UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Identificación de caminos alostéricos en proteínas empleando simulaciones de dinámica molecular, teoría de gráficas y detección de similitudes locales.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

FÍSICO

P R E S E N T A:

Sergio Camposortega Rendón

DIRECTOR DE TESIS:
Dr. Marcelino Arciniega Castro
Ciudad Universitaria, CD. MX., 2019

Hoja de Datos del Jurado

1. Datos del alumno
   Camposortega
   Rendón
   Sergio
   55683861
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Física
   307038957

2. Datos del tutor
   Dr.
   Marcelino
   Arciniega
   Castro

3. Datos del sinodal 1
   Dr.
   Raúl Arturo
   Espejel
   Morales

4. Datos del sinodal 2
   Dr.
   Augusto César
   Poot
   Hernández

5. Datos del sinodal 3
   Dra.
   Claudia
   Alvarez
   Carreño

6. Datos del sinodal 4
   M. en C.
   Loiret Alejandría
   Dosal
   Trujillo

7. Datos del trabajo escrito
   Identificación de caminos alostéricos en proteínas empleando simulaciones de dinámica molecular, teoría de gráficas y detección de similitudes locales.
   94 p
   2019

A mi padre, familia y amigos.

Agradecimientos

A la Universidad Nacional Autónoma De México, a los profesores que lograron implantar su semilla de conocimiento y sabiduría a lo largo de la carrera, así como a mis compañeros con los que pase buenos y malos ratos con tal de obtener este título.

A mi Madre Martha, mis hermanas Tania y Sarita, mi Tía Maly y Lupita que siempre estuvieron ahí cuando lo necesité, apoyándome en todo momento.

A mis colegas y amigos Horz, Roldi, Moiky, Kassy, Jaurez, Cess, Sebas, Pedrito, Eugenio, etc. Que me acompañaron a lo largo de la carrera y la tesis con vivencias, distracciones y apoyo para culminar este proyecto.

Al Dr. Marcelino por su paciencia, esfuerzo y dedicación y a sus integrantes del Laboratorio por hacer ameno el trabajo.

A los sinodales Dr. Raúl Espejel, Dr. César Poot, Dra. Claudia Alvarez y M. en C. Alejandría Dosal que con sus valiosos comentarios y correcciones se mejoró este proyecto.

Al mejor grupo de trabajo JC, Diego, Jaqui, Ana, Betito, Cris, Natham, Alex, Fede y Daniel, gracias por la paciencia y experiencias, seguiremos cosechando éxitos.

Agradezco el apoyo del Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica PAPIIT IA202917 y LANCAD-UNAMDTIC-320.

Resumen

En este trabajo se realizó la búsqueda de los caminos alostéricos generando un algoritmo de detección de similitudes locales y analizando redes en proteínas con una alta similitud estructural, construyendo la red de contacto residuo a residuo (RCRR) y una red de correlación dinámica (RCD).

Se desarrollo un método para la detección de similitudes locales entre proteínas y/o ensambles de proteínas, generando un alineamiento por medio de la red de contacto residuo a residuo de ambas proteínas. Esto se consiguió agrupando por medio de cliques definidas en la teoría de redes y con el uso de dinámicas moleculares se busca el cambio estructural significativo que produce la función característica de la proteína o ensamble de proteínas. Generando una versión alterna al algoritmo de comparación estructural de biomoléculas independiente de su topología conocido como Click [1].

Utilizando las coordenadas cartesianas de las moléculas, su estructura secundaria y mediante la agrupación de residuos con estructura similar por medio de cliques, se busca similitudes locales entre proteínas. Por medio de un ajuste de mínimos cuadrados se sobreponen ambas estructuras, buscando la alineación optima entre residuos y obteniendo relaciones independientes de la topología.

Generando las redes de contacto y alostéricas, se identifican comunidades aplicando el método de Louvain, se analiza la topología de la red y las características de los nodos por medio de la centralidad de intermediación o *betweenness*.

Se generó el software de alineamiento con resultados similares a los reproducidos por Click y se plantea la metodología para la obtención de caminos alostéricos para cualquier proteína por medio de las redes de correlación dinámica y las redes de contacto residuo a residuo.

Índice general

Lista de figuras

Lista de tablas

Capítulo 1 Introducción

1.1 Planteamiento del problema

El estudio de la estructura tridimensional y la dinámica del sistema molecular por medio del seguimiento de los átomos que conforman a la proteína[1] está ligada al entendimiento de las funciones de la proteína [2, 3]. Cada proteína tiene acceso a un conjunto finito de conformaciones, por ejemplo, el espacio de estados sobre el cual es posible establecer la relación entre la estructura y la función biológica. Las transiciones conformacionales son particulares de cada proteína y varían en magnitud, tiempo de transición y diversidad. De esta situación deviene la alta complejidad intrínseca en el estudio de las propiedades dinámicas de estos sistemas biológicos, tanto de perspectivas experimentales como teóricas.

Las simulaciones de dinámica molecular[2] son una de las herramientas con mayor uso para la evaluación de las propiedades de transición de las macromoléculas biológicas [4]. El muestreo del espacio a estados accesibles del sistema se realiza a través de resolver las ecuaciones de movimiento de Newton. Para ello se emplean potenciales clásicos de interacción interatómica e integradores simplécticos de las ecuaciones diferenciales. Si en una simulación se conservaran las variables termodinámicas apropiadas y se simularan la dinámica molecular durante un periodo de tiempo infinito, se cumpliría la hipótesis ergódica[3], que corresponde a muestrear en su totalidad el espacio fase. Dado que el determinar totalmente el espacio fase[4] no es factible en términos prácticos, incluso si dicho espacio fuese conocido en su totalidad, la complejidad en el análisis de las transiciones entre estados funcionales persiste. Esto debido a que el gran número de grados de libertad del sistema hace necesario su estudio a través de la selección de un número reducido variables representativas.

[1] Biomoléculas orgánicas formadas por varios aminoácidos, con diversas funciones según su estructura. Pag 17.
[2] Técnica de simulación que consiste en el uso de la mecánica clásica para modelar la estructura y dinámica de moléculas. Pag 23.
[3] El promedio de las propiedades moleculares individuales coincide con las propiedades macroscópicas del sistema molecular. Pag 24.
[4] Representación del conjunto de posiciones y momentos del sistema de partículas.

El uso de teoría de gráficas para el análisis de las propiedades estructurales y dinámicas de las proteínas ha ganado popularidad en últimos años [5]. El alineamiento de estructura tridimensional entre dos proteínas se establece, comúnmente, empleando como guía a la similitud de secuencia primaria entre éstas. Sin embargo, dicha guía limita el alineamiento a seguir patrones secuenciales [1]. El primer análisis de esta tesis fue implementar un algoritmo análogo al desarrollado por Nygen *et al*. [1], que permite realizar alineamientos sin tal dependencia en la secuencia residual. Es deseable contar con dicho algoritmo, debido a que muchos de los análisis estructurales de proteínas tienen como base el alineamiento a una estructura de referencia. Por ejemplo, los resultados de análisis de cambios conformacionales, en una familia de proteínas o en ensambles de proteínas obtenidas de simulaciones de dinámica molecular, pueden variar significativamente dependiendo de la calidad del alineamiento, utilizando el software de alineamiento con el caso de estudio en la segunda parte. En la segunda parte de la tesis se buscó establecer un algoritmo que permita la identificación de rutas alostéricas[5] empleando teoría de redes [6].

Alostería es el nombre dado al fenómeno, donde el sitio de actividad biológica se ve afectado por los estados que adquieren regiones distantes a dicho sitio [7]. Actualmente no existe una metodología estándar que permita identificar las regiones esenciales que definen los comportamientos característicos de la función biológica, ni detectar rutas alostéricas.

Se reporta que la biosíntesis de proteínas que forman a la pirimidina presenta mutaciones en sitios alostéricos[6] y activan o inhiben la función en ciertas moléculas [7]. Este fenómeno es de interés pues es un elemento fundamental en procesos como la modulación de la reproducción de bacterias. En este trabajo se generó una metodología para identificar caminos alostéricos y la detección de los residuos que pueden modular la reproducción de bacterias, como en el caso de estudio.

[5] Cadena de residuos que relaciona el sitio activo con el sitio alostérico. Pag 45.
[6] Ubicación de la unión del sustrato y de la proteína donde modifica la conformación de la proteína y que se encuentra alejada del sitio activo.

1.2 Objetivos

1.2.1 General

Generar la metodología para la obtención de caminos alostéricos, desarrollando un algoritmo para comparar proteínas y encontrar similitudes locales. Identificar la formación de comunidades de residuos en la estructura tridimensional de las proteínas que expliquen particularidades de su dinámica y funcionamiento.

1.2.2 Específicos

- Implementar algoritmos que empleen la información estructural de residuos, generada a través de la simulación de dinámica molecular, para el establecimiento de la red.

- Implementar algoritmos para la identificación de similitudes locales entre residuos.

- Evaluar la efectividad de la metodología en la identificación de similitudes locales en proteínas con modelos para los cuales la respuesta es conocida.

- Emplear simulaciones de dinámica molecular para la generación de ensambles de la estructura de proteínas

- Emplear la herramienta en el estudio de casos novedosos como el de la proteína PyrR.

- Generar y analizar las redes de correlación dinámica[7] de las proteínas de interés con el fin de encontrar caminos alostéricos.

1.3 Metodología

Para encontrar caminos alostéricos se plantean 2 análisis:

[7] Red que describe el movimiento correlacionado entre residuos en una proteína. Pag 43.

- Detección de similitudes locales
- Análisis de redes de correlación dinámica

Como se muestra en la siguiente figura.

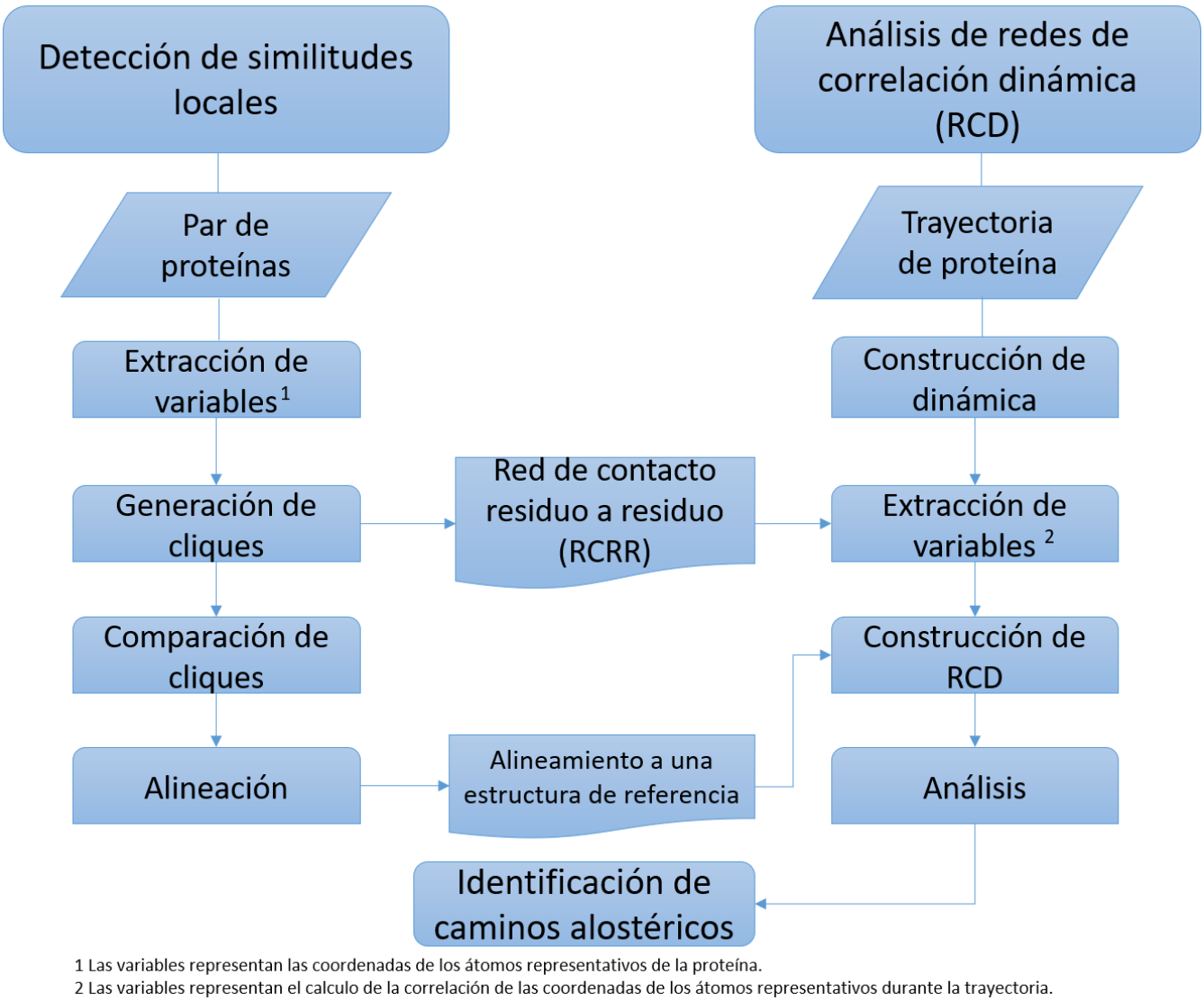


Figura 1: Diagrama de flujo del proceso de la metodología.

1.3.1 Detección de similitudes locales

Extrayendo la información de proteínas conocidas o de ensambles de la estructura de proteínas, en formato PDB[8], se implementa el algoritmo que se divide en 4 fases [1]:

1. Extracción de variables: Se toman las coordenadas cartesianas de los átomos representativos (C_α[9]) de los aminoácidos.
2. Generación de cliques[10]: Se agrupan por medio de cliques los residuos que caen en el umbral de distancia establecido. Se obtiene la estructura secundaria por medio del algoritmo de DSSP y se le asigna a cada aminoácido la estructura secundaria correspondiente [8].
3. Comparación de cliques: Se computa uno a uno la distancia entre los cliques de residuos de las estructuras A y B, dejando solamente los que cumplen con el umbral de distancia, filtrando los diversos prospectos y calificando la similitud en estructura secundaria [11]de los cliques.
4. Alineación[12]: La comparación de cliques ayuda a identificar la estructura equivalente en los residuos de las 2 estructuras; utilizando estas equivalencias se ajusta por mínimos cuadrados y se sobreponen ambas estructuras. Se devuelve un *score* de acoplamiento y se mide la efectividad del modelo.

1.3.2 Redes de correlación dinámica (RCD)

Para la generación de redes de correlación dinámica se construye la red de contacto entre residuos [13]con los pasos 1 y 2 de la metodología de detección de similitudes locales y una red de correlación durante la dinámica molecular con los siguientes pasos:

1. Construcción de dinámica: Simulación de dinámica con la proteína de interés.

[8] Formato universal donde depositan información sobre la estructura de la proteína y el método de como obtuvieron dicha estructura. Pag 19.
[9] Átomo representativo en aminoácidos indicando el primer carbón en el aminoácido.
[10] Red donde todos los elementos están enlazados entre ellos.
[11] Segundo nivel organizacional en las proteínas. Pag 20.
[12] El alineamiento consiste en rotar y trasladar una proteína a otra de manera que la distancia entre los átomos sea mínima.
[13] Construcción de red en la Pag 38.

2. Extracción de variables: Se calcula la correlación de los residuos por medio del movimiento de los átomos representativos.
3. Construcción de RCD: Generación de la RCD donde se hace una multiplicación de las matrices de adyacencia entre la red de contacto y la red de correlación.
4. Análisis: Descripción de la topología de la red, aplicación del algoritmo de detección de comunidades y análisis de las métricas de centralidad.

1.4 Resultados a obtener

- Desarrollo del algoritmo para encontrar similitudes locales en diversas proteínas y/o conjunto de proteínas.
- Evaluación positiva de la efectividad del método en proteínas conocidas.
- Detectar regiones en proteínas donde los dominios o subdominios presentan alostería.
- Encontrar comunidades y dinámicas que expliquen el funcionamiento de diversas proteínas.
- Obtener caminos alostéricos por medio de la RCD.

1.5 Contenido de la tesis.

En los subsecuentes capítulos se abordan los siguientes temas:

1. Una breve introducción de la relevancia del problema, donde se describe la importancia del estudio de los arreglos estructurales en las proteínas y la importancia en la identificación de caminos alostéricos. El uso de las teorías de dinámica molecular y teoría de redes para la evaluación de propiedades macroscópicas a escalas moleculares y la simplificación de las interacciones dentro de la dinámica.

2. Bases teóricas del estudio de estructura y función molecular, dinámica molecular y teoría de redes.

3. La metodología para generar el software que ayude a la comparación de proteínas y la identificación de similitudes locales, utilizando diversas técnicas de teoría de redes y dinámica molecular y la construcción de redes de correlación dinámica.

4. La prueba y análisis de resultados obtenidos de dicho software. La discusión y relevancia del software generado, así como un análisis más amplio de los resultados enfocándose en el caso de estudios de la pirimidina en conformación de dímero y tetrámero con propiedades de particular interés.

5. Conclusiones y trabajo a futuro.

Capítulo 2
Antecedentes

2.1 Introducción al modelado y estructura molecular

Para el estudio de la estructura, función y modelado molecular ha ayudado la interdisciplinaridad de áreas como la biología, química, física y computación por la diversidad de temas y enfoques que se le puede dar al tema. Este tipo de áreas de estudio conjugadas motiva la creación de ramas como la biología computacional, química computacional, la biofísica computacional, biofísica y química teóricas entre muchas más.

El modelado molecular es la técnica computación que por medio de simulaciones busca recrear el comportamiento de moléculas permitiendo conectar la teoría y la experimentación por medio de modelos computacionales, estos modelos presentan mayor robustez dado el avance tecnológico en procesamiento y almacenamiento de datos, al generar simulaciones más reales como se observa en la naturaleza, dichas simulaciones ayudan a generar nuevas teorías y experimentos biológicos.

Los modelos moleculares proveen una manera sistemática de investigar la estructura, dinámica y propiedades termodinámicas del sistema molecular, probando y desarrollando hipótesis.

Se implementan modelos de la física y química, el cálculo de energía y otras propiedades moleculares derivadas a partir de modelos mecánicos dentro de un marco de física clásica. Este modelado molecular representa a la molécula como un sistema mecánico donde los átomos son partículas y las interacciones de los enlaces están conectados por resortes. Se observa que las moléculas rotan, vibran y se trasladan adecuándose al espacio confinado, respondiendo e interactuando por medio de fuerzas intermoleculares e intramoleculares [9, 10]. En la presente tesis se centró en los métodos de dinámica molecular para el modelado molecular.

Dando comienzo a la descripción de las proteínas, se estudia su estructura y función, la unión de aminoácidos se define como enlaces peptídicos y las moléculas resultantes se llaman péptidos, un polipéptido es un péptido suficientemente grande que contiene al menos 50 o más aminoácidos con una estructura tridimensional única y estable [11].

Nombre	Número de aminoácidos
Péptido	De 10 a 50
Polipéptido	De 50 a 100
Proteína	Más de 100

Tabla 1: Definición de conjunto de aminoácidos.

Los bloques principales para los polipéptidos son moléculas orgánicas conocidas como aminoácidos, cada proteína se diferencia por medio de una única secuencia de aminoácidos o residuos. Los aminoácidos son moléculas orgánicas con un grupo amino ($-NH_2$) y un grupo carboxilo ($-COOH$)

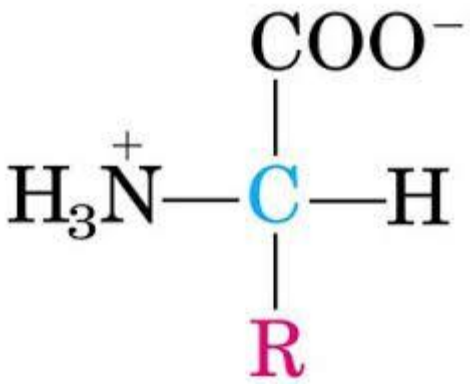


Figura 2: Estructura de un aminoácido que se diferencian por la cadena lateral R.

Las características y funciones de cada aminoácido dependen de la cadena lateral. Los aminoácidos se dividen en 4 grupos principales dadas las características de la cadena lateral [11]:

- Grupo básico compuesto por la arginina, lisina, histidina llamados así porque son receptores de protones.

- Grupo ácido integrado por el ácido aspártico y el ácido glutámico, la cadena lateral y el carboxilo alfa, son propensos a donar un catión y presentan carga negativa.

- Grupo polarizado pero descargado, los aminoácidos con grupos polares como la glutamina y la asparagina, la serina, treonina y la tirosina presentan una cadena lateral con grupos hidroxilos.

- Grupo no polar como la glicina que consiste en un solo átomo de hidrogeno en la cadena lateral, la alanina, isoleucina, leucina y valina que tienen hidrocarburos, la fenilalanina, triptófano son cadenas aromáticas y la metionina y prolina son cadenas con variables únicas.

La representación de los aminoácidos y sus abreviaturas se presentan en la siguiente tabla.

Aminoácido	**Abreviación de 3 letras**	**Abreviación 1 letra**
Alanina	Ala	A
Arginina	Arg	R
Asparagina	Asn	N
Ácido Aspártico	Asp	D
Cisteína	Cys	C
Glutamina	Gln	Q
Ácido Glutámico	Glu	E
Glicina	Gly	G
Histidina	His	H
Isoleucina	Ile	I
Leucina	Leu	L
Lisina	Lys	K
Metionina	Met	M
Fenilalanina	Phe	F
Prolina	Pro	P
Serina	Ser	S
Treonina	Thr	T
Triptófano	Trp	W
Tirosina	Tyr	Y
Valina	Val	V

Tabla 2: Representación y abreviaturas de los aminoácidos.

Para obtener información sobre la estructura de las proteínas se emplean diversas técnicas experimentales como la cristalografía por rayos X, los estudios de resonancia magnética nuclear (NMR, siglas en inglés) y la microscopia tridimensional electrónica (3DEM, siglas en inglés) obteniendo la estructura tridimensional de proteínas a escalas atómicas [12].

Utilizando estas técnicas se escriben los resultados en un formato universal conocido como Protein Data Bank (PDB) [13]. Estos archivos son un repositorio de coordenadas atómicas y otra información que describe las técnicas utilizadas y a la proteína [14].

Estudiando la estructura de las proteínas se distinguen 4 niveles de organización. El primer nivel de organización se le conoce como estructura primaria y es la secuencia de aminoácidos que componen a la proteína, este nivel se mantiene por los enlaces peptídicos, formando el esqueleto de la proteína de donde emergen las cadenas laterales [15].

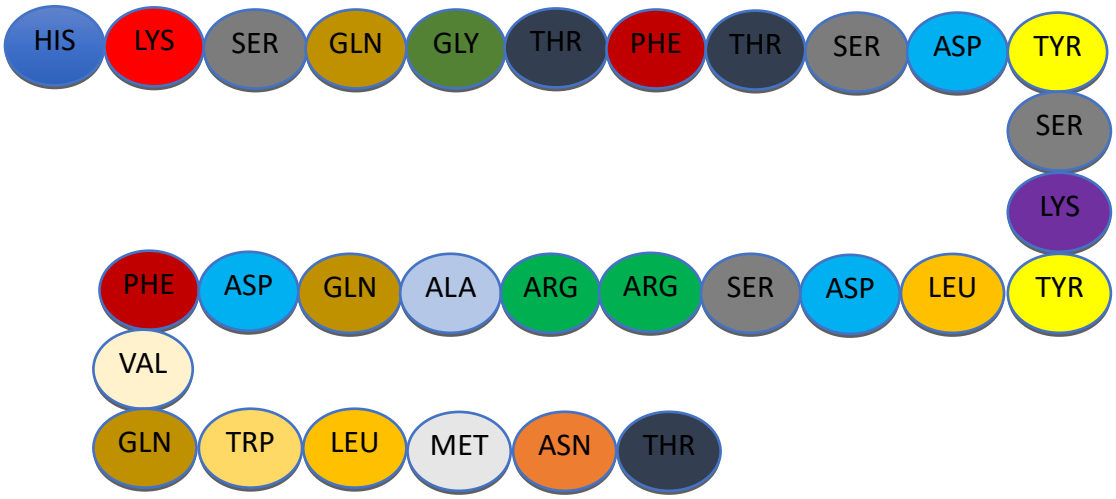


Figura 3: Estructura primaria, secuencia de aminoácidos.

