Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación (GITT)

Algoritmos para el análisis de redes y grafos

Autor: Pedro Hidalgo Tapia
Tutor: Sergio Antonio Cruces Álvarez

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación (GITT)

Algoritmos para el análisis de redes y grafos

Autor:
Pedro Hidalgo Tapia

Tutor:
Sergio Antonio Cruces Álvarez
Catedrático de Universidad

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Algoritmos para el análisis de redes y grafos

Autor: Pedro Hidalgo Tapia

Tutor: Sergio Antonio Cruces Álvarez

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas aquellas personas que me han acompañado durante estos años. Aunque el camino no ha sido fácil, su cariño y apoyo en todo momento me han ayudado a llegar hasta el final.

En primer lugar, quiero agradecer a toda mi familia, quienes me han apoyado en todo momento y me han impulsado a seguir trabajando día a día, recordándome que era capaz de lograr lo que me propusiese cuando ni yo mismo a veces podía pensarlo. Verlos celebrar cada uno de mis logros con casi más emoción que yo y sentir su orgullo ha sido mi principal motivación para seguir esforzándome. También agradezco a mis amigos, que han sido una fuente constante de motivación y alegría. Su compañía me ha ayudado muchísimo a disfrutar y olvidar los malos momentos, tanto en la carrera como en la vida cotidiana.

Por último, agradezco a todos los profesores que he tenido a lo largo de la carrera, quienes me han brindado sus conocimientos y enseñanzas. En especial, quiero destacar a Sergio, mi tutor, por acompañarme durante este trabajo final y estar dispuesto a ayudar en cualquier cosa que necesitase.

Pedro Hidalgo Tapia

Sevilla, 2024

Resumen

El análisis de redes y grafos es una disciplina fundamental en diversos campos como la informática, las ciencias sociales y el transporte. Esta disciplina ofrece la capacidad de modelar y analizar todo tipo de estructuras complejas, desde la red aérea de vuelos hasta las interacciones entre las proteínas en una célula, proporcionando información crucial para la optimización y gestión eficiente de estos sistemas. La teoría de redes permite representar de forma visual conjuntos de datos abstractos en formas de nodos o vértices y las conexiones o relaciones que estos pueden tener con otros nodos a través de aristas, generando un grafo sobre el cual aplicar algoritmos de centralidad, detección de comunidades u otras técnicas matemáticas que permiten el análisis de estos conjuntos de datos.

El objetivo de este trabajo es presentar los conceptos necesarios para llevar a cabo un análisis de la red de transporte aéreo mediante vuelos por Europa y la red de metro de la ciudad de Sevilla. Una vez presentados los conceptos, se implementan algoritmos de centralidad como Betweenness Centrality y PageRank para el estudio de la centralidad de los nodos, y algoritmos de detección de comunidades como el algoritmo de Louvain y el de Spectral Clustering. Todo ello, unido a un estudio del comportamiento de la estructura de la red ante fallos o ataques intencionados a los nodos críticos, permite comprender la estructura y comportamiento de la red estudiada, así como su robustez.

Para la implementación de estas técnicas se hace uso de Google Colab, una plataforma que permite la ejecución de código Python en la nube. Principalmente, con la librería Networkx se han podido implementar las técnicas de análisis descritas anteriormente, además de la implementación propia de algunas funciones como PageRank o las empleadas para el análisis de la robustez de la red. Además, se han empleado otras librerías adicionales como Folium, que permite superponer grafos a mapas reales, Pandas, para la extracción de los datos necesarios para la construcción de la red una vez convertidos a un formato adecuado, o Ipython.display, para la generación y visualización de animaciones que muestran la evolución de la red frente a fallos o ataques, entre otras. Estas herramientas combinadas permitieron una implementación robusta y flexible de los algoritmos, facilitando la exploración detallada y la representación gráfica de los datos y resultados.

En ambos casos, se evaluaron características cruciales como la centralidad de los nodos, la detección de comunidades y la robustez de las redes frente a fallos y ataques. Los resultados obtenidos demuestran la vulnerabilidad de las redes ante ataques dirigidos a nodos con alta conectividad y su mayor resistencia frente a fallos aleatorios. En particular, se destacó la importancia de los nodos con alta centralidad en mantener la conectividad global de la red.

Abstract

Network and graph analysis is a fundamental discipline in various fields such as computer science, social sciences, and transportation. It offers the ability to model and analyze all types of complex structures, from airline flight networks to protein interactions within a cell, providing crucial information for the optimization and efficient management of these systems. This theory aims to visually represent abstract data sets in the form of nodes or vertices and the connections or relationships these may have with other nodes through edges, generating a graph on which to apply centrality algorithms, community detection methods, and other mathematical techniques for data set analysis.

The objective of this work is to present the necessary concepts to carry out an analysis of the air transport network through flights in Europe and the metro network of the city of Seville. Once the concepts are introduced, centrality algorithms such as Betweenness Centrality and PageRank are implemented to study node centrality, along with community detection algorithms like the Louvain algorithm and Spectral Clustering. Combined with a study of the network's structural behavior under failures or intentional attacks on critical nodes, this approach enables an understanding of the structure and behavior of the studied network, as well as its robustness.

Google Colab, a platform allowing the execution of Python code in the cloud, is utilized for implementing these techniques. Primarily using the Networkx library, the described analysis techniques were implemented, along with custom functions such as PageRank and those used for network robustness analysis. Additional libraries like Folium, for overlaying graphs on real maps, Pandas, for extracting the necessary data for network construction once converted to an appropriate format, and Ipython.display, for generating and visualizing animations showing the network's evolution under failures or attacks, were also employed. These combined tools enabled a robust and flexible implementation of the algorithms, facilitating detailed exploration and graphical representation of the data and results.

In both cases, crucial characteristics such as node centrality, community detection, and network robustness against failures and attacks were evaluated. The results demonstrate the vulnerability of networks to attacks on highly connected nodes and their greater resistance to random failures. Particularly, the importance of highly central nodes in maintaining the network's global connectivity was highlighted.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

G	Grafo
v	Conjunto de vértices
ε	Conjunto de enlaces
(v_i, v_j)	Enlace desde el vértice *i* hasta el vértice *j*
v_i	Vértice *i*
N	Órden del grafo (número de nodos del grafo)
L	Tamaño del grafo (número de enlaces del grafo)
G_v	Grado del nodo *v*
$< k >$	Conectividad media de la red
k_v^{in}	Grado de entrada del nodo v
k_v^{out}	Grado de salida del nodo v
$\binom{m}{n}$	Número combinatorio
A^T	Transpuesta
A^n	Potencia *n* de una matriz
p_{uv}	Probabilidad de conexión entre el nodo u y v
$B(v)$	Puntuación de centralidad del nodo v (betweenness centrality)
$\Pr(v)$	Puntuación de PageRank del nodo v
δ	Función delta de Kronecker
C_i	Comunidad i
d	Factor de amortiguación (dumping factor)

1 INTRODUCCIÓN

En la era de la información y la tecnología, el análisis de redes y grafos ha emergido como una herramienta fundamental para comprender y abordar una amplia gama de problemas complejos en campos tan diversos como la informática, la biología, la sociología y más allá. Los grafos, estructuras matemáticas compuestas por nodos y aristas, ofrecen un marco formal para representar relaciones entre entidades de manera abstracta. Aunque esta definición inicial puede parecer algo teórica, la aplicación práctica de los grafos es sorprendentemente amplia y variada.

Es crucial diferenciar entre los conceptos de grafos y redes, ya que, aunque a menudo se utilizan indistintamente, tienen diferencias fundamentales en su enfoque y aplicación. Mientras que un grafo se centra en la representación abstracta de entidades y sus relaciones, una red suele referirse más a sistemas reales. Es decir, los grafos pueden entenderse como una representación matemática de las redes, aunque rara vez se emplea esta distinción, ya que estas terminologías sulen ser sinónimas en este contexto.

La importancia de los grafos en la informática y más allá es innegable. Desde la planificación de rutas óptimas en logística hasta la modelización de redes sociales y la optimización de circuitos electrónicos, los grafos se emplean en una amplia variedad de disciplinas para resolver problemas complejos de manera eficiente y efectiva. Su capacidad para modelar relaciones complejas de una manera intuitiva y estructurada es lo que hace que los grafos sean tan poderosos y esenciales en la resolución de una diversidad de problemas.

En este trabajo, exploraremos los conceptos fundamentales de la teoría de grafos, los cuales facilitarán la comprensión de los algoritmos y técnicas relacionados que se estudian y pueden aplicarse para resolver problemas prácticos en diversos dominios.

1.1 Surgimiento de la teoría de grafos

El surgimiento de la teoría de grafos está estrechamente ligado al conocido problema de los puentes de Königsberg, cuya resolución por el matématico suizo Leonhard Euler en 1736 marca el inicio de esta rama de las matemáticas.

Para comprender este problema, debemos retroceder hasta el año 1735, en la ciudad de Königsberg, capital en ese momento de Prusia Oriental, que actualmente pertenece a Rusia y recibe el nombre de Kaliningrado. Esta ciudad estaba atravesada por el río Pregolia, que se dividía en un punto de la ciudad y rodeaba una pequeña área que recibe el nombre de isla Kneiphof, dividiendo el terreno en cuatro secciones distintas. Debido a la gran afluencia mercantil durante esa época y la importancia del comercio, la ciudad construyó siete puentes sobre el río Pregolia para conectar las distintas partes de la ciudad, cinco de los cuales se conectaban a la isla Kneiphof y los otros dos entre las dos ramas del río, como podemos ver en la siguiente figura.

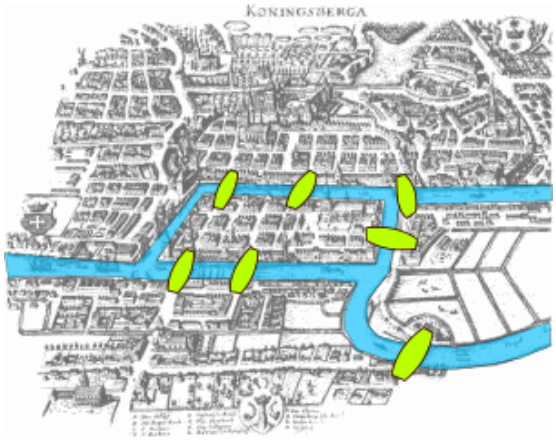


Figura 1-1. Representación de los siete puentes en la ciudad de Königsberg.

Fuente: https://es.wikipedia.org/wiki/Problema_de_los_puentes_de_Königsberg

Esta peculiar disposición de los puentes para interconectar la ciudad dio origen a la siguiente pregunta: ¿Es posible caminar por la ciudad comenzando desde cualquier región, pasar por todos los puentes una sola vez y regresar a la misma región de partida?

El problema podría resolverse, en principio, mediante la prueba de cada una de las posibles rutas para realizar este recorrido, pero fue Euler quien proporcionó una solución matématica y generalizada del problema en 1736 en su publicación "Solutio problematis ad geometriam situs pertinentis" [1]. Para esta solución [2], Euler recurrío a una abstracción del mapa representando las cuatro áreas separadas por el río con las letras A, B, C y D. Luego, conectó con líneas las áreas que tenían un puente entre ellas, designándolas con letras minúsculas desde la 'a' hasta la 'g'. De esta manera, Euler estaba construyendo un grafo para resolver el problema, donde los vértices eran las diferentes áreas de la ciudad y las aristas eran los puentes que las conectaban.

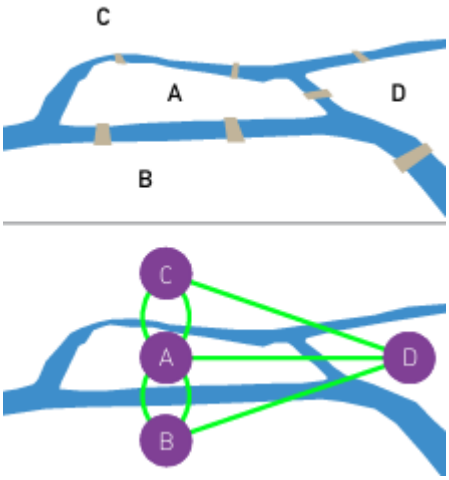


Figura 1-2. Abstracción del problema de los puentes de Königsberg.

Fuente: [8]

Para resolver el problema, Euler representó una ruta como una secuencia de letras, donde cada letra correspondía a una región de la ciudad. Por ejemplo, ADC representa la ruta que iba de A a D y luego de D a C, pasando por dos puentes. Así, para atravesar los siete puentes, la ruta debía contener ocho letras según esta representación. Euler dedujo que si el número m de puentes que llegaban a una región determinada era impar, entonces el

número de veces que dicha región debía aparecer en la ruta es $\frac{m+1}{2}$. Aplicando este principio a las diferentes regiones y las rutas que las conectaban, se concluyó que la región A, con cinco puentes, debía aparecer tres veces en la ruta, mientras que las demás regiones, con tres puentes cada una, debían aparecer dos veces cada una en la ruta. Sin embargo, la suma de estas ocurrencias resultaba en un total de nueve letras, lo cual era imposible para una secuencia de ocho letras que se buscaba, llevando a la conclusión de que no era posible realizar la travesía comenzando desde cualquier región, pasando por todos los puentes una sola vez y regresando a la misma región de partida.

En el mismo artículo [1], Euler examinó la situación más general, buscando dar respuesta a qué ocurriría si el número de puentes que llegaban a una región era par. En este caso, concluyó que el número de ocurrencias de dicha región en la ruta sería la mitad del número de puentes que la conectaban.

Como consecuencia de este estudio llevado a cabo por Euler, se estableció lo siguiente:

- El viaje es posible si todos los vértices del grafo tienen grado par, es decir, si cada región tiene un número par de puentes que la conectan. Este tipo de grafos da lugar a la definición de ciclos eulerianos, nombrados así en honor a Leonhard Euler, que representan cualquier camino que recorre un grafo partiendo desde un vértice y es capaz de recorrer todas las aristas una única vez para regresar nuevamente al vértice inicial.

- Si el grafo tiene exactamente dos vértices con grado impar, es decir, que tengan conectados un número de aristas impar, es posible realizar el recorrido, siempre que se parta de uno de estos vértices y se termine en el otro. Estos son conocidos como caminos eulerianos, donde el punto inicial y final son distintos, a diferencia de los ciclos eulerianos.

Estas conclusiones se derivan de la distinción entre nodos de transición, que son los nodos intermedios de la ruta, y los nodos de inicio y fin. Los nodos de transición deben tener un número par de enlaces, actuando como "puente" entre un vértice y otro, requiriendo al menos una arista de llegada y otra de salida. Por otro lado, los nodos de inicio y fin deben tener exactamente un enlace de entrada o salida, junto con pares de enlaces para permitir múltiples visitas al vértice.

A partir del problema de los puentes de Königsberg, varios hitos han ido marcando el desarrollo de la teoría de grafos. En el siglo XIX, se destacan algunos eventos significativos. En 1847, Gustav Kirchhoff hizo uso de la teoría de grafos para el desarrollo de las conocidas leyes de Kirchhoff, siendo esta la primera aplicación de los grafos a la ingeniería y, en concreto, al análisis de redes eléctricas. Poco después, surgió el famoso problema de los cuatro colores [3], que planteaba la posibilidad de colorear un mapa geográfico con solamente cuatro colores, con la restricción de que países vecinos no compartieran el mismo color. Aunque la resolución de este problema no se alcanzaría hasta el siglo siguiente, su estudio y análisis fomentaron el desarrollo de la teoría de grafos. Este periodo fue fundamental para definir numerosos conceptos y propiedades de los grafos en la búsqueda de la solución deseada.

Sin embargo, antes de la resolución del problema de los cuatro colores, ya se habían hecho importantes contribuciones a la teoría de grafos. En 1936, Dénes Kőnig, un matemático húngaro de origen judío, publicó el primer libro sobre teoría de grafos [4], sentando las bases para el desarrollo posterior de esta rama de las matemáticas.

1.2 Aplicaciones de los grafos

La teoría de grafos encuentra una asombrosa cantidad de aplicaciones en el mundo real [5], desde la logística hasta la biología, desde la informática hasta las redes sociales. Explorar estas aplicaciones revela la versatilidad y la potencia de esta disciplina, así como su papel fundamental en la resolución de problemas complejos en una amplia gama de campos.

Comenzando por el ámbito de la logística y la planificación de rutas, los grafos son esenciales para optimizar el transporte de bienes y personas. Por ejemplo, en el caso de Google Maps, los grafos se utilizan para modelar la red de carreteras, donde los nodos representan ubicaciones y las aristas representan las conexiones entre ellas. Esto permite calcular la ruta más corta entre dos ubicaciones y optimizar los tiempos de viaje, lo que es fundamental para aplicaciones de navegación y planificación de viajes.

En el campo de la ingeniería y el diseño de circuitos, los grafos son fundamentales para la síntesis y optimización de circuitos electrónicos. Aquí, los nodos representan componentes electrónicos y las aristas representan conexiones entre ellos. Mediante el uso de algoritmos de grafos, es posible diseñar circuitos eficientes y optimizar su rendimiento, lo que es fundamental para el desarrollo de dispositivos electrónicos modernos.

En el ámbito de las redes sociales y la sociometría, los grafos se utilizan para modelar y analizar las relaciones entre individuos. Cada usuario de una red social puede representarse como un nodo en un grafo, y las conexiones entre usuarios se representan como aristas. Esto permite analizar la estructura de la red social, identificar comunidades y grupos de interés, y comprender la difusión de información y la influencia dentro de la red.

En el campo de la biología y la bioinformática, los grafos se utilizan para modelar y analizar redes biológicas, como las interacciones entre proteínas en una célula. Aquí, los nodos representan moléculas biológicas y las aristas representan interacciones entre ellas. Mediante el uso de algoritmos de grafos, es posible identificar patrones y estructuras en estas redes, lo que es fundamental para comprender los procesos biológicos y desarrollar nuevos tratamientos médicos.

En la economía digital, empresas líderes como Google y Facebook se apoyan en redes y algoritmos desarrollados por científicos de redes para impulsar desde la búsqueda web hasta la publicidad dirigida. Estos algoritmos, basados en el análisis de redes, permiten comprender y aprovechar las complejas interacciones en línea, generando grandes cantidades de datos valiosos y proporcionando una experiencia de usuario personalizada. La capacidad para analizar influencias, predecir tendencias y optimizar recursos se ha convertido en un activo clave en la competitiva economía digital, redefiniendo los fundamentos de la innovación y la eficiencia en línea.

Desde el ámbito de la seguridad, la ciencia de redes emerge como una herramienta crucial en la lucha contra las organizaciones criminales y otras amenazas globales. A través del análisis de redes, los expertos pueden mapear y comprender las complejas relaciones entre individuos, organizaciones y entidades, permitiendo una visión detallada de su estructura y dinámica. Al utilizar técnicas de grafos y algoritmos especializados, los analistas pueden identificar nodos clave dentro de estas redes, así como patrones de comunicación y comportamiento que podrían indicar actividades sospechosas.

Además, los grafos tienen aplicaciones en la optimización de procesos y la gestión de proyectos, donde se utilizan para modelar y analizar relaciones entre tareas y recursos. Por ejemplo, en la gestión de proyectos, los nodos pueden representar tareas y las aristas pueden representar dependencias entre ellas. Esto permite optimizar la programación de tareas y recursos, lo que es fundamental para garantizar la finalización oportuna y eficiente de proyectos.

En resumen, la teoría de grafos es una herramienta poderosa y versátil que se utiliza en una amplia variedad de campos para modelar, analizar y resolver problemas complejos en el mundo real. En áreas como la medicina, la ciberseguridad, las telecomunicaciones y la economía, los grafos desempeñan un papel fundamental en la comprensión y optimización de sistemas complejos, lo que los convierte en una herramienta indispensable para la investigación y la innovación en numerosas disciplinas.

1.3 Resumen

En resumen, en esta introducción hemos explorado la importancia de los grafos y las redes en el análisis de sistemas complejos en una amplia gama de disciplinas. Los grafos, estructuras matemáticas compuestas por nodos y aristas, proporcionan un marco formal para representar relaciones entre entidades de manera abstracta pero efectiva. Por otro lado, las redes se centran en modelar y analizar sistemas reales, utilizando los grafos como una representación matemática.

Desde su surgimiento ligado al famoso problema de los puentes de Königsberg hasta las numerosas aplicaciones en campos como la logística, la biología, la ingeniería y las redes sociales, los grafos y las redes juegan un papel fundamental en la comprensión y optimización de sistemas complejos. Su capacidad para representar relaciones y estructuras de manera intuitiva y elegante los convierte en herramientas indispensables para abordar una variedad de problemas prácticos en el mundo moderno.

2 FUNDAMENTOS DE LA TEORÍA DE GRAFOS

En este capítulo se establecen los principios fundamentales para la comprensión y análisis de la teoría de grafos, proporcionando una base sólida para el entendimiento de los algoritmos y técnicas que se describirán posteriormente. Se tratan conceptos tales como la definición de grafos, sus propiedades básicas y la distinción entre los distintos tipos de grafos. Además, se abordan conceptos relacionados con la robustez de la red y algunas herramientas que pueden usarse para implementarlos.

2.1 Definición de grafo

Un grafo se define como un conjunto de objetos llamados nodos o vértices, los cuales están conectados entre sí mediante una serie de enlaces llamados aristas o arcos. Estas conexiones permiten representar las relaciones existentes entre los elementos del conjunto. Generalmente, para ilustrar un grafo se emplean puntos para representar los nodos, que se encuentran unidos por líneas que representan las aristas. En la siguiente figura se muestra un ejemplo de representación de un grafo.

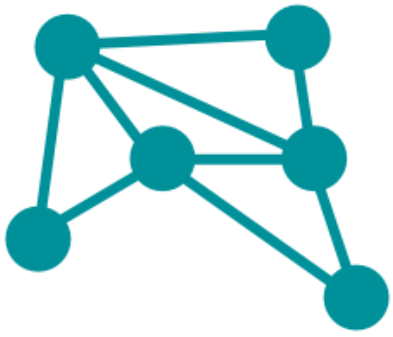


Figura 2-1. Ejemplo representación de un grafo.

Fuente: https://www.setzeus.com/community-blog-posts/graph-theory-basic-properties

Un grafo $\boldsymbol{G}$ se denota mediante un par ordenado de elementos de la siguiente forma $\boldsymbol{G = (v,\ \varepsilon)}$ donde:

$$v = \{A, B, C, D, E\} \tag{2-1}$$

$$\varepsilon = \{(v_i, v_j)\ \forall i, j : exista\ una\ arista\ en\ G\ entre\ ellos\} \tag{2-2}$$

La ecuación (2-1) representa el conjunto de vértices que componen el grafo $\boldsymbol{G}$, mientras que la ecuación (2-2)

contiene el conjunto de aristas que relacionan estos vértices o nodos. Cada elemento de ε contiene una tupla con un par de valores que representan el nodo origen y el nodo destino de la arista, en ese orden. Es importante tener en cuenta el orden de estos nodos en grafos dirigidos, donde las conexiones son posibles en una sola dirección (unidireccional), pero es menos relevante en grafos no dirigidos, donde las aristas son bidireccionales [6]. En la siguiente sección, se abordarán con mayor profundidad los distintos tipos de grafos.

2.3. Tipos de grafos

En la teoría de grafos clásica, el término 'grafo' solía referirse exclusivamente a los grafos simples, los cuales cumplen con la condición de que sus nodos pueden estar conectados únicamente mediante una relación o arista. Sin embargo, en la actualidad, este término se emplea de manera más amplia para referirse indistintamente a los tres tipos principales de grafos, que son, de acuerdo con [7]:

- ***Grafos simples*:** aquellos en los que un par de nodos pueden estar conectados únicamente por una arista.
- ***Multigrafos*:** aquellos en los que un par de nodos pueden estar conectados por más de una arista.
- ***Pseudografos*:** multigrafos en los que puede haber más de una relación o arista entre un par de nodos, y donde los nodos pueden estar relacionados también consigo mismos.

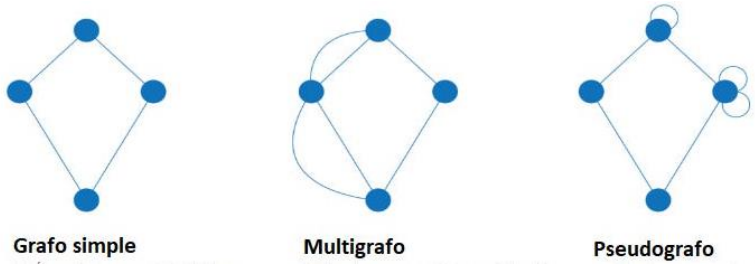


Figura 2-2. Tipos clásicos de grafos.

Fuente: [7]

Sin embargo, esta no es la única distinción posible. Los grafos poseen una serie de atributos comunes que nos permiten clasificarlos en diversos grupos, como se menciona en [7].

2.3.1. Conectado o No Conectado

Se dice que un grafo es conectado si existe un camino entre todo par de nodos que lo componen, es decir, si desde cualquier nodo del grafo se puede acceder a cualquier otro mediante una sucesión de vértices y aristas. Si esta condición no se cumple, se afirma que el grafo es no conectado y está formado por varios subgrafos aislados entre sí, que reciben el nombre de componentes (o *clusters*). Estos componentes son conjuntos de nodos y aristas que están internamente conectados, pero carecen de conexión directa con los nodos de otros componentes.

Si una red consiste en dos componentes, un enlace colocado adecuadamente puede conectarlos, haciendo que la red esté conectada. Este enlace se denomina puente, y su eliminación provoca la desconexión de la red.

Comprobar si una red está conectada o desconectada puede parecer una tarea sencilla en redes pequeñas. Sin embargo, en redes con millones de nodos, el estudio de la conectividad se complica. Para ello, se pueden emplear

diversas herramientas matemáticas y algorítmicas. Por ejemplo, en una red desconectada, la matriz de adyacencia puede reorganizarse en una forma diagonal por bloques, de tal manera que todos los elementos no nulos de la matriz estén contenidos en bloques cuadrados a lo largo de la diagonal, mientras que todos los demás elementos son cero. Cada bloque cuadrado corresponde a un componente, como se verá posteriormente al tratar la matriz de adyacencia. En la práctica, la técnica más eficiente para detectar componentes en redes grandes es el algoritmo de BFS (Breadth-First Search), que es un algoritmo de búsqueda que recorre los nodos del grafo comenzando por uno arbitrario, para luego explorar todos sus vecinos.

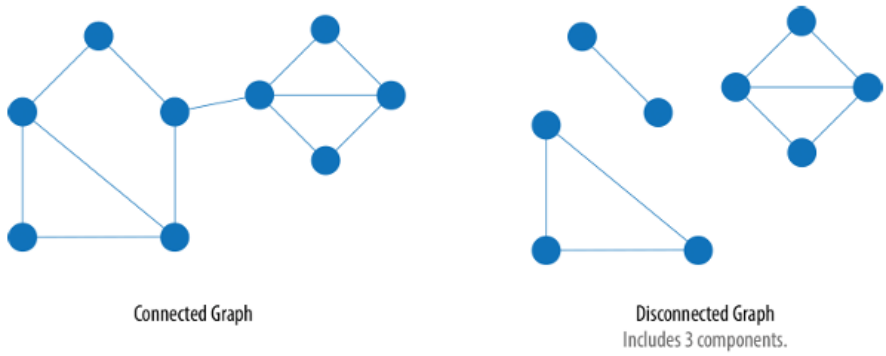


Figura 2-3. Ejemplos de grafo conectado y no conectado.

Fuente: [7]

2.3.2. Ponderado o No Ponderado

Esta distinción se basa en el coste asociado a las aristas que conforman el grafo. Un grafo ponderado es aquel que tiene asignados valores o pesos a sus aristas o a sus nodos. Esta asignación puede utilizarse para representar diversas características de las relaciones, como la distancia, el coste, el tiempo o la capacidad. La mayoría de las redes en el mundo real son ponderadas. Por ejemplo, en un grafo que represente varios lugares emblemáticos de una ciudad, los costes asociados a las aristas que conectan estos lugares podrían representar la distancia en kilómetros entre ellos. En redes que modelan llamadas telefónicas, el peso de los enlaces puede estar asociado al número de minutos que las dos personas hablan.

En contraste, un grafo no ponderado es aquel en el que no existen costes asociados ni a las aristas ni a los nodos. En este tipo de grafo, las aristas simplemente indican la existencia de una conexión entre los nodos, sin tener en cuenta ningún valor asociado a dicha conexión. Por ejemplo, se podría construir un grafo que represente las amistades entre diferentes personas; en este caso, la existencia del enlace indicaría que ambas personas son amigas.

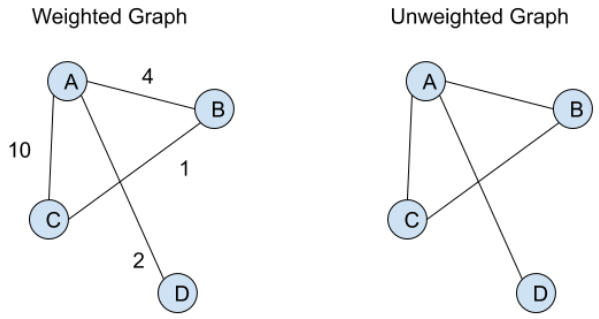


Figura 2-4. Ejemplos de grafo ponderado y no ponderado.

Fuente: https://medium.com/swlh/data-structures-intro-graphs-af165b1e2c95

2.3.3. Dirigido o No Dirigido

Un grafo se cosidera no dirigido cuando las relaciones que contiene son bidireccionales, lo que significa que las aristas entre los nodos no tienen ninguna dirección específica. Como consecuencia, la arista (v_i, v_j) representa la misma relación que (v_j, v_i), puesto que la relación es simétrica. Este sería el caso, por ejemplo, de un grafo que represente un grupo de amigos, donde la relación de amistad es recíproca entre cada miembro del grupo.

Por otro lado, un grafo se clasifica como dirigido si las relaciones que lo constituyen tienen una dirección específica y no son recíprocas. En este caso, una arista (v_i, v_j) representa la relación que parte desde el nodo v_i hasta el nodo v_j. Este tipo de grafo, también conocido como dígrafo, distingue entre dos tipos de relaciones por cada nodo: las relaciones entrantes, que corresponden a aquellas que terminan en el nodo, y las relaciones salientes, que se originan en él. Un ejemplo de este tipo de grafo sería un sistema de carreteras de una ciudad, donde los cruces más importantes se representan como nodos, y las carreteras que los conectan se representan como aristas. Dependiendo de si la carretera es de sentido único o de doble sentido, se representará mediante una arista en la dirección de la carretera o dos aristas en ambas direcciones entre los cruces.

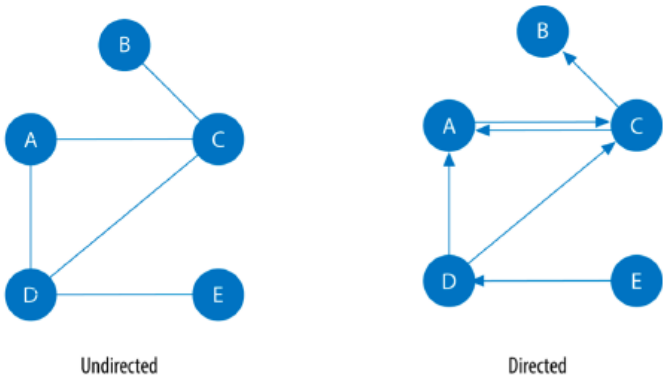


Figura 2-5. Ejemplos de grafo no dirigido y dirigido.

Fuente: [7]

2.3.4. Cíclico o Acíclico

Para explicar la diferencia entre un grafo cíclico y un acíclico, es necesario definir previamente el concepto de ciclo en la teoría de grafos. Un ciclo se produce cuando, partiendo de un determinado nodo origen del grafo, existe al menos un camino que recorre diferentes aristas y vértices, permitiendo regresar al mismo nodo origen.

Cuando esta propiedad se presenta al menos una vez en un grafo, se establece que este es cíclico. Por el contrario, si ningún camino en el grafo cumple con la condición de regresar al nodo de origen, entonces se afirma que el grafo es acíclico.

Es importante destacar que, en el caso de un grafo dirigido, encontrar un posible ciclo implica respetar la dirección de las relaciones entre los vértices. Esto significa que el recorrido debe seguir las flechas que indican la dirección de las aristas como se puese observar en la siguiente figura.

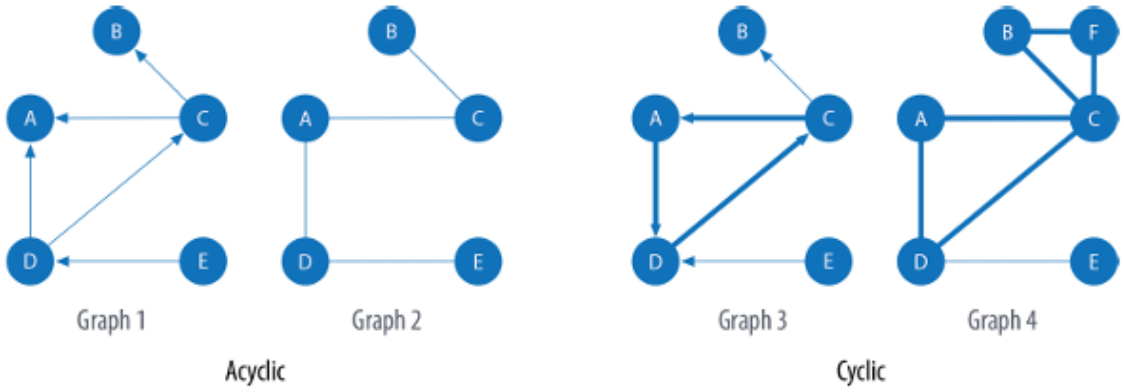


Figura 2-6. Ejemplos de grafo acíclico y cíclico.

