UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

21 | 22

Trabajo fin de grado

Integración de programas de análisis de estructuras macromoleculares para biología estructural, modelos atómicos, y cribado virtual de fármacos

Marcos de las Heras Roncero

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Integración de programas de análisis de estructuras macromoleculares para biología estructural, modelos atómicos, y cribado virtual de fármacos

Autor: Marcos de las Heras Roncero
Tutor: Carlos Óscar Sorzano Sánchez
Ponente: Roberto Marabini Ruiz

junio 2022

Marcos de las Heras Roncero
Integración de programas de análisis de estructuras macromoleculares para biología estructural, modelos atómicos, y cribado virtual de fármacos

Marcos de las Heras Roncero
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres

Intelligence is the ability to avoid doing work, yet getting the work done.

Linus Torvalds

Agradecimientos

En primer lugar, querría agradecer a Carlos Óscar Sorzano por acompañarme en todo este proceso, permitiendome colaborar en un proyecto tan importante como este. También querría agradecer a Daniel del Hoyo por ayudarme en el desarrollo del trabajo.

En segundo lugar, querría agradecer a mis amigos de la universidad, por estar siempre ahí haciendo mi estancia por la EPS lo más amena posible.

Finalmente, pero no menos importante, querría agradecer a mis padres por cuidarme, apoyarme y quererme durante toda mi vida, sin darse por vencidos.

Resumen

Este Trabajo de Fin de Grado (TFG) se enmarca en el ámbito de la Biología Estructural, abordada desde la técnica del análisis de partículas individuales (SPA) mediante criomicroscopía electrónica. Dentro de esta disciplina, el Centro Nacional de Biotecnología del CSIC (CNB) está liderando un proyecto pionero de investigación para el que ha desarrollado la *workflow-engine*, Scipion: un integrador de diferentes softwares de cálculo científico que va concatenando pasos para que, partiendo de los datos proporcionados por el microscopio electrónico (ME), se pueda reconstruir la estructura tridimensional de una proteína y, a partir de ésta, obtener su modelo atómico correspondiente. Esta información es de vital importancia para conocer cómo realizan las proteínas su función dentro de la célula, para desarrollar nuevos fármacos, para la detección precoz de enfermedades y para proyectos de biotecnología como fabricación de nanosensores, etc.

El objeto de este TFG es integrar TEMPy en Scipion. TEMPy es un paquete de análisis de la estructura tridimensional de macromoléculas biológicas desarrollado por las siguientes universidades: University of London, University of Oxford, University of Southern California y University of Cambridge. Esta aportación contribuirá a mejorar la veracidad de la información que proporciona Scipion sobre el modelo atómico de una proteína.

Para ello se construye un *plugin* en lenguaje Phyton el cual implementa dos protocolos de TEMPY que, a partir de unos formularios de entrada como: el mapa de densidad obtenido del ME, el modelo atómico de Scipion y algunos parámetros adicionales, devuelve información sobre la compatibilidad de ambos. Esta compatibilidad se mostrará, tanto por *scores*, como por histogramas, gráficos e imágenes en 3D. Adicionalmente, aplicando un algoritmo genético a los datos de partida, crea un nuevo modelo atómico alternativo, que se compara también con el mapa de densidad, informando sobre la compatibilidad de estos últimos, mediante *scores*, histogramas, gráficos e imágenes en 3D.

Palabras clave

Microscopia electrónica, mapa de densidad, modelo atómico, Scipion, TEMPy

Abstract

This Bachelor Thesis has been developed within the field of Structural Biology, approached from the technique of Single-particle analysis in cryo-electron microscopy (cryoEM). Within this discipline, the CSIC's National Center for Biotechnology (CNB) is leading a pioneering research project developing a workflow-engine, Scipion, wich allows creating the whole workflow combining the most relevant image processing packages in an integrative way that concatenates steps so that, from the data provided by the electron microscope (EM), the 3D reconstruction of a protein can be obtained and its corresponding atomic model can be built. This information is of vital importance to know how proteins carry out their function within the cell, in developing new drugs, for the early detection of diseases and for biotechnology projects such as the manufacture of nanosensores, etc.

The goal of this Bachelor Thesis is to integrate TEMPy in Scipion. TEMPy is a 3D structure analysis package for biological macromolecules developed by the following universities: University of London, University of Oxford, University of Southern California and University of Cambridge. This contribution will improve the veracity of the information that Scipion provides about the atomic model of a protein.

In order to achieve this goal, a plugin is built in Phyton language to implement two TEMPy protocols that, based on input data such as: the density map obtained from the EM, the Scipion atomic model and some additional parameters, returns information on the compatibility of both. This compatibility will be shown, by scores, as well as by histograms, graphs and 3D images. Additionally, applying a genetic algorithm to the original atomic model, gamma-Tempy creates a new alternative atomic model, which is also compared with the density map, reporting on the compatibility both, through scores, histograms, graphs and 3D images.

Keywords

Electron Microscopy, density map, atomic model, Scipion, TEMPy

Índice

Listas

Lista de códigos

Lista de figuras

1

INTRODUCCION

1.1. Introducción del Tema

La Biología Estructural estudia los principios que determinan la estructura tridimensional (3D) de las macromoléculas biológicas (en particular proteínas y ácidos nucleicos) y los complejos biomoleculares, y explica las relaciones entre su estructura y su función biológica en el organismo. Véase [1].

Aunque existen muchos tipos de moléculas en cada ser vivo, los estudios de la biología estructural generalmente se centran en los genes y las proteínas. La razón es que estas últimas desempeñan una enorme diversidad de funciones en las células vivas y los genes contienen la información necesaria para fabricar las proteínas, tal y como se indica en [2].

Es necesario entender cómo las proteínas adquieren su estructura 3D y de qué manera las alteraciones en dicha estructura afectan a su funcionamiento, de esta forma, podemos aplicar este conocimiento a la obtención de nuevos fármacos, al diagnóstico precoz y la cura de enfermedades, al diseño de estructuras biológicas complejas con nuevas funcionalidades, a la ingeniería genética y a la biotecnología: elaboración de biosensores, etc. Véase [3].
Para determinar la estructura tridimensional de una macromolécula existen varios métodos, entre los que destaca: el **análisis de partículas individuales (SPA) mediante criomicroscopía (Crio-ME) electrónica** y posterior procesado de las imágenes obtenidas, por métodos computacionales, para obtener una reconstrucción en 3D de la macromolécula en estudio y, a partir ésta, su modelo atómico.

En la criomicroscopía electrónica, la muestra de una macromolécula, embebida en una matriz sólida de hielo amorfo (congelada a -180C con etano líquido a gran velocidad, varios cientos de miles de grados Celsius/segundo), es iluminada por un haz de electrones y los detectores al final de la columna del microscopio producen películas de esta muestra. Cada película está compuesta por miles de imágenes de macromoléculas idénticas en orientaciones aleatorias. Para procesar las imágenes, en el caso objeto del presente trabajo, se usará la *workflow-engine* **Scipion**, un software desarrollado por

el Centro Nacional de Biotecnología (CNB) del CSIC que actualmente se utiliza en todo el mundo, en miles de proyectos de biología estructural.

En [4] se enfatiza en que, la singularidad de Scipion, respecto a los otros enfoques para la recopilación automatizada de las imágenes obtenidas por ME, es que permite combinar los paquetes de procesamiento de imágenes más relevantes de forma integradora: Xmipp9, Relion10, CryoSPARC11, Eman12, Spider13, Cryolo14, Ctffind15, CCP416, Phenix17.... (actualmente utiliza más de 500). Al comparar los resultados obtenidos con varios métodos de procesamiento de imágenes, se consigue generar una reconstrucción 3D de macromoléculas más precisa y veraz. Además, incorpora todas las herramientas necesarias para optimizar la integración, interoperabilidad, trazabilidad y reproducitibilidad para realizar un seguimiento completo de todo el flujo de trabajo de procesamiento de imágenes.

Para incluir un nuevo paquete de procesamiento de imágenes, Scipion dispone de *plugins*, que permiten la incorporación de protocolos que llevan a cabo los cálculos científicos deseados. Un protocolo se compone de una entrada, que se especifica mediante un formulario, y una salida, que se puede ver por pantalla una vez ha terminado el algoritmo de forma satisfactoria. La salida de los protocolos incluye un resumen del proceso, una lista de los métodos empleados, y también puede incluir gráficos representando la fiabilidad del algoritmo en función de las distintas partes de la macromolécula en estudio.

1.2. Objetivos

El presente TFG tiene el objetivo de crear un *plugin* de Scipion, que implemente los protocolos de TEMPy (TEMPy y gamma-TEMPy), un paquete de análisis de la estructura tridimensional de macromoléculas biológicas, desarrollado por cienfíficos de las siguientes universidades: University of London, University of Oxford, University of Southern California y University of Cambridge. Véase [5] y [6].

Este proceso permitirá que una persona que no sepa programar pueda:

- Cuantificar la compatibilidad entre un mapa de densidad obtenido por microscopía electrónica y su correspondiente modelo atómico. Esta compatibilidad se mostrará, tanto por *scores*, como por histogramas, gráficos e imágenes en 3D.
- A partir de Gamma TEMPy, crear, aplicando un algoritmo genético al modelo atómico de partida, un nuevo modelo atómico, que incluye ligeras variaciones respecto al original y que se comparará con el mapa de densidad de ME, para mostrar, de nuevo, la compatibilidad entre ambos, mediante *scores*, histogramas, gráficos e imágenes en 3D. Desarrollado en mayor profundidad en [5] y [6].

1.3. Motivaciones

La informática ha recorrido un largo camino desde sus orígenes y actualmente se emplea en gran cantidad de campos científicos por su capacidad de realizar cálculos matemáticos y computacionales con una precisión y rapidez inalcanzables por los seres humanos, permitiendo así descubrimientos que no serían posibles si esta tecnología no existiera y el caso de la Biología Estructural no es distinto al resto.

En este proyecto se abordará uno de los objetivos principales de la introducción de la informática al mundo de la Biología Estructural: encontrar un método universal para obtener el modelo atómico de una estructura proteica tridimensional, de la forma más precisa posible.

Este proceso supone encontrar y minimizar errores en los distintos métodos empleados actualmente, además de la comparación entre varios métodos para cada una de las partes del compuesto, combinando los resultados más favorables para la obtención final del modelo. Mediante el refinamiento a la hora de calcular un modelo atómico se puede entender mejor el comportamiento y las funciones biológicas del componente, identificar problemas que en él se produzcan y que afectan a nuestro organismo y permitir también la creación de fármacos más precisos para el tratamiento de enfermedades así como otras aplicaciones sanitarias e industriales (biotecnología).

La colaboración en el proyecto que se está desarrollando en el CNB en relación con esta disciplina me pareció especialmente ilusionante, no sólo por su aplicación práctica en el campo de la medicina, sino por la originalidad del enfoque integrador de la *workflow-engine* que ha desarrollado el equipo de Carlos Óscar Sorzano, Scipion, que permite aunar los esfuerzos de múltiples proyectos de investigación desarrollados en todo el mundo, para la obtención de un bien que beneficiará a la humanidad. Como un valor añadido, me gustaría recalcar lo orgulloso que estoy de poder colaborar en un proyecto científico tan interesante, que ha sido desarrollado enteramente por científicos españoles y en nuestro país.

2

Estado del Arte

En este apartado se explicará el contexto en el que nos encontramos actualmente dentro del campo de la Biología Estructural y analizaremos con mayor detalle, el proyecto desarrollado por el CNB: Métodos Computacionales aplicados a la Biología Estructural, en el cual se enmarca el desarrollo del presente trabajo de fin de grado.

2.1. Biología Estructural

Como ya hemos mencionado anteriormente, **la Biología Estructural** estudia los principios que determinan la estructura tridimensional (3D) de las macromoléculas biológicas (en particular proteínas y ácidos nucleicos) y los complejos biomoleculares, y explica las relaciones entre su estructura y su función biológica en el organismo, tal y cómo se indica en [1]

El proyecto del CNB en el que participa este trabajo se centra en el estudio de las proteínas ¿Por qué es tan importante conocer su estructura tridimensional? Porque en todas las reacciones enzimáticas y del metabolismo celular intervienen proteínas y su mecanismo de acción está ligado a la disposición espacial de los átomos que las componen; conociendo su estructura tridimensional, podremos entender cómo funcionan, con qué otros componentes celulares interaccionan, etc. En definitiva, podremos conocer cómo se desarrollan las funciones celulares de los seres vivos.

Cada proteína, es como una pequeña máquina y su estructura es dinámica: mueven componentes celulares y se mueven ellas mismas para poder realizar su función dentro de las células.

La aplicación práctica de la Biología Estructural es muy importante en el campo de las ciencias de la salud y de la biotecnología. Al permitir el avance en el conocimiento de los procesos biológicos normales y patológicos, facilita el desarrollo de terapias dirigidas a vías moleculares celulares específicas, así como el diagnóstico más preciso y cada vez menos invasivo de algunas enfermedades, el diseño de medicamentos, la ingeniería de proteínas y la optimización de procesos biotecnológicos, entre otras

aplicaciones.

2.2. Métodos de Visualización de Proteínas

Actualmente, los métodos que nos ofrecen mayor fiabilidad y precisión para visualizar la disposición tridimensional de las proteínas son los siguientes:

- **Difracción de Rayos X**: Se basa en la difracción que experimenta un haz de Rayos X al atravesar un sólido en estado cristalino. Esta técnica precisa de una gran cantidad de muestra y la cristalización de las proteínas, algo que, no sólo no es sencillo, sino que puede alterar la estructura tridimensional de la molécula en estudio.
- **Resonancia Magnética Nuclear**: Se basa en la aplicación de un campo magnético a macromoléculas biológicas en solución. Estudia el comportamiento de ciertos núcleos atómicos (aquellos que poseen *spin* nuclear distinto de cero) y permite estudiar la información estructural o química de la muestra. Presenta varios condicionantes: requiere también una gran cantidad de muestra, enriquecimiento isotópico de la misma y la limitación del tamaño de la molécula en estudio (proteínas y moléculas de ARN pequeñas o dominios aislados de una macroproteína de tamaño medio).
- **Microscopía electrónica**: En el microscopio electrónico (ME), un haz de electrones "ilumina" una fina muestra colocada en una columna de alto vacío. Los electrones impactan contra la muestra y algunos son transmitidos y otros dispersados. Debajo de la muestra se sitúan unos detectores que captan imágenes de la muestra en estudio. Existen diferentes tipos de microscopio electrónico pero el principio de su funcionamiento es el mismo en todos ellos.

Entre 1931 y 1933 el físico alemán Ernest Ruska y el ingeniero en electricidad, también alemán, Max Knoll, desarrollaron el primer microscopio electrónico de transmisión (MET) para la observación de materiales.

