**Pipeline desarrollo de complejos antígeno- anticuerpo.**

Se desea obtener un modelo de los complejos correspondientes a auto antígenos y anticuerpos de pacientes con leucemia. Debido al reducido número de anticuerpos (45) que poseen una cadena pesada y ligera, estos fueron modelados empleando AlphaFold2 (tarea no realizada por mi). Por otra parte, el número de antígenos presentes en esta tarea hace muy difícil realizar un modelado de estas estructuras, debido a que cada modelo demora algunas horas, y modelar cerca de 8000 estructuras requeriría meses. Es por esto que se optó por emplear blast contra la base de datos de PDB. Cada secuencia de antígeno fue sometida a esta búsqueda, y los resultados obtenidos para cada antígeno se almacenaron en un archivo CSV. Estos archivos contienen la información con respecto al código PDB de la estructura con la cual hizo match, la cadena específica con la cual se encontró similitud, el largo del alineamiento y el e-value. Dentro del script que genera esta búsqueda y registra los resultados, se filtra que las estructuras corresponden a homo sapiens y que posean un e-value menor o igual a 0.5.

*Los archivos con los resultados se llaman Id\_antígeno\_result\_search.txt*

*Hs-MGC-BC011913.2-uORF-IOH14590\_result\_search.txt*

Luego, se descarga el mejor resultado obtenido para este antígeno, y que además se encuentre disponible en formato PDB. Esto es requisito debido a que los scripts que emplean python y el pipeline en general se basan en estructuras en formato PDB.

Estos antígenos son procesados por la función *get\_chain\_antigen* la cual toma la estructura descargada para el antígeno correspondiente y extrae la cadena con la cual se hizo match de acuerdo a los resultados entregados por blast, a la cual se le cambia el Id para trabajar en todas las estructuras con el mismo patron. Esta cadena es guardada en un archivo PDB aparte, y será la estructura modelo del antígeno para los siguientes pasos. Se realizó de esta forma debido a que, contando solo con la secuencia del antígeno, no se puede asegurar si es un fragmento de otra molécula más grande, o conforma una macromolécula de mayor número de cadenas. Es por esto que se trabaja solo con la información entregada por la secuencia, realizando una aproximación dado que la secuencia del antígeno, y la estructura, no es exactamente la que se emplea experimentalmente. Es posible conseguir resultados más cercanos a la realidad generando modelos de estas moléculas, pero como fue mencionado anteriormente, por motivos de tiempo esto no se realizó.

Posteriormente, la función *make\_folder\_and\_copy\_structures* genera una nueva carpeta en la cual almacenar las estructuras a trabajar dentro del complejo, es decir, cadena pesada y ligera del anticuerpo, y el archivo PDB que contiene solo la cadena de antígeno con la cual trabajar. El script filtra que se encuentren ambas cadenas de anticuerpo, y un solo archivo PDB de antígeno. Esto se debe a que aún faltan algunos modelos para cadenas pesadas y ligeras de anticuerpo, y se requiere de ambas estructuras para generar los modelos. Por otra parte, se evalúa que en la interacción empleada se encuentre un antígeno que coincida con los resultados guardados para ese antígeno. La información de los archivos, los ids de las cadenas presentes, y las cadenas a emplear, para cada interacción específica, son retornados a una lista, que se emplea en las funciones posteriores. La función requiere que se pase la ruta donde se encuentran los anticuerpos, la ruta donde se encuentran los antígenos, el Id del antígeno y del anticuerpo para la interacción a realizar, un archivo resumen con la información de estos antígenos, y la ruta en la cual se guardará la carpeta con los archivos correspondientes a cada complejo.

El script que genera el archivo resumen con la información almacenada para los antígenos corresponde a *make\_antigen\_information\_file.py* el cual requiere se especifique la ruta donde se encuentran los txt con los resultados, la ruta donde se almacenaron los antígenos con las estructuras seleccionadas para cada una, y una ruta en la cual guardar el archivo CSV con toda esta información.

A continuación, los archivos correspondientes al complejo son reparados. Esto como precaución a que existan aminoácidos incompletos dentro de las estructuras. Para esto, se empleó la herramienta PDBfixer, la cual puede ser descargada empleando Conda en el siguiente link <https://anaconda.org/omnia/pdbfixer> . Dentro de la función *repair\_pdbs\_to\_complex* se encuentran y completan los aminoácidos que podrían estar incompletos, se reemplaza cualquier residuo no estándar, se remueven los heteroátomos dentro del archivo PDB (moléculas de agua, iones o otros posibles residuos de cristalización o ligandos) y se añaden los hidrógenos correspondientes a los aminoácidos en pH 7.0. Finalmente, estas estructuras son almacenadas en la carpeta en la cual se encuentra el complejo, con la palabra repaired en el nombre, para reconocer aquellas que están corregidas.

