ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

CONSTRUCCIÓN DE UN ALGORITMO COMPUTACIONAL PARA MEDIR PROPIEDADES TOPOLÓGICAS DE REDES DE COMUNICACIÓN CELULAR

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE FÍSICO

PROYECTO DE INVESTIGACIÓN

JUAN FRANCISCO CHICA MIRANDA

juan.chica01@epn.edu.ec

DIRECTOR: RAMON XULVI-BRUNET, PhD

ramon.xulvi@epn.edu.ec

Quito, marzo, 2023

DECLARACIÓN

Yo, Juan Francisco Chica Miranda, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

La Escuela Politécnica Nacional puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Juan Francisco Chica Miranda

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por JUAN FRANCISCO CHICA MIRANDA, bajo mi supervisión.

Ramón Xulvi-Brunet. PhD

DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

A Dios, que no me ha desamparado, aunque el camino ha sido largo y sinuoso.

A mi madre, Paola, por estos años de vida que ha procurado en mi bienestar y fomentado un pensamiento crítico, sensible y empático. Por mostrarme la resiliencia, voluntad y sacrificios necesarios para afrontar las adversidades y continuar con la vida.

A mi padre, Diego, por mostrarme como es el cariño y amor incondicional. Que la vida es importante por como la disfrutas.

A mi Jossy, por el apoyo, metas y propósitos necesarios para que valga la pena. Por mostrarme que los pequeños logros deben ser celebrados y los pequeños momentos son tan importantes como los grandes.

A mi tutor, Ramón, por aportar considerablemente en mi formación académica. Por el tiempo, dedicación y conocimientos impartidos que me provocan a buscar más. A mis profesores, Eliana y Nicolás, que me demuestran que la ciencia va de la mano con el aporte a la comunidad.

A mis abuelos y tíos, por heredarme las cosas más preciadas e intangibles, como la dedicación, la responsabilidad, el amor y el cariño. A mis hermanas por el apoyo y enseñarme tantas cosas sin proponérselo.

A mis amigos universitarios, Carlos, Juanfra, Rubén, Jenn, Cristo, Marck y Kevin. Por tantas experiencias, camaradería y amistad. Por los retos y el apoyo académico. A mis amigos colegiales, Moi, Jonny, Sebas, Hugo, Harry, Richi y Shutter. Por una amistad que ha trascendido el tiempo y distancia.

A todas las personas que son nombradas en este agradecimiento y a las que no, pero han tocado mi vida. Sepan que no hubiera llegado hasta aquí si no fuera por cada una de ustedes.

Juan Francisco

DEDICATORIA

Para mis primos y mis hermanas. Que logren más de lo que se proponen.

Para JuanFra, con tranquilidad y paciencia lo logrará.

Índice general

Índice de figuras

Abstract

The structure of cell signaling networks determines the dynamics of physiological processes in living organisms, processes that belong both to regular functions and diseases. In the area of complex networks some metrics are used to determine the topological structure of networks, such as the degree sequence, the local clustering coefficient, the closeness centrality, and the degree-degree correlations. Here, we built a computational algorithm to compute all these metrics both to characterize cell signaling networks and to find structural differences between the networks corresponding to regular and disease processes. Our results show some differences between both kinds of networks.

Resumen

La estructura de las redes de comunicación celular determina la dinámica de los procesos fisiológicos presentes en los organismos vivos, tanto si los procesos corresponden a funciones regulares como a enfermedades. Medidas utilizadas en el campo de redes complejas, como degree sequence, clustering coefficient, local clustering coefficient, closeness centrality y degree degree correlation, caracterizan la estructura de la red. Aquí, construimos un algoritmo computacional que calcula todas estas medidas con el fin de encontrar diferencias estructurales entre redes de comunicación celular asociados a procesos regulares y de enfermedades. Encontramos estas diferencias estructurales entre ambos tipos de procesos de comunicación celular en las medidas utilizadas.

Capítulo 1

INTRODUCCIÓN

1.1. Redes de comunicación celular

El término 'comunicación celular' se utiliza para nombrar a los procesos mediante los cuales las células se comunican con su entorno. Las células reciben señales externas a estas, llamadas estímulos, procesan las señales mediante reacciones entre los elementos internos de la célula y emiten una señal de respuesta. Este tipo de procesos también se conocen como transducción de señales o comunicación transmebranal.

Los estudios sobre la comunicación celular iniciaron con el descubrimiento de las hormonas. Las hormonas son proteínas complejas que alteran el comportamiento de las funciones de las células, produciendo una respuesta excitada (aceleración del proceso) o inhibida (disminución o anulación de la velocidad del proceso)[1]. Estos estudios iniciaron el área de conocimiento que actualmente investiga la comunicación celular.

Los procesos de comunicación celular se pueden dividir en tres etapas: recepción de un estímulo, transducción de la señal y la respuesta. Este proceso se da mediante una secuencia de interacciones o reacciones entre los elementos celulares (moléculas y biomoléculas[¶]). La secuencia de interacciones en cadena es la ruta por la cual se transmite el mensaje. Esta ruta es conocida como vía de comunicación.

Las moléculas en la superficie de la célula, llamadas receptores, reciben la señal y la transmiten hacia las moléculas en el interior de la célula. En el interior de la célula se produce la etapa de transducción. En esta etapa las moléculas modifican o alteran otras moléculas en una cadena de reacciones. En esta etapa también es usual que se produzcan múltiples bifurcaciones de la señal. Este fenómeno se da cuando las moléculas interaccionan con múltiples elementos en cada paso de la cadena[2].

Una vez realizado el proceso de transducción de la señal, se obtiene la respuesta. La respuesta celular es, básicamente, cualquier actividad realizada por la célula. Entre estas respuestas se incluyen procesos regulatorios y expresiones de genes. Estos últimos son los encargados de producir proteínas necesarias para el funcionamiento normal de la célula[2].

A continuación, presentamos un ejemplo de comunicación celular llamado transporte vesicular. Este es un proceso por el cual la célula secreta moléculas como respuesta. El proceso

[¶]Compuestos químicos orgánicos propios de los organismos vivos que se encuentran formados principalmente por cadenas de carbono e hidrógeno y otros elementos como el azufre, nitrógeno, oxígeno y fósforo. Las biomoléculas incluyen: lípidos, carbohidratos, proteínas y nucleótidos.

comienza con el estímulo del receptor **fosfolípido C** que hidroliza (reacción química que incluye moléculas de agua) la molécula de fosfato **PIP2** liberando una molécula de diglicerina que activa las proteínas cinasa C **PKC** e **IP3**. Estas movilizan el ion Ca^{2+}, que produce el fosfato **PIP3**. Esta molécula es la respuesta para este proceso que es un caso de transporte vesicular, proceso por el cual la célula secreta moléculas[1].

Los procesos de comunicación poseen dos características importantes: múltiples receptores para múltiples estímulos y la amplificación de la señal. Debido a estas características, existen múltiples vías por las cuales una señal produce una respuesta o, también, distintos estímulos producen una respuesta. Estas múltiples vías de comunicación se conectan entre sí, formando una estructura de red. Por ello, dichos procesos en la actualidad se conocen como redes de comunicación celular[1, 2].

La cantidad de elementos e interacciones presentes de los procesos de comunicación celular se ha incrementado con los años. Esto se debe al avance de las técnicas experimentales para detección de los elementos y a un entendimiento más profundo de estos procesos.

Por mencionar un ejemplo, el ciclo de Krebs, que se desarrolla en la mitocondria de la célula, es una cadena con 12 moléculas y 13 interacciones. En 2005, Albert R.[3] mencionó que la red de comunicación celular más grande hasta ese tiempo, poseía 1279 interacciones entre 545 elementos. Actualmente se puede hallar en sitios especializados de redes de comunicación redes de cerca de 15000 biomoléculas con 700000 interacciones, aproximadamente 30 veces más que hace 17 años.

Se ha visto que muchos de los procesos con un gran número de interacciones están relacionados con enfermedades o cáncer. Esto debido a la gran cantidad de investigación de estos procesos en áreas de ciencias médicas, y a la disponibilidad de muestras para estudiarlos[4, 5].

1.2. Relevancia biológica del estudio de redes de comunicación celular

Las respuestas de los procesos de comunicación celular en algunos casos se vuelven estímulos para otros. Por esta razón, varios procesos se concatenan aumentando la complejidad de estos. El resultado de miles de procesos celulares simultáneos resulta en las funciones necesarias para la supervivencia de un organismo vivo. Bajo esta perspectiva, estudiar las redes de comunicación celular es, en esencia, estudiar los procesos de la vida[5].

Estudios en el campo de comunicación celular[4, 5] resaltan que características de las respuestas, como los tiempos de respuesta y la intensidad[‡], se relacionan con la particular estructura que forman las conexión en las redes.

Para recalcar la relevancia de la estructura de una red, podemos tomar como ejemplo a las vías de comunicación terrestre de una ciudad (calles, autopistas) que conectan domicilios con sitios estratégicos de la ciudad, como trabajos, instituciones educativas, centros médicos, etc. Para establecer estrategias que permitan mejorar el flujo de automóviles, es necesario conocer el estado de las vías de acceso.

[‡]Ya que las respuestas difieren para cada proceso celular, la intensidad de la respuesta hace referencia a la cantidad de elementos producidos como iones, hormonas, proteínas, genes o energía según sea el caso.

Para saber qué vías son funcionales y qué rutas son óptimas, es útil conocer condiciones como: cuáles son las vías más rápidas, las menos utilizadas, conocer qué tantas opciones o vías alternas se pueden tomar para alcanzar cierto destino. Así, podemos determinar qué sitios de la ciudad son más concurridos o cuáles se hallan más accesibles. También se puede hallar qué sitios se encuentran más centrales para la mayoría de domicilios y cuáles se encuentran en los límites de la ciudad.

De manera similar, en los procesos de comunicación celular, extraer información sobre la estructura de la red permite tener un mejor entendimiento de los procesos y funciones biológicas subyacentes.

1.3. Análisis de la estructura de una red

Una red, llamada grafo en el área de matemática discreta y red compleja en física, es un objeto matemático que posee dos tipos de elementos básicos. El primero es un conjunto contable de puntos llamados vértices o nodos. El segundo es un conjunto de segmentos que unen dos vértices. A estos segmentos se les conoce como aristas, ejes o enlaces ((i, j) es la arista que conecta los nodos i y j). En el caso de los procesos de comunicación celular, los nodos representan a las moléculas del proceso y las aristas son las reacciones entre estas biomoléculas[6].

En la figura 1.1* se muestra la representación de una porción de un proceso de comunicación celular. En la parte superior izquierda se muestra el esquema gráfico del proceso. A la derecha de este se encuentra el conjunto de nodos. Etiquetamos a cada una de las moléculas del proceso con un número natural que indica el nodo asociado. Se representa las reacciones entre las biomoléculas participantes como las aristas de la red de forma gráfica en la parte inferior. Así, cada una de las aristas presentes en la red se muestra como un segmento que une dos vértices. Tomemos, por ejemplo, el segmento que conecta los nodos 19 y 20 como se observa en el gráfico. Este segmento simboliza la arista $(19, 20)$ y representa la interacción entre el segundo mensajero ciclo aminofosfato **cAMP** (nodo 19) con la proteína cinasa A **PKA**(nodo 20) mediante un proceso llamado activación.

El análisis de la estructura de redes se basa en calcular sus medidas topológicas. Algunas de estas medidas se relacionan con el análisis de caminos, grado de conectividad, correlaciones de las conexiones, etc. Estas medidas caracterizan la estructura de la red y se han estudiado formalmente por áreas de matemática discreta y sistemas complejos.

1.4. Medidas topológicas

Un camino en una red se define como una secuencia alternada de nodos y aristas por las cuáles un caminante ficticio puede llegar desde un nodo i hasta un nodo j atravesando las aristas y nodos de la secuencia consecutivamente. Un esquema de camino se muestra en la figura 1.2. Nos enfocaremos en caminos, llamados caminos simples, que tienen la característica de que los nodos contenidos en los camino no se repiten. Se debe aclarar que, en general, existen múltiples caminos entre dos nodos[6].

Se define la longitud de un camino como él número de aristas que este posee (métrica del taxista). El camino de la figura 1.2, por ejemplo, tiene una longitud de 4. En las redes de comunicación celular la longitud de un camino representa cuántas reacciones hay en una vía

*La imagen del esquema gráfico de la red de comunicación celular fue tomada de Azeloglu(2015)[4]

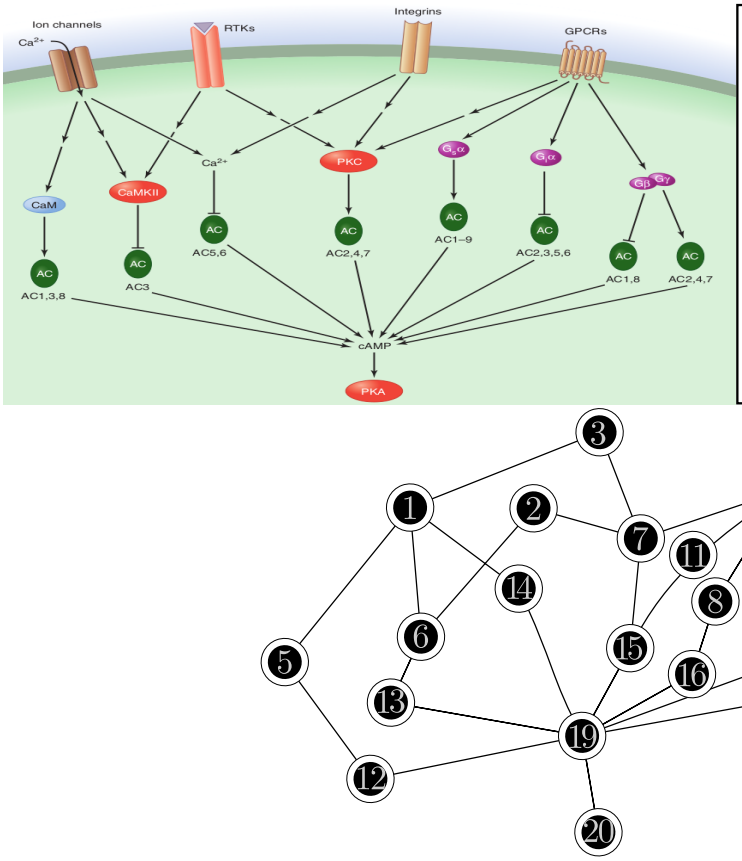

Figura 1.1: En la figura mostramos un fragmento de una red de transducción de señales. Dicho fragmento es el encargado de la recepción de señales en la membrana por diferentes tipos de receptores. Por interacción electrostática mediante el ión CA^{2+}, **rTKS** que recolecta señales producidas por hormonas. Los integrins se encargan de la recepción de señales mediante la adhesión de lípidos, los **GPCRs** también reciben señales por adhesión pero de proteínas y concatenan a estas con proteínas G. Este proceso después de señales intermedias, mediadas principalmente por isoformas de adenil ciclasa **AC** que interaccionan con el segundo mensajero **cAMP** el cuál cumple el rol de colectar todas las señales y regularlas para transmitirlas hacia otros elementos intracelulares. Se muestra también la abstracción de la red de comunicación celular a una red compleja.

de comunicación.

El **SAPL**(shortest average path length) es el valor promedio de las longitudes de los caminos con menor longitud entre todos los pares de nodos de la red. Este promedio se calcula si existe al menos un camino que conecte cada par de nodos. Esta medida topológica nos revela cuántas reacciones en promedio suceden en todos los procesos "ligados"de comunicación celular.

El grado de cada arista se define como el número de aristas a las que un nodo está conectado. El histograma de grados, llamado degree sequence, es una medida que indica la probabilidad p_k de que al escoger aleatoriamente un nodo en la red, este tenga un grado k. En muchas redes de comunicación celular, este histograma se puede aproximar con una ley de potencias[3]. Este comportamiento revela la presencia de nodos con una alta conectividad llamados **hubs**. En las redes de comunicación celular los nodos hubs son las moléculas que intervienen en muchas de las reacciones de varios procesos.

Se define el *clustering coefficient* $C(A)$, o coeficiente de agrupamiento, para un nodo de la red A, como la razón entre el número de triángulos con vértice A (en la figura 1.2 se muestra el

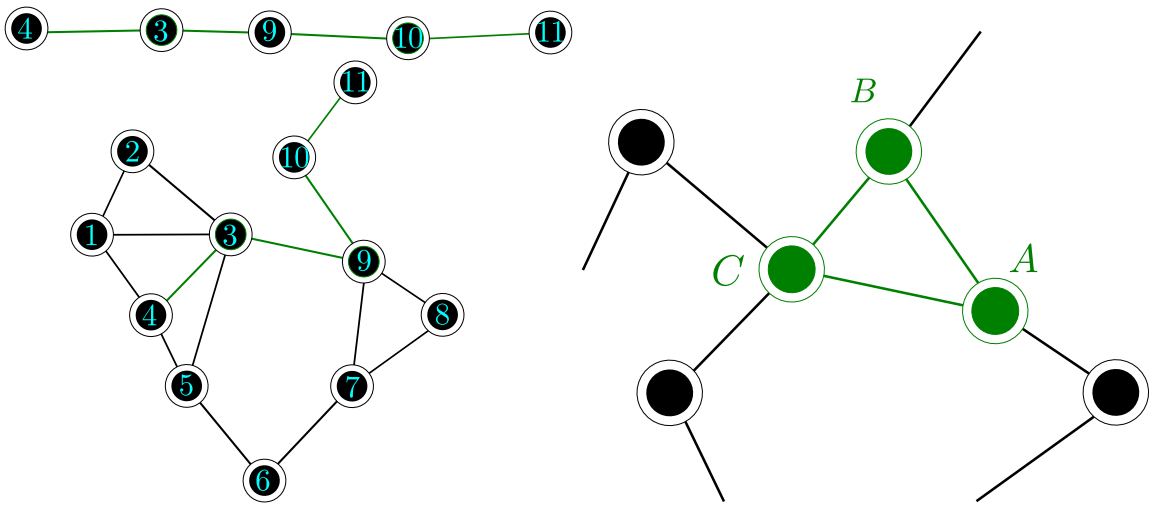

Figura 1.2: A la izquierda esquema de uno de los caminos más cortos entre los nodos 4 y 11 de la red, el camino pasa a través de los nodos 3, 9 y 10 en ese orden para conectar los nodos 4 y 11. A la derecha se muestra el triángulo $A - B - C$ formado por estos vértices y sus aristas correspondientes. El nodo A tiene 3 aristas conectadas a el, por lo cual su grado es $k_A = 3$. El coeficiente de este nodo es $C(A) = 1/3$.

triángulo de vértices A, B y C) sobre $\frac{1}{2}k_A(k_A - 1)$, con k_A el grado del nodo A. El *clustering coefficient* de la red se define como el valor promedio de los coeficientes de los nodos en la red. Este indicador es una medida de la transitividad entre nodos y también nos permite analizar la densidad de aristas en ciertas zonas de la red[6, 7]. Estudios[3] mencionan que las redes biológicas que poseen altos coeficientes $C(k)$ en función del grado son capaces de desempeñar múltiples tareas.

Otra medida topológica es la correlación de grados. Esta medida se define como una función de dos variables (k, k') y es la razón entre el número de aristas que enlazan un nodo de grado k con un nodo de grado k' y el total de aristas de la red. Esta medida nos da información acerca de si los nodos tienden una tendencia a enlazarse con nodos de grado similar o diferente. Las redes de comunicación celular tienden a poseer un comportamiento 'disortativo', el cuál dice que los nodos con grado alto tienden a enlazarse con nodos de grado bajo[6].

La centralidad es una medida que se define para cada nodo de la red y nos proporciona información sobre los nodos "más importantes"de la red. Ma H. et al.[8] encontró resultados similares al analizar varias medidas de centralidad en redes biológicas. Se encontró que las moléculas con un valor alto de centralidad son importantes para mantener la conectividad y estabilidad de la red.