Fuente: [7]

2.3.5. Disperso o Denso

La densidad de un grafo es una medida que indica qué tan conectados están los nodos entre sí, es decir, cuántas relaciones posee cada vértice. Cuando cada vértice del grafo está conectado a todos los demás, se dice que la densidad es máxima, y en este caso, el grafo se denomina grafo completo.

Para determinar si un grafo es disperso o denso, se calcula su densidad. Comparando este valor con la densidad máxima, que corresponde al caso de un grafo completo, se establece que el grafo es denso si ambos valores son suficientemente próximos. No existe un umbral estricto para esta clasificación, ya que depende del contexto y de la interpretación del usuario.

Las redes reales tienden a ser dispersas, con una fracción muy pequeña de los enlaces posibles presentes en comparación con un grafo completo. Esto se debe a que la mayoría de los nodos en las redes reales tienen solo unas pocas conexiones en relación con el número total posible de conexiones. Por ejemplo, en una red social como Facebook, aunque cada usuario puede tener cientos o incluso miles de amigos, estas conexiones individuales representan solo una fracción insignificante del total de posibles conexiones entre todos los usuarios de la plataforma.

La dispersión de las redes reales tiene importantes implicaciones en su almacenamiento y exploración. Las matrices de adyacencia de las redes reales son generalmente dispersas, lo que significa que la mayoría de sus elementos son ceros. Por lo tanto, es más eficiente almacenar solo la lista de enlaces no nulos en lugar de la matriz completa, ahorrando así espacio de memoria.

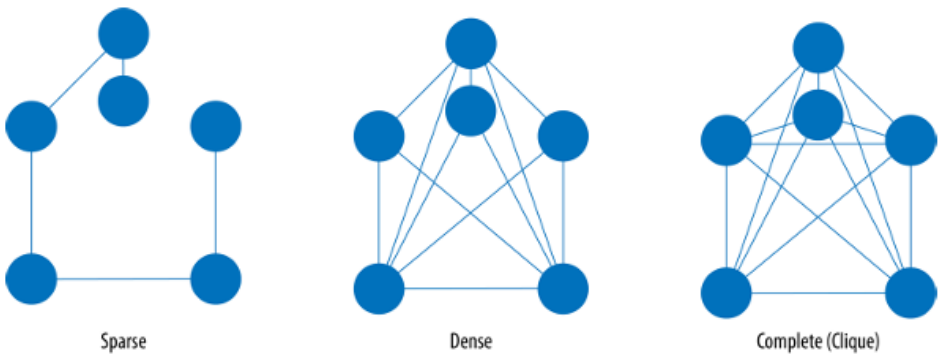


Figura 2-7. Ejemplos de grafo disperso, denso y completo.

Fuente: [7]

2.3.6. Monopartito, Bipartito o K-partito

Para el estudio de los grafos, la mayoría de ellos se consideran que tienen un único tipo de nodo y relación, lo que facilita la aplicación de algoritmos de análisis. Este tipo de grafo recibe el nombre de monoportatito.

Sin embargo, hay ocasiones en las que un grafo presenta vértices que se pueden dividir en dos conjuntos distintos, resultando en dos tipos de nodos. En estos grafos, conocidos como bipartitos, las relaciones solo ocurren entre nodos de diferentes conjuntos; es decir, las aristas no pueden conectar vértices del mismo conjunto. Este concepto puede generalizarse al de un grafo K-partito, donde los nodos se pueden clasificar en K conjuntos.

Un ejemplo de grafo bipartito podría ser un grafo de asignación de tareas. Donde se tiene un conjunto de empleados y un conjunto de tareas que deben ser completadas. Cada arista conecta un empleado con una tarea que está asignada a él o ella. En este caso, los nodos se pueden dividir en dos conjuntos distintos: el conjunto de empleados y el conjunto de tareas, y las aristas solo conectan nodos de diferentes conjuntos.

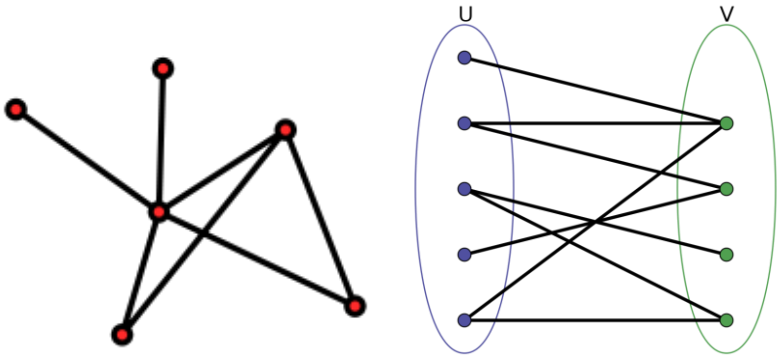


Figura 2-8. Ejemplos de grafo monopartito y bipartito.

Fuente: https://es.m.wikipedia.org/wiki/Archivo:Grafo_simple.svg
https://www.wikiwand.com/es/Grafo_bipartito

Es interesante destacar que es posible representar los grafos K-partitos (incluyendo los grafos bipartitos como una especificación con dos conjuntos) proyectándolos sobre uno de los conjuntos. Para representar las relaciones, se utiliza una arista que conecta directamente cada par de nodos que antes estaban relacionados a través de un nodo de otro tipo. En la figura 2-9 se muestra un ejemplo de grafo bipartito y sus proyecciones sobre ambos conjuntos de nodos. Se puede observar que, por ejemplo, en la proyección sobre el conjunto U, el nodo u_4 está directamente conectado con u_1 y u_2, ya que en el grafo bipartito estos nodos están relacionados a través del nodo v_1.

Otro ejemplo representativo de este tipo de grafo es la red de actores de Hollywood, en la cual un conjunto de nodos corresponde a películas y el otro a actores. Una película está conectada a un actor si este participa en dicha película. Una proyección de esta red bipartita es la red de actores, en la que dos nodos están conectados si actuaron en la misma película. La otra proyección es la red de películas, en la que dos películas están conectadas si comparten al menos un actor en su reparto.

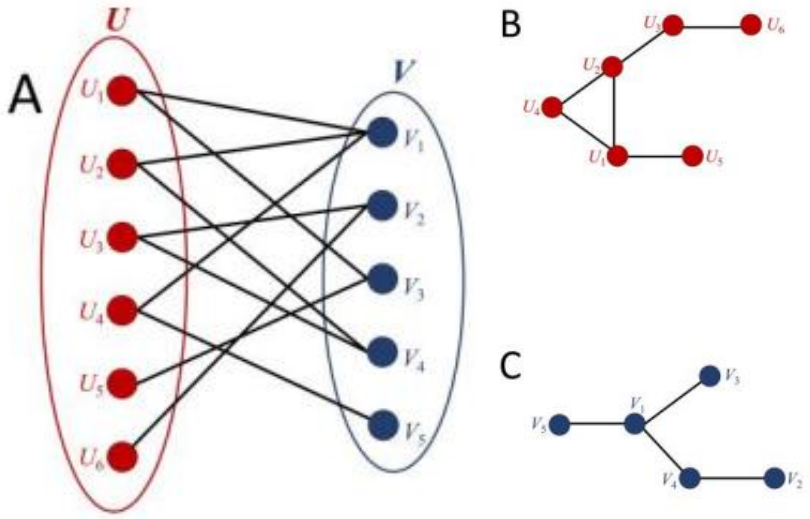


Figura 2-9. Ejemplo de grafo bipartito y sus proyecciones.

Fuente: [8]

2.4. Aspectos fundamentales de los grafos

A continuación, se describen varias características y aspectos importantes que diferencian y definen a los grafos, permitiendoel uso de diversos algoritmos para su análisis [6].

- ***Orden y tamaño del grafo***: se trata de dos medidas fundamentales que caracterizan un grafo y proporcionan información relevante sobre la estructura del mismo.

  El orden del grafo, denotado como N, se refiere al número total de nodos que componen el grafo. Este concepto nos permite entender la cantidad de entidades o elementos representados en el grafo. Por ejemplo, en una red social, el orden del grafo podría indicar el número de usuarios registrados.

  El tamaño del grafo, denotado como L, se refiere al número total de aristas que conectan los nodos del grafo. El tamaño del grafo proporciona información sobre la densidad de las relaciones dentro de la red. Por ejemplo, en una red de transporte, el tamaño del grafo podría indicar la cantidad total de conexiones directas entre diferentes ubicaciones.

- ***Grado de un nodo***: el grado de un nodo se define como el número de aristas que inciden sobre ese nodo y se denota como k_v para el nodo v. Además, se define el parámetro de conectividad media de la red, representado como $< k >$. Este valor se calcula dividiendo la suma total de los grados de todos los nodos que conforman el grafo entre el número total de nodos. Es importante diferenciar entre grafos dirigidos y no dirigidos para comprender plenamente este concepto.

  En un grafo no dirigido, el grado de un nodo se corresponde al número de aristas que tenga conectadas, consecuentemente un mismo enlace será incluido en el grado de los dos nodos que conecta. Esto nos lleva a la ecuación (2-3), que indica que la suma de los grados de todos los nodos es igual a dos veces el número de aristas del grafo, ya que cada arista se tiene en cuenta dos veces.

$$\sum_{v} k_v = 2L \tag{2-3}$$

Nos queda definir la conectividad media para grafos no dirigidos, para lo que bastaría con dividir el resultado anterior entre el número total de nodos:

$$< k >= \frac{2L}{N} \tag{2-4}$$

En el caso de grafos dirigidos, el cálculo de estos grados es diferente. Como las aristas tienen una única dirección, se pueden distinguir entre dos tipos de grados para un nodo: el grado de entrada (k_v^{in}), que corresponde a los enlaces entrantes en el nodo, y el grado de salida (k_v^{out}), que se refiere a los enlaces que parten del nodo. Por lo tanto, el grado de un nodo en grafos dirigidos es la suma de su grado de entrada y su grado de salida. En este caso, cada enlace será incluido una vez en el grado de entrada del nodo al que se diriga y otra en el grado de salida del nodo emisor. Así, la suma de los grados de entrada de todos los nodos es igual a la suma de los grados de salida y, por tanto, también al número total de aristas:

$$\sum_{v} k_v^{in} = \sum_{u} k_u^{in} = L \tag{2-5}$$

Además se sigue cumpliendo así la ecuación (2-3) entendiendo k_v como la suma de k_v^{in} y k_v^{out}. Sin embargo, ahora el valor de la conectividad media lo podemos dividir en los dos tipos de grado tal que:

$$< k^{in} >=< k^{out} >= \frac{L}{N} \tag{2-6}$$

La distribución de grado, p_k, proporciona la probabilidad de que un nodo seleccionado al azar en la red tenga grado k. Esta se ha asumido un papel fundamental en la teoría de redes, ya que puede ser empleado para el cálculo de diversas propiedades de la red, incluida la conectividad media y el estudio de la robustez de la red, entre otros. Dado que p_k es una probabilidad, debe cumplir con la normalización:

$$\sum_{k=1}^{\infty} p_k = 1 \tag{2-7}$$

Para una red con N nodos, la distribución de grado normalizada se expresa como:

$$p_k = \frac{N_k}{N} \tag{2-8}$$

Donde N_k es el número de nodos de grado k. Por lo tanto, el número de nodos de grado k se puede calcular a partir de la distribución de grado como: $N_k = N * p_k$.

- ***Densidad del grafo***: como se ha comentado en el apartado anterior sobre la diferenciación de los tipos de grafos, la densidad es una medida que indica la cantidad de aristas que existen entre los nodos del grafo y puede ser empleada para la clasificación de los grafos en varios tipos. Para calcular la densidad, se compara el número máximo de aristas que tendría el grafo si fuese completo con el número real de aristas que posee.

  Para calcular el número máximo de aristas en un grafo de N nodos, podemos utilizar la siguiente fórmula:

$$MaxSize = \binom{N}{2} = \frac{N(N-1)}{2} \tag{2-9}$$

Donde $MaxSize$ representa el máximo número de aristas que tendría el grafo competo. Luego, para calcular la densidad del grafo anterior, basta con dividir el número real de enlaces L entre el valor obtenido de la fórmula anterior:

$$D = \frac{Size}{MaxSize} = \frac{L}{\binom{N}{2}} \tag{2-10}$$

Este valor D estará en el rango de 0 a 1, donde 1 corresponde a un grafo completo, es decir, un grafo en el que todos los nodos están conectados entre sí por aristas.

- ***Matriz de adyacencia***: esta matriz es una herramienta fundamental en el estudio y análisis de redes y grafos. Mediante la matriz de adyacencia se puede proporcionar una descripción clara de las relaciones entre los nodos de un grafo. Esta matriz tiene dimensiones $N \times N$, donde N es el número total de nodos en el grafo. Tanto las filas como las columnas de esta matriz representan cada uno de los nodos, y los elementos de la matriz indican si existe una arista que conecta los nodos correspondientes. Específicamente, siendo A la matriz de adyacencia, sus elementos, denotados como A_{ij}, quedarían de la siguiente manera:

  $A_{ij} = 1$ si existe una arista desde el nodo i hasta el nodo j

  $A_{ij} = 0$ si no existe ninguna arista entre el nodo i y el nodo j

  Es importante señalar que hay una pequeña distinción entre los grafos dirigidos y los no dirigidos en relación a la matriz de adyacencia. En el caso de los grafos no dirigidos, la matriz de adyacencia es simétrica, lo que significa que la matriz es igual a su transpuesta ($A = A^T$). Esto refleja la naturaleza bidireccional de las aristas en los grafos no dirigidos, donde la conexión entre dos nodos es recíproca y no tiene una dirección específica. Por lo tanto, si existe una arista entre el nodo 1 y el nodo 2 tanto A_{12} como A_{21} serán igual a uno. A continuación se muestra un ejemplo de una matriz de adyacencia para un grafo no dirigido:

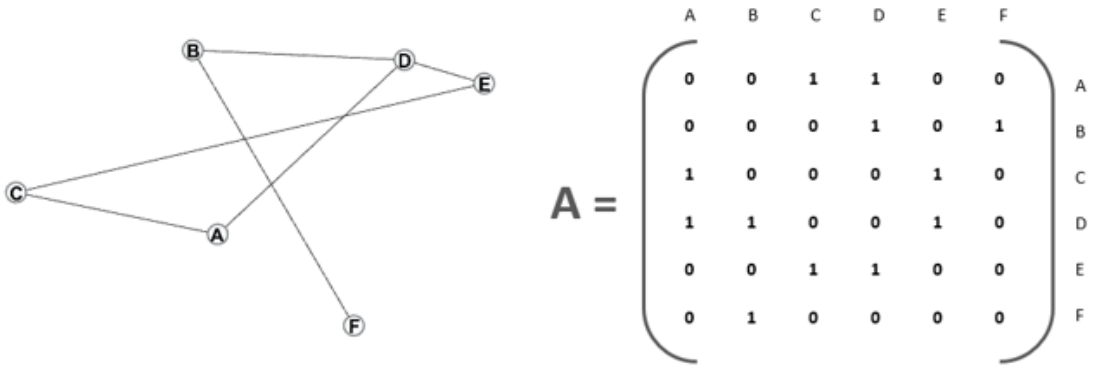


Figura 2-10. Matriz de adyacencia de un grafo no dirigido

Fuente: Elaboración propia

En el caso de grafos dirigidos, la matriz de adyacencia ya no es simétrica, puesto que las relaciones a través de enlaces tienen una dirección única. Por ejemplo, si existe un enlace desde el nodo 1 hasta el nodo 2, la entrada correspondiente en la matriz de adyacencia sería $A_{12} = 1$, indicando que hay un enlace desde el nodo 1 hacia el nodo 2. Sin embargo, la entrada correspondiente con la dirección opuesta, es decir, desde el nodo 2 hacia el nodo 1, sería $A_{21} = 0$, ya que no hay enlace en esa dirección.

Por convenio, se define la matriz de adyacencia de manera que el nodo representado por la fila indica el nodo de origen del enlace, mientras que el nodo representado por la columna indica el nodo de destindo del enlace. Siguiendo este convenio, la matriz *A* del ejemplo anteriormente mencionado tendría un uno en el elemento de la fila 1 y columna 2 para indicar el enlace del nodo 1 al nodo 2, y un cero en el elemento de la fila 2 y columna 1 para indicar que no hay enlace en la dirección opuesta. A continuación, se muestra un ejemplo de una matriz de adyacencia para un grafo dirigido:

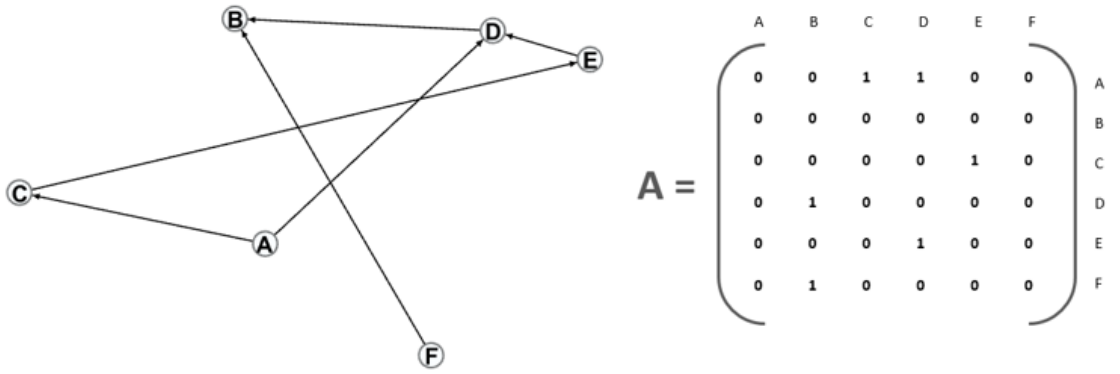


Figura 2-11. Matriz de adyacencia de un grafo dirigido

Fuente: Elaboración propia

Además, en el caso de grafos ponderados, los elementos de la matriz de adyacencia no serían simplemente uno o cero para indicar la presencia o ausencia de un enlace, sino que se corresponderían con la ponderación o peso de dicho enlace en el grafo. En otras palabras, el valor en cada entrada de la matriz sería el peso del enlace entre los nodos correspondientes, en lugar de simplemente uno o cero.

Por último, es importante destacar algunos de los usos que hacen de la matriz de adyacencia una

herramienta tan importante y representativa en el análisis de grafos. Uno de estos usos es la determinación del grado de los nodos, propiedad que se comentó anteriormente relacionada con el número de aristas de cada nodo. En grafos no dirigidos, para calcular el grado de un nodo basta con sumar todos los elementos de la misma fila o columna que se corresponden con el nodo en cuestión. De tal forma que el grado k_i del nodo i se puede obtener mediante:

$$k_i = \sum_{j=1}^{N} A_{ij} = \sum_{j=1}^{N} A_{ji} \tag{2-11}$$

En grafos dirigidos, siguiendo el convenio mencionado anteriormente, basta con sumar todos los elementos de la columna correspondiente al nodo para obtener el grado entrante, y todos los elementos de la fila correspondiente para obtener el grado saliente.

$$k_i^{in} = \sum_{j=1}^{N} A_{ji} \ , \qquad k_i^{out} = \sum_{j=1}^{N} A_{ij} \tag{2-12}$$

Además, las potencias de la matriz de adyacencia tienen una interpretación muy interesante en la búsqueda de caminos entre nodos. Siendo A^n la matriz de adyacencia elevada a la potencia n, cada elemento (i, j) de esta matriz indica el número de caminos de longitud n que existen desde el nodo i hasta el nodo j. Esta propiedad es especialmente útil para comprender la conectividad y las rutas posibles dentro de un grafo, lo que facilita el análisis de la estructura y las relaciones entre los nodos. A continuación, se muestra un ejemplo de este uso con el grafo de la figura 2-10:

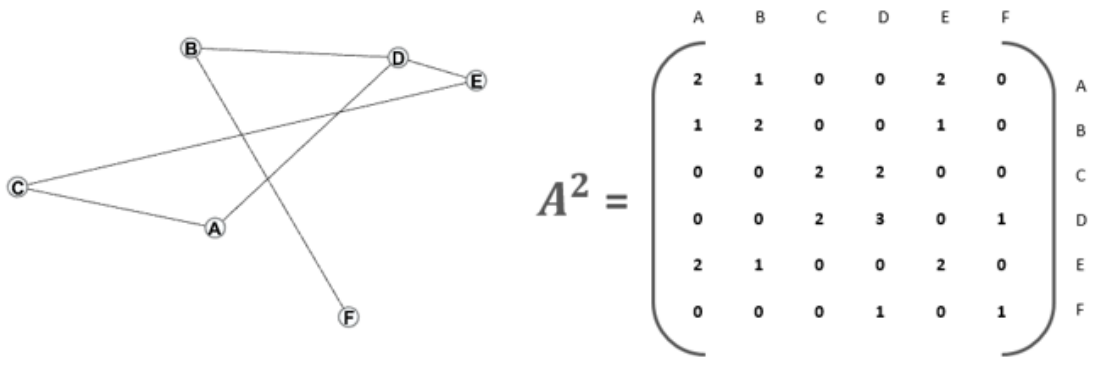


Figura 2-12. Ejemplo potencia de la matriz de adyacencia

Fuente: Elaboración propia

Como se puede observar en el cuadrado de la matriz de adyacencia reflejada en la figura anterior, los elementos de dicha matriz indican el número de caminos de longitud 2 que existen entre los nodos del grafo. Por ejemplo, entre el nodo A y E, se indica que hay dos posibles caminos de longitud 2. Al

examinar el grafo, se puede observar que uno de estos caminos pasa a través del nodo C, mientras que el otro atraviesa el nodo D.

En apartados anteriores, hemos mencionado cómo los grafos pueden ser clasificados en conectados y no conectados. La matriz de adyacencia puede ser utilizada también para identificar componentes conectados dentro de un grafo no conectado. Como vimos anteriormente, es posible reordenar la matriz de adyacencia de tal manera que cada componente conectado del grafo se represente como un bloque cuadrado en la diagonal de la matriz. Los elementos no nulos dentro de estos bloques indican conexiones dentro de cada componente, mientras que los elementos fuera de estos bloques son cero, indicando la falta de conexiones entre componentes distintos.

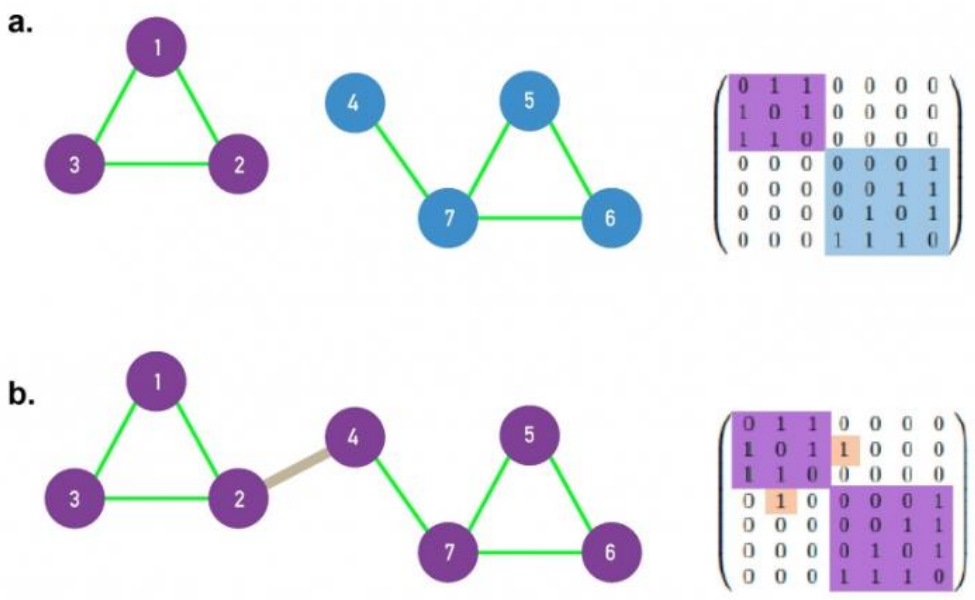


Figura 2-13. Conectividad de grafos con la matriz de adyacencia

Fuente: [8]

En la figura anterior, se puede apreciar un ejemplo de grafo no conectado (apartado a de la figura), en el que se puede identificar claramente los dos bloques en la matriz de adyacencia que representa ambos componentes, y como el resto de elementos es 0. En la parte b de la figura, podemos ver ese mismo ejemplo, pero con un enlace puente entre los dos componentes. En este caso, habrá dos elementos no nulos fuera de los cuadrados formados por los componentes en la matriz de adyacencia, que representan la conectividad o la existencia de este puente.

2.5. Estructuras de grafos

En este apartado se aborda la distinción entre distintos tipos de grafos según su estructura [7].

En primer lugar, se presenta el concepto de grafo aleatorio descrito en [8]. Este tipo de grafo consiste en nodos interconectados por aristas de forma aleatoria, con una probabilidad determinada. En un grafo aleatorio no se observa ningún tipo de jerarquía o patrón discernible que permita identificar nodos influyentes o distinguir áreas específicas. Todos los nodos son exactamente iguales, sin apreciar la existencia de ninguna característica destacable que haga que un nodo se diferencie de los demás.

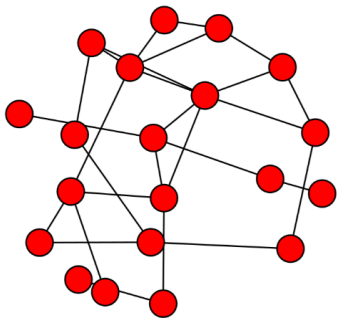


Figura 2-14. Ejemplo de grafo aleatorio

Fuente:https://math.libretexts.org/Bookshelves/Scientific_Computing_Simulations_and_Modeling/Introduction_to_the_Modeling_and_Analysis_of_Complex_Systems_%28Sayama%29/15%3A_Basics_of_Networks/15.06%3A_Generating_Random_Graphs

Uno de los modelos fundamentales para la generación de grafos aleatorios, es el modelo de Erdös-Rényi (modelo ER) [9], nombrado en honor a los matemáticos Paul Erdös y Alfréd Rényi, quienes lo desarrollaron. La idea fundamental tras este modelo consiste en que, dada una red de *N* nodos que inicialmente no tienen ninguna conexión entre sí, se procede a conectar pares de nodos de manera aleatoria, evitando repeticiones de nodos ya enlazados. Al repetir este proceso M veces, se generan finalmente M enlaces entre nodos. Dependiendo del valor de *M*, algunos nodos pueden permanecer desconectados si *M* es muy pequeño en comparación con el número total de nodos o, por el contrario, estar casi todos interconectados si *M* es lo suficientemente grande en relación con el número de nodos. Este proceso nos permite obtener la probabilidad $P(k)$ de que un nodo tenga k conexiones aleatorias en una red con valores grandes de *N* y *M* mediante la siguiente fórmula:

$$P(k) = e^{-z}\frac{z^k}{k!} \tag{2-13}$$

Para estas redes grandes, la distribución de *P(k)* es una distribución de Poisson de promedio en *z*, cuyo valor se obtiene de la siguiente ecuación:

$$z = \frac{2M}{N} \tag{2-14}$$

Otro modelo ampliamente utilizado para la generación de redes aleatorias es el modelo de configuración [6], especialmente útil como referencia para redes sociales en la vida real. A diferencia del modelo de Erdös-Rényi, en el modelo de configuración es necesario conocer los grados de los nodos para aplicar el algoritmo. Estos grados deben definirse por el número de medias aristas que posee un nodo, también conocidadas como "*stubs*", que es esencialmente la mitad del número total de enlaces. Es decir, para formar un enlace entre dos nodos habrá que unir dos medias aristas. El algoritmo consiste en formar un enlace entre dos nodos seleccionando dos medias aristas de forma aleatoria y repitiendo este proceso hasta que no queden medias aristas disponibles para asignar.

En las redes aleatorias, a medida que aumenta la probabilidad de conexión entre los nodos, aparece lo que se denomina componente gigante, denotada como G_c. La componente gigante representa la parte del grafo en la que la mayoría de los nodos están interconectados, dando lugar a una estructura que conecta prácticamente todos los nodos. Este fenómeno no ocurre de manera gradual, sino que se produce de manera abrupta, como un cambio de fase.

La transición hacia la presencia de la componente gigante se observa claramente en dos fases distintas: la fase subcrítica y la fase supercrítica. En la fase subcrítica, caracterizada por valores bajos de conectividad de los nodos ($< k >< 1$), los nodos tienden a estar desconectados o formar pequeñas componentes dispersas en el grafo. En esta fase, no hay presencia de componente gigante.

Cuando la probabilidad de conexión alcanza un umbral crítico ($< k >= 1$), que corresponde a una probabilidad de conexión entre nodos de $p_c = \frac{1}{N}$, donde N es el número total de nodos, emerge la componente gigante en la

red. A partir de este punto, ingresamos a la fase supercrítica, donde la componente gigante comienza a crecer rápidamente.

Finalmente, en la fase conectada, la componente gigante abarca todos los nodos de la red. En la siguiente figura se puede ver representada una gráfica con estas fases, donde *s* representa la proporción de nodos que pertencen a la componente gigante en función del total de nodos del grafo.

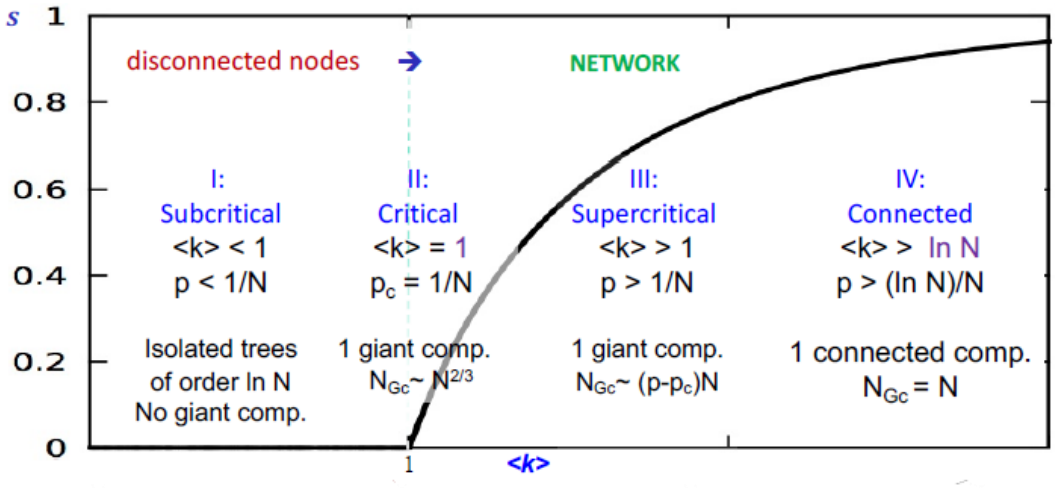


Figura 2-15. Evolución de una red aleatoria

Fuente: [6]

Una vez descrita la red aleatoria, otro de los tipos de grafo según su estructura son los grafos denominados "de mundo pequeño" [10]. Estos grafos se caracterizan porque la mayoría de los nodos no están directamente conectados entre sí; sin embargo, los nodos cercanos entre sí tienen múltiples conexiones, pero están conectados por un número reducido de enlaces a otros nodos más distantes. Lo distintivo de este tipo de grafos es que desde cualquier nodo se puede alcanzar otro nodo a través de un número relativamente pequeño de saltos.

Uno ejemplo ilustrativo de este fenómeno es la teoría de los seis grados de separación, propuesta por el escritor húngaro Frigyes Karinthy. Esta teoría sostiene que cualquier persona en el planeta está conectada a cualquier otra persona a través de, como máximo, seis conexiones, es decir, únicamente cinco intermediarios. Este concepto se basa en la idea de que a medida que agregamos conexiones en una cadena, el número de personas alcanzadas aumenta exponencialmente. Por lo tanto, se necesitaría un número reducido de conexiones para que esta red de contactos abarque a toda la población humana.

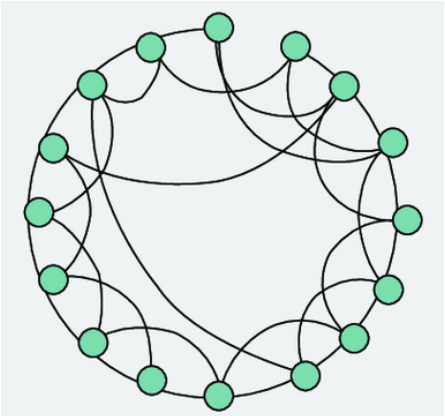


Figura 2-16. Ejemplo de grafo de mundo pequeño

Fuente: https://www.researchgate.net/figure/An-example-of-a-a-small-world-network-generated-using-the-WS-small-world-generation_fig2_224453450

La red de escala libre [8] es otro de los tipos de grafo que distinguimos según su estructura. Estos grafos se caracterizan porque su distribución de grado sigue la ley de potencias como se puede ver en la figura 2-17. Esta ley representa una relación funcional entre dos cantidades, donde un cambio en una de estas cantidades implica una variación proporcional a una potencia del cambio en la otra cantidad.