Los siguientes 3 niveles de organización se establecen debido a interacciones débiles no covalentes como los enlaces de hidrógeno, los enlaces iónicos, la interacción hidrofóbica y las interacciones de van der Waals. Al ser interacciones débiles permite que se formen y rompan de una manera sencilla, permitiendo a la proteína tener diferentes conformaciones y por ende diversas funciones.

El segundo nivel de organización que corresponde a la estructura secundaria se estabiliza por medio de los puentes de hidrógeno y describe los patrones del plegamiento dentro del segmento de la cadena polipeptídica que contiene a los residuos formando α-hélices, β-laminas o *C* que corresponde a un plegamiento azaroso o *coil* en inglés.

Se observa en la naturaleza que estos patrones estructurales se definen por medio de los ángulos de torsión o diedros [16].

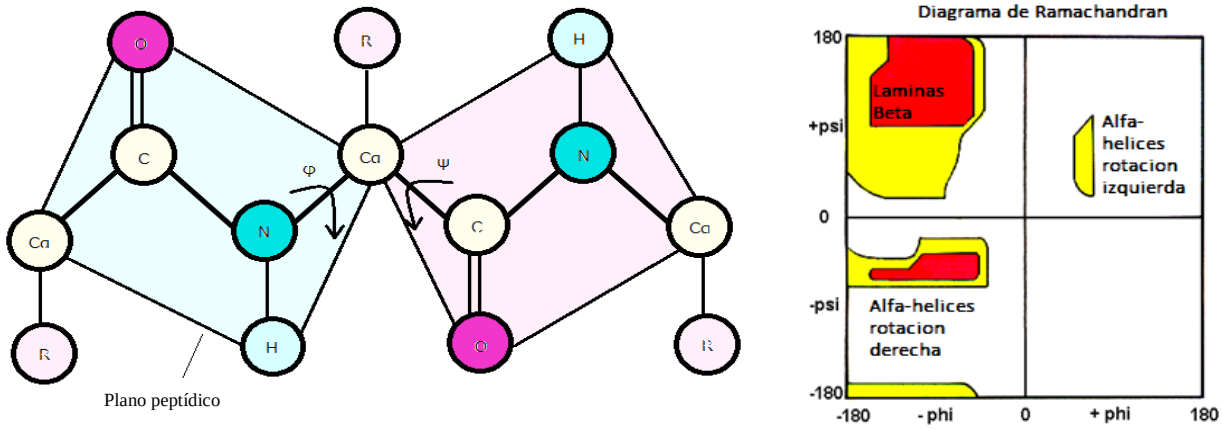


Figura 4: Representación de los ángulos diedros Phi y Psi y Diagrama de Ramachandran, este diagrama mapea la estructura secundaria de proteínas conocidas y su relación con el valor de los ángulos phi y psi.

El ángulo de torsión se puede calcular como el ángulo entre dos planos de los dos enlaces peptídicos contiguos. Para obtener los patrones se calcula con base a dos ángulos phi y psi. El ángulo ψ es la rotación en torno al enlace $C - C_\alpha$, del plano *C*, *N*, C_α y *C*, el ángulo φ es la rotación en torno al enlace $C_\alpha - N$ del plano generado por *N*, C_α, *C* y *N*.

Al momento de representar estos ángulos se obtiene el diagrama de Ramachandran, en la que se distinguen tres regiones donde se identifican estos patrones como $\alpha_{helices}$ y $\beta_{laminas}$.

Las $\alpha_{helices}$ se generan por medio de los enlaces de hidrógeno que se localizan dentro de la hélice, generando un patrón regular. El átomo de oxígeno en el grupo carboxilo del *n* residuo forma un enlace de hidrógeno con el átomo de hidrógeno del grupo N – H del $n + 4$ residuo. Las $\beta_{laminas}$ se alinean una al lado de la otra para que los grupos C = O y N – H en las cadenas adyacentes interactúen a través de enlaces de hidrógeno para producir una estructura casi plana. Las láminas pueden estar organizadas de una forma paralela si, la cadena polipeptídica tiene la misma dirección del amino carboxilo o antiparalela si, están orientados en direcciones opuestas como se muestra en la figura 5.

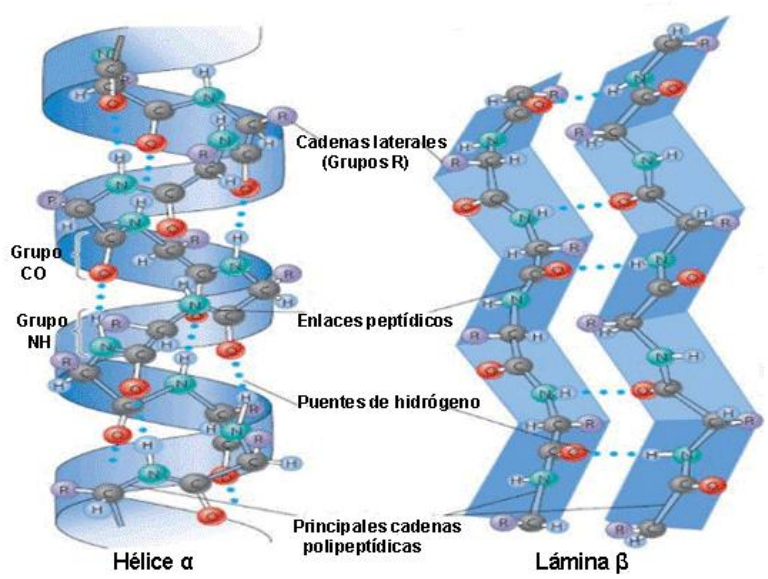


Figura 5: Representación gráfica de $\alpha_{helices}$ y $\beta_{laminas}$ [11].

El tercer nivel de organización provee una visión tridimensional de la proteína y se origina por el plegamiento de la estructura secundaria, la conformación particular de cada proteína caracteriza su función. Existen 2 clases de estructuras terciaras, fibrosas y globulares, las proteínas fibrosas se caracterizan por una estructura más alargada y estructurada como la queratina y el colágeno, se diferencia de las proteínas globulares por tener una forma más esférica. La mayoría de las proteínas tienen este tipo de estructura.

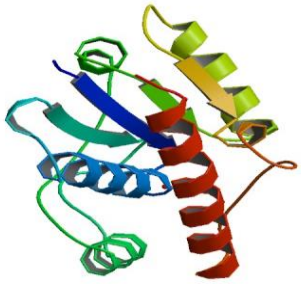


Figura 6: Estructura terciaria de la Phosphotyrosine protein phophatase (1phr), se colorea la estructura secundaria para diferenciarla. [14].

El cuarto nivel de organización solo se presenta si existe más de una cadena polipeptídica en proteínas, la formación de esta estructura es mediante la unión de enlaces débiles de varias cadenas polipeptídicas. Por ejemplo, en la figura 7 se observan las cadenas de la proteína PyrR en forma de tetrámero.

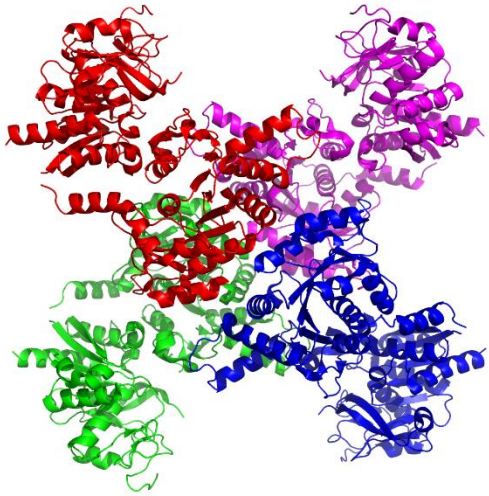


Figura 7:Estructura cuaternaria de la proteína E. coli CTP síntesis en forma de tetrámero. Las cadenas polipeptídicas se colorearon para diferenciarlas.

2.2 Dinámica Molecular

La dinámica molecular es una técnica que nos permite numéricamente estudiar los cambios conformacionales de un modelo molecular mediante una simulación [3]. Esta técnica se ha visto beneficiada por el avance tecnológico de nuestra época al generar ordenadores con mayor poder de cómputo y la capacidad de almacenar más información logrando simular la dinámica de miles de átomos en pocos segundos [9].

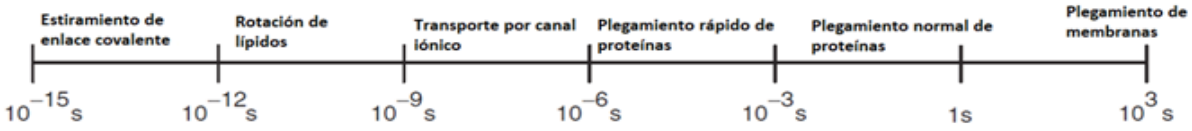


Figura 8: Tiempo que tarda en generar una simulación computacional en cada fenómeno biológico.

Esta técnica nos detalla el movimiento individual de cada átomo y puede ser seguido para entender la estructura y dinámica de la molécula. El campo de fuerza es el concepto central del modelado molecular. Este es una función potencial que permite asignar un valor energético a la conformación del sistema de átomos. El campo de fuerza está formado por una serie de potenciales clásicos [17]:

$$U = \sum_{bandas} \frac{1}{2} k_l (b - b_0)^2 + \sum_{angulos} \frac{1}{2} k_b (\theta - \theta_0)^2 + \sum_{diedro} \sum_{n} k_{\tau,n} (1 - \cos(n\tau + \delta_n)) + \sum_{i} \sum_{j<i} \left(\frac{A_{ij}}{r_{ij}^{12}} + \frac{B_{ij}}{r_{ij}^{6}} + \frac{q_i q_j}{r_{ij}} \right) \quad (2.2.1)$$

Donde el primer término corresponde a la vibración en la longitud de enlaces covalentes si suponemos que dicha interacción corresponde a los átomos ligados por un resorte. El siguiente termino se refiere a las variaciones en el ángulo formado por los enlaces entre 3 átomos. De igual manera al termino anterior, se describen por medio de la ley de Hooke. El tercer término representa la torsión de los ángulos diedros y representa la torción ejercida a través de un enlace en particular. Finalmente, el ultimo termino describe el potencial de Lennard-Jones que representa la atracción y repulsión a distancias interatómicas, junto con las fuerzas electrostáticas entre las cargas elementales de las moléculas modeladas.

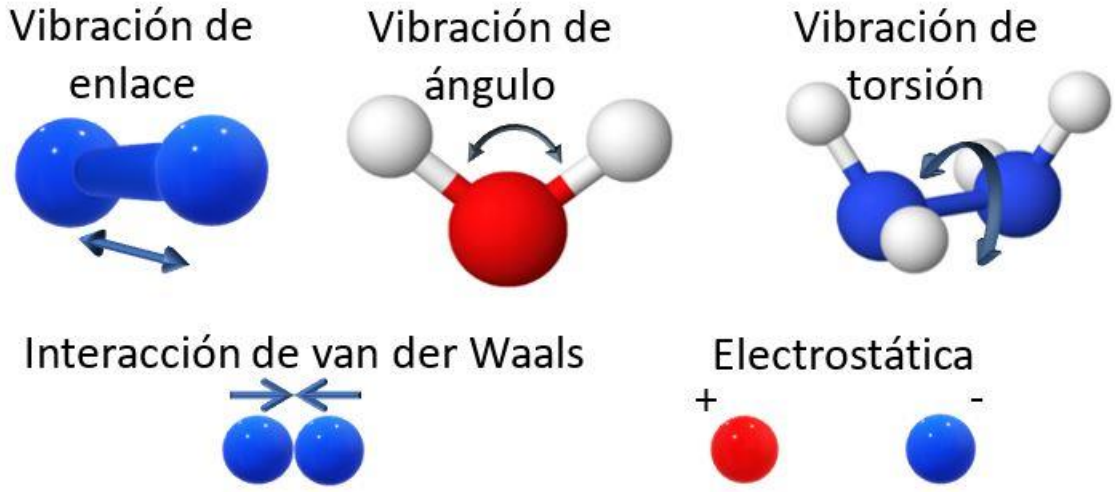


Figura 9: Campos de fuerza que interaccionan en la dinámica molecular.

Todas las posibles vibraciones de la molécula pueden ser descritas como una superposición de oscilaciones fundamentales o modos normales. Cada molécula de N átomos tiene $3N - 6$ modos normales que consisten en 3 grados de libertad por átomo menos 3 grados traslacionales y 3 grado rotacionales que describen a la molécula. Los modos de flexión del enlace incluyen 2 tipos de deformaciones en el plano, uno de tijereo y otro de balanceo y a su vez existen otras 2 deformaciones fuera del plano conocidas como meneo y torsión [9].

Dados los potenciales y fuerzas de todos los átomos, durante la dinámica, las coordenadas son actualizadas para nuevamente calcular los potenciales en el siguiente paso. Estos cálculos de dinámica molecular se ejecutan integrando las ecuaciones de Newton [10].

$$F_i = -\frac{\partial V(r_1, \ldots, r_N)}{\partial r_i} \qquad (2.2.2)$$

$$m_i \frac{\partial^2 r_i}{\partial t^2} = F_i \qquad (2.2.3)$$

Comúnmente este tipo de dinámica y de cálculos simulan alrededor de 1 a 2 femtosegundos $[10^{-15}\ s]$. Por otro lado, estas simulaciones utilizan condiciones en la frontera periódicas, es decir, si una molécula sale por la frontera izquierda, reaparece en la frontera derecha. Para que no interactúe consigo misma se genera una caja lo suficientemente grande.

La minimización de la energía se utiliza para brindar estabilidad numérica a la simulación. Al minimizar la estructura experimental con respecto a un campo de fuerza dado se reduce el riesgo de generar desbalances en los cálculos numéricos. También ayuda a refinar la estructura con baja resolución y es necesario estudiar los puntos donde la energía se minimice, ya que corresponden a los estados estables del sistema.

Las propiedades macroscópicas medidas no se obtienen por observación directa, sin embargo, el promedio de las propiedades moleculares individuales de miles de moléculas coincide con las propiedades macroscópicas, cumpliendo así la hipótesis de ergodicidad [3]. Gracias a esta hipótesis, se obtienen implicaciones importantes:

- No es suficiente trabajar con estructuras individuales, pero el sistema se extiende, generado ensambles estructurales representativos a ciertas condiciones experimentales dadas.
- Para el cálculo de las propiedades en el equilibro, se examina el ensamble de estructuras, sin necesidad de reproducir las trayectorias individuales de cada átomo.
- Las propiedades en el equilibrio termodinámico relacionadas a la energía libre no pueden ser calculadas por simulaciones individuales, pero existen técnicas para sus cálculos.

2.3 Teoría de redes

El estudio de las redes tiene diversas aplicaciones como vemos en redes computacionales, sociales o biológicas, siendo un área interdisciplinaria que combina teorías de matemáticas, física, química, biología, entre otras más [6]. Al poder describir fenómenos de la naturaleza con la teoría de redes es posible simplificar sistemas complejos, describiendo las interacciones fundamentales o con mayor impacto en el sistema en algunos casos. Para la presente tesis se utilizó la teoría de redes para encontrar las interacciones fundamentales dentro de la proteína y observar al conjunto de residuos que son similares entre ellos dada su distancia y calculando sus métricas de centralidad.

Una red o grafo *G* está compuesta por un par de conjuntos $G = (V, E)$, donde *V* es el conjunto de vértices también conocidos como nodos y *E* el conjunto de enlaces también llamadas aristas.

Existen redes dirigidas y no dirigidas. En una red dirigida los enlaces son pares ordenados de elementos de *V*, es decir, cada enlace tiene una dirección y en una red no dirigida los enlaces son pares no ordenados de elementos de *V* y se representa en la siguiente figura. [6]

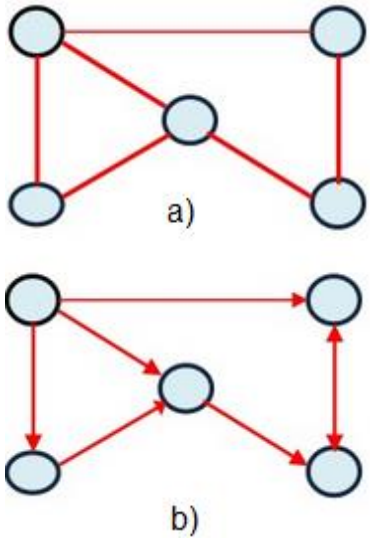
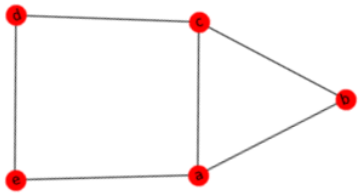


Figura 10: Representación de a) grafo no dirigido y b) grafo dirigido.

El conjunto de definiciones para el estudio de la teoría de redes es:

- Una red es un conjunto de componentes con una relación. Representada por nodos y enlaces.
- Un componente es un elemento de la red.

- El orden es la cantidad de nodos en la red y se denota por *n*.
- El tamaño es la cantidad de enlaces en la red y se denota por *m*.
- El camino es una sucesión alternada de nodos y enlaces.
- Una ruta o trayectoria es el camino que no repite nodos.
- La matriz de adyacencias A_{ij} es una matriz que representa el número de enlaces entre cada nodo, si el grafo es no dirigido la matriz es simétrica.



	a	b	c	d	e
a	0.0	1.0	1.0	0.0	1.0
b	1.0	0.0	1.0	0.0	0.0
c	1.0	1.0	0.0	1.0	0.0
d	0.0	0.0	1.0	0.0	1.0
e	1.0	0.0	0.0	1.0	0.0

Figura 11: Ejemplo de red y su correspondiente matriz de adyacencias.

- Una red es pesada si cada enlace tiene un peso w_{ij} y las redes no pesadas tienen peso de 1. Los pesos se almacenan en la matriz de pesos W_{ij}.

- Dos nodos son vecinos si existe un enlace entre ellos.

- El grado de un nodo es el número de enlaces que tiene. Se denota el número del grado del nodo *i* como k_i, para un grafo no dirigido se define en términos de la matriz de adyacencias.

$$k_i = \sum_{j=1}^{n} A_{ij} \qquad (2.3.1)$$

- El grado promedio de la red es la suma de todos los grados de los nodos de la red dividido entre el orden de la red, denotado por $\langle k \rangle$.

- El número máximo de enlaces que puede haber en una red es $\frac{n(n-1)}{2}$ [6].

- Un clique o clan es el subconjunto de nodos, tal que, todos los nodos son adyacentes entre ellos, es decir, existe un enlace que los conecta. Llamaremos n-clique al grafo o subgrafo[14] no dirigido que se obtiene de los nodos del clique.
- Un componente es fuertemente conexo si para cada par de nodos en el grafo o subgrafo existe un camino que los une.
- Una *ij*-geodésica es un camino de longitud mínima entre dos nodos, es decir, la distancia entre *i* y *j* es mínima y se denota por g_{ij}.
- La distancia d_{ij} entre dos nodos (i, j) se define como el número de enlaces del camino más corto que los conecta.
- El diámetro de la red se define como la mayor distancia entre todo par de nodos perteneciente a la red.
- La longitud promedio o trayectoria característica, *L*, se define como la media de las distancias entre todos los pares de nodos, es decir, la separación típica entre pares de nodos.

$$L = \frac{1}{n(n-1)} \sum_{i \neq j} d_{ij} \qquad (2.3.2)$$

- El coeficiente de *clustering*, que toma valores $[0\,,1]$, relaciona la probabilidad de tener un enlace con otro nodo si este está conectado con un vecino. Su cuantificación depende de la red y varía según los autores. Supongamos que un nodo *i* de la red tiene *s* vecinos. A lo más pueden existir $\frac{n(n-1)}{2}$ enlaces entre ellos [6]. El coeficiente de *clustering*, C_i, del nodo *i* se define como la proporción entre, E_i, el número de enlaces que de verdad existen entre los vecinos de i, y la máxima cantidad posible.

$$C_i = \frac{E_i}{\left(\frac{n(n-1)}{2}\right)} \qquad (2.3.3)$$

[14] Una subgráfica H es subgráfica de G si los nodos y enlaces de H son un subconjunto de los nodos y enlaces de G.

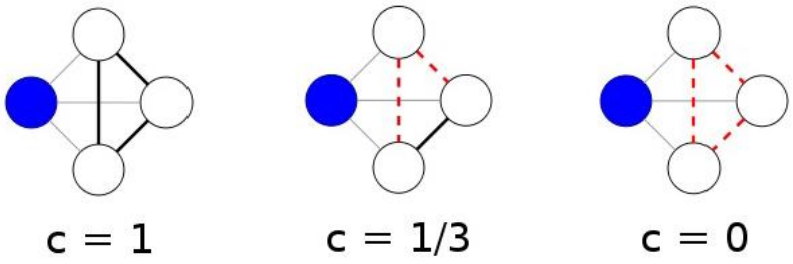


Figura 12: Ejemplo del coeficiente de clustering [33].

- La centralidad de intermediación (*betweenness centrality*) se define como el porcentaje de caminos cortos que atraviesan al nodo:

$$x_i = \sum_{js} \frac{p_{js}^i}{q_{js}} \qquad (2.3.4)$$

  Donde p_{js}^i denota el número de caminos geodésicos que van de *j* a *s* a través de *i*, y q_{js} es el número total de caminos geodésicos que van de *j* a *s*.

- Una característica importante de las redes es la distribución de su grado, el cual viene dada por la función $P(k)$ de probabilidad e indica que un nodo seleccionado al azar tenga *k* enlaces.

Con el fin de describir a la naturaleza y sus interacciones, la teoría de redes fue desarrollando modelos para la generación de redes. La diferenciación entre tipo de redes se da por medio de la distribución del grado y de la topología de la red, como las siguiente:

2.3.1 Red Completa

Una red es completa si el grado para todo nodo es $k_i = N - 1$ con *N* el número de nodos. Se observa en estas redes las siguientes propiedades, presentan una longitud de camino proporcional al número de nodos de la red, de manera que $L = \frac{n}{2m}$, donde *n* es el número de

nodos y *m* el número de enlaces, el coeficiente de clustering se define como $C = \frac{3(k-1)}{4(k-2)}$. En la siguiente figura se observan distintos ejemplos de graficas completas.

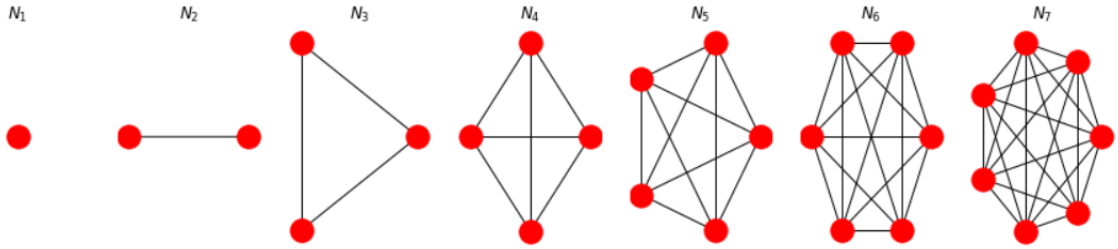


Figura 13: Graficas N-completo.

Un grafo N-completo es un n-clique. Un clique maximal es aquel que no está contenido en un n-clique más grande. Cada n-clique es un subgrafo de un clique maximal por un grado menor. Por ejemplo, en un 4-clique, se encuentran subgrafos que corresponden a un 3-clique y a 2-clique.

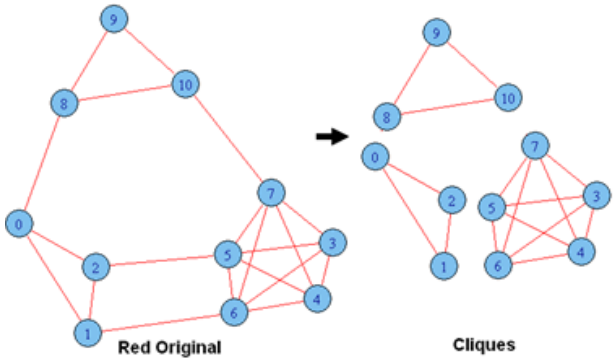


Figura 14: Ejemplo de cliques desde una red, formando dos grupos de 3-cliques y un grupo de 5-clique.

2.3.2 Red aleatoria

Una red aleatoria o el modelo de Erdös-Renyi es aquella donde los enlaces se generan de forma aleatoria. Los investigadores Paul Erdös y Alfred Renyi se dieron a la tarea de modelar las redes aleatorias, y descubrieron que en estas redes la distribución del grado se describe por una distribución de Poisson.

