominios obtenidos con Clustal, que son los que presentan un *evalue* más significativo y corresponden a la secuencias unplaced_360_5 y unplaced_113_3, que no se encuentran con MUSCLE. Estas dos regiones candidatas identificadas con HMMER también se encontraron con BLAST. De hecho los resultados

obtenidos con Clustal Omega concuerdan mejor con los obtenidos con BLAST, identificando las mismas regiones candidatas excepto las dos correspondientes a la secuencia unplaced_253_1 que no se encontró con BLAST. Un ejemplo de la correlación en los resultados se puede obtener comparando las dos figuras anteriores, FIGURA 13 Y 14, con la FIGURA 10 del apartado 2.4.2 para la familia *xylosidase* en que se mostraba la región correspondiente a la secuencia unplaced_537_1. Por tanto, la opción de Clustal Omega resulta más útil por su mayor rapidez y mejor predicción de dominios cuando se utiliza con HMMR3.

HMMR3 filtra los *hits* que son reportados en el *output*. Los *hits* por secuencia y por domino se ordenan en función de su significación estadística (*evalue*). En el *output* por secuencia (*per-target*) únicamente se reportan aquellos *hits* con un *evalue* <= 10. En el *output* por dominio (*per-domain*), que es el que estamos utilizando, para cada secuencia que haya pasado el umbral anteriormente indicado (*per-target*) se reportarán aquellos dominios con un *evalue* <= 10. Los *evalue* descritos son los que utiliza HMMER3 por defecto y son bastantes laxos, por lo que es poco probable que se pierda información en el *output*. De hecho, para identificar claras regiones candidatas se podría restringir algo más este criterio por lo que en la automatización se incluirá la opción de filtrar los resultados por un *evalue* más restrictivo.

Para procesar el output de HMMER en el terminal se ha utilizado un script de Python, parse_domtblout.py, descrito anteriormente (https://www.biostars.org/p/134579/). Se ha incluido la siguiente línea en el script y se ha configurado como ejecutable para que sea totalmente funcional en nuestro sistema:

```
#!/usr/bin/python

$ chmod +x parse_domtblout.py
```

El script parse_domtblout.py, localizado en el directorio "/home/uoc/tools" de la máquina virtual, permite simplificar el *output* de manera que se visualicen únicamente el nombre de la secuencia y las coordenadas de los dominios que cumplan con los parámetros anteriormente descritos. Además permite filtrar fácilmente por *evalue* y solapamiento (*overlapping*) el *output* de HMMR3. No obstante, la función también crea un archivo intermedio en formato txt, genomic_prot_hmm_domtbl, que se grabará en la carpeta correspondiente y servirá, en este caso, para obtener, como en el caso de BLAST, el archivo de formato bed para las regiones candidatas. Este script no será necesario en la automatización porque el *output* de HMMR3 puede manipularse fácilmente en el entorno de R.

Para crear el bed de regiones únicas (*merge*) y recuperar las secuencias fasta se han utilizado los mismos comandos que para el *output* de BLAST pero adaptándolos a los archivos y formato del *output* de HMMR3.

2.5.1 Automatización con la función cazyhmmer

Como en las fases anteriores, también se ha automatizado el análisis de homología por secuencia basado en alineamiento múltiple. Se ha creado la función *cazyhmmer* en R, cuyo script completo se muestra en el ANEXO III (apartado 7.3) con los siguientes argumentos:

```
hmmercazy(align = c("clustalo", "muscle"), evalue = 10, output = c("dense","full"))
```

- align: programa a utilizar en el alineamiento múltiple:
  - "clustalo": es la opción por defecto, se utilizará Clustal Omega.
  - "muscle": como alternativa se puede utilizar MUSCLE.
- evalue: número indicando el umbral para el *evalue*, por defecto es 10.
- output: tipo de output que retornará la función:
  - "dense": únicamente se retornarán las coordenadas de las regiones candidatas
  - "full": se retornará una tabla con más información sobre el resultado de BLAST.

De la misma manera que con BLAST se ha incorporado la posibilidad de mostrar un *output* simplificado, con las coordenadas de los dominios identificados, o un *output* más extenso con algunos de los campos que reporta HMMER y que se describen en la TABLA 3. Además, la función también ejecuta un script en bash para crear los archivos bed y fasta necesarios para utilizarse en caso de que se requiera en el último tipo de análisis de búsqueda por patrón funcional.

TABLA 3. Campos incluidos en el *ouput* de HMMR3.

1.	target_name	name of the target sequence
2.	tlen	length of the target sequence
3.	query_name	name of the query profile (hmm)
4.	qlen	length of the qury profile (hmm)
5.	E-value	E-value of the overall sequence/profile comparison
6.	score_seq	bit score of the overall sequence/profile comparison
7.	#	the domain's number (X of ndom)
8.	of	The total number of domains reported in the sequence (ndom)
9.	c-Evalue	conditional E-value
10.	i-Evalue	independent E-value
11.	evalue	expect value
12.	score_dom	bit score for this domain
13.	hmm_from	start of the alignment of this domain with respect to profile
14.	hmm_to	end of the alignment of this domain with respect to profile
15.	ali_from	start of the alignment of this domain with respect to profile
16.	ali_to	end of the alignment of this domain with respect to sequence
17	acc	measure how realiable the overall alignment is (from 0 to 1)

El *evalue* configurado por defecto es el mismo que utiliza HMMER pero se ha incorporado la posibilidad que el usuario cambie este parámetro.

Aunque es recomendable utilizar Clustal Omega para los alineamientos múltiples, también se da la posibilidad de utilizar el programa MUSCLE si se escoge esta opción en el argumento align de la función hmmercazy.

2.6 Búsqueda por patrón funcional

2.6.1 Búsqueda por perfiles Pfam

Como primera aproximación para la búsqueda por patrón funcional y aprovechando las herramientas hasta ahora utilizadas, se decidió realizar una búsqueda en la *database* pfam (http://pfam.xfam.org/) (7) de perfiles ya conocidos que pudiera escanearse posteriormente en nuestra base de datos de proteínas traducidas a partir del genoma. De hecho, esta es una aproximación muy similar a la anterior por lo que se ha estudiado incluir alguna otra alternativa como la búsqueda de patrones regulares en las secuencias pertenecientes a una determinada familia funcional que se discutirá más adelante.

En primer lugar se descargó la última versión disponible de la base de datos Pfam del siguiente link: ftp://ftp.ebi.ac.uk/pub/databases/Pfam/releases/Pfam30.0/Pfam-A.hmm.gz

A continuación, se preparó la base de datos Pfam para utilizar con el programa HMMER:

```
$ hmmpress Pfam-A.hmm
```

A partir de aquí se pueden buscar los dominios conocidos en las proteínas de la familia funcional CAZy con el comando hmmscan de HMMR3:

```
$ hmmscan -E 0.01 --domE 0.01 --domtblout xylosidase _fam_prot_pfam_0.01.tbl Pfam-A.hmm
xylosidase_fam_prot.fa
```

Los dominios Pfam se grabarán en una tabla, como la que se muestra en la FIGURA 15, con el mismo formato que el que se obtiene con hmmsearch *per domain*, donde se muestran el número de acceso del dominio y múltiples parámetros asociados, como el *evalue*.

Glyco_hydro_43	PF04616.12	288	AAB97967.1	-	538	1e-104	349.8	5.0	1	1	1.9e-108	1.6e-104	349.2	5.0	1	288	3	304	3	304	0.97	Glycosyl hydrolases family 43
DUF1349	PF07081.9	178	AAB97967.1	-	538	1.3e-07	31.4	0.0	1	1	2.7e-11	2.2e-07	30.6	0.0	37	142	379	489	368	536	0.74	Protein of unknown function (DUF1349)
Glyco_hydro_43	PF04616.12	288	AAC27699.1	-	533	1.5e-109	365.6	0.5	1	1	2.6e-113	2.1e-109	365.2	0.5	3	288	5	301	3	301	0.97	Glycosyl hydrolases family 43
DUF1349	PF07081.9	178	AAC27699.1	-	533	3.9e-05	23.3	0.4	1	1	7.9e-09	6.4e-05	22.6	0.4	41	176	379	531	366	533	0.75	Protein of unknown function (DUF1349)
Glyco_hydro_31	PF01055.24	435	AAC62251.1	-	762	5.1e-106	355.4	3.6	1	1	1.5e-109	8e-106	354.7	3.6	1	435	238	674	238	674	0.89	Glycosyl hydrolases family 31
Gal_mutarotas_2	PF13802.4	66	AAC62251.1	-	762	5.8e-18	64.8	0.5	1	1	3.5e-21	1.9e-17	63.2	0.2	1	65	157	216	157	217	0.94	Galactose mutarotase-like
Melibiase	PF02065.16	347	AAC62251.1	-	762	0.0027	16.6	0.9	1	1	4.9e-07	0.0027	16.6	0.9	50	235	272	463	267	466	0.72	Melibiase
Glyco_hydro_43	PF04616.12	288	AAC67554.1	-	328	1e-22	80.7	7.8	1	1	9.2e-26	1.5e-21	76.8	7.8	12	287	18	315	6	316	0.76	Glycosyl hydrolases family 43
Glyco_hydro_43	PF04616.12	288	AAC73374.1	-	536	1.4e-109	365.8	0.4	1	1	2.3e-113	1.8e-109	365.4	0.4	1	288	3	303	3	303	0.98	Glycosyl hydrolases family 43
DUF1349	PF07081.9	178	AAC73374.1	-	536	1e-09	38.2	0.1	1	1	2.4e-13	2e-09	37.3	0.1	49	177	390	535	367	536	0.78	Protein of unknown function (DUF1349)
Glyco_hydro_31	PF01055.24	435	AAC76680.1	-	772	5e-120	401.5	3.8	1	1	8.7e-124	7.1e-120	401.0	3.8	1	433	240	667	240	669	0.92	Glycosyl hydrolases family 31
Gal_mutarotas_2	PF13802.4	66	AAC76680.1	-	772	3.4e-16	59.2	0.1	1	1	1.5e-19	1.3e-15	57.4	0.1	1	65	159	218	159	219	0.93	Galactose mutarotase-like
Glyco_hydro_43	PF04616.12	288	AAC97375.1	-	535	2.8e-106	354.9	0.7	1	1	5e-110	4.1e-106	354.4	0.7	3	288	5	301	3	301	0.98	Glycosyl hydrolases family 43
DUF1349	PF07081.9	178	AAC97375.1	-	535	4.4e-07	29.7	0.2	1	1	9.1e-11	7.4e-07	28.9	0.2	40	177	378	532	367	533	0.77	Protein of unknown function (DUF1349)
Glyco_hydro_3	PF00933.19	319	AAC99628.1	-	861	8e-61	206.0	0.0	1	1	2.7e-64	1.4e-60	205.2	0.0	1	315	35	359	35	360	0.86	Glycosyl hydrolase family 3 N terminal domain
Glyco_hydro_3_C	PF01915.20	198	AAC99628.1	-	861	5.1e-43	147.2	0.0	1	1	1.4e-46	7.6e-43	146.6	0.0	1	198	414	648	414	648	0.88	Glycosyl hydrolase family 3 C-terminal domain
Fn3-like	PF14310.4	71	AAC99628.1	-	861	1.8e-17	63.1	0.0	1	1	7.7e-21	4.2e-17	61.9	0.0	1	71	685	754	685	754	0.95	Fibronectin type III-like domain
Glyco_hydro_3_C	PF01915.20	198	AAD13106.1	-	804	4.3e-42	144.2	0.0	1	1	1.2e-45	6.4e-42	143.6	0.0	1	198	423	656	423	656	0.85	Glycosyl hydrolase family 3 C-terminal domain
Glyco_hydro_3	PF00933.19	319	AAD13106.1	-	804	8e-33	114.0	0.0	1	1	2.3e-36	1.2e-32	113.4	0.0	85	317	134	378	95	380	0.90	Glycosyl hydrolase family 3 N terminal domain
Fn3-like	PF14310.4	71	AAD13106.1	-	804	9.8e-07	28.7	0.0	1	1	4.3e-10	2.4e-06	27.4	0.0	9	66	719	777	712	780	0.87	Fibronectin type III-like domain
Glyco_hydro_43	PF04616.12	288	AAD20247.1	-	269	1e-89	300.6	1.5	1	1	6.9e-94	1.1e-89	300.4	1.5	1	260	4	268	4	269	0.97	Glycosyl hydrolases family 43
Glyco_hydro_3	PF00933.19	319	AAD35170.1	-	778	5e-104	348.0	0.0	1	1	1.2e-107	6.8e-104	347.5	0.0	1	319	23	347	23	347	0.98	Glycosyl hydrolase family 3 N terminal domain

