Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Analíticas de redes y grafos

Autor: Santiago Arenado Serrano

Tutor: Sergio Antonio Cruces Álvarez

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

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Analíticas de redes y grafos

Autor: Santiago Arenado Serrano

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, 2023

Autor:

Tutor:

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mis padres y mi novia.

Agradecimientos

Me gustaría expresar mi máximo agradecimiento a mis principales apoyos en lo que ha sido este duro camino, pero muy gratificante, a mis padres, que desde el primer día han estado apoyándome y concienciándome en que esto es una carrera de fondo y no un sprint, y confiar en que lo que se prepara bien, sale bien.

También a todos los profesores que me han ido dando clase todos estos años, y especialmente a mi tutor por acompañarme en este trabajo final, por todo los esfuerzos mostrados y apoyo en cada asignatura que estaba matriculado.

Santiago Arenado Serrano

Sevilla, 2023

Resumen

La teoría de grafos puede ser una herramienta muy útil para analizar y optimizar los sistemas de transporte, la arquitectura urbana y muchos otros aspectos de la planificación urbana, así como conocer las personas más influyentes en una red social o incluso en una red de empresas o trabajadores según sus conexiones. Por ejemplo, se pueden usar grafos para representar la red de carreteras de una ciudad, donde los nodos serían los cruces y las aristas serían las calles que las conectan y así poder analizar la distribución de las calles y la organización de ella gracias a los diferentes algoritmos de grafos.

El objetivo de este trabajo es el análisis de la estructura y conexiones de las calles de Sevilla a través de aplicaciones de metodologías de grafos para la propuesta de diferentes soluciones a partir de los datos obtenidos, en función del conexionado de sus carreteras y las relaciones que puedan haber entre ellas, y el análisis y demostración del funcionamiento del algoritmo de PageRank en una red de seguimiento entre un grupo de personas en función de sus conexiones entre ellos y la importancia que ha tenido en el buscador más importante hoy en día que es Google.

Para ello, y gracias a la librería NetworkX y OSMnX y siendo programado en Google Collab, se utilizarán las técnicas de análisis PageRank, Betweenness Centrality y detección de comunidades de Louvain, implementadas mediante el lenguaje de programación Python. En primer lugar, se recopilarán los datos de la ciudad de Sevilla, es decir, la interconexión de sus carreteras y se graficarán para ver su conexionado, y a continuación, se aplicarán los diferentes algoritmos de Betweenness Centrality y el de detección de comunidades de Louvain y su posterior grafo para ver cómo han analizado estos algoritmos la ciudad de Sevilla. También, se desarrollará otro ejemplo en el que se aplicará el algoritmo de PageRank y se demostrará el funcionamiento de éste en cuanto a las relaciones y conexiones de diferentes personas en una red social y cómo puede variar tanto las puntuaciones y la influencia de las personas en cuanto a sus seguidores.

Cómo resultado de estos algoritmos, comprenderemos mejor las interacciones y las relaciones de los grafos y redes, pues tienen implicaciones muy significativas en campos como el análisis de organización en una ciudad, gestión del tráfico o interacciones e influencia en redes sociales.

Abstract

Graph theory can be a very useful tool to analyse and optimise transport systems, urban architecture, and many other aspects of urban planning, as well as to know the most influential people in a social network or even in a network of companies or workers according to their connections. For example, graphs can be used to represent the road network of a city, where the nodes would be the crossroads and the edges would be the streets that connect them and thus be able to analyse the distribution of the streets and the organisation of it thanks to the different graph algorithms.

The aim of this work is the analysis of the structure and connections of the streets of Seville through applications of graph methodologies for the proposal of different solutions from the data obtained, depending on the connection of its roads and the relationships that may exist between them, and the analysis and demonstration of the operation of the PageRank algorithm in a tracking network between a group of people according to their connections between them and the importance it has had in the most important search engine today which is Google.

For this purpose, and thanks to the NetworkX and OSMnX libraries and being programmed in Google Collab, the PageRank, Betweenness Centrality and Louvain's community detection analysis techniques will be used, implemented using the Python programming language. First of all, the data of the city of Seville will be collected, i.e., the interconnection of its roads and plotted to see how they are connected, and then the different algorithms of Betweenness Centrality and Louvain's community detection algorithm and its subsequent graph will be applied to see how these algorithms have analyzed the city of Seville. Also, another example will be developed in which the PageRank algorithm will be applied, and it will be demonstrated how it works in terms of the relationships and connections of different people in a social network and how it can vary both the scores and the influence of people in terms of their followers.