En este trabajo realizamos una primera descripción exploratoria de la topología de redes de comunicación celular. Para estimar la centralidad de los nodos en estas redes es suficiente usar una medida estándar de centralidad, ya que no buscamos precisión. Aquí, decidimos caracterizar esta medida mediante la centralidad *closeness* por su cálculo más eficiente[9], en comparación con otras medidas. Esta medida se define para un nodo i como el inverso de la suma de las longitudes de camino mínimas entre el nodo i y el resto de nodos de la red. Esta medida indica qué tan cercano, con respecto a longitudes de camino más cortas, se encuentra un nodo de los demás.

1.5. Objetivo

El objetivo de este proyecto es caracterizar la estructura topológica de varias redes de comunicación celular para extraer información que se pueda relacionar con propiedades de los procesos. También buscamos comparar los resultados obtenidos de redes de procesos naturales con redes de procesos asociados a enfermedades y analizar si hay diferencia entre estos a nivel topológico.

Al poseer estas redes una gran cantidad de elementos interactuantes es necesario realizar los cálculos de las medidas topológicas y estadísticas mediante métodos computacionales. Se diseñará varios algoritmos que puedan calcular las medidas descritas (SAPL, CC, degree sequence, correlaciones de grado y centralidad closeness) de una red general y así aplicarlos sobre redes de comunicación celular para analizar estos indicadores.

Se validará la correcta ejecución de los algoritmos antes de aplicarlos sobre las redes de comunicación celular. Para esto, los ejecutaremos primero sobre una red ya conocida, la red Barabási-Albert, buscando reproducir los resultados.

Las redes serán obtenidas en bases de datos especializadas en este ámbito. Utilizaremos la base de datos NDEx. Este es un repositorio muy completo y confiable que agrupa la información de varias bases de datos de procesos celulares y genéticos. Entre las redes del repositorio podemos encontrar redes de procesos regulares o sanos de transducción de señales y de procesos asociados a enfermedades. Obtendremos estos dos tipos de redes y calcularemos sus medidas para contrastar si existe diferencia en la estructura de estas.

Aquí presentamos un estudio exploratorio que analiza las medidas más usuales en teoría de redes. Nuestro estudio se restringirá al análisis de nueve redes de comunicación celular. Se calcularán las medidas topológicas básicas anteriormente descritas. Las medidas serán comparadas con el modelo teórico de Barabási-Albert y se intentará inferir características de los procesos en base a su estructura.

1.6. Outline

Este trabajo se encuentra dividido en 5 capítulos, empezando con el presente capítulo introductorio. El segundo capítulo contiene toda la teoría necesaria sobre redes y sus medidas topológicas. En el capítulo 3 se describirán los algoritmos creados y la metodología empleada en esta tesis. En el cuarto capítulo se expondrán los resultados obtenidos sobre las redes reales y la comparación con el modelo teórico. En el último capítulo se presentarán las conclusiones del proyecto.

Capítulo 2

MARCO TEÓRICO

2.1. Redes Complejas

2.1.1. Red simple indirecta

Una red simple e indirecta se define como $G = (V, E)$, donde V es el conjunto de nodos y E representa el conjunto de aristas. Sean $i, j \in V$ un par de nodos, el par $(i, j) = e$ es una arista si $e \in E$. Una red es indirecta si para cada arista $e_1 = (i, j) \in E$ existe una arista $e_2 = (j, i) \in E$[7].

2.1.1.1. Primeros vecinos

Sea una red $G = (V, E)$. Se dice que un nodo j es primer vecino de i, si la arista (i, j) o (j, i) pertenecen a E. Se define $\kappa_1(i)$, el conjunto de los primeros vecinos de i:

$$\kappa_1(i) = \{j \in V : (j, i) \vee (i, j) \in E\}. \tag{2.1}$$

Se cumple $j \in \kappa_1(i) \Leftrightarrow i \in \kappa_1(j)$ entre los nodos $i, j \in V$, lo que indica una propiedad de reflexividad para este conjunto.

2.1.2. *Degree*

Degree o grado de un nodo se define como el número de aristas conectadas este[6].

Notamos $Card(x) = |x|$ para un conjunto x. Con la ecuación 2.1 el grado de un nodo $i \in V$ es $k_i = |\kappa_1(i)|$.

2.1.3. Componentes

Sea $G = (V, E)$ una red, $G_1 = (V_1, E_1)$ es una sub red de G si $V_1 \subseteq V$ y $E_1 \subseteq E$, se denota $G_1 \subseteq G$.
Una red es conexa si existe al menos un camino entre par de nodos de la red. El componente de una red, es una sub red $G_1 \subseteq G$ que es conexa. Se le llama componente gigante (*giant component*) si $G_1 = (V_1, E_1)$ y V_1 reúne la mayor parte de nodos de V[7].

2.2. Medidas Topológicas

En el siguiente apartado se define varias de las medidas topológicas usuales básicas que caracterizan la estructura de una red.

2.2.1. *Degree Sequence*

Se define p_k la fracción de nodos en la red que tienen grado k. También representa la probabilidad de que un nodo escogido aleatoriamente tenga grado k. Una gráfica de p_k en función de k para una red dada es equivalente a realizar el histograma de frecuencias relativas de los grados, el *degree sequence* de la red[6].

Se define el conjunto de nodos que poseen grado k:

$$N_1(k) = \{i \in V : |\kappa_1(i)| = k\}. \tag{2.2}$$

Entonces, se calcula $p_k = |N_1(k)|/|V|$.

2.2.2. *Clustering Coefficient* (CC)

Sea $G' = (V', E')$ una sub red conexa de $G = (V, E)$, tal que $|V'| = 3$. A G' se le llama triple conectado. Si dos de los nodos no se encuentran conectados entre sí, este se le llama triple abierto. Si hay al menos una arista entre cada par de nodos se le dice triple cerrado. A los triples cerrados también se les llama triángulos. Definimos el conjunto de los triángulos centrados en un nodo i:

$$\Delta(i) = \{(i, j, k) \in V^3 : j \in \kappa_1(k) \wedge j, k \in \kappa_1(i)\}. \tag{2.3}$$

Si $t = (i, j, k) \in V^3$ y $j \notin \kappa_1(k)$ para j y k vecinos de i, entonces t es un triple abierto centrado en i. Si el nodo i tiene grado k_i, existen $\frac{1}{2}k_i(k_i - 1)$ triples centrados en i, para $k_i \geq 2$ o no hay suficientes vecinos para formar el triple.

Se define el clustering coefficient de un nodo i como:

$$C(i) = \frac{\text{número de triángulos centrados en } i}{\text{número de triples abiertos centrados en } i} = \frac{|\Delta(i)|}{\frac{1}{2}k_i(k_i - 1)}. \tag{2.4}$$

El clustering coefficient de la red G se define como el promedio de los coeficientes de cada nodo[6].

$$CC = \frac{1}{|V|}\sum_{i \in V} C(i). \tag{2.5}$$

2.2.3. Tomografía de redes

Este método se basa en examinar la estructura local de la red. Se escoge un nodo en un componente de la red, al que se le llama nodo raíz. A este nodo se le asigna la capa 0. Se busca todos los nodos enlazados a este y se agrupan en la capa número 1. En la capa 2 se agrupan todos los nodos que se encuentran enlazados a los nodos de la capa 1 y que no se encuentren en capas anteriores. Este proceso se realiza iterativamente hasta asociar a todos los nodos del componente, donde está el nodo raíz, con una capa[10].

2.2.4. Vecinos de orden superior

Siguiendo la metodología de tomografía de redes, tomando el nodo raíz i, la capa 0 es $\kappa_0(i) = \{i\}$. La capa 1 representa el conjunto de los primeros vecinos de i $\kappa_1(i)$ de la ecuación 2.1. El conjunto de los nodos en la capa 2 es:

$$\kappa_2(i) = \{j \in V : [(j, k) \vee (k, j) \in E \forall k \in \kappa_1(i)] \wedge [j \notin \kappa_1(i) \vee \kappa_0(i)]\}. \tag{2.6}$$

A los nodos en este conjunto también se les llama segundos vecinos de i. La capa de orden l se forma en base a las anteriores:

$$\kappa_l(i) = \{j \in V : [(j,k) \vee (k,j) \in E \forall k \in \kappa_{l-1}(i)] \wedge [j \notin \kappa_m(i) \forall 0 \leq m \leq l-1]\}. \tag{2.7}$$

Para dos nodos se cumple la reflexividad $i \in \kappa_l(j) \Leftrightarrow j \in \kappa_l(i)$, con el mismo orden de capa.

Si la cantidad de nodos $|V|$ de la red es finita, existe un l_{m_i} orden de capa máximo para cada nodo raíz i en la red.

2.2.5. Longitud de camino mínima

El camino más corto entre dos nodos se le conoce como geodésica. En general, pueden existir múltiples geodésicas entre un par de nodos. Si no existe un camino entre dos nodos la longitud de camino se dice infinita.

Sea $G = (V, E)$ una red conexa e indirecta y $L(i, j)$ la longitud de una geodésica entre los nodos $i, j \in V$. Entonces se cumple la proposición:

$$i \in \kappa_l(j) \Leftrightarrow l = L(i,j), \tag{2.8}$$

y el SAPL es:

$$\bar{l} = \frac{1}{\frac{1}{2}n(n-1)} \sum_{j>i} L(i,j). \tag{2.9}$$

Para redes no conexas $\bar{l} = \infty$. Por esta razón se suele trabajar el SAPL en cada componente.

2.2.6. Excentricidad

Se define la excentricidad de un nodo i como la mayor de las longitudes de las geodésicas $L(i, j)$, siendo j un nodo cualquiera del componente $G(i)$ donde se encuentra i .

$$\epsilon(i) = \max_{j \in V(i)} (L(i,j)). \tag{2.10}$$

Si la red es conexa e indirecta, la excentricidad es $\epsilon(i) = l_{m_i}$, el orden máximo de la capa para el nodo raíz i.

2.2.7. Diámetro

El diámetro de una red se define como la máxima longitud de las geodésicas entre los pares de nodos de la red[7].

$$D = \max_{i,j \in V}(L(i,j)). \tag{2.11}$$

En función de las excentricidades, el diámetro se calcula como:

$$D = \max_{i \in V}(\epsilon(i)). \tag{2.12}$$

2.2.8. Radio

El radio de una red se define como la mínima de las excentricidades de los nodos en la red:

$$R = \min_{i \in V}(\epsilon(i)). \tag{2.13}$$

2.2.9. *Closeness Centrality*

La centralidad de un nodo $i \in V$ se define como:

$$C_{clo}(i) = \left(\sum_{j \in V} L(i,j) \right)^{-1}, \tag{2.14}$$

2.2.10. Correlación

Si profundizamos un poco en la estadística de las redes, podemos preguntarnos qué tipo de nodos se unen entre sí. Se puede calcular esta 'elección' de emparejamiento entre nodos que poseen una de las características del conjunto $\hat{A}$ mediante una función de dos variables $f : E \longrightarrow A \times A$. f toma una arista de la red $(i,j) \in E$ y relaciona a los nodos i,j con las características $a_i, a_j \in A$ respectivamente. Así, $f[(i,j)] = (c_i, c_j)$[6].

Dado el conjunto de características A, se define el conjunto de aristas que enlazan dos nodos con características a_1, a_2:

$$N(a_1, a_2) = \{e \in E : f(e) = (a_1, a_2)\} \tag{2.15}$$

Sea $E_{a_1,a_2} = |N(a_1, a_2)|$ el número de ejes en una red que conecta nodos con la característica a_1 con nodos de característica a_2, $\hat{E}$ es la matriz que contiene a los elementos E_{a_i,a_j}. Se define la matriz de correlación entre nodos:

$$\hat{e} = \frac{\hat{E}}{|\hat{E}|}, \tag{2.16}$$

2.2.10.1. *Degree degree correlation*

Si el conjunto de características es k el conjunto de grados de los nodos de la red, $f[(i,j)] = (k_i, k_j)$ con k_i y k_j los grados de los nodos i y j respectivamente. Los elementos de la matriz $\hat{e}$ representan la probabilidad de que al elegir aleatoriamente una arista de la red, esta conecte a nodos con grado i y j.

Con la matriz $\hat{e}$ se puede calcular el grado medio $\langle K \rangle$ de un nodo conectado a un nodo de grado k_1:

$$\langle K \rangle (k_1) = \sum_{k_2} k_2 P(k_2|k_1), \tag{2.17}$$

con $P(k_2|k_1)$ la probabilidad condicional de hallar una arista con un nodo de grado k_2 en un extremo dado que al otro extremo el nodo tiene grado k_1.

Se define q_k, como la probabilidad de hallar una arista que se conecta a un nodo de grado k. Esta se relaciona con p_k la probabilidad de escoger un nodo en la red de grado k:

$$q_k = \frac{k p_k}{\langle k \rangle}. \tag{2.18}$$

También se relaciona con la matriz de correlación entre nodos $q_k = \sum_j e_{jk}$.

2.2.10.2. *Assortative mixing*

Newman[11] menciona que varias redes, como redes sociales, muestran 'assortative mixing' o assortatividad en sus grados, es decir, la preferencia de nodos de alto grado de conectarse con otros nodos de alto grado. Por otro lado, tipos de red, como redes de internet y redes biológicas, muestran que nodos de alto grado se conectan preferentemente con nodos de grado bajo, esta propiedad se refiere como disortatividad[12].

En una red que no posea una tendencia assortativa o disortativa, los elementos de la matriz de correlación tienen la forma $e_{ij} = q_i q_j$. Este es el comportamiento esperado en redes aleatorias como el modelo de Erdös-Rényi[11, 13]. Se define un coeficiente de correlación r:

$$r = \frac{1}{\sigma^2} \sum_{jk} jk(e_{jk} - q_j q_k). \tag{2.19}$$

Notamos que si el comportamiento es no correlacionado $r = 0$. Para un comportamiento perfectamente assortativo, $e_{ij} = \delta_{ij} q_j$, el numerador de la ecuación 2.19 toma el valor de σ^2:

$$\sigma^2 = \sum_k k^2 q_k - \left(\sum_k k q_k \right)^2. \tag{2.20}$$

En el caso de redes perfectamente disortativas $r = -1$. Se interpreta que una red es disortativa si $r < 0$ y assortativa si $r > 0$.

Para una red no correlacionada el grado medio de un nodo conectado a un nodo de grado k:

$$\langle K \rangle(k) = \frac{\langle k^2 \rangle}{\langle k \rangle}, \tag{2.21}$$

toma un valor constante. Para una red perfectamente assortativa $\langle K \rangle(k)$ 2.17 crece con respecto al grado, si es disortativa decrece con respecto a k[14].

2.2.11. Redes libres de escala

Las redes libres de escala o *Scale-free* son llamadas así porque su degree sequence es libre de escala. Esto significa que el degree sequence se puede ajustar a una ley de potencias de la forma $p_k \sim k^{-\gamma}$, donde γ es conocido como el exponente de grado.

2.2.12. Modelo de Barabási y Albert

En 1965 el físico Derek de Solla Price publicó uno de los estudios más antiguos de lo que hoy conocemos como redes libres de escala. Estudió una red de citaciones de papers científicos. Halló que el degree sequence de estas redes describen una ley de potencias. Años después explica este comportamiento en el degree sequence proponiendo que la cantidad de nuevas citaciones que un paper recibe se debe a la cantidad de citas que ya posee. Price le llamó ventaja acumulativa a este principio[6].

La ventaja acumulativa de Price, motivó el principio llamado *preferencial attachment* por Barabási y Albert[15]. Crearon el modelo Barabási-Albert (BA) siguiendo este principio. El modelo se basa en añadir progresivamente nodos a una red inicial. Cada nodo añadido se enlaza a m nodos de la red y la elección de estos nodos es aleatoria con una probabilidad proporcional a sus grados.

Al construir una red del modelo BA, en el límite de una red con infinitos nodos, se obtiene un degree sequence de la forma:

$$p_k = \frac{2m(m+1)}{k(k+1)(k+2)}. \tag{2.22}$$

Esta distribución, en el límite de $k \gg 1$, describe una ley de potencias $p_k \sim k^{-3}$ con exponente $\gamma = 3$[6].

Bollobás B. et. al[16, 17], demostraron que el SAPL de una red de BA para $m \geq 2$ tiende asintóticamente a:

$$\bar{l} \sim \frac{\ln |V|}{\ln \ln |V|}. \tag{2.23}$$

Y el CC de una red de BA para $m \geq 1$ decrece en función del número de nodos:

$$CC \sim \frac{(\ln |V|)^2}{|V|}. \tag{2.24}$$

Capítulo 3

DATOS Y METODOLOGÍA

3.1. Datos

3.1.1. Plataforma NDEx

La plataforma NDEx[18] es un proyecto de código abierto dedicado al almacenamiento, manipulación y publicaciones de redes biológicas. Esta base de datos trabaja con otras bases del área como Pathway Commons, KEGG o Reactome. También proporciona un enlace directo a la interfaz de software libre Cytoscape para la visualización de redes.

Los archivos de datos se descargan de forma gratuita con extensión '.cx'. Este archivo es compatible con Cytoscape, sin embargo, para redes con gran cantidad de datos, la renderización gráfica de la red no es eficiente. Este tipo de archivos tiene un formato de lectura tipo JSON, un formato usual en páginas web.

Para extraer los datos que utilizamos en este proyecto utilizamos las librerías de Python `json` y `pandas` junto al código que se encuentra en el anexo 6.3.1. Extraemos dos tablas de datos con la siguiente estructura:

	@id	n
1	1	SKY
2	2	MAPK3

	@id	s	t
1	1	2356	33
2	2	127	2800

Cuadro 3.1: Tabla de datos de un segmento de nodos a la izquierda y aristas a la derecha de la red de comunicación celular *Human gene regulatory network of mesothelioma* 3.1.3.4.

En este proyecto, que trabajamos sobre redes indirectas y no pesadas, utilizamos los datos que se muestran en la tabla 3.1. Los archivos en la base de datos poseen más información de las redes que las descritas, como pesos de aristas, tipo de interacciones o tipos de moléculas. Sin embargo, para el análisis que realizamos no utilizamos dichos datos.

3.1.2. Base de datos HGNC

La asociación HGNC es responsable de aprobar los nombres y símbolos únicos de proteínas, ncRNA, genes y pseudogenes, para permitir una comunicación científica clara y sin ambigüedad[19]. Esta base de datos posee una extensa información sobre proteínas y genes que incluye un nombre común, identificador alfanumérico, abreviación, nombre científico y demás símbolos aprobados.

En ocasiones se renombran las moléculas de la base, por esta razón también se guarda la información de nombres antiguos y también de alias que se usan en el campo. Esta base de datos se puede descargar y se encuentra organizada en forma de tablas, útil para la consulta de grandes volúmenes de datos.

En este trabajo utilizamos la base de datos HGNC para encontrar el tipo de moléculas que componen la red 3.1.3.6 y demostramos que es una red bipartita.

3.1.3. Descripción de las redes

En esta sub sección describimos las redes que obtuvimos de las bases de datos y después de extraer los datos necesarios para formar las redes, ejecutamos los algoritmos que caracterizan la estructura topológica.

3.1.3.1. *Human Phosphorylation Pathways*

SIGnaling Network Open Resource SIGNOR fue desarrollada para apoyar y almacenar la evidencia experimental de relaciones causales entre entidades biológicas a nivel molecular, principalmente sobre comunicación celular. La información recopilada en la base de datos puede ser representada como una red directa con pesos[20]. La red contiene 2225 nodos y 6051 aristas. En adelante nos referiremos a la red como red de comunicación celular 1 (CC1). Esta red podemos encontrarla en SIGNOR[21].

3.1.3.2. *NCI PID Complete Interactions*

Red de interacción molecular de procesos regulatorios a nivel celular. Estos eventos, clave en el funcionamiento de una célula, poseen una estructura de redes de comunicación. Los datos para formar la red se tomaron de la base Pathway Interaction Database[22].La red está formada por 2794 nodos y 25296 aristas. En este trabajo nos referimos a esta red como red de comunicación celular 2 (CC2). Obtuvimos la red de la base de datos NDEx[23].

3.1.3.3. *BioPlex-3-HEK293T*

Red de funcionamiento y comunicación celular en células de tipo **HEK293T**. Los genes y proteínas interactuantes se obtuvieron mediante técnicas de purificación por afinidad y espectrometría de masas. En total se obtuvieron 14586 proteínas interactuantes y 127732 interacciones que conforman la red[24]. Esta red la descargamos de la base de datos NDEx[25]. Nos referimos a la red como red de comunicación celular 3 (CC3).