FIGURA 15. Extracto de los dominios Pfam identificados en las proteínas *xylosidase*.

A partir del *ouptut* anterior de la cual podemos extraer los números de acceso:

```
$ sed -e '1,3d' xylosidase_fam_prot_pfam_0.01.tbl | head -n -10 | awk '{print $2}' | sort |
uniq > xylosidase_pfam_acc.txt
```

Obtenemos los perfiles HMM de la base de datos Pfam correspondientes al listado de números de acceso de los dominios que hemos identificado en proteínas *xylosidase*:

```
$ hmmfetch -f -o xylosidase_pfam_acc.hmm Pfam-A.hmm xylosidase_pfam_acc.txt
```

Con esta subselección de perfiles HMM de Pfam escaneamos, con el comando hmmscan, las secuencias proteicas traducidas a partir del genoma.

```
$ hmmscan -E 0.01 --domE 0.01 -domtblout sga-scaffolds_prot_xylosidase_pfam.tbl xylosidase_
pfam_acc.hmm sga-scaffolds_prot.fa
```

En el caso de *glucuronoyl* hemos obtenido 3 dominios Pfam mientras que en el caso de las *xylosidases* 50. Cuando se buscan estos dominios en el genoma, se encuentran múltiples hits que cumplen los criterios de bondad (20 y 546, respectivamente). Hay que tener en cuenta que, con esta estrategia, los dominios Pfam que se han obtenido pueden ser dominios específicos que definen la familia funcional o dominios particulares de alguna de las proteínas que componen la familia por lo que, para que sea una estrategia válida, debería revisarse manualmente la descripción del dominio Pfam por el usuario antes de realizar el escaneo en el genoma de referencia. Por tanto, esta estatregia además de similar a la anterior requiere una interacción mucho mayor por parte del usuario por lo que, definitivamente, no se ha considerado útil para su automatización.

2.6.2 Búsqueda por patrones funcionales PROSITE

En las secuencias de proteínas que pertenecen a la misma familia, existen regiones que se han conservado mejor a lo largo de la evolución. Estas regiones generalmente tienen una función importante para la proteína y/o para su estructura tridimensional. Analizando las propiedades constantes y variables de estos grupos de secuencias, es posible crear una “firma” o patrón de esa familia proteica o domino que distingue a sus miembros de otras proteínas no relacionados. PROSITE actualmente contiene patrones y perfiles para más de mil familias proteicas o dominios. Cada uno de estos patrones está documentados y proporcionan información relacionada con la estructura y función de estas proteínas. Por esa razón se ha decidio utilizar como referencia la base de datos PROSITE como segunda aproximación para la búsqueda por patrón funcional.

Para realizar la búsqueda se ha utilizado una instalación local de la herramienta ScanProsite (ps_scan) y la base de datos PROSITE (prosite.dat).

El primer paso es buscar patrones funcionales en nuestro grupo de proteínas que pertenecen a la misma familia funcional. Utilizaremos como ejemplo la familia *xylosidase*. El programa utilizado no incorpora prácticamente opciones de filtrado por parámetros que evalúan la significación estadística,

en casos anteriores, el *evalue*. Sin embargo se puede incluir la opción (*flag*) "-s" que omite los *hits* que ocurren frecuentemente de manera inespecífica con determinados patrones y perfiles.

```
perl /home/uoc/ps_scan/ps_scan.pl xylosidase_fam_prot.fa -d $PROSITE -o pff -s > xylosidase_fam_prot_prosite
```

En total se encuentran 74 patrones, aunque algunos están repetidos, ya que el mismo patrón puede encontrarse en distintas proteínas de la misma familia. Se crea una lista con los IDs de PROSITE de los patrones tal y como se describe a continuación para poderla utilizar posteriormente para buscar los mismos motivos en nuestra secuencia genómica de referencia.

```
$ awk '{print $4}' xylosidase_fam_prot_prosite | sort | uniq | sed -e 's/^/-p /'
> xylosidase_fam_prot_prosite_uniq
```

Se encontraron 27 patrones únicos en la familia CAZy de *xylosidases*. Cabe destacar que el proceso de búsqueda de patrones realizado por ps_scan es muy lento, tardando en este caso alrededor de 25 minutos. Generalmente, la búsqueda en secuencias de patrones o motivos de pequeño tamaño es un proceso lento y computacionalmente costoso, punto que se debe tener en cuenta antes de iniciar este tipo de análisis.

Por las razones comentadas anteriormente, se ha creído más conveniente buscar estos patrones en el *subset* de regiones definidas por los análisis de BLAST o HMMR3. Realizar la búsqueda en toda la secuencia genómica de referencia conllevaría dos inconvenientes:

- Múltiples *hits* inespecíficos debido al pequeño tamaño de los motivos y el gran tamaño de nuestras secuencia de referencia.
- Tiempo de ejecución excesivo y gran requerimiento computacional.

La estrategia de búsqueda en las regiones definidas por los análisis anteriores no nos servirá para identificar nuevas regiones candidatas pero si para aportar nueva información sobre esas regiones, como la presencia de patrones funcionales. Todos estos elementos permitirán tomar una mejor decisión sobre la idoneidad o no de la región candidata.

Para realizar la búsqueda de los patrones identificados en la familia funcional utilizaremos los archivos fasta recuperados para las regiones candidatas únicas tal y como se describe en los apartados 2.4.2 y 2.5.

```
perl /home/uoc/ps_scan/ps_scan.pl xylosidase_fam_prot_sga_blastp_uniq_merge.fa $(paste -s -d " " xylosidase_fam_prot_prosite_uniq) -o pff -s -d $PROSITE
```

Un resumen de los resultados indica que se han identificado 8 patrones en 13 regiones candidatas previamente identificadas por BLAST.

En la FIGURA 16 se muestra los patrones identificados en nuestras regiones candidatas. En la primera columna se muestra el nombre de la secuencia y el inicio y final de la región candidata identificada

por BLAST. En la tercera y cuarte columna se muestra la posición del patrón en la región candidata y la quinta columna representa el ID de PROSITE.

unplaced-1_5:2309-2778	340	469	PS51175
unplaced-249_4:2-680	188	205	PS00775
unplaced-3_6:1668-1842	120	162	PS00041
unplaced-3_6:1668-1842	70	168	PS01124
unplaced-405_1:7045-7221	89	176	PS50231
unplaced-4_1:7963-8734	1	36	PS00430
unplaced-41_3:15645-16453	228	245	PS00775
unplaced-468_1:6776-7449	214	231	PS00775
unplaced-483_2:18044-18327	181	279	PS01124
unplaced-537_1:3627-4423	279	296	PS00775
unplaced-68_2:3877-4343	356	364	PS00572
unplaced-68_2:3877-4343	5	19	PS00653
unplaced-96_2:14728-15194	328	466	PS51175

FIGURA 16. Patrones PROSITE encontrados en las regiones candidatas a pertenecer a *xylosidases* identificadas por BLAST.

Si realizamos una búsqueda de estos patrones en la página web de PROSITE podemos obtener una descripción del motivo y relacionarlo con la función de la familia CAZy. Este punto se discutirá más extenso en el apartado de comprobación de la automatización donde se ha utilizado el mismo *dataset* que para crear el *pipeline*.

2.6.3 Automatización con la función patterncazy

Finalmente se ha creado una función en R que realiza la búsqueda de patrones funcionales en PROSITE. El script completo de la función se muestra en el ANEXO III (apartado 7.3).

```
patterncazy (type= c("blast","hmmer"))
```

Esta es la función más sencilla ya que previamente se han creado los archivos necesarios para la búsqueda y, además, el programa utilizado es poco flexible en cuanto a la configuración de las opciones. Se ha preparado el sistema para que pueda ejecutarse sin problemas, incorporando, por ejemplo, la *database* de PROSITE en el directorio del programa ps_scan. La función *patterncazy* únicamente tiene un argumento:

- type: análisis a escoger sobre el *output* del cual se realizará la búsqueda de patrones:
    - blast: se utilizaran las secuencias fasta de las regiones candidatas identificadas por BLAST.
    - hmmer: se utilizaran las secuencias fasta de las regiones candidatas identificadas por HMMER.

3 Comprobación del *workflow*

3.1 Paquete cazypredict

3.1.1 Creación del paquete

Para facilitar la ejecución de las distintas funciones se ha creado el paquete "cazypredict" que se adjunta con memoria. El paquete se ha creado con Rstudio y se incluyen las cuatro funciones descritas en esta memoria, el script completo se puede consultar en el ANEXO III (apartado 7.3). Es un paquete de uso local, a utilizar con la máquina virtual que se creó para este proyecto, por lo que la documentación es la estrictamente necesaria para poder ejecutarse sin problemas en este contexto. Las funciones incluidas en el paquete invocan distintos comandos de sistema para ejecutar programas como BLAST, HMMER, bedtools o samtools instalados localmente. Además, utiliza scripts en otros lenguajes, como en bash .sh, creados específicamente para realizar el *workflow* y genera archivos intermedios por lo que es recomendable fijar previamente un directorio de trabajo. El paquete ya se ha instalado en la máquina virtual por lo que puede utilizarse ejecutando el comando library(cazypredict) una vez iniciada la sesión en R.