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k} \qquad (2.3.5)$$

En esta investigación notaron la formación de subredes y consisten en una agrupación de nodos y enlaces que pertenecen a la red completa. Otra propiedad en las redes aleatorias es la longitud de camino que se escala con el número de nodos y es proporcional al logaritmo del número de nodos e inversamente proporcional al logaritmo del grado promedio como la siguiente ecuación.

$$L_{aleatoria} = ln\,(n)/ln(\langle k \rangle) \qquad (2.3.6)$$

Con $\langle k \rangle$ el grado promedio de la red, este valor suele ser pequeño. El coeficiente de *clustering* está dado por $C_{aleatoria} = \frac{\langle k \rangle}{n}$. A continuación, se muestra una red aleatoria y la gráfica de su distribución de grado.

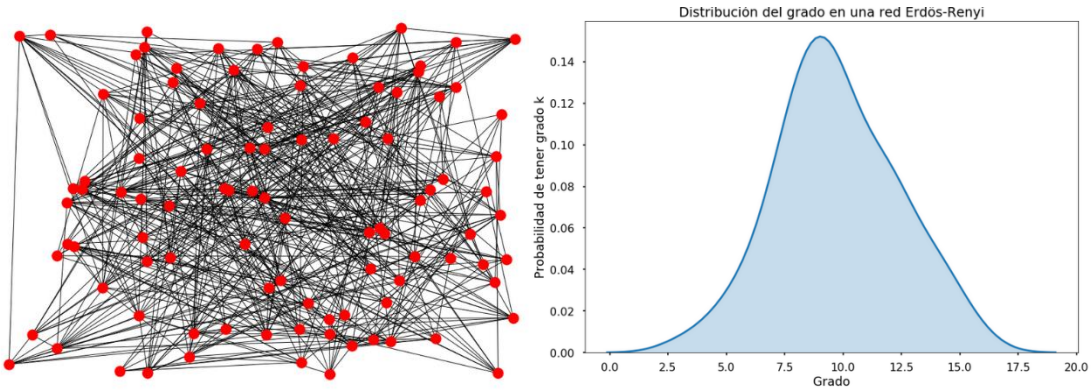


Figura 15: Representación de una red aleatoria y la distribución del grado.

Al estudiar las redes aleatorias, se dieron cuenta que las redes generadas por la naturaleza no se representaban por una red aleatoria, generando nuevos modelos.

2.3.3 Red de mundo pequeño

Una red de mundo pequeño es un tipo de grafo para el que la mayoría de los nodos no son vecinos entre sí, sin embargo, la mayoría de los nodos pueden ser alcanzados desde cualquier nodo origen a través de un número relativamente corto de saltos entre ellos, es decir, teniendo un promedio de longitud de camino corto y un valor de *clustering* alto.

Este modelo fue propuesto por los investigadores Watts y Strogatz, partiendo de la generación de un grafo aleatorio que reordena los enlaces con probabilidad *p* y dio cimientos a la teoría de los 6 grados de separación que indica, cuales quieras 2 personas en cualquier lugar

del mundo, probablemente, existen en promedio 6 personas que se conocen y mediante puedan establecer una cadena que conecte una persona con la otra [6].

En la siguiente figura se observa la formación de este tipo de redes partiendo desde cualquiera de ellas [15, 16].

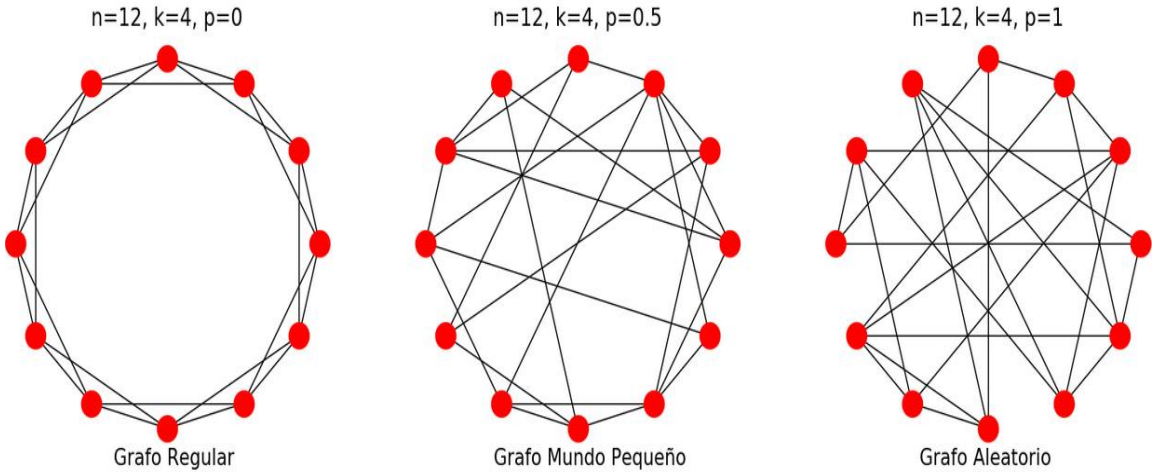


Figura 16: Transición de una red regular a una aleatoria, se observan los distintos tipos de redes, intercambiando los enlaces p aleatoriamente, mientras se aumenta el valor de p.

En la Figura 16 se muestra la transición entre una red regular[15] a una aleatoria, variando la probabilidad *p* que se redireccione un enlace aleatoriamente. Para redes regulares el diámetro crece linealmente con el número de nodos y para redes aleatorias crece como el logaritmo del número de nodos (2.3.6) [6]. Ambas redes se distinguen de la transitividad o clusterización de sus conexiones, sin embargo, lo que descubrieron Watts y Strogatz es que al aumentar ligeramente *p* el diámetro de la red pasaba de crecer linealmente a logarítmicamente, por lo que la red mantenía la propiedad de alto grado de transitividad de las redes regulares y el diámetro de las redes aleatorias. Otra propiedad interesante es que presentan robustez, es decir, no se afecta crucialmente al eliminar aleatoriamente nodos o conexiones y en varias redes de este tipo notamos que presentan una estructura de libre escala.

[15] Una red regular corresponde a una red donde todos los nodos tienen el mismo grado. $k_i = r$.

2.3.4 Red libre de escala

Se encontraron propiedades interesantes en distintos tipos de redes como la World Wide Web, la red de Artículos científicos y Citas, entre otros, demostrando que son distintas a las redes aleatorias, una propiedad que resalta de estas redes es el entrelazamiento preferencial (*Preferential Attachment*). Este nos indica que un nodo de grado alto tiene mayor probabilidad de obtener otro enlace de la que tiene un nodo de grado menor. En una red libre de escala presentan una distribución de grado en forma de ley de potencia.

$$P(k) = Ck^{-\gamma} \tag{2.3.7}$$

Con *C* una constante, *k* el grado y γ con valores que están entre 2 y 3 comúnmente.

Las comunidades son subredes de la red original que presentan un entrelazamiento de nodos más grande que con otros nodos fuera de la subred [6]. En la figura 17 se observa la representación gráfica de una red de entrelazamiento preferencial y su correspondiente distribución en forma de ley de potencia.

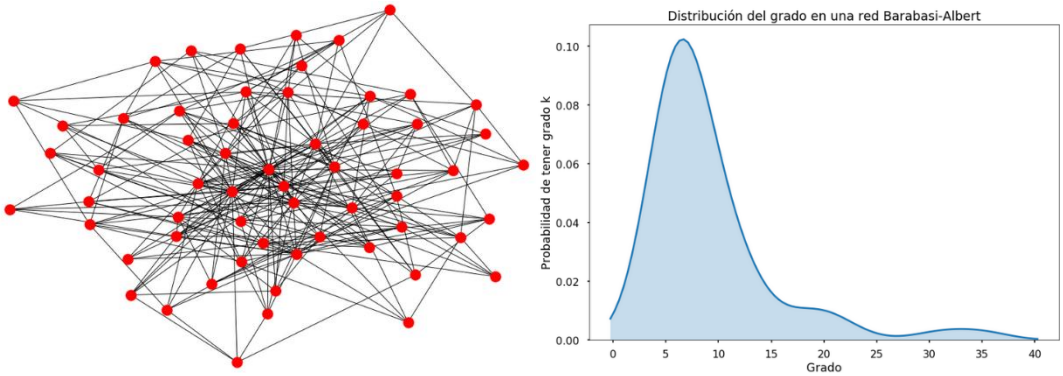


Figura 17: Representación de una red libre de escala y la distribución de grado

2.4 Detección de comunidades

Un área de estudio relevante para la teoría de redes complejas es la detección de comunidades, la estructura de comunidades se define como un conjunto de nodos que están más densamente conectados entre ellos que con el resto de la red, y estos comparten características

similares o interacciones que los relacionan. Para definir una comunidad se utilizan métricas que miden la calidad de la partición y se parten en dos tipos, locales y globales. En las definiciones locales se analiza la estructura interna de la comunidad sin tomar en cuenta la topología de la red completa. En contraste, en la definición global se analiza la importancia de la comunidad en la estructura global de la red.

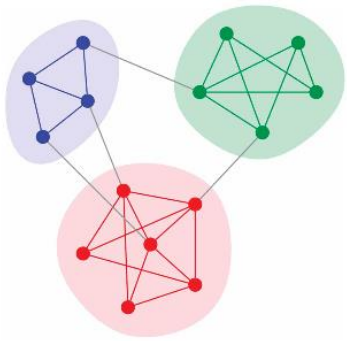


Figura 18: Red con estructura de comunidades [18].

Las particiones son divisiones de una red en comunidades o grupos, de manera que cada nodo pertenece a un grupo. A mayor número de nodos se dificulta elegir la mejor partición, dado que un nodo puede ser asignado a diversas comunidades.

Para saber si se elegio la mejor partición se utiliza la modularidad como un indicador global de la red. La modularidad se define como la diferencia entre el número de enlaces dentro de las particiones y el número de enlaces esperados en las particiones si la red se construye colocando enlaces aleatoriamente.

Descrita por la siguiente ecuación:

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(C_i, C_j) \qquad (2.4.1)$$

Donde *A* es la matriz de adyacencia de la red, k_i el grado del nodo *i*, *m* el número de enlaces totales, δ es la delta de Kronecker y vale 1 si los nodos *i, j* pertenecen a la misma comunidad *C* y cero en otro caso. El valor de *Q* se encuentra en el rango de -0.5 (inclusivo) a 1 (exclusivo). Si los enlaces cuantificados pertenecen a la misma comunidad, la modularidad será alta y las particiones propuestas describen una muy buena estructura de comunidad. La

modularidad cercana a -0.5 significa que los nodos de la misma comunidad no son enteramente adyacentes entre ellos.

El uso de esta métrica tiene un límite de resolución, ya que es incapaz de detectar comunidades de tamaño menor a un cierto límite determinado por el número de enlaces y la topología de la red, esta medida no debe usarse para comparar la calidad de la estructuración en comunidades de redes de tamaños diferentes.

Existen diversas técnicas para particionar el grafo, como espectrales, divisivos y por agrupamiento, estas técnicas buscan eliminar los enlaces que conectan a los nodos que pertenecen a distintas comunidades, quedando aisladas unas de otras. Aquí se presentan 2 algoritmos que se utilizaron para la investigación.

2.4.1 Método de percolación de clique [19]

Se utiliza comúnmente para detectar comunidades superpuestas, el método de percolación de clique acumula las comunidades de k-cliques que corresponden a una red completa de subgrafos de *n* nodos.

Estableciendo que, dos subredes son adyacentes si comparten $n-1$ nodos. Una comunidad se define como la unión de una serie de n-cliques adyacentes, dado que un nodo puede pertenecer a múltiples grupos de n-cliques al mismo tiempo las comunidades se pueden superponer.

Este algoritmo nos permite establecer el número de elementos que pertenecen al clique y generando comunidades por medio de los cliques adyacentes, en el presente trabajo se utilizó para obtener cliques con *n* elementos.

El algoritmo consiste en los siguientes pasos:

1. Se elige un valor para n.
2. Se identifican todos los n-cliques de la red.

3. Se construye una red de cliques, donde dos n-cliques estarán unidos si comparten n-1 nodos.
4. Cada componente fuertemente conexa de esta red forma una comunidad.

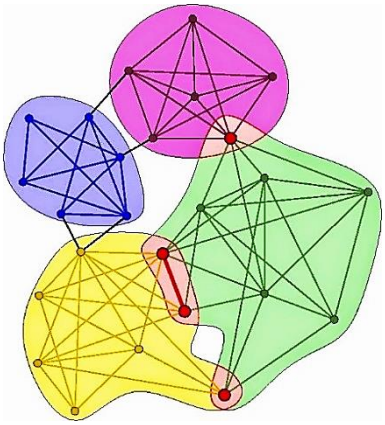


Figura 19: Comunidades de n-cliques con n = 5, con nodos compartidos por comunidades denotados de color rojo. Al ser un 5-clique estos cliques no son adyacentes entre ellos, así que no se sobreponen las comunidades [6].

2.4.2 Método por optimización de modularidad algoritmo de Louvain *[18]*

Idealmente se busca particionar una red, maximizando la modularidad, comúnmente un valor ≥ 0.6 ya muestra una estructura de comunidad notoria en la red. Para encontrar la mejor partición, se tiene que calcular la modularidad para cada posible partición y posteriormente seleccionar la mejor. Existen soluciones aproximadas para logarlo, para ejemplo el algoritmo de Louvain. Este algoritmo se divide en dos fases la primera se encarga de formar las particiones y la segunda procesa las comunidades obtenidas para identificar relaciones jerárquicas y se repiten iterativamente:

1. En un inicio todos los nodos pertenecen a distintas comunidades.
2. Se selecciona el nodo *i* y se cambia a la comunidad del vecino *j*, se calcula la modularidad de este cambio y para cada uno de los vecinos por la siguiente ecuación:

$$\Delta Q = \left[\frac{\sum_{in} k + 2k_{i,in}}{2m} - \left(\frac{\sum_{tot} k + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in} k}{2m} - \left(\frac{\sum_{tot} k}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \qquad (2.4.2)$$

Donde $\sum_{in} k$ es la suma de los pesos de los enlaces dentro de la comunidad *i* a la que se está moviendo, $\sum_{tot} k$ es la suma de los pesos de los enlaces incidentes a los nodos de la

comunidad *i*, k_i es el grado del nodo *i* y $k_{i,in}$ es la suma de los pesos de los enlaces entre *i* y los otros nodos de la comunidad a la que se movió el nodo, por último, *m* es la suma de los pesos de los enlaces de toda la red.

3. La Primera fase calcula el cambio de la modularidad para todos los vecinos conectados al nodo *i.* Se coloca a *i* donde tenga el mayor aumento de modularidad. Si no aumentó, *i* se mantiene en su comunidad original. Repitiendo estos pasos hasta que todos los nodos no tengan un aumento en la modularidad.

4. Se agrupan todos los nodos de la misma comunidad en nodos-comunidad. Los enlaces de nodos de la misma comunidad se agrupan en auto enlaces pesados. Los enlaces de nodos que van de una comunidad a otra se representan por enlaces pesados entre comunidades. Una vez generada la red, se puede volver a aplicar el paso 3 del algoritmo a la red ponderada resultante y agrupar nodos comunidad hasta que no haya un cambio significativo en la modularidad.

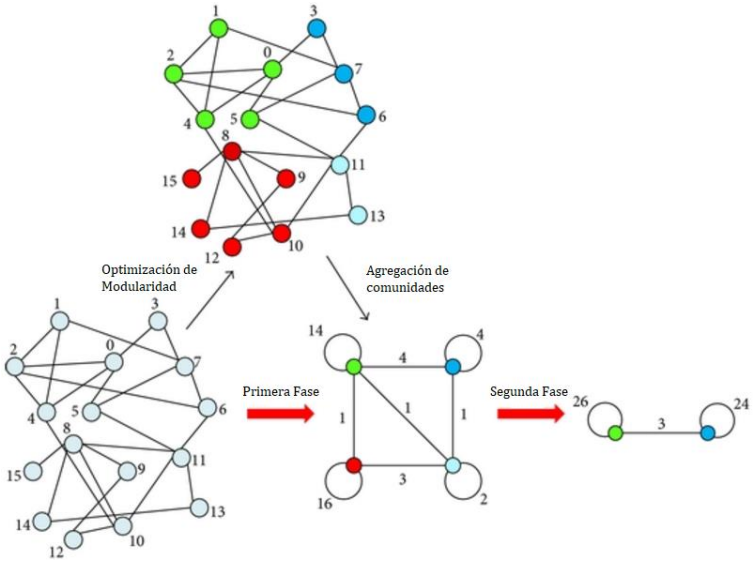


Figura 20:Proceso del algoritmo de Louvain [18].

Capítulo 3 Metodología

Actualmente no existe una metodología estándar que permita identificar los caminos alostéricos en una proteína. Con el algoritmo desarrollado se propone identificar los residuos que son alienables y que puedan describir los residuos con mayor relevancia para el alineamiento y detectar los residuos característicos de los caminos alostéricos por medio de las redes de correlación dinámica.

Como se indicó en la introducción se plantearon 2 análisis que corresponde a la detección de similitudes locales y al análisis de la red de correlación dinámica, se procede a detallar cada paso del respectivo análisis. El algoritmo generado durante el análisis de detección de similitudes locales se utiliza para la construcción de la RCD, por medio de las parejas de residuos que se alinean, con una estructura de referencia, con el fin de mejorar la calidad del análisis de los cambios conformacionales. De igual manera, se hace un análisis de detección de comunidades con la red de contacto residuo a residuo (RCRR) generada en el primer análisis.

3.1 Detección de similitudes locales.

Con el fin de generar un alineamiento de estructuras proteínicas tal que este sea independiente de la secuencia de aminoácidos. El algoritmo se basó en la metodología propuesta por Nguyen *et al*. [1], se implementa el algoritmo en lenguaje Python y se proponen modificaciones a la metodología.

El algoritmo sobrepone un par de estructuras de proteínas independiente de su topología, basándose en mera geometría, es decir, tomando las coordenadas cartesianas de los átomos representativos de los residuos, generando trasformaciones y filtros, que se explican más adelante, obteniendo cuales residuos y localidades son alienables.

Para medir la similitud y la calidad del alineamiento se utilizó la desviación del error cuadrático medio, mejor conocido como *root-mean-square deviation* (RMSD, por sus siglas en ingles), calculado con las posiciones de los átomos:

$$RMSD(\theta) = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(\theta - \theta_0)^2} \qquad (3.1.1)$$

Donde $\theta = (x, y, z)$ son las coordenadas cartesianas, comúnmente medidas por los átomos pesados de Carbón C, Nitrógeno N, Oxigeno O y Carbonos alfa C_α, y n el número de átomos comparados y te devuelve en unidades de ángstroms Å $[10^{-10}m]$. En los análisis desarrollados en este trabajo se consideró únicamente el carbono alfa de cada residuo.

3.1.1 Extracción de variables

Si tomamos en cuenta las primeras cadenas polipeptídicas llamadas A, B, C, ... se recolectó la información de proteínas o de ensambles en formato PDB, extrayendo las coordenadas de los átomos representativos de la primera cadena polipeptídica "A", para la presente tesis se tomó los C_α de cada residuo.

Se calculó la estructura secundaria utilizando el algoritmo DSSP, basándose en el cálculo de los ángulos diedros [8]. Se guardan atributos del residuo, como las coordenadas, la estructura secundaria y el número de residuo.

3.1.2 Generación de cliques

Para generar los cliques y la red de contacto, se calculó la distancia euclidiana definida como:

$$D_E = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \qquad (3.1.2)$$

Donde x, y, z corresponde a las coordenadas de cada átomo y se calcula entre cada residuo de la proteína.

Por medio de este cálculo se genera una matriz de adyacencias y se construyó una red, donde los nodos son los C_α de los residuos y los enlaces serán 1 o 0 dependiendo de la distancia entre ellos, generando los enlaces de la siguiente manera:

$$A_{\mathrm{ij}} = \begin{cases} 1 \; si \; d_{ij} \leq d_{th} \\ 0 \; si \; d_{ij} > d_{th} \end{cases} \qquad (3.1.3)$$

En este trabajo se consideró $d_{th} = 10\,\text{Å}$ para generar la red de contacto residuo a residuo como lo indica Nguyen *et al* [1].

Se construyó la red usando la librería NetworkX, utilizando la matriz de adyacencia [20]. Obteniendo la RCRR que nos ayudó para el análisis de detección de comunidades del segundo análisis.

Posteriormente se extraen los cliques de la red por medio del algoritmo de percolación de cliques, utilizando NetworkX, indicando el valor *n* que es el número de elementos del clique a generar, se comienza con $n = 7$, guardando la lista de 7-cliques para posteriores cálculos.

Una vez filtrados los 7-cliques se generan los 3-cliques, tomando un 7-clique se hacen las permutaciones de 3 elementos con los residuos de la siguiente manera:

$$[2,3,4,5,6,7] \rightarrow [2,3,4], [4,3,2], \ldots, [6,7,5], [5,6,7]$$

Este procedimiento se realiza sobre cada elemento del par de proteínas a comparar; proteínas A y B.

3.1.3 Comparación de cliques

La comparación de cliques se realizó uno a uno, es decir, el primer clique de la proteína A se comparó con todos los cliques de la proteína B. Después el segundo clique de la proteína A se comparó con todos los cliques de la proteína B, y así sucesivamente.

Cabe notar que para cada 7-clique se generan combinaciones de 3 elementos donde importa el orden y no se pueden repetir los residuos, por ende, se obtienen 210 3-cliques. Si vamos a comparar alrededor 1000 cliques vs 1000 de la otra proteína, se realizarían alrededor de 210 millones de cálculos en cada iteración del algoritmo.

Para el acoplamiento de cliques y con él fin de no hacer cálculos innecesarios se generaron diversos filtros para posteriormente computar el RMSD entre cliques de la proteína A con la proteína B.

Antes de comparar los 7-cliques se implementó un filtro sobre los ángulos diedros. Este filtro, no considerado en el trabajo de Nguyen *et al*. [1], calcula la diferencia entre los ángulos de los elementos del clique con el ángulo de los elementos del clique, dejando aquellos que obtienen un valor menor al establecido.

Para obtener un alineamiento adecuado se aplicó el siguiente filtro que utiliza el *score* de estructura (SE) y se define de la siguiente manera.

$$SE(A_i, B_j) = \begin{cases} 0 & si \quad ES(A_i) == ES(B_j) \\ 1 & si\ ES(A_i) \neq ES(B_j) \wedge (ES(A_i) = Coil \mid ES(B_j) = Coil) \\ 2 & Otro\ caso \end{cases} \qquad (3.1.4)$$

Donde ES corresponde a la Estructura Secundaria y la tabla 3 define los casos del ES:

Estructura Secundaria (ES)	Coil	α-hélice	β-lamina
Coil	0	1	1
α-hélice	1	0	2
β-lamina	1	2	0

Tabla 3: Matriz de equivalencia entre elementos de estructura secundaria en proteínas.

El filtro consiste en comparar aquellos cliques si $SE[A_i, B_j] < 2$, buscando, de preferencia, estructuras iguales entre residuos para que sean posibles candidatos para alinearse. Los cliques seleccionados (A_i, B_j) se toman los elementos del clique y se comparan en parejas, de manera que $ES(A_{i_1})$ se compara con $ES(B_{j_1})$ y $ES(A_{i_2})$ se compara con $ES(B_{j_2})$, así sucesivamente, evaluándose por medio de la ecuación *(3.1.4)* y dejando aquellos cliques que

cumplan él filtro.

Previo al cálculo de la distancia (RMSD) entre cada par cliques, se realizó un alineamiento mediante un ajuste de mínimos cuadrados basado en quaterniones *[21, 22]*. Detalles de este procedimiento sobre la rotación y traslación óptima de la proteína se encuentra resumidos en el apéndice A.

Para cada clique de la proteína A se comparó uno a uno con los cliques de la proteína B y serán candidatos para alinearse todos aquellos cliques que no sobrepasen la siguiente restricción de RMSD que depende del número de elementos del clique (*n)*.

n	3	4	5	6	7	8	9
$RMSD_n$(Å)	0.15	0.30	0.60	0.90	1.50	1.80	2.10

Tabla 4: Restricción del RMSD para diferentes valores del tamaño del clique (valores tomados de [1])

Una vez que se obtienen los posibles candidatos a alinearse de la primera iteración de 3-cliques de A y B, se extienden los cliques A_i^3 y B_i^3 a un 4-clique A_i^4 y B_i^4 respectivamente, agregando un residuo A_s y B_l de los 7-cliques que no está en el actual 3-clique, de modo que $A_i^4 = A_i^3 \cup A_s$ y $B_i^4 = B_i^3 \cup B_l$ y nuevamente se calculó el RMSD rotando y trasladando los nuevos cliques uno a uno, aplicando los filtros de distancia mínima, SE y de restricción del RMSD con ahora valor a 0.30.