As a result of these algorithms, we will better understand the interactions and relationships of graphs and networks, as they have very significant implications in fields such as organisational analysis in a city, traffic management or interactions and influence in social networks.

.

Índice

NOTACIÓN

A^*	Matriz conjugada
A^{-1}	Matriz inversa
$\mathcal{V}$	Índice conjunto de vértices
$\mathcal{E}$	Índice conjunto de aristas
i	Índice para indicar el nodo
Σ	Sumatorio
e	número e
L	Índice para indicar los lados
p_{uv}	Probabilidad entre nodo u y v
A^T	Traspuesta
X	Vector de influencias
$Pr(v_j)$	Puntuación PageRank nodo j
K_{vj}^{out}	Nodos salientes de j
D	Factor Dumping
P	Matriz de permutación
P^T	Matriz de permutación traspuesta
δ	Función Delta
Ci	Comunidad i
$\binom{m}{n}$	Número combinatorio

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Lo que se prepara bien, sale bien.
- Santi -

1.1 Breve explicación de la teoría de grafos y de su utilidad en diversos campos.

La teoría de grafos es una rama de las matemáticas que se enfoca en el estudio de las relaciones entre objetos. Un grafo se define como un conjunto de puntos que están conectados por líneas que representan relaciones entre ellos.

La teoría de grafos es utilizada en diversos campos debido a su capacidad para modelar y representar situaciones complejas en términos simples y visuales. Algunos de los campos en los que se utiliza la teoría de grafos son:

- Ciencias de la Computación: La teoría de grafos es fundamental para el diseño y análisis de algoritmos. Por ejemplo, los algoritmos de búsqueda, recorrido y ordenamiento de grafos son esenciales para la programación.
- Ingeniería: La teoría de grafos se utiliza para el diseño de redes de comunicación, de transporte, de suministro de energía, entre otras.
- Economía: La teoría de grafos se utiliza para modelar las interacciones entre empresas, mercados y consumidores.
- Biología: La teoría de grafos se utiliza para modelar redes de interacciones entre especies en un ecosistema, para el análisis de redes neuronales y para la identificación de patrones en datos genéticos.
- Física: La teoría de grafos se utiliza para modelar y entender la estructura de redes complejas como las redes de interacciones de partículas en un material.

En resumen, la teoría de grafos es una herramienta poderosa y versátil que se utiliza en una gran variedad de campos para analizar y modelar situaciones complejas de manera visual y sencilla.

1.2 Descripción de los algoritmos Pagerank y betweenness centrality y de su relevancia en la teoría de grafos

Pagerank y Betweenness Centrality son dos algoritmos importantes en la teoría de grafos para analizar la estructura y la importancia de los vértices en un grafo.

Pagerank es un algoritmo utilizado por el motor de búsqueda de Google para determinar la relevancia de una página web en los resultados de búsqueda. Pagerank funciona asignando una puntación a cada página web basado en el número de enlaces entrantes de otras páginas web y en la importancia de las páginas que contienen esos enlaces. Cuanto mayor sea la puntuación de Pagerank de una página web, mayor será su probabilidad de aparecer en los primeros resultados de búsqueda.

Betweenness Centrality es un algoritmo que mide la importancia de un vértice en un grafo en términos de cuántos caminos más cortos pasan a través de él. Un vértice con una alta Betweenness Centrality se considera importante porque actúa como un puente entre otros vértices en el grafo y puede influir en la comunicación y la transmisión de información en el mismo.

La relevancia de estos algoritmos en la teoría de grafos se debe a su capacidad para identificar y medir la importancia de los vértices en un grafo. Estos algoritmos son útiles para analizar la estructura de redes complejas y para identificar los nodos más importantes en términos de la transmisión de información y la influencia en el grafo. Además, estos algoritmos tienen aplicaciones en diversas áreas, como la optimización de motores de búsqueda, el análisis de redes sociales, la identificación de nodos críticos en redes de transporte y la detección de comunidades en redes complejas.