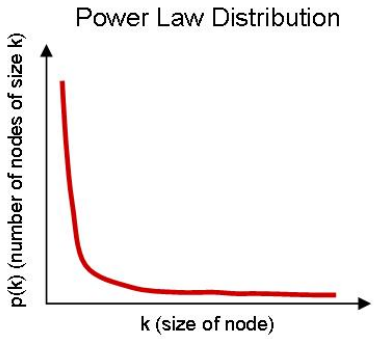


Figura 2-17. Ley de potencias.

Fuente: https://www.comscore.com/Insights/Blog/Part-2-Why-the-Power-of-Habit-Drives-Power-Law-Distributions-in-Mobile-App-Usage

En una red de escala libre, esta distribución de grado significa que solo hay unos pocos nodos altamente conectados, conocidos como “hubs”, que tienen una conectividad significativamente mayor en comparación con la mayoría de los nodos, que presentan un grado de conexión mucho más bajo. Es decir, una red de escala libre es aquella cuya distribución de grado sigue una ley de potencias.

La presencia de estos hubs es una de las características distintivas de las redes de escala libre y es lo que las hace especialmente interesantes para su estudio. La principal diferencia entre este tipo de redes y las redes aleatorias radica en su distribución de grado, la cual podemos comparar en la siguiente figura:

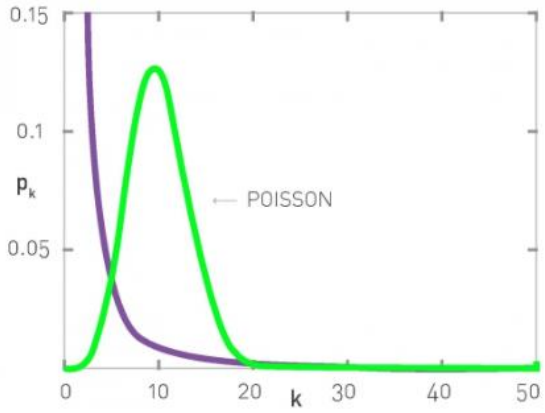


Figura 2-18. Comparación Ley de Potencias - Poisson.

Fuente: [8]

Como podemos ver en la figura, para valores pequeños de k la curva de la ley de potencias está por encima de la curva de Poisson, lo que indica que la probabilidad de observar un “hub” o un nodo de grado muy alto es

mucho mayor en redes de escala libre que en redes aleatorias. Lo mismo ocurre con aquellos nodos de grado elevado. Para un valor de k cercano a la conectividad media de la red, se puede apreciar que ahora la curva de Poisson está por encima, lo que sugiere que el número de nodos en una red aleatoria que tienen un grado similar a la conectividad media es muy alto en comparación con las redes de escala libre. En resumen, la diferencia clave entre una red aleatoria y una red de escala libre radica en las diferentes formas de la distribución de Poisson y de la función de ley de potencias: en una red aleatoria, la mayoría de los nodos tienen grados comparables y, por lo tanto, los "hubs" son poco comunes. En cambio, estos "hubs" no sólo se permiten, sino que se esperan en redes de escala libre. Además, cuantos más nodos tenga este tipo de red, más grandes serán sus "hubs". De hecho, el tamaño de los hubs crece polinomialmente con el tamaño de la red, por lo que pueden crecer considerablemente en redes de escala libre.

La propiedad más notable de las redes de escala libre es su invariancia de escala, lo que significa que su estructura y comportamiento no cambian significativamente cuando se escala la variable que representa el grado de los nodos, denotada como "k". Esto implica que, aunque la distribución de grado varíe enormemente entre los nodos, el patrón general de conectividad sigue siendo consistente en diferentes escalas. Esta propiedad es fundamental para comprender la robustez y la evolución de estas redes en diversas aplicaciones del mundo real.

Muchas redes reales son redes de escala libre. Un ejemplo reprentativo de ello es la World Wide Web. En la web, algunas páginas importantes como Google o Yahoo, actúan como hubs, con numerosos enlaces que las conectan con una gran cantidad de otras páginas. En contraste, la mayoría de las páginas web tienen solo unas pocas conexiones. Esta distribución de grados de escala libre es lo que permite que la web funcione eficientemente, ya que facilita la rápida difusión de información entre páginas altamente conectadas y las menos conectadas.

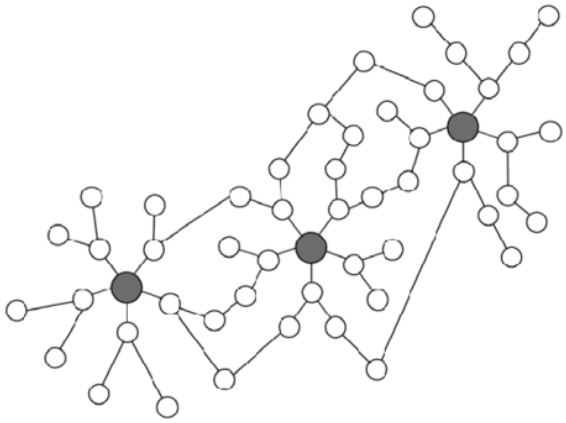


Figura 2-19. Ejemplo de grafo de escala libre.

Fuente: https://www.researchgate.net/figure/The-random-network-and-scale-free-network-A-Random-network-follows-Poisson_fig1_257301086

2.6. Robustez en las redes

La robustez de una red se define como su capacidad para mantener su correcto funcionamiento y estructura frente a fallos o ataques intencionados [8]. Esta característica es fundamental en el diseño y mantenimiento de redes reales, ya que una red robusta es capaz de resistir eventos inesperados sin que su funcionamiento se vea significativamente afectado.

Una herramienta clave en el estudio de la robustez de las redes es la teoría de percolación (o *percolation theory*). Esta teoría se centra en el estudio del comportamiento de una red cuando se agregan nodos o enlaces. Al llevar

a cabo un procedimiento inverso, es posible determinar cuántos nodos son necesarios eliminar del grafo para que la red se fragmente en componentes aislados. Esta fragmentación ocurre abruptamente cuando se alcanza una tasa crítica de eliminación f_c, a partir de la cual la componente gigante de la red desaparece. Por lo tanto, eliminar una pequeña fracción de nodos tendrá únicamente un impacto limitado en la integridad de la red, siempre y cuando esta fracción no alcance el umbral crítico mencionado anteriormente.

Para que exista una componente gigante en la red, la mayoría de los nodos que la conforman deben estar conectados a al menos otros dos nodos. Esto nos conduce al criterio de Molley-Reed, que establece que solo existe una componente gigante si se cumple la siguiente condición:

$$K = \frac{< k^2 >}{< k >} > 2 \tag{2-15}$$

Por lo tanto, las redes con $K < 2$ carecen de componente gigante. Aplicando el criterio anterior, la tasa de borrado critica f_c está determinada por la conectividad media de la red y su cuadrado, como se expresa en la ecuación (2-16).

$$f_c = 1 - \frac{1}{1 - \frac{< k^2 >}{< k >}} \tag{2-16}$$

En el caso de las redes de escala libre, es importante considerar tanto la posibilidad de sufrir ataques intencionados como la de que se produzcan errores aleatorios en la red. Para simular ambos escenarios, se procede a ir eliminando nodos de forma progresiva. Cuando se trata de fallos en la red, los nodos se escogerán de forma aleatoria, ya que estos fallos ocurren de manera impredecible y sin un patrón específico. Por otro lado, en el caso de que se trate de ataques intencionados, se seleccionan los nodos de forma estratégica, ya que el objetivo del ataque es impactar en áreas específicas de la red para causar el mayor daño posible.

Si la eliminación de los nodos es aleatoria, se puede observar que el valor de la frecuencia crítica tiende a ser cercano a uno. Esto indica que la red es muy robusta ante fallos puesto que es necesario eliminar casi todos los nodos para deshacer la componente gigante. Sin embargo, si se produce un ataque intencionado, el nodo escogido para su eliminación es intencionado y, por tanto, el valor de la frecuencia crítica es significativamente menor que uno. Esto implica que eliminar unos pocos nodos con alta conectividad puede afectar considerablemente la integridad de la red. Por lo tanto, las redes de escala libre son más vulnerables a los ataques que a los fallos aleatorios.

A continuación, se presenta la pregunta de cómo podemos mejorar la red para resistir de manera más efectiva los ataques, ya que este parece ser el punto débil en el correcto funcionamiento de las redes de escala libre [6]. Una solución inicial se relaciona con el valor de la tasa de borrado critica f_c. Como se explicó anteriormente, esta tasa marca el límite en el que la componente gigante de la red desaparece. Por lo tanto, cuanto mayor sea este umbral, más difícil será desmantelar la componente gigante, y se requerirá eliminar más nodos para lograrlo. Dado que este valor depende de la conectividad media de la red, una estrategia para mejorar la robustez de la red es aumentar dicha conectividad, lo que hará que la red sea más resistente ante los ataques.

Durante el estudio previo sobre la robustez, se asumió que la caída de los nodos se producía de forma independiente entre sí. Sin embargo, en una red real, la actividad de cada nodo está interconectada con la actividad de sus nodos vecinos. Por lo tanto, el fallo de un nodo podría provocar fallos en los nodos conectados a él, un fenómeno conocido como fallo en cascada. Este fenómeno está estrechamente relacionado con la conectividad de la red.

En un régimen subcrítico, donde la conectividad media es menor que uno, la propagación del fallo en cascada termina rápidamente, ya que los nodos no están muy conectados entre sí. En este caso, los fallos son localizados y no se extienden ampliamente por la red. Sin embargo, si pasamos al régimen crítico, donde la conectividad

media es igual a uno, el fallo en cascada se propagará más ampliamente por la red, afectando a diversas particiones o comunidades. Por último, si la conectividad media es mayor que uno (en la región supercrítica), el fallo en cascada se propagará a lo largo de la mayor parte de la red, afectando a prácticamente la totalidad de la misma. En consecuencia, comprender y gestionar la conectividad de la red es fundamental para su robustez y capacidad de resistir ataques.

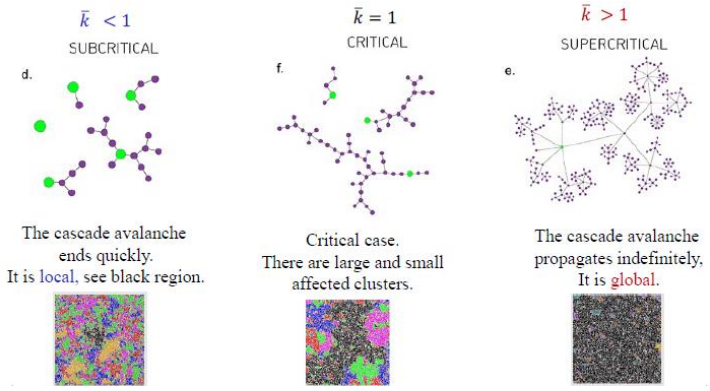


Figura 2-20. Representación propagación de un ataque en la red.

Fuente: [6]

2.7. Plataformas para implentación

En esta sección se presentarán diferentes plataformas que se pueden emplear tanto para el procesamiento de los datos como para la representación y análisis de los grafos resultantes. Estas herramientas proporcionan una amplia gama de funcionalidades para manipular, visualizar y analizar datos basados en grafos.

Una de las plataformas más extendidas es Neo4j. Se trata de una base de datos orientada a grafos, lo que significa que los datos que se almacenan lo hacen en forma de nodos y relaciones en lugar de en tablas o documentos. Neo4j utiliza un lenguaje de consultas propio, diseñado específicamente para trabajar con grafos, que recibe el nombre de Cypher. Esto puede suponer un reto si no se ha trabajado previamente con este tipo de lenguajes. Además de ser una base de datos, esta plataforma también proporciona herramientas para visualizar y analizar grafos.

Existen numerosas librerías y paquetes de Python que ofrecen también herramientas muy útiles para el análisis y la representación de grafos. El paquete más popular es Networkx, que se emplea para la creación, manipulación y estudio de la estructura y funciones de una red compleja. Networkx permite construir grafos aleatorios, convertir grafos a diferentes formatos y encontrar características de los mismos como grados, subgrafos, centralidad, entre otros. Adicionalmente, Python cuenta con otros paquetes como PyGSP o scikit-network, orientados al procesamiento de señal en grafos y al aprendizaje automático, respectivamente.

Matlab también ofrece funcionalidades para trabajar con grafos a través de su función *graph*. Esta función permite crear grafos dirigidos y no dirigidos, así como analizar sus propiedades estructurales y realizar operaciones como búsqueda de caminos, cálculo de centralidad y detección de comunidades. Matlab también proporciona herramientas para visualizar grafos de forma interactiva y personalizada

Por último, se presenta la aplicación Gephi. Esta es una aplicación de software de código abierto especializada en visualización y el análisis de redes. Gephi proporciona una interfaz gráfica intuitiva y potentes herramientas de visualización para explorar y analizar grafos de manera interactiva. Además, permite importar datos y generar grafos a partir de estos.

Estas son solo algunas de las opciones más comunes para trabajar con grafos. Sin embargo, existen muchas otras posibilidades que no se mencionan en este trabajo, ya sea porque no se utilizan en su elaboración o porque no son tan ampliamente distribuidas como las mencionadas anteriormente.

2.8. Resumen

En este capítulo, se ha proporcionado una descripción general de los fundamentos teóricos y prácticos de los grafos, con el objetivo de establecer una base sólida para comprender estos conceptos y los algoritmos utilizados en su análisis.

Comenzamos discutiendo la definición formal de un grafo, destacando sus dos componentes básicos: nodos (vértices) y aristas (enlaces), que representan los elementos y las relaciones entre ellos, respectivamente. Luego, exploramos los diferentes tipos de grafos según varios atributos comunes, como si son dirigidos o no, ponderados o no ponderados, conectados o no conectados, cíclicos o acíclicos, monopartitos o k-partitos, dispersos o densos.

Después, profundizamos en las características y propiedades fundamentales de los grafos, como la densidad, el grado de sus nodos, el orden y el tamaño del grafo, y la matriz de adyacencia. Esta última es especialmente importante ya que proporciona una descripción de las relaciones entre los nodos y es útil para calcular los grados de los nodos y encontrar caminos entre ellos utilizando las potencias de la matriz de adyacencia.

Posteriormente, exploramos la estructura de los grafos, centrándonos en tres tipos principales: los grafos aleatorios, los grafos de mundo pequeño y los grafos de escala libre. Cada uno de estos tipos de grafos exhibe características únicas en términos de conectividad y distribución de nodos, lo que los hace relevantes para una variedad de aplicaciones del mundo real. También discutimos el concepto de la componente gigante en un grafo y su importancia en el estudio y comportamiento de la red.

Además, analizamos la importancia de la robustez de la red, destacando su capacidad para resistir fallos y ataques intencionados sin perder su estructura ni afectar considerablemente a su funcionalidad. Esto nos llevó a considerar cómo evaluar la robustez de una red y las implicaciones prácticas de fortalecerla frente a posibles amenazas.

Finalmente, exploramos algunas de las plataformas y herramientas más populares para trabajar con algoritmos de grafos, como Neo4j, NetworkX, MATLAB y Gephi. Estas herramientas proporcionan a los investigadores y profesionales capacidades avanzadas para analizar, visualizar y manipular grafos en una variedad de dominios y aplicaciones.

3 ALGORITMOS DE ESTUDIO

Los algoritmos de grafos constituyen un conjunto de herramientas esenciales empleadas para el análisis de grafos. En ellos reside la verdadera utilidad y potencial de estas estructuras de datos, permitiendo desarrollar métodos para manipular y analizar grafos de manera efectiva. Estos algoritmos permiten resolver una amplia gama de problemas, desde encontrar la ruta más corta entre dos nodos hasta identificar comunidades dentro de redes complejas. Basados en la teoría matemática de grafos, estos algoritmos utilizan las relaciones entre nodos para revelar la organización y dinámicas de un sistema complejo.

Existen diferentes tipos de algoritmos de grafos, que se clasifican en tres grupos fundamentales de acuerdo con [11]: búsqueda de caminos, centralidad y detección de comunidades.

Los algoritmos de búsqueda de caminos se centran en descubrir el camino más corto entre dos nodos, el mínimo árbol de expansión (también conocido como spanning tree), entre otros. Algunos ejemplos de este tipo son BFS (Breadth First Search), DFS (Depth First Search), Shortest Path (probablemente el más empleado en los grafos) y Minimum Spanning Tree.

Por otro lado, los algoritmos de centralidad buscan identificar los nodos más importantes dentro del grafo. Algunos de los más destacados incluyen Betweenness Centrality y PageRank.

Por último, los algoritmos de detección de comunidades buscan encontrar agrupaciones de nodos en el grafo y, dependiendo de su conectividad, calificar la calidad de estas comunidades. Algunos de los más destacados son el Algoritmos de Louvain y el Spectral Clustering.

En este capítulo, nos centraremos tanto en los algoritmos de centralidad como en los de detección de comunidades, explicando en detalle aquellos cuya implementación será llevada a cabo en este trabajo y explicada en el siguiente capítulo. Estos incluyen el algoritmo de Betweenness Centrality y PageRank para la centralidad, así como el Algoritmos de Louvain y el Spectral Clustering para la detección de comunidades.

3.1 Centralidad

En teoría de grafos, el concepto de centralidad se refiere a la importancia de los nodos o vértices dentro de un grafo [7]. Existen numerosas medidas de centralidad para determinar esta importancia, las cuales no necesariamente implican que un nodo esté en el centro del grafo o tenga características similares, lo que podría llevar a confusiones por el nombre. La centralidad no es un atributo intrínseco de los nodos en una red, sino que se trata de una característica que depende de las relaciones que estos mantienen con el resto de nodos.

Los algoritmos de centralidad se emplean con el objetivo de hallar los diferentes roles que desempeñan los nodos en el grafo. Su objetivo es identificar los nodos más importantes de la red en función de diversas medidas, como, por ejemplo, aquellos que distribuyen información con mayor rapidez o aquellos que actúan como puentes para conectar diferentes comunidades dentro del grafo.

3.1.1 Algoritmo de Betweenness Centrality

En la teoría de grafos, el término centralidad de intermediación (también conocido como betweenness centrality) se refiere a una medida de centralidad que cuantifica la cantidad de veces que un nodo se encuentra en la ruta del camino más corto entre otros dos nodos [12]. Es decir, la centralidad de intermediación de un nodo es igual al número de caminos más cortos que pasan a través de este, entendiéndo el término “caminos más cortos” como todos los caminos más cortos que hay entre cada par de nodos del grafo. De esta manera, un vértice del grafo

tendrá una alta centralidad de intermediación si es un vértice intermedio para muchas rutas entre nodos.

Esta medida es empleada para medir la influencia que un nodo tiene sobre el flujo de información o de recursos en el grafo, detectando cuellos de botella o nodos que actuán como puentes para la interconexión de diferentes comunidades dentro del mismo grafo. La importancia de estos nodos con alta centralidad radica en que, cuando son eliminados, provocan la desconexión de alguna sección del grafo, dividiéndolo en diferentes partes aisladas e incomunicadas entre sí. También pueden retrasar o dificultar el flujo de información o recursos en una red u organización, al ser los nodos más transitados en las comunicaciones. Estos vértices pueden considerarse tanto una debilidad, al representar puntos críticos de fallo en la red, como una solidez, debido a su capacidad para controlar y canalizar el flujo de información.

El algoritmo de Betweenness Centrality es el que nos va a permitir calcular esta medida de centralidad de los nodos. Funciona de la siguiente manera según [6]:

- Primero, se calcula el camino más corto para cada par de nodos en el grafo haciendo uso de otro algoritmo diseñado para la detección de estos caminos, como podría ser el algoritmo de BFS (Breadth First Search) o Dijkstra.
- Posteriormente, cada nodo recibe una puntuación basada en el número de caminos más cortos calculados anteriormente que pasen a través de dicho nodo. Si se desea normalizar esta medida, bastaría con dividir la cantidad anterior entre el número total de caminos más cortos en el grafo.
- Por último, los nodos con mayor puntuación serán aquellos con una centralidad de intermediación mayor y, por tanto, se consideran más importantes para el grafo basándose en esta medida. Estos nodos suelen ser identificados como cuellos de botella o puentes debido a su influencia en la conectividad del grafo.

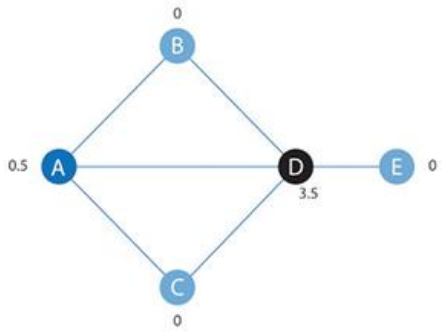
La fórmula para obtener la centralidad de intermediación es la siguiente:

$$B(u) = \sum_{o \neq u \neq d} \frac{n\_shortest\_paths_u(o,d)}{Total\_n\_shorthest\_paths(o,d)} \tag{3-1}$$

Donde u es el nodo para el que se está calculando la medida, $n\_shortest\_paths_u(o,d)$ se refiere al número de camino más cortos desde el nodo origen (o) hasta el nodo destino (d) que pasan por el nodo u y $Total\_n\_shorthest\_paths(o,d)$ se refiere al número total de caminos más cortos entre dos nodos, incluyendo también los que no pasan por el nodo u.

Este algoritmo es costoso computacionalmente, especialmente para grafos grandes, debido a la necesidad de calcular todos los caminos más cortos entre pares de nodos. Sin embargo, es una herramienta invaluable para comprender la estructura y dinámica de los grafos, así como para identificar puntos críticos y mejorar la eficiencia de las redes.

A continuación, se muestra un ejemplo simple extraído de [6] para ver el funcionamiento del algoritmo:



Node D Calculation

Pairs with Shortest Paths Through D	Total Possible Shortest Paths for That Pair	% of Total Through D (1/Total)
A, E	1	1
B, E	1	1
C, E	1	1
B, C	2 (through D & A)	0.5
Betweenness Score		3.5

Figura 3-1. Ejemplo del uso del algoritmo de Betweenness Centrality en el nodo D.

Fuente: [6]

En ejemplo anterior, se siguió el siguiente procedimiento de acuerdo con el algoritmo:

- Primero, se calculan todos los posibles caminos más cortos entre cada par de nodos.
- Segidamente se comprueba, para cada nodo, cuántos posibles caminos cortos pasan a través de él. En este caso, no existe ningún camino más corto que pase a través de los nodos B, C o E, por lo que se les asigna un valor de centralidad de intermediación de 0. Luego, para el resto de nodos restantes (A y D), se calcula el número de caminos más cortos entre cada par de nodos y se determina el porcentaje de estos que pasan a través de los nodos mencionados.
- Por último, la suma de los porcentajes anteriores nos proporciona el valor de centralidad de intermediación para estos nodos.

Aunque la figura representa únicamente el cálculo para el nodo D, se procedería de forma análoga con el nodo A.

Este algoritmo tiene innumerables usos en las redes del mundo real, siendo principalmente empleado para encontrar cuellos de botella o vulnerabilidades en la red [11].

Uno de los usos destacados de este algoritmo es en la optimización de infraestructuras de transporte. En sistemas de transporte como redes ferroviarias, carreteras o rutas de transporte público, el algoritmo de Betweenness Centrality se utiliza para identificar los nodos que actúan como puntos críticos de congestión o como conectores clave entre diferentes áreas. Al analizar la centralidad de intermediación de los nodos en estas redes, los planificadores de transporte pueden identificar dónde se concentra el tráfico y diseñar estrategias para mejorar la eficiencia y reducir los tiempos de viaje, como la adición de nuevas rutas, la redistribución de recursos o la implementación de sistemas de gestión del tráfico más efectivos.

Otro uso importante del algoritmo de Betweenness Centrality es en el análisis de redes sociales. En este contexto, se utiliza para identificar individuos o nodos que actúan como intermediarios clave en la difusión de información dentro de la red. Estos nodos con alta centralidad de intermediación pueden ser personas influyentes o conectores clave que facilitan la propagación de información, opiniones o influencias a través de la red. El análisis de centralidad de intermediación en redes sociales es fundamental para comprender la dinámica de difusión, identificar líderes de opinión, detectar comunidades de interés y diseñar estrategias de marketing viral o campañas de influencia.

3.1.2 Algoritmo de PageRank

El algoritmo de PageRank es un algoritmo de centralidad desarrollado por Larry Page y Serget Brin minetras eran estudiantes en la Universidad de Stanford en 1996, como parte de su proyecto de investigación para un nuevo motor de búsqueda [13]. Junto con otros colaboradores, Page y Brim fueron los autores del primer artículo sobre el proyecto, que describe PageRank y el prototipo incial del motor de búsqueda de Google, publicado en

1998. Poco después, Larry Page y Serget Brin fundaron Google Inc., la empresa detrás del famoso motor de búsqueda cuyo funcionamiento se basaba en el algoritmo de PageRank, aunque actualmente ya no es el único algoritmo empleado para ordenar los resultados de las búsquedas. El nombre "PageRank" es un juego de palabras que combina el apellido del desarrollador, Larry Page, con el concepto de página web. A diferencia de otros algoritmos de centralidad, PageRank no solo mide la influencia directa de un nodo, sino que también considera la influencia de sus vecinos y de los vecinos de estos.

PageRank fue inicialmente diseñado con el objetivo de clasificar las páginas web en los resultados de búsqueda de Google. Sin embargo, en la actualidad, su uso se ha ampliado a diversos campos, desde la clasificación de usuarios en redes sociales hasta la recomendación de contenido relevante. Este algoritmo estima la importancia de una página web en función del número y la calidad de los enlaces que apuntan hacia ella, considerando que una página es más relevante si está vinculada desde otras páginas importantes.

El resultado de aplicar PageRank es una distribución de probabilidad que representa la probabilidad de que un usuario que hace clic aleatoriamente en los enlaces llegue a una página específica. Para calcular esta probabilidad, se realizan múltiples iteraciones a través del grafo de páginas web, lo que permite reflejar de manera más precisa el valor real de cada página.

Ahora surge la pregunta de cómo se genera este grafo de páginas web sobre el cual se aplica el algoritmo de PageRank cuando un usuario realiza una búsqueda en Google. Al llevar a cabo una búsqueda, Google registra las palabras clave introducidas por el usuario. Google dispone de una vasta lista de páginas web, las cuales son tratadas como documentos que contienen una variedad de palabras. Para procesar esta información, Google analiza estos documentos y crea un resumen mediante un histograma que representa la frecuencia de las palabras en cada página.

A continuación, Google emplea una técnica de búsqueda inversa para relacionar las páginas web con las palabras clave que contienen. Esto se logra mediante la creación de listas inversas que indican qué palabras o temáticas aparecen en ciertas páginas web. Cuando se realiza una búsqueda, Google construye un grafo bipartito donde se encuentran los términos introducidos por el usuario y las páginas web asociadas a estos términos según las listas inversas.

Este grafo bipartito proporciona una base para crear un grafo de páginas web interconectadas de forma dirigida. Este proceso implica proyectar las conexiones entre páginas que comparten referencias a los términos de búsqueda. Como resultado, se obtiene un grafo compuesto por millones de páginas relevantes para los términos de búsqueda del usuario.

Sin embargo, la tarea crucial es discernir cuáles de estas páginas son las más importantes y relevantes para mostrar al usuario. Aquí es donde entra en juego el algoritmo de PageRank. Este algoritmo se aplica sobre este grafo gigantesco y ordena las páginas web en función de su importancia relativa, asegurando que solo se muestren las más relevantes y útiles para el usuario en los resultados de búsqueda.

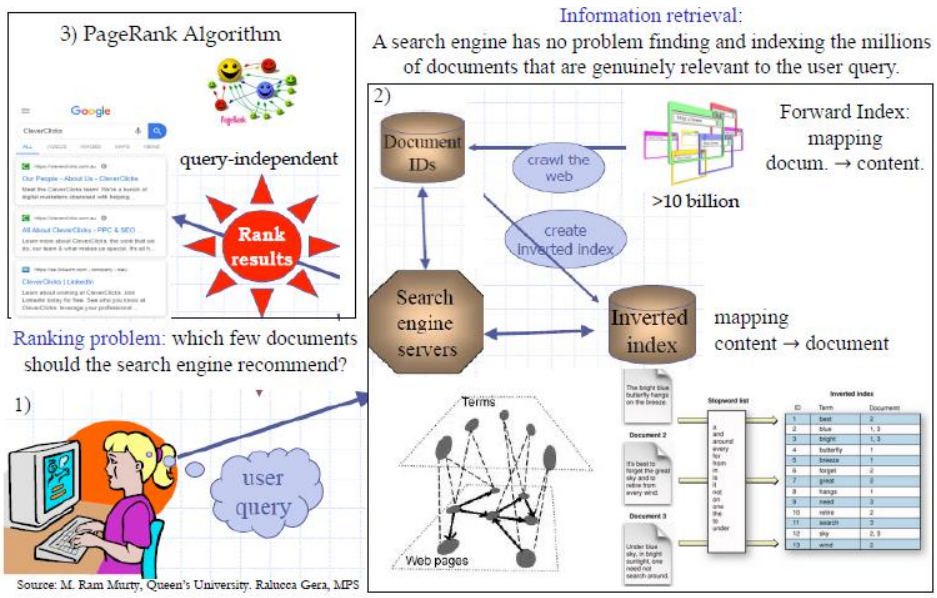


Figura 3-2. Representación funcionamiento Google e importancia de PageRank

Fuente: [6]

A continuación, se detalla el funcionamiento del algoritmo simplificado de PageRank, el cual genera una distribución de probabilidad que indica la probabilidad de que un usuario que hace clic aleatoriamente en enlaces acceda a una página en particular.

Para entender este algoritmo, es fundamental comprender cómo adquieren valor las páginas web, que son los vértices del grafo, y cómo este valor se distribuye entre los nodos vecinos. Cada página (nodo) tendrá asociada una puntación de PageRank, denotada como $P_r(A)$, que representa su importancia relativa. Inicialmente, esta puntación puede ser cualquier valor, pero se ajustará durante las iteraciones del algoritmo. Aunque el valor inicial puede ser cualquiera, comúnmente se elige el mismo valor para todos los nodos, ya que no se sabe inicialmente cuál será más influyente o menos.

Otro concepto crucial es cómo la importancia o valor de un nodo influye en los nodos vecinos con los que está conectado a través de sus enlaces salientes. Esto se ilustra mejor con el siguiente ejemplo:

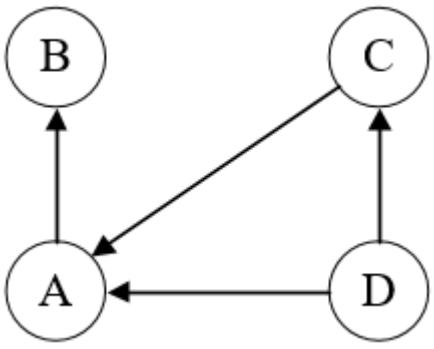


Figura 3-3. Grafo dirigido para explicar PageRank

Figura: https://www.researchgate.net/figure/Ejemplos-de-un-grafo-dirigido-y-un-grafo-no-dirigido_fig7_309278789

En el grafo anterior, se presenta un grafo dirigido simple para explicar los conceptos mencionados anteriormente. En este caso, cada nodo A, B, C y D tiene un valor inicial asociado que, como veremos más adelante en la explicación del algoritmo simplificado de PageRank, generalmente está vinculado con la puntuación de

PageRank ($P_r()$). Ahora bien, un nodo transfiere su importancia a través de los flujos salientes que tiene. Es decir, el valor de A afecta únicamente a B mientras que el valor de D afecta tanto a A como a C.

Ahora se detalla el funcionamiento del algoritmo simplificado de PageRank [14], utilizando como referencia el grafo presentado en la Figura 3-4, compuesto por cuatro nodos: A, B, C y D. PageRank se inicia asignando el mismo valor a todos los nodos. En la versión inicial de PageRank, la suma de las puntuaciones de todos los nodos equivalía al número total de nodos en el grafo, resultando en un valor inicial de 1 para cada nodo. Sin embargo, en las versiones más recientes de PageRank, se utiliza una distribución de probabilidad que oscila entre cero y uno, así, en este caso, cada nodo recibiría un valor inicial de 0.25.

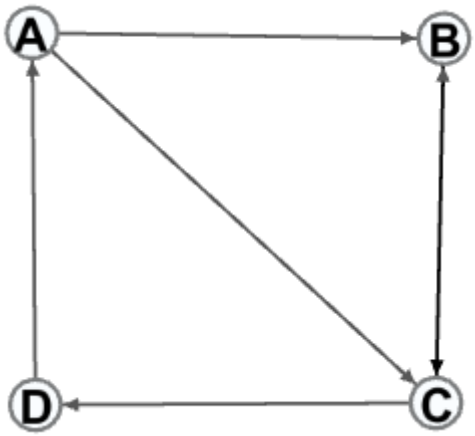


Figura 3-4. Grafo para explicar el algoritmo simplificado de PageRank

Fuente: Elaboración propia.