Para poder aproximar las cadenas pesadas y livianas del anticuerpo a su disposición tridimensional (colocar cerca a ambas cadenas), se emplea una estructura tridimensional modelo, que presenta las cadenas pesada y ligera, contra la cual es posible comparar ambas cadenas de un anticuerpo de interés. De esta forma, la función *change\_chains\_antibodies* cumple dos tareas. En primer lugar, para cada cadena se modifica el ID, para coincidir con la información requerida (de A a H en cadena pesada y a L en cadena ligera). Luego, se carga la estructura experimental de anticuerpo contra la cual se realizará el alineamiento. Una vez cargada la estructura, se alinean o arrastran las moléculas de la cadena de anticuerpo a las posiciones establecidas por la estructura guía.

Posterior a esto, es necesario unir en un solo archivo las cadenas pesadas y ligeras del anticuerpo, y la cadena seleccionada del antígeno. En la función *make\_complex\_pdb* se cargan en pymol los 3 archivos reparados de anticuerpo y antígeno. Se seleccionan estas tres estructuras y posteriormente se guardan en un archivo único, el cual lleva por nombre el Id del anticuerpo - Id del antígeno.

Es necesario, para ejecutar los siguientes pasos dentro de Rosetta, que las cadenas sean remuneradas, de esta forma, el residuo de Id hace referencia solo a una molécula dentro del PDB. Esto se realiza en la función *renumber\_chains*, el cual lee la primera línea del archivo que contiene el complejo. Esto se debe a que, dado que la primera cadena en el archivo corresponderá a la cadena A (la única de la cual se tiene seguridad no comienza en 1), este será el punto de referencia para renumerar los siguientes aminoácidos. Se selecciona cada cadena por separado, y se guardan en variables el largo de la cadena de antígeno y de la cadena pesada. Esto se debe a que en el archivo del complejo, las cadenas siempre se encuentran ordenadas alfabéticamente. Con esta información, los residuos en la cadena A son re enumerados comenzando desde el entero encontrado originalmente menos este mísmo menos 1. Luego, la cadena pesada es re numerada comenzando desde el entero correspondiente al largo de la cadena A. Similar aproximación se aplica para la cadena pesada, comenzando desde el largo de la cadena de antígeno más la cadena pesada del anticuerpo. Este proceso genera que el complejo sea enumerado comenzando desde el 1 hasta el largo total de la cadena pesada + ligera + el antígeno. Luego, las cadenas re enumeradas son guardadas dentro de un archivo PDB, correspondiente al complejo corregido.

Con el complejo preparado de esta forma, ya es posible comenzar con los preparativos para realizar el docking. En primer lugar, es necesario realizar una preparación o prepacking de las cadenas. Este proceso lo que hace es tomar las cadenas del archivo PDB, y someterlas a una serie de modificaciones geométricas que eliminen posibles colisiones entre los aminoácidos, u orientaciones altamente energéticas. Este proceso es realizado a todas las posiciones dentro de las cadenas presentes en el complejo. Esto ayuda a que todos los aminoácidos, incluyendo aquellos que sean relevantes en la interacción, alcancen un mínimo energético, y una posible posición que ayude a su interacción. Este proceso se ejecuta en ambos pipelines de forma diferente, en el primer pipeline, se ejecuta el comando a rosetta prepack desde el mismo python, empleando la función *repack\_complex*. El segundo pipeline, en cambio, ejecuta este comando dentro del sbatch, entregando al comando la estructura del complejo y la ruta en la cual generar los archivos PDB con los modelos obtenidos, así como la tabla de puntajes de estos. En ambos casos se considera generar 50 estructuras, de forma que sean exploradas varias posiciones en las cadenas laterales de los aminoácidos.

Luego, se analizan los resultados dentro de la tabla de puntajes, y se selecciona el mejor resultado de acuerdo a dos criterios. En primer lugar se analiza el score total, el cual refleja en si la suma de las energías dentro del complejo, idealmente, dentro de un sistema equilibrado esta energía será más negativa. Es por esto que se define que el score tiene que ser menor a cero. Por otra parte, se consideran las fuerzas de repulsión dentro del sistema, las cuales corresponden a fuerzas de repulsión Lennard- Jones dentro de átomos de distintos aminoácidos. De esta forma se puede evaluar si existen muchos residuos en posiciones desfavorables energéticamente. Para esto se estableció un parámetro de 20 (de acuerdo a las observaciones de los puntajes en el proceso de prueba del pipeline), pero este valor puede modificarse con respecto a los resultados observados de forma general. El objetivo de esto es lograr un equilibrio, entre el menor score, y el menor valor de fuerzas de repulsión. Estos valores se van actualizando de acuerdo a los resultados almacenados anteriormente. De esta forma, debería quedar almacenado el mejor resultado como el nombre del complejo a utilizar en los siguientes pasos, el cual corresponde a la variable que se retorna en la función *process\_results\_repack.*