Entre los avances obtenidos en microscopía electrónica, desde su invención, hay que destacar la criomicroscopía electrónica, porque ha supuesto una revolución en la resolución del ME, tal y como se indica en [7]. Esta técnica permite observar muestras a 1,15 A e incluso, a nivel atómico, dada la mejora del software y de los métodos matemáticos de procesamiento de imágenes. Por ello, se ha convertido en una técnica clave para la investigación en Biología Estructural, en el desarrollo de nuevos fármacos, nanosensores en sanidad y biotecnología, etc. Desarrollado en [8]

La Criomicroscopía electrónica, considerada el método del año 2015 por la revista *Nature Methods*, ha sido considerada en las dos últimas décadas como una técnica de Biología Estructural aplicable al estudio de los sistemas biológicos complejos porque solo precisa de pequeñas cantidades de muestra, en estado "nativo".

Los últimos avances en instrumentación y programación han mejorado enormemente la capacidad de la criomicroscopía electrónica para obtener las estructuras de partículas aisladas, con resolucio-

nes que permiten la modelización atómica directa en los mapas de densidad, y también para estudiar mezclas de composición y conformación diversa. Además, el tiempo para reconstruir la estructura ha disminuido notablemente con la automatización de la adquisición y del procesamiento de los datos, así como gracias al menor número de imágenes de partículas requerido para obtener una señal de alta resolución (gracias a nuevas técnicas de detección). Véase [8]

Tal y como se desarrolla en [8], las Bases de Datos de estructura tridimensionales de Proteínas y de Ácidos Nucleicos, entre otras, se están alimentando a un ritmo inédito de nuevas estructuras aportadas por la criomicroscopía electrónica, entre ellas, las de complejos macromoleculares críticos que hasta hace muy poco se consideraban metas fantasiosas. Nos hallamos ante un enorme avance en términos de aplicabilidad, rendimiento y resolución de dicha técnica, lo que la sitúa en el centro de atención de la comunidad científica internacional.

Este método fue desarrollado gracias a la colaboración de Jacques Dubochet, Joachim Frank y Richard Henderson. En 2017 se les otorgó el premio Nobel de Química en 2017 por "desarrollar la criomicroscopía electrónica para la determinación estructural en alta resolución de biomoléculas en soluciones".

Richard Henderson trabajó en los detectores directos de electrones para la obtención de imágenes en 2D y en cómo construir el ME para aumentar su resolución; Jacques Dubochet desarrolló el método de obtención de muestras embebidas en una matriz sólida de hielo amorfo, Joachim Frank mejoró los métodos de procesado de las micrografías electrónicas que se obtienen del ME para la reconstrucción tridimensional de la muestra en estudio.

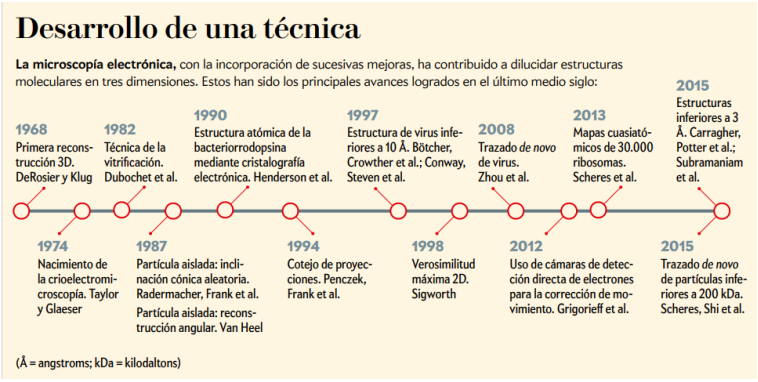


Figura 2.1: Principales avances de la ME en el último siglo. Extraido de [8]

Este TFG se desarrolla dentro de un proyecto de **análisis de partículas individuales (SPA) a través de la criomicroscopía electrónica** (Crio-ME), que se está desarrollando en el CNB para el

cual, disponen del criomicroscopio más potente de España, el JEOL CryoARM 300.

En [8] se describen brevemente las fases de esta técnica:

Preparación de la muestra: se toma una muestra de proteína purificada y se deposita en una rejilla especial de 3 mm que consiste en una lámina microperforada (normalmente de carbono amorfo) encajada en un soporte metálico.
En el caso ideal, las partículas de proteína se reparten uniformemente por los orificios de la rejilla siguiendo distintas orientaciones.
A continuación, la rejilla se sumerge en un criógeno, como etano líquido, que la ultracongela instantáneamente (a -180C) y deja así atrapadas las partículas en una fina película de hielo amorfo. Además de capturar la estructura de la proteína en el momento de la congelación, el proceso protege hasta cierto punto la muestra del daño por radiación y evita la evaporación de la solución donde se encontraba suspendida la proteína en las condiciones de ultravacío con las que opera el microscopio electrónico de transmisión. La muestra se introduce en el ME.

De imágenes 2D al modelo 3D: los detectores posicionados bajo la muestra obtienen muchas fotos en milisegundos, que se agrupan en películas (actualmente se pueden obtener aproximadamente un millón de electromicrografías bidimensionales) que retratan partículas individuales de proteína depositadas en la rejilla. Dada la necesidad de aplicar bajas dosis de electrones para no dañar las muestras radiosensibles, las proyecciones bidimensionales contienen demasiado ruido para poder resolver las estructuras con detalle atómico. Pese a ello, las señales pueden mejorarse obteniendo el promedio de un elevado número partículas.

A menudo, éstas quedan congeladas en la rejilla en orientaciones al azar, por lo que el promediado no es un proceso sencillo. Pero ello supone una ventaja, porque se necesitan muchas vistas bidimensionales distintas de la proteína para reconstruir su estructura tridimensional. Con sofisticados métodos de procesamiento, las imágenes se alinean y los datos se combinan. A continuación, se traza un mapa 3D preliminar que se refina de forma iterativa y se valida con herramientas informáticas especializadas. Por último, la secuencia proteica se ajusta en el mapa 3D para construir un modelo en tres dimensiones de la proteína.

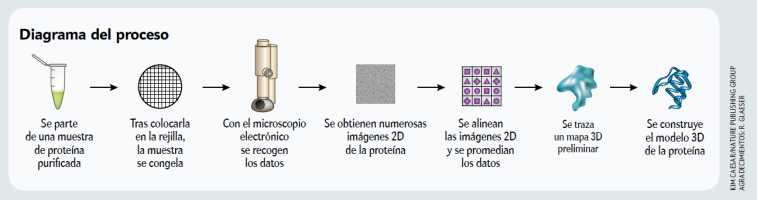
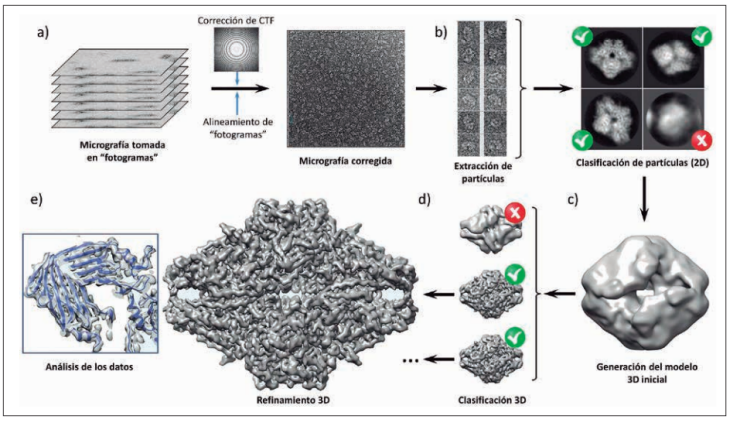


Figura 2.2: Descripción de las fases de la técnica de análisis de partículas individuales (SPA) a través de la Crio-ME. Extraída de [8]



[9]

Figura 2.3: Procesamiento de los datos obtenidos por Crio-ME. Una vez las imágenes han sido adquiridas deben ser procesadas. Consiste básicamente en cinco etapas: 1) corrección de los "defectos" de las imágenes generadas por el microscopio, 2) extracción, clasificación y selección de las partículas de interés, 3) generación de un modelo inicial, 4) determinación de la heterogeneidad presente en la muestra y 5) refinamiento y obtención de la(s) estructura(s) final(es)

2.2.1. *Workflow-engine* Scipion

En el CNB se ha desarrollado el software Scipion. Tal y como se describe en [4], Scipion es un integrador de diferentes softwares de cálculo científico que va concatenando pasos para que, partiendo de los datos del ME, se pueda obtener la estructura tridimensional de una proteína y, a partir de ésta, su modelo atómico. Actualmente integra más de 500 métodos de cálculo encaminados a:

- Detectar problemas en las condiciones de la obtención de imágenes del ME que afectan a la resolución de las mismas, tales como: alineamiento, desenfoque, ruido, oscilación de imágenes, la existencia de rejillas vacías etc.
- Procesar las imágenes obtenidas por el ME para obtener la estructura tridimensional de la partícula original.
- Obtener el modelo atómico de la molécula en estudio.
- Creación virtual de compuestos que interaccionen con la proteína concreta: fármacos, inhibidores de movimiento etc. Para promover la colaboración con compañías farmacéuticas y biotecnológicas.

La integración de métodos de cálculo desarrollados por científicos y matemáticos que trabajan en el campo de la Biología Estructural da la posibilidad de comparar los resultados obtenidos por diferentes enfoques, lo que enriquece y ayuda a que los resultados, en cuanto a la estructura tridimensional y el modelo atómico de una proteína, se aproximen lo más posible a la molécula en estado nativo.

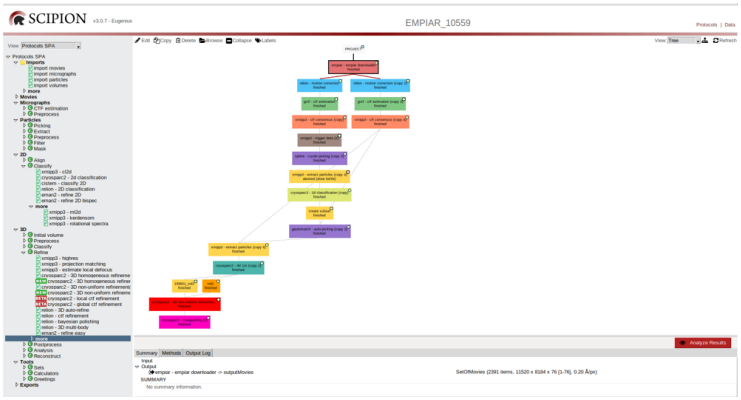


Figura 2.4: Ejemplo de proyecto de workflow-engine Scipion. Extraída de [10]

Hoy en día, Scipion se está utilizando por todo el mundo en miles de proyectos. El usuario únicamente tiene que manejar formularios sencillos, que se generan de forma automática.

El objetivo del proyecto que se está desarrollando en el CNB sería que Scipion procesara de la forma más automática posible los flujos de información que recibiera del ME y, si fuera posible, que lo hiciera en *streaming* de forma que, a la vez que se obtuvieran las imágenes del ME, se analizaran las condiciones del proceso de obtención de fotos y se calcularan la estructura tridimensional y el modelo atómico de la proteína en estudio.

Varios estudios (véase [11], [12], [13] y [14]) muestran como Scipion está en constante crecimiento debido a la continua incorporación de algoritmos y análisis matemáticos, algunos datos procedentes de tomografía ME y nuevos algoritmos de cálculo de modelos atómicos.

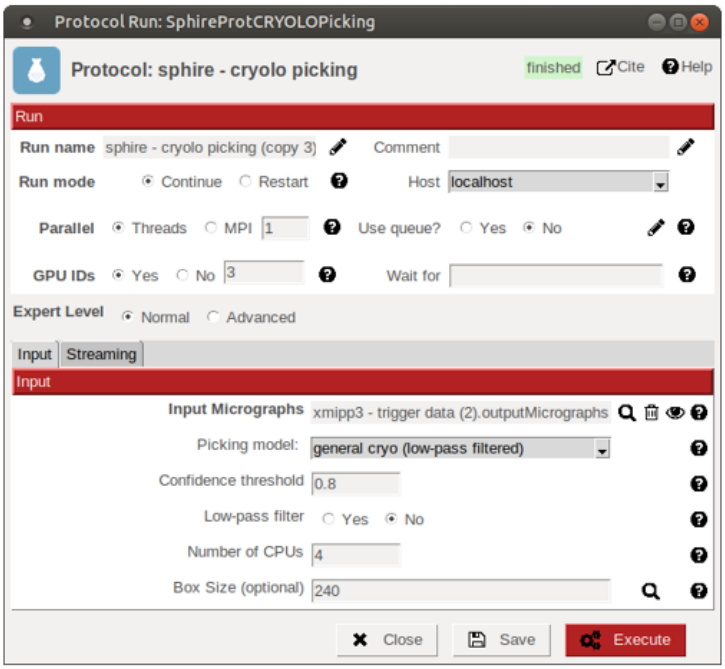


Figura 2.5: Formulario Scipion. Extraída de [10]

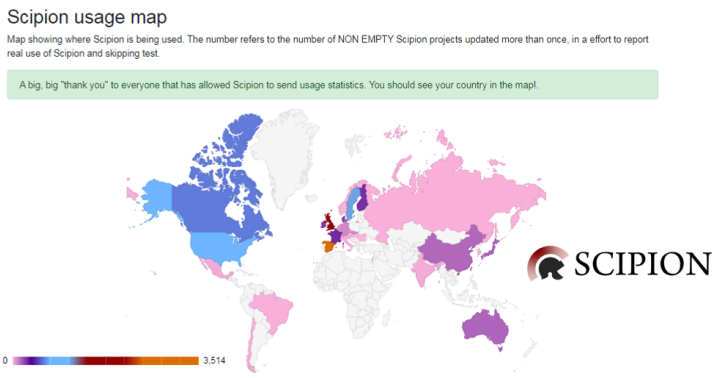


Figura 2.6: Uso de Scipion. Proyectos en los que actualmente de usa Scipion a lo largo del mundo. Extraída de [10]

El objeto del presente TFG está enfocado en esta línea de trabajo, siendo su ámbito de aplicación el ajuste o compatibilidad del modelo atómico de la muestra en estudio.