En el paquete se incluye la base de datos CAZy, cuya preparación se describe a continuación:

```
dbCAZy <- read.table("~/Desktop/R_pipeline/CAZy_DB_02-11-2016_parsed.txt", sep = "\t",
header = FALSE, fill = TRUE, quote = NULL)

colnames(dbCAZy) <- c("protein_name","family","tag","ec","genbank","uniprot","subfamily",
"organism","pdb")

dbCAZy <- data.frame(lapply(dbCAZy, as.character), stringsAsFactors=FALSE)

save(dbCAZy, file="data/dbCAZy.RData")
```

La base de datos se grabó en la carpeta data del paquete y se configuró el archivo DESCRIPTION para que la base de datos se cargue automáticamente con el paquete añadiendo la siguiente línea:

```
LazyData: TRUE
```

A continuación se describe la configuración del entorno de R para utilizar el paquete cazypredict. El primer comando es para configurar la variable entorno PATH ya que no se ha podido fijar esta variable permanentemente en la sesión de RStudio, aunque si es correcta en la consola de R que se utiliza desde el terminal. El segundo comando sirve para configurar el directorio de trabajo.

```
Sys.setenv(PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games:/snap/bin:/home/uoc/ncbi-blast-2.5.0+/bin:/home/uoc/hmmer-
3.1b2/binaries:/home/uoc/edirect:/home/uoc/ps_scan:/home/uoc/ps_scan")
setwd("~/Desktop/Cazy_pipeline")
```

Finalmente cargamos el paquete "cazypredict" que ha sido instalado previamente:

```
library(cazypredict)
```

En la máquina virtual, además de los programas instalados, se incluyen los archivos necesarios e imprescindibles para utilizar el paquete y otros complementarios:

- Scripts para creación de archivos bed, merge.bed y recuperación de secuencias fasta a partir de los outputs de BLAST y HMMR3:
    - /home/uoc/Desktop/Scripts/getfasta_blast.sh
    - /home/uoc/Desktop/Scripts/getfasta_hmmer.sh
- Base de datos PROSITE en/home/uoc/Desktop/ps_scan
- Copia del archivo fasta de la secuencia de referencia que se ha utilizado para crear el pipeline y como ejemplo del workflow en /home/uoc/Desktop/Data/scaffolds/sga-scaffolds.fa. Hay que tener en cuenta que para realizar todo el *workflow* el usuario deberá grabar su secuencia en la máquina virtual y proporcionar la ruta y el nombre del archivo en la función correspondiente.
- Copia de la base de datos CAZy en /home/uoc/Desktop/Data/cazy_db/CAZy_DB_02-11-2016_parsed.txt
- Copia del paquete cazypredict en /home/uoc/cazypredict
- Aunque no se haya utilizado en el paquete se guarda la base de datos Pfam en /home/uoc/hmmer-3.1b2/database/Pfam-A.hmm

3.1.2 Predicción de regiones candidatas con cazypredict

A continuación se muestra como ejemplo de utilización del paquete cazypredict en la predicción de regiones candidatas para la familia funcional *xylosidase*, documentado previamente para el diseño del pipeline.

Primero utilizamos la función de búsqueda y recuperación de secuencias fasta. Buscaremos las proteínas que pertenezcan a la familia *xylosidase*. Especificamos que queremos realizar la búsqueda por nombre de la proteína y no por ID de familia. Para realizar la búsqueda por ID de familia CAZy, del tipo GH1 o GT5 como se especifica en la TABLA 1 del apartado 2.2, se debería utilizar la opción de la función type=”FAM”. En este caso no concretaremos el organismo al que queremos que pertenezcan las proteínas.

```
searchcazy(pattern = "xylosidase" ,type = "PROT")
```

```
## [1] "285 sequences retreived"
```

Con la función anterior se han recuperado 285 secuencias fasta correspondientes a los números de acceso que se han identificado en la base de datos CAZy con los parámetros indicados en la función.

A continuación realizamos el BLAST. Como primer argumento, debemos indicar la ruta del archivo de la secuencia genómica que queremos utilizar y que se habrá guardado previamente en la máquina virtual. Los otros argumentos serían opcionales, ya que hemos introducido los valores por defecto:
evalue = 0.01

identity = 0
coverage = 0
hits = "uniq"
output ="dense"

Como prevemos que se obtendrá un gran número de *hits*, ya que tenemos 285 secuencias de referencia, filtramos por una identidad mayor al 30%, un *coverage* mayor al 80% y escogemos la opción de "uniq" en la que obtendremos un único *hit* para cada una de las proteínas de la familia CAZy. En este caso para simplificar el output del ejemplo, utilizaremos la opción "dense".

```
blastcazy("/home/uoc/Desktop/Data/scaffolds/sga-scaffolds.fa", identity=30, coverage = 80,
hits = "uniq")
```

```
##          sequence start_candidate_region end_candidate_region
## 1  unplaced-112_2                  4812                5538
## 2  unplaced-113_3                 17709               18235
## 3    unplaced-1_4                  2959                3439
## 4    unplaced-1_5                  2309                2778
## 5  unplaced-229_3                  1933                2600
## 6  unplaced-238_2                 38153               38455
## 7  unplaced-238_2                 38458               38750
## 8  unplaced-238_3                 31791               32186
## 9  unplaced-249_4                     2                 680
## 10 unplaced-349_4                   299                 627
## 11 unplaced-359_3                  1900                2275
## 12 unplaced-360_4                  5871                6343
## 13 unplaced-360_5                  8260                8783
## 14   unplaced-4_1                  7963                8734
## 15  unplaced-41_3                 15653               16453
## 16 unplaced-468_1                  6819                7449
## 17  unplaced-48_6                 18341               18759
## 18 unplaced-537_1                  3627                4423
## 19  unplaced-68_2                  3877                4343
## 20  unplaced-95_2                    69                 378
```

En el *output* simplificado todas las regiones solapantes identificadas se condensan en una sola región. En el *output* completo obtendríamos un total de 267 hits y podríamos observar información detallada de cada uno de ellos.

Probaremos utilizando el análisis de búsqueda de perfiles HMM. En este caso, escogemos el alineamiento de Clustal Omega (para escoger MUSCLE debemos introducir la opción align=”muscle”) y filtramos por un *evalue* de dominio (*c-evalue*) de 1e-5. De nuevo escogemos la opción “dense” para simplificar el output.

```
hmmercazy(align = "clustalo", evalue = 1e-5, output = "dense")
```

```
##          sequence start_candidate_region end_candidate_region
## 1  unplaced-249_4                    23                 663
## 2  unplaced-349_4                   304                 463
## 3  unplaced-360_4                  5863                6032
## 4  unplaced-360_5                  8257                8702
## 5  unplaced-373_4                   596                 866
## 6  unplaced-373_4                  1024                1192
## 7   unplaced-41_3                 15676               16430
## 8  unplaced-537_1                  3739                4394
## 9   unplaced-96_2                 14727               14904
## 10  unplaced-96_2                 15217               15410
```

Si nos fijamos en los *outputs* de los dos análisis anteriores observamos que ambas estrategias predicen, en algunos casos, regiones candidatas similares. Como en el caso anterior, en el *output* simplificado se condensan, si las hubiera, las regiones solapantes. Para obtener más información utilizamos la opción "full".

```
hmmercazy(align = "clustalo", evalue = 1e-5, output = "full")
```

```
##       target_name  tlen     query_name qlen E-value_seq score_seq # of
## 1  unplaced-249_4  1777 list_acc_align  420     1.2e-56     190.5 1  1
## 2  unplaced-537_1 14428 list_acc_align  420     2.7e-54     182.8 1  1
## 3   unplaced-41_3 22500 list_acc_align  420     3.8e-54     182.3 1  1
## 4  unplaced-360_5 29379 list_acc_align  420     9.0e-33     111.9 1  1
## 5   unplaced-96_2 22410 list_acc_align  420     1.9e-24      84.5 1  2
## 6   unplaced-96_2 22410 list_acc_align  420     1.9e-24      84.5 2  2
## 7  unplaced-373_4  1911 list_acc_align  420     1.6e-23      81.4 1  2
## 8  unplaced-373_4  1911 list_acc_align  420     1.6e-23      81.4 2  2
## 9  unplaced-349_4   973 list_acc_align  420     4.2e-21      73.5 1  1
## 10 unplaced-360_4 29378 list_acc_align  420     4.0e-16      57.0 1  1
##    c-Evalue_dom i-Evalue_dom score_dom hmm_from hmm_to ali_from ali_to
## 1       4.3e-59      1.8e-56     190.0       15    411       23    663
## 2       1.9e-56      7.9e-54     181.2       15    409     3739   4394
## 3       1.5e-56      6.3e-54     181.6        7    403    15676  16430
## 4       3.8e-35      1.6e-32     111.1       23    235     8257   8702
## 5       6.6e-18      2.8e-15      54.3      303    412    14727  14904
## 6       5.2e-11      2.2e-08      31.5       27    139    15217  15410
## 7       1.1e-13      4.6e-11      40.4      237    413      596    866
## 8       3.2e-13      1.4e-10      38.8       24    180     1024   1192
## 9       3.0e-23      1.3e-20      71.9      309    413      304    463
## 10      1.8e-18      7.5e-16      56.1      304    411     5863   6032
##     acc
## 1  0.79
## 2  0.78
## 3  0.73
## 4  0.76
## 5  0.82
## 6  0.76
## 7  0.67
## 8  0.73
## 9  0.88
## 10 0.84
```

La información detallada de cada uno de los campos de la tabla se detalla en el apartado 2.5.1.

Finalmente utilizamos la función para buscar patrones funcionales de la base de datos PROSITE en las regiones candidatas identificadas por HMMER. La búsqueda también puede realizarse en las regiones identificadas por BLAST con la opción type="blast", el *output* que se obtendría para este ejemplo se muestra en la FIGURA 16 del apartado 2.6.2. Este último paso es lento y exigente a nivel computacional:

```
patterncazy(type = "hmmer")
```

```
##               target_domain start_pattern end_pattern name_pattern
## 1     unplaced-249_4:23-663           167         184      PS00775
## 2  unplaced-373_4:1024-1192           150         167      PS00775
## 3 unplaced-41_3:15676-16430           197         214      PS00775
## 4  unplaced-537_1:3739-4394           167         184      PS00775
```

Observamos que hay un único patrón que se identifica en múltiples regiones, la mayor parte de las cuales se han identificado de manera similar con HMMER y BLAST. Si realizamos una búsqueda en la web de PROSITE confirmamos que el patrón con ID PS00775 (http://prosite.expasy.org/PS00775) corresponde al sitio activo de la familia 3 de las *glycosyl hydrolases*. Esta familia contiene dos actividades enzimáticas conocidas: β-*xylosidase* y α-L-*iduronidase*. El patrón de búsqueda por patrón funcional podría repetirse con las regiones candidatas identificadas por BLAST y comparar los resultados.