2 FUNDAMENTOS TEÓRICOS DE LA TEORÍA DE GRAFOS

2.1 Introducción histórica de los grafos

La teoría de grafos es una rama de la topología que se ocupa de objetos matemáticos engañosamente simples llamados grafo. Se originó en el siglo XVIII y se convirtió en una de las primeras formas de topología. La historia comienza con un rompecabezas cuya solución no se puede clasificar en ninguna rama conocida de las matemáticas en ese momento. Gradualmente, en el siglo XIX, más curiosidades matemáticas comenzaron a caer en la misma categoría que los grafos. Para el siglo XX, la teoría de grafos se había convertido en un campo riguroso, con innumerables aplicaciones no solo en las matemáticas en sí, sino en muchas disciplinas científicas. En este trabajo se revisan algunos de los aspectos más destacados de la historia de la teoría de grafos, partiendo de sus inicios y continuando de forma lineal. A lo largo de la exposición se resuelven diferentes problemas que surgen en la teoría de grafos, como el problema de los puentes de Königsberg, la enumeración de árboles, el juego de Icos, la conjetura de los cuatro colores, el teorema de Kuratowski y el problema del viajante de comercio. En conjunto, estas instantáneas y las historias de las personas implicadas ofrecerán una comprensión básica de la evolución del campo de la teoría de grafos.

En general, se cree que el concepto de grafo nació en 1736. Leonhard Euler, a la edad de 29 años, ya se había establecido como matemático dos años antes al resolver el rompecabezas de Basilea. Cuando publicó un artículo en Commentarii Academiae Scientiarum Imperialis Petropolitanae titulado "Solución del problema de la geometría de posición", fue contratado por la Academia Imperial de Ciencias de St. En él, ofreció una solución al problema del puente de Königsberg. Hoy en día, este artículo se considera el primer trabajo en grafos y topología.

Königsberg, ahora conocida como Kaliningrado, fue la capital de Prusia Oriental. El río Pregel (hoy Pregel) fluye a través de la ciudad, dividiéndola en cuatro distritos, que están conectados por siete puentes. Dicen que a la gente de Königsberg le gustaba caminar cerca de los puentes. Sin embargo, se preguntaron si les sería posible caminar cruzando un puente a la vez. Nadie ha tenido éxito o probado que no funciona.

La respuesta dada por Euler no es la que daría un grafólogo moderno. No menciona términos como grafo, arista o vértice. Sin embargo, la estructura lógica del programa es casi la misma, aunque no es la misma que la de los estándares actuales. El primer paso de Euler fue eliminar los cuatro puntos de tierra como zonas independientes A, B, C, D y los siete puentes como puntos a, b, c, d, e, f y g. La siguiente imagen es un diagrama de la

problemática a resolver de los puentes de Königsberg.

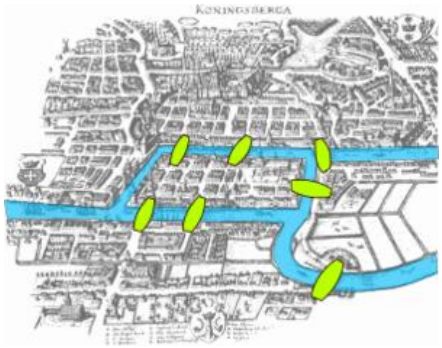


Figura 2-1. Problema de los puentes de Königsberg.

Un paseo podría representarse entonces como una secuencia de letras. Por ejemplo, ABD representaría cruzar un puente de la zona de tierra A, a la zona de tierra B y luego a la zona de tierra D. No especificaba qué puente se utilizaba si había varios puentes disponibles entre dos partes de tierra cualesquiera. Euler examinó primero el caso de una sola zona de tierra y observó que, si una zona de tierra M tiene un número impar de n puentes conectados a ella, cada uno de los cuales debe cruzarse una vez, entonces M debe aparecer en la secuencia final (n+1) veces. Así, en el problema de Königsberg, como cada 2 zonas de tierra tienen un número impar de puentes que conectan con ella, podemos aplicar la idea anterior para contar cuántas veces debe aparecer cada zona de tierra en el recorrido completo. Como A conecta cinco puentes, para cruzarlos todos debe haber la letra A $\frac{5+1}{2}$ = 3 veces. Asimismo, las letras B, C y D deben aparecer dos veces en el curso. Por lo tanto, todo el camino debe contener 3 + 2 + 2 + 2 = 9 letras. Sin embargo, si se van a cruzar siete puentes una vez, la secuencia resultante debe contener solo ocho letras. Se encontró que el conflicto hacía imposible tal migración.