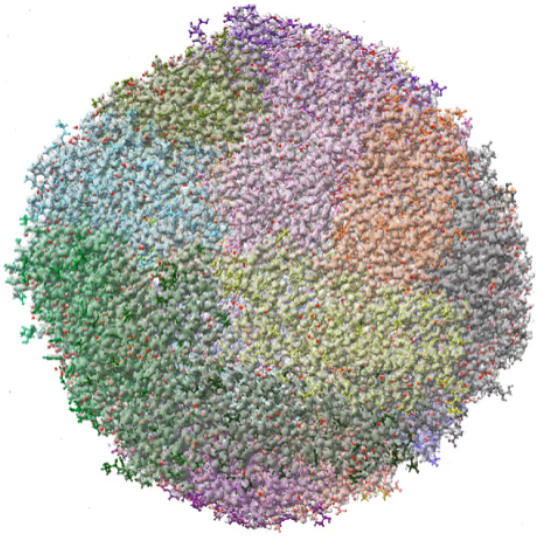


Figura 2.7: Estructura 3D de la Apoferritina. [11]

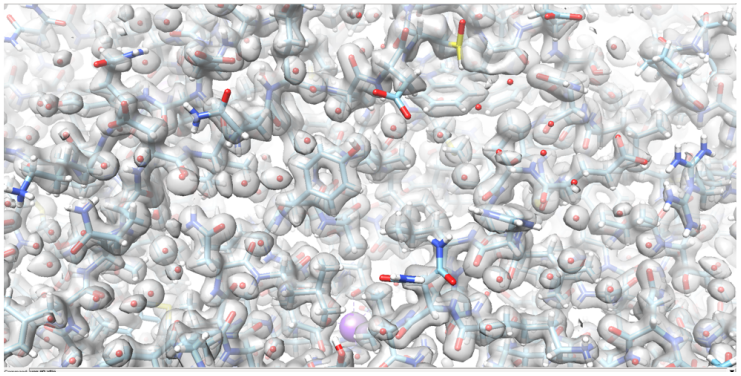


Figura 2.8: Modelo atómico de Apoferritina. [11]

3

Diseño

En este capítulo se describirán las principales funcionalidades de Scipion y TEMPy; Se detallará la interacción entre ambos y se desarrollarán el entorno y el proceso de instalación de los dos. Así mismo, se detallarán los requisitos funcionales y no funcionales imprescindibles para el desarrollo del *plugin*.

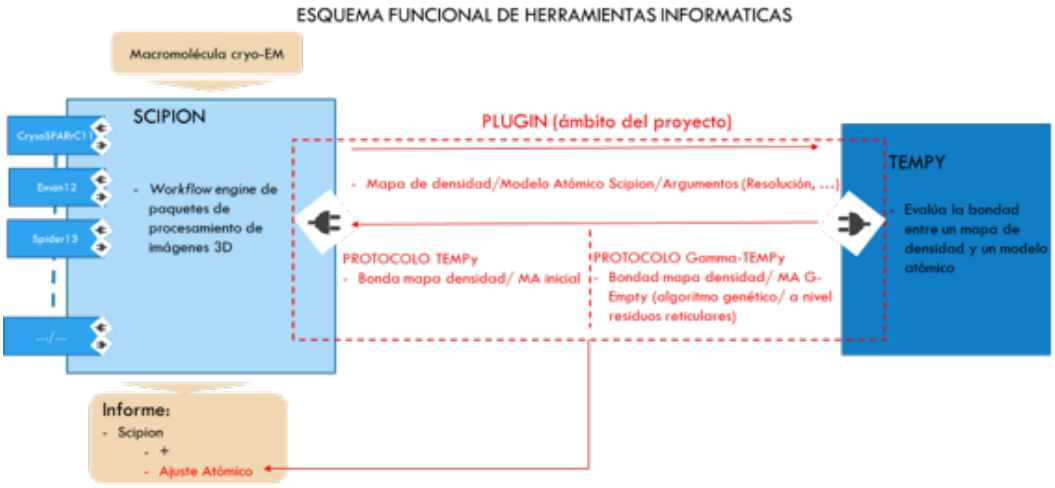


Figura 3.1: Esquema del *plugin* TEMPy.

En la figura 3.1 podemos ver una representación esquemática con el diseño funcional a alto nivel del *plugin* TEMPy, donde podemos diferenciar los dos protocolos: TEMPY y gamma-TEMPy.

3.1. Introducción a Scipion

La herramienta Scipion es un *workflow-engine* de procesamiento de imagenes diseñado para obtener modelos en 3D de componentes macromoleculares, partiendo de las imagenes proporcionadas por el microscopio electrónico (3DEM).

Está compuesto por múltiples paquetes de sofware que se organizan en *plugins*, en función de sus similitudes u origen de código. Cada *plugin* tiene a su vez uno o más protocolos donde se ejecutan

los pasos a seguir por el algoritmo que se quiera ejecutar. Scipion hace un seguimiento de todos los protocolos ejecutados, recopilando tiempos de ejecución y ficheros de salida de cada protocolo, permitiendo también replicar el proceso siempre que se quiera.

3.1.1. Entorno e Instalación

Es una herramienta de código abierto enfocada a entornos Linux que emplea Python. El contenido de este apartado se puede encontrar en la página web de Scipion bajo el apartado de instalación [15].

Para instalar Scipion necesitamos cumplir los siguientes requisitos de software:

- GCC: GCC 8 recomendado
- OpenMPI
- CUDA: recomendada la versión 10.1

En el caso de Ubuntu habría que ejecutar el siguiente comando:

```
$ sudo apt-get install gcc-8 g++-8 libopenmpi-dev make
```

Para el proceso de instalación se requiere de conda, en caso de no disponer de este software, se recomienda instalar miniconda.

Una vez instalado conda procedemos a instalar Scipion:

1.– Crear un entorno virtual con conda e instalar el instalador de Scipion con pip3.

```
$ conda activate
$ pip3 install --user scipion-installer
```

2.– Instalar el núcleo de Scipion y generar los ficheros de configuración por defecto.

```
$ python3 -m scipioninstaller -conda -noXmipp -noAsk
/path/for/scipion /path/for/scipion/scipion3 config --overwrite
```

3.– Abrir el fichero /path/for/scipion/config/scipion.conf y apendear las variables mostradas a continuación al final del fichero. Hay que asegurarse de que las ubicaciones del software empleado es el correcto para el uso de Xmipp:

```
CUDA = True
CUDA_BIN = /usr/local/cuda-10.1/bin
CUDA_LIB = /usr/local/cuda-10.1/lib64
MPI_BINDIR = /usr/lib64/mpi/gcc/openmpi/bin
MPI_LIBDIR = /usr/lib64/mpi/gcc/openmpi/lib
MPI_INCLUDE = /usr/lib64/mpi/gcc/openmpi/include
OPENCV = False
```

4.– Instalamos el *plugin* Xmipp

```
$ /path/for/scipion/scipion3 installp -p scipion-em-xmipp
 -j 12 | tee -a install.log
```

5.– (Opcional) Creamos un alias para Scipion en el fichero .bashrc:

```
alias scipion3="/path/for/scipion/scipion3"
```

3.2. Estructura y Uso de Scipion

En esta sección nos centraremos en mostrar cómo esta diseñado Scipion y de qué partes principales se compone la aplicación, centrandonos en los protocolos.

La aplicación funciona por proyectos de forma que, al ejecutarla, lo primero que nos mostrará por pantalla será un administrador de proyectos. En esta primera ventana podremos elegir entre crear un nuevo proyecto, importar un proyecto ya existente o seleccionar uno que se hubiera creado con antelación. Podemos ver un ejemplo de la ventana en la figura 3.2

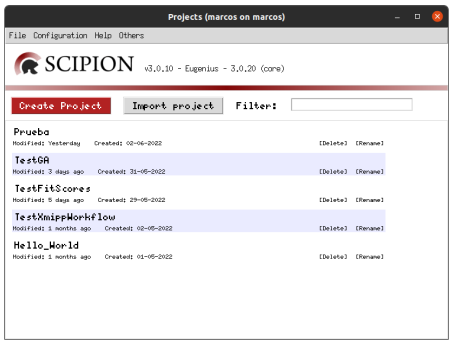


Figura 3.2: Ventana de inicio de Scipion.

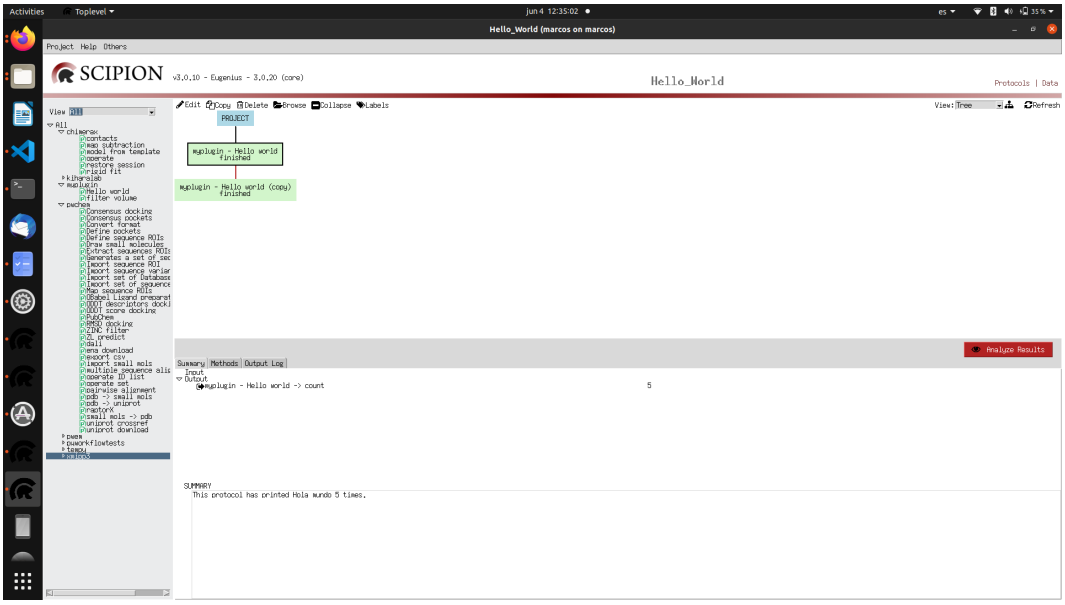


Figura 3.3: Ventana de un proyecto de Scipion

Una vez que seleccionamos un proyecto (figura 3.3) podremos ver un menu a modo de árbol de

plugins y sus correspondientes protocolos en el lado izquierdo la ventana. En el medio se podrá encontrar una distribución del proyecto, mostrarando a modo de árbol qué protocolos se han ido ejecutando, siendo la raiz de este árbol el proyecto en si. Finalmente, en la parte de abajo, muestra la salida del último protocolo ejecutado, de la cual podemos seleccionar tres pestañas:

- *Summary* (figura 3.4) : muestra las entradas del protocolo y un resumen de los resultados obtenidos.
- *Methods* (figura 3.5) : muestra una breve explicación de las acciones que se han realizado y permite acceder a la bibliografia. Esta pestaña es opcional.
- *Output log* (figura 3.6) : muestra detalles del protocolo ejecutado. Está compuesto por una pestaña run.stdoutz otra run.stderr". La primerta pestaña contiene la información de la salida por pantalla del protocolo, así como los pasos que se han ejecutado, cuándo han empezado y terminado y si lo han hecho de forma satisfactoria. La segunda pestaña muestra la información de error que ha devuelto el programa, en caso de devolver alguno.

Figura 3.4: Summary de Scipion

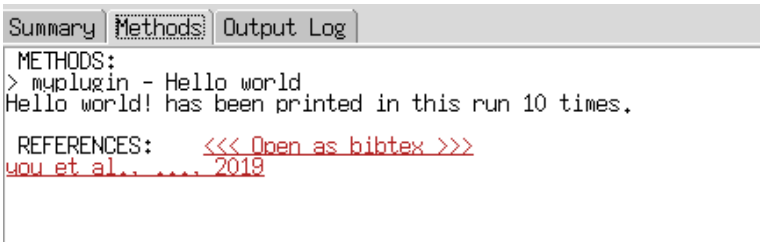


Figura 3.5: Methods de Scipion

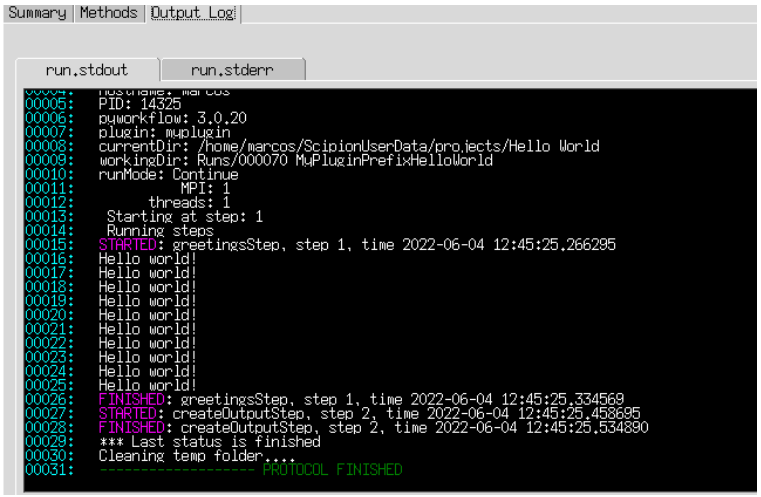


Figura 3.6: Output logs de Scipion

3.3. Introducción a TEMPy

La biblioteca orientada a objetos TEMPy fue diseñada con el objetivo de analizar estructuras ensambladas de macromoleculas, especializada en el contexto de mapas de densidad 3D tomados por un ME.

La funcionalidad de TEMPy nos permite evaluar lo bueno que es el ajuste entre un modelo atómico y un mapa de densidad de ME mediante múltiples funciones de puntuación. También es capaz de generar modelos atómicos alternativos con una buena compatibilidad con el mapa de densidad original.

Para este proyecto separamos las distintas formas de calcular la bondad entre modelo atómico y mapa de densidad a través TEMPy en dos protocolos.
El primer protocolo llevará a cabo las puntuaciones globales: *Local Correlation*(LCCC), *Local Mutual Information*(LMI), *Overlap* (OVR), *Correlation* (CCC), *Normalised Mutual Information* (NMI), *Surface Distante* (SD), y *Normal Vector* (NV). Así como las puntuaciones locales: *Segment Based Cross-Correlation score* (SCCC) y *Segment Based Manders' Overlap Coefficient* (SMOC).
El segundo protocolo se encargará de comprobar la bondad entre un mapa de densidad y un modelo atómico mediante un algoritmo genético (gamma-TEMPy). Este algoritmo genera una permutación del modelo atómico inicial para obtener un mejor encaje con el mapa de densidad.

3.3.1. Entorno e Instalación

TEMPy hace uso de código abierto, y será necesario preparar el entorno, tal y como se especifica en su página web, antes de usarlo [16]:

- Python: Recomendada la versión 3.8
- NumPy: recomendada la última versión, soportado desde la versión v1.6 en delante

- Scipy: recomendada la última versión, soportado desde la versión v.0.10.0 en delante
- Biopython: recomendada la última versión, soportado desde la versión v1.58 en delante
- matplotlib: recomendada la última versión.
- Parallel Python: opcional para el uso de gamma-TEMPy. Soportado desde la versión v1.6.5 en adelante

Para usar TEMPy se necesita descargar un fichero .tar.xz . Una vez descomprimido el archivo podemos destacar la carpeta TEMPy, que contiene el código fuente, el *script* setup.py, los ficheros de pipenv y un fichero README.md que indica cómo se ejecuta el código desde la herramienta de entorno de trabajo de pipenv.

Pipenv es una herramienta que crea y maneja automáticamente entornos virtuales. Su uso no es obligatorio pero es altamente recomendable, ya que contiene la configuración del entorno especificada en la lista mostrada previamente. Antes de poder usarla debemos instalarla:

```
$ pip install pipenv
```

Acto seguido podemos proceder a construir e instalar el código utilizando el entorno de pipenv desde el directorio raiz:

```
$ pipenv run python setup.py build
$ pipenv run python setup.py install —user
```

En nuestro caso hemos empleado el argumento *user* para evitar instalarlo con permisos de superusuario. En el caso de querer ejecutar todo el código debemos instalar también los siguientes paquetes:

- pandas
- sckit-learn
- MDAnalysis
- prettytable: este paquete adicional es usado para mostrar los resultados de una forma más elegante.