Todos los archivos generados en la comprobación se han guardado en la carpeta /home/uoc/ Desktop/Cazy_pipeline.

3.2 Aplicación Shiny cazypredict_app

3.2.1 Creación de la aplicación

Además del paquete cazypredict se ha creado una aplicación Shiny para poder realizar la predicción de una manera más interactiva con el usuario. El paquete cazypredict, que ya contiene la base de datos CAZy, se ha cargado en el servidor de la aplicación. Además en el servidor se incluye la configuración del entorno de R, como la variable PATH, tal y como se especifica en el apartado 3.1.1. De la misma manera que el paquete, la aplicación está destinada a ejecutarse en la máquina virtual creada específicamente para el proyecto ya que utiliza múltiples comandos de sistema y scripts que ejecutan programas instalados localmente.

En el ANEXO IV se adjuntan los scripts de la aplicación server.R y iu.R de la aplicación cuyo directorio es /home/uoc/Desktop/R_shiny/cazypredict_app en la máquina virtual. La carpeta que contiene estos dos archivos también se adjunta con la memoria. Un pequeño inconveniente de la aplicación es que, a diferencia del paquete, los archivos intermedios se grabarán siempre en el directorio de ejecución de la aplicación o en el directorio de trabajo que esté configurado en el servidor, en este caso /home/uoc/Desktop/R_shiny/shiny_demo. Si se quiere recuperar los archivos intermedios deberán copiarse en otro directorio antes de ejecutar un nuevo análisis o cambiar el directorio de trabajo en el servidor server.R con la función setwd() tal y como se muestra en el script del apartado 7.4.1.

Para ejecutar la aplicación se debe cargar la librería shiny en RStudio y utilizar la función de R runApp especificando el directorio de la aplicación:

```
> library(shiny)

> runApp("/home/uoc/Desktop/R_shiny/cazypredict_app")
```

La interfaz se divide en dos regiones: la interfaz de usuario a la izquierda, donde se ubican los diferentes apartados para introducir los parámetros del análisis y los botones correspondientes para ejecutarlos, y la interfaz del output a la derecha donde se mostrarán los resultados.

La aplicación es muy sencilla y, aunque es funcional, no se ha previsto posibles errores que puedan surgir durante su utilización por lo que debería pasar una fase más amplia de evaluación y mejora. La aplicación tampoco incorpora una salida resumen de todos los outputs de las diferentes fases de análisis.

3.2.2 Predicción de regiones candidatas con cazypredict_app

A continuación se describirá un ejemplo de utilización de la aplicación en Shiny con el mismo *dataset* y parámetros que en el caso del paquete cazypredict y con el que se obtiene los mismos resultados.

En primer lugar, se debe introducir el nombre de la familia funcional CAZy para la cual queremos realizar los análisis y escoger si éste se refiere al ID de la familia CAZy o a la descripción de la proteína. Si clicamos en "organism" aparecerá un espacio donde podemos introducir el género o especie al que queramos que pertenezca nuestra familia de interés. Una vez configurado el input clicamos en el botón 1 para recuperar las secuencias fasta.

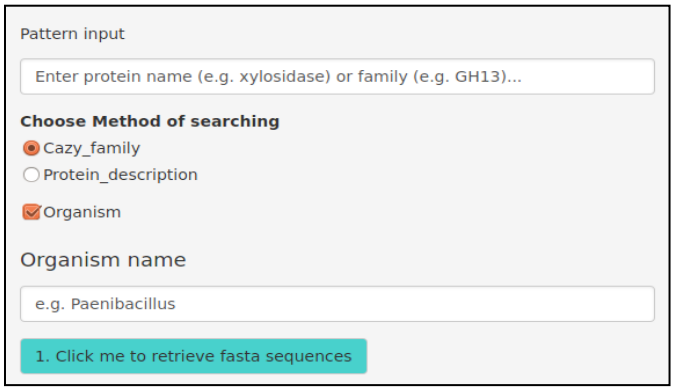


FIGURA 17. Configuración para la recuperación de las secuencias fasta con la aplicación con cazypredict_app.

En el ejemplo con la familia *xylosidase* no utilizaremos la opción de "organism". Una vez hayamos clicado en el botón 1 aparecerá un texto indicando el número de secuencias recuperadas:

FIGURA 18. Recuperación de las secuencias fasta para la familia de las *xylosidase* con cazypredict_app.

En el siguiente apartado introduciremos los parámetros para el análisis con BLAST en el apartado correspondiente de la interfaz de usuario, tal y como muestra la FIGURA 19. Como campo obligatorio debemos indicar la ruta y archivo que contiene la secuencia genómica en formato fasta. Las opciones de identidad, *coverage* y *evalue* están configuradas por defecto pero el usuario puede modificarlas según su criterio. Para configurar el *evalue* debemos escoger uno de una lista limitada de valores. Además, se puede escoger entre las dos opciones para el número de *hits* y los dos tipos de outputs que ya se describieron en el apartado 2.4.2. Una vez configuradas las distintas opciones se debe clicar el botón 2. A continuación se muestro un ejemplo con los parámetros utilizados para la familia *xylosidase*. Para facilitar la visualización de los resultados en la memoria se ha escogido el *output* "dense".

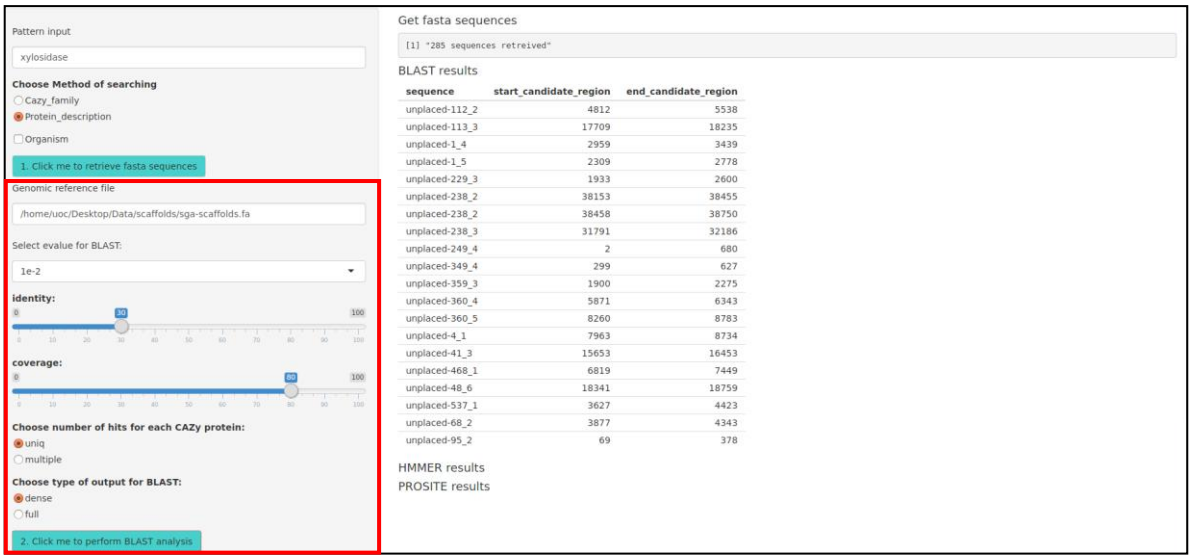


FIGURA 19. Análisis con BLAST con cazypredict_app. En el ejemplo se muestra el análisis para las proteínas *xylosidase* y la secuencia genómica de P*aenibacillus barcinonensis* En rojo se ha marcado el área de la interfaz donde se debe configurar los parámetros.

En el apartado de HMMER debemos escoger el programa para el alineamiento múltiple, el *evalue* de filtrado y el tipo de *output* que queremos visualizar tal y como se muestra en la FIGURA 20. Para nuestro *dataset* utilizaremos como programa de alineamiento Clustal Omega, un *evalue* relativamente restrictivo de 1e-5 y, de nuevo, optaremos por el *output* "dense". Como en el caso de BLAST el *ouput* compacto retornará la secuencia y coordenadas de las regiones únicas, no solapantes, candidatas a pertenecer a la familia de interés y que se han identificado con el análisis y parámetros establecidos.

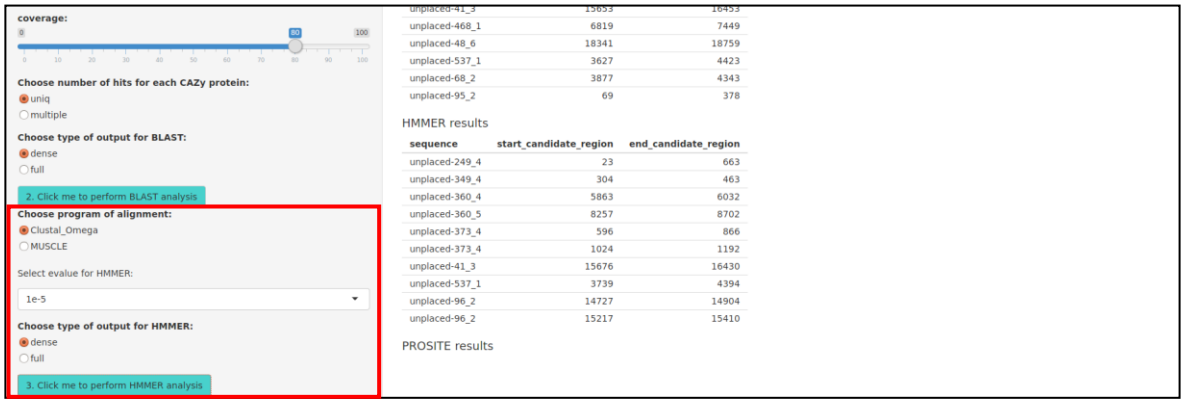


FIGURA 20. Análisis con HMMER con cazypredict_app. En el ejemplos se muestra el análisis para las proteínas *xylosidase* y la secuencia genómica de *Paenibacillus barcinonensis*. En rojo se ha marcado el área de la interfaz donde se debe configurar los parámetros.

Se puede volver a realizar el análisis reconfigurando alguna de las opciones, en este caso cambiaremos del output "dense" a "full" y clicando de nuevo el botón 2. La interfaz del *output* se actualizará con los nuevos resultados como se muestra en la FIGURA 21. La descripción de los diferentes campos del output se especifica en la TABLA 2 del apartado 2.5.1.