Se obtienen nuevos candidatos y a todos los posibles 4-cliques candidatos se extienden agregando un residuo más en cada paso, hasta el máximo clique posible, en este trabajo se consideró hasta 7-cliques, aplicando los filtros correspondientes que se mencionaron anteriormente en cada iteración.

3.1.4 Alineación

Se obtiene una lista de candidatos de 7-cliques de la proteína A y B, un clique puede ser candidato para alinearse con más de un clique, por lo que, para la alineación se considera el *score* de sobreposición (SS) definido por:

$$SS = \frac{número\ de\ residuos\ alineados\ a\ menor\ de\ 3.5\ Å}{número\ de\ residuos\ alineables}\ \% \qquad (3.1.5)$$

Y se guarda el alineamiento y emparejamiento que obtengan mayor *SS*.

Para generar este alineamiento se calcula el RMSD global utilizando la matriz de rotación de los candidatos y su respectivo baricentro aplicando a toda la proteína y haciendo el respectivo ajuste de mínimos cuadrados para obtener el RMSD de cada residuo. Un aspecto de suma importancia es encontrar que residuo se alinea con cual, por lo que al momento de calcular el RMSD para cada candidato se ejecuta un alineamiento de residuo a residuos que no pertenecen al clique, alineando los residuos que estén a menor de 3.5 Å y si hay más de un residuo que cumple se empareja el residuo con el de menor distancia entre ellos.

Se ejemplifica los pasos anteriormente descritos por medio de la siguiente figura.

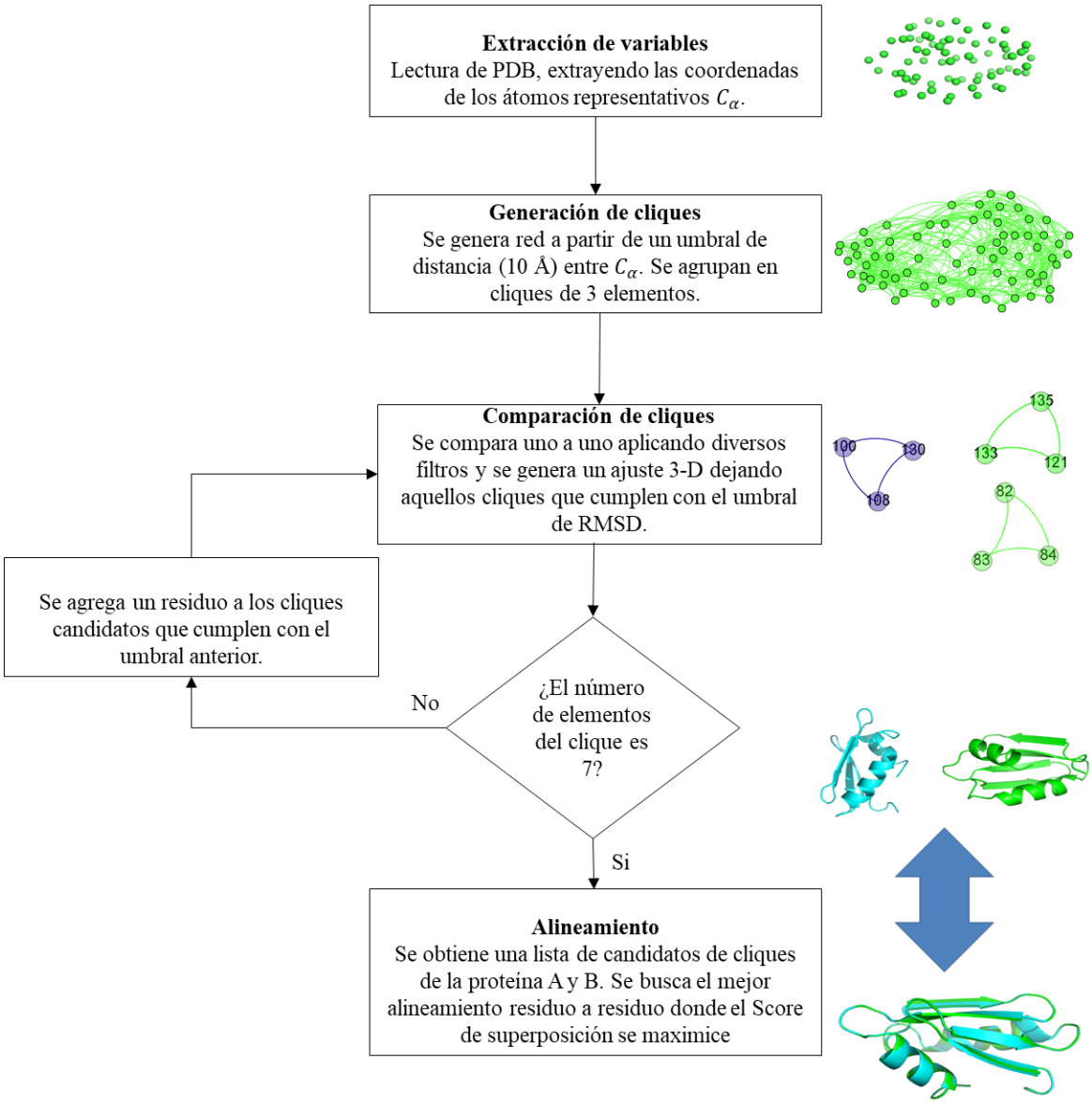


Figura 21: Metodología del algoritmo para el alineamiento de proteínas.

Finalmente, se obtiene un archivo de salida con la pareja de 7-cliques con el mejor alineamiento, las coordenadas de los residuos alineados, las parejas de residuos donde maximizan el SS y para validar el alineamiento también se calcula el RMSD global del alineamiento.

3.2 Red de correlación dinámica

La red de correlación dinámica (RCD) es una manera de modelar el fenómeno de la alostería donde los residuos transmiten información y regulan sus actividades catalíticas a pesar de estar alejadas unas de otras [23]. Las simulaciones de dinámica molecular permiten explorar el espacio de configuraciones al que una proteína tiene acceso. El muestreo de dicho espacio configuracional está representado en la trayectoria (secuencia de configuraciones con respecto al tiempo) que resulta de la integración de las ecuaciones de movimiento. En principio, los procesos de comunicación que gobiernan el fenómeno de la alostería deberían emerger de un análisis apropiado de dichas trayectorias. La construcción de una RCD es un intento para obtener dicha información a partir de simulaciones de dinámica molecular. En nuestro caso particular, se emplea el algoritmo de detección de comunidades locales para identificar el conjunto de residuos que participarán en el alineamiento de la trayectoria con una proteína de referencia. La relevancia de dicho alineamiento reside en que los métodos de correlación involucran la desviación con respecto a una estructura de referencia. Se establece analizar las posibles comunidades en la red de contacto promedio durante la trayectoria, así como las correlaciones entre ellas.

3.2.1 Construcción de dinámica

Se implementan las simulaciones con condiciones iniciales diferentes para comparar la proteína de interés. Por ejemplo, una proteína con y sin cofactor debería generar RCD diferentes. Esto bajo el supuesto de la presencia del cofactor induce un efecto alostérico. Las trayectorias de la simulación se alinean a una estructura de referencia con el fin de identificar los cambios conformacionales.

Se analiza el alineamiento de la trayectoria y la estructura de referencia graficando la evolución de los valores de SS y RMSD, analizando las dinámicas que presentan un comportamiento anómalo.

3.2.2 Extracción de variables[16]

[16] Por variables se entiende las coordenadas atómicas de los carbono alfa de los residuos en la proteína.

Para el análisis del alineamiento y de la búsqueda de caminos alostéricos de las proteínas de interés, se genera la RCRR, utilizando la metodología de generación cliques (3.1.2). En este caso, se genera la red de contacto de cada estructura de la trayectoria. Para construir la red promedio de contacto residuo a residuo, se agrupan las redes en una única red en donde se mantienen aquellos enlaces que aparecen con cierta frecuencia en las redes de la trayectoria (>70%).

Se genera la matriz de correlación cruzada de la dinámica de residuos que describe las fluctuaciones atómicas [24]. Para la construcción de la red se calcula la correlación de Pearson durante la dinámica con respecto a la matriz de covarianza de las coordenadas definida de la siguiente manera:

$$C_{i,j} = \frac{\langle (r_i - \langle r_i \rangle) \cdot (r_j - \langle r_j \rangle) \rangle}{\sqrt{(\langle r_i^2 \rangle - \langle r_i \rangle^2)(\langle r_j^2 \rangle - \langle r_j \rangle^2)}} \qquad (3.2.1)$$

Donde los brackets cerrados representan valores promedio y r_i, r_j son las coordenadas de los átomos *i* y *j*, los valores de la correlación C_{ij} están entre –1 y 1 que corresponde a negativamente correlacionado y positivamente correlacionado.

3.2.3 Construcción de RCD

Generada la red y los cálculos apropiados para describir la topología se comparan y se construye una última red que consiste en multiplicar el peso de los enlaces generados por la matriz de correlación cruzada y los enlaces de la red de contacto, describiendo el nuevo valor de los enlaces de la siguiente manera:

$$w_{ij} = \begin{cases} b_{ij} \times d_{ij} & si\ b_{ij} = 1 \\ d_{ij} & si\ b_{ij} = 0 \wedge |d_{ij}| > 0.6 \end{cases} \qquad (3.2.2)$$

Donde b_{ij} corresponde a 1 o 0 si la distancia entre *i* a *j* es menor a 10 Å y d_{ij} es la correlación durante la dinámica. Dado que la alostería se presenta entre residuos que aparentemente no están conectados directamente, se agregan los enlaces que sobrepasen el umbral de correlación.

3.2.4 Análisis

La hipótesis principal consiste en obtener caminos alostéricos por medio de la intermediación, calculando el camino más corto entre los residuos *m* y *k* tal que pasen por el residuo *l*, definido como:

$$c_i = \frac{1}{N} \sum_{l \neq m \neq k} \left(\sigma_l(m,k) \right) \qquad (3.2.3)$$

Donde *N* es el número de caminos más cortos que pasan *m* y *k* por medio de *l*. Ordenando por esta métrica a los residuos y obteniendo los residuos que conecten partes distantes y tengan alta centralidad de intermediación.

Se hace el análisis topológico, aplicando el algoritmo de detección de comunidades a la red de contacto residuo a residuo y se analizan las métricas de centralidad de los residuos, comparando el cambio en métricas de centralidad en la red de correlación dinámica y se analiza el cambio de correlación en trayectorias de la misma proteína con distintas condiciones iniciales.

Resumiendo, la metodología de Red de correlación dinámica en la siguiente figura:

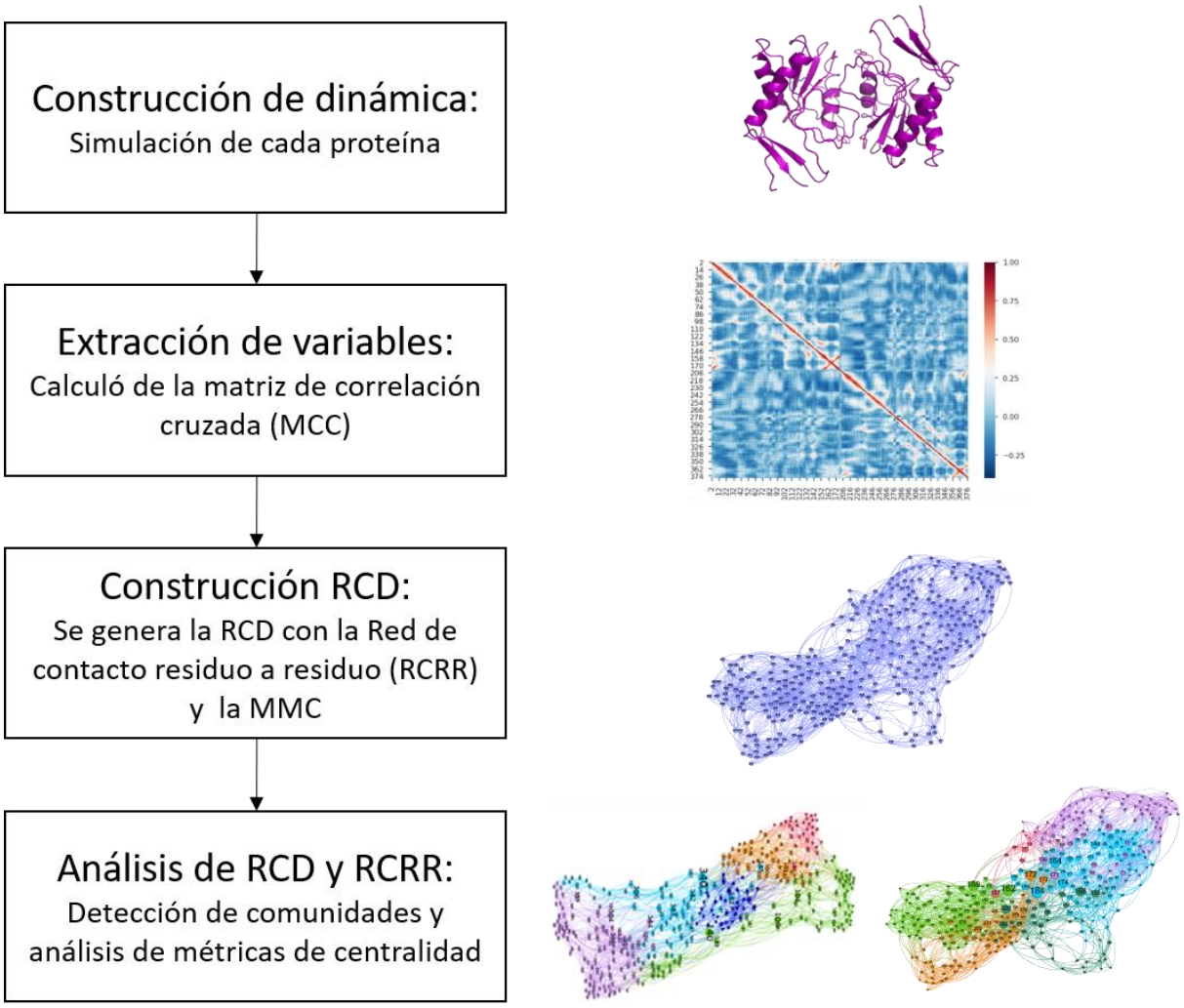


Figura 22: Metodología del algoritmo para la red de correlación dinámica.

3.3 Código

El código se desarrolló en lenguaje Python versión 3.6 y la paquetería necesaria para ejecutar el código es:

Librería	Versión
Python	3.6 +
Scipy	1.1.0
Pandas	0.23.4
Numpy	1.15.1
NetworkX	2.2
Python-louvain	0.13

Anaconda (Recomendado)	2018.12
DSSP	3.0.7
Itertools	3.2
Math	3.5
Copy	3.0
Glob	3.4

Tabla 5: Requerimientos de librerías para ejecutar el código

El software se desarrolla en una PC con sistema operativo Linux (Ubuntu 16.4) con un procesador Intel Core i7-7700HQ @2.80Ghz y memoria RAM DDR4 de 16 Gb y los experimentos se implementaron en un servidor de 32 núcleos físicos con 8Gb de RAM.

El software generado es llamado Mani y se procedió a comparar su rendimiento con el software generado por Nguyen *et al* [1] alias Click.

Repositorio oficial del proyecto: https://github.com/marciniega/pdbmani

Capítulo 4 Discusión y Resultados

El primer experimento que demuestra la correcta implementación del algoritmo fue utilizando la proteína 1xxa, utilizada como ejemplo en Click [1], y la misma proteína en otro estado inicial, obteniendo el regreso a la posición inicial con valores de $SS = 100$ y un $RMSD = 0$ comprobando el adecuado alineamineto.

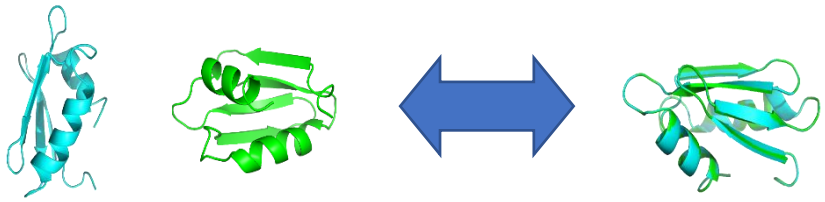


Figura 23: La proteína 1xxa rotada y trasladada (azul turquesa) y la misma proteína en un estado inicial (verde limón). Obteniendo como resultado la configuración original con los colores traslapados.

4.1 Resultados de alineamiento Mani vs alineamiento Click

Con el objetivo de comprobar la eficiencia de nuestro algoritmo, llamado Mani, se comparó el rendimiento con el software Click. Para ello se analizó el rendimiento de los algoritmos al alinear 96 parejas de proteínas con cierto grado de similitud; similitudes del 30, 50, 70 y 95 %, agrupados en clústeres de acuerdo con MMseqs2[17] [25]. Las métricas empleadas para realizar la comparación del rendimiento fueron el *Score* de Sobreposición (SS), RMSD y el número de parejas de residuos alineadas.

[17] En los cluster clasificados por MMseqs2 el grado de similitud es la cota mínima del cluster, por lo que se pueden encontrar parejas de proteínas con alta similitud en un cluster de bajo grado.

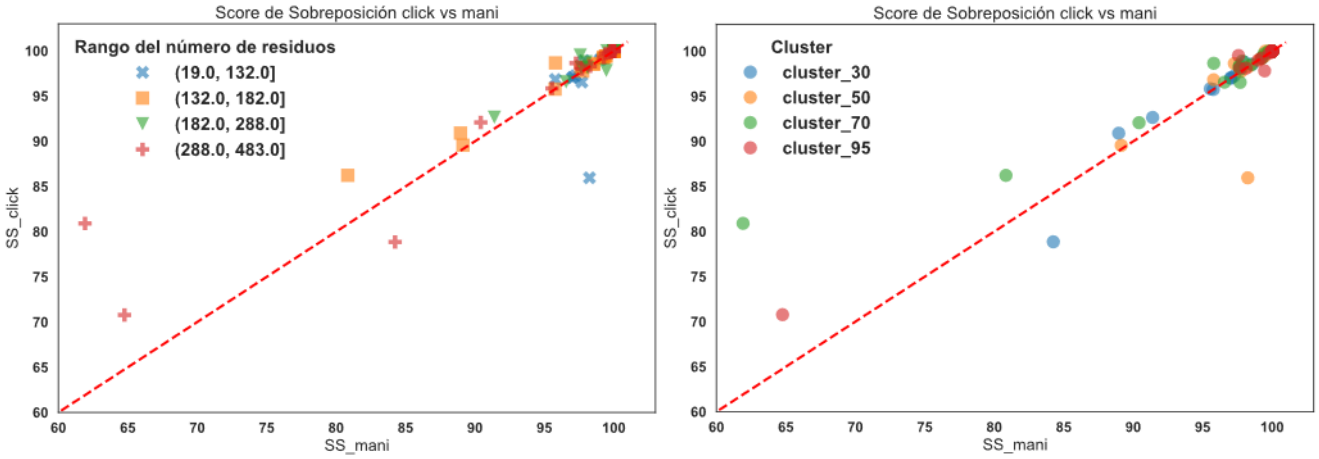


Figura 24: Comparación de Score de Sobreposición. En la primera gráfica se denotan por distinta figura dependiendo del rango del número de residuos. En la segunda gráfica el color del punto depende de la similitud del clúster que pertenecen.

Existen pocas diferencias entre Mani y Click de acuerdo con el *Score* de Sobreposición (Figura 24). No se observó una relación directa entre mayor número de residuos con los *Scores* de Sobreposición (SS) que exija a ambos programas a tener un menor desempeño. Las estructuras que están encima o por debajo de la franja roja tienen un SS mayor o menor que el obtenido por Click, respectivamente. En general, se observa que un 71% de los experimentos cumplen un buen desempeño, es decir, tienen un *score* igual o mayor que al obtenido por Click.

Ahora se grafica el RMSD:

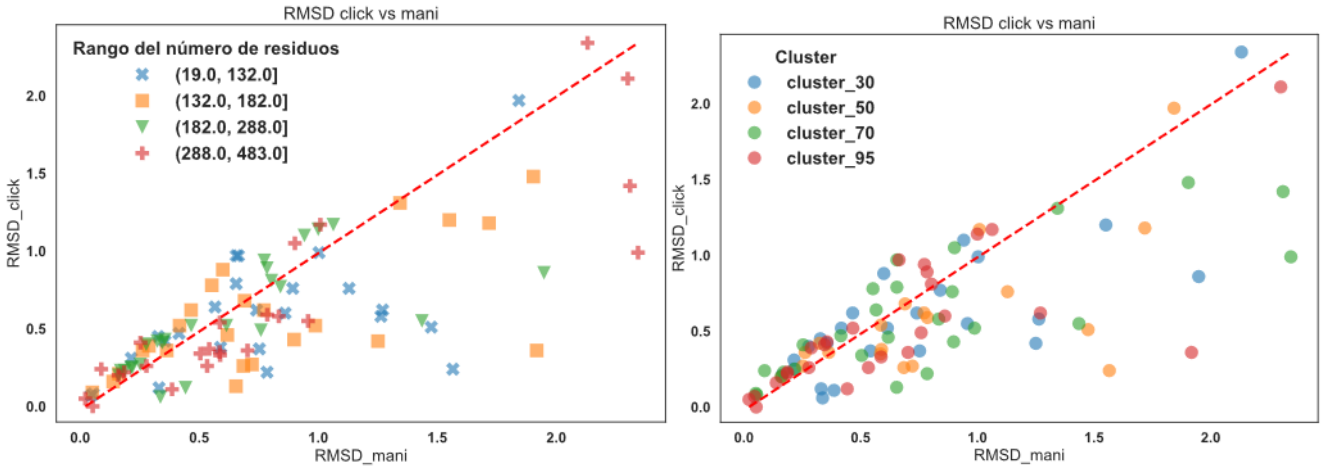


Figura 25: Comparación del RMSD. En la primera gráfica se denotan por distintas figuras dependiendo de un rango del número de residuos. En la segunda gráfica el color del punto depende del clúster al que pertenecen ambas proteínas.

La siguiente tabla muestra las métricas de evaluación sobre SS Mani y SS Click.

	Correlación	RMSE	Residuo Promedio
SS	0.91	2.58	0.29
RMSD	0.75	0.40	-0.16

Tabla 6: Métricas de evaluación del Score de Sobreposición (SS) y del RMSD.

La correlación nos indica la semejanza entre valores obtenidos y como se observa en la Figura 24 se obtienen valores similares. La raíz del error cuadrático medio nos indica que se tiene un error en promedio de 2.5 puntos a lo esperado que corresponde al 2.6% en promedio, siendo un error bajo para los estándares de esta investigación. El residuo promedio es la diferencia entre el resultado esperado y el generado por Mani, obteniendo que Click tiene un SS de 0.29 puntos porcentuales mejor que el desarrollado.

En la evaluación del RMSD, se obtiene un mayor valor, en promedio de 0.16 a comparación del de Click. Esto indica que se obtiene un alineamiento entre residuos que en algunos casos no es el mejor, esto se debe por establecer umbrales en los filtros muy estrictos y no permitiendo encontrar el alineamiento óptimo, por cuestiones de costo computacional se mantiene estos umbrales estrictos.

El tiempo de ejecución para el alineamiento de 2 proteínas con 350 átomos de C_α tardo $57.2s \pm 695ms$.

4.2 Análisis de alineamiento en dinámicas moleculares

La alostería es el sitio de actividad biológica que se ve afectado por los estados que adquieren regiones distantes a dicho sitio. Actualmente no existe una metodología estándar que permita identificar las regiones esenciales que definen los comportamientos característicos de la función biológica, ni la obtención de caminos alostéricos.

La finalidad de este software es identificar regiones muy similares entre proteínas y para la presente investigación se estudiaron los residuos que generan un cambio conformacional y funcional a diversas condiciones, analizando los residuos que generan el alineamiento.

4.2.1 Caso de estudio

Para la presente tesis, se estudiaron la biosíntesis de proteínas que forman a la pirimidina, en particular las proteínas de la familia PyrR en forma de dímero y tetrámero en complejo con los cofactores UTP y GTP, respectivamente. Las estructurales cristalográficas empleadas fueron provistas por el laboratorio del Dr. Alfredo Torres Larios, investigador del Instituto de Fisiología Celular, UNAM.