Estos paquetes se pueden instalar mediante el comando:

```
$ pipenv install <paquete>
```

3.4. Requisitos Funcionales

A continuación se recopilan una lista de requisitos funcionales necesarios para el satisfactorio desarrollo del *plugin*:

1.– Permitir la selección de los protocolos del *plugin*:
El usuario podrá seleccionar qué protocolo debe ser ejecutado a través de un menú desplegable a la izquierda de la pantalla del proyecto de Scipion.

2.– Indicar los parametros del protocolo mediante un formulario
El usuario podrá introducir todos los parámetros del protocolo mediante un formulario. Ambos protocolos requieren de un mapa de densidad, un modelo atómico y un valor número que indica la resolución del mapa de

densidad como argumentos, pero también precisan de otros parámetros:

- El primer protocolo (TEMPy) usará adicionalmente un parámetro para indicar el intervalo de residuos a analizar que será de tipo json y se formará tal y como se especifica en el requisito funcional 3.
- El segundo protocolo (gamma-TEMPy) usará como parámetros adicionales valores numéricos para la ejecución del algoritmo genético, tales como: el número de soluciones obtenidas, el número de generaciones, el tamaño de la población y el número de CPUs que se utilizarán en la ejecución del algoritmo. Adicionalmente existirán unos parametros para usuarios avanzados que permitirá seleccionar la función de *score* empleada para medir el encaje entre mapa y modelo, las probabilidades de que exista una mutación o recombinación y si se partirá de una población inicial ("*seed*") o no.

En el caso del mapa de densidad y el modelo atómico se deberán seleccionar de la base de datos que contiene todos los ficheros importados al proyecto de Scipion.

3.– Selección de intervalo de residuos de forma dinámica
El usuario podrá seleccionar mediante el uso del ratón un intervalo de residuos de un modelo atómico seleccionando. Para ello deberá especificar primero a qué cadena pertenece el intervalo de residuos que se quiere seleccionar.

4.– Mostrar un *summary* a la finalización del protocolo
El usuario podrá ver un resumen final del protocolo mostrando los datos más relevantes obtenidos a la salida del protocolo. Este resumen también incluirá una lista de los objetos de entrada y salida del protocolo.

5.– Mostrar las acciones realizadas durante el protocolo
El usuario podrá ver de forma descriptiva qué es lo que ha hecho el protocolo, incluyendo citas a documentación relevante del proceso ejecutado.

6.– Mostrar representaciones visuales
El usuario podrá ver los resultados del protocolo mediante gráficos y representaciones 3D de las siguientes maneras:

- Histograma: se muestra la cantidad de residuos que tienen una determinada puntuación.
- Gráfico: se muestra mediante un gráfico la puntuación obtenida por cada residuo.
- Representación 3D: se muestra de forma visual, una representación 3D en la que se muestra la puntuación de cada uno de los componentes del modelo atómico a partir de una gama de colores.

3.5. Requisitos No Funcionales

En este apartado se definirán los requisitos no funcionales del *plugin*:

1.– El *plugin* debe ser accesible mediante GitHub:
Se podrá acceder al *plugin* mediante un repositorio público en la página web de GitHub.

2.– El *plugin* debe poderse ejecutar desde cualquier entorno Linux. :
El *plugin* se podrá ejecutar desde cualquier entorno Linux que tenga instalado Scipion y que cumpla los requisitos de software de éste.

3.– El *plugin* debe hacer uso de la última versión del código TEMPy:
El *plugin* se encargará de descargar la útlima versión de TEMPy y de comprobar que cumple todos los requisitos para su correcto uso.

4

Implementación

En este capítulo se describirá el proceso que se ha seguido para implementar el *plugin* y los dos protocolos: TEMPy y gamma-TEMPy. La implementación descrita en este apartado partirá del proceso de instalación descrito en el apartado 3.1.1.

Empezaremos describiendo el proceso a seguir para instalar y configurar el *plugin* en Scipion y terminaremos describiendo la implementación de los dos protocolos. Para la implementación de estos últimos haremos una división entre la parte común y la parte específica de cada uno de ellos.

Gran parte de la documentación que se utilizará en este cápitulo es accesible desde la página web de Scipion para *developers* [17]. Las funciones más importantes empleadas en la implementación se pueden ver en el anexo A.

4.1. Instalación y Configuración del *Plugin* de Scipion

El primer paso a seguir para la implementación del *plugin* es descargar el template (scipion-em-template [18]) y lo renombramos con el nombre deseado, en este caso se llamaría scipion-em-tempy. Acto seguido procedemos a borrar el directorio .git y a crear una nueva instancia de este. A continuación ya podemos instalar el *plugin* como developer con el siguiente comando:

```
$ Scipion_HOME/scipion3 installp -p
  [PATH_WHERE_CLONED]/scipion-em-tempy --devel
```

Posteriormente hacemos modificaciones en los ficheros de tal forma que sigan el siguiente árbol de ficheros:

```
/
├── CHANGES.txt
├── LICENSE
├── MANIFEST.in
├── README.rst
├── setup.py
└── tempy
    ├── bibtex.py
    ├── constansts.py
    ├── icon.png
    ├── __init__.py
    ├── objects.py
    ├── protocols
    │   └── __init__.py
    ├── protocols.conf
    ├── protocols.py
    ├── tests
    │   └── __init__.py
    ├── viewers
    │   └── __init__.py
    ├── wizards
    │   └── __init__.py
    └── wizards.py
```

Para la instalación del *plugin* ha sido necesario configurar o implementar los siguientes ficheros, que son detallados a continuación:

- **setup.py**
- **protocols.conf**
- **__init__.py**
- **constants.py**

4.1.1. setup.py

El fichero setup.py especifica los parámetros que se usaran para instalar el *plugin*, los cuales son:

- name: atributo obligatorio, donde se especifica el nombre del *plugin*.
- version: atributo obligatorio, donde se especifica la version del *plugin*.
- long_description: atributo opcional, donde se hace una descripción extensa del *plugin*. En este caso se recopila la información del fichero README.rst".
- url: atributo opcional en el que se indica donde esta el código fuente del *plugin*.
- author: atributo opcional en el que se especifica el autor.
- author_email: atributo opcional en el que se indica el correo del autor.
- keywords: atributo opcional, palabras clave por las que buscar el *plugin*.
- packages: atributo opcional, busca los paquetes necesarios para la ejecución del *plugin*.
- instll_requirements: lista de bibliotecas de Python necesarias para la ejecución del *plugin*.

- entry_points:
- package_data: atributo opcional, en nuestro caso lo usamos para especificar la foto de TEMPy y el fichero "protocols.conf"que veremos a continuación.

Código 4.1: Representación de la funcion steup.

```
setup(
    name='scipion-em-tempy', # Required
    version='0.1', # Required
    description='Scipion tempy \textit{plugin}.', # Required
    long_description=long_description, # Optional
    url='https://github.com/HerasRoncero/scipion-em-tempy', # Optional
    author='Marcos de las Heras', # Optional
    author_email='coss@cnb.csic.es', # Optional
    keywords='scipion cryoem imageprocessing scipion-3.0', # Optional
    packages=find_packages(),
    install_requires=[requirements],
    entry_points={'pyworkflow.\textit{plugin}': 'tempy = tempy'},
    package_data={ # Optional
       'tempy': ['icon.png', 'protocols.conf'],
    }
)
```

4.1.2. protocols.conf

El fichero se encarga de especificar donde se mostrará el protocolo dentro del árbol de protocolos en el panel izquierdo del proyecto. En nuestro caso solo describiremos los protocolos que hemos desarrollado, explicando de que se trata de un protocolo (especificado en la variable tag), que clase de Python implementará la función del protocolo (especificado en la variable value) y el nombre con el que se representará en el árbol (especificado en la variable text).

Código 4.2: Representación de la configuración del protocolo.

```
[PROTOCOLS]
Protocols SPA = [
    {"tag": "section", "text": "Tools", "openItem": "False", "children": [
        {"tag": "protocol_group", "text": "Greetings", "openItem": "False", "children": [
            {"tag": "protocol", "value": "ProtFitScores", "text": "TEMPy fit scores"},
            {"tag": "protocol", "value": "ProtGA", "text": "TEMPy GA fit"}
        ]}
    ]}]
```

4.1.3. __init.py__

El fichero __init__.py contiene la definición de la clase *plugin* que se utilizará durante todo el proceso, además del proceso de instalación y preparación del entorno de TEMPy. Este *plugin* realiza las siguientes funciones:

- *_defineVariables(cls)*: Define las variables empleadas por el *plugin*. Se ejecuta siempre que se instale el *plugin* en Scipion
- *defineBinaries(cls, env)*: Esta función se ejecuta siempre. En nuestro caso llama a la función .ªddTempyPackageçuya implementación describiremos más adelante.
- *runTEMPy(cls, protocol, args, outDir, clean)*: Es una función que se llama cuando se ejecuta un protocolo de este *plugin*. En primer lugar accede al entorno virtual creado previamente en la anterior función, después crea una carpeta donde se guardará la salida del programa, prepara los argumentos de llamada y ejecuta el programa en función del protocolo que llamará a esta función. Finalmente cierra el entorno virtual y copia la carpeta donde se ha guardado la salida del programa en una carpeta perteneciente a la ejecución del protocolo.
- *getEnviron(cls)*: Obtiene una serie de variables de entorno virtual.
- *getTEMPYEnvActivation*: Obtiene el comando necesario para activar el entorno virtual empleado en el *plugin*.

Código 4.3: Instalación y preparación del entorno de TEMPy.

```
1        # try to get CONDA activation command
2        installationCmd = cls.getCondaActivationCmd()
3        # Create the enviroment
4        installationCmd += "conda create -y -n %s" \
5                           " python=%s &&& " % (DEFAULT_ENV_NAME, pythonVersion)
6        # Activate the new enviroment
7        installationCmd += " %s &&& " % (DEFAULT_ACTIVATION_CMD)
8        # Solve dependencies
9        installationCmd += "pip install pipenv &&& "
10       # Install download code
11       installationCmd += "pipenv run python setup.py build &&& "
12       installationCmd += "pipenv run python setup.py install --user &&& "
13       installationCmd += "pipenv install pandas &&& "
14       installationCmd += "pipenv install scikit-learn &&& "
15       installationCmd += "pipenv install MDAnalysis &&& "
16       installationCmd += "pipenv install prettytable &&& "
17       # Copy script in destiny directory
18       installationCmd += "cp %s . &&& " % _tempyCalculateScoresPath
19       installationCmd += "cp %s . &&& " % _tempyGAPath
20       installationCmd += "mkdir Predict_Result &&& "
21       # Flag installation finished
22       installationCmd += "touch %s" % TEMPY_INSTALLED
23       tempy_commands = [(installationCmd, TEMPY_INSTALLED)]
24       env.addPackage("tempy",
25                      url="https://tempy.ismb.lon.ac.uk/tempy2.tar.xz",
26                      commands=tempy_commands,
27                      default=default)
```

La función *addTempyPackage* se encargará de descargar e instalar el código de TEMPy además de preparar el entorno para la correcta ejecución de los protocolos. El proceso constará de los siguientes

pasos:

1.– Descarga y descompresión del paquete de TEMPy accesible mediante url.
2.– Creación de entorno virtual y activación de este.
3.– Instalación del paquete pipenv dentro del entorno virtual.
4.– Instalación del código TEMPy usando pipenv y el modo de instalación *setup.py build* y *setup.py install*
5.– Instalación de los paquetes de Python: pandas, scikit-learn, MDAnalysis y prettytable.
6.– Copiar los *scripts* que se usarán en cada protocolo para facilitar la ejecución de estos.
7.– Creación de un directorio donde se guardaran las salidas del protocolo.
8.– Creación de fichero para comprobar que todos los comandos anteriores se han realizado correctamente.

Esta función llamará a la función *addPackage* definida por Scipion la cual crea un comando que sera ejecutado desde el sistema operativo. En caso de que el código TEMPy ya este instalado en nuestro sistema e incluido en el software de Scipion (que se comprobará buscando la existencia del fichero descrito en el punto 8) se ignorará este paso y pasará a proseguir a la instalación del protocolo.

4.1.4. constansts.py

Este fichero se encarga de almacenar constantes. Estas constantes indican:

- Número de versión del código TEMPy
- Nombre del entorno virtual empleado en este *plugin*.
- Comando para activar el entorno virtual.
- Nombre de la carpeta donde se instalará el código de TEMPy
- Url donde se encuentra el código comprimido de TEMPy.

4.2. Implementación de los Protocolos de TEMPy y gamma-TEMPy

Hay elementos comunes a ambos protocolos y elementos específicos de cada uno de ellos. En primer lugar desarrollaremos los elementos comunes y a continuación profundizaremos en los elementos específicos de cada uno de los protocolos.

4.2.1. Elementos Comunes de los Protocolos

Los dos protocolos tienen algunas funcionalidades comunes, como son algunas partes del formulario, pasos de ejecución, recolección de los datos de salida, *views* de salidas de ejecución y bibliografía.

Elementos Comunes a Nivel de Formulario y Pasos de Ejecución

Los protocolos definidos para este proyecto son muy similares, por ello se usa una clase común de las que heredarán los dos protocolos.

El primer paso para desarrollar los protocolos consiste en crear un formulario donde se pueda indicar el mapa de densidad y el modelo atómico que se quiere comparar, además de la resolución con la que se evaluará el mapa de densidad. El resultado final es el que se puede ver en la figura 4.1. Cada protocolo tendrá adicionalmente una serie de parámetros específicos para su correcta ejecución.

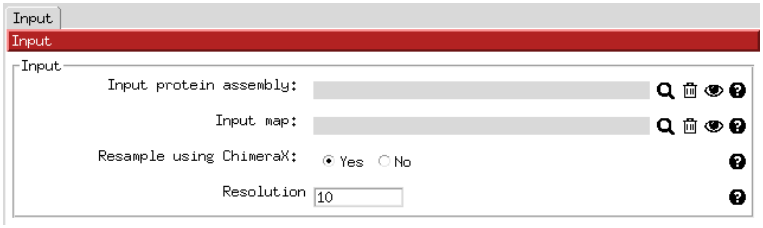


Figura 4.1: Formulario común

La siguiente parte en común es relativa a los pasos de ejecución. Los pasos comunes a seguir consisten en procesar los argumentos (tanto específicos como comunes) y la llamada al *plugin* que ejecutar el *script* correspondiente. Los dos *script*s empleados para cada uno de los protocolos son modificaciones de los ficheros del tutorial proporcionados por TEMPy en su página web.