Una vez establecido este valor inicial, el PageRank transferido desde un nodo dado a los destinos de sus enlaces salientes en la siguiente iteración se distribuye equitativamente entre todos los enlaces salientes. En términos generales, la puntuación de PageRank para cualquier nodo v_i se puede expresar como:

$$Pr(v_i) = \sum_{v_j \in vecinos\ de\ v_i} \frac{Pr(v_j)}{k^{out}_{v_j}} \tag{3-2}$$

Esta fórmula indica que la puntuación de PageRank de un nodo es la suma de las puntuaciones de PageRank de sus nodos vecinos, dividida por el número de enlaces salientes que tiene cada vecino. Es decir, un nodo distribuye su importancia de manera equitativa entre los diferentes enlaces de salida.

En el ejemplo presentado en la Figura 3-4, los cálculos de PageRank resultarían en los siguientes valores:

$$\Pr(A) = \frac{\Pr(D)}{1} = 0.25$$

$$\Pr(B) = \frac{\Pr(A)}{2} + \frac{\Pr(C)}{2} = \frac{0.25}{2} + \frac{0.25}{2} = 0.25$$

$$\Pr(C) = \frac{\Pr(A)}{2} + \Pr(B) = \frac{0.25}{2} + 0.25 = 0.375$$

$$\Pr(D) = \frac{\Pr(C)}{2} = \frac{0.25}{2} = 0.125$$

Estos valores representan la primera iteración del algoritmo, donde los nuevos valores de PageRank para los diferentes nodos se calculan aplicando la fórmula mencionada anteriormente. Este proceso se repetiría, distribuyendo nuevamente la importancia de los nodos entre sus vecinos a través de los enlaces salientes, utilizando los nuevos valores de PageRank obtenidos en la primera iteración, como se muestra en la Figura 3-5. Este ciclo continúa hasta alcanzar un punto de convergencia o hasta que se complete el número de iteraciones predefinido. En las siguientes iteraciones, los valores de PageRank se ajustarán aún más, refinando la estimación de la importancia de cada página dentro del grafo.

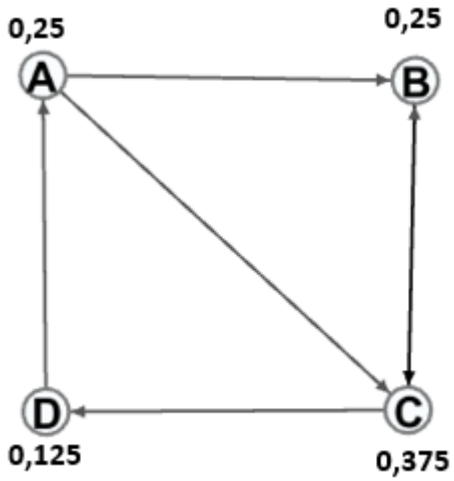


Figura 3-5. Puntuaciones para la segunda iteración del algoritmo simplificado de PageRank.

Fuente: Elaboración propia.

Sin embargo, este algoritmo de PageRank presenta algunos inconvenientes que pueden interferir en su objetivo final [6].

El primero de estos problemas es el denominado "spider trap" o tela de araña. Esto ocurre cuando un nodo de la red tiene enlaces únicamente consigo mismo, lo que resulta en que la influencia quede atrapada en ese nodo y no pueda distribuirse hacia otros nodos vecinos. En el contexto de la ordenación de páginas web, este problema se presenta cuando una web solo contiene enlaces a sí misma y no a otras páginas externas. Como consecuencia, dicho nodo adquiere una relevancia desproporcionada, ya que el PageRank se queda atrapado y no puede fluir hacia otras páginas.

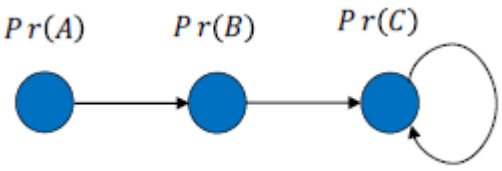


Figura 3-6. Ejemplo de spider trap con PageRank.

Fuente: [6]

En el ejemplo representado en la figura anterior, se puede ver claramente cómo el nodo C absorberá toda la influencia. El nodo A, en este caso, pasaría a tener una puntuación de 0, puesto que no tiene ningún enlace entrante. El nodo B adquerirá la influencia del nodo A y, finalmente, el nodo C obtendrá la suma de la importancia de B y la suya propia. Por lo tanto, el algoritmo convergirá en el nodo C, que terminará con un PageRank igual a 1, absorbiendo de esta forma toda la influencia.

Además de las “spider traps”, otro problema similar del algoritmo simplificado de PageRank son los puntos muertos, conocidos también como “dead ends”. Este término se aplica a aquellos nodos del grafo que reciben la influencia de otros nodos vecinos, pero que no poseen ningún enlace saliente para distribuir su propia influencia. Esto se traduce a que una página web no tenga ningún hipervínculo con otra web. En estos casos, el flujo de influencia se acabaría perdiendo y alcanzaría el valor nulo para todos los nodos, lo que incumpliría la normalización que establece que la suma de todos los valores de PageRank debe ser igual a 1.

Además, surgen problemas cuando hay un grupo de nodos que no están conectados al resto de la red, sino que solo se enlazan entre ellos de forma circular, distribuyendo su influencia únicamente dentro de este grupo. En la figura siguiente, se ilustra un ejemplo de ambos casos: el nodo A actúa como un punto muerto, mientras que el grupo de nodos C, D y E representa una referencia circular que retiene la influencia sin permitir su escape.

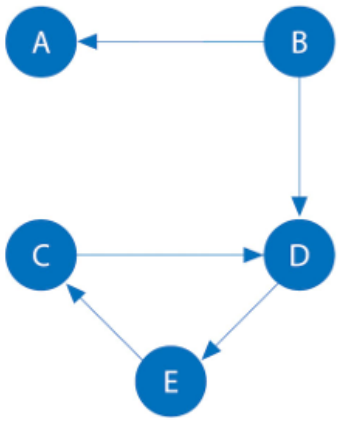


Figura 3-7. Ejemplo de dead end y referencias circulares.

Fuente: [7]

Para mitigar estos problemas, una de las soluciones implementadas es la "teletransportación". Cuando PageRank alcanza un nodo sin enlaces salientes, se asume que este nodo tiene conexiones salientes con todos los demás nodos del grafo. Este concepto considera que, al navegar por diferentes páginas web, el usuario puede llegar a un punto donde decide reiniciar su búsqueda en una página aleatoria. Así, si se queda atrapado en un spider trap o en un dead end, la teletransportación permite saltar a otros nodos, evitando que la influencia quede atrapada en un solo nodo o que se pierda debido a la falta de enlaces salientes.

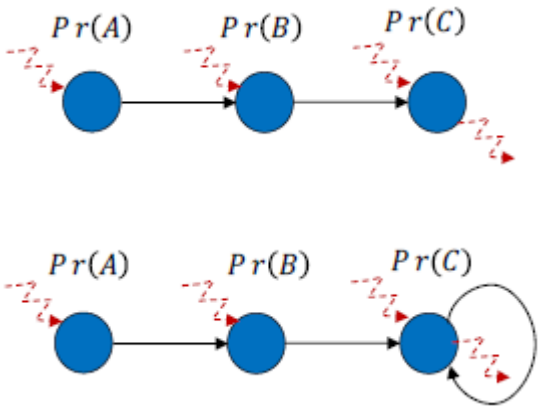


Figura 3-8. Representación de la teleportación.

Fuente: [6]

Para tener en cuenta esta posibilidad, se introduce lo que denominamos factor de atenuación (damping factor), que representa la probabilidad de que un usuario continúe haciendo clic a través de los diferentes vínculos de las páginas web. Este factor se denota por la letra d [6]. De esta manera, la probabilidad de que la navegación comience nuevamente en una página aleatoria, entre las N páginas disponibles, es de $1 - d$.

Varios estudios han probado diferentes factores de atenuación, pero generalmente se supone que el factor de amortiguación se establecerá alrededor de 0,85.

Por lo tanto, la fórmula para calcular la puntuación de PageRank del nodo v_i se puede expresar como:

$$Pr(v_i) = (1-d)\frac{1}{N} + d \sum_{v_j \in vecinos\ de\ v_i} \frac{Pr(v_j)}{k_{v_j}^{out}} \tag{3-3}$$

En esta ecuación, la probabilidad de alcanzar el nodo v_i, que equivale a su puntuación de PageRank, resulta de sumar la probabilidad de alcanzar dicho nodo mediante teletransportación y la probabilidad de alcanzarlo mediante la navegación a través de los enlaces. Así, al sumar todas las probabilidades de acceder a todos los nodos, el resultado será igual a uno.

En conclusión, el algoritmo de PageRank, inicialmente desarrollado para clasificar páginas web en los resultados de búsqueda de Google, ha demostrado ser una herramienta poderosa y versátil que se utiliza en una variedad de campos más allá del indexado web. Su capacidad para evaluar la importancia relativa de nodos en un grafo lo convierte en una herramienta invaluable cuando se busca influencia en una red.

Las utilidades del algoritmo de PageRank son diversas y abarcan desde la recomendación de cuentas en redes sociales hasta la predicción del flujo de tráfico humano en espacios públicos [11]. Por ejemplo, en Twitter se emplea el Personalized PageRank para sugerir a los usuarios otras cuentas que podrían interesarles. Del mismo modo, en el ámbito de la salud y los seguros, PageRank se utiliza en sistemas de detección de anomalías y fraudes para identificar a médicos o proveedores que puedan estar comportándose de manera inusual.

Además, este también se ha utilizado para predecir flujos de tráfico en calles y espacios públicos, así como para identificar genes con mayor impacto en funciones biológicas. Estos ejemplos muestran cómo el algoritmo de

PageRank puede aplicarse en una variedad de contextos para proporcionar información valiosa y tomar decisiones fundamentadas.

3.2 Detección de Comunidades

La detección de comunidades en grafos tiene una amplia variedad de aplicaciones en diferentes campos. En redes sociales, se utiliza para identificar grupos de usuarios con intereses comunes. En biología y genética, ayuda a descubrir grupos funcionales de genes o proteínas. En el ámbito académico, facilita la identificación de colaboraciones y tendencias de investigación. En transporte y logística, se emplea para optimizar rutas y gestionar congestiones. Por último, en redes de internet y telecomunicaciones, permite segmentar usuarios y personalizar servicios.

La detección de comunidades, también conocida como partición de grafos, hace referencia al conjunto de técnicas no supervisadas utilizadas para encontrar grupos de nodos altamente interconectados que reciben el nombre de comunidades [15]. Una comunidad se define como una agrupación de vértices altamente conectados entre sí, pero que tienen pocas conexiones hacia nodos fuera del grupo. Este concepto de comunidad se fundamenta en dos hipótesis fundamentales:

- **Hipótesis de conectividad:** Cada comunidad corresponde a un subgrafo conectado en la red. Esto implica que, si una red tiene componentes aisladas, cada comunidad estará restringida a una de esas componentes, y dentro de una misma componente, no puede haber comunidades formadas por subgrafos no interconectados.
- **Hipótesis de densidad:** Los nodos dentro de una comunidad tienden a estar más conectados entre sí que con nodos fuera de la comunidad, lo que resulta en una mayor densidad de conexiones internas en comparación con las conexiones intercomunitarias.

En resumen, para que exista una comunidad, debe haber cohesión significativa entre sus miembros y una baja cohesión con el resto del grafo.

La tarea de detectar comunidades es intrínsecamente compleja, pero puede simplificarse cuando los nodos están bien organizados. A través de la matriz de adyacencia de un grafo, es posible identificar comunidades y sus interconexiones. En la siguiente figura se presenta un ejemplo:

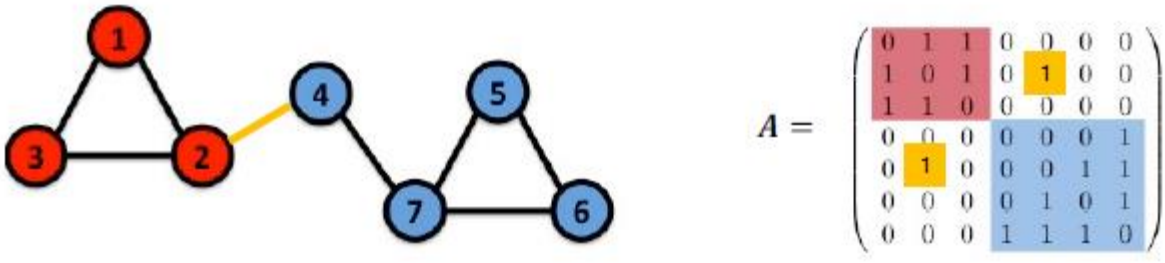


Figura 3-9. Ejemplo simple de detección de comunidades.

Fuente: [15]

En el ejemplo proporcionado en la Figura 3.9, se pueden distinguir dos comunidades altamente conectadas, representadas en rojo y azul, con un enlace de unión entre ellas. Sin embargo, en la práctica, los nodos suelen estar desordenados, lo que complica la identificación de comunidades. Para abordar esta dificultad, es necesario transformar la matriz de adyacencia mediante una matriz de permutación. En la Figura 3.10, se puede observar cómo se representa la matriz de adyacencia con nodos desordenados y cómo queda tras aplicar las permutaciones

correspondientes. Esta reorganización facilita la detección de comunidades, mostrando claramente las diferentes comunidades en la diagonal de la matriz y algunas conexiones entre ellas en el resto de la matriz.

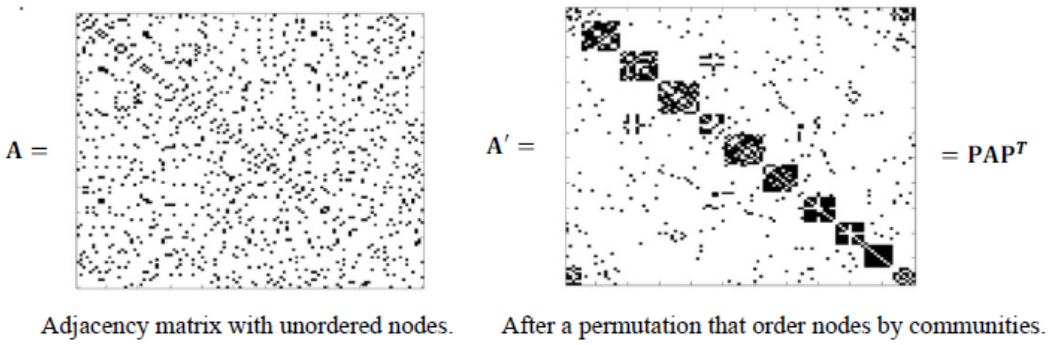


Figura 3-10. Permutación matriz de adyacencia para detectar comunidades.

Fuente: [15]

Además, podemos distinguir entre comunidades débiles y fuertes. Un subgrafo conectado o comunidad C de N_c nodos se considera una comunidad fuerte si cada nodo tiene más enlaces dentro de la propia comunidad que con nodos que no pertenezcan a ella. Por otro lado, una comunidad se considera débil si la suma total de los grados internos de sus nodos supera la suma total de los grados externos, aunque esto no se cumpla para cada nodo de manera individual, como sucede en las comunidades fuertes.

En los últimos años, se han propuesto numerosos algoritmos para la detección de comunidades, utilizando técnicas y herramientas de diversas disciplinas como biología, física, ciencias sociales, matemáticas aplicadas y ciencias de la computación. En esta sección, nos centraremos en dos de estos algoritmos: el de Louvain y el de Spectral Clustering.

3.2.1 Algoritmo de Louvain

El algoritmo de Louvain fue propuesto por primera vez en 2008 por Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte y Etienne Lefebvre en un artículo titulado "Fast unfolding of communities in large networks" [16] durante su trabajo en la Universidad Católica de Lovaina (UCL) en Bélgica.

El desarrollo del algoritmo de Louvain surgió de la necesidad de abordar eficientemente el problema de detección de comunidades en grafos grandes y complejos. Antes de la propuesta de Louvain, los métodos existentes para detectar comunidades en estas redes eran computacionalmente costosos y no escalaban bien a medida que aumentaba el tamaño del grafo.

El algoritmo de Louvain se basa en la optimización de la modularidad, como se describe en [17]. La modularidad mide la calidad de asignación de los nodos a las comunidades al evaluar la densidad de conexiones internas en comparación con las conexiones externas. La idea principal detrás del algoritmo es maximizar la modularidad del grafo al encontrar la partición de nodos que maximiza la calidad de las comunidades detectadas.

Para comprender correctamente el algoritmo, es fundamenteal profundizar en la definición de modularidad. Esta se define como la diferencia entre el número de conexiones observadas dentro de las comunidades y el número esperado de conexiones al azar. Es una medida que evalúa la calidad de una partición de nodos en comunidades dentro de una red, siendo crucial para evaluar la efectividad del algoritmo. La fórmula utilizada para calcular la modularidad en grafos no dirigidos es la siguiente:

$$M = \frac{1}{2L}\sum_{u}\sum_{v}\left(A_{uv} - \frac{k_u k_v}{2L}\right)\delta(C_u, C_v) \qquad \epsilon\,[-0.5, 1] \tag{3-4}$$

Donde:

- A_{uv} es la matriz de adyacencia del grafo.
- k_u y k_v representan el grado de los nodos u y v respectivamente, es decir, el número de conexiones que poseen. En el caso de un grafo ponderado, este valor corresponde a la suma de los pesos de los ejes que conectan a los nodos u y v.
- L representa el número de ejes del grafo. Si el grafo es ponderado, este valor se corresponde a la suma de todos los pesos de la matriz de adyacencia
- δ es la función delta de Kronecker, que tienen un valor igual a uno si las comunidades C_u y C_v son la misma comunidad, y cero en caso contrario.

La función delta en la ecuación solo toma valores distintos de cero cuando los nodos pertenecen a la misma comunidad, permitiendo así evaluar la modularidad dentro de cada grupo. Maximizar la modularidad implica optimizar la ecuación 3-4, buscando que el primer término sea lo más alto posible y el segundo, lo más bajo. La primera parte de la ecuación representa el número de enlaces internos en una comunidad, mientras que la segunda se refiere al número de enlaces esperados al azar entre dos nodos. Cuando el número de enlaces dentro de la comunidad supera el esperado aleatoriamente, la modularidad se vuelve positiva. Cuanto mayor sea la diferencia entre los enlaces reales y los esperados, mayor será el valor de la modularidad.

El valor de la modularidad se encuentra en el rango de -0.5 a 1. A partir de un valor aproximado de 0.3, se considera que la modularidad es alta, lo que indica que las comunidades encontradas tienen más conexiones de las esperadas al azar. Esto sugiere una estructura comunitaria significativa en la red, ya que no es posible que una red aleatoria se divida naturalmente en comunidades.

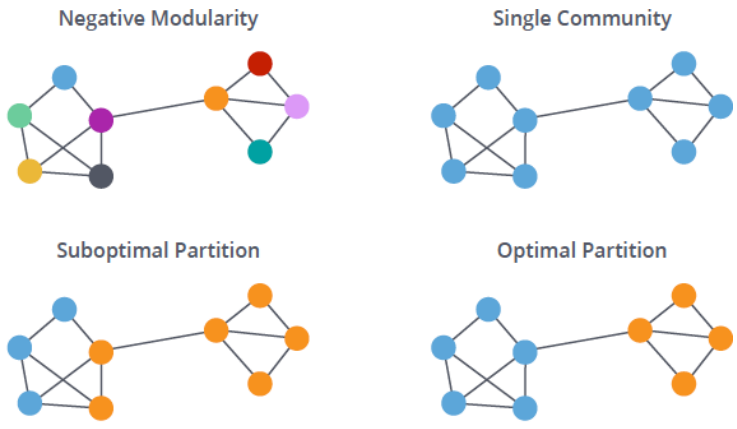


Figura 3-11. Ejemplo de diferentes valores de modularidad en un grafo simple.

Fuente: [17]

En la figura anterior se muestran diferentes valores para la modularidad. La modularidad máxima corresponde

a la partición óptima, mientras que una partición subóptima tiene una modularidad positiva pero no identifica correctamente las comunidades. Cuando hay una única comunidad, el valor de la modularidad es cero. Por último, para obtener una modularidad negativa, se asigna a cada nodo una comunidad propia.

Una vez aclarado el concepto de modularidad, es momento de detallar el algoritmo de Louvain [15]. Este algoritmo se ha consolidado como uno de los más empleados para la detección de comunidades debido a su velocidad y eficacia en redes de gran tamaño. Además de identificar comunidades, el algoritmo revela la jerarquía dentro de estas a diferentes escalas, lo que resulta fundamental para comprender y analizar la estructura de la red. El funcionamiento del algoritmo es el siguiente:

- **Paso 1:** Inicialmente, cada nodo del grafo constituye su propia comunidad, lo que resulta en tantas comunidades iniciales como nodos tenga el grafo.

- **Paso 2:** Se escoge un nodo arbitrario del grafo y se calcula como cambiaría la modularidad si este nodo se uniese en una comunidad con uno de sus nodos vecinos directamente conectados. Para esto, se utiliza la fórmula de modularidad presentada anteriormente, aplicada únicamente a ese par de nodos, sin necesidad de extenderlo a todo el grafo. Este proceso se repite para todos los vecinos del nodo seleccionado, y los nodos se agrupan en la misma comunidad si el aumento de la modularidad es máximo en comparación con otras uniones posibles. Este proceso iterativo se repite para cada nodo del grafo varias veces hasta que la modularidad no aumente más.

- **Paso 3:** Conocido como la fase de condensación o “community aggregation”, este paso implica la creación de un nuevo grafo a partir de las comunidades identificadas en el paso anterior. En este nuevo grafo, las comunidades anteriores se condensan en un único nodo o supernodo, lo que resulta en un nodo en el nuevo grafo por cada comunidad obtenida en el Paso 2. Los nuevos enlaces entre nodos tienen un peso igual al número de enlaces entre esas comunidades en el paso anterior, si el grafo es no ponderado. En caso de ser ponderado, el peso del enlace corresponde a la suma de los pesos de los enlaces entre las comunidades anteriores. Para representar los enlaces dentro de una comunidad, se utiliza un "self-loop" o enlace a sí mismo, cuyo peso se corresponde con el número de medias aristas internas en la comunidad, es decir, el doble del número de enlaces.

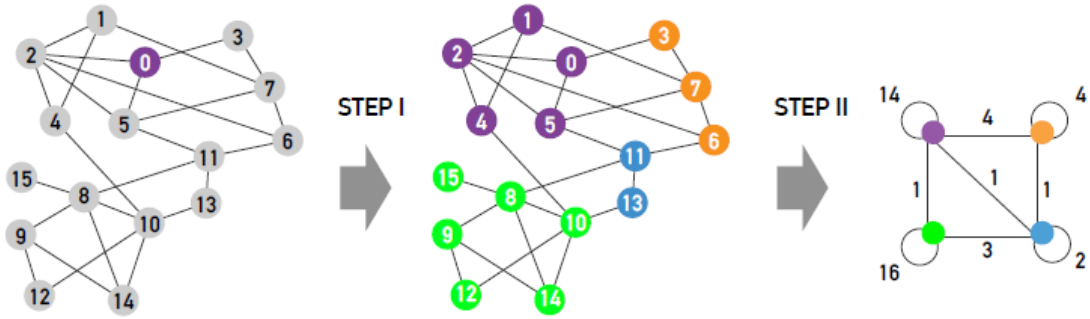


Figura 3-12. Primera iteración del algoritmo de Louvain.

Figura: https://wikidocs.net/93414

En esta figura, se muestra una primera iteración del algoritmo, siguiendo los pasos descritos anteriormente. Inicialmente, cada nodo se considera una comunidad diferente, lo que resulta posteriormente en cuatro comunidades distintas identificadas por diferentes colores. Posteriormente,

utilizando estas comunidades, se construye un nuevo grafo sobre el cual se vuelve a aplicar el algoritmo.

- Por último, se repiten los pasos dos y tres hasta que ya no se produzca un aumento significativo de la modularidad. Es importante mencionar que al construir un nuevo grafo en el Paso 3 en iteraciones posteriores, solo se duplicará el número de enlaces dentro de una comunidad si este proceso no se ha realizado previamente en esos enlaces. Es decir, los "self-loops" que existen dentro de una comunidad ya están contabilizados por los enlaces internos previamente sumados, mientras que los enlaces que conectan diferentes nodos entre sí deberán duplicarse. Esto se puede observar en la siguiente figura, donde se representa un ejemplo para aclarar este concepto.

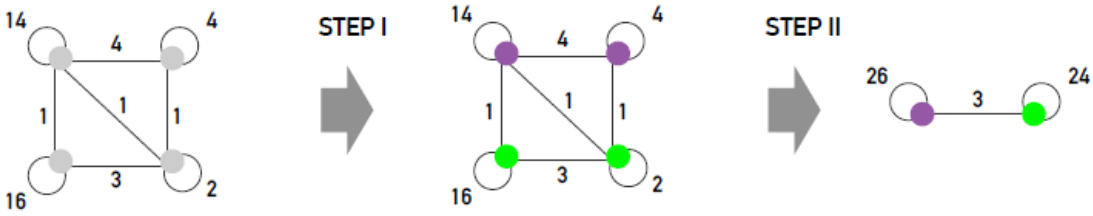


Figura 3-13. Segunda iteración del algoritmo de Louvain.

Fuente: https://wikidocs.net/93414

Se puede apreciar en la Figura 3-13 cómo, en la construcción del nuevo grafo en el paso 3, solo se utilizan las medias aristas nuevamente para las conexiones entre nodos. Si observamos, por ejemplo, la comunidad representada en verde, podemos ver que el nuevo grafo tiene un "self-loop" de peso 24. Este valor resulta de sumar los pesos de los "self-loops" de los nodos pertenecientes a dicha comunidad en el grafo anterior (16 y 2), y luego duplicar el peso del enlace que une ambos nodos. Por lo tanto, la suma sería 16 + 2 + 3 * 2 = 24.

El algoritmo de Louvain, aunque ampliamente utilizado en la detección de comunidades en redes, presenta ciertos problemas significativos que pueden afectar su rendimiento en contextos específicos. Uno de los problemas más significativos es el límite de resolución, donde el algoritmo tiende a fusionar comunidades intuitivas en redes grandes durante su segunda pasada, dificultando así la detección de comunidades más pequeñas. Además, el problema de degeneración es otro desafío relevante, ya que la gran cantidad de asignaciones de comunidad con modularidades similares puede dificultar la identificación del máximo global, lo que podría llevar a la selección de soluciones subóptimas. Ajustar el tamaño de las comunidades mediante el parámetro de resolución del algoritmo puede ayudar a mitigar estos problemas, permitiendo una mayor sensibilidad en la detección de comunidades pequeñas o grandes. Sin embargo, determinar el valor óptimo de resolución es en sí mismo un desafío, ya que requiere experimentación y ajustes específicos según las características de la red y los objetivos de la detección.

Este algoritmo se presenta como una herramienta versátil que se ha aplicado en una variedad de contextos para analizar y entender la estructura de redes complejas. Por ejemplo, ha sido utilizado para recomendar subreddits similares a los usuarios de Reddit, basándose en el comportamiento general de los usuarios. También se ha empleado para extraer temas de plataformas sociales en línea, como Twitter y YouTube, mediante el modelado de temas basado en el grafo de co-ocurrencia de términos en documentos. Además, el método de Louvain ha sido fundamental en la investigación de la estructura de comunidades dentro de la red funcional del cerebro humano, revelando estructuras jerárquicas. Estos ejemplos ilustran cómo el algoritmo de Louvain puede ser útil para evaluar la estructura de redes complejas, permitiendo identificar niveles de jerarquía y encontrar subcomunidades dentro de comunidades más grandes.

3.2.2 Algoritmo de Spectral Clustering

El algoritmo de Spectral Clustering es una técnica popular de aprendizaje automático que se utiliza en inteligencia artificial para diversas tareas, como la segmentación de imágenes, la detección de comunidades en redes sociales o la segmentación de clientes en el comercio electrónico. Este algoritmo se aplica ampliamente en el aprendizaje automático y la minería de datos. Se basa en la teoría de grafos y técnicas de álgebra lineal, especialmente en el uso de valores y vectores propios del grafo, para dividir conjuntos de datos en diferentes grupos, demostrando ser especialmente eficaz para identificar estructuras en redes complejas.

El desarrollo del Spectral Clustering tiene sus raíces en la teoría espectral de grafos, remontándose a la década de 1970. Sin embargo, su popularización y aplicación práctica en el campo de la minería de datos y el aprendizaje automático se dieron en las décadas de 1990 y 2000. Durante estos años, varios investigadores contribuyeron al refinamiento y aplicación del algoritmo en diferentes dominios. Una de las contribuciones más significativas fue la de Shi y Malik en su artículo seminal de 2000 titulado "Normalized Cuts and Image Segmentation" [18]. En este trabajo, los autores introdujeron el concepto de "cortes normalizados" (normalized cuts) como una forma de segmentación de imágenes basada en técnicas espectrales, demostrando así su eficiencia para problemas de segmentación.

El Spectral Clustering funciona transformando el problema de clustering en un problema de corte de grafo. Primero, construye un grafo donde los nodos representan los puntos de datos y las aristas representan las similitudes entre estos puntos. La implementación de Spectral Clustering se basa en la idea de utilizar los autovectores (eigenvectors) de una matriz derivada del grafo para determinar una representación de los nodos en un espacio de menor dimensión, donde los métodos tradicionales de clustering, como k-means, se pueden aplicar de manera efectiva. La matriz escogida para esta tarea suele ser la matriz laplaciana del grafo, definida como $L = D - A$, donde D es la matriz de diagonal de grados y A es la matriz de adyacencia del grafo.

Mediante la descomposición espectral del grafo, es decir, el cálculo de los valores y vectores propios de su matriz laplaciana, el algoritmo identifica las particiones del grafo que minimizan el coste de corte, facilitando así la identificación de grupos naturales en los datos.

La matriz diagonal de grados D es una matriz diagonal $n \times n$ donde cada elemento diagonal D_{ii} representa el grado del nodo i, es decir, el número de enlaces que tiene con otros nodos Este grado se calcula como la suma de los elementos de la fila correspondiente en la matriz de adyacencia. Con esta matriz y la matriz de adyacencia, que representa las conexiones de los nodos en la red, es posible calcular la matriz laplaciana como la diferencia entre estas dos matrices: $L = D - A$. La matriz laplaciana L tiene varias propiedades importantes que la hacen útil para el análisis de grafos. La propiedad clave que motiva su uso en Spectral Clustering es que sus autovectores proporcionan información sobre las particiones naturales del grafo. A continuación, se puede ver un ejemplo de estas matrices:

$$\mathbf{A} = \begin{array}{c} \begin{matrix} A & B & C & D & E \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \\ E \end{matrix} \qquad \mathbf{D} = \begin{array}{c} \begin{matrix} A & B & C & D & E \end{matrix} \\ \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \\ E \end{matrix} \qquad \mathbf{L = D - A} = \begin{pmatrix} 2 & 0 & -1 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

Figura 3-14. Ejemplo de cáculo de la matriz Laplaciana.

Fuente: [15]

El último de los aspectos a aclarar antes de pasar al funcionamiento del algoritmo de Spectral Clustering, es el

método de k-means que se emplea para agrupar los nodos por comunidades cuando nos encontramos ya en un espacio más reducido. El algoritmo k-means busca dividir n observaciones en k clusters, donde cada observación pertenece al cluster con la media más cercana (centroide). La distancia típica usada para medir la cercanía es la distancia euclidiana. Es decir, se asigna un centroide por cada comunidad que se desee identificar y se asocian los nodos a la comunidad cuyo centroide este más cerca.

El algoritmo k-means se puede resumir en los siguientes pasos:

- **Inicialización:** En el algoritmo k-means, el proceso de inicialización implica seleccionar k puntos iniciales (centroides) de manera aleatoria desde el conjunto de datos o utilizando métodos más avanzados como k-means++. La inicialización juega un papel crucial ya que el resultado final del algoritmo está estrechamente relacionado con los centroides asignados inicialmente. Por lo tanto, es necesario probar diversos escenarios iniciales para obtener un resultado preciso y evitar malos resultados debido a una inicialización deficiente.

  En k-means++, los centroides se eligen con una probabilidad ponderada según la distancia cuadrada de los puntos al centroide más cercano ya seleccionado. Este método asegura una distribución inicial más representativa de los datos, mejorando así la convergencia del algoritmo y la calidad de los clusters obtenidos.
- **Asignación:** Cada observación se asigna al cluster cuyo centroide esté más cercano. Esta asignación se realiza calculando la distancia euclidiana de cada punto a cada uno de los centroides y asignando el punto al centroide más cercano en términos de esta distancia. Este proceso asegura que cada punto sea agrupado con el centroide que minimiza la distancia euclidiana
- **Actualización de centroides:** Después de asignar todos los puntos a los clusters en el algoritmo k-means, se procede a actualizar los centroides. Este proceso consiste en recalcular cada centroide como la media aritmética de todos los puntos asignados a ese cluster. Es decir, para cada cluster, se calcula un nuevo centroide que representa el punto medio de todos los puntos que pertenecen a ese cluster. Esta actualización garantiza que los centroides se ajusten de manera iterativa hacia el centro de gravedad de sus respectivos clusters, lo que mejora la precisión y coherencia de los clusters en cada iteración del algoritmo
- **Iteración:** se repiten los pasos de asignación de puntos a clusters (paso 2) y actualización de centroides (paso 3) hasta que los centroides no cambien significativamente entre iteraciones consecutivas, lo cual indica que el algoritmo ha convergido. La convergencia se determina típicamente mediante una condición de parada, como alcanzar un número máximo de iteraciones o cuando el cambio en los centroides entre dos iteraciones consecutivas es menor que un umbral predefinido.