3.1.3.4. *Human gene regulatory network of mesothelioma*

Mesotelioma es un raro tipo de cáncer humano el cual emerge en las células mesoteliales que cubren ciertas partes del cuerpo. Uno de los sitios más comunes es en la pleura. La red es generada por estudios TCGA RNA-seq de mesotelioma usando la base GeneRep[26]. La red consta de 17490 nodos y 631556 aristas . Descargamos la red de la base de datos NDEx[27]. En este estudio nos referiremos a esta red como red de comunicación celular 4 (CC4).

3.1.3.5. *Human gene regulatory network of adrenocortical carcinoma*

Cáncer adrenocortical es una rara enfermedad que es muy difícil de pronosticar. Los progresos para hallar biomarcadores genéticos para el diagnóstico y tratamiento se enfocan en identificar genes asociados a ciertas características médicas en una red compleja[28, 26]. La red se generó mediante estudios TCGA RNA-seq. La red se compone de 16859 nodos y 713649 aristas.

Descargamos la red de la base de datos NDEx[29]. En este estudio nos referimos a esta red como red de comunicación celular 5 (CC5).

3.1.3.6. *The basal gene network in the involvement of respiratory viruses infection including SARS-CoV-2*

Se presenta una red[30] de procesos regulatorios en respuesta a SARS-CoV-2 y otros virus respiratorios usando un modelo de red Bayesiana. La red muestra que la señalización ïnterferon"gradualmente cambia a las cascadas subsecuentes de los procesos inflamatorios-citoscínicos. La red contiene 121264 aristas y 15258 nodos. Descargamos la red de la base de datos NDEx[31]. En este estudio nos referimos a la red como red de comunicación celular 6 (CC6).

3.1.3.7. *Regulon HNSC Head and Neck Squamous Cell Carcinoma*

La red se construyó mediante el uso de ARACNe, un algoritmo que usa las expresiones e información de expedientes y perfiles médicos para diseñar redes de procesos regulatorios en células[26]. La red cuenta con 19566 nodos y 418066 interacciones entre biomoléculas. Descargamos esta red de la plataforma NDEx[32]. Nos referimos a la red como red de comunicación celular 7 (CC7).

3.1.3.8. *Human Gene Regulatory Network of Ovarian Serous Cystadenocarcinoma*

El Cistadenocarcinoma de ovarios es uno de los cánceres ginecológicos más malignos. La red asociada a los procesos celulares de este tipo de cáncer se desarrolló mediante TCGA RNA-seq[26]. La red posee 17893 nodos que que se unen mediante 495894 aristas. Descargamos la red de la plataforma NDEx[33]. En este estudio nos referimos a la red como red de comunicación celular (CC8).

3.1.3.9. *Pathways Affected by Melanoma Genes*

La información biológica de esta red se desarrolló gracias a la MGDB (Melanoma Gene Database). Se estableció una red de genes en los procesos de células melánicas cancerígenas para el estudio del desarrollo a una fase de tumores[34]. La red posee 826 nodos y 2953 aristas. Descargamos la red de la plataforma NDEx[35]. En este estudio nos referimos a la red como red de comunicación celular 9 (CC9).

3.2. Redes BA

Construimos la subrutina `BA_net`, anexos 6.1, para replicar redes tipo Barabási-Albert. Esta subrutina toma como argumentos el número de nodos iniciales n_0, número de nodos finales n y número de aristas por nodo añadido m, como se explicó en 2.2.12. La red inicial es una red desconectada de n_0 nodos, donde utilizamos $n_0 = m$. Creamos m aristas entre el nodo $n_0 + 1$ y los n_0 nodos de la red inicial. Creamos un conjunto w_k, donde añadimos todos los nodos que unimos con una arista. Es decir, como creamos m aristas entre los nodos $1, 2, \ldots, n_0$ y $n_0 + 1$, añadimos los nodos 1 hasta n_0 y m veces el nodo $n_0 + 1$ al conjunto w_k. El conjunto w_k contiene los nodos de la red y estos se repiten cuantas veces sea su grado.

A partir del nodo $n_0 + 2$ hasta n se realiza el proceso iterativamente. Se escogen aleatoriamente m nodos diferentes que se encuentren en w_k con la subrutina `sample`. Aquí estamos utilizando el criterio de enlace preferencial, porque al escoger un nodo i que se enlace al nuevo

nodo añadido de la red del conjunto w_k la probabilidad de que i sea escogido, depende del grado de i. Una vez escogidos los enlaces, se guarda la información de las nuevas aristas en cada paso y se agregan los nodos enlazados a w_k también.

La subrutina retorna el conjunto las aristas de la red. El número de aristas finales es $|E| = (n - n_0)m$.

3.3. Redes

3.3.1. Preparación de redes

Antes de calcular las medidas topológicas preparamos los ficheros que contienen las redes descritas anteriormente. Este proceso lo realizamos en el lenguaje de programación Python, el cual posee librerías que facilitan la lectura de la información y la extracción de los datos necesarios. Las subrutinas empleadas para este proceso se encuentran en la sección de Anexos 6.3.1.

Importamos los datos de los ficheros tipo '.cx' con la subrutina `import_net` y retorna dos tablas de datos, la primera con la lista de nodos y sus respectivos nombres de las moléculas y la segunda con la lista de aristas.

En muchos casos las redes pueden contener aristas repetidas o aristas que unen un nodo consigo mismo. Ya que trabajamos con redes simples no direccionadas eliminamos este tipo de aristas de las redes, en el caso de haberlas, con la subrutina `filtering edges`.

En algunas redes descargadas existen nodos aislados que no se encuentran unidos con ninguna arista. También es posible que la numeración de los nodos se salte algunos números. Para resolver estos problemas y que los algoritmos se puedan correr con mayor efectividad numeramos a los nodos de 0 hasta $|V| - 1$ con las subrutinas `di_nodesid` y `change_id`.

Después de los procesos anteriores, la subrutina `export_data` exporta dos ficheros llamados 'red_r.dat' y 'nodes.dat' . Este es el formato de entrada para el programa que calcula de medidas topológicas.

3.3.2. Cálculo de medidas

En este trabajo utilizamos el método de tomografía de redes 2.2.3 y listas enlazadas 6.2.1 para el cálculo de las medidas topológicas. Las subrutinas utilizadas para este cálculo se encuentran en la sección de Anexos en la sección 6.2.2.5.

El análisis se concentra en el componente gigante de la red. Utilizamos la subrutina `components` para hallar el componente gigante a partir de un fichero que contiene las aristas de una red dada. En este algoritmo construimos la primera capa 2.1 de cada uno de los nodos a partir de las aristas de la red. Después escogemos aleatoriamente un nodo i de la red y construimos iterativamente desde la capa $\kappa_2(i)$ hasta la capa máxima $\kappa_{l_{m_i}}(i)$. Para la construcción de estas capas utilizamos la ecuación 2.7 de la forma:

$$\kappa_l(i) = \bigcup_{k \in \kappa_{l-1}(i)} \kappa_1(k) - \bigcup_{n<l} \kappa_n(i). \tag{3.1}$$

Una vez finalizada la construcción de las capas, obtenemos los nodos que pertenecen al componente de i:

$$V'(i) = \bigcup_{l \leq l_{m_i}} \kappa_l(i). \tag{3.2}$$

Si $V'(i)$ comprende la mayor parte de nodos de la red, hallamos el componente gigante $G' = (V', E')$. Aquí, E' son las aristas que conectan unicamente nodos de V'. Si $V'(i)$ comprende solo unos pocos nodos de la red, se repite el proceso escogiendo otro nodo. A partir de aquí, cuando nos referimos a la red $G = (V, E)$ hablamos del componente gigante.

Construimos las capas para cada uno de los nodos de la red siguiendo el proceso descrito en anteriores párrafos. Utilizamos la subrutina `construct_kapa` la cual exporta un array donde cada elemento elemento i del array es el conjunto de capas del nodo i.

Para el cálculo de longitudes de camino utilizamos la subrutina `caminos`. Encontramos la centralidad closeness 2.14 con el número de elementos de cada capa:

$$C_{clo}(i) = \left(\sum_{l=1}^{l_{m_i}} l|\kappa_l(i)| \right)^{-1}. \tag{3.3}$$

Una vez calculada esta medida, encontramos el SAPL 2.9 de la red:

$$\bar{l} = \frac{1}{|V|(|V|-1)} \sum_{i \in V} \frac{1}{C_{clo}(i)}. \tag{3.4}$$

Se obtiene $C(i)$ de la ecuación 2.4 con la subrutina `clusteringc`. Se calcula el grado de los nodos contando el número de vecinos en la capa 1 para cada nodo con la subrutina `degree`.

Podemos normalizar a la centralidad $C_{clo}(i)$ para todos los nodos $i \in V$. La longitud mínima de camino $L(i, j)$ para $j \neq i$ como mínimo puede ser 1 en el caso de que i y j sean primeros vecinos. Como máximo la longitud puede ser el valor de la excentricidad del nodo i. La excentricidad más grande posible que puede poseer un nodo cualquiera en una red es$|V| - 1$, en el caso de que la red sea una cadena. Entonces se cumple la desigualdad:

$$1 \leq L(i, j) \leq |V| - 1 \qquad \forall i \neq j \in V. \tag{3.5}$$

Sumamos sobre todos los nodos j de la red:

$$\sum_{j \neq i}^{|V|} 1 \leq \sum_{j \neq i}^{|V|} L(i, j) \leq \sum_{j \neq i}^{|V|} (|V| - 1) \qquad \forall i \in V. \tag{3.6}$$

Tomamos el inverso y notamos que el término acotado entre las desigualdades es 2.14. Por tanto:

$$\frac{1}{|V|(|V|-1)} \leq C_{clo}(i) \leq \frac{1}{|V|} \qquad \forall i \in V. \tag{3.7}$$

Podemos acotar de forma general a la centralidad *closeness* para una red en función de su número de nodos. Definimos una medida normalizada:

$$C'_{clo}(i) = \frac{C_{clo}(i) - m}{M - m}, \tag{3.8}$$

con $m = \frac{1}{|V|(|V|-1)}$ y $M = \frac{1}{|V|}$. Ya que esta normalización es lineal, se preservan sus momentos estadísticos centrados y normalizados. Con esto compararemos la centralidad entre redes.

Capítulo 4

RESULTADOS Y DISCUSIÓN

4.1. Redes Barabási-Albert

Empezamos esta sección ejecutando los algoritmos para el cálculo de las medidas topológicas sobre redes de Barabási y Albert y validamos su correcto funcionamiento. Reproducimos los resultados conocidos de las redes Barabási y Albert descritos en el marco teórico.

Ejecutamos el cálculo del degree sequence sobre las redes BA. Elegimos cuatro redes con un número de nodos $|V| = \{10^3, 10^4, 10^5, 10^6\}$ con $m = 2$ y $m = 3$. Comparamos con la función teórica, dada por la ecuación 2.22.

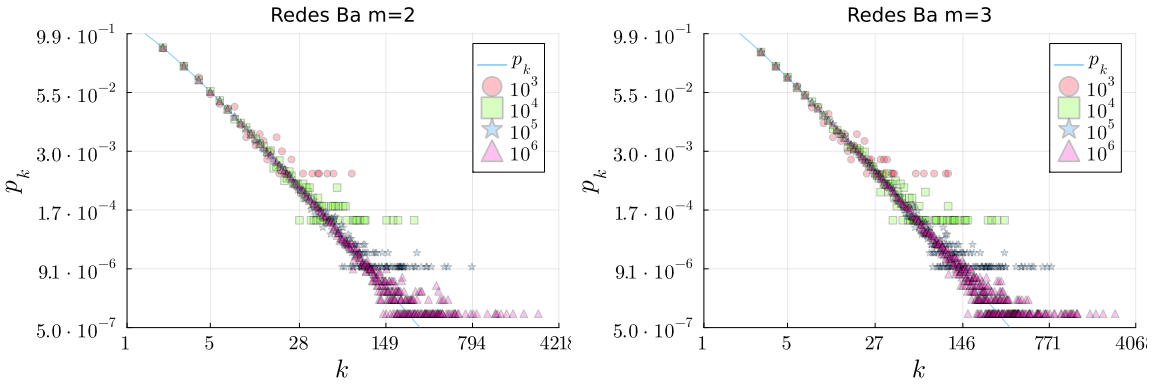

Figura 4.1: Degree sequence para redes con $|V| = \{10^3, 10^4, 10^5, 10^6\}$ y $m = 3$ a la derecha y $m = 2$ a la izquierda. Se muestra en escala log-log para apreciar el comportamiento de ley de potencias.

El parámetro m en la función de distribución de grados 2.22 no cambia el comportamiento de la pendiente en la gráfica 4.1, solo traslada a la función. Observamos que en las dos gráficas de m diferente las redes se ajustan a la ley de potencias con $\gamma = 3$. El ajuste es más confiable mientras $|V|$ crece. Esto es lo esperado, ya que la función teórica es un límite asintótico de $|V| \to \infty$.

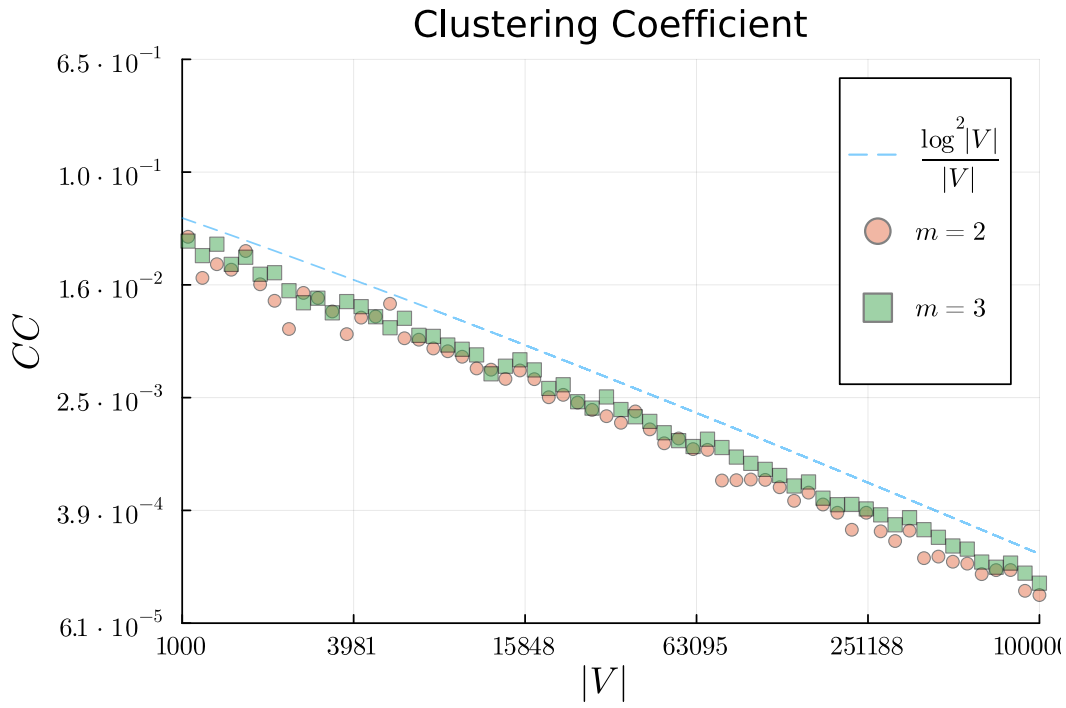

Figura 4.2: Gráfico del clustering coefficient en función de los nodos para 60 redes simuladas en escala log-log. Se muestra también la función teórica 2.24,que es una función asintótica.

En el gráfico 4.2 se observa el comportamiento decreciente del clustering coefficient de las redes conforme el número de nodos de la red crece. Notamos también que el comportamiento no es diferente para los valores de m analizados. Aquí también podemos observar que los valores tienden a la función teórica asintóticamente por lo que es normal que los datos no topen a la función.

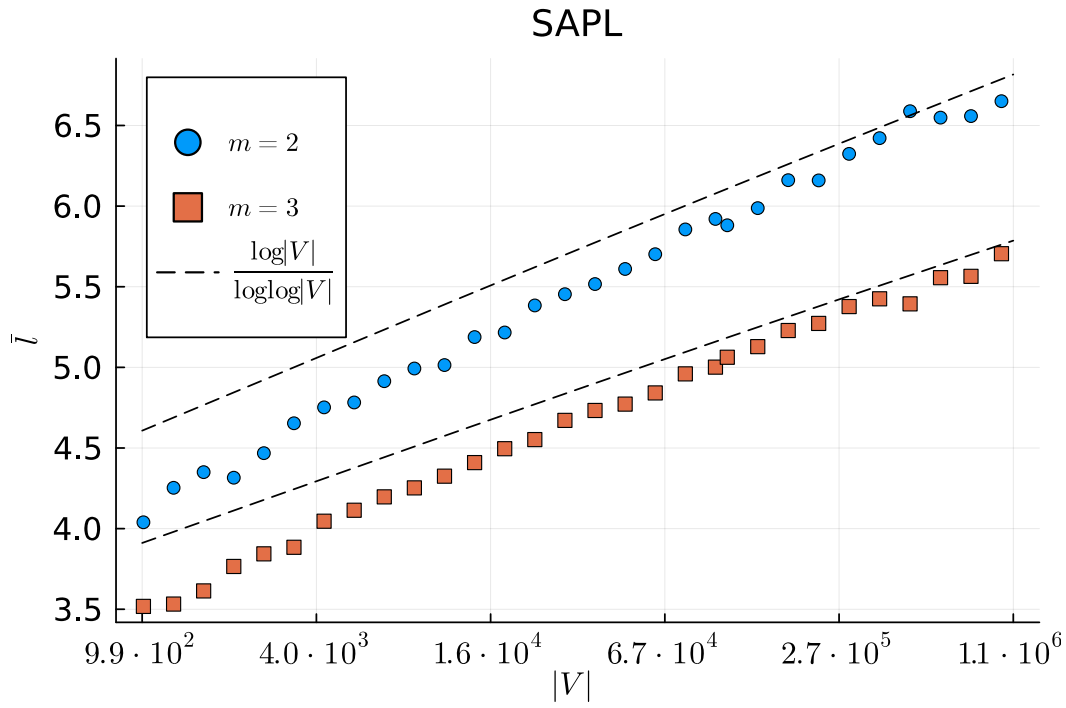

Figura 4.3: Gráfico de SAPL en función del número de nodos de 20 redes BA en escala semilogarítmica. Mostramos también la función teórica 2.23.

En el gráfico 4.3 notamos que el SAPL de las redes se acerca asintóticamente a la función teórica. Observamos también que para redes BA con el mismo número de nodos el SAPL disminuye para redes con m mayor.

En base al cálculo de estas tres medidas topológicas, podemos concluir que la implementación de los algoritmos descritos en el capítulo 3 es correcta.

Ejecutamos los algoritmos que calculan las medidas de centralidad closeness y la matriz de correlación entre nodos para comparar las características de las redes BA con el comportamiento en estas medidas de las redes de comunicación celular. Mostramos a continuación los resultados obtenidos.