HMMER results

target_name	tlen	query_name	qlen	E-value_seq	score_seq	#	of	c-Evalue_dom	i-Evalue_dom	score_dom	hmm_from	hmm_to	ali_from	ali_to	acc
unplaced-249_4	1777	list_acc_align	420	0.00	190.50	1	1	0.00	0.00	190.00	15	411	23	663	0.79
unplaced-537_1	14428	list_acc_align	420	0.00	182.80	1	1	0.00	0.00	181.20	15	409	3739	4394	0.78
unplaced-41_3	22500	list_acc_align	420	0.00	182.30	1	1	0.00	0.00	181.60	7	403	15676	16430	0.73
unplaced-360_5	29379	list_acc_align	420	0.00	111.90	1	1	0.00	0.00	111.10	23	235	8257	8702	0.76
unplaced-96_2	22410	list_acc_align	420	0.00	84.50	1	2	0.00	0.00	54.30	303	412	14727	14904	0.82
unplaced-96_2	22410	list_acc_align	420	0.00	84.50	2	2	0.00	0.00	31.50	27	139	15217	15410	0.76
unplaced-373_4	1911	list_acc_align	420	0.00	81.40	1	2	0.00	0.00	40.40	237	413	596	866	0.67
unplaced-373_4	1911	list_acc_align	420	0.00	81.40	2	2	0.00	0.00	38.80	24	180	1024	1192	0.73
unplaced-349_4	973	list_acc_align	420	0.00	73.50	1	1	0.00	0.00	71.90	309	413	304	463	0.88
unplaced-360_4	29378	list_acc_align	420	0.00	57.00	1	1	0.00	0.00	56.10	304	411	5863	6032	0.84

FIGURA 21. *Output* "full" de HMMER en cazypredict_app. En el ejemplos se muestra el análisis para las proteínas *xylosidase* y la secuencia genómica de *Paenibacillus barcinonensis*.

Como se observa en la figura anterior, los diferentes valores numéricos se redondean con dos decimales y no se puede apreciar correctamente los *evalue* y *cvalues* específicos de dominio. Si el *output* se hubiera procesado de otra manera, por ejemplo, se hubiera generado una salida de texto con renderPrint o se configurase el formato de la tabla de una manera más específica estos valores posiblemente se hubieran visualizado de manera correcta. La aplicación Shiny creada tiene, por tanto, mucha margen de mejora. Este punto se discutirá en el apartado 4 de conclusiones.

El último apartado de la interfaz de usuario está dedicado a la búsqueda por patrón funcional. Como se comentó previamente, este análisis requiere de tiempo, alrededor de 30 minutos pero variable en función de las características de las secuencias, y es exigente a nivel computacional. Al no haber introducido puntos de control en la aplicación es difícil comprobar si el proceso se está llevando a cabo correctamente hasta que no finaliza. En este apartado únicamente disponemos de dos opciones, realizar la búsqueda en las regiones candidatas identificadas por BLAST o por HMMER tal y como muestra la FIGURA 22. En el siguiente ejemplo se muestra la configuración y el output para la búsqueda por patrón funcional en las regiones candidatas de la secuencia genómica identificadas con HMMER a pertenecer a la familia de las *xylosidase*.

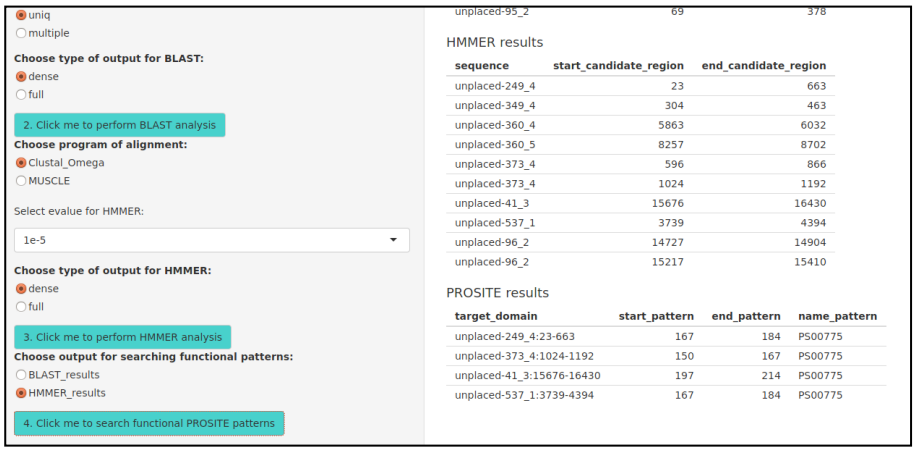


FIGURA 22. Búsqueda por patrón funcional con la cazipredict_app. La búsqueda se ha realizado en las regiones candidatas de la secuencia genómica identificadas con HMMER a pertenecer a la familia de las xylosidase

Como se había comentado en los resultados con el paquete cazypredict, se identifica un patrón en cuatro de las regiones candidatas, PS00775, que se relaciona estrechamente con la actividad enzimática de las *xylosidases*. Estos segmentos de secuencia son claros candidatos a pertenecer a la familia funcional. Para localizarlos en la secuencia genómica de DNA únicamente debe multiplicarse por tres las coordenadas indicadas ya que la posición de inicio y final de la región en la secuencia se refiere a la proteína. Tres de las regiones candidatas identificadas con HMMR3, en las que se ha identificado el patrón, también se identificaron de manera similar con BLAST, hecho que reforzaría su potencial a pertenecer a la familia de interés. La búsqueda por patrón funcional también podría realizarse con las regiones identificadas con BLAST para incrementar la información que nos permita o bien refinar la predicción o bien aumentar la lista de posibles candidatos.

Al finalizar el análisis, visualizaremos en el interfaz de *output* las salidas de todos los análisis facilitando la comparación de todos los resultados. De esta manera, se podrán reajustar los

parámetros de configuración a criterio del usuario para intentar refinar la selección de regiones candidatas.

Un punto importante a tener en cuenta en la utilización de la aplicación Shiny es que los análisis deben realizarse secuencialmente: botón 1 > botón 2 > botón 3 > botón 4. Aunque cualquiera de los análisis puede realizarse múltiples veces con diferentes configuraciones sin tener que empezar desde el inicio, se debe haber ejecutado como mínimo una vez el paso que le precede. Por ejemplo, para realizar el análisis con BLAST (botón 2) se deben haber recuperado previamente las secuencias fasta (botón 1). Una vez se hayan recuperado las secuencias fasta, el análisis de BLAST se puede repetir múltiples veces con distintos parámetros sin tener que volver a realizar el primer paso. La aplicación no ejecutará ningún análisis hasta que se clique alguno de los botones.

4 Conclusiones

En este proyecto se ha diseñado un *pipeline* para la predicción de familias funcionales CAZy en una secuencia de DNA genómica y se ha automatizado mediante la creación de funciones en R, agrupadas en el paquete cazypredict, destinadas a ejecutarse en una máquina virtual configurada específicamente para este objetivo. Además se ha creado una aplicación en Shiny para realizar la predicción de manera más interactiva.

El proyecto ha supuesto un gran reto para mí en cuanto el diseño y automatización aunque, debido a esto, he podido adquirir competencias en:

- Crear un pipeline desde su inicio y diseñar y llevar a cabo diversas diferentes fases de análisis obteniendo resultados satisfactorios en relación al objetivo planteado. Los puntos más exigentes pero que, a la vez, me han reportado un mayor conocimiento en esta área son los siguientes:
    - Manipulación de los datos, especialmente recuperación de la base de datos CAZy.
    - Decidir las diferentes fases del análisis y buscar herramientas específicas y coherentes con el tipo de análisis escogido. Estas fases debían ser complementarias y proporcionar información útil y adicional en el *worflow*.
    - Establecer los criterios de bondad de ajuste que debían introducirse en el *pipeline* para cada fase.
    - Testar cada una de las herramientas, algunas desconocidas para mí, y modificar el pipeline en función de su utilidad, del formato del *output*, de las opciones de configuración o en previsión de su automatización.
- Automatizar un *pipeline* que pudiera ejecutarse de manera seriada, relativamente fácil y sencilla. Los puntos que más han aportado en mis conocimientos en Bionformática son los siguientes:
    - Automatización del pipeline mediante funciones en R. Aunque previamente tenía experiencia con el entorno de R, ésta era a nivel de usuario, utilizando los paquetes ya disponibles. La creación de funciones en R, por tanto, ha sido por tanto una nueva área de trabajo para mí. En el mismo sentido, la creación final del paquete cazypredict me ha dado la oportunidad de conocer y aprender este tipo de herramientas. Además para diseñar las funciones se tuvo que tener en consideración:
        - el grado de configuración que se debía permitir al usuario sin complicar excesivamente su uso.
        - el tipo de *output* que debía retornar la función y que debía ser útil en el objetivo final, en este caso predicción de regiones candidatas.

- Creación de una aplicación en Shiny. Como en el caso anterior, no tenía ninguna experiencia en la utilización de este tipo de herramienta. El aprender a utilizarla quizás fue, en un determinado punto de desarrollo del trabajo, lo que me llevó mucho más tiempo del previsto.

En cuanto a la consecución de los objetivos se ha logrado, en líneas generales, los previstos inicialmente en el plan de trabajo que en resumen son: diseño del *pipeline*, automatización y mejora de la visualización de los resultados, enfocado siempre en obtener una predicción razonable de regiones candidatas a pertenecer a una familia funcional CAZy. Sin embargo, y como se comentará al final de las conclusiones, cada uno de los objetivos alcanzados requiere de mejoras que debido a la envergadura del proyecto y los retrasos en las diferentes etapas de planificación no se han podido llevar a cabo. Existen diversos puntos a destacar en cuanto al desarrollo y éxito del proyecto:

- *Análisis incluidos en el workflow*

Finalmente, se han incluido tres tipos de análisis en el *pipeline,* descritos en los apartados 2.4, 2.5 y 2.6. Inicialmente se había planteado la posibilidad de introducir también una fase de análisis de similitud 3-D. Sin embargo, desde el primer momento esta opción se mantuvo condicionada a los tiempos de ejecución de las otras fases del proyecto ya que éste era muy ambicioso, se disponía de poca experiencia en algunos de los puntos a desarrollar y, además, el tipo de análisis planteado presentaba una difícil incorporación en la automatización. Es por esta razón que, tal y como se describió en la PEC2, se decidió descartar este tipo de análisis en una fase ya intermedia de desarrollo del trabajo.

- *Desvíos y modificaciones en la planificación inicial*

Se ha producido un retraso en la entrega de los diferentes documentos de seguimiento aunque las diferentes tareas se han ido ejecutando en el orden previsto con algunas modificaciones que se comentarán a continuación. Además del orden, algunos de los tiempos no se planificó correctamente por lo que las entregas se realizaron en fechas no previstas aunque incorporaban en todos los casos las modificaciones y actividades realizadas que se desviaban de la planificación inicial.