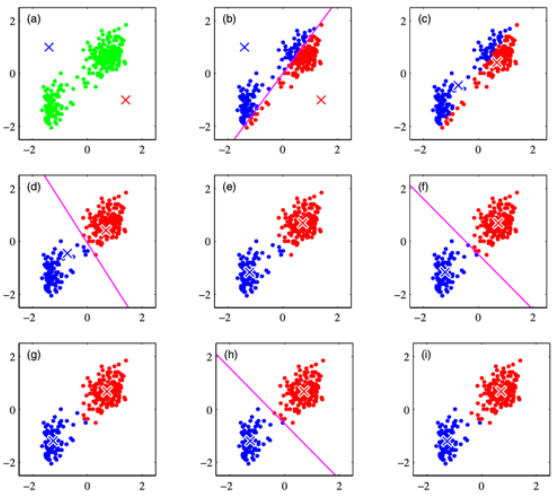


Figura 3-15. Ejemplo de uso de k-means.

Fuente: [19]

Una vez definidos estos conceptos, se desarrolla el funcionamiento del algoritmo de Spectral Clustering, uno de los algoritmos más empleados para la detección de comunidades. Este algoritmo se destaca por su capacidad para detectar estructuras de comunidades en grafos complejos con distribuciones irregulares de enlaces. Los pasos que sigue son los siguientes para dividir la red en k comunidades de acuerdo con [15]:

- **Paso 1**: El primer paso crucial es elegir una representación adecuada del grafo. La opción más común es utilizar el 'radio cut', que emplea la matriz Laplaciana estándar $L = D - A$, donde D es la matriz de diagonal de grados y A es la matriz de adyacencia del grafo. Otra opción ampliamente utilizada es el 'normalized cut' (Ncut), que utiliza la matriz Laplaciana normalizada. La matriz Laplaciana normalizada se obtiene multiplicando la matriz Laplaciana estándar por la matriz diagonal inversa de grados D^{-1}. Esto se representa como $L_{RW} = D^{-1} \times L$.

  La elección entre 'ratio cut' y 'normalized cut' depende del contexto y de la naturaleza de los datos del grafo. Esta fase de preparación del grafo es fundamental, ya que establece la base para la posterior aplicación de técnicas de descomposición espectral y clustering para identificar las comunidades en el grafo de manera efectiva.

- **Paso 2:** se realiza la descomposición de la matriz Laplaciana seleccionada en sus componentes espectrales. Esta descomposición proporciona los autovectores $(v_1, v_2, \dots, v_k)$ y los autovalores correspondientes $(\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_k)$. La matriz laplaciana L, ya sea estándar o normalizada según la elección del 'ratio cut' o 'normalized cut', se descompone para obtener estos autovectores y autovalores. Los autovectores v_i representan las direcciones en el espacio donde el grafo exhibe variaciones significativas, y los autovalores λ_i indican la importancia de estos modos de variación.

  Una vez obtenidos los autovectores y autovalores, se procede a incrustar el grafo en el espacio definido por los autovectores correspondientes a los k autovalores más pequeños. Estos autovectores representan las características principales del grafo en términos de estructura y conectividad, y se utilizan para representar cada nodo en un espacio de dimensión reducida que facilita la separación y agrupación de los nodos en comunidades distintas.

- **Paso 3:** Se procede a formar la matriz de incrustación utilizando los k autovectores más pequeños obtenidos en el paso anterior. Si $v_1, v_2, \dots, v_k$ representan estos autovectores, entonces la matriz de incrustación U se construye concatenando estos autovectores como columnas: $U = [v_1 \quad v_2 \quad \dots \quad v_k]$.

  La matriz U resultante tiene dimensiones $n \times k$ donde n es el número de nodos del grafo y k es el número de comunidades que se desea identificar. Cada fila de U representa la representación reducida de un nodo del grafo en términos de los autovectores seleccionados. El espacio definido por esta matriz permite que las técnicas de clustering como K-means pueden ser aplicadas de manera efectiva para agrupar los nodos en las comunidades deseadas.

- **Paso 4:** Se procede entonces a aplicar el algoritmo de K-means a estas representaciones de los nodos en k-dimensiones. El objetivo es asignar los nodos a k clusters de manera que los nodos dentro de un mismo cluster sean más similares entre sí en términos de sus características estructurales representadas por los autovectores.

  El algoritmo de K-means agrupa iterativamente los nodos en k clusters minimizando la suma de las distancias al cuadrado entre cada nodo y el centroide de su cluster asignado. Este proceso de clustering se repite hasta que la asignación de nodos a clusters converge y los centroides de los clusters no cambian significativamente entre iteraciones consecutivas dando lugar a la detección efectiva de comunidades.

En la siguiente figura se muestra un ejemplo de la aplicación del algoritmo para la segmentación de imágenes. Para generar este grafo se creó una imagen con cuatro círculos y se añadió ruido para simular variaciones en la intensidad de los píxeles y se aplicó una máscara para enfocarse solo en los círculos. Luego, se convirtió la

imagen en un grafo donde cada píxel es un nodo y las conexiones reflejan las diferencias de intensidad entre píxeles vecinos.

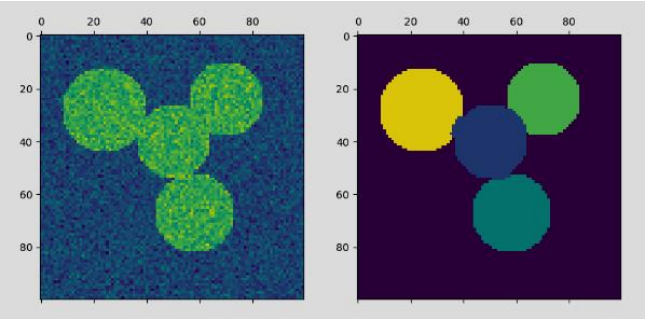


Figura 3-16. Ejemplo de uso de Spectral Clustering para la segmentación de imágenes.

Fuente: https://scikit-learn.org/stable/auto_examples/cluster/plot_segmentation_toy.html#sphx-glr-auto-examples-cluster-plot-segmentation-toy-py

El Spectral Clustering presenta varias ventajas significativas. En primer lugar, es altamente efectivo en la detección de estructuras de comunidades, incluso en grafos con formas complejas y distribuciones irregulares de enlaces. Esta capacidad se debe a su capacidad para analizar propiedades globales del grafo. Además, es menos sensible a la inicialización en comparación con otros algoritmos de clustering como k-means. En términos de ventajas adicionales, el agrupamiento espectral simplifica y reduce la dimensionalidad del conjunto de datos, facilitando así el análisis al hacer que sea más manejable. Además, es conocido por su popularidad en el campo del aprendizaje automático, superando a otros enfoques en muchos casos.

No obstante, el Spectral Clustering también tiene sus desventajas. El cálculo de los autovectores puede ser computacionalmente costoso, especialmente para grafos muy grandes, y la especificación previa del número de comunidades k puede ser un desafío en situaciones donde este valor no es conocido de antemano. Para abordar esto, es común explorar varios valores de k y utilizar técnicas como el "elbow plot" (gráfica de codo) para determinar el número óptimo de clústeres. Este método consiste en graficar la variación total dentro de los clusters en función de k, buscando identificar el punto donde esta variación deja de disminuir significativamente. Este punto, conocido como el "codo", indica el número óptimo de clusters para una partición efectiva de los datos.

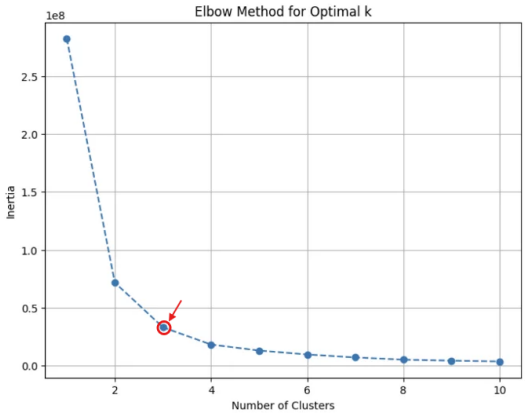


Figura 3-17. Ejemplo de elbow plot con el codo en 3 clusters.

Fuente: https://databasecamp.de/en/ml/k-means-clustering

En resumen, el Spectral Clustering es una técnica poderosa que encuentra aplicaciones en una amplia gama de campos, gracias a su capacidad para manejar estructuras complejas y su flexibilidad en el análisis de datos.

4 IMPLEMENTACIÓN Y SIMULACIONES

En este capítulo se presentará la implementación y simulación de los conceptos explicados durante esta memoria, aplicados a dos casos de estudio: la red de vuelos europeos y la red de metro de Sevilla. El objetivo es analizar la estructura y características de estas redes.

En primer lugar, pondremos en práctica los conceptos previamente explicados comenzando con la creación de los grafos que representan estas redes. Luego, procederemos a aplicar diversos algoritmos para el análisis de estos grafos, tales como el algoritmo de Betweenness Centrality o el algoritmo de Louvain, entre otros. Finalmente, cerraremos el análisis examinando la robustez de estas redes mediante la simulación de posibles fallos o ataques intencionados. Esto se logrará estudiando la red mientras se eliminan nodos y enlaces de la misma.

Para llevar a cabo estas tareas, hemos optado por utilizar Google Colab. Este servicio alojado de Jupyter Notebook no requiere configuración alguna y es proporcionado de forma gratuita. Jupyter es el proyecto de código abierto en el que se basa Colab, ofreciendo un entorno de desarrollo interactivo que muestra la ejecución del código a través del navegador web. Permite dividir el código en partes y ejecutarlas por separado, observando los resultados paso a paso. La elección de Google Colab se basa en su conveniencia y versatilidad; a diferencia de Jupyter, no requiere descargas y es compatible con varios lenguajes, incluyendo Python, que es el seleccionado para este trabajo. El código fuente completo de ambos casos de estudio se encuentra disponible a través de: https://drive.google.com/file/d/1jW9ZoLhflPYfDyt3M5tN69OUB8Zuki72/view?usp=drive_link

La elección de Python se justifica por su amplia gama de librerías, destacando NetworkX en el análisis de redes. NetworkX ofrece una variedad de funciones para crear y analizar grafos, incluyendo algoritmos para calcular métricas de centralidad, encontrar caminos más cortos y detectar comunidades. Su interfaz intuitiva facilita la visualización de grafos, y su integración con otras librerías como Matplotlib y Pandas garantiza un flujo de trabajo eficiente.

4.1 Red de vuelos europeos

La red de vuelos europeos representa el primer caso de estudio que se tratará en este trabajo. La construcción del grafo que representa esta red se basará en una cuidadosa recopilación de información, la cual será extraída de [https://openflights.org/data]. Este recurso proporciona una amplia gama de datos relacionados con los aeropuertos, las aerolíneas, las rutas y los países, ofreciendo así una base sólida para la creación de nuestro grafo.

El proceso de construcción del grafo no se limitará únicamente a la recopilación de datos. Será necesario realizar un preprocesamiento adecuado para garantizar la precisión y la integridad de la red resultante, eliminando posibles entradas duplicadas o datos inconsistentes, la falta de especificación de la posición del aeropuerto o la ausencia de origen/destino en un vuelo. Una vez construido el grafo, procederemos a aplicar los algoritmos ya mencionados, que nos permitirán explorar diversas características de la red de vuelos europeos, como la centralidad de los aeropuertos, la identificación de comunidades de rutas aéreas y la robustez de la red frente a posibles fallos o ataques.

4.1.1 Descripción de librerías

- **Pandas:** Se trata de una librería de Python especializada en la manipulación y el análisis de datos. Proporciona estructuras de datos flexibles y eficientes, como DataFrames, que facilitan la manipulación y análisis de grandes conjuntos de datos, es como un Excel de Python. Además, permite leer ficheros en formato CSV o Excel, acceder a sus datos mediante índices o nombres para las filas o columnas y también ofrece métodos para reordenar, dividir y combinar conjuntos de datos.

  En este trabajo, Pandas se utilizará para procesar los datos de aeropuertos, aerolíneas y rutas extraídos de OpenFlights, permitiendo su trasformación en formatos válidos para la construcción de grafos a partir de estos.

- **Numpy:** Se trata de una librería en Python especializada en el cálculo numérico y el análisis de datos. Esta da soporte para crear vectores y matrices multidimensionales, junto con una gran colección de funciones matemáticas para operar con ellas. La funcionalidad principal que ofrece esta librería, que la hace una de las más empleadas en Python, es su estructura de datos "narray", para una matriz de n dimensiones. Estas matrices o listas, aunque debe contener todos los elementos del mismo tipo, proporcionan un procesamiento considerablemente más alto que las listas tradiciones de Python y es por eso que son tan empleadas.

  En este trabajo, Numpy se empleará para realizar cálculos numéricos necesarios para el análisis de redes, tales como la media del grado de los nodos, así como para trabajar con los grandes conjuntos de datos de manera eficiente.

- **Matplotlib:** Se trata de una librería para la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o arrays en el lenguaje de programación Python. Permite crear y personalizar una amplia variedad de gráficos como el diagrama de barras, el histograma o el diagrama de sectores, entre otros.

  En este trabajo, Matplotlib.pyplot se utilizará para generar representaciones visuales de los grafos y para ilustrar los resultados de los análisis realizados, como la centralidad de los aeropuertos y la detección de comunidades en la red de vuelos. Pyplot es el módulo de Matplotlib que propone varias funciones sencillas para añadir elementos tales como líneas, imágenes o textos a los ejes de un gráfico.

- **Networkx:** Se trata de, problemente, la librería de Python mas relevante en nuestro trabajo, en lo que al análisis de los grafos se refiere. Es una librería especializada en el estudio de grafos y análisis de redes. Entre sus funcionalidades destaca, el uso de clases para la creación de grafos no dirigidos y dirigidos, conversión de grafos entre diversos tipos, capacidad de encontrar subgrafos e incorpora diversos algoritmos para el análisis de los grafos como adyacencia, grado, betweenness, pagerank…

  En este trabajo tendrá multitud de utilidades, desde la creación del grafo de la red de vuelos hasta para aplicar algoritmos como el de betweenness centrality o el algoritmo de Louvain.

- **Cartopy:** Se trata de un paquete de Python diseñado para el procesamiento de datos geográficos con el fin de producir mapas y hacer análisis de datos geospaciales. La clase principal para integrar cartopy en matplotlib es GeoAxes, que es una subclase de un matplotlib Axes normal. La clase GeoAxes agrega funcionalidad adicional a un eje que es específico para dibujar mapas. Resumiendo, podemos decir que cartopy utiliza matplotlib para crear unos "ejes" o entorno para el mapa, que se enriquezcan para poder utilizar coordenadas geográficas y no solo cartesianas. Proporciona una interfaz fácil de usar para la proyección de datos geográficos y la creación de mapas temáticos. Cartopy integra características como la visualización de costas, fronteras y otros elementos geográficos, lo que permite crear mapas detallados y precisos.

En este trabajo, Cartopy se integrará para generar mapas que representen gráficamente las rutas aéreas entre aeropuertos europeos, proporcionando una visualización geográfica clara y precisa de la red de vuelos.

- **Random:** La librería Random de Python se utiliza para generar números aleatorios y realizar selecciones aleatorias. Ofrece una variedad de funciones para generar valores aleatorios, seleccionar elementos aleatoriamente de una secuencia, y más.

  En este trabajo, Random se empleará para simular fallos aleatorios, seleccionando los nodos de la red que tendrán un fallo y ver como se comportaría la red en ese caso.

- **IPython.display:** La librería IPython.display proporciona herramientas para la visualización de medios enriquecidos en entornos Jupyter. Permite incrustar y mostrar imágenes, videos, HTML, y otros tipos de medios dentro de un notebook de Jupyter.

  En este trabajo, se utilizará IPython.display para mostrar animaciones y videos que ilustran la dinámica de la red de vuelos y los efectos de los fallos simulados en la misma. Esto incluirá la visualización de la evolución del grafo a medida que se eliminan nodos, ayudando a comunicar de manera efectiva los resultados del análisis de robustez de la red.

- **Sklearn.cluster:** es parte del paquete de aprendizaje automático scikit-learn en Python. Esta sub-librería proporciona una variedad de algoritmos para realizar clustering, una técnica no supervisada que agrupa datos similares en conjuntos (clusters).

  En este trabajo, se empleará esta librería como herramienta para aplicar una de las técnicas de detección de comunidades, el algoritmo de Spectral Clustering, mediante funciones que ya tiene implementadas como KMeans o SpectralClustering.

```
!pip install cartopy pycountry

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import networkx.algorithms.community as nx_comm
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from matplotlib.animation import FuncAnimation
import random
from IPython.display import Video, display
from sklearn.cluster import KMeans
```

Figura 4-1. Instalación e importación de librerías para la red de vuelos.

4.1.2 Diseño y representación del grafo

Primeramente, será necesario crear el grafo para poder trabajar sobre él y analizar sus características. Una vez preparadas las librerías y paquetes requeridos, el primer paso será la extracción de los datos relacionados con los vuelos y aeropuertos, lo cual nos permitirá posteriormente crear el grafo de la red de vuelos europeos. Para ello, se utilizará la librería pandas para leer los datos desde la página correspondiente y rellenar la tabla con estos datos. Es necesario definir previamente las columnas que posee esta tabla, que se corresponderán con aquellas que serán leídas en formato CSV. Deberemos hacer esto tanto para la información de los aeropuertos como para la de los vuelos.

```
#Nombres de las columnas de datos
names_routes = ('airline,airline_id,'
         'source,source_id,'
         'dest,dest_id,'
         'codeshare,stops,equipment').split(',')

#Lectura de datos para rellenar la tabla
routes = pd.read_csv(
    'https://github.com/jpatokal/openflights/blob/master/data/routes.dat?raw=true',
    names=names_routes,
    header=None)
```

Figura 4-2. Fragmento para la obtención de los datos de los vuelos.

Una vez extraídos los datos sobre los aeropuertos, será el momento de seleccionar solo aquellos vuelos y aeropuertos que nos interesan, en este caso vuelos que se produzcan dentro de Europa y que sean válidos. Para ello, se seleccionan, de la tabla "airports_df" donde se ha almacenado la información sobre los aeropuertos, solo aquellas filas cuyo código IATA o ICAO del aeropuerto no sea nulo y que sean aeropuertos, puesto que en el enlace también aparece información acerca de estaciones de trenes o de ferry. Por último, se seleccionan aquellos aerupertos que se encuentren en algún país de Europa y se filtran los vuelos para quedarnos únicamente con aqullos que se dan entre dos aeropuertos europeos.

```
names_airports = ('id,name,city,country,iata,icao,lat,lon,'
        'alt,timezone,dayligthst,tz,type,source').split(',')
airports_df = pd.read_csv(
'https://github.com/jpatokal/openflights/blob/master/data/airports-extended.dat?raw=true',
    header=None,
    names=names_airports,
    na_values='\\N')



#Buscamos aquellas filas con nombre del aeropuerto no nulo
airports_df['nom'] = airports_df.apply(lambda x: not pd.isnull(x['iata']) or not pd.isnull(x['icao']), axis=1)
airports = airports_df[airports_df['nom']]


#Filtramos los datos para solo aquellos que sean aeropuertos (el enlace contiene tambien estaciones de tren, de ferry ...)
airports = airports[airports['type'] == 'airport']

#Nos quedamos solo con los vuelos europeos
european_airports = airports[airports['country'].isin(['Albania','Andorra','Armenia','Austria','Azerbaijan','Belarus','Belgium','Bosnia and Herzegovina','Bulg
european_routes = routes[routes['source'].isin(european_airports['iata']) & routes['dest'].isin(european_airports['iata'])]
```

Figura 4-3. Fragmento para la obtención de los datos de los aeropuertos y filtro para quedarse con los datos europeos.

Una vez obtenidos todos los datos necesarios, se puede proceder a la construcción del grafo. Para ello, se crea un grafo dirigido usando la librería Networkx con la función 'DiGraph()', a partir de la lista de enlaces que formarán el grafo y que se corresponden con los vuelos, con su origen y su destino.

```
edges = european_routes[['source','dest']].values
graph = nx.from_edgelist(edges, create_using=nx.DiGraph())
```

Figura 4-4. Creación del grafo de los vuelos europeos.

Por último, tras la creación del grafo dirigido que representa los vuelos entre aeropuertos europeos, se procede a representar dicho grafo sobre el mapa de Europa usando la librería Cartopy. Para conseguirlo, se crea un diccionario que almacene la posición de cada aeropuerto, es decir, que para cada código IATA del aeropuerto se almacena tanto la longitud como la latitud correspondiente a su ubicación, para posteriormente poder situarlo en el mapa. Seguidamente, se define una lista para escalar los diferentes aeropuertos en función de su grado.

```
# Diccionario para almacenar las posiciones de los aeropuertos
pos = {row['iata']: (row['lon'], row['lat']) for idx, row in european_airports.iterrows()}

# Lista para almacenar el tamaño de los nodos en función de su grado
deg = nx.degree(graph)
sizes = [deg[iata]/10 for iata in subgraph.nodes]
```

Figura 4-5. Diccionario para las posiciones de los aeropuertos y tamaños de estos para representarlos en el mapa.

Ahora solo queda crear la proyección del mapa y dibujar el grafo sobre el mismo. Primero, se crea una figura con Matplotlib que contendrá la proyección del mapa, añadiendo una serie de opciones adicionales para mostrar las fronteras entre países. Tras esto, se selecciona la extensión que se quiere mostrar en la figura, en nuestro caso el mapa de Europa, y se dibuja el grafo con Networkx, usando el diccionario de posiciones para los aeropuertos y escalando el tamaño de los nodos según su grado.

```
# Proyección del mapa
crs = ccrs.PlateCarree()
fig, ax = plt.subplots(1, 1, figsize=(12, 8), subplot_kw=dict(projection=crs))
ax.coastlines()
ax.add_feature(cfeature.BORDERS, linestyle='-', linewidth=0.1, edgecolor='black')

#Ponemos la extension para el mapa de Europa y dibujamos el grafo
ax.set_extent([-40, 120, 20, 90])
nx.draw_networkx(subgraph, ax=ax, font_size=1, alpha=1, width=.02,
                 node_size=sizes, pos=pos, with_labels=False, arrowsize=1)
```

Figura 4-6. Representación del grafo de los vuelos sobre el mapa.

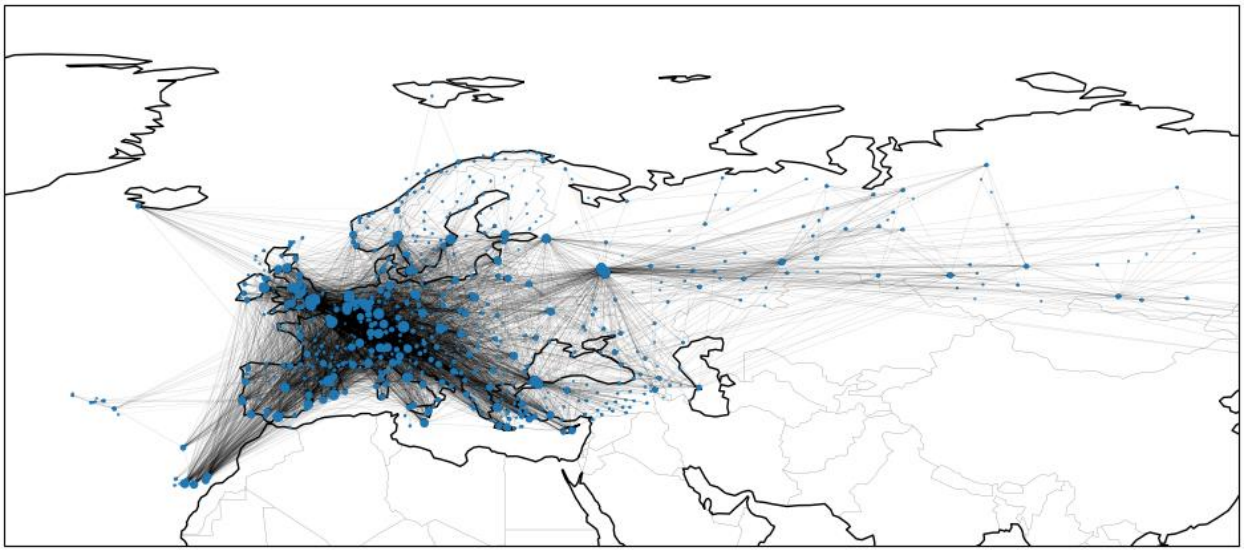


Figura 4-7. Resultado de la representación del grafo usando Cartopy.

Resulta esencial realizar un análisis detallado del tipo de red que representa la red de vuelos europeos después de visualizar su estructura. A primera vista, parece seguir una distribución de grado que sugiere una red de escala libre. Este fenómeno se puede verificar visualmente mediante la representación de un histograma del grado de los nodos.

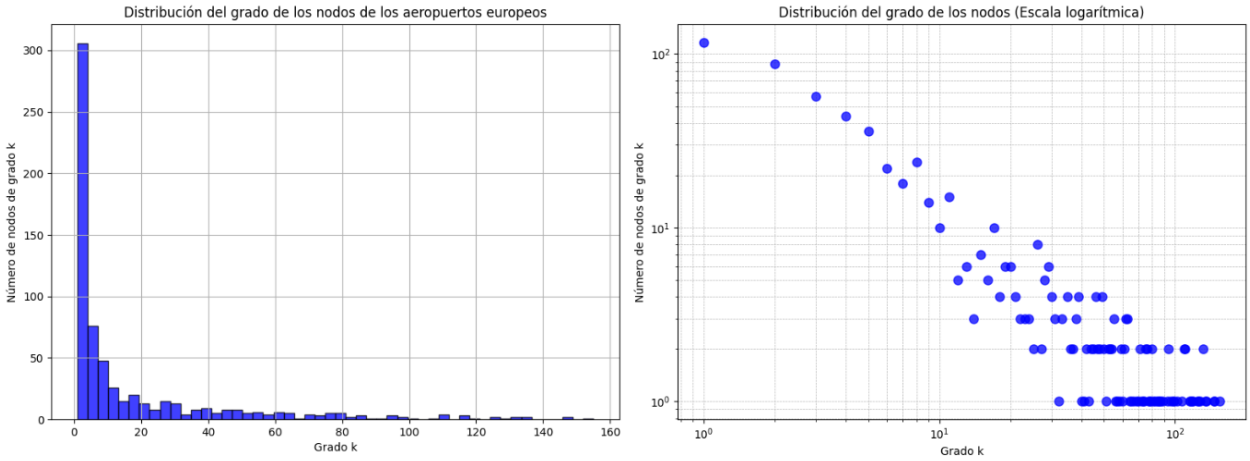


Figura 4-8. Distribución del grado de los nodos en la red de vuelos europeos.

En la parte izquierda de la figura anterior, se muestra claramente la distribución del grado de los nodos en la red de vuelos europeos. Esta distribución sugiere seguir una ley de potencias, como se observa en la gráfica log-log en la parte derecha de la figura, donde la pendiente aproximada de -1.02 indica esta característica de escala libre. Esta interpretación se fundamenta en la relación logarítmica entre el grado de los nodos y su frecuencia, como se explicó anteriormente al distinguir entre los diferentes tipos de redes según su estructura. En redes de escala libre, esta distribución implica que solo unos pocos aeropuertos tienen una conectividad muy alta y son considerados "hubs", mientras que la mayoría de los aeropuertos tienen un grado muy bajo.

4.1.3 Aplicación del Algoritmo de Betwenness Centrality

Una vez formado el grafo sobre el que se va a trabajar, ya es posible aplicar los algoritmos que deseemos para analizar y estudiar sus características. En esta sección, aplicaremos el algoritmo de Betweenness Centrality, que asigna un valor a cada nodo en función de su centralidad de intermediación. Es decir, se evalúa el número de caminos más cortos entre dos nodos que pasan por el nodo en cuestión. Para realizar esto, se emplea la función betweenness_centrality() de la librería Networkx, la cual implementa este algoritmo y devuelve un diccionario donde las claves son los nodos de los grafos y los valores son los coeficientes de centralidad de intermediación de esos nodos. Posteriormente, seleccionaremos aquellos nodos con los mayores coeficientes.

```
betweenness_centr = nx.betweenness_centrality(graph)

#Los ordenamos por valor de centralidad
betweenness_centr = sorted(betweenness_centr.items(), key=lambda x: x[1], reverse=True)

#Cogemos los 30 nodos/aeropuertos con mayor valor
most_betcen = betweenness_centr[:30]
```

Figura 4-9. Obtención de los aeropuertos con mayor centralidad de intermediación.

Nodo	Centralidad de Intermediación
Domodedovo International Airport, Russia	0.1270
Stockholm-Arlanda Airport, Sweden	0.0810
Istanbul Airport, Turkey	0.0762

Sheremetyevo International Airport, Russia	0.0751
Oslo Lufthavn, Norway	0.0688
Eleftherios Venizelos International Airport, Greece	0.0548
Helsinki Vantaa Airport, Finland	0.0501
Pulkovo Airport, Russia	0.0459
London Stansted Airport, United Kingdom	0.0427
Barcelona International Airport, Spain	0.0344
Paris-Orly Airport, France	0.0342
Amsterdam Airport Schiphol, Netherlands	0.0328
Sabiha Gökçen International Airport, Turkey	0.0317
Palma De Mallorca Airport, Spain	0.0305
Vnukovo International Airport, Russia	0.0300
Munich Airport, Germany	0.0294
London Gatwick Airport, United Kingdom	0.0287
Tromsø Airport,, Norway	0.0279
Düsseldorf Airport, Germany	0.0245
Alicante International Airport, Spain	0.0234
Edinburgh Airport, United Kingdom	0.0227
Glasgow International Airport, United Kingdom	0.0225
Frankfurt am Main Airport, Germany	0.0222
Adolfo Suárez Madrid–Barajas Airport, Spain	0.0214
Málaga Airport, Spain	0.0213
Humberto Delgado Airport (Lisbon Portela Airport), Portugal	0.0210
Vienna International Airport, Austria	0.0202
Copenhagen Kastrup Airport, Denmark	0.0198
Dublin Airport, Ireland	0.0196
Leonardo da Vinci–Fiumicino Airport, Italy	0.0181

Tabla 4-1. Puntuaciones de centralidad de intermediación de la red de vuelos europeos.

A continuación, se muestra en la siguiente figura el grafo de la red de vuelos europeos, pero únicamente se incluyen los nodos con mayor valor de centralidad de intermediación según la tabla mencionada, así como los enlaces o vuelos entre estos nodos.

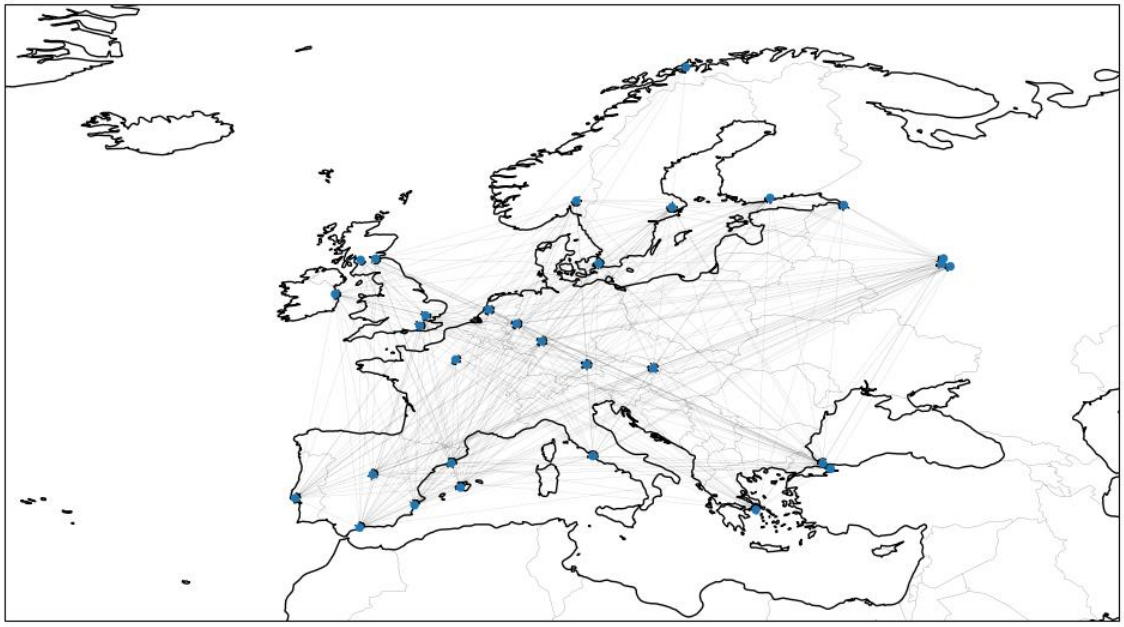


Figura 4-10. Representación de los aeropuertos con mayor centralidad de intermediación.