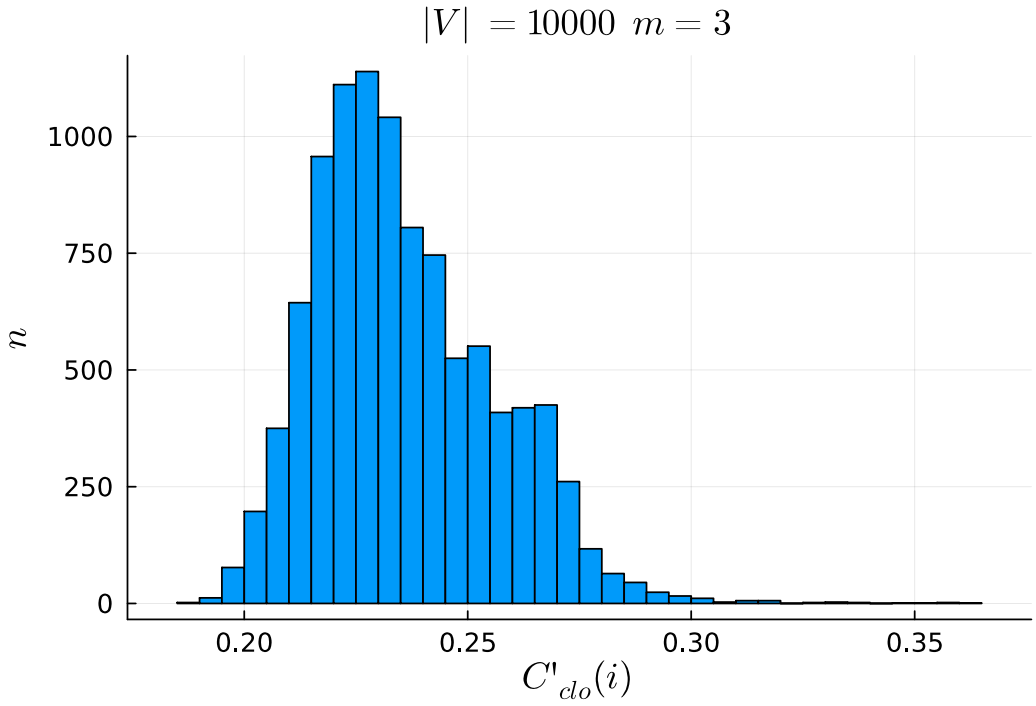

Figura 4.4: Histograma de frecuencias de la centralidad de un nodo, sobre los nodos de la red BA de parámetros $|V| = 10000$ y $m = 3$.

En la gráfica 4.4 mostramos el histograma de la centralidad closeness de los nodos de la red de BA de 10000 nodos y $m = 3$. Podemos notar que la mayoría de datos se encuentran agrupados alrededor de la media $\mu_1 = 0.237$. Observamos también que esta distribución es asimétrica hacia la izquierda con una cola hacia la derecha más larga que la de la izquierda.

Por la alta concentración de los datos alrededor de la media, concluimos que la mayoría de nodos en la red posee una centralidad similar. La asimetría de la distribución hacia la izquierda nos dice que hay un poco porcentaje de nodos que se encuentran en la periferia de la red.

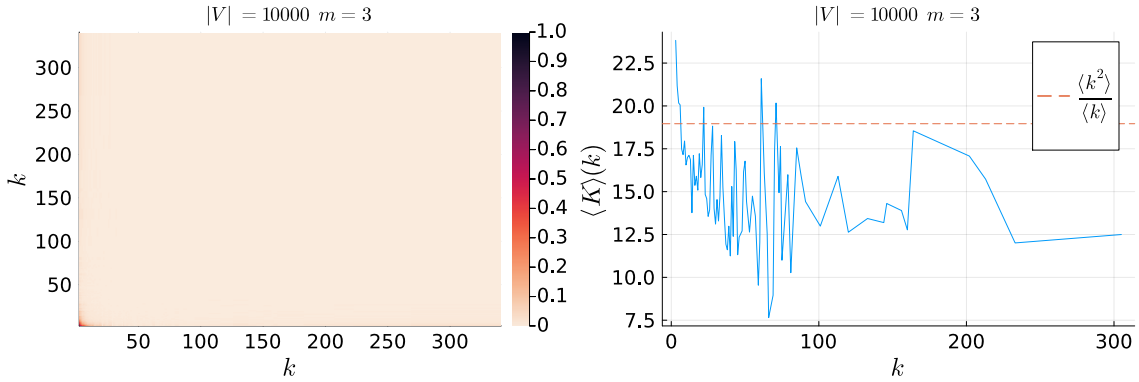

Figura 4.5: A la izquierda una gráfica de calor de la matriz de probabilidades 2.16. Esta matriz es muy dispersa ya que los elementos son cero, para la mayoría de elementos. Los valores relevantes se encuentran en los espacios de grado bajo $k \leq 25$. A la derecha se encuentra la gráfica del grado promedio de un nodo al que se conecta un nodo de grado k, también se muestra la constante que describe el comportamiento de una red no correlacionada.

La matriz de correlación entre grados e_{ij} nos indica que los nodos se encuentran estrechamente conectados con nodos de grado muy bajo, lo que se espera para este modelo. Los nodos de grado bajo, al contrario, se conectan con los nodos de mayor grado por lo cual esta matriz nos indica una tendencia disortativa.

El coeficiente de correlación de Newman 2.19 para esta red es $r = -0.044$. El valor negativo, pero cercano a cero, nos indica una tendencia ligeramente disortativa, como pudimos constatar en la matriz.

En la figura derecha de 4.5 se muestra una tendencia decreciente para nodos de grado bajo. El valor máximo que observamos es de $k \approx 25$. Un comportamiento esperado porque las redes BA poseen pocos nodos hub con alto grado, pero muchos nodos de bajo grado, por lo que en promedio resulta en un valor de $\langle K \rangle$ bajo.

4.2. Redes de comunicación celular

4.2.1. Degree sequence

Calculamos el degree sequence para cada proceso de comunicación celular y sus respectivas leyes de potencia:

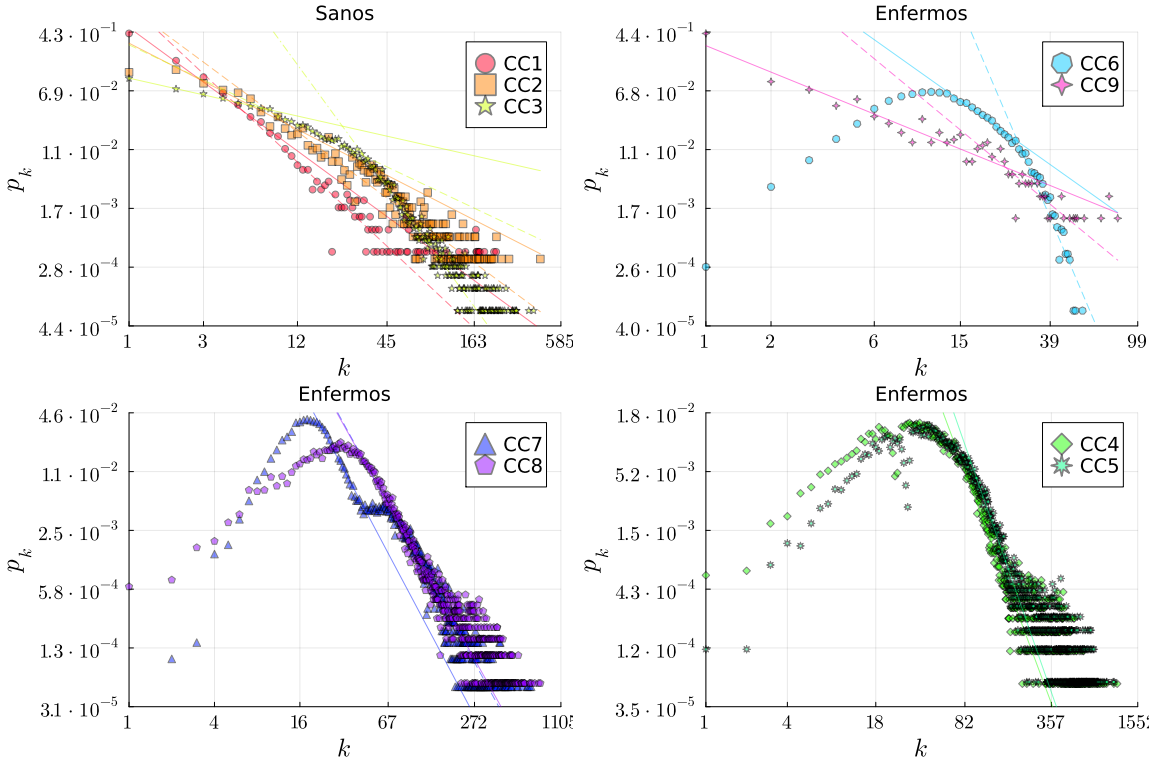

Figura 4.6: Gráficas de degree sequence para las redes de comunicación celular en escala log-log. En la gráfica superior izquierda, se grafican los procesos regulares de comunicación celular y en las gráficas derecha superior e inferiores, se encuentran los procesos asociados a enfermedades. Se muestran también los ajustes de las leyes de potencias asociadas a cada proceso con su respectivo color.

En cada uno de los procesos de la figura 4.6 se calcularon los ajustes mediante regresiones lineales para hallar los exponentes de grado. En algunos de los casos existen varias leyes de potencia que se solapan. Los exponentes de grado se muestran en la siguiente tabla:

ID	k	γ	ρ^2
1	$[1, 10]$	1.552	0.988
	$[5, 50]$	1.99	0.889
2	$[2, 9]$	1.077	0.817
	$[6, 110]$	1.57	0.86
3	$[1, 8]$	0.477	0.994
	$[9, 30]$	1.002	0.961
	$[30, 100]$	2.9	0.933
4	$[60, 200]$	3.351	0.938
5	$[100, 220]$	3.572	0.889
6	$[13, 25]$	2.108	0.979
	$[24, 55]$	6.969	0.914
7	$[20, 50]$	2.884	0.975
	$[60, 180]$	2.857	0.881
8	$[36, 200]$	2.739	0.953
9	$[1, 14]$	1.206	0.923
	$[18, 50]$	2.418	0.783

Cuadro 4.1: Exponentes de grado con su dominio de ajuste y su coeficiente de correlación lineal para cada una de las redes de comunicación celular.

Notamos que en todas las redes de procesos enfermos, salvo la red 7, hay una tendencia creciente para grados bajos, alcanza un máximo y después el una tendencia decreciente donde ajustamos las leyes de potencia. En los procesos sanos únicamente observamos las tendencias decrecientes.

Este diferencia entre los procesos parece indicar que en los procesos sanos abundan las cadenas lineales, ya que la mayoría de nodos poseen un grado 2. También existe un alto porcentaje de nodos de grado ligeramente mayor a dos, esto parece mostrar que en general no existen muchas bifurcaciones en los procesos sanos. Por el contrario, en los procesos relacionados con enfermedades, los resultados pueden indicarnos que hay una mayor proporción de bifurcaciones que cadenas lineales.

Observamos en la tabla 4.1 que hay una buena correlación entre las leyes de potencia estimadas y los datos. Notamos también que en las redes de comunicación de procesos normales, los exponentes de grado son bajos y menores que el exponente de grado en el degree sequence de una red BA.

Para las redes de procesos enfermos, podemos notar que hay varios exponentes que son mayores que 3. Encontramos el mayor de los exponentes de grado en la red 6 con un valor de $\gamma = 6.969$. Este valor es mayor al doble del exponente en una red BA, lo que indica una caída muy rápida. Los valores altos de exponente de grado en las redes enfermas parecen indicarnos una menor proporción de nodos hub en comparación con los procesos sanos.

4.2.2. Clustering Coefficient

Calculamos el clustering coefficient de cada una de las redes de comunicación celular y comparamos con los valores de clustering coefficient de redes BA con un número de nodos y aristas similares a las redes de comunicación. Los resultados se muestran en la siguiente tabla:

	Comunicación C		**BA**		
ID	CC	δCC	CC	δCC	$\frac{\Delta CC}{CC_{ba}}$
1	0.115	0.239	0.017	0.064	5.797
2	0.538	0.33	0.028	0.026	18.22
3	0.116	0.195	0.007	0.013	14.534
4	0.09	0.064	0.018	0.007	4.003
5	0.087	0.062	0.021	0.008	3.168
6	0.137	0.079	0.007	0.014	18.513
7	0.036	0.039	0.011	0.008	2.304
8	0.141	0.104	0.014	0.008	8.77
9	0.0	0.0	0.042	0.071	1.0

Cuadro 4.2: Clustering Coefficient de las redes de comunicación celular con su desviación. Se muestran también los valores de clustering coefficient en redes BA que poseen aproximadamente el mismo número de nodos y aristas que las redes de comunicación celular.

Reproducimos los resultados del clustering coefficient de redes biológicas[4, 7, 5]. Los valores del clustering coefficient de redes de comunicación son mucho mayores que las redes BA de tamaño similar. En el cuadro 4.2 observamos que el valor de los coeficientes de redes de comunicación son al menos 2 veces mayores y hasta 18 veces.

Estos resultados nos indican que las redes de comunicación celular poseen nodos con una

mayor densidad de aristas locales que las redes del modelo Barabási-Albert. Los procesos de comunicación tienen un alto agrupamiento en promedio. Sin embargo, esta medida no muestra una diferencia relevante entre los procesos sanos y procesos enfermos de comunicación celular.

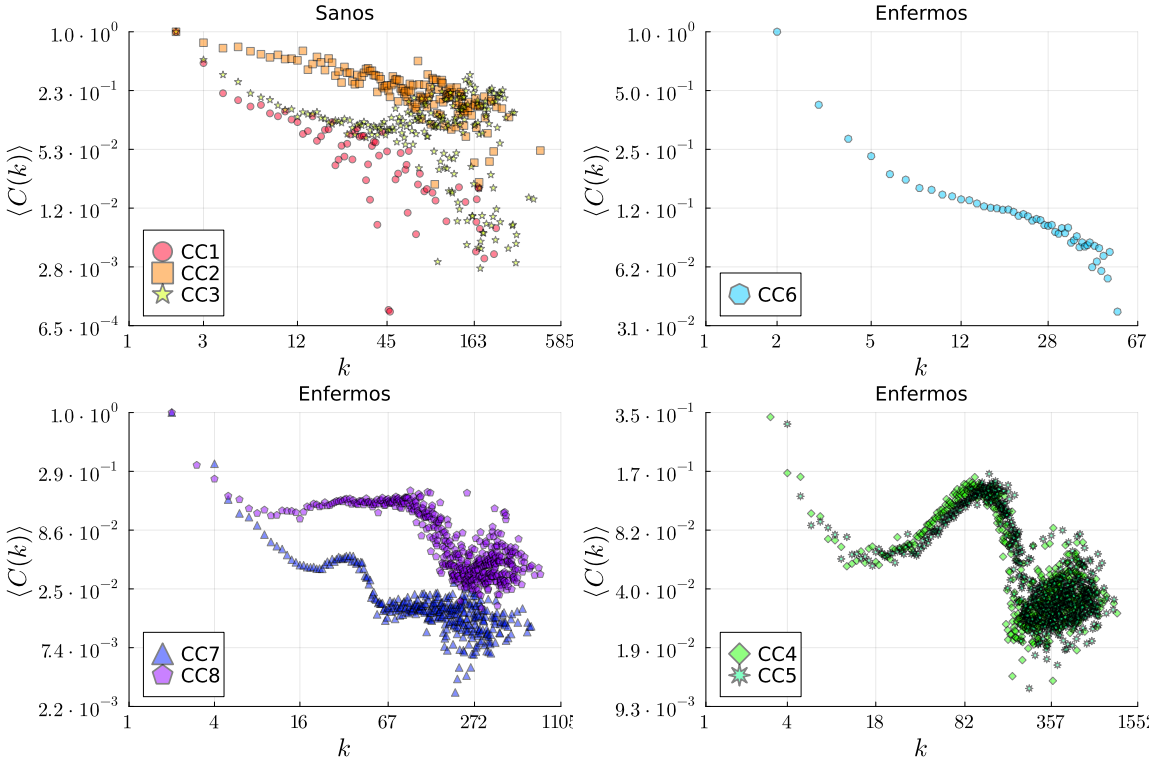

Figura 4.7: Gráfico de dispersión de los coeficientes locales $C(k)$ en función de sus grados en escala semilogarítmica. En la figura superior izquierda se muestran los procesos de comunicación regulares y en la figura superior derecha y figuras inferiores se muestran los procesos asociados a enfermedades.

Podemos examinar más a fondo la estructura de la red calculando el valor promedio del coeficiente de agrupamiento $\langle C(k) \rangle$ para los nodos con grado k. En la figura 4.7, en el gráfico de los procesos sanos y el gráfico de la red 6, observamos una tendencia decreciente. También observamos que estos coeficientes alcanzan valores altos.

En los procesos asociados a enfermedades notamos que existe un pico en el valor del coeficiente local. Los máximos de estos coeficientes en función de k se dan para valores similares a los que ocurren los máximos en el degree sequence. El alto agrupamiento que indica el pico, nos dice que hay una densidad de aristas concentrada entre nodos de estos grados. Estos resultados no indican algo concreto sobre la biología de los procesos, pero si nos incentivan a realizar nuevos análisis enfocados en nodos con grado en los valores donde alcanzan los máximos de las redes de procesos enfermos.

Observamos en la tabla 4.2 que tanto el promedio como la desviación de los clustering coefficient de cada nodo es cero en la red 6. Esto indica que no existe ningún triángulo 2.3 dentro de toda la red. Analizamos para esta red la distribución de sus aristas, en función de qué tipo de molécula unen, con el propósito de aclarar este comportamiento en la medida topológica del clustering coefficient.

Asociamos cada nodo con el tipo de molécula encontrada en la base de datos y contamos las aristas que unen a cada tipo de nodo. Obtenemos los resultados de la matriz de correlación

entre nodos:

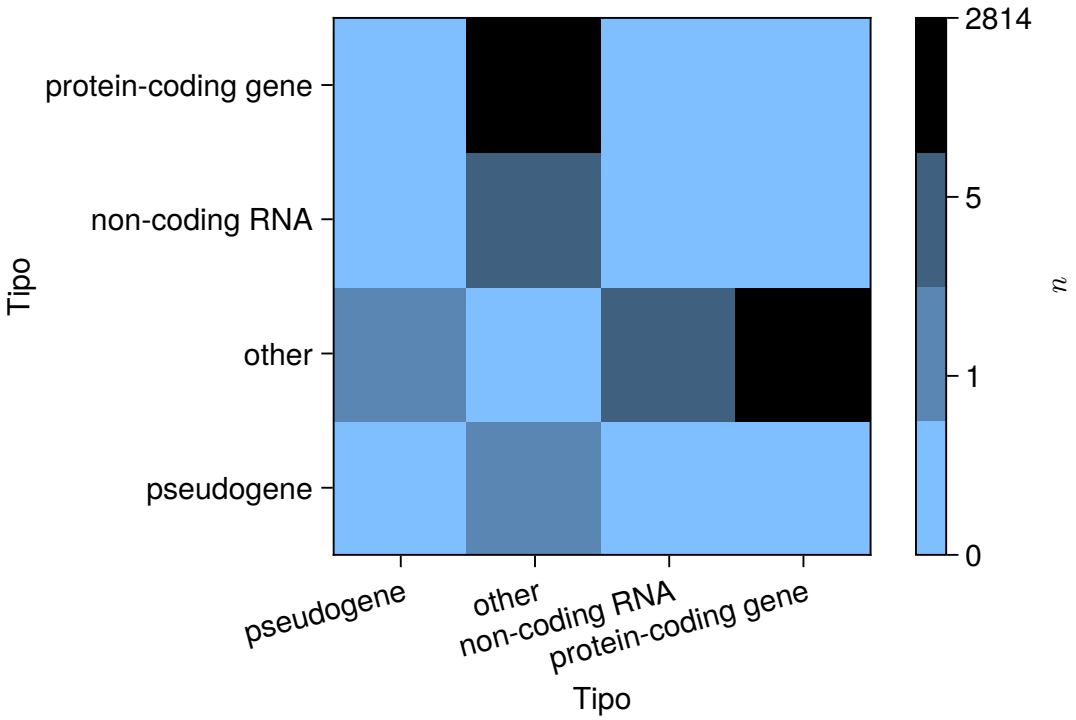

Figura 4.8: Mapa de calor de la matriz de correlación para enlaces entre tipos de molécula de la red de comunicación celular 6. Se muestran las 4 categorías obtenidas.

En la gráfica 4.8 observamos que podemos separar a los grupos de nodos en dos tipos: el primero son los tipos pseudogene, non-coding RNA y protein-code gene; en el segundo la categoría others. Esta última categoría son las moléculas interactuantes que no se encontraron en la base de datos HGNC. Este resultado podría indicar que estamos tratando con una red que posee reacciones enzimáticas[7] y en este caso la categoría others estaría compuesta por enzimas.

El hecho de que el conjunto de todos los nodos en la red se particione en dos conjuntos, tal que todas las aristas de la red conectan a los nodos entre estos dos conjuntos, nos dice que la red 6 es una red bipartita. Esto explica que en toda la red no existan triángulos ya que no hay aristas que unan a nodos del mismo tipo.