Las dinámicas para el dímero PyrR se generaron por medio del software GROMACS con 2 condiciones iniciales distintas [26]; en presencia y ausencia del cofactor (UTP – Uridina trifosfato). Similarmente, para el tetrámero PyrR se simularon 2 condiciones; en presencia y ausencia del cofactor (GTP – Guanosín trifosfato). Para ambos casos, dímero y tetrámero (Figura 26 y 27), Se simularon durante 250ns, obteniendo 500 estructuras llamados *frames*.

En lo subsiguiente, llamaremos:

- Dímero en presencia del cofactor UTP - dímero con cofactor
- Dímero en ausencia del cofactor UTP - dímero sin cofactor
- Tetrámero en presencia del cofactor GTP - tetrámero con cofactor
- Tetrámero en ausencia del cofactor GTP - tetrámero sin cofactor

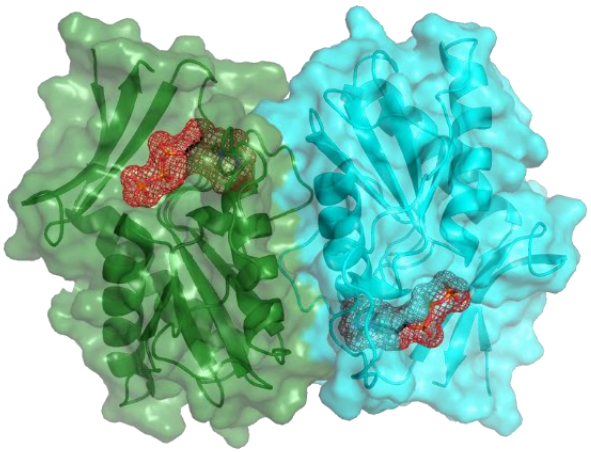


Figura 26: Representación del dímero con cofactor, las cadenas fueron coloreadas para distinguirse, así como, el cofactor en rojo.

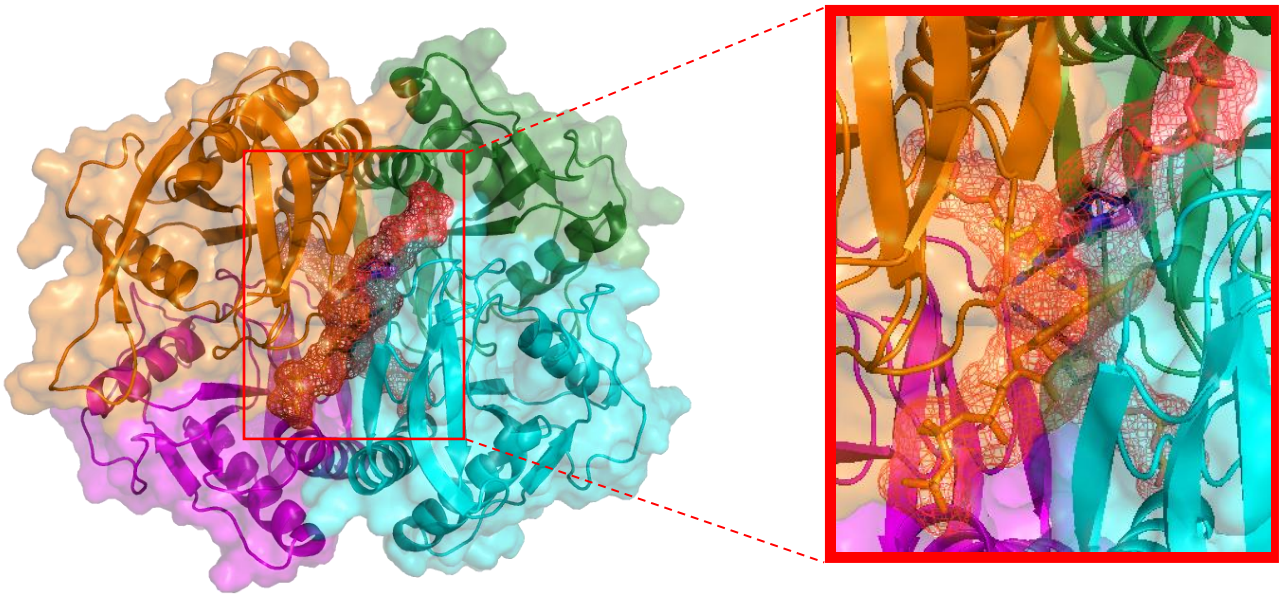


Figura 27: Representación del tetrámero con cofactor.

Posterior a la producción de las trayectorias de simulación de dinámica molecular, se realizaron alineamientos con respecto a la estructura de referencia (estructura cristalográfica). Se graficó la evolución del SS y RMSD.

4.2.2 Resultados de alineamiento

Se graficó la evolución del SS y el RMSD para analizar la dinámica de interés donde hubo un cambio abrupto a su evolución y poder identificar que originaron los cambios funcionales y conformacionales.

Se alinean con respecto a un dímero y un tetrámero de referencia y se graficó la evolución de sus valores.

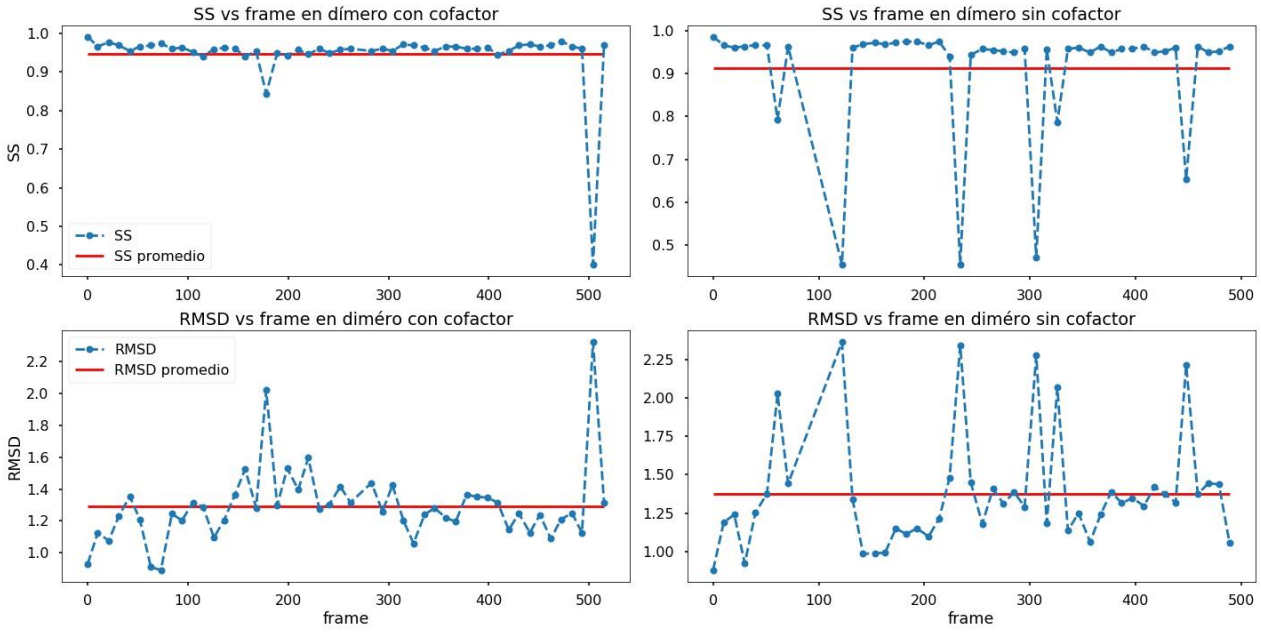


Figura 28: Score de Sobreposición y RMSD vs frame donde se alinea con respecto a un dímero de referencia.

Se observa que existe una relación negativa con los valores de *Score* de Sobreposición y el RMSD, tiene sentido dado que si se logran alinear un mayor número de parejas el RMSD disminuye.

Al analizar las gráficas de SS vs el número de frame, se observan caídas abruptas en el valor de SS, esto se debe a que el umbral del filtro de los ángulos diedros con el que se hizo el alineamiento es estricto y no elige otros posibles cliques donde podría encontrar un mejor alineamiento. El *score* mejora al probar el alineamiento con un umbral menos estricto.

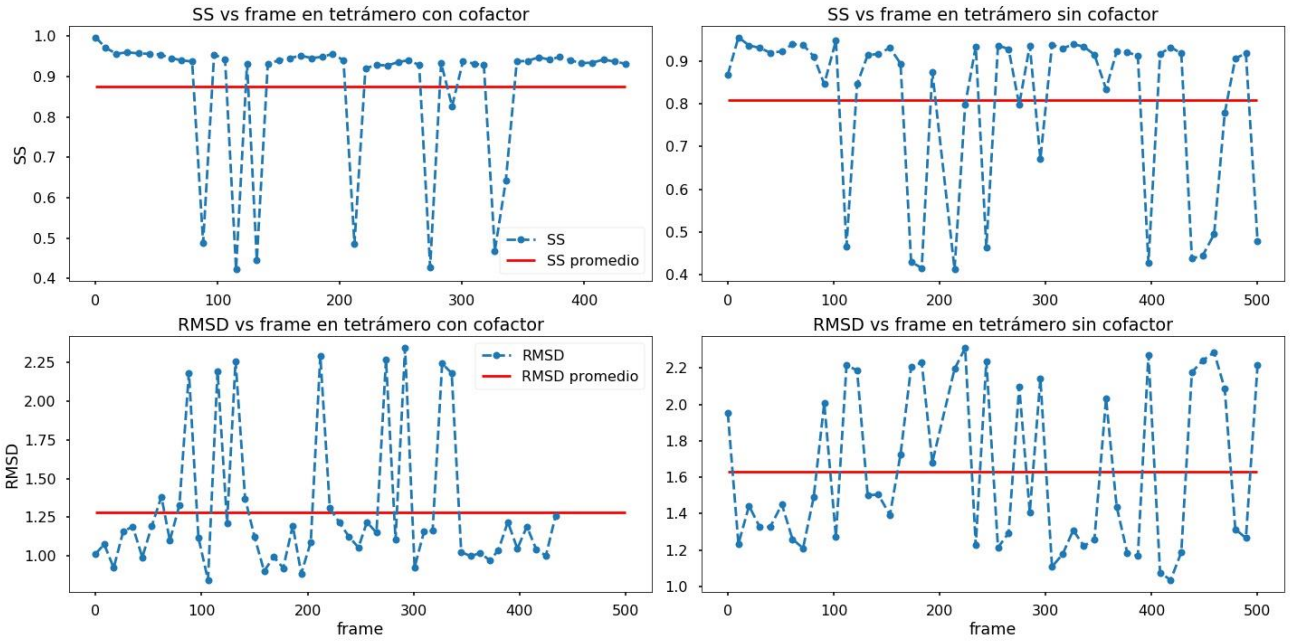


Figura 29: Score de Sobreposición y RMSD vs frame donde se alinea con respecto a un tetrámero de referencia.

Al comparar las métricas de la evolución en el tetrámero se observa que el alineamiento tiene mayor número de caídas en el *score* de sobreposición sin el cofactor que con él. Esto sugiere que el cofactor ayuda a una conformación más natural con respecto al tetrámero de referencia que sin él.

Todas las conformaciones, a pesar de tener altibajos en el *score* de sobreposición tienen un alineamiento adecuado al mantener un RMSD promedio bajo de 1.6 Å durante la dinámica. No se encontraron repeticiones en los cliques que generaron el mejor alineamiento en cada frame, por lo que se sugiere probar con mayor cantidad de frames el alineamiento.

4.3 Análisis de RCD

Una vez que se construyeron las redes de contacto y de correlación dinámica, se muestran en forma de red de contacto con las comunidades coloreadas y su símil en la proteína. Los nodos son los carbonos alfa de los residuos y etiquetados por el número de residuo

correspondiente a la proteína, los números del 0 al 180 corresponden a residuos de la cadena A y los residuos del 200 al 380 corresponden a residuos de la cadena B. El tamaño del nodo corresponde al valor de la centralidad de intermediación.

Dada la definición de alostería, el análisis de la red es de sumo intereses, ya que la hipótesis que se plantea para encontrar caminos alostéricos es por medio de la definición de intermediación (*Betweenness*), analizando residuos con una alta centralidad de intermediación y los caminos que pasan entre comunidades de una región a otra por medio de estos residuos para encontrar caminos alostéricos. El calculó de la intermediación se generó en la RCRR que corresponde a una red no pesada.

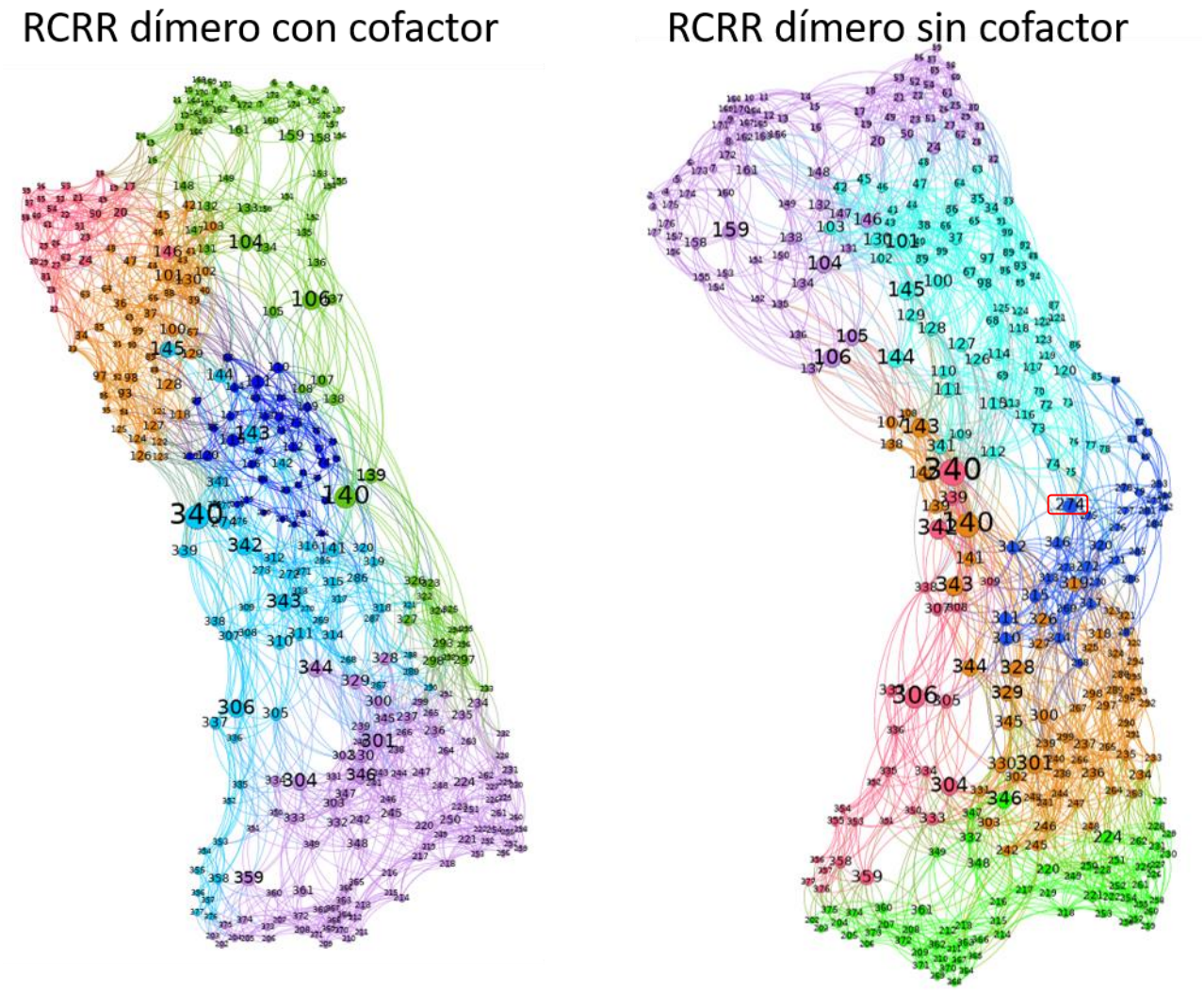


Figura 30: RCRR del dímero con cofactor y sin cofactor. Con las comunidades coloreadas y el tamaño de los nodos depende del valor de la intermediación.

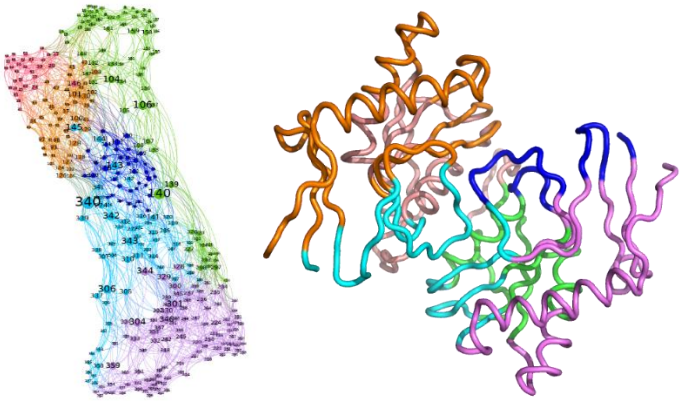


Figura 31: Red de contacto del dímero con cofactor, con las comunidades detectadas, coloreadas en la red y en la proteína.

Se observa en las comunidades de la parte superior de la red del dímero con cofactor se unen al momento de quitar el cofactor (Figura 30). Esto es debido a que el cofactor se encuentra entre esos residuos, alejando los nodos una distancia considerable para evitar estar en "contacto", sin embargo, prevalece en 6 comunidades en ambas redes. De igual manera se mantienen los nodos con una intermediación alta como los residuos 140, 340, 342, 343, 306, 304, 301, 346, 145, 106. Lo interesante en esta red de contacto son los residuos que tienen una alta intermediación y comparten enlaces con otra cadena, como el residuo 274 en la red sin cofactor (Figura 30). En ese caso se observa que comparte enlaces con los residuos del 70-110 y ese mismo residuo en la red de cofactor se observa una disminución en su valor, dejando de ser un residuo con alta transmisión de información de una cadena a otra.

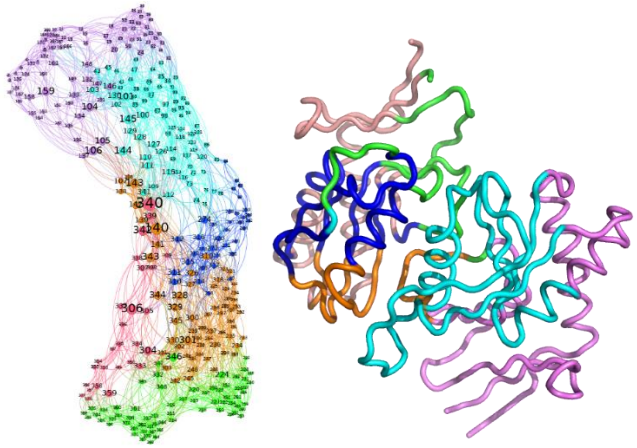


Figura 32: Red de contacto del dímero sin cofactor y sus comunidades en la proteína.

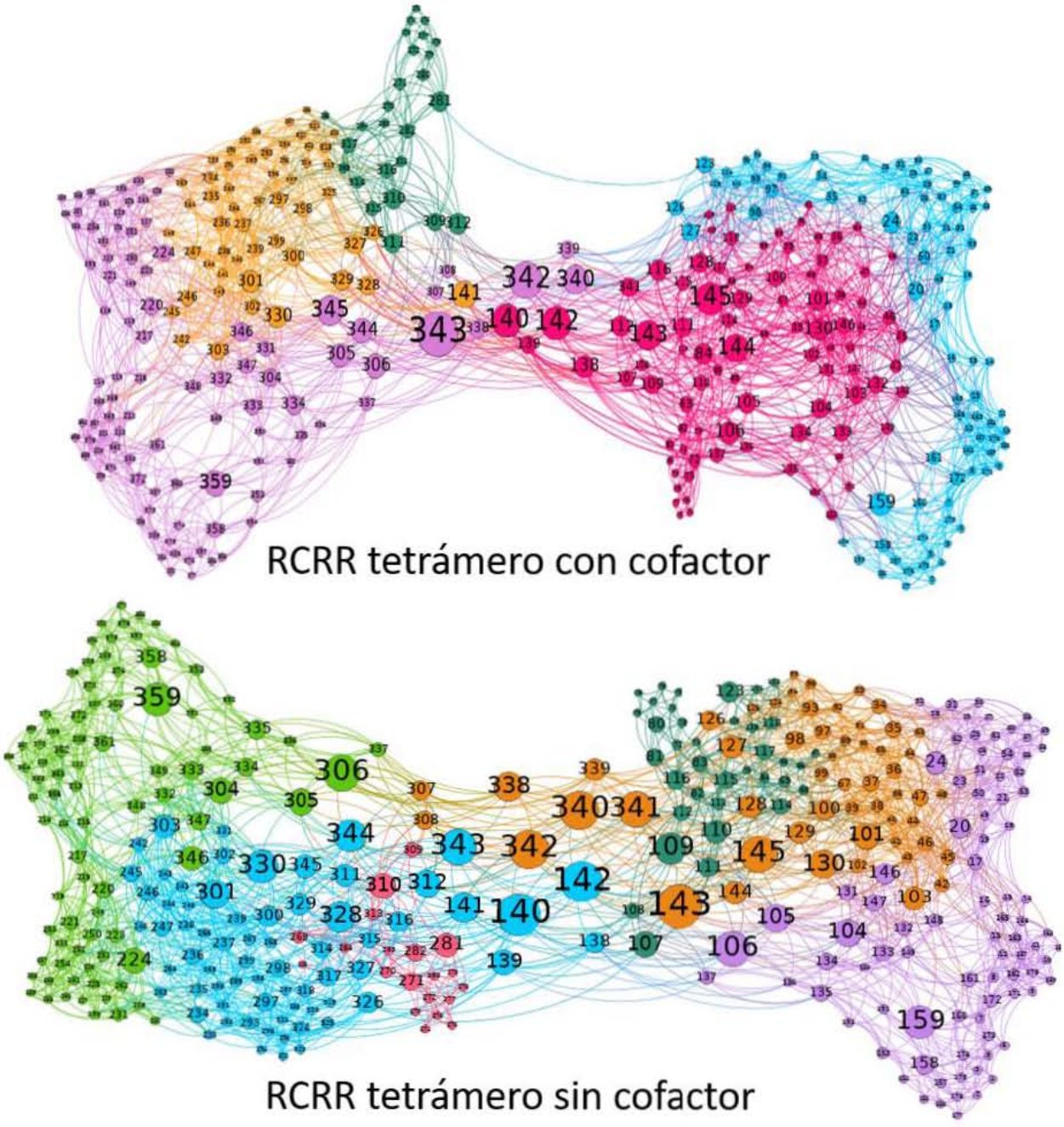


Figura 33: Red de contacto del tetrámero con y sin cofactor y sus comunidades en la proteína.

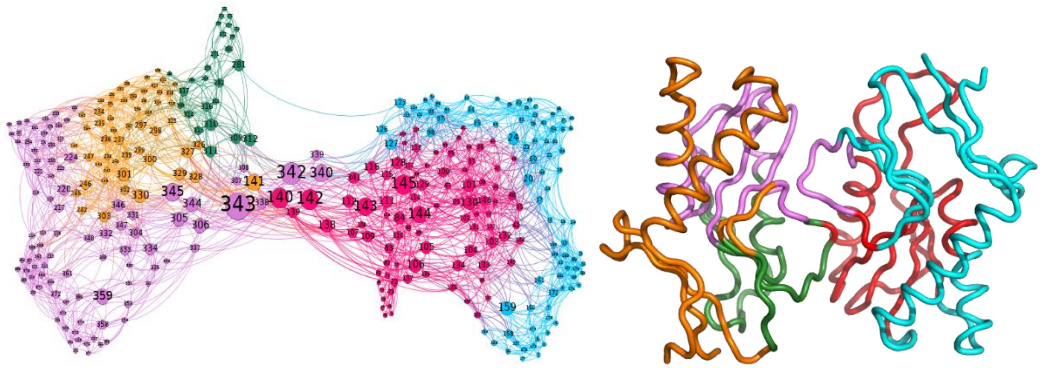


Figura 34: Red de contacto del tetrámero con cofactor y sus comunidades en la proteína.