Siguiendo el mismo argumento, Euler continúa considerando el tema del número de zonas y puentes. Resuma sus resultados en los siguientes tres puntos:

1. Si hay más de dos zonas de tierra y hay un número de puentes impar, entonces tal viaje es imposible.
2. Si, por el contrario, el número de puentes es impar para exactamente dos zonas de tierra, entonces el viaje es posible si parte de una de estas dos zonas.
3. Finalmente, si no hay zonas a las que conduzcan un número impar de puentes, el viaje puede realizarse partiendo de cualquier zona.

Hoy, el problema del puente de Königsberg va un paso más allá del diagrama de Euler y se convierte en lo que ahora se conoce como un grafo. Las zonas de tierra se representan mediante puntos llamados vértices, mientras que los puentes están representados por líneas llamadas aristas. Por lo tanto, según [1], el problema se puede representar de la siguiente manera:

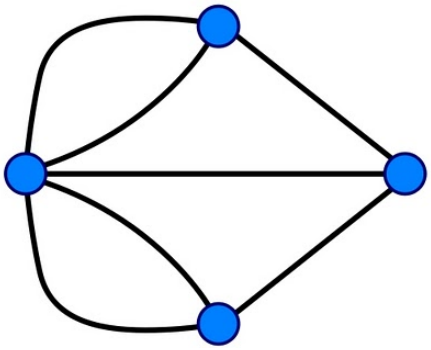


Figura 2-2. Grafo de los puentes de Königsberg.

2.2 Definición formal de un grafo

En esta sección se describen los conceptos básicos necesarios para comprender el estudio de problemas estructurados en forma de grafos.

La teoría de grafos (Graph Theory) es la disciplina encargada de estudiar las estructuras formadas por vértices, también conocidos como nodos, que representan los elementos u objetos que se relacionan entre sí en el grafo, y aristas, que son las conexiones de los vértices, y representan las relaciones que hay dentro de un grafo. Un grafo *G* se define de forma matemática por tuplas, como se indica en [2],

$$G = (\mathcal{V}, \mathcal{E}) \tag{2,1}$$

siendo esta su nomenclatura. Un grafo se compone de un conjunto de vértices

$$\mathcal{V} = \{A, B, C\} \tag{2,2}$$

y $\mathcal{E}$ como conjunto de aristas para la conexión de los vértices, por lo que un grafo está compuesto en su totalidad por *n* vértices y *m* aristas. En la siguiente figura se ve un ejemplo de grafo, que está formado por X vértices y X aristas.

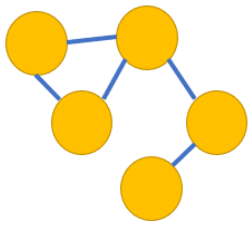


Figura 2-3. Ejemplo de grafo.

2.3 Tipos de grafos

Existen varias propiedades básicas de los grafos, que en función de ellas, se elegirán posteriormente para realizar los ejemplos propuestos, a continuación se explican. Primero, y una de las características más fundamentales, es la orientación del grafo, ya que puede ser un grafo dirigido o no dirigido, en los grafos no dirigidos las relaciones entre nodos no tienen dirección, es decir, $(\mathcal{V}_i\,,\mathcal{V}_j)$ *o* $(\mathcal{V}_i\,,\mathcal{V}_j)$ indican la conexión de dos nodos sin dirección alguna, es decir, la conexión entre el nodo *i* y el nodo *j* [3].

En un grafo dirigido, las relaciones son no recíprocas, por lo que el orden de los nodos indica la orientación que va a tomar esa arista, en este caso $(\mathcal{V}_i\,,\mathcal{V}_j)$, la arista iría en la dirección de *i* a *j*. Un ejemplo de estos dos tipos de grafos es, para el grafo no dirigido, el análisis en las amistades en las que desea asumir que la relación es mutua, y el grafo dirigido sería útil para el análisis de las carreteras de una ciudad pues la mayoría tienen un solo sentido, como se indica en [4].

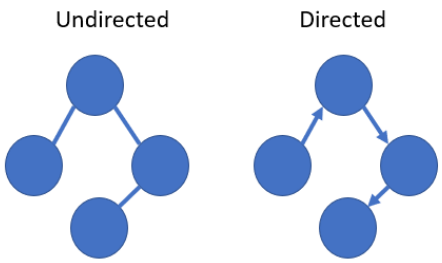


Figura 2-4. Grafo dirigido y no dirigido.