4.2.3. SAPL y longitudes de camino

Calculamos el SAPL de cada red de comunicación celular y de las redes BA de tamaños similares. Se presenta esta comparación en la siguiente tabla:

	Comunicación C			BA			%
ID	$\bar{l}$	D	R	$\bar{l}$	D	R	$\frac{\Delta l}{l_{ba}}$
1	3.719	11	6	3.806	6	4	2.29
2	3.394	9	5	2.84	4	3	19.51
3	3.797	10	6	3.243	5	3	17.08
4	2.7	5	4	2.624	4	3	2.9
5	2.596	5	3	2.536	3	2	2.37
6	4.771	10	6	3.35	5	3	42.42
7	2.908	5	3	2.842	4	3	2.32
8	2.85	5	4	2.747	4	3	3.75
9	3.76	9	5	3.11	5	3	20.9

Cuadro 4.3: Se muestran los valores de SAPL, diámetro y radio de las redes para las redes de comunicación celular y las redes BA. Se muestran también las variaciones porcentuales del SAPL con respecto a las redes BA.

Según los valores de la tabla 4.3, las redes de comunicación celular poseen un valor mayor en SAPL, diámetro y radio que las redes BA. Sin embargo, la diferencia de valores en el SAPL, la diferencia no es sustancial, salvo para la red 6 el cambio porcentual alcanza el 42.42 %.

Estos resultados parecen indicarnos que las redes de comunicación celular poseen un carácter *small world*[36] como las redes BA.

4.2.4. Closeness

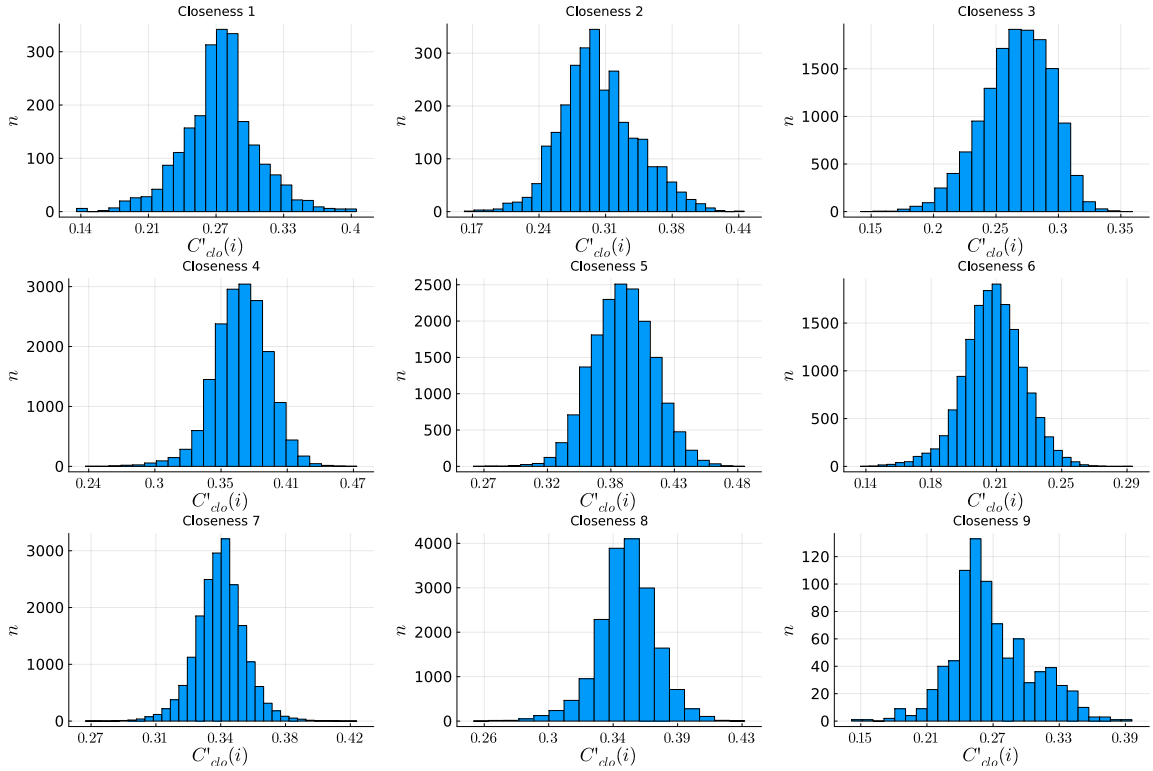

Figura 4.9: Histogramas de frecuencia de centralidad closeness normalizada 3.8 de cada una de las redes de comunicación celular.

Podemos notar en la figura 4.9 una ligera asimetría en la centralidad closeness normalizada hacia la derecha en las redes de comunicación celular, salvo en las redes 2 y 9. La asimetría a la derecha nos dice que hay una mayor proporción de nodos centrales que periféricos en la red. La asimetría hacia la izquierda nos indica lo contrario.

De esta medida no podemos notar diferencias específicas entre procesos de comunicación celular regulares y enfermos. Para obtener más información obtenemos la centralidad promedio entre nodos de grado k.

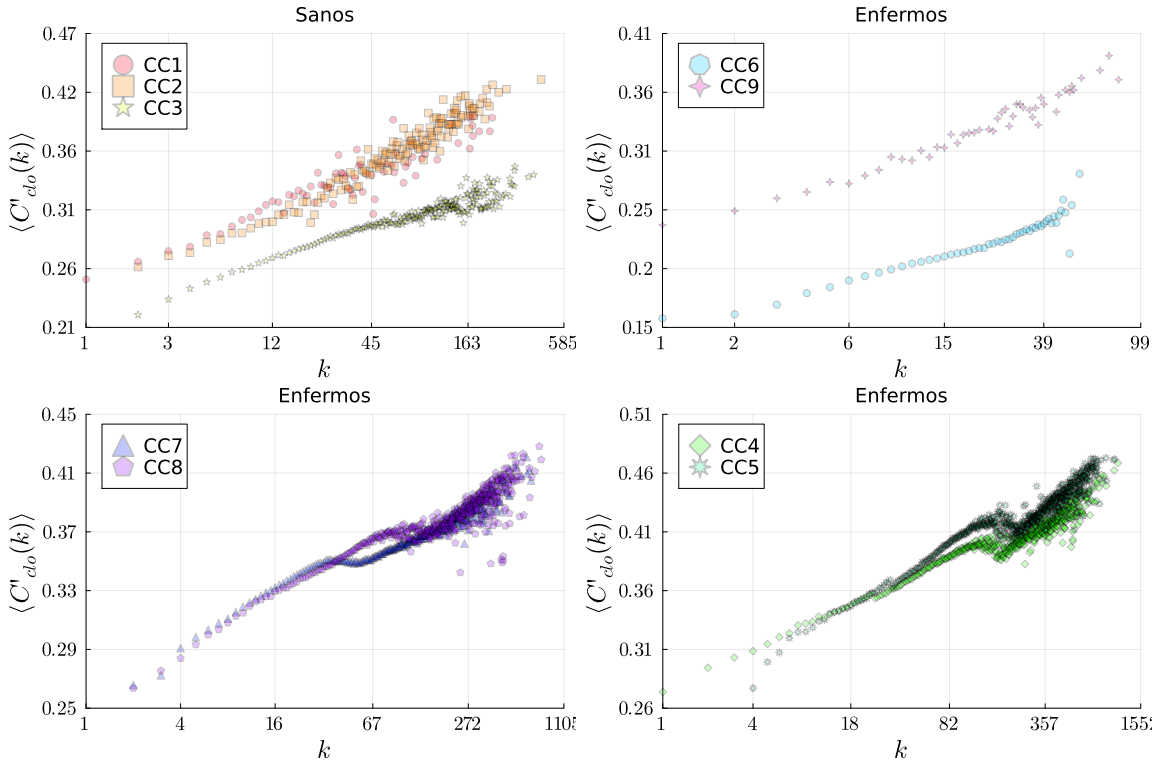

Figura 4.10: Gráfico de centralidad closeness local $\langle C'_{clo}(k)\rangle$ en función de sus grados en escala semilogarítmica. En la figura superior izquierda se muestran los procesos de comunicación normales y en las figuras restantes los procesos asociados a enfermedades.

La figura 4.10 nos muestra una relación positiva entre la medida de centralidad closeness y el grado de un nodo. Podemos observar que los nodos hub en todas las redes también son los nodos más centrales. Esta correlación resalta la importancia de los nodos hub en las redes para mantener la conectividad de la red.

En las redes de las figuras inferiores de 4.10 podemos observar el pico de la centralidad closeness en grados similares en los que ocurren los picos en las medidas degree sequence y local clustering coefficient de los procesos enfermos. Esta diferencia nos dice que para nodos con grados alrededor del pico son más centrales que los nodos con grados cercanos. Este resultado es lógico también por el comportamiento observado en la medida de local clustering coefficient.

4.3. Disortatividad

Para caracterizar esta propiedad calculamos los coeficientes de correlación de Newman 2.19 de las matrices de correlación grado-grado e_{ij} 2.16. Se presentan estos resultados en la siguiente

tabla:

CC	1	2	3	4	5	6	7	8	9
r	-0.328	-0.02	0.025	-0.202	-0.207	0.169	-0.233	-0.175	-0.055

Cuadro 4.4: Coeficientes de correlación de Newman para las redes de comunicación celular.

Observamos que los valores de los coeficientes r para las redes de comunicación son negativos, a excepción de la red 3 y 6. La red 1 presenta el mayor grado de disortatividad con un valor de $r = -0,328$. Las redes 2 y 3 presentan los valores de r más cercanos a cero, lo que indica un comportamiento más similar a una red sin tendencia assortativa. Podemos profundizar más sobre el comportamiento assortativo o disortativo de las redes si graficamos las matrices e_{ij}:

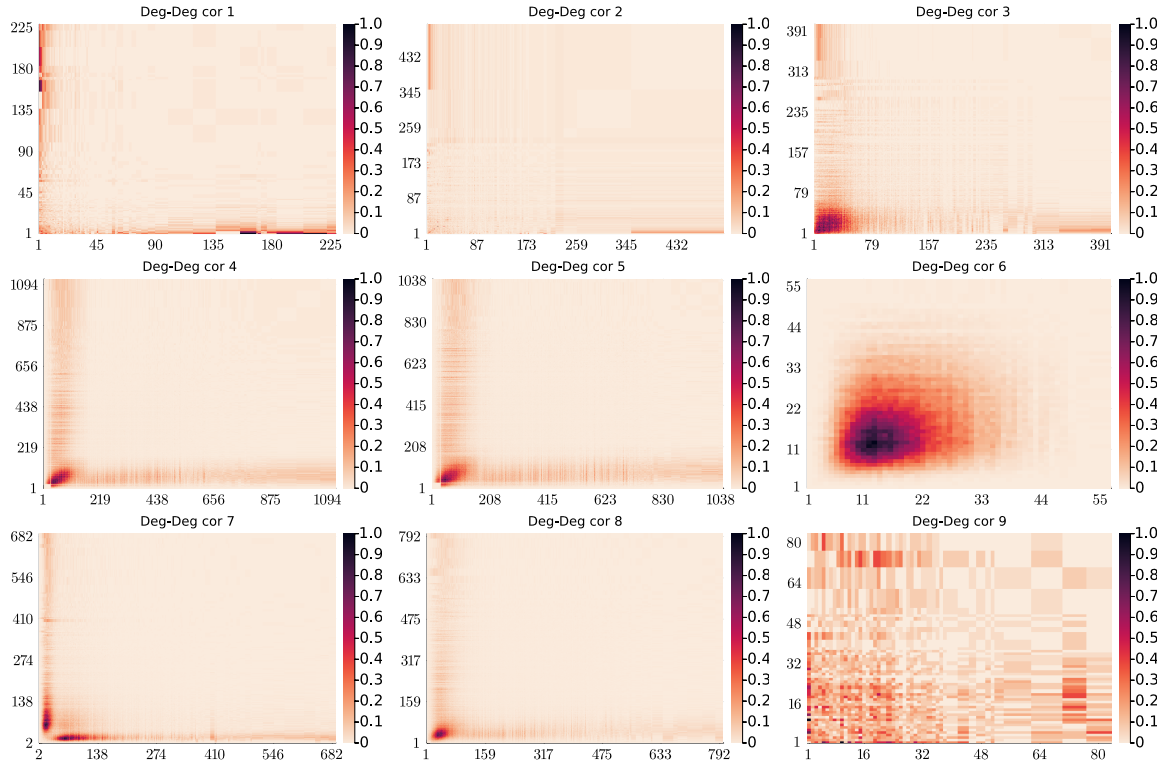

Figura 4.11: Mapa de calor de las matrices e_{ij} de correlación de grados de las redes de comunicación celular.

Observamos en la figura 4.11 para las redes de procesos sanos en la red 1 y 2 una tendencia disortativa. En la red 3 se observa un comportamiento mixto, es decir, los nodos de grado alto tienen una alta probabilidad de conectarse con nodos de grado bajo. Pero, los nodos de grado bajo poseen también una alta probabilidad de conectarse entre sí. Esto explica el coeficiente de correlación cercano a cero para esta red.

Este comportamiento mixto, se puede apreciar también en las redes 4, 5, 7 y 8. En la red 9 podemos observar un comportamiento más desordenado, similar a una red sin asociación, esto puede ser un resultado de que esta red sea bipartita. En la red 6, por el contrario, se aprecia un comportamiento assortativo.

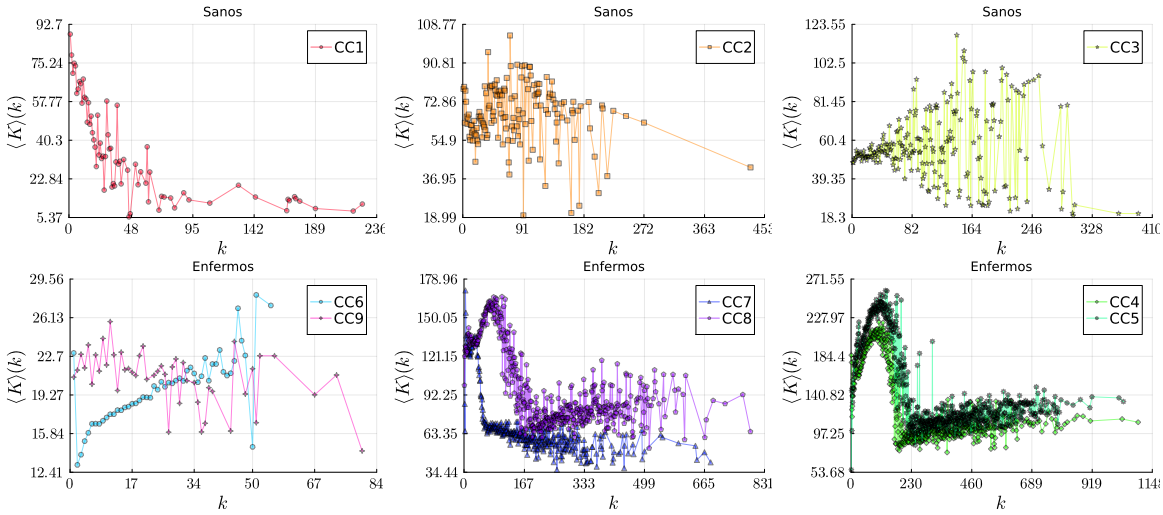

Figura 4.12: Gráfica del promedio de los grados de nodos conectados a un nodo de grado k para las redes de comunicación celular.

Podemos observar en la gráfica 4.12 una tendencia decreciente, esperada en redes disortativas, en la red de comunicación celular 1. En el caso de las redes 2 y 3, que poseen los coeficientes r más cercano a cero, observamos un comportamiento más similar a una red sin tendencia de asociación definida.

En los procesos enfermos, notamos un pico para nodos de grado bajo. Esto es muestra del comportamiento mixto observado en las matrices de la gráfica 4.11. Para la red 6 observamos el comportamiento decreciente para los nodos de menor grado y después una clara tendencia assortativa.

El pico observado en las redes 4,5,7 y 8 se presenta en las mismas regiones de k en las que se presentan los picos en las anteriores medidas estudiadas. Este resultado nos podría indicar que los nodos ubicados cerca de los picos conectan nodos de grado muy bajo con los nodos hub de la red. Estos resultados parecen indicar una posibilidad de que las redes de procesos enfermos presenten un comportamiento jerárquico.

Capítulo 5

CONCLUSIONES

Encontramos diferencias a nivel topológico entre las redes de comunicación celular asociadas a procesos procesos sanos y a procesos enfermos. Estas diferencias se observaron en las medidas de $\langle C(k) \rangle$, degree sequence, centralidad closeness y degree-degree correlation. Los resultados y diferencias obtenidas entre los dos procesos no nos permiten extraer conclusiones biológicamente relevantes.

El degree sequence de las redes muestra un comportamiento asintótico libre de escala en ambos tipos de procesos. Las redes de procesos sanos muestran una tendencia decreciente en esta medida. En las redes de procesos enfermos, se puede observar un pico en la distribución de grados de la red. La comparación de estos resultados nos sugiere que en los procesos sanos predominan las cadenas lineales de reacciones y existen menos bifurcaciones, mientras que en los enfermos existen una mayor cantidad de bifurcaciones en las reacciones.

En la gráfica de $\langle C(k) \rangle$ vs k observamos que en los procesos sanos la tendencia es decreciente. En las redes de procesos enfermos, hay un máximo en los valores del coeficiente para nodos de grado bajo. Este resultado nos permite interpretar que la red posee una densidad de aristas más concentrada cerca de los nodos con estos grados.

En la medida de centralidad closeness no pudimos hallar diferencias relevantes entre los procesos sanos y enfermos, esto porque la distribución de centralidad en los nodos de la red es similar. La medida local de centralidad en función del grado, sin embargo, nos permitió observar un pico alrededor de los nodos con grado similar a los mencionados en las medidas antes descritas en los procesos enfermos. Este resultado apunta a que los nodos con dicho grado son mucho más centrales que los nodos con grados cercanos.

Las matrices de correlación de grados e_{ij} para los procesos enfermos, muestran un comportamiento mixto de assortatividad en cierta región y disortatividad para otras. En los procesos sanos predomina la disortatividad. En las gráficas del promedio de los grados de nodos conectados a nodos de grado k, encontramos nuevamente el pico observado en las otras medidas en los procesos enfermos. Este resultado nos sugiere que los nodos con grado en esta región conectan los pocos nodos hub de la red con los nodos de grado más bajo, un posible comportamiento jerárquico de las redes.

Los resultados obtenidos sobre las redes de procesos enfermos en este análisis preliminar no nos permitieron obtener conclusiones biológicas claras de los procesos. Sin embargo, este análisis nos permitió dilucidar en qué direcciones tomar para realizar nuevos estudios aclaren los resultados biológicos de este tipo de procesos.

Por los picos observados en los nodos de la red en las medidas de degree sequence, local clustering coefficient, local closeness centrality y grado medio de enlace, podríamos preguntarnos la relevancia biológica de las moléculas con grados en estos picos. En base a esto, puede ser útil analizar la distribución de aristas de qué tipos de moléculas conectan a los nodos con estos grados y entender mejor las bifurcaciones en estos procesos. También podemos analizar la medida hierarchical clustering sobre estas redes y así determinar si existe una jerarquía de nodos, con énfasis en los nodos de grado cercano a los picos.

Si las redes analizadas se componen de distintos tipos de moléculas, como la red 6 que es bipartita, podemos encontrar las diferentes sub redes que se forman con cada tipo de molécula. Analizar la topología y la distribución de aristas entre estas sub redes nos puede permitir obtener conclusiones biológicas relevantes de estos procesos y entender el porque de una mayor proporción de bifurcaciones en los procesos enfermos.

BIBLIOGRAFÍA

[1] Bradshaw, Ralph A. y Dennis, Edward A. *Handbook of Cell Signaling*. en. Academic Press, nov. de 2009. ISBN: 9780080920917.

[2] Silverthorn, Dee. *Human Physiology: An Integrated Approach*. English. 8th edition. New York: Pearson, ene. de 2018. ISBN: 9780134605197.

[3] Albert, Réka. «Scale-free networks in cell biology». En: *Journal of Cell Science* 118.21 (nov. de 2005), págs. 4947-4957. ISSN: 0021-9533. DOI: 10.1242/jcs.02714. URL: https://doi.org/10.1242/jcs.02714.