Con el fin de poder establecer una comparación directa con el dímero, en las simulaciones del tetrámero se analizó únicamente uno de los pares del tetrámero. El número de comunidades detectadas en la red de contacto del tetrámero con cofactor son 5 mientras en la red del tetrámero sin cofactor son 6 comunidades (Figura 33), esto quiere decir que una comunidad se unió a otras, en la Figura 33 se observa que los grupos de color violeta y azul cielo son dos grandes comunidades y en su símil de red de tetrámero sin cofactor se representan por las comunidades verde alga, naranja y morado. En la comunidad naranja sin cofactor tiene nodos de la cadena A y B, siendo de posible interés dado que agrupa enlaces que van de una cadena a otra y por el otro lado las comunidades violeta y morada con cofactor comparten enlaces de una cadena a otra a pesar de que las comunidades están separadas. Los posibles caminos alostéricos para comunicar la interfaz de la proteína son con los residuos 345,343,342,340,140,142,143,145,144 en la red del tetrámero con cofactor.

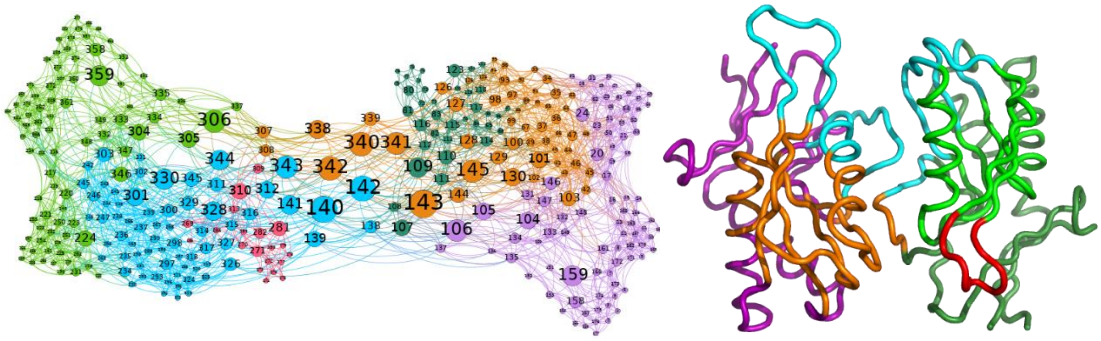


Figura 35: Red de contacto del tetrámero sin cofactor y la proteína coloreada por las comunidades.

Una vez que se calculó la correlación se hace un análisis de los mapas de calor de las correlaciones de las dinámicas.

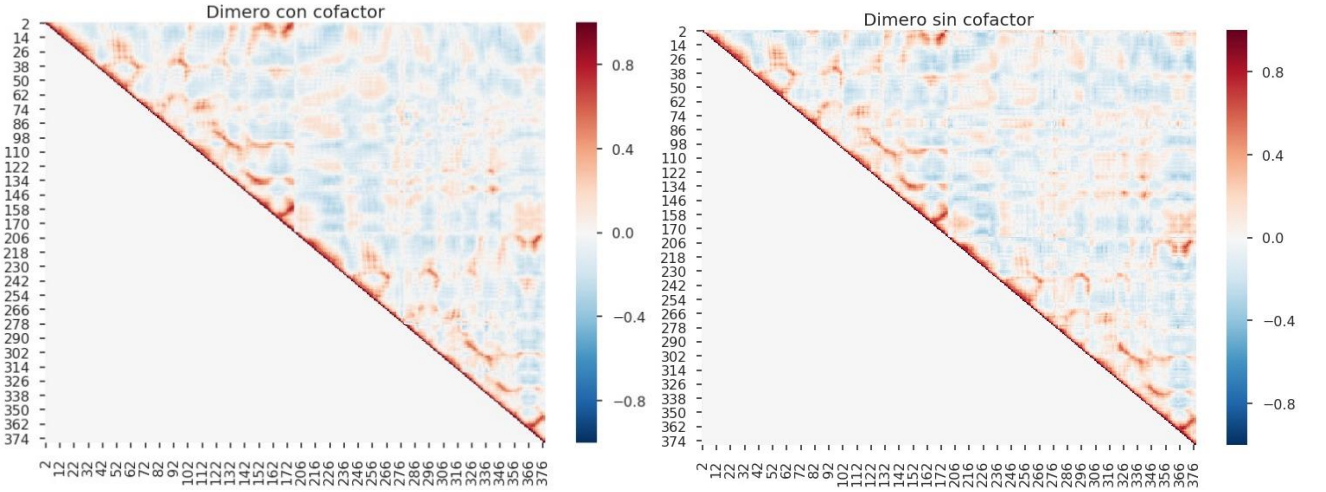


Figura 36: Mapa de correlación de los dímeros con y sin cofactor.

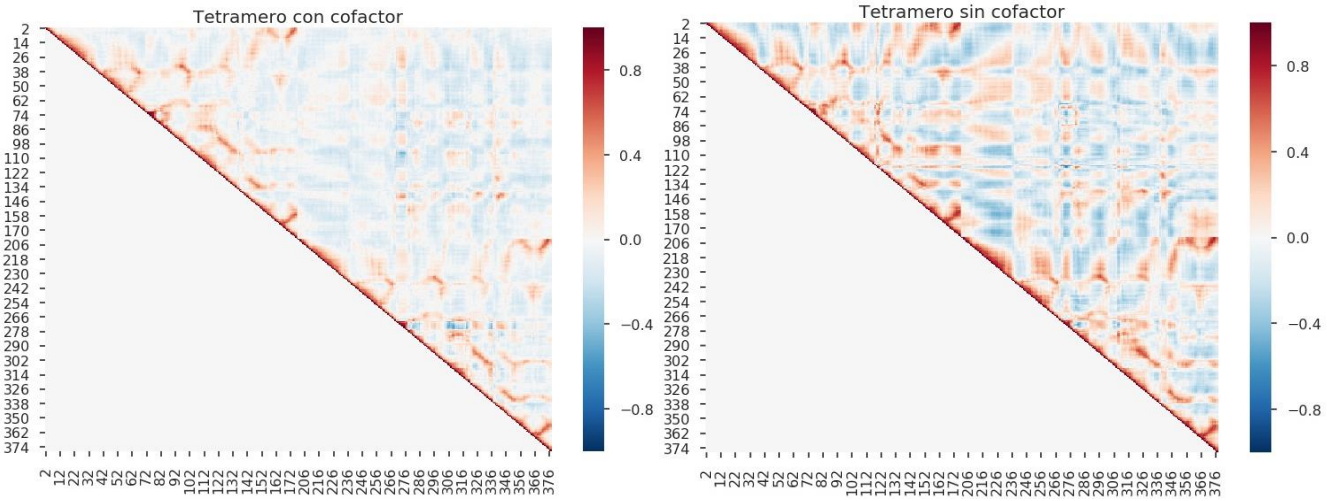


Figura 37: Mapa de calor de la correlación de los tetrámeros con y sin cofactor.

En el mapa de calor del tetrámero (Figura 37) se nota claramente que la correlación es mayor sin cofactor que con él, dado que es el estado natural de la proteína, por lo que podemos notar que, donde se obtiene una correlación positiva, disminuyo el valor a no estar correlacionados. Por otro lado, se observa que hay residuos correlacionados con o sin cofactor, permitiendo filtrar los residuos del análisis y centrar la atención en aquellos residuos que tuvieron un cambio de comportamiento notorio para la posible detección de caminos

alostéricos, como se observa en la franja de los residuos 176 al 226 en la Figura 37 y el cambio en la correlación con los residuos de la cadena A.

Al generar las redes de correlación dinámica se observó que los residuos con mayor intermediación cambian con respecto a las redes de contacto, esto es debido a que se pueden generar mayor número de enlaces, dado que al menos obtendría el número de enlaces de la red de contacto por construcción y se agregan aquellos enlaces que no se encontraron en la primera red. Es posible que los nuevos residuos con mayor intermediación puedan ser de igual o mayor relevancia para la identificación de caminos alostéricos.

Se considera que al construir la red de correlación cruzada se tienen algunas desventajas que no describen adecuadamente a las interacciones del sistema. Una de ellas es, si el producto punto entre ambos vectores de coordenadas son perpendiculares, da una correlación de 0, a pesar, de estar perfectamente correlacionados. Otra limitante asume qué al promediar, las correlaciones son lineales durante el tiempo, es decir, dos residuos pueden estar perfectamente correlacionados si están en la misma fase del movimiento, pero si uno de ellos está desfasado obtendrán una no correlación, perdiendo así el enlace. Por último, el coeficiente de correlación solo nos muestra relaciones lineales, por lo que si estuviesen correlacionados no-linealmente este coeficiente no obtiene estas relaciones. Una manera de encontrar las relaciones no lineales es generando variables en las matrices de correlación aplicando una función, por ejemplo, una cuadrática o logarítmica y realizar el mismo análisis.

Los análisis generados por medio de las RCD en la presente tesis pueden no tener la resolución temporal suficiente, dado que al generar las dinámicas de las proteínas y para que se cumpla la hipótesis ergódica, habría de simular la dinámica el suficiente tiempo para encontrar la mayor cantidad de posibles estados del sistema, mejorando la robustez estadística. A su vez, al iniciar la dinámica presenta un factor de aleatoriedad debido a que se establecen velocidades y momentos aleatorios al inicio de la simulación. La manera de solucionar estos experimentos es simulando por mayor tiempo las dinámicas de la proteína o repitiendo el análisis un mayor número de veces y promediando los resultados del análisis.

Capítulo 5 Conclusión

Como se expuso en los resultados la implementación del software Mani para generar alineamientos entre proteínas es similar al software de Click, obteniendo resultados adecuados en los diversos experimentos. CLICK se encuentra entre los mejores programas de alineamiento [27], al tener una réplica comparable con dicho software presenta una gran ventaja para los fines de la investigación, con el beneficio de tener un software editable y con documentación.

Una principal ventaja del software es ser programado en Python y alojado en un repositorio público, donde cualquier persona pueda colaborar mejorando y/o implementando el código con fines de investigación en diversos casos de estudio.

Un primer inconveniente al generar Mani es el tiempo de ejecución[18]. Si los filtros del alineamiento no son lo suficientemente estrictos, el número de comparaciones asciende a 210 millones. Para poder hacer las comparaciones con una mayor velocidad se tendría que hacer un "speed-up" del código, dado que la velocidad de compilación en C es menor que en Python por ser un lenguaje de bajo nivel [28], se sugiere que se implemente en Cython (lenguaje de programación basado en C y Python). De igual manera, el código aún puede mejorarse al implementar funciones de multiproceso y eliminar operaciones redundantes.

El estudio de las redes de correlación dinámica nos abre un campo de posibilidades al encontrar métricas que nos permiten abordar de una manera distinta la identificación de los caminos alostéricos. Si se calculan métricas de centralidad como la cercanía, el grado, el grado de clustering y se implementan otros algoritmos como el Page Rank para obtener mayor información de la red, se robustece el análisis. Hay que tomar en cuenta que para redes muy grandes con un número de nodos

[18] Pag 52.

mayor a 10,000 el método de Louvain se vuelve inaplicable dado que la optimización de la modularidad es un problema NP-completo, aunque existen algoritmos similares para su implementación con un límite en la resolución de comunidades.

La implementación de esta metodología es en general para cualquier proteína, por lo que se puede hacer el alineamiento y el análisis de las redes de correlación dinámica para la proteína que sea del interés del investigador. Por otro lado, la metodología para encontrar los caminos alostéricos nos acerca a residuos de interés, filtrando residuos donde no cambia la dinámica, y en este caso de estudio describiendo los residuos cuando tienen o no cofactor y analizando si presentan un cambio en la RCRR o en la RCD.

Al ser una metodología novedosa la única manera de demostrar si las hipótesis generadas durante el análisis son las adecuadas, como la obtención de caminos alostéricos, es mutando los residuos de dicho camino. Si al realizar la simulación de dinámica molecular y analizar los cambios en la topología de la red se obtienen alteraciones en los caminos propuestos de comunicación alostérica, entonces se tendría cierta confianza en la evaluación de dichas mutaciones a nivel experimental.

Apéndice A

Pasos por seguir para el cálculo del alineamiento por medio de quaterniones para obtener la rotación y la traslación óptima.

Se selecciona un 3-clique de la proteína A y otro 3-clique de la proteína B, llamados A_i y B_j, el clique A_i se rotará y se trasladará para alinearse con el clique B_j. Por medio de las coordenadas de los elementos del 3-clique elegido se calculan las coordenadas del baricentro de cada clique de la siguiente manera:

$$\bar{x} = \left(\frac{1}{n}\sum_{i=0}^{n} x_i , \frac{1}{n}\sum_{i=0}^{n} y_i , \frac{1}{n}\sum_{i=0}^{n} z_i \right) \qquad (A.1)$$

Con *n* el número de elementos del clique y *(x, y, z)* las correspondientes coordenadas del elemento del clique.

Utilizando el baricentro de cada clique, se calculó los vectores céntricos que se definen como:

$$\hat{x} = x_k - \bar{x} \qquad (A.2)$$

Con $x_k = (x_i, y_i, z_i)$ las coordenadas del elemento *k* del clique seleccionado y $\bar{x}$ las coordenadas del baricentro del clique, Obteniendo un vector $\hat{x}$ centrado en el origen.

Una vez hecho el calculó se aplicó el filtro de distancia mínima para generar los candidatos a alinearse, no considerado en la metodología propuesta en Nguyen *et al. [1]*, de manera que se calcula la distancia euclidiana promedio del origen $(0,0,0)$ a los vectores céntricos $\hat{x}$ de cada clique, posteriormente se hace la diferencia de distancias promedio entre cada clique A_i con el clique B_j y se mantienen los candidatos que tengan una diferencia de distancia menor o igual al límite de distancia mínima que depende del número de elementos del clique *(n)* y definida por la siguiente tabla:

Número de elementos en clique (n)	Limité de diferencia de distancia promedio mínima
3	0.45
4	0.9
5	1.8
6	3.6
7	4.5
8	8.0

Tabla 7: Valores para filtro de distancia promedio mínima que depende del número de elementos del clique.

Ahora se procederá al cálculo de la matriz de rotación por medio de los quaterniones, que resuelve el siguiente problema; dado dos conjuntos de vectores y_k (objetivo) y x_k (modelo), encuentra una transformación ortogonal μ y una traslación *r* tal que el residuo *E* se minimice, ponderado por un peso w_k, para la presente tesis $w_k = 1$, de manera que *[21]*:

$$E \coloneqq \frac{1}{N}\sum_{k=1}^{N} |\mu(x_k) + r - y_k|^2 \qquad (A.3)$$

Para obtener la matriz de rotación primero se obtienen los vectores céntricos definidos en *(A.2)*, se selecciona el máximo eigenvalor y los correspondientes eigenvectores de la matriz de simetría *F* dada por la siguiente ecuación *[29]*:

$$F = \begin{pmatrix} R_{11}+R_{22}+R_{33} & R_{23}-R_{32} & R_{31}-R_{13} & R_{12}-R_{21} \\ R_{23}-R_{32} & R_{11}-R_{22}-R_{33} & R_{12}+R_{21} & R_{13}+R_{31} \\ R_{31}-R_{13} & R_{12}+R_{21} & -R_{11}+R_{22}-R_{33} & R_{23}+R_{32} \\ R_{12}-R_{21} & R_{13}+R_{31} & R_{23}-R_{32} & -R_{11}-R_{22}+R_{33} \end{pmatrix} \qquad (A.4)$$

Donde los R_{ij} están en términos de los vectores céntricos $\hat{x}$ por medio de:

$$R_{ij} = \sum_{k=1}^{N} \hat{x}_{ik}\hat{y}_{jk}\,, i, j = 1,2,3 \qquad (A.5)$$

i, j corresponde a las coordenadas en *x, y, z* y *k* el vector que se utiliza para generar la matriz de rotación.

Al obtener el máximo eigenvalor q_1 y sus correspondientes eigenvectores (q_2, q_3, q_4) se genera el quaternion, donde los quaterniones son vectores de cuatro elementos que consisten en un vector de dimensiones 3×1 y una componente escalar.

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} q \\ q_4 \end{bmatrix} \qquad (A.6)$$

De manera que la matriz de rotación μ se define de la siguiente manera:

$$\mu = \begin{pmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2q_3 - q_1q_4) & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_3q_4 - q_1q_2) \\ 2(q_2q_4 - q_1q_3) & 2(q_3q_4 - q_1q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{pmatrix} \qquad (A.7)$$

La matriz de rotación se aplica a los vectores céntricos y se suma el baricentro del clique a comparar $\tilde{x_i} = \mu(\hat{x_i}) + \bar{x_j}$, a estos nuevos vectores $\tilde{x_i}$ del clique se calcula el RMSD por medio de *(3.1.1)*.

Apéndice B

Código principal utilizado en este trabajo

El código se parte en dos scripts, funciones_click.py que contiene las funciones que se utilizan en general para hacer los alineamientos y click_align.py que corresponde al script que ejecuta dichas funciones siguiendo la metodología anteriormente explicada para generar el alineamiento entre proteínas.