Una vez se ha seleccionado la mejor estructura de acuerdo a la minimización energética realizada anteriormente, es necesario preparar los archivos para el docking. Este procedimiento requiere un archivo con las opciones y un archivo con el protocolo correspondiente. El protocolo se encuentra en formato XML, y considera una serie de parámetros no modificables entre complejos. No obstante, se define un parámetro específico para cada complejo, dependiendo del anticuerpo involucrado. Esto corresponde a la información de los CDR para cada cadena, los cuales corresponden a las posibles regiones en las cuales se produzca la interacción. Es por esto que, pasada la minimización anterior, dentro del protocolo para docking se definen estos aminoácidos para evitar que su disposición tridimensional se modifique dentro de las posiciones de docking. La función *make\_protocol\_docking* encuentra el anticuerpo correspondiente dentro del archivo de predicciones de CDRs, y extrae la secuencia para cadena pesada y ligera. Luego, el archivo PDB del complejo es leído como una secuencia lineal. Las secuencias cdr son buscadas dentro de la secuencia del complejo, y se retorna el índice en el cual comienza a encontrarse este patrón. En base a esta información, y al largo del cdr correspondiente, se genera una lista con los índices de los residuos correspondientes a estas regiones. Esta lista se emplea para especificar dentro del protocolo que éstos aminoácidos no deben ser modificados. Este protocolo es posteriormente exportado al archivo XML. Por otra parte, dentro de la función *make\_docking\_options* se modifica el archivo con las opciones, en el cual se especifica la estructura complejo a emplear. A esta función se le entrega la selección generada de los resultados de repacking.

De forma similar a la ejecución de repacking, la ejecución del comando de docking se realiza de forma distinta en ambos posibles pipelines. En el caso del primer pipeline, la ejecución de esta línea de comando se realiza dentro del script de python. En cambio en el segundo pipeline se ejecuta dentro del sbatch, entregando el archivo de opciones, el protocolo y la ruta donde se guardarán los resultados de docking. De acuerdo con las indicaciones realizadas por Rosetta, se aconseja realizar alrededor de 10000 dockings, con el fin de explorar ampliamente todas las posibles opciones. Este número podría ser re definido de acuerdo a la disposición de espacio en disco duro. SIn embargo, es importante recordar que estos son los parámetros sugeridos por Rosetta, por lo cual, reduciendo significativamente el número de resultados, es posible que se dejen opciones viables fuera de los resultados.

Una vez finalizado este proceso, se analiza la tabla de puntajes entregada por Rosetta en la función *select\_best\_docking*, y se evalúan dos criterios. El primero, al igual que en el prepacking, corresponde al score total, el cual refleja la suma de las energías dentro del complejo, y por lo tanto, mientras más negativo, mejor. Por otra parte, se utiliza el parámetro correspondiente al puntaje de interfaz, el cual corresponde a la puntuación total del complejo menos la puntuación total de cada cadena de forma aislada. De acuerdo con lo presentado por Rosetta, este valor debe fluctuar entre -5 y -10 para que sea considerado un buen docking. Es por esto que se filtran resultados que posean el menor valor de energía y cuyo puntaje de interfaz se encuentre dentro de estos límites.

Finalmente, una vez seleccionado el mejor docking de acuerdo a estos parámetros, el resultado seleccionado es movido a la carpeta en la cual almacenar los resultados correspondientes a todos los complejos a realizar, mientras que los archivos considerados temporales (todos los PDB que sufrieron cambios, así como los archivos de complejo temporales), son eliminados para liberar espacio. Esta labor se realiza en la función *move\_file\_selected*.

En cuanto al desarrollo de los pipelines, la diferencia principal está dada por la ejecución de los procesos dependientes de Rosetta. En el primer pipeline se considera que se imprime al terminal el comando a ejecutar, empleando parámetros entregados dentro del mismo script. Por otra parte, el segundo pipeline ejecuta estas líneas de comando dentro del sbatch, el cual también llama a los script en python que realizan la preparación y análisis de los resultados. Desconozco cual de estas dos aproximaciones sea la recomendada, por lo cual se presentan ambas opciones. Sin embargo, como se mencionó anteriormente, la diferencia se encuentra simplemente en la ejecución de las dependencias de Rosetta