[4] Azeloglu, Evren e Iyengar, Ravi. «Signaling Networks: Information Flow, Computation, and Decision Making». En: *Cold Spring Harbor Perspectives in Biology* 7 (abr. de 2015), a005934. DOI: 10.1101/cshperspect.a005934.

[5] Tavassoly, Iman, Goldfarb, Joseph e Iyengar, Ravi. «Systems biology primer: the basic methods and approaches». eng. En: *Essays in Biochemistry* 62.4 (oct. de 2018), págs. 487-500. ISSN: 1744-1358. DOI: 10.1042/EBC20180003.

[6] Newman, M. E. J. «The Structure and Function of Complex Networks». En: *SIAM Review* 45.2 (ene. de 2003), págs. 167-256. ISSN: 0036-1445. DOI: 10.1137/S003614450342480. URL: https://epubs.siam.org/doi/10.1137/S003614450342480.

[7] Pavlopoulos, Georgios A., Secrier, Maria, Moschopoulos, Charalampos N., Soldatos, Theodoros G., Kossida, Sophia, Aerts, Jan, Schneider, Reinhard y Bagos, Pantelis G. «Using graph theory to analyze biological networks». En: *BioData Mining* 4.1 (abr. de 2011), pág. 10. ISSN: 1756-0381. DOI: 10.1186/1756-0381-4-10. URL: https://doi.org/10.1186/1756-0381-4-10.

[8] Ma, Hong-Wu y Zeng, An-Ping. «The connectivity structure, giant strong component and centrality of metabolic networks». eng. En: *Bioinformatics (Oxford, England)* 19.11 (jul. de 2003), págs. 1423-1430. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg177.

[9] Jalili, Mahdi, Salehzadeh-Yazdi, Ali, Gupta, Shailendra, Wolkenhauer, Olaf, Yaghmaie, Marjan, Resendis-Antonio, Osbaldo y Alimoghaddam, Kamran. «Evolution of Centrality Measurements for the Detection of Essential Proteins in Biological Networks». En: *Frontiers in Physiology* 7 (ago. de 2016), pág. 375. ISSN: 1664-042X. DOI: 10.3389/fphys.2016.00375. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4999434/ (visitado 02-03-2023).

[10] Xulvi-Brunet, R, Pietsch, W y Sokolov, I. «Correlations in Scale-Free Networks: Tomography and Percolation». En: *Physical review. E, Statistical, nonlinear, and soft matter physics* 68 (oct. de 2003), pág. 036119. DOI: 10.1103/PhysRevE.68.036119.

[11] Newman, M. E. J. «Assortative mixing in networks». En: *Physical Review Letters* 89.20 (oct. de 2002). arXiv:cond-mat/0205405, pág. 208701. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.89.208701. URL: http://arxiv.org/abs/cond-mat/0205405 (visitado 20-12-2022).

[12] Xulvi-Brunet, R. y Sokolov, I. M. «Changing Correlations in Networks: Assortativity and Dissortativity». En: *Acta Physica Polonica B* 36 (mayo de 2005). ADS Bibcode: 2005AcPPB..36.1431X, pág. 1431. ISSN: 0587-4254. URL: https://ui.adsabs.harvard.edu/abs/2005AcPPB..36.1431X (visitado 20-12-2022).

[13] Newman, M. E. J. «Mixing patterns in networks». En: *Physical Review E* 67.2 (feb. de 2003). arXiv:cond-mat/0209450, pág. 026126. ISSN: 1063-651X, 1095-3787. DOI: 10.1103/PhysRevE.67.026126. URL: http://arxiv.org/abs/cond-mat/0209450 (visitado 21-12-2022).

[14] Barabási, Albert-László y PÃ3sfai, MÃ¡rton. *Network Science*. en. Google-Books-ID: ZVHesgEACAAJ. Cambridge University Press, jul. de 2016. ISBN: 9781107076266.

[15] Barabási, Albert-László y Albert, Réka. «Emergence of Scaling in Random Networks». En: 286 (1999), págs. 509-512. ISSN: 0036-8075. DOI: 10.1126/science.286.5439.509.

[16] Bollobás, Béla y Riordan, Oliver M. «Mathematical results on scale-free random graphs». en. En: *Handbook of Graphs and Networks*. John Wiley & Sons, Ltd, 2002. Cap. 1, págs. 1-34. ISBN: 9783527602759. DOI: 10.1002/3527602755.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/3527602755.ch1 (visitado 11-12-2022).

[17] Bollobás, Béla y Riordan, Oliver. «The Diameter of a Scale-Free RandomGraph». en. En: *Combinatorica* 24.1 (ene. de 2004), págs. 5-34. ISSN: 1439-6912. DOI: 10.1007/s00493-004-0002-2. URL: https://doi.org/10.1007/s00493-004-0002-2 (visitado 11-12-2022).

[18] *NDEx About*. en. Nov. de 2014. URL: http://www.home.ndexbio.org/about-ndex/ (visitado 11-09-2022).

[19] *Gene Names HGNC About*. URL: https://www.genenames.org/about/.

[20] Perfetto, Livia, Briganti, Leonardo, Calderone, Alberto, Cerquone Perpetuini, Andrea, Iannuccelli, Marta, Langone, Francesca, Licata, Luana, Marinkovic, Milica, Mattioni, Anna, Pavlidou, Theodora, Peluso, Daniele, Petrilli, Lucia Lisa, Pirrò, Stefano, Posca, Daniela, Santonico, Elena, Silvestri, Alessandra, Spada, Filomena, Castagnoli, Luisa y Cesareni, Gianni. «SIGNOR: a database of causal relationships between biological entities». En: *Nucleic Acids Research* 44.D1 (ene. de 2016), págs. D548-D554. ISSN: 0305-1048. DOI: 10.1093/nar/gkv1048. URL: https://doi.org/10.1093/nar/gkv1048 (visitado 15-09-2022).

[21] SIGNOR. *Human Phosphorylation*. Website. URL: https://signor.uniroma2.it/downloads.php#phospho_data.

[22] Schaefer, Carl F., Anthony, Kira, Krupa, Shiva, Buchoff, Jeffrey, Day, Matthew, Hannay, Timo y Buetow, Kenneth H. «PID: the Pathway Interaction Database». En: *Nucleic Acids Research* 37.Database issue (ene. de 2009), págs. D674-D679. ISSN: 0305-1048. DOI: 10.1093/nar/gkn653. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2686461/ (visitado 12-09-2022).

[23] NDEx. *NCI PID - Complete Interactions*. Website. URL: https://www.ndexbio.org/viewer/networks/640e2cef-795d-11e8-a4bf-0ac135e8bacf.

[24] Huttlin, Edward L., Bruckner, Raphael J., Navarrete-Perea, Jose, Cannon, Joe R., Baltier, Kurt, Gebreab, Fana, Gygi, Melanie P., Thornock, Alexandra, Zarraga, Gabriela, Tam, Stanley, Szpyt, John, Gassaway, Brandon M., Panov, Alexandra, Parzen, Hannah, Fu, Sipei, Golbazi, Arvene, Maenpaa, Eila, Stricker, Keegan, Guha Thakurta, Sanjukta, Zhang, Tian, Rad, Ramin, Pan, Joshua, Nusinow, David P., Paulo, Joao A., Schweppe, Devin K., Vaites, Laura Pontano, Harper, J. Wade y Gygi, Steven P. «Dual proteome-scale networks reveal cell-specific remodeling of the human interactome». en.

En: *Cell* 184.11 (mayo de 2021), 3022-3040.e28. ISSN: 0092-8674. DOI: 10.1016/j.cell.2021.04.011. URL: https://www.sciencedirect.com/science/article/pii/S0092867421004463 (visitado 12-09-2022).

[25] NDEx. *BioPlex 3 - HEK293T*. Website. URL: https://www.ndexbio.org/viewer/networks/6b995fc9-2379-11ea-bb65-0ac135e8bacf.

[26] Le, Son B., Riva, Alberto y Tran, David D. « A high-performance pipeline for genome-wide network reconstruction from gene expression data ». En: *F1000Research* 6 (nov. de 2017). DOI: 10.7490/f1000research.1115059.1. URL: https://f1000research.com/posters/6-1979 (visitado 12-09-2022).

[27] NDEx. *Human Gene Regulatory Network of Mesothelioma*. URL: https://www.ndexbio.org/viewer/networks/2c4bc231-dace-11e7-adc1-0ac135e8bacf.

[28] Yuan, Lushun, Qian, Guofeng, Chen, Liang, Wu, Chin-Lee, Dan, Han C., Xiao, Yu y Wang, Xinghuan. «Co-expression Network Analysis of Biomarkers for Adrenocortical Carcinoma». En: *Frontiers in Genetics* 9 (2018). ISSN: 1664-8021. URL: https://www.frontiersin.org/articles/10.3389/fgene.2018.00328 (visitado 12-09-2022).

[29] NDEx. *Human gene regulatory network of adrenocortical carcinoma*. Website. URL: https://www.ndexbio.org/viewer/networks/3bb11a95-dace-11e7-adc1-0ac135e8bacf.

[30] Tanaka, Yoshihisa, Higashihara, Kako, Nakazawa, Mai Adachi, Yamashita, Fumiyoshi, Tamada, Yoshinori y Okuno, Yasushi. *Dynamic change of gene-to-gene regulatory networks in response to SARS-CoV-2 infection*. Inf. téc. ArXiv:2008.09261 [q-bio] type: article. arXiv, ago. de 2020. DOI: 10.48550/arXiv.2008.09261. URL: http://arxiv.org/abs/2008.09261.

[31] NDEx. *The basal gene network in the involvement of respiratory viruses infection including SARS-CoV-2*. Website. URL: https://www.ndexbio.org/viewer/networks/116f99ca-f0fd-11ea-99da-0ac135e8bacf.

[32] NDEx. *Regulon HNSC - Head and Neck Squamous Cell Carcinoma*. Website. URL: https://www.ndexbio.org/viewer/networks/38b31a74-70c5-11e8-a4bf-0ac135e8bacf.

[33] NDEx. *Human Gene Regulatory Network of Ovarian Serous Cystadenocarcinoma*. URL: https://www.ndexbio.org/viewer/networks/1d2a298a-dace-11e7-adc1-0ac135e8bacf.

[34] Zhang, Di, Zhu, Rongrong, Zhang, Hanqian, Zheng, Chun-Hou y Xia, Junfeng. «MGDB: a comprehensive database of genes involved in melanoma». En: *Database: The Journal of Biological Databases and Curation* 2015 (sep. de 2015), bav097. ISSN: 1758-0463. DOI: 10.1093/database/bav097. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4589692/ (visitado 15-09-2022).

[35] NDEx. *Pathways Affected by Melanoma Genes*. URL: https://www.ndexbio.org/viewer/networks/6bde432c-0b40-11e7-aba2-0ac135e8bacf.

[36] Sola Pool, Ithiel de y Kochen, Manfred. «Contacts and influence». en. En: *Social Networks* 1.1 (ene. de 1978), págs. 5-51. ISSN: 0378-8733. DOI: 10.1016/0378-8733(78)90011-4. URL: https://www.sciencedirect.com/science/article/pii/0378873378900114 (visitado 02-01-2023).

Capítulo 6

Anexos

6.1. Modelo Barabási-Albert

```
module Barabasi


    contains

    !ran1
    !   Numeros Aleatorios con semilla idum
    !   Tomada de Fortran Numerical recipes
    !
    real function ran1(idum)

        implicit none

        INTEGER ,INTENT(INOUT) ::idum

        INTEGER ::IA,IM,IQ,IR,NTAB,NDIV
        REAL :: AM,EPS,RNMX
        PARAMETER (IA=16807,IM=2147483647,AM=1./IM,IQ=127773,IR=2836,&
        NTAB=32,NDIV=1+(IM-1)/NTAB,EPS=1.2e-7,RNMX=1.-EPS)
        INTEGER :: j,k,iv(NTAB),iy


        save iv, iy
        DATA iv/NTAB*0/, iy/0/


        if (idum.le.0.or.iy.eq.0) then

            idum=max(-idum,1)

            do j=NTAB+8,1,-1

                k=idum/IQ

                idum=IA*(idum-k*IQ)-IR*k

                if (idum.lt.0) idum=idum+IM

                if (j.le.NTAB) iv(j)=idum

            end do

            iy=iv(1)
```

```
44
45          end if
46
47          k=idum/IQ
48          idum=IA*(idum-k*IQ)-IR*k
49          if (idum.lt.0) idum=idum+IM
50          j=1+iy/NDIV
51          iy=iv(j)
52          iv(j)=idum
53          ran1=min(AM*iy,RNMX)
54          return
55
56      end function
57
58      !rand_int
59      !   Enteros aleatorios entre 0 y n-1
60      !
61      integer function rand_int(idum,n) result(r)
62
63          implicit none
64          integer, intent(inout) :: idum
65          integer, intent(in) :: n
66
67          r=int(ran1(idum)*n)
68
69
70      end function rand_int
71
72      !sample
73      !   Toma una muestra aleatoria de n enteros de un arreglo l1
74      !   sin repetir el valor
75      !
76      function sample(l1,n,idum) result(lr)
77
78          implicit none
79          integer, intent(in)                 :: n,l1(:)
80          integer, intent(inout)              :: idum
81          integer, allocatable,dimension(:)   :: lr
82          logical, allocatable, dimension(:)  :: lp
83          integer                             :: n0, i, j, v_min, v_max
84
85          n0=size(l1,1)
86          v_min=minval(l1)
87          v_max=maxval(l1)
88
89          allocate(lr(n),lp(v_min:v_max))
90
91          lp(:)=.false.
92          i=0
93
94          do
95              if (i.ge.n) exit
96
97              j=rand_int(idum,n0)+1
98
99              if (.not.lp(l1(j))) then
100
101                 lp(l1(j))=.true.
102                 i=i+1
103                 lr(i)=l1(j)
104
105             end if
```

```
106
107
108         end do
109
110         deallocate(lp)
111
112
113     end function sample
114
115     !BA_net
116     !   Retorna el array edges con la lista de aristas de una red BA de n
        nodos
117     !   con m conexiones a partir de una red de n0 nodos.
118     !
119     subroutine BA_net(n0,n,m,idum,edges)
120
121         implicit none
122
123         integer, intent(in)                 :: n, n0, m
124         integer, intent(inout)              :: idum
125         integer,intent(out), allocatable    :: edges(:,:)
126
127         integer,allocatable                 :: nodes(:), wk(:), nodes_aux(:)
128         integer                             :: i, ne, offset, j, ii
129
130         if ((n .gt. n0).and.(n0.ge.m)) then
131
132
133             allocate(nodes(n))
134
135             !Red sin conectar
136             do i=1,n
137                 nodes(i)=i-1
138             end do
139
140             !Primeras m conexiones al nodo n0+1
141             nodes_aux=sample(nodes(1:n0),m,idum)
142
143             deallocate(nodes)
144
145             !Numero total de aristas
146             ne=(n-n0)*m
147
148             allocate(edges(2,ne), wk(2*ne))
149
150             offset=1
151             j=1
152
153             do i=1,m
154                 edges(1,i)=n0
155                 edges(2,i)=nodes_aux(i)
156                 j=j+1
157                 wk(offset)=n0
158                 offset=offset+1
159                 wk(offset)=nodes_aux(i)
160                 offset=offset+1
161
162             end do
163
164             do i=n0+2,n
165
166                 deallocate(nodes_aux)
```

```
167
168                 !Preferential attachment
169                 nodes_aux=sample(wk(1:offset),m,idum)
170
171                 do ii=1,m
172
173                     edges(1,j)=i-1
174                     edges(2,j)=nodes_aux(ii)
175                     j=j+1
176
177                     wk(offset)=i-1
178                     offset=offset+1
179                     wk(offset)=nodes_aux(ii)
180                     offset=offset+1
181
182                 end do
183
184             end do
185
186             deallocate(wk)
187             deallocate(nodes_aux)
188
189         else
190
191             print*, "Datos Erroneos"
192             stop
193
194         end if
195
196
197
198     end subroutine BA_net
199
200 end module Barabasi
```

6.2. Cálculo Medidas topológicas

6.2.1. Listas simples

```
1  ! linkedlist.f90 --
2  !     Include file for defining linked lists where each element holds
3  !     the same kind of data
4  !
5  !     See the example/test program for the way to use this
6  !
7  !     Note:
8  !     You should only use pointer variables of this type, no
9  !     ordinary variables, as sometimes the memory pointed to
10 !     will be deallocated. The subroutines and functions
11 !     are designed to minimize mistakes (for instance: using
12 !     = instead of =>)
13 !
14 !     $Id: linkedlist.f90,v 1.3 2007/01/26 09:56:43 arjenmarkus Exp $
15 !
16 ! Define the linked-list data type
17 !
18 type LINKED_LIST
19     type(LINKED_LIST), pointer :: next
20     type(LIST_DATA)            :: data
21 end type LINKED_LIST
22
23 !
```

```
24 ! Define the subroutines and functions
25 !
26 contains
27
28     ! list_create --
29     !     Create and initialise a list
30     ! Arguments:
31     !     list        Pointer to new linked list
32     !     data        The data for the first element
33     ! Note:
34     !     This version assumes a shallow copy is enough
35     !     (that is, there are no pointers within the data
36     !     to be stored)
37     !     It also assumes the argument list does not already
38     !     refer to a list. Use list_destroy first to
39     !     destroy up an old list.
40     !
41     subroutine list_create( list, data )
42         type(LINKED_LIST), pointer  :: list
43         type(LIST_DATA), intent(in) :: data
44
45         allocate( list )
46         list%next => null()
47         list%data =  data
48     end subroutine list_create
49
50     ! list_destroy --
51     !     Destroy an entire list
52     ! Arguments:
53     !     list        Pointer to the list to be destroyed
54     ! Note:
55     !     This version assumes that there are no
56     !     pointers within the data that need deallocation
57     !
58     subroutine list_destroy( list )
59         type(LINKED_LIST), pointer  :: list
60
61         type(LINKED_LIST), pointer  :: current
62         type(LINKED_LIST), pointer  :: next
63
64         current => list
65         do while ( associated(current%next) )
66             next => current%next
67             deallocate( current )
68             current => next
69         enddo
70         deallocate( current )
71         nullify(list)
72     end subroutine list_destroy
73
74     ! list_count --
75     !     Count the number of items in the list
76     ! Arguments:
77     !     list        Pointer to the list
78     !
79     integer function list_count( list )
80         type(LINKED_LIST), pointer  :: list
81
82         type(LINKED_LIST), pointer  :: current
83         !type(LINKED_LIST), pointer  :: next
84
85         if ( associated(list) ) then
```

```
            list_count = 1
            current => list
            do while ( associated(current%next) )
                current => current%next
                list_count = list_count + 1
            enddo
        else
            list_count = 0
        endif
    end function list_count

    ! list_next
    !     Return the next element (if any)
    ! Arguments:
    !     elem        Element in the linked list
    ! Result:
    !
    function list_next( elem ) result(next)
        type(LINKED_LIST), pointer :: elem
        type(LINKED_LIST), pointer :: next

        next => elem%next

    end function list_next

    ! list_insert
    !     Insert a new element
    ! Arguments:
    !     elem        Element in the linked list after
    !                 which to insert the new element
    !     data        The data for the new element
    !
    subroutine list_insert( elem, data )

        type(LINKED_LIST), pointer  :: elem
        type(LIST_DATA), intent(in) :: data

        type(LINKED_LIST), pointer :: next

        allocate(next)

        next%next => elem%next
        elem%next => next
        next%data =  data

    end subroutine list_insert

    ! list_insert_head
    !     Insert a new element before the first element
    ! Arguments:
    !     list        Start of the list
    !     data        The data for the new element
    !
    subroutine list_insert_head( list, data )

        type(LINKED_LIST), pointer  :: list
        type(LIST_DATA), intent(in) :: data

        type(LINKED_LIST), pointer :: elem

        allocate(elem)
        elem%data =  data
```

```
148
149         elem%next => list
150         list      => elem
151
152     end subroutine list_insert_head
153
154     ! list_delete_element
155     !     Delete an element from the list
156     ! Arguments:
157     !     list       Header of the list
158     !     elem       Element in the linked list to be
159     !                removed
160     !
161     subroutine list_delete_element( list, elem )
162
163         type(LINKED_LIST), pointer  :: list
164         type(LINKED_LIST), pointer  :: elem
165
166         type(LINKED_LIST), pointer  :: current
167         type(LINKED_LIST), pointer  :: prev
168
169         if ( associated(list,elem) ) then
170             list => elem%next
171             deallocate( elem )
172         else
173             current => list
174             prev    => list
175             do while ( associated(current) )
176                 if ( associated(current,elem) ) then
177                     prev%next => current%next
178                     deallocate( current ) ! Is also "elem"
179                     exit
180                 endif
181                 prev    => current
182                 current => current%next
183             enddo
184         endif
185         !    allocate(next)
186         !
187         !    next%next => elem%next
188         !    elem%next => next
189         !    next%data =  data
190
191     end subroutine list_delete_element
192
193     ! list_get_data
194     !     Get the data stored with a list element
195     ! Arguments:
196     !     elem       Element in the linked list
197     !
198     function list_get_data( elem ) result(data)
199
200         type(LINKED_LIST), pointer :: elem
201
202         type(LIST_DATA)            :: data
203
204         data = elem%data
205     end function list_get_data
206
207     ! list_put_data
208     !     Store new data with a list element
209     ! Arguments:
```

```
210     !     elem       Element in the linked list
211     !     data       The data to be stored
212     !
213     subroutine list_put_data( elem, data )
214
215         type(LINKED_LIST), pointer  :: elem
216         type(LIST_DATA), intent(in) :: data
217
218         elem%data = data
219
220     end subroutine list_put_data
```