El paso final consiste en la recopilación de los ficheros y datos de salida del protocolo correspondiente. Ambos protocolos procesarán las salidas y las guardarán en un fichero de tipo cif el cual nos servirá para que Scipion nos permita ver las salidas de forma visual, tal y como explicaremos en el siguiente apartado.

Elementos Comunes a Nivel de View

Tal y como se ha explicado en el apartado anterior las salidas de los protocolos se recopilarán en uno o varios ficheros de tipo cif (dependiendo del protocolo). A partir de este tipo de fichero y partiendo del código implementado en scipion-chem en la parte de vistas de Chimera podremos mostrar la compatibilidad entre el mapa de densidad y el modelo atómico de las siguientes formas:

- Histograma: mostrará en un histograma a nivel cadena del modelo atómico la cantidad de residuos que han obtenido una determinada puntuación.
- Chimera: mostrará, a través del uso de la aplicación, una representación 3D interactiva por colores de cómo bueno es el encaje entre el mapa de densidad y el modelo atómico. Por defecto, los colores para indicar la bondad del encaje irán de rojo a azul, siendo el rojo la mínima puntuación posible y el azul la mejor.
- Gráfico: mostrará un gráfico donde se muestra la puntuación de encaje para cada residuo, dividido en cadenas.

El resultado de este menu se puede ver en la figura 4.2

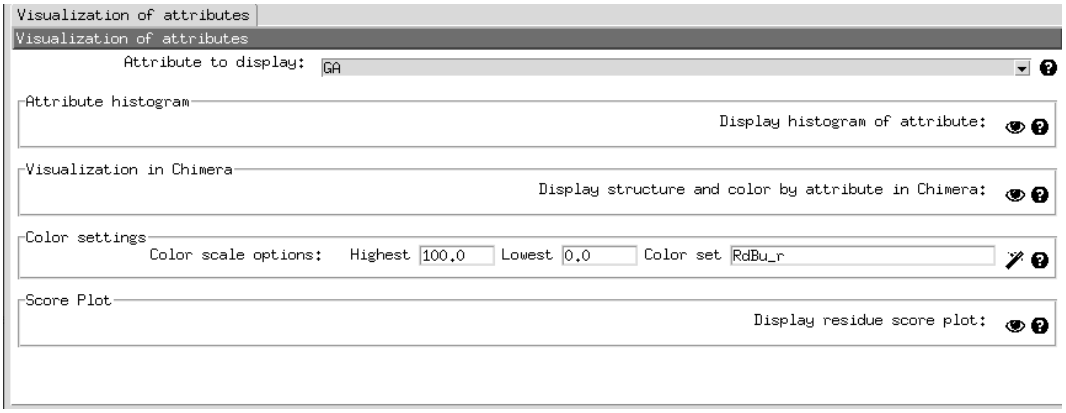


Figura 4.2: Menu de visualización de salidas

Los resultados de estas vistas lo veremos en la sección 5

Elementos Comunes a Nivel de Bibliografía

Ambos protocolos tienen un apartado de nombre methods donde se indica la *bibliografia* empleada [19] en el código de este *plugin*.

4.2.2. Elementos Específicos de TEMPy

El protocolo TEMPy tiene una serie de casos específicos a nivel protocolo y a nivel de wizards.

Elementos Específicos de TEMPy a Nivel de Formulario y *Summary*

La primera diferencia de este protocolo es la introducción de un nuevo argumento para la selección de un intervalo de residuos4.3.

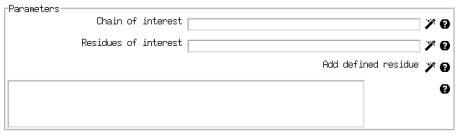


Figura 4.3: Argumento de selección de intervalo de residuos

Para ello se han creado tres wizards que se explicarán en el próximo apartado.

La siguiente diferencia consiste en el contenido de la pestaña *Summary*. Esta pestaña estará compuesta de las salidas que indican la puntuación en modo texto *ASCII* del protocolo TEMPy. Las puntuaciones se mostrarán en formato tabla para los algoritmos globales. Para el algoritmo *SMOC* también

se muestra en formato tabla los valores de *1º y 4º quartil, average y el ratio HIGH/LOW*. .

Elementos Específicos de TEMPy a Nivel de *Wizard*

La siguiente diferenciación está en los *wizards*, que se utiliza para la selección de un intervalo de residuos dentro de un modelo atómico. Este proceso consistirá en tres *wizards*, los cuales son unas modificaciones de los que hay en el protocolo scipion-chem [20].
El resultado se podra apreciar en el formulario de entrada del protocolo tal y como se ve en la figura 4.3.
El primer *wizard* implementado muestra las cadenas presentes en el modelo atómico, y nos permite elegir una de ellas. Al pulsarlo se abrirá un formulario como el de la figura 4.4.

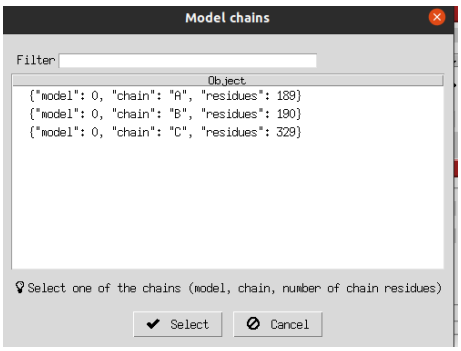


Figura 4.4: Wizard para seleccionar una cadena de residuos.

El segundo *wizard* nos permite la selección de un intervalo de residuos de la cadena seleccionada previamente. Al pulsarlo se abrirá un formulario como el de la figura 4.5.

Figura 4.5: Wizard para seleccionar una un intervalo de registros.

Finalmente el último *wizard* nos sirve para añadir el intervalo de residuos como argumento del *script* que se ejecutará. Esto permitirá seleccionar una o más intervalos de residuos para analizar, repitiendo el proceso descrito. Al pulsarlo se abrirá un formulario como el de la figura 4.6.

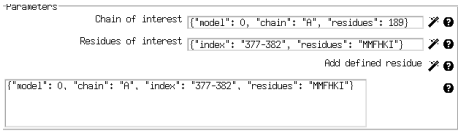


Figura 4.6: Wizard para añadir un intervalo de registros

4.2.3. Elementos Específicos de gamma-TEMPy

Para el caso del protocolo gamma-TEMPy solo existen diferencias a nivel de formulario.

Elementos Específicos de gamma-TEMPy a Nivel de Formulario

La única diferencia que existe en el protocolo gamma-TEMPy, consiste en la introducción de nuevos parámetros para especificar los argumentos con los que se ejecutará el algoritmo genético. Dichos parámetros estarán diferenciados en argumentos básicos y argumentos para usuarios avanzados. Estos serán los siguientes 4.7:

- ***Number of solution:*** Determina la cantidad de salidas que sacará el algoritmo genético. Actualmente el *plugin* solo es capaz de mostrar una única salida.
- ***Number of generations:*** Determina el número de generaciones que se realizarán para obtener el resultado final.
- ***Population Size:*** Indica la población de la que partirá el algoritmo para obtener la solución.
- ***Number of CPUs***
- ***Goodnes of fit scores:*** Determina que función se empleará para determinar la bondad del encaje entre el mapa de densidad y el modelo atómico suministrado. Actualmente se puede elegir entre *Mutual Information Score* (*MI*) y *CCC*.
- ***Mutation rate:*** Probabilidad de que exista una mutación.
- ***Crossover rate:*** Probabilidad de que exista una recombinación.
- ***PDB seed:*** Fichero que servirá como semilla para crear la población inicial.

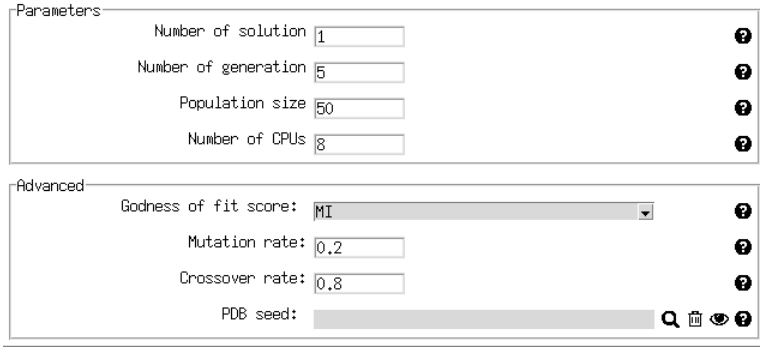


Figura 4.7: Argumentos de entrada para el algoritmo genético.

5

Pruebas

En este apartado se describirán las pruebas realizadas para comprobar el correcto funcionamiento de los protocolos del *plugin* (TEMPy y gamma-TEMPy). Para ello haremos uso de de los tests que proporciona Scipion. Cada prueba consta de los siguientes pasos:

1.– Crear un proyecto específico en Scipion para realizar el test. En el caso de TEMPy se llamará *TestFitSCores* y en el caso de gamma-TEMPy se llamará *TestGA*.

2.– Importar un mapa de densidad y un modelo atómico de la misma macromolécula. En este caso utilizaremos los datos proporcionados por los tutoriales de TEMPy.

3.– Crear el protocolo que se quiere probar para introducir los parámetros especificos de dicho protocolo.

4.– Ejecutar el protocolo creado previamente y verificar que se obtienen las salidas eperadas. En ambos casos comprobaremos los gráficos obtenidos, confirmando que los residuos del modelo atómico no tienen siempre la misma puntuación.

A continuación explicaremos en detalle la creación y los resultados de las pruebas para cada uno de los protocolos.

5.1. Pruebas sobre TEMPy

En este apartado probaremos el primer protocolo. Podemos ver el primer resultado tras ejecutar la pueba directamente en la terminal, como muestra la figura 5.1
Al ejecutar el test se crea un proyecto que importa un mapa de densidad y un modelo atómico de la base de datos de uno de los *dataset* predeterminados de Scipion y con ellos se crea el primer protocolo TEMPy, tal y como se puede ver en la figura 5.2

Al seleccionar el protocolo podemos ver en el *Summary* y comprobar que el protocolo se ejecuta satisfactoriamente. Tal y como se ve en la figura 5.3.

Una vez que hemos comprobado que el *Summary* funciona correctamente comprobamos igualmente que el histograma funciona bien (figuras 5.4 y 5.5).

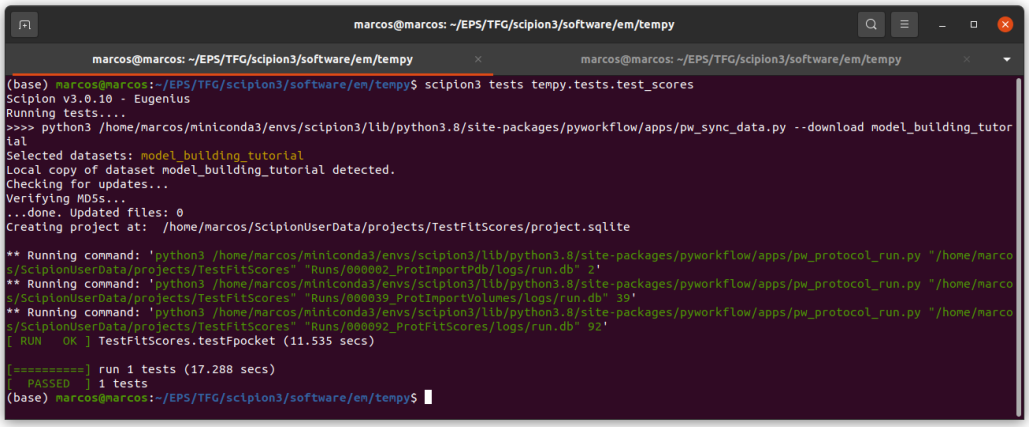


Figura 5.1: Salida de la ejecución del test para el primer protocolo

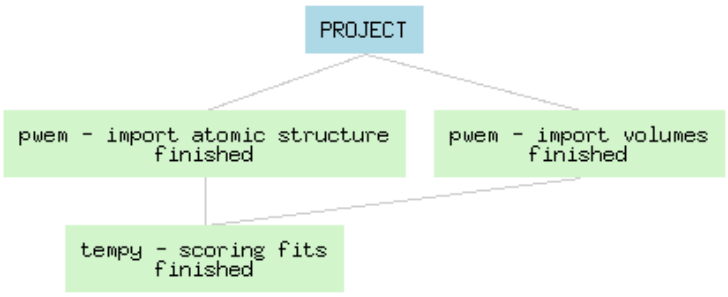


Figura 5.2: Tras ejecutar el test TEMPy desde la terminal se crea el siguiente proyecto

SUMMARY

GLOBAL SCORES

Model	LCCC	LMI	OVR	CCC	NMI	SD	NV
protein	0.9628	0.1836	1.0	0.9234	1.4934	0.7646	0.7597

SCCC LOCAL SCORE for segment 0.985300

SMOC LOCAL SCORE

CHAIN	TOP 25%	LOW 25%	HIGH/LOW	AVG
A	0.9712	0.9636	1.0079	0.9635
B	0.9685	0.9622	1.0065	0.9628
C	0.9648	0.9592	1.0059	0.9584

** Mean SMOC for protein is 1.0

Figura 5.3: *Summary* tras ejecutar el test TEMPy desde la terminal

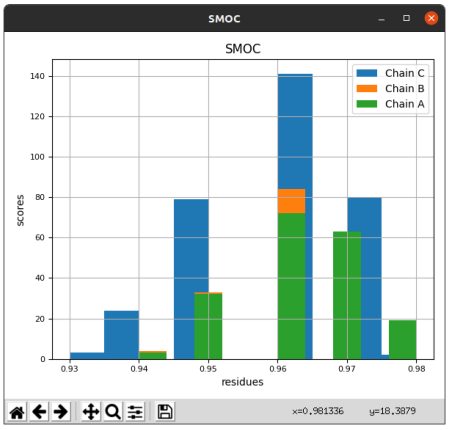


Figura 5.4: Histograma SMOC tras ejecutar el test TEMPy desde la terminal

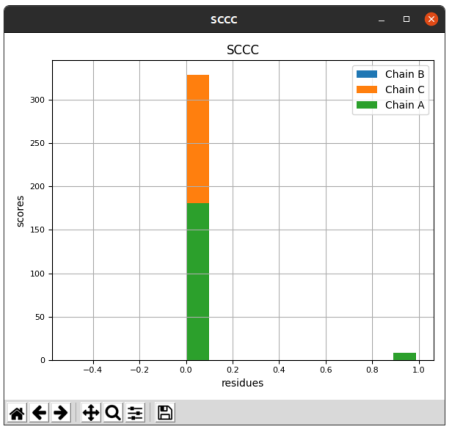


Figura 5.5: Histograma SCCC tras ejecutar el test TEMPy desde la terminal

También podemos comprobar que las gráficas funcionan correctamente (figuras 5.6 y 5.7)

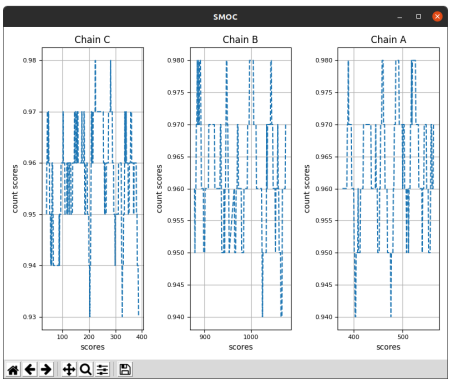


Figura 5.6: Gráfico SMOC tras ejecutar el test TEMPy desde la terminal

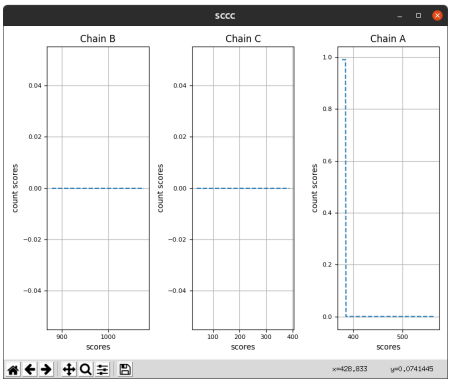


Figura 5.7: Gráfico SCCC tras ejecutar el test TEMPy desde la terminal

Finalmente podemos comprobar visualmente en 3D en encaje gracias a Chimera (figuras 5.8 y 5.9)

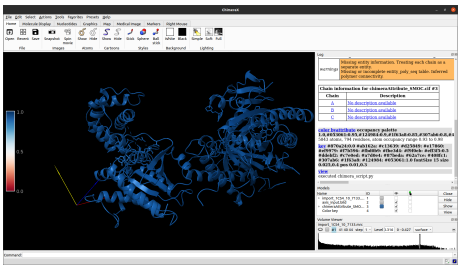


Figura 5.8: Representación 3D SMOC tras ejecutar el test TEMPy desde la terminal

Mediante estos resultados podemos ver que en el caso de SMOC que todos los residuos tienen puntuaciones, mientras que para el caso de SCCC solo los residuos que han sido seleccionados para el análisis tienen una puntuación asociada.