Los resultados obtenidos y ordenados nos muestran los 30 aeropuertos con las puntuaciones más altas de centralidad de intermediación. Un punto importante a destacar es que el aeropuerto con mayor centralidad es el Domodedovo International Airport de Rusia. Esto indica que este aeropuerto es un nodo crucial en la red, probablemente debido a su papel como un importante centro de conexiones en Rusia y posiblemente en Europa del Este. Además, se puede observar cómo los aeropuertos con alta centralidad están distribuidos a lo largo de Europa, incluyendo países de Europa Occidental (Reino Unido, España, Francia), Europa del Este (Rusia) y el norte de Europa (Suecia, Noruega, Finlandia). Esto refleja una distribución variada de nodos críticos que facilitan el flujo de pasajeros y carga a través del continente.

Este análisis de centralidad es una métrica crucial para entender la estructura y dinámica de la red de transporte aéreo. Los aeropuertos con alta centralidad son puntos críticos que, si se interrumpen, pueden afectar significativamente el flujo de pasajeros y mercancías en toda la red. Realizar este análisis resulta realmente útil para la planificación y optimización de redes, así como para desarrollar planes de contingencia en caso de interrupciones en estos puntos clave.

4.1.4 Aplicación del Algoritmo de PageRank

A continuación, procedemos a aplicar el algoritmo de PageRank. Este algoritmo, al igual que el algoritmo de Betweenness Centrality, se trata de un algoritmo de centralidad que asigna una puntuación a cada nodo del grafo en función de su importancia en la red. Originalmente desarrollado por Google para clasificar páginas web, el algoritmo de PageRank asigna una puntuación a cada nodo del grafo basado en la cantidad y calidad de los enlaces entrantes. En el contexto de una red de aeropuertos, una puntuación alta de PageRank indica que un aeropuerto es importante no solo por su conectividad directa, sino también por la importancia de los aeropuertos

que lo conectan. Para aplicar este algoritmo, hemos desarrollado una función que simula su funcionamiento.

```
def pagerank (graph, iterator, damping=0.85, tolerance=1e-06):
    num_nodos = graph.number_of_nodes()
    pagerank = {node: 1.0/num_nodos for node in graph.nodes()}

    for _ in range (iterator):
      new_pagerank = {}
      infl = {node: pagerank[node] / graph.out_degree(node) if graph.out_degree(node) != 0 else 0 for node in graph.nodes()}


      for n in graph.nodes():
        dir_a_nodo = list(graph.predecessors(n))
        new_pagerank[n] = (1-damping) * 1/num_nodos + damping * sum(infl[predecesor] for predecesor in dir_a_nodo)

      # Verificar convergencia
      convergence = sum(abs(new_pagerank[node] - pagerank[node]) for node in graph.nodes())
      if convergence < tolerance:
          break

      pagerank = new_pagerank

    return pagerank
```

Figura 4-11. Definición de la función para calcular el PageRank.

En esta función tendremos cuatro parámetros de entrada: el grafo, el número de iteraciones máximo, el factor de atenuación o "dumping factor", que determina la probabilidad de continuar navegando por los nodos, y un último argumento que indica el umbral a partir del cual se considera que el algoritmo ha convergido, es decir, cuando los valores dejan de cambiar significativamente. Los dos últimos tienen asignados valores recomendados por defecto, aunque es posible especificar otros si se desea. La función asigna un valor inicial de PageRank idéntico para todos los nodos y, a partir de entonces, calcula repetidamente el nuevo valor utilizando la fórmula 3-3, hasta alcanzar el valor de convergencia o hasta que se hayan realizado el número máximo de iteraciones indicadas en la llamada a la función.

Una vez definida la función para calcular el valor de PageRank de los nodos, simplemente la aplicamos a la red de vuelos europeos y extraemos los aeropuertos con una puntuación mayor.

```
pr = pagerank(subgraph, 300)

# Se ordenan en función del valor de PageRank
pr = sorted(pr.items(), key=lambda x: x[1], reverse=True)

# Se seleccionan los 30 nodos/aeropuertos con mayor puntuación
most_pr = pr[:30]
```

Figura 4-12. Obtención de los aeropuertos con mayor PageRank.

Nodo	PageRank
Domodedovo International Airport, Russia	0.0158
Istanbul Airport, Turkey	0.0118
Stockholm-Arlanda Airport, Sweden	0.0105
London Stansted Airport, United Kingdom	0.0103

Sheremetyevo International Airport, Russia	0.0101
Amsterdam Airport Schiphol, Netherlands	0.0099
Munich Airport, Germany	0.0094
Barcelona International Airport, Spain	0.0093
Eleftherios Venizelos International Airport, Greece	0.0086
Frankfurt am Main Airport, Germany	0.0086
Pulkovo Airport, Russia	0.0085
Oslo Lufthavn, Norway	0.0084
Palma De Mallorca Airport, Spain	0.0083
London Gatwick Airport, United Kingdom	0.0083
Dublin Airport, Ireland	0.0081
Helsinki Vantaa Airport, Finland	0.0080
Düsseldorf Airport, Germany	0.0080
Vienna International Airport, Austria	0.0078
Charles de Gaulle International Airport, France	0.0076
Sabiha Gökçen International Airport, Turkey	0.0075
Málaga Airport, Spain	0.0072
Leonardo da Vinci–Fiumicino Airport, Italy	0.0072
Copenhagen Kastrup Airport, Denmark	0.0071
Manchester Airport, United Kingdom	0.0071
Adolfo Suárez Madrid–Barajas Airport, Spain	0.0070
Brussels Airport, Belgium	0.0070
Paris-Orly Airport, France	0.0067
Berlin-Tegel Airport, Germany	0.0066
Alicante International Airport, Spain	0.0065
Zürich Airport, Switzerland	0.0061

Tabla 4-2. Puntuaciones de PageRank de la red de vuelos europeos.

A continuación, se muestra en la siguiente figura el grafo de la red de vuelos europeos, pero solo se incluyen los

nodos con mayor puntuación de PageRank según la tabla mencionada, así como los enlaces o vuelos entre estos nodos.

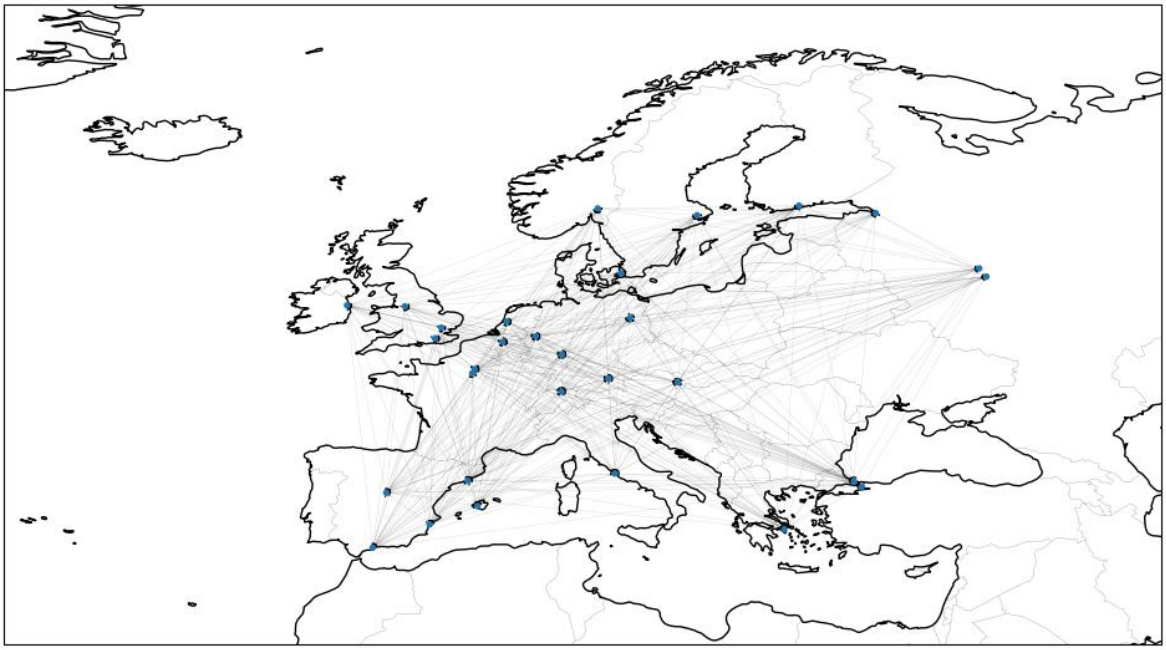


Figura 4-13. Representación de los aeropuertos con mayor PageRank.

Los resultados obtenidos, ordenados por sus valores de PageRank, revelan los 30 aeropuertos más importantes según este algoritmo. Destaca el Domodedovo International Airport en Rusia, que encabeza la lista, lo que indica su alta importancia dentro de la red. Este aeropuerto es un nodo significativo, no solo por su conectividad directa, sino también por su influencia en la red global. La lista de aeropuertos con mayor puntuación de PageRank tiene similitudes con la lista basada en centralidad de intermediación, lo que sugiere que muchos de los aeropuertos no solo son puentes clave en la red, sino también puntos de referencia importantes. Sin embargo, hay algunos aeropuertos, como el Charles de Gaulle International Airport, que aparecen en la lista de PageRank pero no en la de centralidad de intermediación, lo que indica que, aunque no son puentes cruciales, tienen una alta conectividad con otros aeropuertos importantes.

Los aeropuertos con alta puntuación de PageRank están repartidos por toda Europa, lo que refleja la diversidad y la interconexión de la red aérea europea. Sin embargo, hay ciertos países como el Reino Unido y Alemania que tienen múltiples aeropuertos en la lista, lo que subraya su fuerte conectividad y la importancia de su infraestructura aeroportuaria. El análisis de PageRank aplicado a la red de aeropuertos europeos proporciona una visión profunda de la importancia relativa de cada aeropuerto dentro de la red. Este conocimiento es crucial para una variedad de aplicaciones prácticas, desde la optimización de redes de transporte hasta la gestión de riesgos y la planificación estratégica en diversos campos.

4.1.5 Aplicación del Algoritmo de Louvain

Una vez aplicados algunos algoritmos de centralidad a la red de vuelos europeos, procedemos a emplear el algoritmo para la detección de comunidades de Louvain. Este algoritmo se basa en la idea de encontrar grupos de nodos altamente conectados entre sí, pero poco conectados con el exterior. Para lograr esto, el algoritmo de Louvain se centra en la optimización de la modularidad, que se define como la diferencia entre el número de conexiones observadas dentro de las comunidades y el número esperado de conexiones al azar. Para aplicar este algoritmo, utilizamos la función ‘louvain_communities’ de la librería ‘networkx.algorithms.community’.

```
# Una resolución menor supone un menor número de comunidades, pero de mayor tamaño
# Una resolución mayor supone un mayor número de comunidades, pero de menor tamaño

resolution=1.0
communities = nx_comm.louvain_communities(subgraph,resolution=resolution)

# Se asignan los nodos del grafo a las comunidades
set_node_community(subgraph, communities)

# Se crea una lista de colores para las diferentes comunidades
node_color = [get_color(subgraph.nodes[v]['community']) for v in subgraph.nodes]
```

Figura 4-14. Obtención de comunidades en la red de vuelos mediante el algoritmo de Louvain.

La función utilizada para la detección de comunidades consta de dos argumentos importantes: el grafo y la resolución. La resolución determina el nivel de detalle o precisión con el que se identifican las comunidades. Un valor de resolución bajo identifica un menor número de comunidades, pero estas comunidades tienden a ser más grandes, mientras que un valor de resolución alto identifica un mayor número de comunidades, pero estas tienden a ser más pequeñas.

Una vez identificadas las comunidades, se utiliza una función llamada 'set_node_community', que asigna a cada nodo un atributo indicando a qué comunidad pertenece. Finalmente, se crea una lista con diferentes colores para los nodos generados mediante la función 'get_color'. Se itera sobre todos los nodos en el grafo y se asigna un color a cada nodo basado en la comunidad a la que pertenece.

A continuación, se presentan los resultados obtenidos al aplicar la resolución por defecto, que es igual a uno.

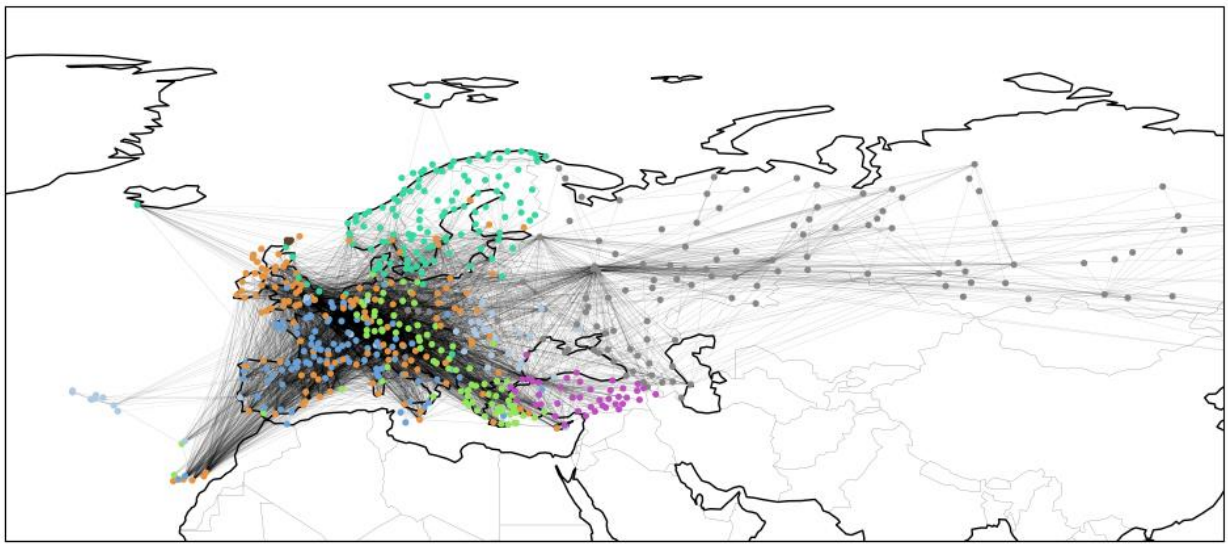


Figura 4-15. Representación de las comunidades obtenidas mediante el algoritmo de Louvain para la red de vuelos europeos.

El número de comunidades obtenidas mediante el algoritmo con la resolución indicada es de ocho, que se describen seguidamente de mayor a menor tamaño:

- **Comunidad 1 (color gris claro):** esta comunidad agrupa principalmente Rusia y Europa del Este. Esto puede ser debido a la fuerte conectividad interna entre estos aeropuertos, que forman una subred densa en la región.
- **Comunidad 2 (color naranja claro):** Esta comunidad parece agrupar aeropuertos secundarios y

terciarios de Europa Occidental, incluyendo Irlanda, España, Italia, Reino Unido y otros. La agrupación sugiere que estos aeropuertos tienen una alta conectividad entre sí, posiblemente por rutas de aerolíneas de bajo coste.

- **Comunidad 3 (color verde azulado claro):** Esta comunidad está formada por aeropuertos de los países nórdicos y bálticos (Suecia, Noruega, Dinamarca, Finlandia, etc.). Los aeropuertos en esta comunidad tienen una fuerte conectividad interna debido a la cercanía geográfica y la integración económica y social entre estos países.
- **Comunidad 4 (color azul):** Esta comunidad agrupa aeropuertos principalmente de Francia, España e Italia, incluyendo algunos aeropuertos en Suiza y Bélgica. La alta conectividad entre estos aeropuertos refleja la interconexión de vuelos dentro de Europa Occidental y el Mediterráneo, posiblemente influenciada por vuelos domésticos y turísticos.
- **Comunidad 5 (color verde claro):** Aeropuertos en esta comunidad abarcan principalmente Alemania, Suiza, Austria, y algunos aeropuertos en los Balcanes. Esto indica una conectividad regional específica centrada en vuelos entre estos países.
- **Comunidad 6 (color morado claro):** Esta comunidad agrupa aeropuertos de Turquía. La alta conectividad interna se debe a los vuelos domésticos dentro del país.
- **Comunidad 7 (color azul claro):** Esta comunidad se centra principalmente en aeropuertos de Rumanía y algunos de Bulgaria. La fuerte conectividad interna refleja vuelos domésticos y regionales dentro de estos países.
- **Comunidad 8 (color marrón):** Esta comunidad agrupa aeropuertos ubicados en las Islas Orcadas y Shetland en Escocia. La alta conectividad entre estos aeropuertos indica una red de vuelos regionales en esta área, donde los vuelos son necesarios para conectar estas islas remotas con el resto del Reino Unido.

El análisis de comunidades, basado en la estructura de la red de aeropuertos y sus conexiones, ha revelado patrones geográficos y de conectividad significativos. Cada color en el mapa representa una comunidad de aeropuertos que están más densamente conectados entre sí que con aeropuertos de otras comunidades. Esta visualización permite identificar rápidamente las principales regiones de conectividad aérea y comprender mejor las dinámicas de conectividad aérea y los patrones de tráfico en diferentes áreas geográficas.

4.1.6 Aplicación del Algoritmo de Spectral Clustering

Seguidamente, se procede a aplicar el algoritmo de Spectral Clustering para la detección de comunidades en esta red. La implementación de este algoritmo se basa en el uso de autovectores de una matriz construida a partir del grafo que permita representar los nodos en un espacio de menor dimensión, donde se puede aplicar técnicas como la de k-means de forma eficaz.

Para llevar a cabo esta implementación, primeramente, se obtiene la matriz de adyacencia del grafo y posteriormente se emplea la función SpectralClustering de la librería sklearn indicando esta matriz, el número de comunidades que se desea hallar y se indica el valor ‘discretize’ en el argumento ‘assign_labels’ que indica el método que se utilizará para asignar las etiquetas a los clusters. Otro posible valor para este argumento es ‘kmeans’ pero en nuestro caso se emplea ‘discretize’ ya que se trata de un método más robusto y con resultados más estables.

```
# Obtener la matriz de adyacencia
A = nx.adjacency_matrix(undirected)
num_clusters = 10  # Número de clusters deseados

# Aplicar el algoritmo de SpectralClustering
clustering = SpectralClustering(n_clusters=num_clusters, assign_labels='discretize', random_state=0).fit(A)
cluster_labels = {airport: int(clustering.labels_[i]) for i, airport in enumerate(undirected.nodes())}
```

Figura 4-16. Fragmento para la aplicación de Spectral Clustering para la red de vuelos europeos.

Al igual que con el algoritmo de Louvain, se asigna un color para cada comunidad. A continuación, se presentan los resultados obtenidos al aplicar la resolución por defecto, que es igual a uno.

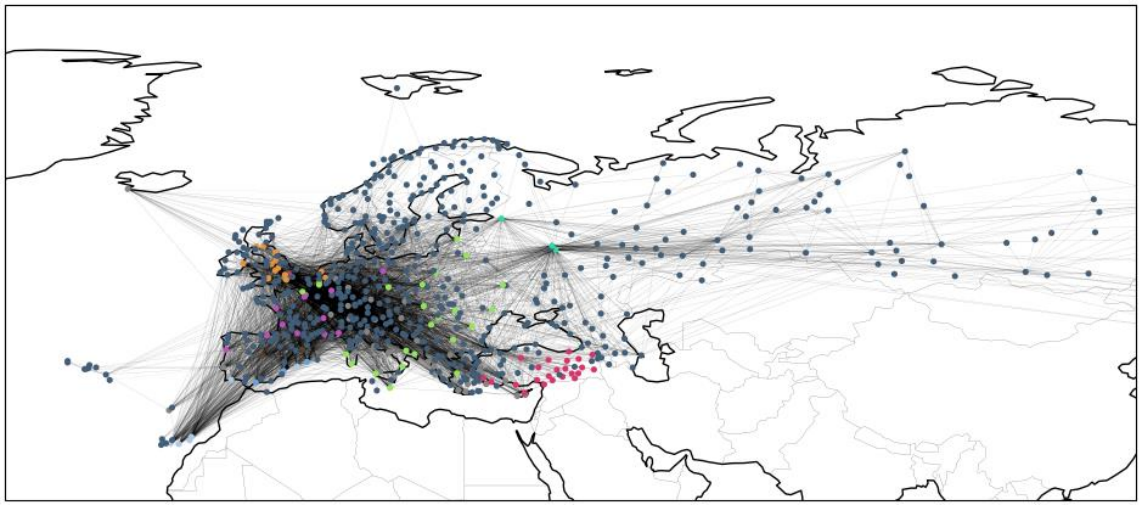


Figura 4-17. Representación de las comunidades obtenidas mediante el algoritmo de Spectral Clustering para la red de vuelos europeos.

A continuación, se describen brevemente las diez comunidades obtenidas:

- **Comunidad 1:** Esta comunidad agrupa una gran cantidad de aeropuertos, principalmente de Rusia y Europa del Este, formando una subred densa en esta región.
- **Comunidad 2:** Incluye aeropuertos como AGP, PMI y ALC en España, que son destinos turísticos populares.
- **Comunidad 3**: Agrupa aeropuertos de Europa Central y del Este, como BDS, ATH, BEG, y BUD, indicando fuerte conectividad regional.
- **Comunidad 4:** Principalmente incluye aeropuertos en Francia y Portugal, como BOD, BIO, y NCE, reflejando la interconexión en Europa Occidental.
- **Comunidad 5:** Comprende aeropuertos en Reino Unido, Países Bajos, Bélgica e Irlanda, como BRS, AMS, y BRU, sugiriendo alta conectividad regional.
- **Comunidad 6:** Agrupa aeropuertos importantes de Moscú como DME, LED y SVO, reflejando su importancia para la aviación rusa.
- **Comunidad 7:** Incluye los principales aeropuertos de España, BCN y MAD, indicando su importancia como hubs de transporte.
- **Comunidad 8:** Agrupa aeropuertos en Europa Occidental y el Mediterráneo, como ZRH, GVA y LCA, sugiriendo una densa red de vuelos internacionales.
- **Comunidad 9:** Incluye destinos turísticos como LPA, TFS e IBZ, reflejando la popularidad de estos destinos.
- **Comunidad 10:** Agrupa aeropuertos en Turquía como STN, ADA y BJV, debido a vuelos domésticos y su importancia turística.

Al analizar los resultados obtenidos mediante el algoritmo de Spectral Clustering, se observa una tendencia a formar una comunidad gigante que agrupa la mayoría de los aeropuertos, mientras que las demás comunidades resultan ser significativamente más pequeñas en comparación. Esta característica puede llevar a resultados menos definidos y más difíciles de interpretar. En contraste, el algoritmo de Louvain demuestra una capacidad

superior para producir comunidades de tamaños más equitativos, optimizando la modularidad y ofreciendo particiones más balanceadas y significativas. Por lo tanto, los resultados obtenidos con Spectral Clustering, aunque útiles en ciertos contextos, no son los más adecuados en este caso para identificar comunidades con una estructura más clara y equilibrada entre sí, a diferencia del algoritmo de Louvain.

4.1.7 Estudio de la Robustez de la Red

Para finalizar, se realiza un estudio de la robustez de esta red. La robustez de una red se define como su capacidad para mantener su correcto funcionamiento y estructura frente a fallos o ataques intencionados. Esta característica es fundamental en el diseño y mantenimiento de redes reales, ya que implica que el tráfico de pasajeros y carga a través del continente se vea afectado lo menos posible frente a estos eventos inesperados.

Para llevar a cabo este estudio, se ha aplicado el criterio de Molley-Reed, que establece que solo existe una componente gigante si el cociente $K = < k^2 > / < k >$ es mayor que dos. Se ha elaborado la función correspondiente que calcula tanto la conectividad media de la red como la media del cuadrado de los grados, y devuelve el cociente que corresponde con K. Es importante mencionar que, para la elaboración de este estudio, se ha convertido el grafo dirigo a un grafo no dirigido para simplificar el análisis de la componente gigante.

```python
# Función para calcular K
def calculate_K(G):
    degrees = [d for n, d in G.degree()]
    if len(degrees) == 0:
        return 0
    mean_k = np.mean(degrees)
    mean_k_square = np.mean([k**2 for k in degrees])
    return mean_k_square / mean_k
```

Figura 4-18. Función para el cálculo de K.

Ahora pasamos a definir la función encargada de ir eliminando los nodos y enlaces del grado y analizar la red, calculando el tamaño de la componente gigante o almacenando la lista de nodos eliminados, entre otros. La primera función, denominada "fallos_aleatorios", se encarga de realizar este análisis simulando fallos en nodos aleatorios de la red. Esto se asemeja a la situación en la que algunos aeropuertos de forma repentina y aleatoria dejan de estar operativos por cualquier fallo o inconveniente. Esta función recibe dos argumentos de entrada: el grafo que se desea analizar y el número de nodos a eliminar de la red, es decir, el número máximo de nodos que se simularán que fallan al mismo tiempo.

```python
# Función para eliminar nodos aleatorios y analizar la componente gigante
def fallos_aleatorios(G, num_nodes):
    G_copy = G.copy()
    K_values = []
    giant_component_sizes = []
    steps = []
    proportion_giant_component = []
    snapshots = []
    nodes_removed = []

    for step in range(num_nodes):
        if len(G_copy) == 0:
            break  # Salir del bucle si la red está vacía

        # Encontrar un nodo aleatorio
        node_to_remove = random.choice(list(G_copy.nodes))
        # Eliminar el nodo
        G_copy.remove_node(node_to_remove)
        nodes_removed.append(node_to_remove)

        # Calcular K
        K = calculate_K(G_copy)
        K_values.append(K)

        # Calcular el tamaño de la componente gigante
        if len(G_copy) > 0:
            largest_cc = max(nx.connected_components(G_copy), key=len)
            size_largest_cc = len(largest_cc)
            proportion_giant = size_largest_cc / G.number_of_nodes()
        else:
            size_largest_cc = 0
            proportion_giant = 0

        proportion_giant_component.append(proportion_giant)
        steps.append(step + 1)

        # Guardar una copia del grafo cada 5 pasos
        if step % 5 == 0 or step == num_nodes - 1:
            snapshots.append(G_copy.copy())

        # Si K <= 2, considerar que la componente gigante ha desaparecido y detener el proceso
        if K <= 2:
            break

    return steps, K_values, proportion_giant_component, snapshots, nodes_removed
```

Figura 4-19. Función para la simulación de fallos aleatorios en la red.

En esta función, se utilizan una serie de variables para almacenar diferentes valores útiles para el análisis de la

robustez de la red. El primer paso consiste en seleccionar un nodo aleatorio de la red utilizando la librería random. Este nodo se eliminará del grafo (se realiza una copia del grafo original al inicio de la función para no modificarlo) y se almacenará en una lista que contendrá todos los nodos eliminados. A continuación, se utiliza la función 'calculate_K', explicada anteriormente, para calcular K, y se almacena este valor en una lista que contendrá los valores de K en los diferentes pasos.

Posteriormente, se calcula el tamaño de la componente gigante, así como la proporción de nodos del grafo que forman parte de la componente gigante en función del número total de nodos del grafo. Además, cada cinco pasos se guarda una copia del grafo, que se utilizará posteriormente para crear una animación que representará la evolución de la red ante estos fallos aleatorios en diferentes nodos. Estos pasos se repiten hasta que el grafo desaparezca por completo o hasta que no se cumpla el criterio de Molloy-Reed.

Para el estudio de la robustez de la red frente a ataque intencionados, se podría reutilizar la función 'calculate_K'. Sin embargo, en este caso, sería necesario definir una nueva función llamada 'ataques_intencionados', que realizaría las mismas operaciones que la función 'fallos_aleatorios'. La implementación de la función sería exactamente la misma, excepto que la selección del nodo a eliminar no sería aleatoria, sino que se escogería aquel con el mayor grado, ya que el objetivo es simular ataques intencionados a la red, los cuales suelen dirigirse a los puntos críticos y más importantes de la misma.

```python
# Función para eliminar los nodos de mayor grado y analizar la componente gigante
def ataques_intencionados(G, num_nodes):
    G_copy = G.copy()
    K_values = []
    giant_component_sizes = []
    steps = []
    proportion_giant_component = []
    snapshots = []
    nodes_removed = []

    for step in range(num_nodes):
        if len(G_copy) == 0:
            break  # Salir del bucle si la red está vacía

        # Escoger el nodo de mayor grado de la red
        node_to_remove = max(G_copy.degree, key=lambda x: x[1])[0]
```

Figura 4-20. Función para la simulación de ataques intencionados a la red (solo el fragmento distinto a la función para la simulación de fallos aleatorios).

Una vez definidas estas funciones, es posible invocarlas para el estudio de la robustez de la red. Después de invocar cualquiera de las funciones definidas anteriormente, ya sea la que simula los ataques intencionados o la que simula fallos aleatorios, se necesita definir la proyección del mapa sobre la cual se dibujarán los diferentes grafos obtenidos en estas funciones, con los nodos y enlaces correspondientes eliminados.

Para crear la animación, se utiliza la librería matplotlib.animation, a partir de una función denominada "update", que va dibujando el estado del grafo en los diferentes pasos de la simulación sobre la proyección definida previamente del mapa. Una vez creada la animación, se puede mostrar mediante el uso de la librería IPython.display.

```python
# Función para actualizar la animación
def update(num):
    ax.clear()
    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle='-', linewidth=0.1, edgecolor='black')
    ax.set_extent([-40, 120, 20, 90])  # Ajusta la extensión del mapa a Europa

    # Dibujar el grafo en el mapa
    sizes = [subgraph.degree(node)/10 for node in snapshots[num].nodes()]
    nx.draw_networkx(snapshots[num], ax=ax, font_size=1, alpha=1, width=.02,
                     node_size=sizes, pos=pos, with_labels=False)

    ax.set_title(f"Proporción Componente Gigante {proportion_giant_component[num]}, Nodo eliminado: {nodes_removed[num]}")

# Crear la animación
ani = FuncAnimation(fig, update, frames=len(snapshots), repeat=False)
plt.tight_layout()

# Guardar la animación como un video y descargarla
ani.save('/tmp/Fallos_aleatorios.mp4', writer='ffmpeg')

# Mostrar la animación
display(Video('/tmp/Fallos_aleatorios.mp4', embed=True))
```

Figura 4-21. Creación y visualización de la animación correspondiente a la simulación de fallos aleatorios en la red de vuelos europeos.

Después de realizar ambas simulaciones, se pueden graficar los resultados obtenidos para analizar el comportamiento de la red en estos casos. Se han creado dos gráficos: uno que muestra cómo varía la proporción de nodos que pertenecen a la componente gigante en función de los nodos eliminados, y otro que ilustra cómo varía el valor de K, utilizado para comprobar si existe una componente gigante en la red, también en función del número de nodos eliminados. Estos gráficos permiten observar y comparar la robustez de la red frente a diferentes estrategias de eliminación de nodos.

Los resultados obtenidos en el caso de la simulación de los ataques intencionados muestran cómo la proporción de la componente gigante disminuye rápidamente en la primera de las gráficas de la figura 4-22. La eliminación de los nodos de mayor grado provoca una fragmentación rápida de la red, lo que indica que esta es muy vulnerable ante ataques que se dirigen a sus nodos/aeropuertos más conectados. En la segunda de las gráficas que muestra la evolución de K, se puede apreciar cómo este valor cae abruptamente, lo que indica que la red pierde su estructura de conectividad de manera significativa. Alcanza un valor menor de dos con la desaparición de unos 160 nodos, punto a partir del cual la red se considera ya altamente fragmentada y desaparece la componente gigante siguiendo el criterio de Molloy-Reed.

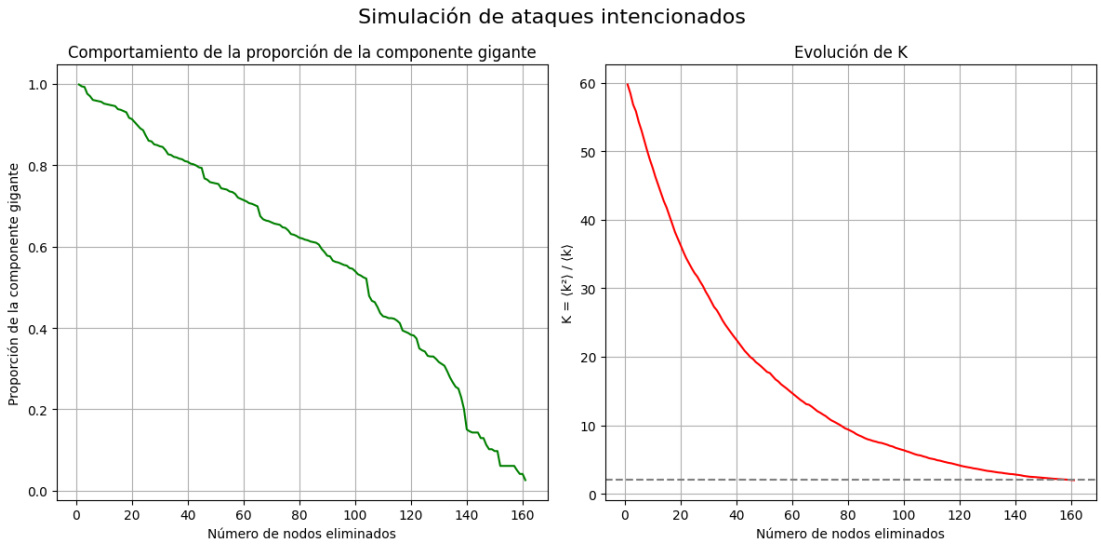


Figura 4-22. Resultados de la simulación de ataques intencionados en la red de vuelos europeos.