funciones_click.py

```
# coding: utf-8

# libreria de analisis de datos y su facil lectura.
import pandas as pd
# libreria de generacion de redes y cliques
import networkx as nx, community

# libreria de calculo de distancia euclidiana
from scipy.spatial.distance import pdist, euclidean

# libreria de calculos de algebra lineal
import numpy as np

# libreria de iteraciones
import itertools as it

# Libreria de MA para RMSD
import sys
sys.path.append('math_tricks/')
import math_vect_tools as mymath

# librería para verificar orden de pdbs
import copy
# libreria para calculo de RMSD
import math


def verify_file_order(pdb1, pdb2, pdb11, pdb22):
    """
   Verifica que archivo es mas grande y lo cambia para que siempre alinea
   el pdb mas pequenio al otro mas grande.
   :param pdb1: object PdbStruct
   :param pdb2: object PdbStruct
   :param pdb11: list of residues
   :param pdb22: list of residues
   :return: pdb's and lists
   """
    if len(pdb22) < len(pdb11):
```

```
            pdb1_temp = copy.copy(pdb1)
            pdb2_temp = copy.copy(pdb2)

            pdb11_temp = copy.copy(pdb11)
            pdb22_temp = copy.copy(pdb22)

            pdb1 = pdb2_temp
            pdb2 = pdb1_temp

            pdb11 = pdb22_temp
            pdb22 = pdb11_temp

            del [pdb1_temp]
            del [pdb2_temp]
            del [pdb11_temp]
            del [pdb22_temp]

            print("Intercambio de nombre ya que la proteina 1 es mas grande que la
2")
            print(pdb1.name, len(pdb11))
            print(pdb2.name, len(pdb22))
            print("No te preocupes ya quedo :) pero te recomendamos que cambies el
orden de los archivos en el alineamiento")

        return pdb1, pdb2, pdb11, pdb22


    def get_df_distancias(ref):
        """Funcion para obtener los enlaces de distancias de cada residuo"""
        # se generan listas con coordenadas y numero de atomo
        # calcula distancia y regresa una lista de enlaces
        enlaces = []
        for res1 in ref[1:-1]:
            for res2 in ref[1:-1]:
                if res2.resx >= res1.resx:
                    if mymath.distance(res2.GetAtom('CA').coord,
res1.GetAtom('CA').coord) < 10:
                        enlaces.append([res1.resx, res2.resx])

        # se genera la matriz de adyacencias para la red
        return enlaces


    def eval_dihedral(ang_ref, ang_tar, cutoff=30):
        """
       Evaluacion de los angulo diedro, manteniendo aquellos que presentan un
cierto cutoff.
       :param ang_ref: angulo phi o psi
       :param ang_tar: angulo phi o psi
       :param cutoff: valor de corte para filtro
       :return: flag (1 o 0)
       """
        if ang_ref * ang_tar > 0:
            if abs(ang_ref - ang_tar) < cutoff:
                return 1
        elif ang_ref < 0:
            ang_ref = 360 - ang_ref
        elif ang_tar < 0:
            ang_tar = 360 - ang_tar
```

```
        if abs(ang_ref - ang_tar) < cutoff:
                return 1

        return 0


    # Filtro score Estructura Secundaria
    def score_ss(clq1, clq2):
        """
       Evaluacion del score de estructura.
       :param clq1: pareja clique de proteina A
       :param clq2: pareja clique de proteina B
       :return: flag [0,1] si cumple o no.
       """

        flag = 1
        for k in range(3):
            res1 = clq1[k]
            res2 = clq2[k]
            if SSM(res1.ss, res2.ss) == 2:
                flag = 0
                break

        return flag


    # Rotacion y traslacion
    def matrix_R(vecs_c_1, vecs_c_2):
        """
       Se genera matriz de rotacio utilizando las coordenadas de la misma dimension
de los atomos de interes
       :param vecs_c_1: coordenadas del vector del residuo y clique
       :param vecs_c_2: coordenadas del vector del residuo y clique
       :return: matriz de rotacion con las parejas de cliques seleccionados
       """

        number_of_atoms = vecs_c_1.shape[0]

        def R_ij(i, j):
            """
           Funcion que calcula el valor de R_ij basado en Using quaternions to
calculate RMSD
           :float i  Coordenada (0=x,y=1,z=2):
           :float j Coordenada (0=x,y=1,z=2):
           :return : valor R_ij
           """
            valor = sum([vecs_c_1[k, i] * vecs_c_2[k, j] for k in
range(number_of_atoms)])
            return valor

        """cliques a comparar: i,j
       desde aqui se itera sobre cada i y hay que variar los vectores
       coordenada
       Regresa la matriz gigante
       (matriz simetrica del articulo Using quaternions to calculate RMSD!!!!)"""
        # primer renglon
        R11R22R33 = (R_ij(0, 0) + R_ij(1, 1) + R_ij(2, 2))
        R23_R32 = (R_ij(1, 2) - R_ij(2, 1))
```

```
        R31_R13 = (R_ij(2, 0) - R_ij(0, 2))
        R12_R21 = (R_ij(0, 1) - R_ij(1, 0))
        # segundo renglon
        R11_R22_R33 = (R_ij(0, 0) - R_ij(1, 1) - R_ij(2, 2))
        R12R21 = (R_ij(0, 1) + R_ij(1, 0))
        R13R31 = (R_ij(0, 2) + R_ij(2, 0))
        # tercer renglon
        _R11R22_R33 = (-R_ij(0, 0) + R_ij(1, 1) - R_ij(2, 2))
        R23R32 = (R_ij(1, 2) + R_ij(2, 1))
        # cuarto renglon
        _R11_R22R33 = (-R_ij(0, 0) - R_ij(1, 1) + R_ij(2, 2))

        matriz_R = [
            [R11R22R33, R23_R32, R31_R13, R12_R21],
            [R23_R32, R11_R22_R33, R12R21, R13R31],
            [R31_R13, R12R21, _R11R22_R33, R23R32],
            [R12_R21, R13R31, R23R32, _R11_R22R33]
        ]
        return (np.array(matriz_R))


    # Metodoo de quaterniones
    def rotation_matrix(matriz_R):
        """utilizando la funcion giant_matrix, fijando los valores de i,j
       se calcula la matriz de rotacion con los eigenvectores y eigenvalores
       arroja una matriz de rotacion que depende de la matriz gigante
       """
        eignvalues, eigenvectors = np.linalg.eig(matriz_R)
        q = eigenvectors[:, np.argmax(eignvalues)]

        # matriz de rotacion con eigenvectores forma USING QUATERNIONS TO CALCULATE
RMSD
        q0, q1, q2, q3 = q[0], q[1], q[2], q[3]
        matriz_rotacion = np.array([
            [(q0 ** 2 + q1 ** 2 - q2 ** 2 - q3 ** 2), 2 * (q1 * q2 - q0 * q3), 2 *
(q1 * q3 + q0 * q2)],
            [2 * (q1 * q2 + q0 * q3), (q0 ** 2 - q1 ** 2 + q2 ** 2 - q3 ** 2), 2 *
(q2 * q3 - q0 * q1)],
            [2 * (q1 * q3 - q0 * q2), 2 * (q2 * q3 + q0 * q1), (q0 ** 2 - q1 ** 2 -
q2 ** 2 + q3 ** 2)]
        ], dtype=np.float64)

        return (matriz_rotacion)


    # aplicacion de matriz de rotacion
    def rotation_vectors(vector_gorro, matriz_rotacion):
        """obtencion de vector rotado,
       utilizando la matriz de rotacion
       y los vectores gorro a rotar y trasladar"""
        coord_rotado = np.array([np.matmul(matriz_rotacion, coord_atom) for
coord_atom in vector_gorro])

        return coord_rotado


    # calculo de RMSD
    def rmsd_between_cliques(clique_trasladado_rotado, atom_to_compare):
        """Calculo de rmsd entre cliques tomando el atomo rotado y trasladado
```

```
    y el atomo a comparar"""

    dim_coord = clique_trasladado_rotado.shape[1]
    N = clique_trasladado_rotado.shape[0]

    result = 0.0
    for v, w in zip(atom_to_compare, clique_trasladado_rotado):
        result += sum([(v[i] - w[i]) ** 2.0 for i in range(dim_coord)])

    rmsd_final = np.sqrt(result / N)

    return (rmsd_final)


# CHECK DE ALINEAMINETO
def align(c_1, c_2, number_elements_clique = None):

    bari_1 = c_1.mean(0)
    bari_2 = c_2.mean(0)

    vecs_center_1 = c_1 - bari_1
    vecs_center_2 = c_2 - bari_2

    # Filtro de distancia minima.
    # if number_elements_clique == 3:
    #     limite_distancia_minima = 0.13
    # elif number_elements_clique == 4:
    #     limite_distancia_minima = 0.15
    # elif number_elements_clique == 5:
    #     limite_distancia_minima = 0.60
    # elif number_elements_clique == 6:
    #     limite_distancia_minima = 0.60
    # elif number_elements_clique == 7:
    #     limite_distancia_minima = 0.60
    # else:
    #     limite_distancia_minima = 100
    #
    # def minimum_distance():
    #
    #     flag = 0
    #     origin = (0, 0, 0)
    #     dist_1 = np.mean([euclidean(origin, j) for j in vecs_center_1])
    #     dist_2 = np.mean([euclidean(origin, j) for j in vecs_center_2])
    #     if abs(dist_1 - dist_2) > limite_distancia_minima:
    #         flag = 1
    #
    #     return flag
    #
    # if minimum_distance():
    #     rmsd_final = 100  # rmsd muy grande
    #     return rmsd_final

    matriz_R = matrix_R(vecs_center_1, vecs_center_2)
    matriz_rotacion = rotation_matrix(matriz_R)

    vector_rotado = rotation_vectors(vecs_center_1, matriz_rotacion)

    protein_trasladado_rotado = vector_rotado + bari_2
```

```
        protein_to_compare = c_2

        # te puedes ahorrar el paso de trasladar si calculas en los vectores
centricos.
        rmsd_final = rmsd_between_cliques(protein_trasladado_rotado,
protein_to_compare)

        if number_elements_clique == 7:
            return rmsd_final, matriz_rotacion

        return rmsd_final


    def gen_cliques(red, k=7):
        """
       Generacion de cliques dada una red, donde po default busca 7-cliques
       :param red: Graph de networkx
       :param k: numero de elementos en el clique
       :return: lista de cliques unicos
       """

        cliques_completos = [clq for clq in nx.find_cliques(red) if len(clq) >= k]
# mayor o igual que cambiarlo
        print('numero de cliques maximos encontrados:', len(cliques_completos))

        lista_cliques = []
        for v in (cliques_completos):
            a = list(it.combinations(v, 3))
            for j in a:
                permutation_temp = it.permutations(j)
                lista_cliques.append(set(permutation_temp))

        lista_cliques = np.unique(lista_cliques)

        return lista_cliques, cliques_completos


    def gen_cliques_3(red):
        """
     Generacion de 3 cliques dado los cliques maximales
       :param red: enlaces
       :return: lista de 3-cliques
       """

        combinations = list(it.combinations(red, 3))
        lista_cliques = [list(it.permutations(nclique)) for nclique in
combinations]
        lista_cliques1 = [list(clq) for combinations in lista_cliques for clq in
combinations]

        return lista_cliques1


    def create_ss_table(list_residues):
        """
     Se genera tabla con los valores de estructura secundaria, numero de resiudo y
cadena
       :param list_residues:
```

```
        :return: DataFrame con valores de estructura secundaria, numero de residuo y
cadena
        """
        ss_list = [i.ss for i in list_residues]
        num_resi_list = [i.resx for i in list_residues]
        chain_list = [i.chain for i in list_residues]

        ss = pd.DataFrame()
        ss['structure'] = ss_list
        ss['residue_number'] = num_resi_list
        ss['chain'] = chain_list
        return ss


    def add_element_to_clique(cliques_candidatos,
                              cliques_maximales_1,
                              cliques_maximales_2):
        """
        Agrega un residuo del clique maximal si el clique completo es un subconjunto
del clique maximal
        :param cliques_candidatos: lista de cliques canidatos
        :param cliques_maximales_1: lista de cliques maximales proteina A
        :param cliques_maximales_2: lista de cliques maximales proteina B
        :return: Lista de cliques nuevos
        """
        cliques_nuevos = []
        # agregando elemento
        for clique in cliques_candidatos:
            cliques1_nuevos = []
            cliques2_nuevos = []
            for i in cliques_maximales_1:

                if set(clique[0]).issubset(i):
                    no_estan_en_clique = set(i).difference(set(clique[0]))
                    for nuevo_elemento in no_estan_en_clique:
                        clique_nuevo1 = list(clique[0]).copy()
                        clique_nuevo1 = np.append(clique_nuevo1, nuevo_elemento)
                        if clique_nuevo1.tolist() not in cliques1_nuevos and
len(clique_nuevo1) > 1:
                            cliques1_nuevos.append(clique_nuevo1.tolist())

            for j in cliques_maximales_2:
                if set(clique[1]).issubset(j):
                    no_estan_en_clique = set(j).difference(set(clique[1]))
                    for nuevo_elemento in no_estan_en_clique:
                        clique_nuevo2 = list(clique[1]).copy()
                        clique_nuevo2 = np.append(clique_nuevo2, nuevo_elemento)
                        if clique_nuevo2.tolist() not in cliques2_nuevos and
len(clique_nuevo2) > 1:
                            cliques2_nuevos.append(clique_nuevo2.tolist())

            if cliques1_nuevos == [] or cliques2_nuevos == []:
                continue
            else:
                cliques_nuevos.append([cliques1_nuevos, cliques2_nuevos])

        return cliques_nuevos
```

```
    def SSM(ss1, ss2):

        """Catalogo SSM siguiendo la tabla 1 y con una funcion extra,
       ss1: string (H,B,C)
       ss2: string (H,B,C)
       devuelve el score: int (0,1,2)"""
        def get_score_from_table(ss1, ss2):

            if (ss1 == 'H') and (ss2 == 'B'):
                score_ss = 2
            elif (ss1 == 'B') and (ss2 == 'H'):
                score_ss = 2
            else:
                print(ss1.ss2)
                print('No se cumplieron los casos, revisar')

            return(score_ss)

        score_ss = 123

        if ss1 == ss2:
            score_ss = 0
        elif (ss1 != ss2) & ((ss1 == 'C') | (ss2 == 'C')):
            score_ss = 1
        else:
            score_ss = get_score_from_table(ss1, ss2)

        return(score_ss)


    def fun_resiudos_match(protein_trasladado_rotado, protein_to_compare, res_1,
res_2):
        """
       genera los Match de parejas de residuos que cumplen un treshold de distancia
(RMSD)
       :param protein_trasladado_rotado:
       :param protein_to_compare:
       :param res_1:
       :param res_2:
       :return: Lista ordenada de parejas de residuos y su distancia
       """
        return sorted([[math.sqrt(sum((c_2 - c_1) ** 2)), (res1.resx, res2.resx)]
for c_1, res1 in zip(
            protein_trasladado_rotado, res_1) for c_2, res2 in zip(
            protein_to_compare, res_2) if math.sqrt(sum((c_2 - c_1) ** 2)) < 3.5])


    def filter_pairs(residuos_match, flag=None):
        """
       Filtra las parejas de residuos de dos maneras si flag = False or None
       solo toma parejas que no esten repetidas, si flag = True toma todas las
parejas
       y solo por orden las filtra
       :param residuos_match:
       :param flag:
       :return: distancia y parejas seleccionadas.
       """
        if flag:
            pairs_1 = [c[1][0] for c in residuos_match]
```

```
            repeat_1 = [i for i in pairs_1 if pairs_1.count(i) > 2]

            pairs_2 = [c[1][1] for c in residuos_match]
            repeat_2 = [i for i in pairs_2 if pairs_2.count(i) > 2]
        else:
            repeat_1, repeat_2 = [], []

        c1 = []
        c2 = []
        cand_n = []
        for i in residuos_match:
            if (i[1][0] in c1) or (i[1][1] in c2) or (i[1][0] in repeat_1) or
(i[1][1] in repeat_2):
                continue
            else:
                c1.append(i[1][0])
                c2.append(i[1][1])
                cand_n.append(i)

        return(cand_n)


    def filter_candidates(pc, flag=True):
        """
     Se filtran candidatos para no tener operaciones redundantes
       :param pc: lista de candidatos parejas de cliques
       :param flag: si es la primera o la ultima iteracion
       :return: lista reducida de parejas candidatas
       """
        if flag:
            pc = np.array([i[0] for i in pc])
        else:
            pc = np.array(pc)

        lista = []
        for i in pc:
            lista.append([i[0], i[1]])

        idx = []
        lista_partenerts = []
        for id, i in enumerate(lista):
            a = i[0]
            b = i[1]
            list_2 = [i for i in zip(a, b)]
            sort_list = sorted(list_2)
            if sort_list not in lista_partenerts:
                lista_partenerts.append(sort_list)
                idx.append(id)

        print('parejas_iniciales', len(pc))
        pc = pc[idx]
        print('comparaciones', len(pc))
        return pc
```

click_align.py

```
#!/bin/sh

# librerias que utilizaras
import numpy as np
# por si no te lee las tools o functions creadas
import sys
sys.path.append("/home/serch/Serch/math_tricks/")
import math_vect_tools as mymath
# herramientas para leer pdbs
import read_pdb_tools as rpt
# funciones de click generadas en pandas
import funciones_CLICK as fc
# cuenta tiempo de ejecucion
import datetime
time_all = datetime.datetime.now()
# networks
import networkx as nx
# formato resultados
import pandas as pd
# check si ya esta resultado
import glob

# lectura de archivo
file1 = sys.argv[1]
file2 = sys.argv[2]
global_cutoff = 3  # float(sys.argv[3])

directory = '/home/serch/Serch/Experimentos/pdbs_frames/'

# por si ya hiciste este experimento termina script.
try:
      fh=open(directory+file1.split("/")[-
1].split(".pdb")[0]+"_"+file2.split("/")[-1].split(".pdb")[0]+'_mani.csv')
      print("ya existe el archivo",directory+file1.split("/")[-
1].split(".pdb")[0]+"_"+file2.split("/")[-1].split(".pdb")[0]+'_mani.csv')
      exit()
except FileNotFoundError:
      pass

# cadena de interes
chain1 ='A' # file1[-10:-4].split("_")[1]
chain2 ='A' # file2[-10:-4].split("_")[1]

# numero de cliques, preguntar en el software para generalizarlo o INPUT...
number_elements_clique = 3 #int(input("Cuantos cliques buscaras?"))

# se define la estructura
pdb1 = rpt.PdbStruct(file1)
pdb2 = rpt.PdbStruct(file2)

# se lee el pdb y se agrega al objeto
pdb1.AddPdbData("%s" % file1)
pdb2.AddPdbData("%s" % file2)

#filtro SS
```

```
pdb1.Set_SS()
pdb2.Set_SS()

# filtro Angulo diedro
pdb1.SetDiheMain(chain_n=chain1)
pdb2.SetDiheMain(chain_n=chain2)

# se obtienen los residuos que perteneces a la cadena de interes por default chain
= 'A'
pdb11 = pdb1.GetResChain(chain=chain1)
pdb22 = pdb2.GetResChain(chain=chain2)

#  check de tamaño de proteina, dado que puede tardar mucho el alineamiento
if len(pdb11) > 500 or len(pdb22) > 500:
    print('El numero de residuos es alto, puede tardar el alineamiento.')
    print(file1, file2)
    pdb_file = open(directory+file1[-10:-4]+'_'+file2[-10:-4]+".delete", "w")
    pdb_file.write("Proteinas muy grandes, muchos residuos")
    pdb_file.close()
    exit()

#verifica tamanio
pdb1, pdb2, pdb11, pdb22 = fc.verify_file_order(pdb1, pdb2, pdb11, pdb22)

# creando tabla de estructura secundaria para filtro de SS
ss1 = fc.create_ss_table(pdb11)
ss2 = fc.create_ss_table(pdb22)

# generacion de enlaces
enlaces1 = (fc.get_df_distancias(pdb11))
enlaces2 = (fc.get_df_distancias(pdb22))

red1 = (nx.Graph(enlaces1))
red2 = (nx.Graph(enlaces2))

cliques_1, cliques_max_1 = fc.gen_cliques(red1, k=7)
cliques_2, cliques_max_2 = fc.gen_cliques(red2, k=7)

lenght_cliquemax_1 = len(cliques_max_1)
lenght_cliquemax_2 = len(cliques_max_2)

print("numero de cliques maximales combinaciones", lenght_cliquemax_1 *
lenght_cliquemax_2)

####################################
# Alineamiento por posicion del pdb#
####################################
list_candidates = []
for clique1 in cliques_max_1:
    res_clq_1 = [pdb1.GetResIdx(j) for j in clique1]
    for clique2 in cliques_max_2:
        res_clq_2 = [pdb2.GetResIdx(j) for j in clique2]

        # Filtro PHI PSI
        val_vec = []
        for res1 in res_clq_1:
            phi_ref = res1.phi
            psi_ref = res1.psi
            val = 0
```

```
            for res2 in res_clq_2:
                phi_tar = res2.phi
                psi_tar = res2.psi
                if fc.eval_dihedral(phi_ref, phi_tar, cutoff=global_cutoff) and (
                        fc.eval_dihedral(psi_ref, psi_tar, cutoff=global_cutoff)):
                    val = val + 1
            val_vec.append(val)
        if val_vec.count(0) < 1:
            # debugg de vector de angulos diedros
            list_candidates.append([clique1, clique2])

# # numero de candidatos y parejas generadas.
print("numero de candidatos despues de filtro ángulos diedro",
len(list_candidates))
print(list_candidates[:3])

# Si tiene un gran numero de candidatos es posible que no termine pronto, por eso
se detiene
if len(list_candidates) > 3000:
    print('no se filtraron los suficientes cambia el cutoff!')
    print(file1, file2)
    pdb_file = open(directory+file1[-10:-4]+'_'+file2[-10:-4]+".delete", "w")
    pdb_file.write("bajar_cutoff_angulo diedro")
    pdb_file.close()
    exit()
# Si no encuentra candidatos se detiene el programa
if len(list_candidates) == 0:
    print('no hubo candidatos cambia filtro angulo diedro')
    exit()

# GENERACION DE CLIQUES DE LA LISTA DE CLIQUES MAXIMALES APLICANDO FILTRO ANGULOS
DIEDROS
cliques_1_temp = []
for clique1 in list_candidates:
    list_candidate = fc.gen_cliques_3(clique1[0])
    if list_candidate not in cliques_1_temp:
        cliques_1_temp.append(list_candidate)

cliques_2_temp = []
for clique2 in list_candidates:
    list_candidate = fc.gen_cliques_3(clique2[1])
    if list_candidate not in cliques_2_temp:
        cliques_2_temp.append(list_candidate)


def iter_align(number_elements_clique, cliques_1_align, cliques_2_align):

    """
Iteraciones de alineamiento para encontrar la mejor pareja de cliques que alinean a
las proteinas.
    :int number_elements_clique: elementos del clique
    :tuple cliques_1_align: lista de cliques a alinear de la proteina A
    :tuple cliques_2_align: lista de cliques a alinear de la proteina B
    :return: cliques_candidate lista de cliques candidatos
    """
# Tabla de filtro RMSD de parejas clique
    restriccion_rmsd = 0.15
    if number_elements_clique == 4:
        restriccion_rmsd = 0.30
```

```
    if number_elements_clique == 5:
        restriccion_rmsd = 0.60
    if number_elements_clique == 6:
        restriccion_rmsd = 0.90
    if number_elements_clique == 7:
        restriccion_rmsd = 1.50
    if number_elements_clique == 8:
        restriccion_rmsd = 1.80

    cliques_candidate = []

    if number_elements_clique == 3:
        for pareja in zip(cliques_1_align, cliques_2_align):
            for clique1 in pareja[0]:
                res_clq_1 = [pdb1.GetResIdx(clq) for clq in clique1]
                for clique2 in pareja[1]:
                    res_clq_2 = [pdb2.GetResIdx(clq) for clq in clique2]
                    if fc.score_ss(res_clq_1, res_clq_2):
                        coord_1 = np.array([res.GetAtom('CA').coord for res in
res_clq_1])
                        coord_2 = np.array([res.GetAtom('CA').coord for res in
res_clq_2])
                        if fc.align(coord_1, coord_2) < restriccion_rmsd:
                            cliques_candidate.append([clique1, clique2])

    else:
        for clique1 in cliques_1_align:
            res_clq_1 = [pdb1.GetResIdx(clq) for clq in clique1]
            for clique2 in cliques_2_align:
                res_clq_2 = [pdb2.GetResIdx(clq) for clq in clique2]
                if fc.score_ss(res_clq_1, res_clq_2):
                    coord_1 = np.array([res.GetAtom('CA').coord for res in
res_clq_1])
                    coord_2 = np.array([res.GetAtom('CA').coord for res in
res_clq_2])
                    if number_elements_clique == 7:
                        rmsd, mat_rot = fc.align(coord_1, coord_2,
number_elements_clique=number_elements_clique)
                        if rmsd < restriccion_rmsd:
                            cliques_candidate.append([clique1, clique2, mat_rot])

                    else:
                        if fc.align(coord_1, coord_2) < restriccion_rmsd:
                            cliques_candidate.append([clique1, clique2])

    return cliques_candidate


print('================ alineamiento de 3-clique y agrego el 4 elemento
==================')

new_df_cliques1 = cliques_1_temp
new_df_cliques2 = cliques_2_temp

# emparejamiento de cliques
cliques_candidatos = iter_align(number_elements_clique, new_df_cliques1,
new_df_cliques2)

# filtro de candidatos repetidos
```

```
cliques_candidatos = fc.filter_candidates(cliques_candidatos, flag=False)

# se agrega un elemento a cada pareja de cliques
new_df_cliques = fc.add_element_to_clique(cliques_candidatos, cliques_max_1,
cliques_max_2)

number_elements_clique = number_elements_clique + 1

print("candidatos con n-cliques, n =", number_elements_clique-1,
      "numero de parejas", len(cliques_candidatos))

print(number_elements_clique, len(new_df_cliques))

print('==================ya acabo con 3-cliques va con 4=========================')
# refiltro
# new_df_cliques = fc.filter_candidates(new_df_cliques, flag=False)

cliques_temp = []
cliques_temp_add = []

for parejas_4clique in new_df_cliques:

    new_df_cliques1 = parejas_4clique[0]
    new_df_cliques2 = parejas_4clique[1]

    cliques_candidatos = iter_align(number_elements_clique, new_df_cliques1,
new_df_cliques2)

    if cliques_candidatos != []:
        cliques_temp.append(cliques_candidatos)
        cliques_temp_add.append(fc.add_element_to_clique(cliques_candidatos,
cliques_max_1, cliques_max_2))

number_elements_clique = number_elements_clique + 1
print(cliques_temp[:1])
print(len(cliques_temp))

print(cliques_temp_add[:1])
print(len(cliques_temp_add))

print('==================ya acabo con 4-cliques va con 5=========================')

cliques_temp_add_0 = [y for x in cliques_temp_add for y in x]

# cliques_temp_add_0 = fc.filter_candidates(cliques_temp_add_0, flag=False)

cliques_temp1 = []
cliques_temp1_add = []
for parejas_5clique in cliques_temp_add_0:

    new_df_cliques1 = parejas_5clique[0]
    new_df_cliques2 = parejas_5clique[1]
    cliques_candidatos = iter_align(number_elements_clique, new_df_cliques1,
new_df_cliques2)

    if cliques_candidatos != []:
        cliques_temp1.append(cliques_candidatos)
        cliques_temp1_add.append(fc.add_element_to_clique(cliques_candidatos,
cliques_max_1, cliques_max_2))
```

```
number_elements_clique = number_elements_clique + 1

print(cliques_temp1[:1])
print(len(cliques_temp1))

print(cliques_temp1_add[:1])
print(len(cliques_temp1_add))
print('==================ya acabo con 5-cliques va con 6=======================')

cliques_temp_add_11 = [y for x in cliques_temp1_add for y in x]

# cliques_temp_add_11 = fc.filter_candidates(cliques_temp_add_11, flag=True)

cliques_temp2 = []
cliques_temp2_add = []
for parejas_6clique in cliques_temp_add_11:

    new_df_cliques1 = parejas_6clique[0]
    new_df_cliques2 = parejas_6clique[1]
    cliques_candidatos = iter_align(number_elements_clique, new_df_cliques1,
new_df_cliques2)

    if cliques_candidatos != []:
        cliques_temp2.append(cliques_candidatos)
        cliques_temp2_add.append(fc.add_element_to_clique(cliques_candidatos,
cliques_max_1, cliques_max_2))

number_elements_clique = number_elements_clique + 1

print(cliques_temp2[:1])
print(len(cliques_temp2))

print(cliques_temp2_add[:1])
print(len(cliques_temp2_add))
print('==================ya acabo con 6-cliques va con 7=======================')

cliques_temp_add_22 = [y for x in cliques_temp2_add for y in x]
# cliques_temp_add_22 = fc.filter_candidates(cliques_temp_add_22, flag=True)

cliques_temp3 = []

for parejas_7clique in cliques_temp_add_22:

    new_df_cliques1 = parejas_7clique[0]
    new_df_cliques2 = parejas_7clique[1]
    cliques_candidatos = iter_align(number_elements_clique, new_df_cliques1,
new_df_cliques2)

    if cliques_candidatos != []:
        cliques_temp3.append(cliques_candidatos)

print(len(cliques_temp3))
print([[x[0][0], x[0][1]] for x in cliques_temp3[:5]])
print('==================se imprimen candidatos
alineables=======================')
# APARTIR DE AQUI VA EL ALINEAMIENTO DE RESIDUOS POR MEDIO DE LAS PAREJAS
CANDIDATAS
parejas_cliques_finales = cliques_temp3
```

```
print(parejas_cliques_finales[0])
print(len(parejas_cliques_finales))

timenow = datetime.datetime.now()
print('Tiempo Total:', timenow - time_all)
print('termina alineamiento de cliques, se procede alineamiento de residuo a
residuo')

# AQUI COMIENZA EL ALINEAMIENTO RESIDUO A RESIDUO
pc = parejas_cliques_finales
pc = fc.filter_candidates(pc)

cliques_finales = []
for cand_1, cand_2, mat_rot in pc:
    cliques_finales.append([cand_1, cand_2])

print(cliques_finales)

res_conclq_1 = [res for res in pdb11]
res_conclq_2 = [res for res in pdb22]

atom_conclq_1 = [res.GetAtom('CA') for res in pdb11]
atom_conclq_2 = [res.GetAtom('CA') for res in pdb22]

coord_conclq_1 = np.array([res.coord for res in atom_conclq_1], dtype=np.float)
coord_conclq_2 = np.array([res.coord for res in atom_conclq_2], dtype=np.float)

bari_con_clq_1 = coord_conclq_1.mean(0)
bari_con_clq_2 = coord_conclq_2.mean(0)

vecs_center_cnclq_1 = coord_conclq_1 - bari_con_clq_1
vecs_center_cnclq_2 = coord_conclq_2 - bari_con_clq_2

number_of_residues_final = len(res_conclq_1)

val = 0
so_winner = 0.0
candidatos = []
winner_matrix_rotation = []
winner_baricenter = []
print('numero de comparaciones', len(pc))

def gen_rot_matrix_ref(parejas):
    """
    Genera la matriz de rotacion por medio de las coordendas de las parejas
seleccionadas
    :param parejas:
    :return: proteina rotada y trasladada, proteina a comparar, matriz de rotacion,
baricentro de parejas
    """
    # aveces truena si los pdbs no tienen los numeros de residuos continuos.
    coord_new_1 = [[res.GetAtom('CA').coord for res in res_conclq_1 if i[0] ==
res.resx] for i in parejas]
    coord_new_2 = [[res.GetAtom('CA').coord for res in res_conclq_2 if i[1] ==
res.resx] for i in parejas]

    coord_new_1 = np.array([y for x in coord_new_1 for y in x], dtype=np.float)
```

```
    coord_new_2 = np.array([y for x in coord_new_2 for y in x], dtype=np.float)

    bari_new_1 = coord_new_1.mean(0)
    bari_new_2 = coord_new_2.mean(0)

    vecs_center_1 = coord_new_1 - bari_new_1
    vecs_center_2 = coord_new_2 - bari_new_2

    # Se genera matriz de rotacion con vectores centricos de parejas anteriores
    matriz_R = fc.matrix_R(vecs_center_1, vecs_center_2)
    matriz_rotacion = fc.rotation_matrix(matriz_R)
    # se aplica matriz de rotacion a coordenadas de proteina a rotar
    # la proteina consiste en solo carbonos alfa
    vector_rotado = fc.rotation_vectors(vecs_center_cnclq_1, matriz_rotacion)

    protein_trasladado_rotado = vector_rotado + bari_new_2

    protein_to_compare = coord_conclq_2

    return (protein_trasladado_rotado, protein_to_compare,matriz_rotacion,
bari_new_2)


# cosas que tengo que declarar para que no me diga nada pycharm...
matriz_rotacion = []
bari_new_2 = []
winner_parejas = []

for cand_1, cand_2, mat_rot in pc:

    # primera iteracion sin cliques se aplica la matriz de rotacion y baricentro

    res_sinclq_1 = [res for res in pdb11 if res.resx not in cand_1]
    res_sinclq_2 = [res for res in pdb22 if res.resx not in cand_2]

    coord_sinclq_1 = np.array([res.GetAtom('CA').coord for res in res_sinclq_1],
dtype=np.float)
    coord_sinclq_2 = np.array([res.GetAtom('CA').coord for res in res_sinclq_2],
dtype=np.float)

    bari_1 = coord_sinclq_1.mean(0)
    bari_2 = coord_sinclq_2.mean(0)

    vecs_center_1 = coord_sinclq_1 - bari_1
    # aplico matriz de rotacion de cliques a vectores centricos sin clique
    vector_rotado = fc.rotation_vectors(vecs_center_1, mat_rot)
    protein_trasladado_rotado = vector_rotado + bari_2

    protein_to_compare = coord_sinclq_2

    # apilo la distancia y la pareja de residuos correspondientes si cumple con que
el RMSD sea menor a 3.5
    residuos_match = fc.fun_resiudos_match(protein_trasladado_rotado,
protein_to_compare,
                                           res_sinclq_1, res_sinclq_2)
    # filtro parejas
    cand_n = fc.filter_pairs(residuos_match, flag=False)
    # calculo el SO
    so_temp = round(len(cand_n) / (number_of_residues_final - 7), 4)
```

```
    so_temp_plus_1 = 0.0

    # Refinamiento por medio de las parejas seleccionadas y el clique.
    while so_temp_plus_1 < so_temp:  # Primer refinamiento
        parejas = [i[1] for i in cand_n]
        for i, j in zip(cand_1, cand_2):
            parejas.insert(0, (i, j))
        # aqui comienza el segundo alineamiento!! Refinamiento
        ptr, ptc, mr, bc = gen_rot_matrix_ref(parejas)
        # match residuos ordenado por distancia
        rm = fc.fun_resiudos_match(ptr, ptc, res_conclq_1, res_conclq_2)

        # quitar residuos repetidos
        cand_n = fc.filter_pairs(rm, flag=False)
        so_temp_plus_1 = round(len(cand_n) / number_of_residues_final, 4)
        so_temp_minus_1 = so_temp

        if so_temp_plus_1 < so_temp:  # evita infinite loop
            break

        # Rerefinamiento por si puede ir encontrando nuevas y mejores parejas
        while so_temp_minus_1 < so_temp_plus_1:  # segundo refinamiento iterativo
            so_temp_minus_1 = so_temp_plus_1
            parejas = [i[1] for i in cand_n]
            for i, j in zip(cand_1, cand_2):
                parejas.insert(0, (i, j))

            # aqui comienza el segundo alineamiento!! Refinamiento
            ptr, ptc, matriz_rotacion, bari_new_2 = gen_rot_matrix_ref(parejas)
            # match residuos ordenado por distancia
            rm = fc.fun_resiudos_match(ptr, ptc, res_conclq_1, res_conclq_2)

            # quitar residuos repetidos
            cand_n = fc.filter_pairs(rm, flag=False)
            so_temp_plus_1 = round(len(cand_n) / number_of_residues_final, 4)

        # actualizacion de datos
        if so_temp_plus_1 < so_temp_minus_1:
            so_temp_plus_1 = so_temp_minus_1
        # actualizacion de datos

        if so_temp_plus_1 > so_temp:
            so_temp = so_temp_plus_1

    # check que si este guardando el SO
    # print(so_winner)

    # Si supera el SO ganador se guardan los parametros y se actualiza el SO
    if so_temp > so_winner:

        so_winner = so_temp  # actualizacion so
        winner_matrix_rotation = matriz_rotacion  # actualizacion mr
        winner_baricenter = bari_new_2  # actualizacion bc
        candidatos = [cand_1, cand_2]  # actualizacion de parejas de cliques
estrella
        winner_parejas = cand_n   # actualizacion de parejas y distancia.
        print('========================'*3)
        print('viejo so:',so_temp_plus_1)
```

```
        print('cliques', candidatos)
        print('numero de parejas', len(cand_n))
        print('iteracion %s' % val, 'SO: %1.4f' % so_temp)
        print('RMSD:', np.mean([x[0] for x in cand_n]))
        # print('parejas:', [x[1] for x in cand_n])

print('================================================================================
====')

    val = val+1

    if so_temp == 1:
        break

print('=====pareja ganadora======')
print('cliques', candidatos)
print('numero de parejas', len(winner_parejas))
print('SO: %1.4f' % so_winner)
print('RMSD:', np.mean([x[0] for x in winner_parejas]))
print('parejas:', sorted([x[1] for x in winner_parejas]))

# se escribe el nuevo pdb rotado y trasladado
print('escribiendo resultados')

num_match_atoms = len(winner_parejas)
num_res_total = number_of_residues_final
rmsd = round(np.mean([x[0] for x in winner_parejas]), 4)
so_out = so_winner
parejas = sorted([x[1] for x in winner_parejas])

df = pd.DataFrame([file1.split("/")[-1].split(".pdb")[0]+"_"+file2.split("/")[-
1].split(".pdb")[0], 'model_s', candidatos, num_match_atoms, num_res_total,
                   so_out, rmsd, parejas],
                   index=['proteinaA_proteinaB', 'grupo', 'cliques_ganadores',
'num_parejas', 'num_residuos_total',
                   'SO', 'RMSD', 'parejas']).T

df.to_csv(directory+file1.split("/")[-1].split(".pdb")[0]+"_"+file2.split("/")[-
1].split(".pdb")[0]+'_mani.csv')


# Actualizacion de coordendas
file_output_name = directory+file1.split("/")[-
1].split(".pdb")[0]+"_"+file2.split("/")[-
1].split(".pdb")[0]+'_'+str(datetime.datetime.now())[:19]

coord_protein_1 = np.array([res.GetAtom(name).coord for res in pdb11 for name in
res.atomnames],
                           dtype=np.float)
bari_full_1 = coord_protein_1.mean(0)
vecs_center_protein_1 = coord_protein_1 - bari_full_1

# aplicacion de la rotacion y traslacion a toda la proteina
vector_rotado = fc.rotation_vectors(vecs_center_protein_1, winner_matrix_rotation)
protein_trasladado_rotado = vector_rotado + winner_baricenter

# actualizacion de coordenadas
k = 0
for res in pdb11:
```

```
    for atom in res.atomnames:
        setattr(res.GetAtom(atom), 'coord', protein_trasladado_rotado[k])
        k = k+1

# escritura del pdb
pdb1.WriteToFile(file_out_name=file_output_name)

#tiempo de ejecucion
timenow = datetime.datetime.now()
print('el cutoff angulo diedro era de:', global_cutoff)
print('Tiempo Total:', timenow - time_all)
print('termine puedes alinear utilizando los pdbs %s %s y el alineamiento %s' %
(file1.split("/")[-1].split(".pdb")[0],file2.split("/")[-1].split(".pdb")[0],

file_output_name))
```