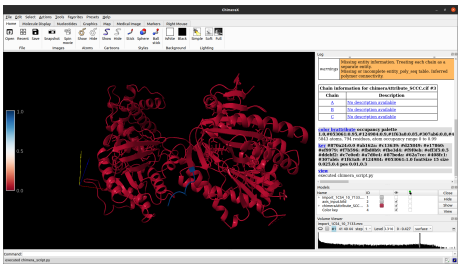


Figura 5.9: Representación 3D SCCC tras ejecutar el test TEMPy desde la terminal

5.2. Pruebas sobre gamma-TEMPy

En este apartado probaremos el segundo protocolo. Podemos ver el primer resultado tras ejecutar la pueba mediante la terminal en la figura 5.10

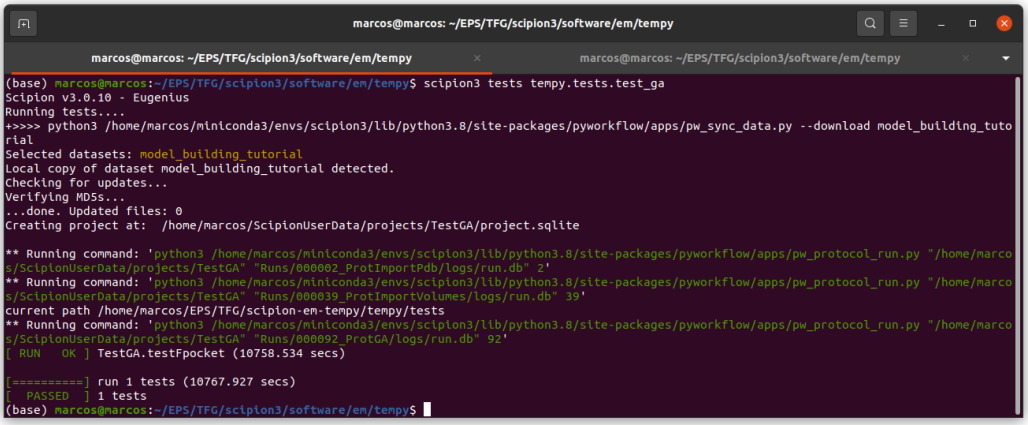


Figura 5.10: Salida de la ejecución del test para el segundo protocolo

Al ejecutar el tests se crea un proyecto que importa un mapa de densidad y un modelo atómico de la base de uno de los *dataset* predeterminados de Scipion y con ellos crea el segundo protocolo gamma-TEMPy, tal y como se puede ver en la figura 5.2

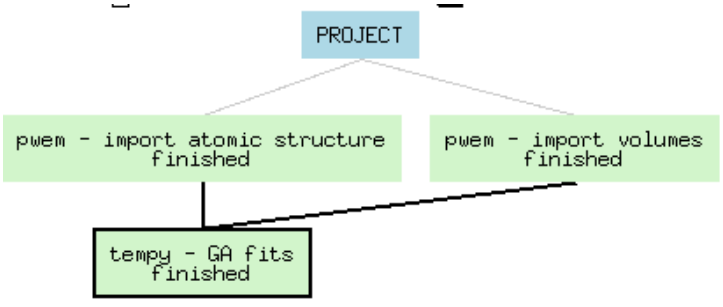


Figura 5.11: Tras ejecutar el test gamma-TEMPy desde la terminal se crea el siguiente proyecto

Dado que este protocolo carece de summary pasaremos directamente a comprobar que el histograma funciona bien

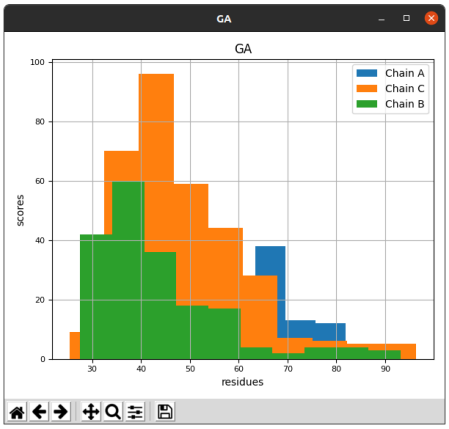


Figura 5.12: Histograma tras ejecutar el test gamma-TEMPy desde la terminal

También podemos comprobar que las gráficas funcionan correctamente (figuras 5.13)

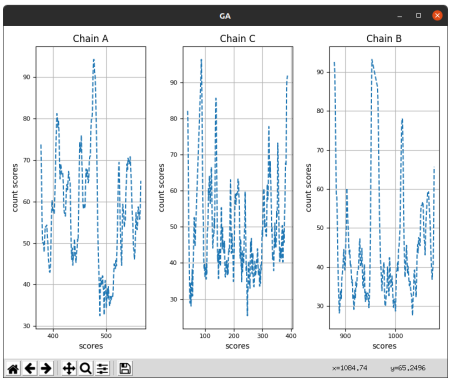


Figura 5.13: Gráfico tras ejecutar el test gamma-TEMPy desde la terminal

Finalmente podemos comprobar visualmente en 3D en encaje gracias a Chimera (figuras 5.14)

Mediante estos resultados podemos ver que los residuos del modelo atómico no toman siempre el mismo valor en ninguno de los casos. De esta forma confirmamos que se obtienen los resultados esperados.

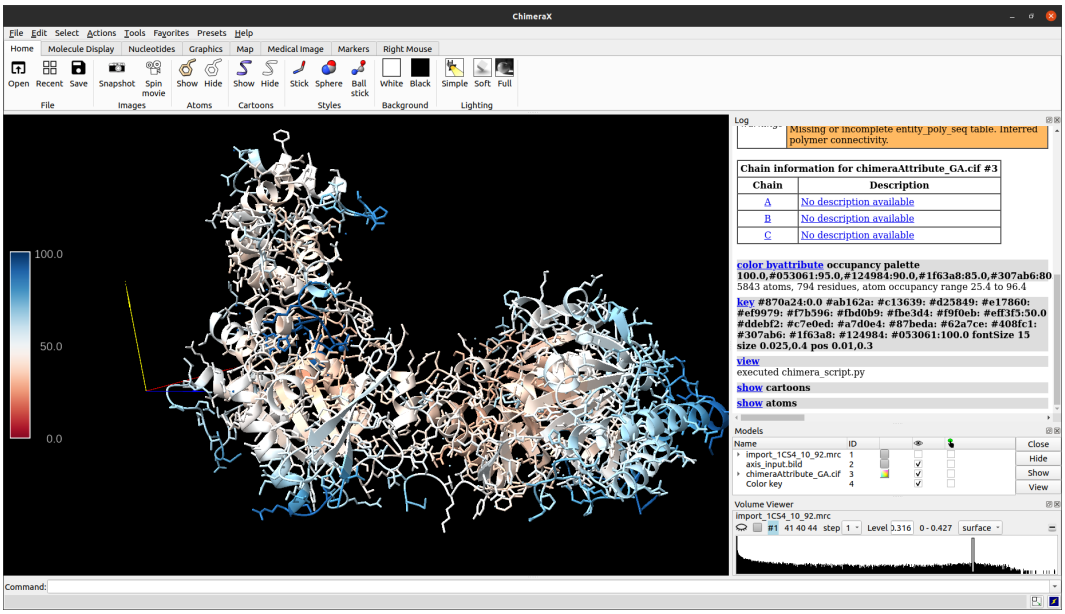


Figura 5.14: Representación 3D tras ejecutar el test gamma-TEMPy desde la terminal

6

Conclusiones

A lo largo de este trabajo se ha desarrollado el *plugin* que implementará la funcionalidad de TEMPy y Gamma TEMPy, el cual era el objetivo de este TFG.

En primer lugar se ha conseguido mostrar los protocolos en el listado de protocolos de Scipion. Acto seguido se ha conseguido instalar la última versión de TEMPy disponible hasta la fecha y preparar un entorno virtual para permitir ejecutar correctamente su código. A continuación se ha conseguido recopilar los argumentos de cada protocolo a partir de un formulario, permitiendo también la selección dinámica de un intervalo de residuos mediante el uso de un wizard, permitiendonos seleccionar los residuos mediante el uso del ratón. Para concluir se han recopilado las entradas y salidas de los protocolos, incluyendo un resumen de los resultados obtenidos, siempre que sea necesario y mostrando de forma visual mediante histrogramas, gráficos y representaciones 3D la compatibilidad entre el mapa de densidad y el modelo atómico.

Se ha realizado un test para cada uno de los protocolos del *plugin*. Mediante la resolución positiva de estos tests y una posterior revisión manual de los resultados obtenidos se puede determinar que la funcionalidad principal del *plugin* se cumple.

El proyecto desarrolla todas las funcionalidades descritas, sin embargo, todavía existen muchos posibles puntos de mejora.

6.1. Trabajo Futuro

Tal y cómo se puede ver en el apartado anterior, consideramos que la funcionalidad del *plugin* se ha desarrollado satisfactoriamente, sin embargo, esto no significa que no exista ningún trabajo a realizar en el futuro.
Centrandonos en el *plugin* desarrollado se podrían llevar a cabo las siguientes mejoras:

- Comparar dos modelos atómicos con un mismo mapa de densidad en el primer protocolo.
- Automatizar el proceso para detectar nuevas versiones de TEMPy y actualizar el código fuente de nuestro *plugin*.
- Sacar varias salidas en el algoritmo genético para el segundo protocolo.
- Incluir nueva funcionalidad implementada por futuras actualizaciones de TEMPy.

Finalmente, dado que Scipion es una aplicación que permite añadir nuevas funcionalidades a partir de *plugins*, las posibilidades de trabajo futuro son infinitas, considerando que cada día se realizan nuevos avances y mejoras en el campo de la Biología Estructural que podrían ser incorporados a Scipion.

7

Definiciones

Macromoléculas biológicas: son moléculas de gran tamaño (generalmente constan de miles de átomos) formadas por la polimerización de varias moléculas más pequeñas, denominadas monómeros. Las macromoléculas biológicas más importantes de los seres vivos son 4: los carbohidratos, los lípidos, las proteínas y los ácidos nucleicos (ADN y ARN).

Proteína: las proteínas son moléculas grandes y complejas que desempeñan muchas funciones críticas en el cuerpo. Realizan la mayor parte del trabajo en las células y son necesarias para la estructura, función y regulación de los tejidos y órganos del cuerpo. Las proteínas están formadas por cientos o miles de unidades más pequeñas llamadas aminoácidos, que se unen entre sí en largas cadenas. Hay 20 tipos diferentes de aminoácidos que se pueden combinar para formar una proteína. La secuencia de aminoácidos determina la estructura tridimensional única de cada proteína y su función específica.

Residuo: es cada uno de los aminoácidos que forman un péptido o una proteína. Son compuestos orgánicos que poseen un grupo carboxilo y un grupo amino. Existen 20 aminoácidos diferentes que forman parte de las proteínas. Todos ellos tienen una parte de su molécula en común (formada por el átomo de carbono unido a los grupos amino y carboxilo) y difieren entre sí en la naturaleza de la cadena lateral (habitualmente llamada grupo R).

Mapa de densidad obtenido por microscopía electrónica: representación tridimensional de macromoléculas o agregados macromoleculares obtenidas aplicando métodos matemáticos y computacionales a las micrografías electrónicas obtenidas de los diagramas de difracción del haz de electrones que “ilumina” la muestra.

Modelo atómico tridimensional de una proteína: representación tridimensional de una proteína en la que se representan la disposición de los átomos que la componen, así como los enlaces que unen dichos átomos.

Bibliografía

[1] Euroinnova, "¿qué estudia la biología estructural?." Main page of Euroinnova for Structural Biology course.

[2] "Biología molecular: definición y aplicaciones," *INFINITIA INDUSTRIAL CONSULTING*, vol. 1, July 2021. Download.

[3] H. Shimizu, "Cómo descifrar estructuras biológicas en busca de medicamentos," *Agencia FAPESP*, vol. 1, Febraury 2019. Download.

[4] A. J. Moreno, L. del Caño, M. Martínez, E. R. Aportela, A. Cuervo, R. Melero, R. S. García, D. Strelak, E. F. Giménez, F. de Isidro Gómez, D. Herreros, P. Conesa, Y. Fonseca, D. Maluenda, J. J. de la Morena, J. Macías, P. Losana, R. Marabini, J. Carazo, and C. Sorzano, "Cryo em and single particle analysis with scipion.," *JoVE*, 2021. See.

[5] B. C. U. of London, "Tempy2 documentation." Main page of TEMPy code documentation, 2015.

[6] F. A. Arun Prasad Pandurangan, Daven Vasishtan and M. Topf, "Tempy: Simultaneous fitting of components in 3d-em maps of their assembly using a genetic algorithm," *ScienceDirect*, vol. 1, November 2015. See.

[7] K. M. Yip, N. Fischer, E. Paknia, A. Chari, and H. Stark, "Breaking the next cryo-em resolution barrier – atomic resolution determination of proteins!," *bioRxiv*, 2020. See.

[8] E. Nogales, "Breve historia de la criomicroscopía electrónica," *INVESTIGACIÓN Y CIENCIA*, vol. 1, May 2016. Download.