Por ejemplo, para la automatización del *input*, es decir la búsqueda en la base de datos CAZy que finalmente se realizó mediante la función *searchcazy*, se habían planificado únicamente dos días. Sin embargo, no se había previsto que para dicha automatización, y para recuperar secuencias fasta que nos sirvieran de *dataset* ejemplo en el diseño del *pipeline*, se debía realizar previamente, en una fase inicial del desarrollo del trabajo, las siguientes tareas:

- Recuperar la base de datos CAZy de la web
- Manipular la base de datos recuperada para que fuera accesible de manera sencilla
- Diseñar el método de búsqueda de las familias funcionales
- Diseñar el método de recuperación de las secuencias fasta

Todas estas tareas no estaban contempladas en la planificación ya que únicamente se incluyó en la planificación del *pipeline* las diferentes fases de análisis y no esta preparación previa de los datos que tardo múltiples días en diseñarse y ejecutarse y cuya dificultad ya se ha comentado en el apartado correspondiente 2.2 y 2.3. Como se puede comprobar estas tareas en conjunto constituyen una parte nada despreciable del diseño del *pipeline* y no se realizó ninguna planificación temporal para éstas.

Otro punto que ha contribuido a la desviación de la planificación temporal han sido las modificaciones que se han ido introduciendo en el diseño del *pipeline* inicialmente propuesto. Una vez decidido el diseño del *pipeline* y testado las diferentes herramientas se inició la automatización con funciones en R. Sin embargo, estos dos procesos no son excluyentes y, por tanto, uno u otro se fueron modificando con el objetivo de mejorar el resultado final. Por ejemplo, se cambió el análisis inicialmente previsto de búsqueda por perfiles Pfam por el de búsqueda por patrones funcionales PROSITE. Este cambió obligó a modificar el *pipeline* y automatización de BLAST y de HMMR3, introduciendo además nuevas herramientas en el *workflow*, provocando una nueva desviación temporal en la planificación. De la misma manera se ha tenido que ir modificando las funciones, obligando a revisar tanto el paquete como, en última instancia, la aplicación en múltiples ocasiones. Aunque no se incluyó finalmente el análisis de similitud 3-D, globalmente el tiempo invertido en realizar el proyecto ha sido más largo del que estaba previsto inicialmente en la planificación temporal. A pesar de esto desvíos, en general se ha seguido la metodología y enfoque inicialmente previsto, y para el cual me tuve que documentarme ampliamente en la redacción de la propuesta del trabajo. Este hecho ha facilitado una mejor comprensión de las fases que se iban a incluir y una correcta elección y aplicación de las herramientas escogidas para cada análisis.

No obstante, y como se comenta al inicio del trabajo, tanto el paquete como la aplicación requieren de múltiples mejoras que, aun habiendo finalizado el periodo de ejecución del trabajo, me gustaría realizar con posterioridad. Las funciones creadas posiblemente deberían incorporar más puntos de chequeo y eliminación de fallos (depuración de errores o *debugging*) para poder guiar al usuario en caso que se introduzcan los argumentos erróneos. Debido a mi poca experiencia en programación en R, este proceso no ha sido llevado a cabo aunque si se ha comprobado que todas las funciones creadas fueran completamente funcionales. Además se ha detectado algún error en contextos muy concretos, como en el comentado en el apartado 2.3.1 de la automatización con la función *searchcazy*, cuya solución no he se ha podido incorporar en el producto final por falta de tiempo. El paquete de cazypredict no está prácticamente documentado y debería introducirse en los apartados correspondientes la descripción de los argumentos de las funciones y ejemplos de uso, entre otros. De la misma manera la aplicación en Shiny es muy simple y permite múltiples mejoras como, por ejemplo, en la mejor visualización de los *outputs*, especialmente cuando se escoge la opción extensa de BLAST ya que generalmente se generan tablas con multitud de *hits*. En este sentido, sería interesante incorporar en la aplicación un *output* resumen de todos los análisis realizados. Aunque se

han testado algunos *datasets*, un uso exhaustivo del *workflow* podría poner en relieve fallos o posibles ideas de mejora del *pipeline* y/o su automatización. Por tanto, se debería realizar un análisis más profundo de su funcionamiento e incorporar las modificaciones apropiadas en los productos finales. Finalmente, creo que sería interesante realizar una guía completa del uso del paquete y la aplicación, incorporando otros ejemplos diferentes a los presentados en la memoria, y agregar en la máquina virtual un directorio que incluya toda la documentación relacionada con el proyecto.

5 Glosario

Máquina Virtual: programa que emula a un ordenador real y que, por lo tanto, dispone de disco duro y memoria ram, entre otros. Se puede instalar el sistema operativo a elección e instalar y ejecutar programas como en cualquier computadora.

Workflow (o flujo de trabajo): sistema informático que permite la automatización de procesos y agilizar los análisis a realizar.

Pipeline: procedimiento que transforma un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior

Input: información (archivos, texto, datos....) de entrada que se incorpora en proceso determinado.

Output: información (archivos, texto, datos…) de salida que genera un proceso determinado.

Fasta: formato de fichero informático basado en texto, utilizado para representar secuencias biológicas como ácidos nucleicos o proteínas en la que cada nucleótido o aminoácido está representado por una letra. Cada secuencia está representado por un ID precedido por el símbolo “>”.

Bed: formato de fichero informático basado en texto utilizado en anotación genómica. Debe contener 3 campos obligatorios: nombre de cromosoma o *scaffold*, posición de inicio del elemento y posición final del elemento.

Scaffold: Serie de *contigs* unidos en la que puede haber discontinuidades o secuencias ambiguas. Los *contigs* son reconstrucciones de secuencia que representan una región continua del genoma.

Dominio: los dominios funcionales son regiones de una proteína asociadas a una función específica, con independencia de su organización estructural, como por ejemplo el dominio de unión a un sustrato o dominio catalítico.

Patrón: expresión que caracteriza un conjunto de secuencias, indicando que posiciones son más importantes, cuáles pueden variar y qué modificaciones pueden sufrir. Determinar patrones presentes en una proteína puede ayudar a postular su función y/o estructura.

Perfil: descripción del consensus de un alineamiento múltiple de secuencias. Los perfiles pueden ser, por ejemplo, tablas indicando las probabilidades de los aminoácidos o una matrices de sustitución específica para cada posición de la secuencia (*position specific scoring matrix*, PSSM).

Hidden Markov Models (HMMs): los perfiles HMM describen los dominios probabilísticamente. Contienen estados para coincidencia, inserción o eliminación que son usados para modelar una familia de secuencias.

Pfam: una amplia colección de alineamientos múltiples de secuencias y perfiles HMM que cubre buena parte de dominios proteicos y familias comunes.

PROSITE: base de datos que contiene registros describiendo dominios, familias y sitios funcionales así como sus patrones y perfiles. Las entradas están verificadas manualmente.

Homología: la homología entre secuencias se refiere a la situación en que dos o más proteínas o ácidos nucleicos presentan un grado elevado de similitud por lo que deduce una relación ancestral común.

Alineamiento a pares: alineamiento de dos secuencias de aminoácidos o nucleótidos. Este tipo de alineamiento es la base es la base de diversas herramientas de análisis de secuencias.

Alineamiento múltiple: colección de tres o más secuencias de aminoácidos o nucleótidos parcial o completamente alineadas.

Identidad: en el contexto de alineamiento de secuencias es el porcentaje indicando el grado de coincidencia entre dos secuencias.

Coverage: en el contexto de alineamiento de secuencias indica el porcentaje de solapamiento entre dos secuencias.

Hit: elemento para el cuál se ha encontrado la mejor correspondencia en un alineamiento o búsqueda en base de datos.

CAZy (*Carbohydrate-Active enZYmes Databas*): base de datos que describe familias de módulos (o dominios funcionales) catalíticos o de unión a carbohidratos estructuralmente relacionados de enzimas que degradan, modifican o crean enlaces glucosídicos.

Command-line (o comando de sistema): es un método para manipular con instrucciones escritas el programa que subyace. Este método de comunicación entre el usuario y el sistema está basado únicamente en un input y output textual.

Bash: programa informático, cuya función consiste en interpretar órdenes, y lenguaje de programación de consola. Está basado en la shell de Unix y es compatible con POSIX.

Python: lenguaje de programación desarrollado como proyecto de código abierto. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, soporta la programación orientada a objetos y tiene una sintaxis clara y visual.

Perl: lenguaje de programación que toma características del lenguaje C, del lenguaje interpretado bourne shell, AWK, sed, Lisp y, en un grado inferior, de muchos otros lenguajes de programación.

R: sistema para computación estadística y creación de gráficos. Proporciona, entre otros, un lenguaje de programación, gráficos de alto nivel, interfaces para otros lenguajes y facilidades para la depuración de errores.

6 Bibliografía

1. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. 1990. Basic local alignment search tool. *J Mol Biol* 215:403-10
2. Bhagwat M, Aravind L. 2007. PSI-BLAST tutorial. *Methods Mol Biol* 395:177-86
3. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, et al. 2009. BLAST+: architecture and applications. *BMC Bioinformatics* 10:421
4. Cantarel BL, Coutinho PM, Rancurel C, Bernard T, Lombard V, Henrissat B. 2009. The Carbohydrate-Active EnZymes database (CAZy): an expert resource for Glycogenomics. *Nucleic Acids Res* 37:D233-8
5. Eddy SR. 2011. Accelerated Profile HMM Searches. *PLoS Comput Biol* 7:e1002195
6. Edgar RC. 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res* 32:1792-7
7. Finn RD, Coggill P, Eberhardt RY, Eddy SR, Mistry J, et al. 2016. The Pfam protein families database: towards a more sustainable future. *Nucleic Acids Res* 44:D279-85
8. Honorato RV. 2016. CAZy-parser a way to extract information from the Carbohydrate-Active enZYmes Database. *The Journal of Open Source Software* 1
9. Lombard V, Golaconda Ramulu H, Drula E, Coutinho PM, Henrissat B. 2014. The carbohydrate-active enzymes database (CAZy) in 2013. *Nucleic Acids Res* 42:D490-5
10. Olson SA. 2002. EMBOSS opens up sequence analysis. European Molecular Biology Open Software Suite. *Brief Bioinform* 3:87-91
11. Park BH, Karpinets TV, Syed MH, Leuze MR, Uberbacher EC. 2010. CAZymes Analysis Toolkit (CAT): web service for searching and analyzing carbohydrate-active enzymes in a newly sequenced organism using CAZy database. *Glycobiology* 20:1574-84
12. Sievers F, Wilm A, Dineen D, Gibson TJ, Karplus K, et al. 2011. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Mol Syst Biol* 7:539
13. Sigrist CJ, Cerutti L, Hulo N, Gattiker A, Falquet L, et al. 2002. PROSITE: a documented database using patterns and profiles as motif descriptors. *Brief Bioinform* 3:265-74
14. Soding J. 2005. Protein homology detection by HMM-HMM comparison. *Bioinformatics* 21:951-60
15. Thompson JD, Higgins DG, Gibson TJ. 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res* 22:4673-80
16. Yin Y, Mao X, Yang J, Chen X, Mao F, Xu Y. 2012. dbCAN: a web resource for automated carbohydrate-active enzyme annotation. *Nucleic Acids Res* 40:W445-51

7 Anexos

7.1 ANEXO I: Documentación de instalación

7.1.1 CAZy-parser

Cazy-parser es una herramienta que extrae información de CAZy en un formato más accesible. Primero un script procesa la estructura HTML y crea una copia de la base de datos en un fichero delimitado por tabulaciones. En segundo lugar, la información se extrae de la base de datos a partir de los parámetros que el usuario introduce y devuelve los códigos de acceso de las proteínas. Esta segunda función no se ha utilizado al ser poco flexible.