Los resultados obtenidos en la simulación de fallos aleatorios en la red muestran que la proporción de la componente gigante disminuye de forma más gradual en comparación con los resultados obtenidos con la simulación de ataques, como se puede apreciar en la primera gráfica de la figura 4-23. La red mantiene su conectividad durante más tiempo debido a que la selección de los nodos a eliminar se realiza de forma aleatoria en lugar de seleccionar los nodos con mayor conectividad. Esto sugiere que la red tiene una estructura inherentemente robusta frente a fallos aleatorios.

En la segunda gráfica, que muestra la evolución de K, se puede observar cómo este valor cae de forma más lenta, indicando que la red mantiene una mejor conectividad global incluso después de eliminar muchos nodos. Además, para que la componente gigante desaparezca y la red se fragmente, es necesario eliminar casi la totalidad de los nodos de la red, alrededor de unos 650. Esta cantidad es mucho mayor que en el caso de los ataques intencionados.

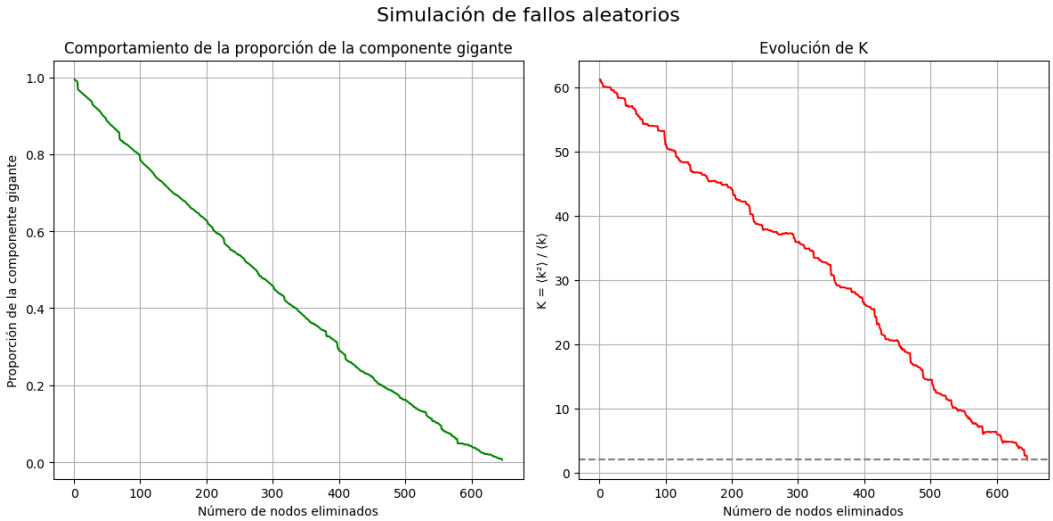


Figura 4-23. Resultados de la simulación de fallos aleatorios en la red de vuelos europeos.

Este análisis revela la vulnerabilidad de las redes frente a diferentes tipos de ataques y fallos. Las redes son particularmente vulnerables a ataques intencionados, donde los nodos más conectados son eliminados, resultando en una rápida fragmentación. Sin embargo, muestran una mayor resiliencia frente a fallos aleatorios. Entender estos aspectos es crucial para diseñar redes más robustas y resilientes, especialmente en aplicaciones críticas como redes de comunicación, infraestructura y sistemas de transporte, como el caso de estudio.

En la siguiente figura se puede visualizar una comparativa entre los resultados obtenidos en ambos casos.

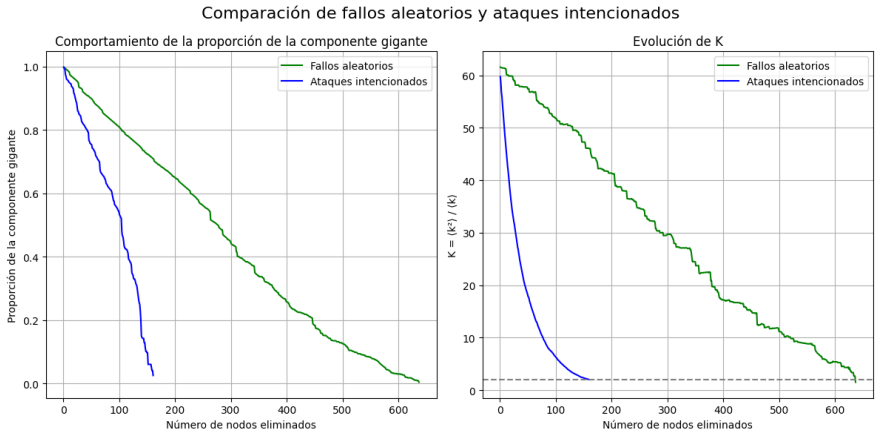


Figura 4-24. Comparativa de los resultados obtenidos sobre el estudio de la robustez de la red de vuelos.

4.2 Red de metro de Sevilla

La red de metro de Sevilla representa el segundo caso de estudio abordado en este trabajo. Actualmente, consta únicamente de la línea Ciudad Expo - Olivar de Quintos. Sin embargo, se está contemplando la posibilidad de mejorar la comunicación en Sevilla mediante la construcción de nuevas líneas de metro. La siguiente figura ilustra el concepto de lo que se pretende lograr con la red de metro en la ciudad.

Figura 4-25. Plano de la red de metro completa de Sevilla.

La idea de transformar la red de metro en un grafo y analizarla mediante las técnicas existentes puede resultar bastante interesante para predecir futuros comportamientos y observar posibles características y medidas aplicables al proyecto actual. Actualmente, no es posible extraer información automáticamente, como en el caso de estudio anterior, para construir el grafo a partir de ella, por lo que será necesario definir manualmente los enlaces entre estaciones y la información correspondiente para hacer una representación que se ajuste a la realidad.

En este caso, la construcción del grafo puede ser más tediosa debido a la labor manual requerida, pero una

vez construido, será posible aplicar todas las técnicas utilizadas anteriormente. Se procederá a emplear los métodos y algoritmos ya mencionados, que nos permitirán explorar diversas características de la red de metro en estudio, como la centralidad de las estaciones, la identificación de comunidades en las líneas y la robustez de la red frente a posibles fallos o ataques.

4.2.1 Descripción de librerías

La mayoría de las bibliotecas necesarias para este caso de estudio ya han sido definidas previamente en el caso de la red de vuelos europeos, como por ejemplo la biblioteca matplotlib para la creación de figuras y representación de gráficos, IPython.display para mostrar la animación creada con la simulación de la desaparición de algunos nodos de la red cuando se lleva a cabo el estudio de su robustez, y random para seleccionar nodos de forma aleatoria. Sin embargo, hay algunas librerías necesarias para la ejecución de las pruebas que no se usaron en el caso anterior:

- **Folium:** Se trata de una biblioteca de Python utilizada para crear mapas interactivos. Se basa en Leaflet.js, una popular biblioteca de JavaScript para mapas interactivos, y permite a los usuarios generar mapas y agregarles capas de datos, incluyendo marcadores, líneas y polígonos, todo dentro de un entorno de Python. Folium es particularmente útil en análisis geoespacial y visualización de datos geográficos.

  En este trabajo, Folium se emplea para crear visualizaciones geográficas interactivas que representen la red de metro de Sevilla, en las que se pueda hacer zoom, visualizar los nombres de las diferentes paradas, etc. A diferencia de la representación de la red de vuelos europeos que se realizaba con la biblioteca Cartopy, en este caso se ha decidido emplear Folium para tener un mayor nivel de detalle.

- **Google_colab_selenium:** Esta biblioteca es un paquete específico que facilita el uso de Selenium dentro del entorno de Google Colab, la plataforma que utilizamos para el desarrollo del programa. Entre los usos más comunes de esta biblioteca destacan: el uso de navegadores en modo headless, configurando Selenium en modo sin cabeza, lo cual es necesario ya que en Google Colab no hay ninguna interfaz gráfica disponible. También destaca la automatización de pruebas web, permitiendo a los usuarios de Google Colab ejecutar scripts de Selenium para la automatización de navegadores web directamente desde los cuadernos de Colab.

  En este trabajo, Google_colab_selenium se utiliza para ayudar a tomar capturas de pantalla de los mapas de la red de metro de Sevilla generados durante la simulación, las cuales se usarán posteriormente para la creación de una animación sobre la evolución de la red.

- **os:** En Python, el módulo os es una biblioteca estándar que proporciona una interfaz para interactuar con el sistema operativo en el que se ejecuta el programa. Permite realizar operaciones relacionadas con la gestión de archivos, directorios, rutas, variables de entorno y otras funcionalidades del sistema operativo. Algunas de las características más comunes son la manipulación de rutas y nombres de archivos, el listado, creación y eliminación de archivos y directorios, y la manipulación de variables de entorno.

  En este trabajo, os se emplea para manejar rutas de archivos, como abrir archivos HTML con Selenium.

- **time:** Esta biblioteca en Python es un módulo estándar que proporciona varias funciones relacionadas con la medición y la manipulación del tiempo. Es una herramienta esencial para controlar la temporización en scripts, ejecutar funciones después de ciertos intervalos y medir la duración de ejecución de segmentos de código. Es capaz de suspender la ejecución del hilo actual durante un tiempo determinado o devolver la hora actual, entre otras funciones.

  En este trabajo, la biblioteca time se emplea para suspender la ejecución del hilo durante unos segundos al utilizar google_colab_selenium para que la página buscada tenga tiempo de cargarse en el navegador.

4.2.2 Diseño y representación del grafo

El primer paso para analizar la red de metro de Sevilla es generar el grafo que represente esta red, donde los nodos corresponden a las estaciones del metro y las aristas o enlaces representan las conexiones entre estas estaciones que forman las diferentes líneas del metro.

Para crear el grafo, es necesario definir manualmente tanto los diferentes enlaces entre las estaciones como la posición de estas estaciones para representarlas en el mapa de forma aproximada. Para almacenar las conexiones entre estaciones, se utiliza la variable 'enlaces', que es una lista de tuplas en la que cada tupla representa un enlace entre dos estaciones del metro de Sevilla. Cada tupla contiene el nombre de la estación de origen y el nombre de la estación de destino. Por otro lado, la variable 'estaciones' es un diccionario donde las claves son los nombres de las estaciones y los valores son tuplas que representan las coordenadas geográficas (latitud y longitud) de cada estación.

Una vez definidas estas variables, se crean dos diccionarios adicionales: uno que contiene los colores para las cuatro líneas del metro y otro que contiene la lista de estaciones que pertenecen a cada línea. En este último diccionario, la clave es el nombre de la línea de metro y los valores son listas de tuplas, donde cada tupla contiene el nombre de la estación y sus coordenadas. Posteriormente, se agregan las diferentes tuplas de estaciones y coordenadas a este último diccionario, que inicialmente se ha definido sin ningún valor.

```
# Definir colores para cada línea de metro
line_colors = {
    "Linea 1": "green",
    "Linea 2": "skyblue",
    "Linea 3": "red",
    "Linea 4": "yellow"
}
# Crear un diccionario para almacenar las estaciones por línea
estaciones_por_linea = {
    "Linea 1": [],
    "Linea 2": [],
    "Linea 3": [],
    "Linea 4": []
}
# Agrupar las estaciones por línea
for estacion, coords in estaciones.items():
    if estacion in ["Ciudad Expo", "Cavaleri", "San Juan Alto", "San Juan Bajo", "Blas Infante", "Parque de los Princ
        estaciones_por_linea["Linea 1"].append((estacion, coords))
    elif estacion in ["Hospital San Juan de Dios", "Bormujos-Gines", "Nueva Sevilla", "CC Aire Sur", "Tomares", "Cama
        estaciones_por_linea["Linea 2"].append((estacion, coords))
    elif estacion in ["Higeron Norte", "Pino Montano Norte", "Pino Montano", "Los Mares", "Los Carteros", "San Lazaro
        estaciones_por_linea["Linea 3"].append((estacion, coords))
    else:
        estaciones_por_linea["Linea 4"].append((estacion, coords))
```

Figura 4-26. Agrupación de las estaciones de metro en las diferentes líneas.

Seguidamente, procedemos a crear el mapa de Sevilla sobre el cual representaremos el grafo y a construir el propio grafo. Para la creación del mapa, necesitaremos introducir las coordenadas aproximadas del centro de Sevilla. Esto nos permitirá, mediante el uso de la librería Folium, extender el mapa desde ese centro, según lo indicado en el argumento 'zoom_start'. Para la construcción del grafo, utilizaremos la librería Networkx, comenzando con la creación del grafo vacío y luego añadiendo los nodos y enlaces.

```python
# Coordenadas del centro de Sevilla
sevilla_coords = (37.3886303, -5.9824231)

# Crear un mapa centrado en Sevilla
mapa_sevilla = folium.Map(location=sevilla_coords, zoom_start=12, zoom_control=False, scrollWheelZoom=False)

# Crear un grafo vacío
metro_sevilla = nx.Graph()

# Agregar nodos al grafo
for estacion, coords in estaciones.items():
    metro_sevilla.add_node(estacion, pos=coords)

# Agregar aristas al grafo
metro_sevilla.add_edges_from(enlaces)
```

Figura 4-27. Generación del mapa de Sevilla y de la red de metro.

Ahora, representaremos el grafo de la red de metro de Sevilla en el mapa generado con Folium. Para esta representación, utilizaremos objetos PolyLine para los enlaces entre estaciones y CircleMarker para las estaciones, los cuales se añadirán al mapa mencionado. En el caso de las conexiones entre estaciones, recorreremos la lista de enlaces de la red, extrayendo las estaciones de origen y destino. Luego, determinaremos a qué línea pertenecen estas estaciones con el objetivo de colorear el enlace con el color correspondiente a esa línea del metro. En cuanto a las estaciones, cabe destacar el uso del argumento 'tooltip', que permite interactuar con el mapa; al pasar el ratón sobre alguna de las estaciones, aparece el nombre de la misma.

```python
# Dibujar las aristas del grafo como líneas en el mapa
for edge in metro_sevilla.edges():
    start_station = edge[0]
    end_station = edge[1]

    # Obtener las líneas a las que pertenece la estación de inicio
    start_lines = [line for line, stations in estaciones_por_linea.items() if start_station in [station[0] for station in stations]]

    # Obtener las líneas a las que pertenece la estación de fin
    end_lines = [line for line, stations in estaciones_por_linea.items() if end_station in [station[0] for station in stations]]

    # Dibujar una línea para cada combinación de líneas
    for start_line in start_lines:
        for end_line in end_lines:
            if start_line == end_line:
                line_color = line_colors[end_line]
                start_node = estaciones[start_station]
                end_node = estaciones[end_station]
                folium.PolyLine(locations=[start_node, end_node], color=line_color).add_to(mapa_sevilla)

# Dibujar las estaciones como marcadores en el mapa
for estacion, coords in estaciones.items():
    folium.CircleMarker(location=coords, radius=2, color='blue', fill=True, fill_color='blue', fill_opacity=1.0, tooltip=estacion).add_to(mapa_sevilla)
```

Figura 4-28. Adicción del grafo de la red de metro sobre el mapa de Sevilla.

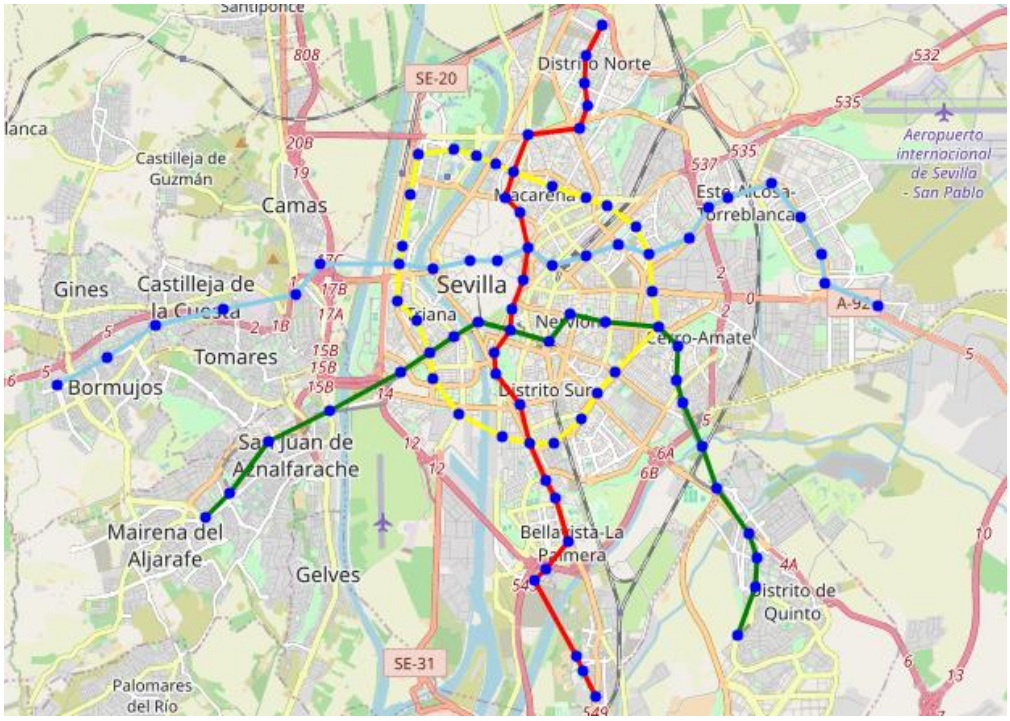


Figura 4-29. Representación de la red de metro de Sevilla.

De igual forma al caso de estudio anterior, vamos a representar a continuación la distribución del grado de los nodos en esta red de metro.

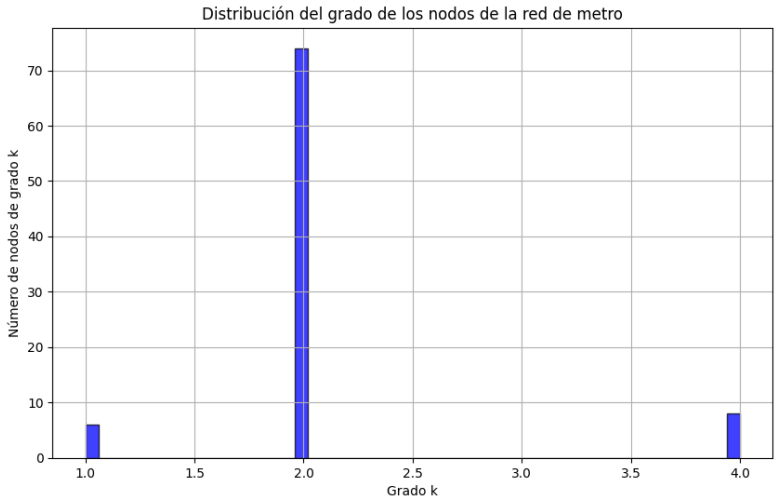


Figura 4-30. Distribución del grado de los nodos en la red de metro de Sevilla.

Esta red es mucho más pequeña que la red de vuelos europeos y presenta solo tres grados distintos para todos los nodos, lo que limita las conclusiones que se pueden extraer. La mayoría de las estaciones de metro tienen un grado dos, ya que pertenecen a una sola línea y se encuentran en el recorrido de esta. Algunas estaciones están conectadas únicamente a otra estación, siendo estas las de inicio o fin de línea. Además, hay estaciones con un grado cuatro, que son estaciones de transbordo que conectan dos líneas de metro diferentes.

4.2.3 Aplicación del Algoritmo de Betwenness Centrality

Una vez construido el grafo sobre el cual trabajaremos, podemos aplicar diversos algoritmos para analizar sus características. En esta sección, emplearemos el algoritmo de Centralidad de Intermediación (Betweenness Centrality), que asigna un valor a cada nodo en función de su importancia como intermediario en la red. Este valor se basa en el número de caminos más cortos entre pares de nodos que pasan a través del nodo en cuestión. Para realizar este cálculo, utilizaremos la función 'betweenness_centrality()' de la biblioteca Networkx, que implementa este algoritmo y devuelve un diccionario donde las claves son los nodos del grafo y los valores son los coeficientes de centralidad de intermediación de esos nodos.

A continuación, ordenaremos las estaciones de mayor a menor centralidad de intermediación y realizaremos una gráfica que represente estas puntuaciones de centralidad respecto a las estaciones.

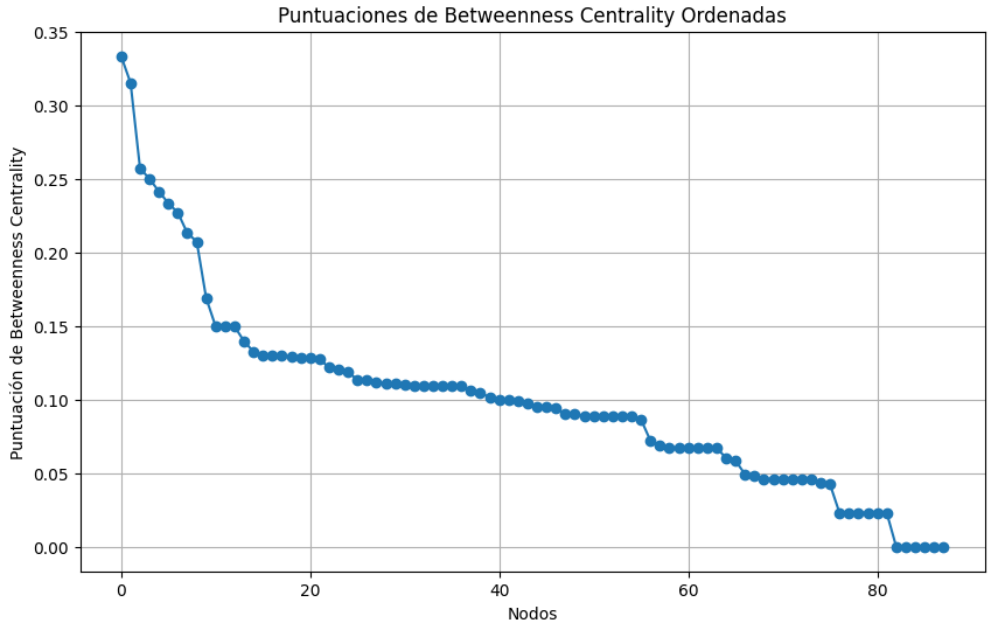


Figura 4-31. Gráfica de las puntuaciones de Betweenness Centrality.

Esta gráfica nos permite analizar cómo varían los valores de centralidad de intermediación. Observamos que los dos primeros nodos tienen una puntuación mucho mayor que el resto, indicando que son muy importantes en la red. Luego, encontramos otros siete nodos con una puntuación también bastante alta. Sin embargo, a partir de estos, se observa un gran salto con respecto al resto de las puntuaciones de los nodos. Es decir, a partir de los nueve primeros nodos, los valores de centralidad de los nodos comienzan a aplanarse. A continuación, se muestra la puntuación de estos nueve nodos más centrales.

Nodo	Centralidad de Intermediación
Clemente Hidalgo - 1 de Mayo	0.3333
Tesalonica	0.3152
Prado San Sebastian	0.2571
Maria Auxiliadora	0.2500
Parque de los Principes	0.2414

La Palmera	0.2333
Avenida Andalucia	0.2270
Torre Triana	0.2132
Hospital Macarena	0.2073

Tabla 4-3. Puntuaciones de centralidad de intermediación de la red de metro de Sevilla.

En la siguiente figura se representa el grafo de la red de metro, marcando en rojo las estaciones que aparecen en la tabla 4-3 y en azul el resto.

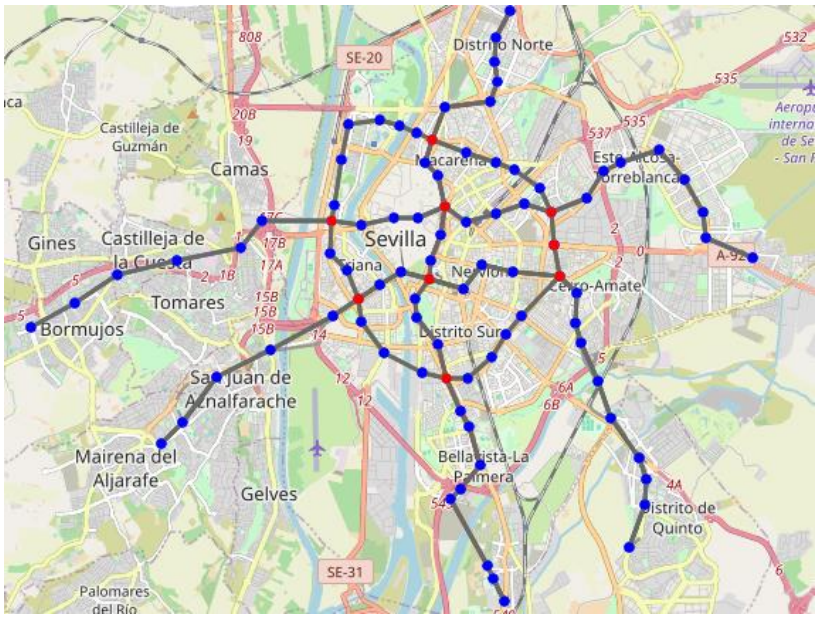


Figura 4-32. Representación de las estaciones con mayor centralidad de intermediación.

Los resultados obtenidos y ordenados nos muestran las nueve estaciones con las puntuaciones más altas de centralidad de intermediación. Es importante destacar que estas estaciones coinciden prácticamente en su totalidad con las estaciones de trasbordo o cambio de líneas, debido a su papel crucial en la interconexión de diferentes rutas y la facilitación del flujo de pasajeros entre ellas.

Este análisis de centralidad es una métrica crucial para entender la estructura y dinámica de esta red. Las estaciones con alta centralidad son puntos críticos que, si se interrumpen, pueden afectar significativamente el tráfico en la red y perjudicar la movilidad de las personas dentro de la ciudad a través de este sistema de transporte. Además, tener identificados estos puntos críticos permite desarrollar planes de contingencia para abordar posibles fallos en su funcionamiento y minimizar así el impacto en la movilidad urbana.

4.2.4 Aplicación del Algoritmo de PageRank

A continuación, procedemos a aplicar el algoritmo de PageRank. Como se explicó anteriormente, este algoritmo asigna a cada nodo un valor en función del número de enlaces entrantes y de la calidad o importancia de los mismos. Para realizar esta asignación, utilizaremos la función ‘pagerank()’ que ya fue definida para el caso de estudio de la red de vuelos europeos. Como se puede ver en la Figura 4-11, esta función calcula los valores de PageRank para los nodos del grafo, teniendo en cuenta la influencia transmitida por cada enlace y el factor de amortiguación (damping factor).

Es importante destacar que, para que la función 'pagerank()' funcione correctamente, el grafo debe ser dirigido. Sin embargo, el grafo definido para la red de metro no es dirigido, por lo que será necesario transformarlo utilizando la librería Networkx. Para ello, duplicaremos cada enlace en el sentido opuesto al que se define, creando así dos enlaces que unan cada par de nodos en ambos sentidos, simulando una conexión bidireccional. Esto nos permitirá aplicar el algoritmo de PageRank de manera efectiva, reflejando adecuadamente las dinámicas de interconexión y flujo de la red de metro.

```
# Convertir el grafo no dirigido en un grafo dirigido
metro_sevilla_dirigido = metro_sevilla.to_directed()

# Duplicar cada enlace para hacerlo bidireccional
for edge in metro_sevilla_dirigido.edges():
    metro_sevilla_dirigido.add_edge(edge[1], edge[0])

# Calcula el PageRank para el grafo dirigido
pagerank = pagerank(metro_sevilla_dirigido, 100)
```

Figura 4-33. Conversión a grafo dirigido la red de metro y cálculo de PageRank.

Una vez calculados estos valores, realizaremos un estudio similar al llevado a cabo con el algoritmo de Betweenness Centrality. Ordenaremos los valores de mayor a menor y los representaremos en una gráfica para identificar el punto de inflexión a partir del cual los valores de PageRank comienzan a aplanarse.

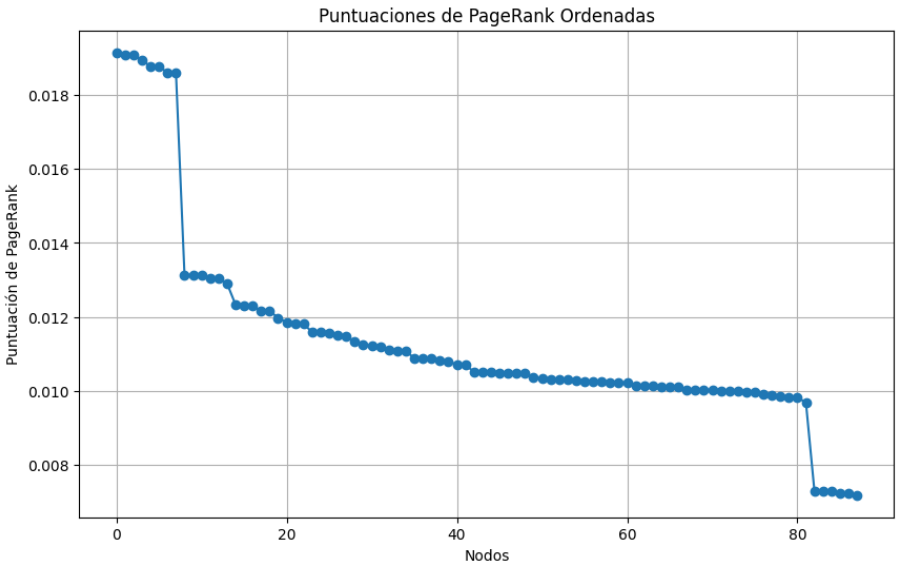


Figura 4-34. Gráfica de las puntuaciones de PageRank.

En la gráfica de la figura anterior, se observa claramente un gran salto entre la puntuación de PageRank del octavo y del noveno nodo, lo que indica el punto de inflexión que estábamos buscando. Como resultado de este análisis, tomaremos como referencia los ocho primeros nodos como los más centrales del grafo.

Nodo	PageRank
Hospital Macarena	0.0191
Torre Triana	0.0190
La Palmera	0.0190

Parque de los Principes	0.0189
Tesalonica	0.0187
Clemente Hidalgo - 1 de Mayo	0.0187
Maria Auxiliadora	0.0186
Prado San Sebastian	0.0185

Tabla 4-4. Puntuaciones de PageRank de la red de metro de Sevilla.

A continuación, se representa en la siguiente figura el grafo de la red de metro, marcando las estaciones que aparecen en la tabla 4-4 en color rojo y el resto en azul.

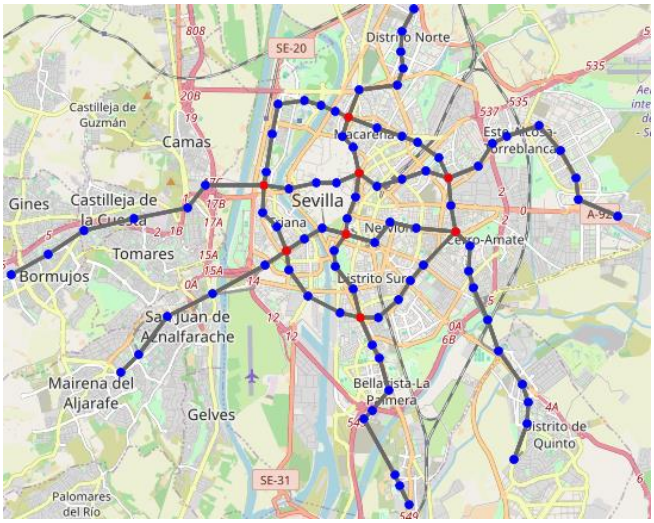


Figura 4-35. Representación de las estaciones con mayor PageRank.

Los resultados obtenidos muestran las ocho estaciones con las puntuaciones más altas de PageRank. Como se puede observar en la figura anterior, estas estaciones coinciden con las estaciones de trasbordo o cambio de líneas debido a su papel crucial en la interconexión de diferentes rutas y la facilitación del flujo de pasajeros entre ellas.

Al comparar estos resultados con los valores de centralidad de intermediación, observamos una fuerte correlación. Las estaciones con alta centralidad de intermediación, como Parque de los Príncipes y Tesalónica, también aparecen en la lista de estaciones con mayor puntuación de PageRank. Esto sugiere que estas estaciones no solo son puntos de transbordo críticos, sino también nodos con alta conectividad global dentro de la red. Identificar estas estaciones clave permite desarrollar planes de contingencia para abordar posibles fallos en su funcionamiento y minimizar así el impacto en la movilidad urbana.