6.2.2. Redes

6.2.2.1. Declaración de tipo de datos en listas simples

```
1  module DATA_MODULE
2
3
4      type vecino_data
5          integer :: v
6      end type
7
8      type distr
9          real :: key
10         integer :: ncount
11         integer :: n_cum
12         real :: rel_count
13         real :: rel_cum
14
15     end type
16
17 end module
18
19 module vecino_LISTS
20     use DATA_MODULE, ONLY: LIST_DATA => vecino_data
21     include "linkedlist.f90"
22
23 end module vecino_LISTS
24
25 module distr_LISTS
26     use DATA_MODULE, ONLY: LIST_DATA => distr
27     include "linkedlist.f90"
28
29 end module distr_LISTS
```

6.2.2.2. Polimorfismo de listas

```
1  module TWO_LIST_TYPES
2
3      use vecino_LISTS, only: vecino_LINKED_LIST   => LINKED_LIST,   &
4                              vecino_LIST_DATA      => LIST_DATA,     &
5                              list_count_vecino     => list_count,    &
6                              list_insert_head_vecino    => list_insert_head,
     &
7                              list_create_vecino    => list_create,   &
8                              list_insert_vecino    => list_insert,   &
9                              list_next_vecino      => list_next,     &
10                             list_get_vecino_data  => list_get_data, &
11                             list_put_vecino_data  => list_put_data, &
12                             list_delete_vecino    => list_delete_element, &
13                             list_destroy_vecino   => list_destroy
14
```

```
15      use distr_LISTS, only: distr_LINKED_LIST   => LINKED_LIST,   &
16                             distr_LIST_DATA       => LIST_DATA,     &
17                             list_count_distr      => list_count,    &
18                             list_insert_head_distr    => list_insert_head,    &
19                             list_create_distr     => list_create,   &
20                             list_insert_distr     => list_insert,   &
21                             list_next_distr       => list_next,     &
22                             list_get_distr_data   => list_get_data, &
23                             list_put_distr_data   => list_put_data, &
24                             list_delete_distr     => list_delete_element, &
25                             list_destroy_distr    => list_destroy
26      private
27      public :: list_create, list_count, list_destroy, list_get_data, &
28                list_put_data, list_insert, list_next, &
29                list_delete_element, list_insert_head
30      public ::  vecino_LINKED_LIST, distr_LINKED_LIST
31      public ::  vecino_LIST_DATA, distr_LIST_DATA
32
33      interface list_count
34          module procedure list_count_vecino
35          module procedure list_count_distr
36      end interface
37
38      interface list_insert_head
39          module procedure list_insert_head_vecino
40          module procedure list_insert_head_distr
41      end interface
42
43      interface list_create
44          module procedure list_create_vecino
45          module procedure list_create_distr
46      end interface
47
48      interface list_destroy
49          module procedure list_destroy_vecino
50          module procedure list_destroy_distr
51      end interface
52
53      interface list_insert
54          module procedure list_insert_vecino
55          module procedure list_insert_distr
56      end interface
57
58      interface list_next
59          module procedure list_next_vecino
60          module procedure list_next_distr
61      end interface
62
63      interface list_get_data
64          module procedure list_get_vecino_data
65          module procedure list_get_distr_data
66      end interface
67
68      interface list_put_data
69          module procedure list_put_vecino_data
70          module procedure list_put_distr_data
71      end interface
72
73      interface list_delete_element
74          module procedure list_delete_vecino
75          module procedure list_delete_distr
76      end interface
```

```
77
78 end module TWO_LIST_TYPES
```

6.2.2.3. Estructura de capas (lista de listas)

```
1  module lis_of_lists
2
3      use TWO_LIST_TYPES
4      type kappa
5          type (kappa), pointer :: next
6          type (vecino_LINKED_LIST), pointer :: root
7      end type
8
9      contains
10
11     subroutine kappa_create( kapa, list)
12
13         implicit none
14         type (kappa), pointer :: kapa
15         type (vecino_LINKED_LIST), pointer :: list
16
17         type (vecino_LINKED_LIST), pointer :: elem1, elem2
18         type (vecino_LIST_DATA) :: v1
19         if (associated(list)) then
20             v1=list_get_data(list)
21
22             allocate(kapa)
23             call list_create(kapa%root,v1)
24             elem2=>kapa%root
25             elem1=>list_next(list)
26             do while(associated(elem1))
27                 v1=list_get_data(elem1)
28                 call list_insert(elem2,v1)
29                 elem1=>list_next(elem1)
30                 elem2=>list_next(elem2)
31
32
33             end do
34
35
36             kapa%next=> null()
37         end if
38
39
40
41     end subroutine
42
43     function kappa_next(elem) result(nxt)
44         implicit none
45
46         type (kappa), pointer :: elem
47         type (kappa), pointer :: nxt
48
49         nxt=>elem%next
50
51     end function
52
53     subroutine kappa_insert_k( elem, list )
54
55         implicit none
56         type (kappa), pointer :: elem
57         type (vecino_LINKED_LIST), pointer :: list
```

```
58
59          type (vecino_LINKED_LIST), pointer :: elem1, elem2
60          type (vecino_LIST_DATA) :: v1
61          type(kappa), pointer  :: siguiente
62
63
64          if (associated(list)) then
65              v1=list_get_data(list)
66
67              allocate(siguiente)
68
69              call list_create(siguiente%root,v1)
70              elem2=>siguiente%root
71              elem1=>list_next(list)
72              do while(associated(elem1))
73
74                  v1=list_get_data(elem1)
75                  call list_insert(elem2,v1)
76                  elem1=>list_next(elem1)
77                  elem2=>list_next(elem2)
78
79              end do
80
81              siguiente%next => elem%next
82              elem%next => siguiente
83
84          end if
85
86      end subroutine
87
88      function kappa_get_data(elem) result(res)
89
90          implicit none
91
92          type(kappa), pointer :: elem
93          type(vecino_LINKED_LIST), pointer :: res
94
95          res=>elem%root
96
97
98      end function
99
100     subroutine kappa_destroy(kapa)
101
102         implicit none
103
104         type(kappa), pointer :: kapa
105
106         type(kappa), pointer :: actual, siguiente
107
108
109         actual => kapa
110         do while ( associated(actual%next) )
111             siguiente => actual%next
112             call list_destroy(kappa_get_data(actual))
113             deallocate( actual )
114             actual => siguiente
115         end do
116         call list_destroy(kappa_get_data(actual))
117         deallocate( actual )
118
119         nullify(kapa)
```

```
120
121     end subroutine
122
123 end module
```

6.2.2.4. Declaración de tipo de dato elemento de la red

```
1  module vector_capas
2      use lis_of_lists
3      type capa
4          type(kappa), pointer :: kappa_i
5
6      end type
7
8  end module
```

6.2.2.5. Funciones y subrutinas para cálculo de medidas topológicas

```
1  module redes_functions
2
3      use vector_capas
4      use distribuciones
5
6      implicit none
7
8      contains
9
10     !add_vecino_ord
11     !   Anade un vecino a la lista ordenado ascendentemente.  NO Acepta
       repitencias
12     !   Arg:
13     !       list    : Puntero a la lista de vecinos ordenada y sin repeticion
14     !       vecino  : vecino de la capa actual.
15
16     subroutine add_vecino_ord(list, vecino)
17
18         implicit none
19         type(vecino_LINKED_LIST), pointer   :: list
20         type(vecino_LIST_DATA), intent(in)  :: vecino
21
22         type(vecino_LINKED_LIST), pointer   :: elem1
23         type(vecino_LIST_DATA)              :: vecino1
24
25         if(associated(list)) then
26             elem1=>list
27             vecino1=list_get_data(elem1)
28             if(vecino1%v .Ge. vecino%v) then
29                 if(vecino1%v .NE. vecino%v) then
30                     call list_insert(list,vecino1)
31                     call list_put_data(list,vecino)
32                 end if
33             else
34                 do while(associated(list_next(elem1)))
35                     vecino1=list_get_data(list_next(elem1))
36                     if(vecino%v .Gt. vecino1%v) then
37                         elem1=>list_next(elem1)
38                     else
39                         exit
40                     end if
41                 end do
42                 if(vecino1%v .NE. vecino%v) call list_insert(elem1,vecino)
43
44             end if
```

```
45
46
47          else
48              call list_create(list,vecino)
49          end if
50
51
52
53      end subroutine
54
55
56      !print_list_vecino
57      !   Imprime una lista de vecinos y un entero asociado a la posicion del
        vecino de la forma:
58      !   'Posicion         Vecino'
59      !   Arg:
60      !       list    : Puntero a la lista de vecinos
61      subroutine print_list_vecino(list )
62
63          implicit none
64
65          type(vecino_LINKED_LIST), pointer  :: list
66          type(vecino_LINKED_LIST), pointer  :: elem_k
67          integer :: i
68
69          type(vecino_LIST_DATA)             :: data
70
71
72
73          i=1
74          elem_k => list
75          do while( associated(elem_k) )
76              data = list_get_data(elem_k)
77              write(*,*)  i, data%v
78              i=i+1
79              elem_k => list_next(elem_k)
80
81          enddo
82
83      end subroutine
84
85      !resta_v
86      !   Resta de listas: En la lista 1 elimina los elementos de la lista 2
87      !   Arg:
88      !       list1   : Puntero de la lista a modificar
89      !       list2   : Puntero de la lista de elementos a eliminar
90      subroutine resta_v(list1,list2)
91          implicit none
92          type(vecino_LINKED_LIST), pointer   :: list1,list2
93
94          type(vecino_LINKED_LIST), pointer   :: elem1, elem2, elem_aux
95          type(vecino_LIST_DATA)              :: vecino1, vecino2
96
97          elem1=>list1
98          elem2=>list2
99          do while(associated(elem1) .AND. associated(elem2))
100             vecino1=list_get_data(elem1)
101             vecino2=list_get_data(elem2)
102             if(vecino1%v .gt. vecino2%v) then
103                 elem2=>list_next(elem2)
104             else if (vecino1%v .lt. vecino2%v) then
105                 elem1=>list_next(elem1)
```

```
106             else
107                 elem_aux=>list_next(elem1)
108
109                 call list_delete_element(list1,elem1)
110
111                 elem1=>elem_aux
112             end if
113         end do
114
115     end subroutine
116
117     function components(file_r) result(n_c)
118
119         implicit none
120         integer                 :: n_c
121         type(capa), dimension(:), allocatable :: kapas
122         character(len=*), intent(in)          :: file_r
123
124         type(kappa), pointer :: elem_kp, aux_kp
125
126         type(vecino_LINKED_LIST), pointer :: elem_k1, elem_k2, list_aux, elem
127         type(vecino_LIST_DATA)            :: vecino1
128         integer :: v, i, s, t, e, v2,e1, i1, i2, j, ii, jj, e2
129         integer, allocatable, dimension(:) :: nodos
130
131         open(unit=23,file=file_r,status='unknown')
132
133         read(23,*) v,e
134
135
136         allocate(kapas(0:v-1),nodos(0:v-1)) !k_ij: capa j del nodo i
137         !Capa 0
138
139         do i=0, v-1
140
141             !anadir vecinos en la capa inicial
142             vecino1%v=i
143             call list_create(list_aux,vecino1)
144             call kappa_create(kapas(i)%kappa_i,list_aux)
145
146             call list_destroy(list_aux)
147             !print*, 'nodo:',i,'capa:', 0
148         end do
149
150         !Capa 1
151
152         do i=1, e
153
154             read(23,*) s, t
155             if (s .ne. t) then
156                 vecino1%v=t
157                 elem_kp=>kapas(s)%kappa_i
158                 if (associated(kappa_next( elem_kp))) then
159                     elem_k1=>kappa_get_data(kappa_next(elem_kp))
160                     call add_vecino_ord(elem_k1,vecino1)
161                 else
162                     call add_vecino_ord(list_aux,vecino1)
163                     call kappa_insert_k(elem_kp,list_aux)
164                     call list_destroy(list_aux)
165                 end if
166
167                 elem_kp=>kapas(t)%kappa_i
```

```
168                 vecino1%v=s
169                 elem_kp=>kapas(t)%kappa_i
170                 if (associated(kappa_next( elem_kp))) then
171                     elem_k1=>kappa_get_data(kappa_next(elem_kp))
172                     call add_vecino_ord(elem_k1,vecino1)
173                 else
174                     call add_vecino_ord(list_aux,vecino1)
175                     call kappa_insert_k(elem_kp,list_aux)
176                     call list_destroy(list_aux)
177                 end if
178
179             end if
180
181         end do
182         close(23)
183
184
185         i=0
186         elem_kp=>kapas(i)%kappa_i
187         j=1
188         nodos=-2
189         nodos(i)=-1
190         do while( associated(elem_kp))
191             !Anade los vecinos de los nodos asociados en la anterior capa
192             elem_k1=>kappa_get_data(elem_kp)
193             do while (associated(elem_k1))
194                 vecino1=list_get_data(elem_k1)
195                 ii=vecino1%v
196                 aux_kp=>kappa_next(kapas(ii)%kappa_i)
197                 elem=>kappa_get_data(aux_kp)
198                 do while (associated(elem))
199                     vecino1=list_get_data(elem)
200                     call add_vecino_ord(list_aux,vecino1)
201                     nodos(vecino1%v)=-1
202                     elem=>list_next(elem)
203                 end do
204                 elem_k1=>list_next(elem_k1)
205             end do
206
207
208             !Eliminar los elementos que pertenecen a anteriores capas
209             aux_kp=>kapas(i)%kappa_i
210             do jj=0, j-1
211                 elem_k1=>kappa_get_data(aux_kp)
212                 call resta_v(list_aux,elem_k1)
213                 aux_kp=>kappa_next(aux_kp)
214             end do
215             if (associated(list_aux)) then
216                 call kappa_insert_k(elem_kp,list_aux)
217                 call list_destroy(list_aux)
218             end if
219
220             elem_kp=>kappa_next(elem_kp)
221             j=j+1
222         end do
223
224         i1=0
225         i2=0
226         do i=0,v-1
227             if(nodos(i).ne.-1) then
228                 vecino1%v=i
229                 call add_vecino_ord(list_aux,vecino1)
```

```
230                 nodos(i)=i2
231                 i2=i2+1
232             else
233                 nodos(i)=i1
234                 i1=i1+1
235             end if
236         end do
237         v2=list_count(list_aux)
238         if (v2 .eq. 0) then
239             n_c=1
240 
241         else
242             n_c=2
243             e1=0
244             e2=0
245             elem_k1=>list_aux
246             vecino1=list_get_data(elem_k1)
247             s=vecino1%v
248             do i=0,v-1
249                 elem_k2=>kappa_get_data(kappa_next(kapas(i)%kappa_i))
250 
251                 if(s .eq. i) then
252                     do while(associated(elem_k2))
253                         vecino1=list_get_data(elem_k2)
254                         if(nodos(s) .gt. nodos(vecino1%v)) then
255                             e2=e2+1
256                         end if
257                         elem_k2=>list_next(elem_k2)
258                     end do
259                     elem_k1=>list_next(elem_k1)
260                     if(associated(elem_k1)) then
261                         vecino1=list_get_data(elem_k1)
262                         s=vecino1%v
263                     end if
264                 else
265                     do while(associated(elem_k2))
266                         vecino1=list_get_data(elem_k2)
267                         if(nodos(i) .gt. nodos(vecino1%v)) then
268                             e1=e1+1
269                         end if
270                         elem_k2=>list_next(elem_k2)
271                     end do
272                 end if
273             end do
274 
275 
276             open(32,file='componente1.dat',status='unknown')
277             open(23,file='componente2.dat',status='unknown')
278 
279             write(23,*) v2, e2
280             write(32,*) v-v2, e1
281             elem_k1=>list_aux
282             vecino1=list_get_data(elem_k1)
283             s=vecino1%v
284 
285             do i=0,v-1
286                 elem_k2=>kappa_get_data(kappa_next(kapas(i)%kappa_i))
287 
288                 if(s .eq. i) then
289                     do while(associated(elem_k2))
290                         vecino1=list_get_data(elem_k2)
291                         if(nodos(s) .gt. nodos(vecino1%v)) then
```

```
292                         write(23,*) nodos(s), nodos(vecino1%v)
293                     end if
294                     elem_k2=>list_next(elem_k2)
295                 end do
296                 elem_k1=>list_next(elem_k1)
297                 if(associated(elem_k1)) then
298                     vecino1=list_get_data(elem_k1)
299                     s=vecino1%v
300                 end if
301             else
302                 do while(associated(elem_k2))
303                     vecino1=list_get_data(elem_k2)
304                     if(nodos(i) .gt. nodos(vecino1%v)) then
305                         write(32,*) nodos(i), nodos(vecino1%v)
306                     end if
307                     elem_k2=>list_next(elem_k2)
308                 end do
309             end if
310         end do
311
312         close(32)
313         close(23)
314         call list_destroy(list_aux)
315     end if
316
317
318     call capa_destroy(kapas)
319
320     deallocate(kapas)
321
322
323 end function
324
325
326 !construct_kapa
327 !   Subrutina para la construccion de capas iterativamente en base a la
    red
328 !   Arg:
329 !       kapas   : Vector de kappas donde la i-esima posicion corresponde
    a la kapa 0 del nodo i de la red
330 !       net     : Vector de nodos donde la i-esima posicion corresponde
    al nodo i de la red
331 subroutine construct_kapa(kapas, file_r)
332
333     implicit none
334
335     type(capa), dimension(:), allocatable :: kapas
336     character(len=*), intent(in)          :: file_r
337
338     type(kappa), pointer :: elem_kp, aux_kp
339
340     type(vecino_LINKED_LIST), pointer :: elem_k1, list_aux, elem
341     type(vecino_LIST_DATA)            :: vecino1
342     integer :: v, i, j, ii, jj, s, t, e
343
344     open(unit=23,file=file_r,status='unknown')
345
346     read(23,*) v,e
347     if(associated(list_aux)) then
348         nullify(list_aux)
349     end if
350
```

```
351         allocate(kapas(0:v-1)) !k_ij: capa j del nodo i
352         !Capa 0
353
354
355         do i=0, v-1
356             if(associated(kapas(i)%kappa_i)) then
357                 nullify(kapas(i)%kappa_i)
358             end if
359             !anadir vecinos en la capa inicial
360             vecino1%v=i
361             call list_create(list_aux,vecino1)
362             call kappa_create(kapas(i)%kappa_i,list_aux)
363
364             call list_destroy(list_aux)
365
366         end do
367
368         !Capa 1
369
370         do i=1, e
371
372             read(23,*) s, t
373             if (s .ne. t) then
374                 vecino1%v=t
375                 elem_kp=>kapas(s)%kappa_i
376                 if (associated(kappa_next( elem_kp))) then
377                     elem_k1=>kappa_get_data(kappa_next(elem_kp))
378                     call add_vecino_ord(elem_k1,vecino1)
379                 else
380                     call add_vecino_ord(list_aux,vecino1)
381                     call kappa_insert_k(elem_kp,list_aux)
382                     call list_destroy(list_aux)
383                 end if
384
385                 vecino1%v=s
386                 elem_kp=>kapas(t)%kappa_i
387                 if (associated(kappa_next( elem_kp))) then
388                     elem_k1=>kappa_get_data(kappa_next(elem_kp))
389                     call add_vecino_ord(elem_k1,vecino1)
390                 else
391                     call add_vecino_ord(list_aux,vecino1)
392                     call kappa_insert_k(elem_kp,list_aux)
393                     call list_destroy(list_aux)
394                 end if
395
396             end if
397
398         end do
399
400         !Capas 2-n
401
402         do i=0, v-1
403             elem_kp=>kappa_next(kapas(i)%kappa_i)
404             j=2
405             do while( associated(elem_kp))
406
407                 !Anade los vecinos de los nodos asociados en la anterior capa
408                 elem_k1=>kappa_get_data(elem_kp)
409                 do while (associated(elem_k1))
410                     vecino1=list_get_data(elem_k1)
411                     ii=vecino1%v
412                     aux_kp=>kappa_next(kapas(ii)%kappa_i)
```

```
413                     elem=>kappa_get_data(aux_kp)
414                     do while (associated(elem))
415                         vecino1=list_get_data(elem)
416                         call add_vecino_ord(list_aux,vecino1)
417                         elem=>list_next(elem)
418                     end do
419                     elem_k1=>list_next(elem_k1)
420                 end do
421                 !Eliminar los elementos que pertenecen a anteriores capas
422                 aux_kp=>kapas(i)%kappa_i
423                 do jj=0, j-1
424                     elem_k1=>kappa_get_data(aux_kp)
425                     call resta_v(list_aux,elem_k1)
426                     aux_kp=>kappa_next(aux_kp)
427                 end do
428                 if (associated(list_aux)) then
429                     call kappa_insert_k(elem_kp,list_aux)
430                     call list_destroy(list_aux)
431                 end if
432                 elem_kp=>kappa_next(elem_kp)
433                 j=j+1
434             end do
435
436
437         end do
438
439         print*, 'Finalizo la creacion de capas correctamente'
440     end subroutine
441
442     !caminos
443     !   Escribe la longitud de camino del nodo i al nodo j en base a la capa
444     !   formada, de la forma:
445     !   'Nodo i     Nodo j      Path Length'
446     !   Arg:
447     !       file_t  : Nombre del archivo de texto para escribir los caminos
448     !       kapas   : Vector de capas
449     subroutine caminos(kapas)
450
451         implicit none
452         type(capa), dimension(:), intent(in) :: kapas
453         character(len=*),parameter :: file_t='caminos2.dat'
454         type(distr_medidas)                  :: pathh
455         type(kappa), pointer :: elem_kp
456         type(vecino_LIST_DATA) :: vecino_1
457         type(vecino_LINKED_LIST), pointer :: elem
458         integer :: l1, i, j
459
460         if  (associated(pathh%inf_t)) then
461             nullify(pathh%inf_t)
462         end if
463
464         l1=size(kapas,1)
465
466         do i=1,l1
467             j=0
468             elem_kp=>kapas(i)%kappa_i
469             do while (associated(elem_kp))
470
471                 elem=>kappa_get_data(elem_kp)
472
473                 do while(associated(elem))
474                     vecino_1=list_get_data(elem)
```

```
475                     if (i-1 .gt. vecino_1%v) then
476                         call add_distr_ord(pathh,real(j))
477                     end if
478                     elem=>list_next(elem)
479                 end do
480                 elem_kp=>kappa_next(elem_kp)
481                 j=j+1
482             end do
483         end do
484         call dist_medidas(pathh)
485
486         call print_medidas(pathh,file_t)
487
488         call list_destroy(pathh%inf_t)
489     end subroutine
490
491     !degree
492     !   Escribe el grado de salida y entrada de cada nodo en la red, de la
        forma:
493     !   'Nodo       Kin     Kout'
494     !   Arg:
495     !       fle     : Nombre del archivo de texto
496     !       net     : Vector de nodos
497     subroutine degree( kapas)
498
499         implicit none
500
501         type(capa), dimension(:), intent(in) :: kapas
502
503
504         character(len=*), parameter         :: file3='degree.dat'
505         type(distr_medidas)                 :: k_k
506         type(vecino_LINKED_LIST), pointer   :: elem
507         integer                             :: l1, i, kk
508
509         l1=size(kapas,1)
510
511
512         if  (associated(k_k%inf_t)) then
513             nullify(k_k%inf_t)
514         end if
515         do i=1, l1
516             elem=>kappa_get_data(kappa_next(kapas(i)%kappa_i))
517
518             kk=list_count(elem)
519             call add_distr_ord(k_k,real(kk))
520         end do
521         call dist_medidas(k_k)
522         call print_medidas(k_k,file3)
523         call list_destroy(k_k%inf_t)
524
525     end subroutine
526
527     !clusteringc
528     !   Escribe el clustering coefficient de cada nodo en la red, de la forma
        :
529     !   'Nodo       CC'
530     !   Arg:
531     !       fle     : Nombre del archivo de texto
532     !       net     : Vector de nodos
533     subroutine clusteringc(kapas)
534         implicit none
```

```
535
536         type(capa), dimension(:), intent(in):: kapas
537         character(len=*), parameter          :: file1='clustering.dat'
538         type(distr_medidas)                  :: cluster
539
540         type(vecino_LINKED_LIST), pointer    :: elem1, elem2, elem3
541         type(vecino_LIST_DATA)               :: arista1, arista2, arista3
542         integer                              :: l1, i, k
543         real                                 :: cc
544
545         l1=size(kapas,1)
546         if  (associated(cluster%inf_t)) then
547             nullify(cluster%inf_t)
548         end if
549
550         do i=1, l1
551             elem1=>kappa_get_data(kappa_next(kapas(i)%kappa_i))
552             k=list_count(elem1)
553             cc=0.
554             if(k .GT. 1) then
555                 do while(associated(elem1))
556
557                     arista1=list_get_data(elem1)
558                     elem2=>list_next(elem1)
559                     do while(associated(elem2))
560                         arista2=list_get_data(elem2)
561                         elem3=>kappa_get_data(kappa_next(kapas(arista2%v+1)%
    kappa_i))
562                         do while(associated(elem3))
563                             arista3=list_get_data(elem3)
564                             if(arista3%v .EQ. arista1%v) then
565                                 cc=cc+1.
566                                 exit
567                             else if(arista3%v .GT. arista1%v) then
568                                 exit
569                             end if
570                             elem3=>list_next(elem3)
571                         end do
572
573                         elem2=>list_next(elem2)
574                     end do
575                     elem1=>list_next(elem1)
576                 end do
577                 cc=2.*cc/(k*k-k)
578                 call  add_distr_ord(cluster,cc)
579             end if
580
581
582
583         end do
584
585
586         call dist_medidas(cluster)
587
588         call print_medidas(cluster,file1)
589
590         call list_destroy(cluster%inf_t)
591
592     end subroutine
593
594     !capa_destroy
595     !   Eliminacion de la las capas de la memoria
```

```
596     !   Arg:
597     !       kp      : vector de capas a destruir
598     subroutine capa_destroy( kp)
599
600         implicit none
601         type(capa), dimension(:), intent(inout):: kp
602         integer :: l1, i
603         l1=size(kp,1)
604
605         do i=1,l1
606             call kappa_destroy(kp(i)%kappa_i)
607         end do
608
609
610
611     end subroutine
612
613
614 end module
```