La instalación se ha realizado con el pip de Python.

```
$ sudo pip install cazy-parser
```

7.1.2 E-Utilities

Entrez Direct (EDirect) es un método avanzado para acceder a la plataforma NCBI y a sus bases de datos interconectadas desde una terminal de UNIX.

Para instalar el programa Edirect se han utilizados los siguientes commandos desde el terminal:

```
cd ~
perl -MNet::FTP -e \
  '$ftp = new Net::FTP("ftp.ncbi.nlm.nih.gov", Passive => 1);
   $ftp->login; $ftp->binary;
   $ftp->get("/entrez/entrezdirect/edirect.zip");'
unzip -u -q edirect.zip
rm edirect.zip
export PATH=$PATH:$HOME/edirect
./edirect/setup.sh
```

Con el código anterior se descargan diversos scripts en la carpeta “edirect” del directorio *home* del usuario y se modifica la variable PATH que permite la ejecución de los programas en ese directorio.

El script setup.sh se utiliza para descargar los módulos de Perl que sean necesarios y edita el archivo de configuración del usuario .bash_profile para incluir la nueva variable PATH.

7.1.3 EMBOSS

La instalación de EMBOSS se ha realizado utilizando el comando apt de Linux:

```
$ sudo apt-get install emboss
```

Esta plataforma se ha utilizado principalmente para la traducción de la secuencia a proteína.

7.1.4 BLAST

NCBI dispone de un paquete que incorpora todos los programas BLAST+ para poder ejecutarlos independientemente desde el command-line. Este paquete está disponible para diversas plataformas en el siguiente link: ftp://ftp.ncbi.nlm.nih.gov/blast/executables/LATEST/.

Bajamos la versión más reciente compatible con nuestro sistema operativo "ncbi-blast-2.5.0+-x64-linux.tar.gz"

Para instalar el programa simplemente extraemos el paquete descargado en el directorio deseado:

```
$ tar zxvpf ncbi-blast-2.5.0+-x64-linux.tar.gz
```

El nuevo directorio contiene el subdirectorio bin que contienen todos los programas de BLAST.

Para poder utilizar desde cualquier ubicación el programa debemos configurar la variable PATH. Incluimos la siguiente línea en el archivo .bashrc:

```
$ export PATH=$PATH:$HOME/ncbi-blast-2.2.29+/bin
```

7.1.5 Bedtools

La instalación de Bedtools se ha realizado utilizando el comando apt de Linux:

```
$ sudo apt-get install bedtools
```

Esta herramienta se ha utilizado principalmente para recuperar las secuencias fasta a partir de un archivo tipo bed.

7.1.6 Samtools

La instalación de Samtools se ha realizado utilizando el comando apt de Linux:

```
$ sudo apt-get install bedtools
```

Esta herramienta se ha utilizado principalmente para crear índices de los archivos fasta.

7.1.7 Clustal Omega

Se ha descargado el archivo correspondiente a la versión más reciente, clustalo-1.2.4-Ubuntu-x86_64, desde la web de Clustal Omega (http://www.clustal.org/omega).

Renombramos el archivo como clustalo y lo movemos al directorio deseado:

```
$ sudo mv /home/uoc/Downloads/clustalo /usr/bin
```

Se debe convertir el archive a ejecutable de la siguiente manera:

```
$ sudo chmod u+x /usr/bin/clustalo
```

7.1.8 MUSCLE

Se ha descargado el archivo correspondiente a la versión más reciente, muscle3.8.31_i86linux64.tar.gz, desde la web de MUSCLE:

http://www.drive5.com/muscle/downloads.htm.

Descomprimimos el archivo:

```
$ tar -xvzf  /home/uoc/Downloads/muscle3.8.31_i86linux64.tar.gz
```

Renombramos el archivo como muscle, lo movemos al directorio deseado y lo convertimos en ejecutable

```
$ sudo mv /home/uoc/Downloads/muscle /usr/bin
$ sudo chmod u+x /usr/bin/muscle
```

7.1.9 HMMER3

Descargamos el paquete con la versión más reciente, hmmer-3.1b2-linux-intel-x86_64.tar.gz, del siguiente link: http://hmmer.org/download.html

Procedemos a la instalación extrayendo el paquete en el directorio deseado e instalándolo con el comando “make”:

```
$ tar zxf hmmer-3.1b2-linux-intel-x86_64.tar.gz
$ cd hmmer-3.1b2
$ ./configure
$ make
$ make check
```

7.1.10 ps-scan

Instalamos PFTOOLS

Para poder instalar ps-scan debemos instalar previamente la herramienta PFTOOLS que descargaremos previamente del siguiente link:

ftp://ftp.lausanne.isb-sib.ch/pub/software/unix/pftools/pftoolsV3/pftoolsV3_fixed.tar.gz

Procedemos a la instalación extrayendo el paquete e instalándolo con el comando “make”

```
$ tar zxf pftoolsV3_fixed.tar.gz
$ cd pftools
$ sudo make
```

Instalamos ps-scan

Descargamos el paquete adecuado específico para nuestra plataforma, en este caso Linux de 64 bits:

ftp://ftp.expasy.org/databases/prosite/ps_scan/ps_scan_linux_x86_elf.tar.gz.tar.gz

Una vez se ha extraído el paquete en el directorio deseado, el ejecutable ps_scan.pl está listo para su uso y se localiza en el directorio "/home/uoc/ps_scan".

```
$ tar zxf ps_scan_linux_x86_elf.tar.gz
```

Descarga de PROSITE

También se necesitará una copia de la base de datos PROSITE que se descargó del link que se muestra a continuación y se guardó en el directorio de ps_scan "/home/uoc/ps_scan".

ftp://ftp.expasy.org/databases/prosite/prosite.dat

7.1.11 RSTUDIO

Instalamos R-base

En primer lugar, añadimos el repositorio de R. Para ello añadimos una línea al archivo /etc/apt/sources.list. Hay que tener en cuenta que "xenial" se refiere a la versión de Ubuntu que estamos utilizando (Ubuntu 16.04). Si se instalara en otro sistema debería cambiarse dicha referencia.

```
$ sudo echo "deb http://cran.rstudio.com/bin/linux/ubuntu xenial/" | sudo tee -a
/etc/apt/sources.list
```

Añadimos las clavers de R a Ubuntu:

```
$ gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9
$ gpg -a --export E084DAB9 | sudo apt-key add -
```

Instalamos R-base:

```
$ sudo apt-get update
$ sudo apt-get install r-base r-base-dev ok
```

Instalamos R-Studio

A partir de aquí se puede instalar R-studio a través del command-line de la siguiente manera:

```
$ sudo apt-get install gdebi-core
$ wget https://download1.rstudio.org/rstudio-0.99.903-amd64.deb
$ sudo gdebi -n rstudio-0.99.903-amd64.deb
$ rm rstudio-0.99.903-amd64.deb
```

Instalación de paquetes de R

La instalación de paquetes de R también se realizará desde el command-line de la siguiente manera:

```
$ sudo su - -c "R -e \"install.packages('shiny', repos='http://cran.rstudio.com/')\""
```

7.2 ANEXO II: Script para la creación de la copia local de la base de datos CAZy

1. Eliminamos la primera fila:

```
$ sed '1d' CAZy_DB_02-11-2016.csv > CAZy_DB_02-11-2016.txt
```

2. Hay caracteres especiales que deberán ser eliminados (*non-breaking space*, M-BM-). Lo hacemos de la siguiente manera:

```
$ sed 's/\xc2\xa0//g' CAZy_DB_02-11-2016.txt > CAZy_DB_02-11-2016_parsed.txt
```

3. Sustituimos los espacios por el cáracter "_" y organizamos cada uno de los registros en una sola línea.

```
$ sed 's/ /_/g' CAZy_DB_02-11-2016_parsed.txt | awk -F "\t" 'ORS=NR%3?" ":"\n"' > tmpfile
&& mv tmpfile CAZy_DB_02-11-2016_parsed.txt
```

4. Rellenamos los valores ausentes por NAs que están representados por espacios.

```
$ awk 'BEGIN { FS = OFS = "\t" } { for(i=1; i<=10; i++) if($i ~ /^ *$/) $i = "NA" }; 1'
CAZy_DB_02-11-2016_parsed.txt > tmpfile && mv tmpfile CAZy_DB_02-11-2016_parsed.txt
```

5. Existen 3 registros que presenten en el nombre de la proteína caracteres adicionales de separación y no se han procesado bien. Modificamos estos registros para que cumplan con el formato de la tabla:

```
$ sed -i '18619,18620s/\t//2' CAZy_DB_02-11-2016_parsed.txt

$ sed -i '18619,18620s/\t//2' CAZy_DB_02-11-2016_parsed.txt

$ sed -i '56560s/\t//2' CAZy_DB_02-11-2016_parsed.txt | sed '18619,18620s/\t//2'
```

6. Eliminamos las columnas 1 y 5 correspondientes a "domain" y "organism_code" ya que todos los valores son NA. Además sustituimos los valores ausentes de la última columna correspondiente a PDB, y que están codificados por "_", por NAs.

```
$ cut -f2-4,6-11 CAZy_DB_02-11-2016_parsed.txt | awk -F'\t' -vOFS='\t' '{gsub(/^_$/, "NA",
$9); print }' > tmpfile && mv tmpfile CAZy_DB_02-11-2016_parsed.txt
```