Apéndice C

Comparación de resultados del software Mani y el software respuesta Click

Se presenta la siguiente tabla ordenada por el *Score* de Sobreposición en Mani, donde se puede comparar el SS, RMSD, el clúster de similitud, el score de similitud secuencial y el número de residuos totales que tiene la proteína a alinear.

Proteína A Proteína B	SO Mani	SO Click	RMSD Mani	RMSD Click	Cluster	Score de similitud secuencial	Número residuos totales
4BWP_A_4BWP_B	61.92	80.91	2.3101	1.42	cluster_70	96.37	372
5FHX_H_5HCG_H	64.76	70.78	2.3003	2.11	cluster_95	97.65	332
2ZMF_A_2ZMF_B	80.84	86.23	1.9041	1.48	cluster_70	97.09	167
4PED_A_5I35_A	84.25	78.86	2.1317	2.34	cluster_30	98.92	369
3SHP_A_3SHP_B	88.96	90.91	1.5509	1.2	cluster_30	92.31	154
4RN3_A_4RN3_B	89.14	89.56	1.7179	1.18	cluster_50	90.55	182
3EUC_A_3EUC_B	90.42	92.08	2.3437	0.99	cluster_70	96.87	341
2XXZ_A_2XXZ_B	91.4	92.66	1.9487	0.86	cluster_30	91.8	286
3E18_A_3E18_B	95.55	95.85	0.9577	0.55	cluster_30	91.88	337
5UEB_A_5UEB_B	95.77	95.77	0.5994	0.88	cluster_30	94	144
6BK4_A_6BK4_B	95.79	96.84	1.4749	0.51	cluster_50	84.4	95
3OI8_A_3OI8_B	95.8	98.67	1.3438	1.31	cluster_70	98.68	150
3FO5_A_3FO5_B	96.57	96.57	1.4358	0.55	cluster_70	91.09	233
4CCG_X_4CCG_Y	97.01	97.01	0.753	0.37	cluster_30	89.33	67
3QQ6_A_3QQ6_B	97.06	97.06	1.2636	0.58	cluster_30	91.67	68
3DOM_A_3DOM_C	97.26	97.26	0.7411	0.62	cluster_30	89.87	77
3C5W_P_3C5V_A	97.28	98.64	1.009	1.17	cluster_50	99.32	294
3PHQ_B_3PHO_B	97.58	99.53	1.0634	1.17	cluster_95	94.34	216
1Q40_B_1Q40_D	97.63	98.22	0.553	0.78	cluster_70	91.76	169
2IJ9_A_2IJ9_B	97.66	98.13	0.7734	0.94	cluster_95	99.07	216
3KCN_A_3KCN_B	97.66	98.44	0.6554	0.79	cluster_70	98.46	130
4RZK_A_4RZK_B	97.7	96.55	0.8928	0.76	cluster_70	97.73	87
5X68_A_5X68_B	97.71	97.99	0.8347	0.58	cluster_70	98.58	350
5EIP_A_5EIP_B	97.76	97.76	0.6907	0.68	cluster_50	100	134
3ZZO_A_3ZZR_A	97.85	98.92	0.6555	0.97	cluster_70	97.89	95
1P91_A_1P91_B	98.01	98.8	0.942	1.1	cluster_30	96.17	251
1W7P_A_1U5T_A	98.12	98.12	0.7848	0.89	cluster_95	100	213
5B52_A_5B52_B	98.25	85.96	1.843	1.97	cluster_50	95	60
5U4Q_A_5U4Q_B	98.3	98.3	0.7864	0.59	cluster_50	87.16	294

4HRV_A_4HRV_B	98.53	98.53	0.9881	0.52	cluster_70	93.01	141
4F5C_E_4F5C_F	98.56	98.56	1.2511	0.42	cluster_30	95.21	139
5IR0_A_5IR0_B	99	99	0.861	0.6	cluster_95	97.06	101
3MKY_B_3MKY_P	99.12	99.12	0.5896	0.38	cluster_50	99.13	114
4ETV_A_4ETV_B	99.23	99.23	0.6649	0.97	cluster_95	68.82	130
1USP_A_1USP_B	99.24	99.24	0.8985	0.43	cluster_70	95.59	132
3FBK_A_3FBK_B	99.27	99.27	0.6855	0.26	cluster_50	98.55	137
3HR6_A_3HTL_X	99.28	99.28	0.5423	0.37	cluster_30	97.88	418
3AUW_B_3AUW_D	99.4	99.43	0.4661	0.62	cluster_30	100	175
5EYB_A_5EYB_B	99.41	99.7	0.9021	1.05	cluster_70	99.12	337
5YHU_A_5YHU_B	99.45	99.45	0.3636	0.36	cluster_50	95.29	182
3I57_A_3I57_B	99.46	97.83	1.0001	1.14	cluster_95	100	184
3S6I_A_3S6I_D	99.51	100	0.3241	0.42	cluster_50	99.52	207
4Y2W_A_4Y2W_B	99.74	100	0.2538	0.41	cluster_70	100	388
3IFR_A_3IFR_B	99.79	100	0.5869	0.54	cluster_50	98.34	475
4PUC_A_4PUC_B	99.79	99.79	0.2783	0.26	cluster_95	99.17	483
3H1Q_A_3H1Q_B	100	100	0.8037	0.81	cluster_95	98.51	269
6BR8_A_6BR8_B	100	100	0.3559	0.43	cluster_95	100	245
1XGY_L_1XGY_M	100	100	0.4669	0.52	cluster_95	100	216
4ZI9_A_4ZI9_B	100	100	0.1849	0.22	cluster_95	100	309
5DLM_H_5DLM_I	100	100	0.1871	0.23	cluster_95	98.62	214
3RYK_A_3RYK_B	100	100	0.2887	0.39	cluster_95	97.79	175
4EM8_A_4EM8_B	100	100	0.1386	0.16	cluster_95	99.32	145
1G3N_C_1G3N_G	100	100	0.4428	0.12	cluster_95	100	233
2BYC_A_2BYC_B	100	100	1.9179	0.36	cluster_95	96.35	132
1BW0_A_1BW0_B	100	100	0.5334	0.26	cluster_95	99.76	412
5UE7_A_5UE7_B	100	100	0.7595	0.49	cluster_95	100	251
4S2S_H_4S2S_A	100	100	0.3454	0.41	cluster_95	99.55	219
1WUF_A_1WUF_B	100	100	0.0219	0.05	cluster_95	100	371
3RGH_A_3RGH_B	100	100	0.0457	0.07	cluster_95	98.97	96
1EDZ_A_1EE9_A	100	100	0.5878	0.33	cluster_95	100	317
3G14_A_3G14_B	100	100	0.6545	0.13	cluster_70	98.29	172
3M8J_A_3M8J_B	100	100	1.2705	0.62	cluster_95	97.78	88
2C2N_A_2C2N_B	100	100	0.7033	0.36	cluster_95	96.55	308
4EX6_A_4EX7_A	100	100	0.2183	0.25	cluster_70	99.09	217
5ZJG_B_5ZJG_D	100	100	0.1699	0.23	cluster_70	99.49	194
2ZSI_B_2ZSH_B	100	100	0.7849	0.22	cluster_70	98.33	59
2B5O_A_2B5O_B	100	100	0.3861	0.11	cluster_30	100	292
1FFT_B_1FFT_G	100	100	0.3372	0.06	cluster_30	100	257
3P1X_A_3P1X_B	100	100	0.3279	0.45	cluster_30	94.12	64
5VG9_A_5V7P_A	100	100	0.6147	0.52	cluster_30	98.93	280
3E4V_A_3E4V_B	100	100	0.1632	0.21	cluster_30	97.28	173
4MZZ_A_4MZZ_B	100	100	0.2149	0.31	cluster_30	96.88	32
1MIJ_A_1XPX_A	100	100	0.4177	0.52	cluster_30	85.35	144
3FZY_A_3FZY_B	100	100	0.2777	0.4	cluster_30	89.54	211
3ONL_C_3ONJ_A	100	100	1.0036	0.99	cluster_30	97.94	97

2A7L_A_2A7L_B	100	100	0.3312	0.12	cluster_30	100	123
5E27_A_5E27_B	100	100	0.8412	0.77	cluster_30	96.92	220
3C4M_C_3C4M_D	100	100	1.5658	0.24	cluster_50	100	20
6DFD_A_6DFD_B	100	100	1.1295	0.76	cluster_50	91.73	122
5LEF_C_5LEF_D	100	100	0.4154	0.47	cluster_70	95.92	47
3VV4_A_3VV4_B	100	100	0.7218	0.27	cluster_50	95.62	160
1XX6_A_1XX6_B	100	100	0.2621	0.36	cluster_50	93.64	162
1U08_A_1U08_B	100	100	0.5858	0.35	cluster_50	98.95	378
2FA5_A_2FA5_B	100	100	0.772	0.62	cluster_50	97.08	136
5CA5_A_5CA5_B	100	100	0.1617	0.2	cluster_50	100	248
6BAK_A_6BAF_A	100	100	0.5049	0.34	cluster_70	99.34	303
3TEK_A_3TEK_B	100	100	0.0513	0.09	cluster_70	100	139
2H1T_A_2H1T_B	100	100	0.2136	0.25	cluster_30	98.92	183
5AX7_A_5AX7_B	100	100	0.0887	0.24	cluster_70	98.5	329
5NA1_A_5NA4_A	100	100	0.1602	0.2	cluster_70	99.75	398
5CWS_F_5CWS_L	100	100	0.356	0.42	cluster_30	100	40
4GCN_A_4GCN_B	100	100	0.5665	0.64	cluster_70	100	127
5IZW_A_5IZW_B	100	100	0.055	0.08	cluster_70	100	101
5LKQ_A_5LKQ_B	100	100	0.6183	0.46	cluster_70	99.44	177
2RIK_A_2RJM_A	100	100	0.2525	0.27	cluster_50	99.65	283
1A0E_A_1A0E_D	100	100	0.0527	0	cluster_95	100	443

Tabla 8: Resultados de alineamientos utilizando el software Mani vs el software Click, reportando el Score de Sobreposición, RMSD, a que clúster pertenecían, el score de similitud secuencial y el número de residuos que contenía la proteína a alinear.

Apéndice D

Tablas de métricas de centralidad de los nodos de las redes de contacto y de redes de correlación dinámica de las diversas dinámicas.

El enlace a la información es: https://drive.google.com/drive/folders/1stPi_eqtW0KPRu82A-9Ip9hzijbY_whX?usp=sharing.

Se alojan 8 tablas con los siguientes datos, número de residuo, grado e intermediación, las métricas de grado e intermediación se muestran en porcentaje. Las tablas corresponden a los datos generados de la red de contacto residuo a residuo y la red de correlación dinámica de los dímeros y tetrámeros con y sin cofactor.

Referencias

[1] M. Nguyen y M. Madhusudhan, «Biological insights from topology independent comparision of protein 3D structures,» *Nucleic Acids Res.*, vol. 39, nº e94, 2011.

[2] C. Orengo, A. Todd y J. Thornton, From protein structure to function, vol. 9, Current Opinion in Structural Biology, 1999, pp. 374-382.

[3] T. M.E, Statistical Mechanics: Theory and Molecular Simulation, New York: Oxford University Press Inc, 2010.

[4] E. Paquet y H. Viktor, «Molecular Dynamics, Monte Carlo Simulations, and Langevin Dynamics: A Computacional Review.,» *BioMed Research International,* p. 18, 2015.

[5] A. Ribeiro y V. Ortiz, «A chemical perspective on Allostery,» *Chemical Review,* vol. 116, 2016.

[6] M. Newman, Networks An Introduction, vol. 33, Michicagan: Machine Learning 33, University of Michigan, Institute Santa Fe, 2010.

[7] P. Tina, K. Yasushi, T. Sandhya P., M. Stephen H., K. Katherine R., Z. Xiuwei, S. Annette, R. Nathalie, C. Jane y T. Sarah A., «Evolution of oligomeric state through allosteric pathways that mimic ligand binding,» *Science,* vol. 346, nº 6216, pp. 1254346-1-7, 19 Diciembre 2014.

[8] K. Wolfgang y S. Christian, «Dictionary of Protein Secondary Structure: Pattern Recognition of Hydroge-Bonded and Geometrical Features,» vol. 22, pp. 2577-2637, 1983.

[9] S. Tamar, Molecular Modeling and Simulation: An Interdisciplinary Guide, Segunda ed., S. Antman, J. Marsden y L. Sirovich, Edits., New York: Springer, 2010.

[10] E. Lindahl, «Molecular Dynamics Simulations,» de *Methods in Molecular Biology*, vol. 443, A. Kukol, Ed., Totowa, Human Press, p. 23.

[11] G. Plopper, D. Sharp y E. Sokorski, Lewin`s Cells, Tercera ed., Jones & Barlett Learning.

[12] «Educational portal of Protein data bank,» 1 Diciembre 2018. [En línea]. Available: https://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/methods-for-determining-structure. [Último acceso: 7 Enero 2019].

[13] M. B. HeleN, W. John, F. Zukang, G. Gary, N. B. T, W. Helge, N. S. Ilya y B. P. E, «The Protein Data Bank,» *Nucleic Acids Research,* vol. 28, nº 1, pp. 235-242, 1 Enero 2000.

[14] «Educational Portal of protein data bank,» 20 Agosto 2018. [En línea]. Available: https://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/introduction. [Último acceso: 8 1 2019].

[15] B. Contreras-Moreira, «Algoritmos en bioinformática estructural,» España, Zaragoza: Fundacion ARAID, 2008.

[16] G. Ramachandran, C. Ramakrishnan y V. Sasisekharan, «Stereochemistry of Polypeptide Chain Configurations,» *J. Mol. Biol,* nº 7, pp. 95-99, 1963.

[17] L. Manuel, B. Rainer, O. Katja y M. Zacharias, «Exploring biomolecular dynamics and interactions using advanced sampling methods,» *Journal of Physics: Condensed Matter,* vol. 27, 21 Julio 2015.

[18] V. Blondel, J. Guillaume, R. Lambiotte y E. Lefebvre, «Fast unfolding of communities in large

networks,» *J. Stat. Mech,* p. 12, 25 Julio 2008.

[19] F. Reid, A. McDaid y N. Hurley, «Percolation Computation in Complex Networks,» p. 12, Abril 2012.

[20] A. H. Aric, A. S. Daniel y J. S. Pieter, «Exploring network structure, dynamics, and function using NetworkX,» *Proceedings of the 7th Python in Science Conference (SciPy2008),* pp. 11-15, Agosto 2008.

[21] A. C. Evangelos, S. Chaok y A. D. Ken, «Using Quaternions to Calculate RMSD,» *J Comput Chem,* vol. 25, pp. 1849-1857, 2004.

[22] K. K. S, «On the Orthogonal transformation used for structural comparisons,» *Acta Cryst,* vol. A45, pp. 208-210, 1989.

[23] B. S y W. J, «Detectung allosteric networks using molecular dynamics simulation,» *Methods in Enzymology,* vol. 578, pp. 429-447, 2016.

[24] H. PH, M. AE y v. G. WF, «Fluctuation and cross-correlation analysis of protein motions observed in nanosecond molecular dynamics simulations.,» *J Mol Biol.,* vol. 252, nº 4, pp. 492-503, 1995.

[25] S. M y S. J, «MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets,» *Nature Biotechnology,* 2017.

[26] J. A. Mark, M. Teemu, S. Roland, P. Szilard, C. S. Jeremy, H. Berk y L. Erik, «GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers,» *ScienceDirect,* Vols. %1 de %21-2, pp. 19-25, 2015.

[27] D. G. Leonardo, M. Milagros y H. J. Andre, «Structure-based classification of FAD binding sites: A comparative study of structural alignment tools,» *Proteins,* vol. 84, pp. 1728-1747, 2016.

[28] F. Mathieu y R. G. Michael, «A comparison of common programming languages used in bioinformatics,» *BMC Bioinformatics,* vol. 9, p. 82, 2008.

[29] S. G, Linear Algebra and Its Applications, Segunda ed., Orlando, Florida: Academic Press, Inc, 1980.

[30] S. Dajms, M. Arciniega, T. Strinmetzer, R. Huber y M. Than, «Structure of the unliganded from of the proprotein convertase furin suggests activation by substrate-induced mechanism,» *PNAS,* vol. 113, pp. 11196-11201.

[31] Y. Kim, S. Seo, Y. Ha, S. Lim y Y. Yoon, «Two applications of clustering thecniques to twitter: Community detection and issue extraction,» vol. 2013, p. 8, 31 Octubre 2013.

[32] G. T. Wouter, B. Coos, B. Jon, A. t. B. Tim, K. E, P. J. Robbie y V. Gert, «A series of PDB related databases fro everyday needs,» *Nucleic Acids Research,* vol. 43, Enero 2015.

[33] J. W. Duncan y H. S. Steven, «Collective dynamics of "small-world" networks,» *Nature,* vol. 393, pp. 440-442, 4 Junio 1998.

[34] A. Rodrigo, Detección de comunidades en redes complejas, Valencia.