[9] bbm.Sociedad Española de Bioquímica y Biología molecular, "Técnicas de imagen nº 201," September 2019.

[10] A. O. C.O.S. Sorzano, "Métodos computacionales para biología estructural," Octouber 2021.

[11] A. S. Martín, "Integration of tridimensional reconstruction and classification algorithms for single particle analysis," Master's thesis, Univ. San Pablo CEU, 2020.

[12] V. Abrishami, *New computational methods toward atomic resolution in single particle cryo-electron microscopy.* PhD thesis, Univ. Autónoma de Madrid, 2016.

[13] P. F. Martínez, "Implementation of molecular dynamics workflow in scipion and its validation," Master's thesis, Univ. Politécnica de Madrid, 2021.

[14] A. P. Pérez, "Integración en scipion del software dedocking molecular rosetta darc para su aplicación en descubrimiento y reposicionamiento de fármacos," Master's thesis, Univ. Autónoma de Madrid, 2021.

[15] S. Team, "How to install scipion," 2022. See.

[16] T. Cragnolini, H. Sahota, A. P. Joseph, A. Sweeney, S. Malhotra, D. Vasishtan, and M. Topf, "How to install tempy," 2015. See.

[17] S. Team, "Developers page," 2022. See.

[18] F. Reina, “scipion-em-template,” Octouber 2020. See.

[19] T. Cragnolini, H. Sahota, A. P. Joseph, A. Sweeney, S. Malhotra, D. Vasishtan, and M. Topf, “Tempy2: a python library with improved 3d electron microscopy density-fitting and validation workflows.,” *Acta crystallographica Section D*, vol. 77, pp. 41–47, 2021.

[20] D. del Hoyo, “Scipion-chem wizards,” 2022. See.

Apéndices

A

Anexo A: Código Implementado

A.1. __init__.py

Código A.1: Función _defineVariables

```
1  @classmethod
2  def _defineVariables(cls):
3      """ Return and write a variable in the config file.
4      """
5      cls._defineVar(TEMPY_ENV_ACTIVATION, DEFAULT_ACTIVATION_CMD)
6      cls._defineEmVar(TEMPY_HOME, TEMPY + '-' + TEMPY_DEFAULT_VERSION)
```

Código A.2: Función defineBinaries

```
1  @classmethod
2  def defineBinaries(cls, env):
3      _url = "https://tempy.ismb.lon.ac.uk/tempy2.tar.xz"
4      cls.addTempyPackage(env, _url)
```

Código A.3: Función runTEMPy

```
        """ Run TEMPY script from a given protocol. """
        tempyDir = os.path.join(cls._tempyHome, 'Predict_Result')
        fullProgram = '%s %s && mkdir -p %s && %s' % (cls.getCondaActivationCmd(),
              cls.getTEMPYEnvActivation(), tempyDir, 'pipenv run python')
        if mode == "SCORES":
            inResidues = protocol.inResidues.get().split("\n")[:-1]
            out = ""
            print("inResidues", inResidues)
            for residuesJson in inResidues:
                residueInterval = json.loads(residuesJson)
                chain = residueInterval["chain"]
                residues = residueInterval["index"].split("-")
                out = out + residues[0]+":"+chain + " " + residues[1]+":"+chain + " "
            args = '{}/calculate_scores.py '.format(cls._tempyHome) + args
            rigid_file_path = os.path.join(cls._tempyHome, "rigid_body.txt")
            rigid_file = open(rigid_file_path, "w")
            rigid_file.write(out)
            rigid_file.close()
            args = args + " -rf " + rigid_file_path
        elif mode == "GA":
            args = '{}/gamma_tempy.py '.format(cls._tempyHome) + args
        args = args + "; conda deactivate"
        protocol.runJob(fullProgram, args, cwd=cls._tempyHome)
        if outDir is None:
            outDir = protocol._getExtraPath('predictions')
        shutil.copytree(tempyDir, outDir)
        if clean:
            shutil.rmtree(tempyDir)

    @classmethod
```

Código A.4: Función getEnviron

```
    @classmethod
    def getEnviron(cls, tempyFirst=True):
        environ = pwutils.Environ(os.environ)
        pos = pwutils.Environ.BEGIN if tempyFirst else pwutils.Environ.END
        environ.update({
            'PATH': cls.getHome('bin'),
            'CLIB': cls.getHome('lib'),
            'CLIBD': cls.getHome('lib/data'),
            'CLIBDMON': cls.getHome('lib/data/monomers'),
        }, position=pos)

        return environ
```

Código A.5: Función getTEMPYEnvActivation

```
1     @classmethod
2     def getTEMPYEnvActivation(cls):
3         return cls.getVar("TEMPY_ENV_ACTIVATION")
```

A.2. constants.py

Código A.6: Fichero constants

```
1  import os
2
3  TEMPY_HOME = 'TEMPY_HOME'
4  TEMPY_HOME = 'TEMPY_HOME'
5
6  # Supported Versions
7  V1_0 = '1.0'
8
9  TEMPY_DEFAULT_VERSION = V1_0
10
11 DEFAULT_ENV_NAME = "tempy-env"
12 DEFAULT_ACTIVATION_CMD = "conda activate " + DEFAULT_ENV_NAME
13 TEMPY_ENV_ACTIVATION = "TEMPY_ENV_ACTIVATION"
14
15 TEMPY_CUDA_LIB = "TEMPY_CUDA_LIB"
16
17 TEMPYLAB = 'tempy'
18 TEMPY = 'TEMPY'
19
20 PYMOL_HOME = 'PYMOL_HOME'
21
22 TEMPY_TAR_XZ = 'tempy.ismb.lon.ac.uk/tempy2.tar.xz'
```

A.3. protocol_tempy.py

Código A.7: Fichero protocol_tempy.py Funcion _defineParams

```
1   def _defineParams(self, form):
2       form.addSection(label=Message.LABEL_INPUT)
3       group = form.addGroup('Input')
4
5       group.addParam('inputAtomStruct', params.PointerParam,
6                      pointerClass='AtomStruct', allowsNull=False,
7                      label="Input protein assembly: ",
8                      help='Select the protein assembly to be fit')
9
10      group.addParam('inputVolume', params.PointerParam,
11                     pointerClass='Volume', allowsNull=False,
12                     label="Input map: ",
13                     help='Select the map you want to fit into')
14
15      group.addParam('chimeraResampling', params.BooleanParam,
16                     label="Resample using ChimeraX: ", default=True,
17                     help='Resample volume to 1.0 using ChimeraX software')
18
19      group.addParam('reso', params.FloatParam,
20                     default=10,
21                     label='Resolution', important=True,
22                     help='Resolution of the map, which will be used by TEMPy to blur the protein
                           conformation, and compare it to the map.')
```

Código A.8: Fichero protocol_tempy.py Funcion _insertAllSteps

```
1   def _insertAllSteps(self):
2       self._insertFunctionStep('convertInputStep')
3       self._insertFunctionStep('ScoreStep')
4       self._insertFunctionStep('createOutputStep')
```

Código A.9: Fichero protocol_tempy.py Funcion convertInputStep

```
1  def convertInputStep(self):
2      name, ext = os.path.splitext(self.getStructFile())
3      pdbFile = self.getPdbStruct()
4      shutil.copy(self.getStructFile(), pdbFile)
5
6      #Renaming volume to add protocol ID (results saved in different directory in scores repo)
7      localVolumeFile = self.getLocalVolumeFile()
8      shutil.copy(self.getVolumeFile(), localVolumeFile)
```

Código A.10: Fichero protocol_tempy.py Funcion ScoreStep

```
1  def ScoreStep(self):
2      args = self.getScoreArgs()
3
4      Plugin.runTEMPy(self, args=args, mode=self._MODE)
```

Código A.11: Fichero protocol_tempy.py Funcion _citations

```
1  def _citations(self):
2      return ["Cragnolini2021"]
```

Código A.12: Fichero protocol_tempy.py Funcion chimeraResampleScript

```
1
2  def chimeraResampleScript(self, inVolFile, newSampling, outFile):
3      scriptFn = self._getExtraPath('resampleVolume.cxc')
4      with open(scriptFn, 'w') as f:
5          f.write('cd %s\n' % os.getcwd())
6          f.write("open %s\n" % inVolFile)
7          f.write('vol resample #1 spacing {}\n'.format(newSampling))
8          f.write('save {} model #2\n'.format(outFile))
9          f.write('exit\n')
10     return scriptFn
```

Código A.13: Fichero protocol_tempy.py Funcion chimeraResample

```
1
2   def chimeraResample(self, inVolFile, newSR, outFile):
3       with weakImport("chimera"):
4           from chimera import Plugin as chimeraPlugin
5           resampleScript = self.chimeraResampleScript(inVolFile, newSampling=newSR, outFile=outFile)
6           chimeraPlugin.runChimeraProgram(chimeraPlugin.getProgram() + ' --nogui --silent',
7                                           resampleScript, cwd=os.getcwd())
8           while not os.path.exists(outFile):
9             time.sleep(1)
10      return outFile
```

A.4. protocol_fit_scores.py

Código A.14: Fichero protocol_fit_scores.py Funcion

```
1   def _defineParams(self, form):
2       """ """
3       super()._defineParams(form)
4
5       group = form.addGroup('Parameters')
6
7       group.addParam('chain_name', params.StringParam,
8                      allowsNull=False, label='Chain_of_interest',
9                      help='Specify_the_chain_of_the_residue_of_interest')
10      group.addParam('resPosition', params.StringParam,
11                     allowsNull=False, label='Residues_of_interest',
12                     help='Specify_the_residue_to_define_a_region_of_interest.\n'
13                          'You_can_either_select_a_single_residue_or_a_range_'
14                          '(it_will_take_into_account_the_first_and_last_residues_selected)')
15      group.addParam('addResidue', params.LabelParam,
16                     label='Add_defined_residue',
17                     help='Here_you_can_define_a_residue_which_will_be_added_to_the_list_of_residues_
                            below.')
18      group.addParam('inResidues', params.TextParam, width=70, default='', allowsNull=True,
19                     help='Input_residues_to_define_the_pocket._'
20                          'The_coordinates_of_the_residue_atoms_will_be_mapped_to_surface_points_
                                closer_than_maxDepth_'
21                          'and_points_closer_than_maxIntraDistance_will_be_considered_the_same_
                                pocket')
```

Código A.15: Fichero protocol_fit_scores.py Funcion createOutputStep

```
def createOutputStep(self):
    outStructSMOCFileName = self._getPath('outputStructureSMOC.cif')
    outStructSCCCFileName = self._getPath('outputStructureSCCC.cif')
    outSMOCFile = self._getExtraPath('predictions/smoc.pdb')
    outSCCCFile = self._getExtraPath('predictions/sccc.pdb')

    ASH = AtomicStructHandler()
    # Write SMOC_score in a section of the output cif file
    ScoresDic = self.parseSMOCScores(outSMOCFile)
    inpAS = toCIF(self.inputAtomStruct.get().getFileName(), self._getTmpPath('inputStruct.cif'))
    cifDic = ASH.readLowLevel(inpAS)
    cifDic = addScipionAttribute(cifDic, ScoresDic, "SMOC")
    ASH._writeLowLevel(outStructSMOCFileName, cifDic)

    AS = AtomStruct(filename=outStructSMOCFileName)
    outVol = self._getInputVolume().clone()
    outVol.setLocation(self.getLocalVolumeFile())
    if self.chimeraResampling and self._getInputVolume().getSamplingRate() != 1.0 and
            self.chimeraInstalled():
        outVol.setSamplingRate(1.0)
    AS.setVolume(outVol)
    self._defineOutputs(**{"SMOC": AS})

    # Write SCCC_score in a section of the output cif file
    ScoresDic = self.parseFitScores(outSCCCFile)
    inpAS = toCIF(self.inputAtomStruct.get().getFileName(), self._getTmpPath('inputStruct.cif'))
    cifDic = ASH.readLowLevel(inpAS)
    cifDic = addScipionAttribute(cifDic, ScoresDic, "SCCC")
    ASH._writeLowLevel(outStructSCCCFileName, cifDic)

    AS = AtomStruct(filename=outStructSCCCFileName)
    outVol = self._getInputVolume().clone()
    outVol.setLocation(self.getLocalVolumeFile())
    if self.chimeraResampling and self._getInputVolume().getSamplingRate() != 1.0 and
            self.chimeraInstalled():
        outVol.setSamplingRate(1.0)
    AS.setVolume(outVol)
    self._defineOutputs(**{"SCCC": AS})
```

Código A.16: Fichero protocol_fit_scores.py Funcion _summary

```
1   def _summary(self):
2       summary = []
3
4       outDir = self._getExtraPath('predictions')
5       with open(os.path.join(outDir, 'calculate_scores.txt'), "r") as f:
6           line = f.readline()
7           while line:
8               summary.append(line.replace("\n", ""))
9               line = f.readline()
10      return summary
```

Código A.17: Fichero protocol_fit_scores.py Funcion getScoreArgs

```
1   def getScoreArgs(self):
2       args = '␣-m␣{}␣-p␣{}␣-r␣{}'. \
3         format(os.path.abspath(self.getLocalVolumeFile()), os.path.abspath(self.getPdbStruct()),
4                self.reso.get())
5
6       return args
```

Código A.18: Fichero protocol_fit_scores.py Funcion parseFitScores

```
1   def parseFitScores(self, pdbFile):
2       '''Return a dictionary with {spec: value}
3       "spec" should be a chimera specifier. In this case: chainId:residueIdx'''
4       fitDic = {}
5       with open(pdbFile) as f:
6           for line in f:
7               if line.startswith('ATOM') or line.startswith('HETATM'):
8                   resId = '{}:{}'.format(line[21].strip(), line[22:26].strip())
9                   if not resId in fitDic:
10                      fitScore = line[60:66].strip()
11                      fitDic[resId] = fitScore
12      return fitDic
```

Código A.19: Fichero protocol_fit_scores.py Funcion parseSMOCScores

```
1   def parseSMOCScores(self, pdbFile):
2       outputFiles = os.listdir(self._getExtraPath('predictions'))
3       csvFiles = [out for out in outputFiles if ".csv" in out]
4       csvSMOC= [out for out in csvFiles if "smoc" in out]
5
6       fitDic = {}
7       from numpy import genfromtxt
8       for SMOC in csvSMOC:
9           data = genfromtxt(self._getExtraPath('predictions/'+SMOC), delimiter=';')[1:]
10          chain = SMOC.replace("smoc_chain_", "").replace(".csv", "").strip()
11
12          for row in data:
13              resId = '{}:{}'.format(chain, int(row[0]))
14              if not resId in fitDic:
15                  fitScore = row[1]
16                  fitDic[resId] = fitScore
17
18      return fitDic
```