7.3 ANEXO III: Script de las funciones del paquete cazypredict

7.3.1 searchcazy

```
searchcazy <- function(pattern, organism, type = c("PROT", "FAM")) {
  ## evaluate choices
  type <- match.arg(type)
  if(type=="PROT")
    df <- dbCAZy[grepl(pattern, dbCAZy$protein_name)&!(grepl("XP_",dbCAZy$genbank)),]
  else if(type=="FAM")
    df <- dbCAZy[grepl(paste0("\\<",pattern,"\\>"),
dbCAZy$family)&!(grepl("XP_",dbCAZy$genbank)),]
  if(missing(organism)) {
    acc <- unique(df$genbank)
  } else {
    df_org <- df[grepl(organism, df$organism),]
    acc <- unique(df_org$genbank)
  }
  write.table(acc, file = "list_acc.txt", quote = FALSE, sep = "\n", row.names = FALSE,
col.names = FALSE)
  system ("efetch -db protein -id $(paste -s -d ',' list_acc.txt) -format fasta >
list_acc.fa")
  paste(system("grep -c '^>' list_acc.fa", intern=TRUE),"sequences retreived")
}
```

7.3.2 blastcazy

```
blastcazy <- function(filename, evalue = 0.01, identity = 0, coverage = 0, hits = c("uniq",
"multiple"), output = c("dense","full")) {
  type <- match.arg(hits)
  output <- match.arg(output)
  system(paste ("transeq -frame 6 -sequence", filename,  "-outseq genomic_prot.fa"))
  system("makeblastdb -in genomic_prot.fa -dbtype 'prot' -parse_seqids",intern=FALSE)
  if(hits=="multiple") {
    blast_out <- read.table(text = system(paste("blastp -query list_acc.fa -task 'blastp' -
db genomic_prot.fa -evalue", as.character(evalue), "-outfmt '6 std qlen qcovhsp'"), intern
= TRUE))
  }
  else if(hits=="uniq") {
    blast_out <- read.table(text = system(paste("blastp -query list_acc.fa -task 'blastp' -
db genomic_prot.fa -evalue", as.character(evalue), "-outfmt '6 std qlen qcovhsp' -
max_target_seqs 1 -max_hsps 1"), intern = TRUE))
  }
  blast_filter <- blast_out[ which( blast_out$V3 > identity & blast_out$V14 > coverage) , ]
  write.table(blast_filter, file = "list_acc_blastp.txt", quote = FALSE, sep = " ",
row.names = FALSE, col.names = FALSE)
```

```
  system("bash /home/uoc/Desktop/Scripts/getfasta_blast.sh")
  if(output=="full"){
    blast_sorted <- blast_filter[order(blast_filter[,2]),]
    colnames(blast_sorted) <-
c("query_id","subject_id","pct_identity","aln_length","n_of_mismatches","gap_openings","q_s
tart","q_end","s_start","s_end","e_value","bit_score","qlen","coverage")
  }
  else if(output=="dense"){
    blast_merged <- read.table("list_acc_blastp.merge")
    blast_sorted <- blast_merged[order(blast_merged[,1]),]
    colnames(blast_sorted) <- c("sequence","start_candidate_region","end_candidate_region")
  }
  return(blast_sorted)
}
```

7.3.3 hmmercazy

```
hmmercazy <- function(align = c("clustalo", "muscle"), evalue = 10, output =
c("dense","full")){
  align <- match.arg(align)
  output <- match.arg(output)
  if (align=="clustalo") {
    system("clustalo -i list_acc.fa -o list_acc_align.fa --force")
  }
  else if(align=="muscle") {
    system("muscle -in list_acc.fa -out list_acc_align.fa")
  }
  system2("hmmbuild", args = c("list_acc_align.hmm", "list_acc_align.fa", stdout = FALSE))
  system2("hmmsearch", args = c("--domtblout genomic_prot_hmm_domtblb", paste("--
domE",evalue), "list_acc_align.hmm", "genomic_prot.fa"), stdout = TRUE)
  system ("bash /home/uoc/Desktop/Scripts/getfasta_hmmer.sh")
  if(output=="dense") {
    hmmerdom <- read.table("genomic_prot_hmm.merge")
    colnames(hmmerdom) <- c("sequence","start_candidate_region","end_candidate_region")
  }
  else if (output=="full") {
    hmmerdom <- read.table("genomic_prot_hmm_domtbl")
    hmmerdom <- hmmerdom[,c(1,3:4,6:8,10:14,16:19,22)]
    colnames(hmmerdom) <- c("target_name","tlen","query_name","qlen", "E-
value_seq","score_seq","#",
                            "of","c-Evalue_dom","i-
Evalue_dom","score_dom","hmm_from","hmm_to","ali_from","ali_to","acc")
  }
  return(hmmerdom)
}
```

7.3.4 patterncazy

```
patterncazy <- function(type= c("blast","hmmer")) {
  type <- match.arg(type)
  system("perl /home/uoc/ps_scan/ps_scan.pl list_acc.fa -o pff -s > list_acc_prosite")
  system("awk '{print $4}' list_acc_prosite | sort | uniq | sed -e 's/^/-p /' >
list_acc_prosite_ID")
  if (type=="blast") {
    prosite <- read.table(text=(system("perl /home/uoc/ps_scan/ps_scan.pl
list_acc_blastp.fa $(paste -s -d ' ' list_acc_prosite_ID) -o pff -s", intern = TRUE)),
fill = TRUE)
  }
  else if(type=="hmmer") {
    prosite <- read.table(text=(system("perl /home/uoc/ps_scan/ps_scan.pl
genomic_prot_hmm.fa $(paste -s -d ' ' list_acc_prosite_ID) -o pff -s", intern = TRUE)),
fill = TRUE)
  }
  prosite <- prosite[,1:4]
  colnames(prosite) <- c("target_domain","start_pattern","end_pattern","name_pattern")
  return(prosite)
}
```

7.4 ANEXO IV: Scripts para la aplicación Shiny cazypredict_app

7.4.1 Script para la interfaz de usuario iu.R.

```
library(shiny)

# Define UI for application
shinyUI(fluidPage(

  # Application title
  titlePanel("CAZy family analysis"),

  # Slider Layout
  sidebarLayout(
    # Sidebar
    sidebarPanel(
      textInput("pattern", label = h5("Pattern input"), value = "Enter protein name (e.g.
xylosidase) or family (e.g. GH13)..."),
      radioButtons("type", 'Choose Method of searching',
                   c(Cazy_family="FAM",
                     Protein_description="PROT")
      ),
      checkboxInput("org", "Organism"),
      conditionalPanel(
        condition = "input.org == true",
        textInput("organismo", label = h4("Organism name"), value = "e.g. Paenibacillus")
      ),
      actionButton("button1", "1. Click me to retrieve fasta sequences",  style =
"background-color:#48D1CC"),
      textInput("filename",label=h5("Genomic reference file"), value = "Paste absolute path
to the file (e.g. /home/uoc/my_seq.fa)..."),
      selectInput("evalue1", label = h5("Select evalue for BLAST:"), choices = list("1e-
30","1e-30","1e-20","1e-10","1e-5","1e-4","1e-3","1e-2","1e-1","1","10","100"),selected
="1e-2"),
      sliderInput("identity", "identity:", min=0, max=100, step=1, value=0),
      sliderInput("coverage", "coverage:", min=0, max=100, step=1, value=0),
      radioButtons("hits", 'Choose number of hits for each CAZy protein:',
                   c(uniq="uniq",
                     multiple="multiple")
      ),
      radioButtons("outype1", 'Choose type of output for BLAST:',
                   c(dense="dense",
                     full="full")
      ),
      actionButton("button2", "2. Click me to perform BLAST analysis",  style =
"background-color:#48D1CC"),
      radioButtons("align", 'Choose program of alignment:',
                   c(Clustal_Omega="clustalo",
                     MUSCLE="muscle")
      ),
      selectInput("evalue2", label = h5("Select evalue for HMMER:"), choices = list("1e-
30","1e-30","1e-20","1e-10","1e-5","1e-4","1e-3","1e-2","1e-1","1","10","100"),selected
="1e-5"),
      radioButtons("outype2", 'Choose type of output for HMMER:',
                   c(dense="dense",
                     full="full")
      ),
      actionButton("button3", "3. Click me to perform HMMER analysis",  style =
"background-color:#48D1CC"),
      radioButtons("results", 'Choose output for searching functional patterns:',
                   c(BLAST_results="blast",
                     HMMER_results="hmmer")
```

```
      ),
      actionButton("button4", "4. Click me to search functional PROSITE patterns",  style =
"background-color:#48D1CC")
    ),

    mainPanel(
      h4("Get fasta sequences"),
      verbatimTextOutput("fasta"),
      h4("BLAST results"),
      tableOutput("blast"),
      h4("HMMER results"),
      tableOutput("hmmer"),
      h4("PROSITE results"),
      tableOutput("prosite")
    )
  )
))
```

7.4.2 Script para el servidor de cazypredict:app

```
# Configurar environment
Sys.setenv(PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games:/snap/bin:/home/uoc/ncbi-blast-2.5.0+/bin:/home/uoc/hmmer-
3.1b2/binaries:/home/uoc/edirect:/home/uoc/ps_scan:/home/uoc/ps_scan")
setwd("~/Desktop/R_shiny/shiny_demo")

# Cargar librerías
library(shiny)
library(cazypredict)

# Definir el servidor
shinyServer(function(input, output) {

  getfasta <- eventReactive(input$button1, {
      if(input$org==FALSE){
        type_arg <- input$type
        pattern_arg <- input$pattern
        cazypredict::searchcazy(pattern = pattern_arg, type = type_arg)
      }
      else if(input$org==TRUE){
        type_arg <- input$type
        pattern_arg <- input$pattern
        organism_arg <- input$organismo
        cazypredict::searchcazy(pattern = pattern_arg, type = type_arg, organism =
organism_arg)
      }
  })

    output$fasta <- renderPrint({
      getfasta()
    })

  evalue1 <- reactive({as.numeric(input$evalue1)})

  getblast <- eventReactive(input$button2, {
    filename_arg <- input$filename
    evalue1_arg <- evalue1()
    id_arg <- input$identity
    cov_arg <- input$coverage
    hit_arg <- input$hits
    out_arg <- input$outype1
    cazypredict::blastcazy(filename = filename_arg, evalue = evalue1_arg, identity =
id_arg, coverage = cov_arg, hits = hit_arg, output = out_arg)
```

```
  })

  output$blast <- renderTable({
    getblast()
  },include.rownames=FALSE, spacing = "xs")

  evalue2 <- reactive({as.numeric(input$evalue2)})
  gethmmer <- eventReactive(input$button3, {
    align_arg <- input$align
    evalue2_arg <- evalue2()
    out2_arg <- input$outype2
    cazypredict::hmmercazy(align = align_arg, evalue = evalue2_arg, output = out2_arg)
  })
  output$hmmer <- renderTable({
    gethmmer()
  },include.rownames=FALSE, spacing = "xs")

  getprosite <- eventReactive(input$button4, {
    results_arg <- input$results
    cazypredict::patterncazy(type = results_arg)
  })

  output$prosite <- renderTable({
    getprosite()
  },include.rownames=FALSE, spacing = "xs")
})
```