4.2.5 Aplicación del Algoritmo de Louvain

Después de aplicar estos algoritmos de centralidad a la red, procedemos a utilizar el algoritmo de Louvain para la detección de comunidades, es decir, la identificación de grupos de nodos altamente conectados entre sí pero poco conectados con el resto. Para esto, haremos uso de la función ‘louvain_communities’ de la librería ‘networkx.algorithms.community’.

```
resolution=1.0
communities = nx_comm.louvain_communities(metro_sevilla,resolution=resolution)

# Asignar un color único a cada comunidad
community_colors = {}
for i, community in enumerate(communities):
    rgb_color = get_color(i)
    # Convertir el color a formato hexadecimal
    color_hex = "#{:02x}{:02x}{:02x}".format(int(rgb_color[0] * 255), int(rgb_color[1] * 255), int(rgb_color[2] * 255))
    for node in community:
        community_colors[node] = color_hex
```

Figura 4-36. Obtención de comunidades en la red de metro mediante el algoritmo de Louvain.

Esta función requiere dos argumentos importantes, como se mencionó en el caso de estudio anterior sobre la red de vuelos europeos: el grafo y la resolución.

Una vez identificadas las comunidades, crearemos una lista con diferentes colores para cada una de ellas mediante la función 'get_color'. Es importante destacar que, una vez generado un color para una comunidad, este se convertirá a formato hexadecimal, ya que es necesario para su posterior uso en la representación del mapa con Folium.

A continuación, se presentan los resultados obtenidos al aplicar la resolución por defecto, que es igual a uno. En el mapa representado, emplearemos el color de cada comunidad para los enlaces internos de la misma, mientras que los enlaces externos entre comunidades se representarán de color negro.

Figura 4-37. Representación de las comunidades obtenidas mediante el algoritmo de Louvain para la red de metro de Sevilla.

El número de comunidades obtenidas mediante el algoritmo con la resolución indicada es de nueve, que se describen seguidamente de mayor a menor tamaño:

- **Comunidad 1 (color azul claro):** Esta comunidad agrupa estaciones que se encuentran principalmente en la zona sur de Sevilla, abarcando áreas como Heliopolis y Palmas Altas. Compuesta por las estaciones de: Cortijo del Cuarto, Hospital de Valme, Heliopolis, Celestino Mutis, Cariñanos, Virgen del Rocio, Bueno Monreal, La Palmera, Reina Mercedes, Delicias,

Palmas Altas, Ciudad de la Justicia, Ermita Virgen Valme, Pineda.

- **Comunidad 2 (color marrón):** Se observa una concentración en torno al aeropuerto y la estación de tren Santa Justa. Compuesta por las estaciones de: San Pablo, Puerta Este, Kansas City, Santa Justa, Montesierra, Pedro Romero, Adelfa, Palacio de Congresos, Ciencias, Luis Uruñuela, Tesalonica, Aeronautica, Torreblanca.
- **Comunidad 3 (color verde oscuro):** Esta comunidad agrupa algunas de las zonas más céntricas y turísticas de Sevilla, incluyendo puntos emblemáticos como la Plaza de España y el Parque Maria Luisa. Compuesta por las estaciones de: Puerta Jerez, Maria Auxiliadora, Puerta de Carmona, Plaza de Cuba, Capuchinos, Macarena, Parque Maria Luisa, Jardines de Murillo, Cristo de Burgos, Prado San Sebastian, Plaza de España.
- **Comunidad 4 (color azul):** Concentra estaciones en el norte de Sevilla, cerca del Hospital Macarena. Compuesta por las estaciones de: Los Carteros, Higeron Norte, Los Mares, Hospital Macarena, Llanes, Pino Montano Norte, Carretera de Carmona, Pino Montano, Pio XII, San Lazaro.
- **Comunidad 5 (color verde claro):** Se trata de una comunidad que abarca estaciones en la parte occidental de Sevilla y municipios cercanos como San Juan. Compuesta por las estaciones de: San Jacinto, San Juan Bajo, Blas Infante, San Juan Alto, Cavaleri, Parque de los Principes, Ciudad Expo, Ronda de Triana, Virgen de la Oliva.
- **Comunidad 6 (color morado):** Esta comunidad está enfocada en la zona de Montequinto y áreas adyacentes. Compuesta por las estaciones de: Olivar de Quintos, Guadaira, Montequinto, La Plata, Cocheras, Pablo de Olavide, Europa, Condequinto, Amate.
- **Comunidad 7 (color naranja):** Concentra estaciones en la zona occidental y municipios cercanos como Camas y Tomares. Compuesta por las estaciones: Hospital San Juan de Dios, Torre Triana, CC Aire Sur, Nueva Sevilla, Bormujos-Gines, Plaza de Armas, Camas, Tomares, Duque
- **Comunidad 8 (color gris):** Agrupa estaciones en la parte oriental de Sevilla, en áreas comerciales y residenciales. Compuesta por las estaciones: Nervion, Gran Plaza, Ramon y Cajal, San Bernardo, Ronda de Tamarguillo, Clemente Hidalgo - 1 de Mayo, Avenida Andalucia.
- **Comunidad 9 (color azul oscuro):** Incluye estaciones en la Isla de la Cartuja y zonas industriales y tecnológicas. Compuesta por las estaciones: Ingenieros, Americo Vespucio, Remo, Cartuja, Expo, Jose Diaz.

El análisis de comunidades de la red de metro de Sevilla revela una clara división geográfica y funcional de la ciudad proporcionando una visión detallada y valiosa de la estructura interna de la red de metro de Sevilla, lo que puede ser utilizado para mejorar tanto la eficiencia operativa como la experiencia del usuario.

4.2.6 Aplicación del Algoritmo de Spectral Clustering

Seguidamente, se aplica el algoritmo de Spectral Clustering para la detección de comunidades. Este algoritmo utiliza autovectores de una matriz construida a partir del grafo, lo que permite representar los nodos en un espacio de menor dimensión, facilitando así la aplicación de la técnica de k-means.

Para desarrollar la implementación se sigue un procedimiento similar al explicado en la sección tres. El proceso comienza con el cálculo de la matriz laplaciana estándar, que se obtiene a partir de las matrices de adyacencia y de grados del grafo de la red de metro. Luego, se calculan los autovalores y autovectores de esta matriz laplaciana, seleccionando el número de autovectores necesario según la cantidad de comunidades que se deseen identificar. Estos autovectores forman una matriz de incrustación, la cual se utiliza para agrupar los nodos en un espacio reducido, y se aplica k-means sobre esta matriz para determinar las comunidades.

```python
# Obtener la matriz de adyacencia y la matriz de grados
A = nx.adjacency_matrix(metro_sevilla)
D = np.diag([d for n, d in metro_sevilla.degree()])

# Calcular la matriz Laplaciana estándar L = D - A
L = D - A
num_clusters = 9  # Número de clusters deseados


# Calcular y ordenar autovalores y autovectores de menor a mayor
eigenvalues, eigenvectors = np.linalg.eigh(L)
sorted_indices = np.argsort(eigenvalues)
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

autovalores_ord = eigenvalues[:num_clusters]
top_eigenvectors = eigenvectors[:, :num_clusters]  # Seleccionar los primeros k autovectores relevantes
embedding_matrix = top_eigenvectors

# Aplicar k-means sobre las filas de la matriz de incrustación
kmeans = KMeans(n_clusters=num_clusters, random_state=0, n_init=90)
labels = kmeans.fit_predict(embedding_matrix)
```

Figura 4-38. Fragmento para la aplicación de Spectral Clustering en la red de metro.

Para este ejemplo, se ha seleccionado un valor de nueve clusters para facilitar la comparación con los resultados obtenidos mediante el algoritmo de Louvain. Un parámetro importante es el argumento "n_init" de la función KMeans, que define cuántas veces se ejecutará el algoritmo con diferentes centroides para mejorar la precisión del resultado.

Una vez identificadas las comunidades, se asigna un color distinto a cada una utilizando la función "get_color", similar al enfoque del algoritmo anterior. Los resultados se ilustran en la siguiente figura, donde los enlaces internos de cada comunidad se representan en su color correspondiente, mientras que los enlaces externos entre comunidades se muestran en color negro.

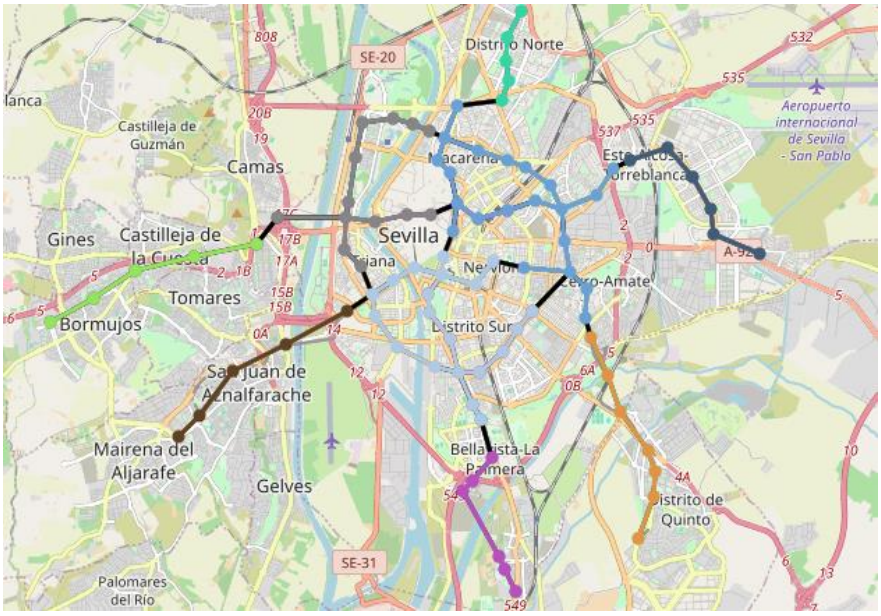


Figura 4-39. Representación de las comunidades obtenidas mediante el algoritmo de Spectral Clustering para la red de metro de Sevilla.

Las comunidades obtenidas son las siguientes:

- **Comunidad 1 (color morado):** Agrupa estaciones en el sur de Sevilla, alrededor de Palmas Altas y Hospital de Valme. Compuesta por las estaciones de: Cariñanos, Ciudad de la Justicia, Palmas Altas, Cortijo del Cuarto, Ermita Virgen Valme, Hospital de Valme.
- **Comunidad 2 (color azul oscuro):** Agrupa estaciones en la zona este de Sevilla, específicamente alrededor de Torreblanca y Adelfa. Compuesta por las estaciones de: Puerta Este, Palacio de Congresos, Ciencias, Adelfa, Aeronautica, Torreblanca.
- **Comunidad 3 (color azul claro):** Esta comunidad agrupa algunas de las zonas más céntricas y turísticas de Sevilla, junto a estaciones del sur. Compuesta por las estaciones de: Parque de los Principes, Ramon y Cajal, Ronda de Tamarguillo, Celestino Mutis, Virgen del Rocio, La Palmera, Reina Mercedes, Delicias, Virgen de la Oliva, Plaza de Cuba, Puerta Jerez, Prado San Sebastian, San Bernardo, Nervion, Jardines de Murillo, Plaza de España, Parque Maria Luisa, Bueno Monreal, Heliopolis, Pineda.
- **Comunidad 4 (color verde oscuro):** Concentra estaciones en el norte de Sevilla. Compuesta por las estaciones de: Higeron Norte, Pino Montano Norte, Pino Montano, Los Mares, Los Carteros.
- **Comunidad 5 (color marrón):** Se trata de una comunidad que abarca estaciones en la parte occidental de Sevilla y municipios cercanos como San Juan. Compuesta por las estaciones de: Blas Infante, San Juan Bajo, San Juan Alto, Cavaleri, Ciudad Expo.
- **Comunidad 6 (color naranja):** Agrupa estaciones en Montequinto y áreas adyacentes. Esta comunidad incluye estaciones en la periferia sur de Sevilla. Compuesta por las estaciones de: Cocheras, Guadaira, Pablo de Olavide, Condequinto, Montequinto, Europa, Olivar de Quintos
- **Comunidad 7 (color verde claro):** Agrupa estaciones en el oeste de Sevilla, incluyendo Bormujos, Gines, y Tomares. Compuesta por las estaciones: Hospital San Juan de Dios, Bormujos-Gines, Nueva Sevilla, CC Aire Sur, Tomares.
- **Comunidad 8 (color azul):** Una comunidad muy grande que agrupa estaciones centrales y en el norte de Sevilla, desde Hospital Macarena hasta San Pablo, y áreas importantes como Santa Justa y Gran Plaza. Compuesta por las estaciones: Hospital Macarena, Pio XII, Llanes, Carretera de Carmona, Pedro Romero, Tesalonica, Avenida Andalucia, Clemente Hidalgo - 1 de Mayo, Gran Plaza, Amate, La Plata, Maria Auxiliadora, Santa Justa, Kansas City, San Pablo, Montesierra, Luis Uruñuela, San Lazaro, Macarena, Capuchinos, Puerta de Carmona.
- **Comunidad 9 (color gris):** Agrupa estaciones centrales y de la Isla de la Cartuja, incluyendo áreas tecnológicas y comerciales. Compuesta por las estaciones: San Jacinto, Ronda de Triana, Torre Triana, Expo, Americo Vespucio, Cartuja, Ingenieros, Remo, Jose Diaz, Camas, Plaza de Armas, Duque, Cristo de Burgos.

El algoritmo de Spectral Clustering tiende a formar comunidades muy grandes en la zona central de Sevilla, abarcando amplias áreas, y comunidades más pequeñas en los extremos, correspondientes a los inicios o finales de las líneas. A diferencia del algoritmo de Louvain, cuyas comunidades identificadas tienen un tamaño más uniforme y límites geográficos más definidos. Las comunidades de los extremos son similares con ambas técnicas, pero Spectral Clustering agrupa muchas estaciones centrales en pocas comunidades, mientras que Louvain divide estas estaciones en un mayor número de comunidades más pequeñas. Spectral Clustering es útil para una visión amplia de la red, identificando grandes comunidades. Louvain es mejor para un análisis detallado con comunidades más pequeñas y definidas. Ambas metodologías ofrecen perspectivas valiosas y complementarias.

4.2.7 Estudio de la Robustez de la Red

Finalmente, vamos a realizar un estudio sobre cómo responde la red ante ataques intencionados y fallos aleatorios en algunos de los nodos. Estos fallos o ataques se simularán mediante el borrado de los nodos

afectados. En el caso de la red de metro de Sevilla, esto equivaldrá a simular que la estación de metro correspondiente no está disponible para su uso por cualquier motivo.

Para llevar a cabo esta simulación, utilizaremos las funciones 'calculate_K', 'ataques_intencionados' y 'fallos_aleatorios' que fueron descritas en el caso de estudio anterior sobre la red de vuelos europeos, donde también se analizó la robustez de dicha red. Estas funciones pueden reutilizarse sin necesidad de realizar ningún cambio. Sin embargo, será necesario modificar la función 'update', que nos permite generar una animación con la librería matplotlib.animation que muestra los diferentes estados de la red a medida que se van eliminando los nodos seleccionados.

En la función 'update', generaremos un nuevo mapa de la ciudad de Sevilla con Folium y añadiremos sobre él los objetos necesarios para representar el grafo después de borrar cada nodo seleccionado. Luego, cada uno de estos mapas se almacenará con un nombre de archivo único en formato HTML. Accederemos a cada archivo mediante el uso de 'os' y lo abriremos con Selenium. A continuación, esperaremos un tiempo para asegurarnos de que la imagen almacenada en el archivo se haya abierto y cargado correctamente, y tomaremos una captura de pantalla que se mostrará mediante Matplotlib junto al nombre del nodo eliminado.

```
# Guardar el mapa en un archivo HTML temporal con un nombre único
filename = f'mapa_metro_sevilla_{num}.html'
mapa_sevilla_frame.save(filename)

# Abrir el archivo HTML con Selenium
driver.get('file://' + os.path.abspath(filename))

# Esperar a que el mapa cargue completamente
time.sleep(5)
# Tomar una captura de pantalla
screenshot_filename = f'mapa_metro_sevilla_{num}.png'
driver.save_screenshot(screenshot_filename)

# Mostrar la imagen con Matplotlib
image = plt.imread(screenshot_filename)
ax.imshow(image)
ax.axis('off')  # Desactivar los ejes
ax.set_title(f"Número de nodos eliminados {len(nodes_removed[:num+1])}, Nodo eliminado: {nodes_removed[num]}")
```

Figura 4-40. Fragmento importante de la función update empleada en el estudio de la robustez de la red de metro.

Por último, creamos la animación mediante el uso de la librería matplotlib.animation, a partir de la función que hemos definido con el nombre de 'update'. Previamente, hemos configurado Selenium para utilizar la librería google-colab-selenium con '*driver = gs.Chrome()*'. Una vez creada la animación, es posible mostrarla mediante el uso de la librería IPython.display.

Una vez realizadas ambas simulaciones, graficamos los resultados obtenidos para analizar el comportamiento de la red en estos casos. Generamos dos gráficas, al igual que en el estudio previo: una que muestra la variación de la proporción de nodos que forman parte de la componente gigante en relación con el número de nodos eliminados, y otra que ilustra cómo cambia el valor de K en función de la eliminación de nodos.

Los resultados obtenidos en la simulación de los ataques intencionados muestran cómo la proporción de la componente gigante disminuye rápidamente a medida que se eliminan los nodos más conectados. Este comportamiento indica que la red es altamente vulnerable a ataques dirigidos a sus nodos más centrales. La rápida caída en la proporción de la componente gigante sugiere que estos nodos desempeñan un papel crucial en la conectividad global de la red. En la segunda de las gráficas, el valor de K disminuye abruptamente con la eliminación de los nodos. Este decrecimiento rápido muestra que la estructura de la red pierde su integridad rápidamente cuando se eliminan los nodos más conectados. Al llegar a un valor de K menor a 2, la red pierde su componente gigante, conforme al criterio de Molloy-Reed, eliminando solo 5 nodos.

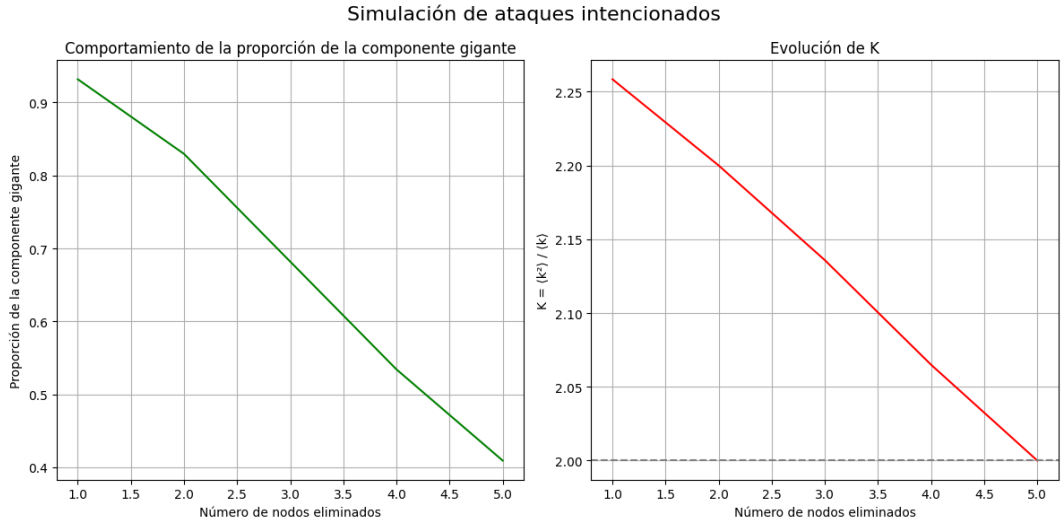


Figura 4-41. Resultados de la simulación de ataques intencionados en la red de metro.

Los resultados obtenidos en la simulación de fallos aleatorios en la red muestran cómo la proporción de la componente gigante disminuye de manera más gradual en comparación con los ataques intencionados. Esto indica que la red es más robusta frente a fallos aleatorios. La disminución más lenta en la proporción de la componente gigante sugiere que los nodos eliminados aleatoriamente no son tan cruciales para la conectividad de la red como los nodos con mayor grado. En la segunda gráfica que muestra la evolución de K, se puede apreciar cómo este valor también disminuye de manera más gradual en este caso. A pesar de la eliminación de múltiples nodos, la red mantiene una conectividad relativamente alta durante más tiempo. Esto resalta la resiliencia de la red frente a la eliminación aleatoria de nodos. Para que la componente gigante desaparezca y la red se fragmente completamente, es necesario eliminar una cantidad significativamente mayor de nodos en comparación con los ataques intencionados.

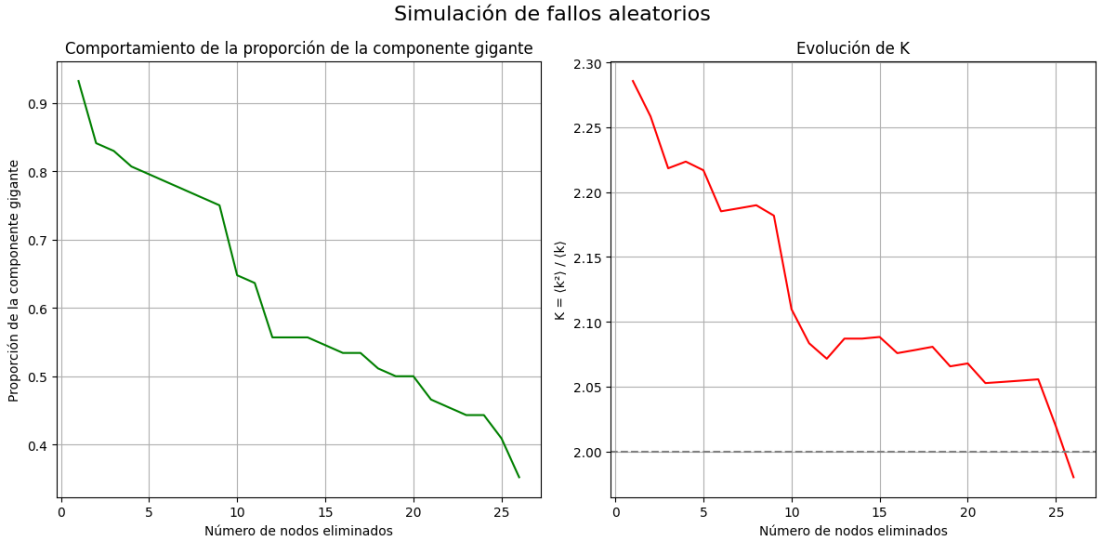


Figura 4-42. Resultados de la simulación de fallos aleatorios en la red de metro.

Algunos de las conclusiones que se pueden extraer de este estudio, es que la red es extremadamente vulnerable a ataques dirigidos a nodos con alta conectividad. La rápida disminución de la componente gigante y la caída abrupta de K indican que la eliminación de unos pocos nodos críticos puede fragmentar la red significativamente. Sin embargo, la red muestra una mayor resistencia a fallos aleatorios. Por lo que los nodos más conectados

juegan un papel crucial en la integridad estructural de la red y protegerlos es vital para mantener la funcionalidad y la conectividad del sistema de transporte.

En la siguiente figura se puede visualizar una comparativa entre los resultados obtenidos en ambos casos.

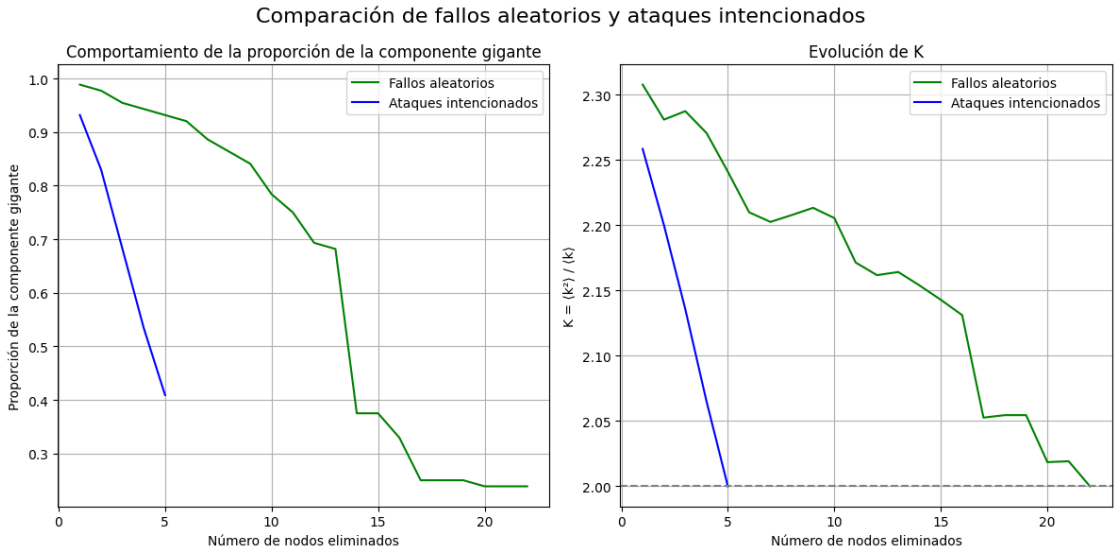


Figura 4-43. Comparativa de los resultados obtenidos sobre el estudio de la robustez de la red de metro.

5 CONCLUSIONES

En este trabajo, se han abordado los conceptos fundamentales de la teoría de grafos, incluyendo sus tipos y características, y se han explorado detalladamente varios algoritmos clave utilizados para el análisis de grafos: Betweenness Centrality, PageRank, Louvain y Spectral Clustering. A través de la implementación práctica de estos algoritmos, junto con la realización de un estudio sobre la robustez frente a fallos u ataques intencionados, se ha analizado la red de metro de Sevilla y una red de vuelos europeos. Estas herramientas nos han permitido analizar la estructura interna y funcionamiento de ambas redes.

Mediante el algoritmo de Betweenness Centrality, ha sido posible identificar nodos críticos cuya eliminación puede fragmentar significativamente la red, destacando su relevancia en el análisis de vulnerabilidades estructurales. Hemos identificado aquellas estaciones y aeropuertos que actúan como puntos críticos en la comunicación de estos medios de transporte cuyas redes se han analizado. El algoritmo de PageRank, conocido por su aplicación en la clasificación de páginas web, también se ha mostrado eficaz en la identificación de nodos influyentes en ambos tipos de redes analizadas. Aplicando PageRank, pudimos determinar qué estaciones del metro de Sevilla y qué aeropuertos europeos tienen mayor relevancia dentro de sus respectivas redes, lo cual es crucial para entender su funcionamiento y optimizar su gestión.

La detección de comunidades en grafos es una tarea fundamental en el análisis de redes complejas, ya que permite identificar grupos de nodos que están más densamente conectados entre sí que con el resto del grafo. En el contexto de este trabajo, la partición de grafos ha demostrado ser una herramienta valiosa para entender la estructura interna de la red de metro de Sevilla y la red de vuelos europeos, permitiendo identificar subconjuntos de estaciones y aeropuertos que están más fuertemente interconectados, lo cual es vital para la planificación y la mejora de la eficiencia operativa.

El algoritmo de Louvain se basa en la optimización de la modularidad, una medida que evalúa la densidad de enlaces dentro de las comunidades en comparación con los enlaces entre diferentes comunidades. Por otro lado, el Spectral Clustering utiliza técnicas de álgebra lineal para particionar el grafo en función de las propiedades de sus matrices de adyacencia o Laplaciana. Este enfoque se basa en la identificación de valores propios y vectores propios que reflejan la estructura global del grafo, proporcionando una partición que minimiza los cortes entre comunidades. En este trabajo, hemos mostrado cómo el algoritmo de Louvain produce resultados mucho más definidos que los del Spectral Clustering, convirtiéndose en una mejor opción a la hora de detectar clústeres.

Por último, el análisis de robustez llevado a cabo para ambas redes ha revelado que presentan vulnerabilidades similares, aunque a distinta escala, ya que la red de vuelos es mucho mayor. En ambos casos, se ha concluido que las redes son vulnerables ante ataques intencionados que se centran en los nodos más importantes de estas, llevando a una rápida fragmentación de la red que podría interrumpir significativamente su funcionamiento. Sin embargo, cuando se trata de fallos en nodos aleatorios, ambas redes de transporte presentan una resiliencia mucho mayor con una gran robustez.

En resumen, el trabajo ha proporcionado una base sólida para el análisis de grafos y ha demostrado la importancia y utilidad de los algoritmos estudiados en la comprensión y mejora de la robustez y funcionamiento de diferentes redes. Las técnicas y herramientas presentadas son fundamentales para el análisis y la optimización de redes de transporte, redes de Internet, redes biológicas y otros tipos de sistemas complejos.

Una posible línea futura para este trabajo es la integración de técnicas de inteligencia artificial y machine learning en el análisis de grafos. La aplicación de modelos de aprendizaje automático puede mejorar la predicción de la evolución de la red y la detección de patrones ocultos. Usar algoritmos de machine learning para identificar anomalías en las redes de transporte y optimizar su funcionamiento sería un avance significativo. Estas técnicas permitirán abordar problemas complejos y mejorar la toma de decisiones. Otra posibilidad sería la ampliación del análisis a diferentes tipos de redes, como redes sociales, biológicas o tecnológicas, para validar la aplicabilidad y generalización de los algoritmos estudiados. Esto permitirá comparar y contrastar el comportamiento de diferentes redes y explorar nuevas oportunidades de optimización y mejora.

REFERENCIAS

[1] Leonhard Euler, "Solutio problematis ad geometriam situs pertinentis," Acad. Sci. U. Petrop 8, 128-40 (en latín), 1736. [Online]. Available: http://eulerarchive.maa.org//docs/originals/E053.pdf.

[2] Héctor Hevia, "El problema de los Siete Puentes de Königsberg: Leonhard Euler y la Teoría de Grafos," Universidad Católica de Valparaíso, Valparaíso, Chile, 1996. [Online]. Available: https://www.revista-educacion-matematica.org.mx/descargas/Vol8/2/10Hevia.pdf.

[3] Weisstein, Eric W. "Four-Color Theorem." From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/Four-ColorTheorem.html.

[4] Tutte, W.T. "Graph Theory," Cambridge University Press, p. 30, ISBN 978-0-521-79489-3, 2001.

[5] Agustín Iñiguez, "Graph theory and its uses with 5 examples of real life problems," Xomnia, 2022. [Online]. Available: https://www.xomnia.com/post/graph-theory-and-its-uses-with-examples-of-real-life-problems/.

[6] Sergio Cruces, "Graph and Network Analytics." Curso de máster "Técnicas de inteligencia artificial en sistemas distribuidos". Universidad de Sevilla, 2022.

[7] Mark Needham and Amy E. Hodler, O. Autor, "Graph Algorithms. Practical Examples in Apache Spark and Neo4j," Mayo 2019.

[8] Albert-László Barabási, "Network Science," Julio 2016.

[9] Oscar Cordón García, "Redes y Sistemas Complejos," Cuarto Curso del Grado en Ingeniería Informática, Dpto. Ciencias de la Computación e Inteligencia Artificial, 2013-14. [Online]. Available: https://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/RedesSistemasComplejos/Tema05-1-ModelosdeRedesAleatorias-13-14.pdf.

[10] Natarajan Meghanathan, "Small World Networks," Jackson State University, Jackson, MS, 2015. [Online]. Available: https://www.jsums.edu/nmeghanathan/files/2015/08/CSC641-Fall2015-Module-6-Small-World-Networks-reduced.pdf.

[11] Mark Needham and Amy E. Hodler, O. "A Comprehensive Guide to Graph Algorithms", Neo4j, 2021. [Online]. Available: https://go.neo4j.com/rs/710-RRC-335/images/Comprehensive-Guide-to-Graph-Algorithms-in-Neo4j-ebook-EN-US.pdf.

[12] J. Bisht, "Betweenness Centrality (Centrality Measure)," 21 Julio 2022. [Online]. Available: https://www.geeksforgeeks.org/betweenness-centrality-centrality-measure/.

[13] Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," 1998. [Online]. Available: http://infolab.stanford.edu/~backrub/google.html.

[14] "Page Rank Algorithm and Implementation," 6 Septiembre 2022. [Online]. Available:

https://www.geeksforgeeks.org/page-rank-algorithm-implementation/.

[15] Sergio Cruces, "Community Detection and Graph Partitioning." Curso de máster "Técnicas de inteligencia artificial en sistemas distribuidos". Universidad de Sevilla, 2022.

[16] Vincent D Blondel, Jean-Loup Guillaume,Renaud Lambiotte and Etienne Lefebvre, "Fast unfolding of communities in large networks," Universit´e Catholique de Louvain, 2008. [Online]. Available: https://perso.uclouvain.be/vincent.blondel/publications/08BG.pdf.

[17] Fernando Carazo y Joaquín Amat, "Detección de comunidades en grafos y redes con python," Abril, 2023. [Online]. Available: https://cienciadedatos.net/documentos/pygml02-detecion-comunidades-grafos-redes-python.

[18] Jianbo Shi and Jitendra Malik, "IEEE Transactions on pattern analysis and machine intelligence", VOL. 22, NO. 8, AUGUST 2000. [Online]. Available: https://people.eecs.berkeley.edu/~malik/papers/SM-ncut.pdf.

[19] Sergio Cruces, "The K-means clustering algorithm." Curso de máster "Técnicas de inteligencia artificial en sistemas distribuidos". Universidad de Sevilla, 2022.