A.5. protocol_ga.py

Código A.20: Fichero protocol_ga.py Funcion

```
def _defineParams(self, form):
    """ """
    super()._defineParams(form)

    group = form.addGroup('Parameters')

    group.addParam('nga', params.IntParam,
                   default=1, allowsNull=False,
                   label='Number_of_solution', important=True,
                   help='Number_of_solution_to_generate_using_genetic_algorithm')
    group.addParam('ngen', params.IntParam,
                   default=100, allowsNull=False,
                   label='Number_of_generation', important=True,
                   help='Number_of_generation_used_in_the_genetic_algorithm')
    group.addParam('popsize', params.IntParam,
                   default=160, allowsNull=False,
                   label='Population_size', important=True,
                   help='Size_of_the_population_describing_the_number_of_members_(assembly_fits)')
    group.addParam('ncpu', params.IntParam,
                   default=4, allowsNull=False,
                   label='Number_of_CPUs', important=True,
                   help='Number_of_CPUs_to_use_in_parallel_for_the_calculation')
    group_advanced = form.addGroup('Advanced')
    group_advanced.addParam('gof', params.EnumParam,
                choices=self.godChoices,
                default = 0,
                label='Godness_of_fit_score:_',
                help='Goodness-of-fit_score_to_use._Mutual_information_score_(MI)_or_CCC'
                )
    group_advanced.addParam('mrate', params.FloatParam,
                   default=0.2, allowsNull=False,
                   label='Mutation_rate:', important=True,
                   help='Mutation_rate_of_the_genetic_algorithym')
    group_advanced.addParam('crate', params.FloatParam,
                   default=0.8, allowsNull=False,
                   label='Crossover_rate:', important=True,
                   help='Crossover_rate_of_the_genetic_algorithym')
    group_advanced.addParam('seedpoints', params.PointerParam,
                   pointerClass='AtomStruct', allowsNull=True,
                   label="PDB_seed:_",
                   help='Supply_a_PDB_formated_file_containing_the_coordinate_to_use_as_initial_
                       VQ_points_(By_default_the_VQ_points_will_be_generated_from_the_input_
                       density_map)')
```

Código A.21: Fichero protocol_ga.py Funcion createOutputStep

```
    def createOutputStep(self):
        outStructFileName = self._getPath("outputStructureSMOC.cif")
        outFile = self._getExtraPath('predictions/ga_fit_1.pdb')
        outputs = [[self._MODE, outStructFileName, outFile]
]
        ASH = AtomicStructHandler()
        for out in outputs:
            ScoresDic = self.parseFitScores(out[2])
            inpAS = toCIF(self.inputAtomStruct.get().getFileName(), self._getTmpPath('inputStruct.cif'))
            cifDic = ASH.readLowLevel(inpAS)
            cifDic = addScipionAttribute(cifDic, ScoresDic, out[0])
            ASH._writeLowLevel(out[1], cifDic)

            AS = AtomStruct(filename=out[1])
            outVol = self._getInputVolume().clone()
            outVol.setLocation(self.getLocalVolumeFile())
            if self.chimeraResampling and self._getInputVolume().getSamplingRate() != 1.0 and
                    self.chimeraInstalled():
                outVol.setSamplingRate(1.0)
            AS.setVolume(outVol)
            self._defineOutputs(**{out[0]: AS})
```

Código A.22: Fichero protocol_ga.py Funcion getScoreArgs

```
    def getScoreArgs(self):
        args = '␣--imap␣{}␣--ipdb␣{}␣--res␣{}␣--nga␣{}␣--ngen␣{}␣--outfile␣ga_fit␣--outdir␣
                Predict_Result␣--popsize␣{}␣--ncpu␣{}␣--gof␣{}␣--mrate␣{}␣--crate␣{}'. \
            format(os.path.abspath(self.getLocalVolumeFile()),
                os.path.abspath(self.getPdbStruct()),
                    self.reso.get(),
                    self.nga.get(),
                    self.ngen.get(),
                    self.popsize.get(),
                    self.ncpu.get(),
                    self.gof.get()+1,
                    self.mrate.get(),
                    self.crate.get()
                    )
        if self.seedpoints.get() is not None:
            args = args + "␣--seedpoints␣{}".format(self.getPdbStructSeed())
            print("null")
        return args
```

Código A.23: Fichero protocol_ga.py Funcion parseFitScores

```
1   def parseFitScores(self, pdbFile):
2       '''Return a dictionary with {spec: value}
3       "spec" should be a chimera specifier. In this case: chainId:residueIdx'''
4       fitDic = {}
5       with open(pdbFile) as f:
6           for line in f:
7               if line.startswith('ATOM') or line.startswith('HETATM'):
8                   resId = '{}:{}'.format(line[21].strip(), line[22:26].strip())
9                   if not resId in fitDic:
10                      fitScore = line[60:66].strip()
11                      fitDic[resId] = fitScore
12      return fitDic
```

A.6. viewer_fit_scores.py

Código A.24: Fichero viewer_fit_scores.py Funcion _defineParams

```
def _defineParams(self, form):
    # super()._defineParams(form)
    form.addSection(label="Visualization␣of␣attributes")
    form.addParam('attrName', params.EnumParam,
                  choices=self.actionChoices,
                  default = self._SMOC_ID,
                  label='Attribute␣to␣display:␣',
                  help='Display␣this␣attribute␣of␣the␣structure'
                  )

    group = form.addGroup('Attribute␣histogram')
    group.addParam('displayHistogram', params.LabelParam,
                   label='Display␣histogram␣of␣attribute:␣',
                   help='Display␣a␣histogram␣with␣the␣count␣of␣the␣attribute.\n'
                        'The␣color␣palette,␣intervals,␣lowest␣and␣highest␣values␣can␣be␣chosen␣in␣the␣'
                        'parameters␣below.␣If␣the␣highest␣and␣lowest␣values␣you␣input␣are␣the␣same,␣'
                        'the␣lowest␣and␣higher␣values␣in␣the␣date␣are␣used.'
                   )

    group = form.addGroup('Visualization␣in␣Chimera')
    group.addParam('displayStruct', params.LabelParam,
                   label='Display␣structure␣and␣color␣by␣attribute␣in␣Chimera:␣',
                   help='Display␣the␣AtomStruct,␣coloured␣by␣the␣attribute.\n'
                        'The␣color␣palette,␣intervals,␣lowest␣and␣highest␣values␣can␣be␣chosen␣in␣the␣'
                        'parameters␣below.␣If␣the␣highest␣and␣lowest␣values␣you␣input␣are␣the␣same,␣'
                        'the␣lowest␣and␣highest␣values␣in␣the␣date␣are␣used.'
                   )
    # Overwrite defaults
    from pwem.wizards.wizard import ColorScaleWizardBase
    group = form.addGroup('Color␣settings')
    ColorScaleWizardBase.defineColorScaleParams(group, defaultLowest=self._DEFAULT_LOWEST,
        defaultHighest=self._DEFAULT_HIGHEST, defaultIntervals=21,
                                                defaultColorMap='RdBu_r')
    group = form.addGroup('Score␣Plot')
    group.addParam('displayPlot', params.LabelParam,
                   label="Display␣residue␣score␣plot:␣",
                   help="Display␣the␣score␣of␣fitnes␣for␣each␣residue␣in␣the␣current␣chain")
```

Código A.25: Fichero viewer_fit_scores.py Funcion _getOutputObject

```
def _getOutputObject(self):
  return getattr(self.protocol, self.actionChoices[self.attrName.get()])
```

Código A.26: Fichero viewer_fit_scores.py Funcion _getData

```
def _getData(self):
    cifFile = self.getAtomStructObject().getFileName()
    cifDic = AtomicStructHandler().readLowLevel(cifFile)
    #TODO: admit non float attributes
    names, values = np.array(cifDic['_scipion_attributes.specifier']),
            np.array(cifDic['_scipion_attributes.value'])
    data = zip(names, values)
    dataDic = {}

    for d in set(data):
        resId, score = d[0].split(":"), float(d[1])
        resId[1] = int(resId[1])
        if not resId[0] in dataDic:
            dataDic[resId[0]] = {"residues": [], "scores": []}
        dataDic[resId[0]]["residues"].append(resId[1])
        dataDic[resId[0]]["scores"].append(score)
    return dataDic
```

Código A.27: Fichero viewer_fit_scores.py Funcion _displayPlot

```
def _displayPlot(self, paramName=None):
    attrname = self.getEnumText('attrName')

    dataDic = self._getData()
    self.plotter = EmPlotter(x=1, y=len(dataDic), windowTitle=attrname)
    ax = []
    for i, chain in enumerate(dataDic):
        ax.append(self.plotter.createSubPlot("Chain␣"+chain, "scores", "count␣scores"))
        residues, scores = dataDic[chain]["residues"], dataDic[chain]["scores"]

        ziped_list = zip(residues, scores)
        sorted_pairs = sorted(ziped_list, key=lambda y: y[0])
        tuples = zip(*sorted_pairs)
        residues, scores = [ list(tuple) for tuple in tuples]

        ax[i].plot(residues, scores, linestyle='--', markevery=None)
        ax[i].grid(True)

    return [self.plotter]
```

Código A.28: Fichero viewer_fit_scores.py Funcion _showHistogram

```
1   def _showHistogram(self, paramName=None):
2     attrname = self.getEnumText('attrName')
3
4     dataDic = self._getData()
5     self.plotter = EmPlotter(x=1, y=1, windowTitle=attrname)
6     # ax = []
7     a = self.plotter.createSubPlot(attrname, "residues", "scores")
8     for i, chain in enumerate(dataDic):
9       a.hist(dataDic[chain]["scores"], label="Chain␣"+chain)
10
11    a.legend(loc="upper␣right")
12    a.grid(True)
13    return [self.plotter]
```

Código A.29: Fichero viewer_fit_scores.py Funcion _getVisualizeDict

```
1   def _getVisualizeDict(self):
2       return {
3         'displayStruct': self._showChimera,
4         'displayHistogram': self._showHistogram,
5         'displayPlot': self._displayPlot
6       }
```

A.7. wizard_select_chain.py

Código A.30: Fichero wizard_select_chain.py Clase AddResidueWizard2

```
1   class AddResidueWizard2(EmWizard):
2       _targets = [(ProtFitScores, ['addResidue'])]
3
4       def show(self, form, *params):
5           protocol = form.protocol
6           chainDic, resDic = json.loads(protocol.chain_name.get()), json.loads(protocol.resPosition.get())
7           form.setVar('inResidues', protocol.inResidues.get() +
8                       '{"model":␣%s,␣"chain":␣"%s",␣"index":␣"%s",␣"residues":␣"%s"}\n' %
9                       (chainDic['model'], chainDic['chain'], resDic['index'], resDic['residues']))
```

Código A.31: Fichero wizard_select_chain.py Clase SelectChainWizard2

```
class SelectChainWizard2(VariableWizard):
    '''Opens the input AtomStruct and allows you to select one of the present chains'''
    _targets, _inputs, _outputs = [], {}, {}
    @classmethod
    def getModelsChainsStep(cls, protocol, atomStructName='inputAtomStruct'):
      """ Returns (1) list with the information
          {"model": %d, "chain": "%s", "residues": %d} (modelsLength)
          (2) list with residues, position and chain (modelsFirstResidue)"""
      structureHandler = AtomicStructHandler()
      fileName = ""
      AS = getattr(protocol, atomStructName).get()
      if AS is not None:
        fileName = os.path.abspath(AS.getFileName())
        if str(type(AS).__name__) == 'SchrodingerAtomStruct':
          fileName = os.path.abspath(AS.convert2PDB())
        else:
          fileName = os.path.abspath(AS.getFileName())
      structureHandler.read(fileName)
      structureHandler.getStructure()
      return structureHandler.getModelsChains()
    def editionListOfChains(self, listOfChains):
      chainList = []
      for model, chainDic in listOfChains.items():
        for chainID, lenResidues in chainDic.items():
          chainList.append(
            '{"model": %d, "chain": "%s", "residues": %d}' %
            (model, str(chainID), lenResidues))
      return chainList
    def show(self, form, *params):
      inputParam, outputParam = self.getInputOutput(form)
      protocol = form.protocol
      try:
        listOfChains, listOfResidues = self.getModelsChainsStep(protocol, inputParam[0])
      except Exception as e:
        print("ERROR: ", e)
        return
      chainList = self.editionListOfChains(listOfChains)
      finalChainList = []
      for i in chainList:
        finalChainList.append(String(i))
      provider = ListTreeProviderString(finalChainList)
      dlg = dialog.ListDialog(form.root, "Model chains", provider,
                              "Select one of the chains (model, chain, "
                              "number of chain residues)")
      form.setVar(outputParam[0], dlg.values[0].get())
```

Código A.32: Fichero wizard_select_chain.py Clase SelectCSelectResidueWizard2hainWizard2

```
class SelectResidueWizard2(SelectChainWizard2):
    _targets, _inputs, _outputs = [], {}, {}
    def editionListOfResidues(self, modelsFirstResidue, model, chain):
        residueList = []
        for modelID, chainDic in modelsFirstResidue.items():
            if int(model) == modelID:
                for chainID, seq_number in chainDic.items():
                    if chain == chainID:
                        for i in seq_number:
                            residueList.append('{"index": %d,"residue": "%s"}' % (i[0], str(i[1])))
        return residueList
    def getResidues(self, form, inputParam):
        protocol = form.protocol
        inputName = inputParam[0]
        if type(inputName) == list:
            inputName = inputName[getattr(protocol, inputParam[1]).get()]
        inputObj = getattr(protocol, inputName).get()
        if issubclass(type(inputObj), AtomStruct):
            try:
                modelsLength, modelsFirstResidue = self.getModelsChainsStep(protocol, inputName)
            except Exception as e:
                print("ERROR: ", e)
                return
            selection = getattr(protocol, inputParam[1]).get()
            struct = json.loads(selection) # From wizard dictionary
            chain, model = struct["chain"].upper().strip(), int(struct["model"])
            residueList = self.editionListOfResidues(modelsFirstResidue, model, chain)
            finalResiduesList = []
            for i in residueList:
                finalResiduesList.append(String(i))
        elif issubclass(type(inputObj), Sequence) or issubclass(type(inputObj), SequenceVariants):
            finalResiduesList = []
            for i, res in enumerate(inputObj.getSequence()):
                stri = '{"index": %s,"residue": "%s"}' % (i + 1, RESIDUES1TO3[res])
                finalResiduesList.append(String(stri))
        return finalResiduesList
    def show(self, form, *params):
        inputParam, outputParam = self.getInputOutput(form)
        finalResiduesList = self.getResidues(form, inputParam)
        provider = ListTreeProviderString(finalResiduesList)
        dlg = dialog.ListDialog(form.root, "Chain residues", provider,
                                "Select one residue (residue number, "
                                "residue name)")
        roiStr = ''
        idxs = [json.loads(dlg.values[0].get())['index'], json.loads(dlg.values[-1].get())['index']]
        for i in range(len(finalResiduesList)):
            residue = json.loads(finalResiduesList[i].get())
            if idxs[0] <= residue['index'] <= idxs[1]:
                roiStr += RESIDUES3TO1[residue['residue']]

        intervalStr = '{"index": "%s-%s","residues": "%s"}' % (idxs[0], idxs[1], roiStr)
        form.setVar(outputParam[0], intervalStr)
```


UAM
UNIVERSIDAD AUTONOMA
DE MADRID