6.2.2.6. Funciones y subrutinas de indicadores estadísticos

```
1  module distribuciones
2
3      use TWO_LIST_TYPES
4
5      type distr_medidas
6          type(distr_LINKED_LIST), pointer    :: inf_t
7          real                                :: max
8          real                                :: min
9          integer                             :: total
10         real                                :: med
11         real                                :: mediana
12         real                                :: moda
13         real                                :: desv_abs
14         real                                :: var
15         real                                :: skew
16         real                                :: curt
17
18     end type
19
20     contains
21
22     subroutine add_distr_ord(x_i, kkey)
23
24         implicit none
25
26         type(distr_medidas)                 :: x_i
27         real, intent(in)                    :: kkey
28
29         type(distr_LINKED_LIST), pointer    :: list
30         type(distr_LINKED_LIST), pointer    :: elem1
31         type(distr_LIST_DATA)               :: key, key1
32         integer :: i
33         key%key=kkey
34         key%ncount=1
35         list=>x_i%inf_t
36         if(associated(list)) then
37             elem1=>list
38             key1=list_get_data(elem1)
39             if(key1%key .Ge. key%key) then
40                 if(key1%key .NE. key%key) then
```

```
                    call list_insert(list,key1)
                    call list_put_data(list,key)
                    !print*, 'Se inserta al principio', key%key, key%ncount
                else
                    key%ncount=key1%ncount+1
                    call list_put_data(list,key)
                    !print*, 'Se aumenta el contador al principio'
                end if
            else
                i=1
                do while(associated(list_next(elem1)))
                    i=i+1
                    key1=list_get_data(list_next(elem1))
                    if(key%key .Gt. key1%key) then
                        elem1=>list_next(elem1)
                    else
                        exit
                    end if
                end do

                if(key1%key .NE. key%key) then
                    call list_insert(elem1,key)
                    !print*, 'Se inserta en la posicion', i+1
                else
                    key%ncount=key1%ncount+1
                    !print*, 'Se aumenta en la posicion', i+1
                    call list_put_data(list_next(elem1),key)
                end if

            end if


        else
            !print*, 'primer item'
            call list_create(x_i%inf_t,key)

        end if


    end subroutine

    subroutine dist_medidas(x_i)
        implicit none

        type(distr_medidas)                 :: x_i

        type(distr_LINKED_LIST), pointer    :: list
        type(distr_LINKED_LIST), pointer    :: elem
        type(distr_LIST_DATA)               :: data_i
        integer :: n_max
        real :: sr1
        logical :: st


        list=> x_i%inf_t
        elem=>list
        x_i%total=0
        n_max=-1
        if (associated(elem)) then
            data_i=list_get_data(elem)
            x_i%min=data_i%key
        end if
```

```
103         do while(associated(elem))
104             data_i=list_get_data(elem)
105             x_i%total=x_i%total+data_i%ncount
106             if(data_i%ncount .gt. n_max) then
107                 n_max=data_i%ncount
108                 x_i%moda=data_i%key
109             end if
110             data_i%n_cum=x_i%total
111             call list_put_data(elem,data_i)
112             elem=>list_next(elem)
113         end do
114         x_i%max=data_i%key
115         elem=>list
116         sr1=0.
117         x_i%med=0.
118         st=.True.
119 
120         do while(associated(elem))
121             data_i=list_get_data(elem)
122             data_i%rel_count=(1.*data_i%ncount)/(1.*x_i%total)
123             x_i%med=x_i%med+data_i%rel_count*data_i%key
124             sr1=sr1+data_i%rel_count
125             if(sr1 .ge. 0.5 .and. st) then
126                 x_i%mediana=data_i%key
127                 st=.False.
128             end if
129             data_i%rel_cum=sr1
130             call list_put_data(elem,data_i)
131             elem=>list_next(elem)
132         end do
133 
134         elem=>list
135         x_i%desv_abs=0.
136         x_i%var=0.
137         x_i%skew=0.
138         x_i%curt=0.
139         do while(associated(elem))
140             data_i=list_get_data(elem)
141             x_i%desv_abs=x_i%desv_abs+abs(x_i%mediana-data_i%key)*data_i%
    rel_count
142             x_i%var=x_i%var+((x_i%med-data_i%key)**2)*data_i%rel_count
143             x_i%skew=x_i%skew+((x_i%med-data_i%key)**3)*data_i%rel_count
144             x_i%curt=x_i%curt+((x_i%med-data_i%key)**4)*data_i%rel_count
145             elem=>list_next(elem)
146         end do
147         x_i%skew=x_i%skew/(x_i%var)**1.5
148         x_i%curt=x_i%curt/(x_i%var)**2-3.
149     end subroutine dist_medidas
150 
151     subroutine print_medidas(x_i,fle)
152 
153         implicit none
154         type(distr_medidas), intent(in) :: x_i
155         character(len=*), intent(in)    :: fle
156 
157 
158         type(distr_LINKED_LIST), pointer    :: list
159         type(distr_LINKED_LIST), pointer    :: elem
160         type(distr_LIST_DATA)               :: data_i
161 
162 
163         open(unit=23, file=fle, status='unknown')
```

```
164
165         write(23,*) '## Medidas Estadisticas ##'
166         write(23,*) '# |Min=', x_i%min, '|Max=',x_i%max, '|#Elementos=',x_i%
    total
167         write(23,*) '# |Media=', x_i%med, '|Mediana=', x_i%mediana, '|Moda=',
     x_i%moda
168         write(23,*) '# |Desviacion estandar=', sqrt(x_i%var), '|Desviacion
    absoluta=', x_i%desv_abs
169         write(23,*) '# |Skewness=', x_i%skew, '|Kurtosis=', x_i%curt
170         write(23,*) '
    #-------------------------------------------------------------------------#
    '
171         write(23,*) '#    x_i                 n_i          N_i         f_i
                F_i'
172         write(23,*) '
    #-------------------------------------------------------------------------#
    '
173         list=>x_i%inf_t
174
175         elem=>list
176
177         do while(associated(elem))
178             data_i=list_get_data(elem)
179             write(23,*) data_i%key, data_i%ncount, data_i%n_cum, data_i%
    rel_count, data_i%rel_cum
180             elem=>list_next(elem)
181         end do
182
183         close(unit=23)
184
185
186     end subroutine print_medidas
187
188 end module
```

6.3. Modulo de Python

6.3.1. Importe y limpieza de redes

```
1  import json
2  import pandas as pd
3  from collections import defaultdict
4
5  # importamos el archivo file y obtenemos dos DataFrames del conjunto de nodos
6  # y de aristas
7
8  def import_net(file):
9
10     file1=open(file,"r")
11
12     # Obtenemos los datos del archivo ".cx" de la red en formato json
13     o1=json.load(file1)
14
15     file1.close()
16
17     # Obtenemos los lugares del array o1 donde se encuentran los datos
    relacionados
18     # a los nodos y aristas de la red
19     i1=0
20     i2=0
21     for i in range(len(o1)):
22         if "nodes" in o1[i].keys():
```

```
23              i1=i
24          if "edges" in o1[i].keys():
25              i2=i
26
27      # Estructuramos los datos de los nodos de la red en un DataFrame
28      df1=pd.DataFrame(o1[i1]["nodes"])
29
30      # Guardamos solo las columnas relacionadas al numero del nodo y el nombre
31      # de la molecula asociada [@id: int, n: str ]
32
33      if "r" in df1.columns:
34          df1.drop("r",axis=1,inplace=True)
35
36
37      # Estructuramos los datos de las aristas en dos columnas [s: int,t: int]
38      df2=pd.DataFrame(o1[i2]["edges"]).drop(["@id"],axis=1)
39
40      return df2
41
42  # filtramos las aristas de la red para evitar loops, aristas repetidas y
        retorna
43  # un DataFrame de aristas ordenadas.
44
45  def filtering_edges(df2: pd.DataFrame):
46
47      # Duplicamos las aristas existentes e invertimos el orden de las aristas
48      # en el segundo grupo
49
50      dfi=pd.DataFrame(columns=["s","t"])
51      dfi["t"]=df2.s
52      dfi["s"]=df2.t
53      dfi=pd.concat([df2,dfi],ignore_index=True)
54
55      # Mantenemos solo las aristas tal que t>s para eliminar loops
56      dfi=dfi[dfi.s<dfi.t]
57      # Ordenamos
58      dfi=dfi.sort_values(by="s")
59      # Eliminamos las aristas repetidas
60      dfi=dfi.drop_duplicates()
61
62      return dfi
63
64  # Retorna un diccionario (: int, list(:int,:str)) con keys el conjunto de
65  # nodos vi y les asigna un nuevo nodo de 0 a |V|-1 con su nombre de molecula
66  def di_nodesid(df1 : pd.DataFrame, vi: list):
67
68      di1=defaultdict(int)
69
70      #inicializacion de V'
71      for i in range(len(vi)):
72          di1[vi[i]]=[i,""]
73
74      # mapeo entre el conjunto de nodos declarado en df1 a V'
75      for i in range(len(df1)):
76          di1[df1.loc[i,"@id"]][1]=df1.loc[i,"n"]
77
78      return di1
79
80
81  # mapea las aristas del espacio E en V a E' en V' y retorna un DataFrame con
82  # las aristas de E'
83
```

```
84 def change_id(di_nodes: defaultdict ,df2: pd.DataFrame):
85
86     # Funcion local que retorna el nodo de V'
87     f=lambda y: di_nodes[y][1]
88
89     df2p=pd.DataFrame(columns=["s","t"])
90
91     # Cambio de E->E'
92     df2p.s=df2.s.apply(f)
93     df2p.s=df2.t.apply(f)
94
95     # Ordenar las aristas
96     df2p=df2p.sort_values(by=["s","t"])
97
98     return df2p
99
100 # Exporta los datos de nodos y aristas de la red en dos ficheros "nodes.dat"
      y
101 # "red_r.dat"
102
103 def export_data(df2: pd.DataFrame, di_nodes: defaultdict):
104
105     file1=open("nodes.dat","w")
106
107     for item in di_nodes:
108         file1.write("{}\t{}".format(item[0],item[1]))
109
110     file1.close()
111
112     file1=open("red_r.dat","w")
113
114     file1.write("{}\t{}".format(len(di_nodes),len(df2)))
115
116     for i in range(len(df2)):
117         file1.write("".format(df2.loc[i,"s"],df2.loc[i,"t"]))
118     file1.close()
```

6.3.2. Busqueda en base de datos HGNC y asociacion de moleculas con su tipo

```
1 import json
2 import pandas as pd
3 # A partir de la base de datos HGNC repartida en df1 (nombres actuales)
4 # df2 (nombres antiguos)
5 # df3 (nombres reconocidos analogos)
6
7 def types(df1: pd.DataFrame,df2: pd.DataFrame,
8             df3: pd.DataFrame, df_n: pd.DataFrame):
9
10     # Fijamos los indices con las columnas del nombre de cada df
11     df1.set_index("symbol",inplace=True)
12     df2.set_index("prev_symbol",inplace=True)
13     df3.set_index("alias_symbol",inplace=True)
14
15     # Agregamos la columna types al dataframe de nodos con valor "None"
16
17     df_n["type"]=["None" for i in range(len(df1))]
18
19     # ciclo para asociar a las moleculas con su tipo registrado en las bases
20     # df1m df2 y df3 en "locus_group"
21
22     for dfi in [df1,df2,df3]:
```

```
23
24          # agregamos a var1 el nombre de las moleculas que aun no se asocian
    con
25          # un tipo
26
27          var1=df_n.loc[df_n["type"]=="None"]["n"]
28
29          # Si el nombre utilizado de la molecula se encuentra en las bases de
    datos
30          # se agrega su tipo, caso contrario se agrega "None"
31          for i in var1.index:
32              try:
33                  df_n.loc[i,"type"]=dfi.loc[var1.loc[i],"locus_group"]
34              except:
35                  df_n.loc[i,"type"]="None"
36
37
38      # Ciclo para reemplazar si es que existen moleculas con las siglas en
39      # mayuscula
40      for dfi in [df1,df2,df3]:
41
42          var1=df_n.loc[df_n["type"]=="None"]["n"]
43          for i in var1.index:
44              try:
45                  df_n.loc[i,"type"]=dfi.loc[var1.loc[i].replace("ORF","orf"),"
    locus_group"]
46              except:
47                  df_n.loc[i,"type"]="None"
48
49      # a las moleculas que no se encontraron en la base se les coloca el tipo
    "other"
50
51      df_n["type"]=df_n["type"].apply(lambda x: "other" if x =="None" else x)
52
53      # Eliminamos la columna del nombre de las moleculas
54      df_n.drop("n",axis=1,inplace=True)
55
56      return df_n
```