Fuente: elaboración propia

La segunda propiedad es la ciclicidad, un grafo se considera cíclico si existe al menos un ciclo en él, es decir, una secuencia de vértices que comienza y acaba en el mismo vértice, si un grafo no tiene ciclos se considera acíclico. Para determinar si un grafo es cíclico o no, se pueden utilizar diferentes métodos para recorrerlo, como por ejemplo para un grafo no dirigido se puede usar el algoritmo de búsqueda en profundidad (DFS) para recorrer todos los vértices y aristas, y si se da el caso de que ese vértice ya ha sido visitado y no es el padre del vértice actual, significa que hay un ciclo en ese grafo. Si el grafo tiene un ciclo, no es posible hacer este ordenamiento topológico y, por lo tanto, se puede concluir que el grafo es cíclico [5].

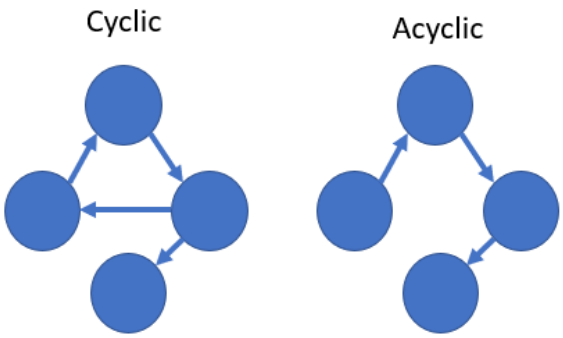


Figura 2-5. Grafo cíclico y acíclico.

Fuente: elaboración propia

En tercer lugar, están los grafos según su ponderación, los grafos ponderados asignan valores (o pesos) a los nodos o a sus relaciones, como puede ser por ejemplo el kilometraje entre dos ciudades, y en el caso de no ponderado no tendrán ningún peso. Para analizar el caso de un grafo no ponderado entre dos nodos se utilizará un algoritmo para ver cuál es el camino más corto en función de los saltos, y si fuese ponderado, sería más importante analizar en función del coste total en vez de saltos, véase [4].

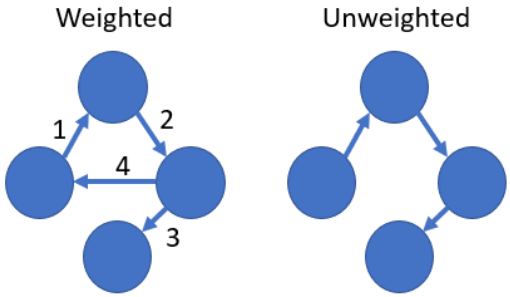


Figura 2-6. Grafo ponderado y no ponderado.

Fuente: elaboración propia

También veremos cómo esto es relevante para los algoritmos de detección de comunidades, especialmente los componentes débil y fuertemente conectados.

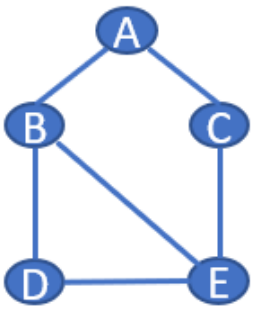
2.4 Conceptos básicos de la teoría de grafos

Sea $G = (\mathcal{V}, \mathcal{E})$ un grafo (p, q). Consideremos la matriz

$$A = [A_{ij}], \tag{2,3}$$

con dimensiones $p \times p$, donde cada fila (y cada columna) de A corresponde a un vértice distinto de V. La matriz de adyacencia G representa la relación entre los nodos o vértices del grafo, que, como se explicó anteriormente, es un grupo de nodos conectados por aristas. Esta matriz se utiliza para almacenar de una forma eficiente esta información pues ofrece una representación compacta y permite acceder rápidamente a la información de la adyacencia entre los nodos.

La matriz de adyacencia se representa como una matriz cuadrada donde las filas y columnas representan los vértices del grafo. En el caso que una arista conecta con dos nodos, el valor correspondiente en la matriz será un 1 o cualquier otro valor, pero no nulo. Este valor diferente a 1 dependerá del contexto en el que estemos hablando. En caso contrario, si un vértice no está conectado a otro por una arista, el valor que aparecerá en la matriz es un 0. En el caso de los grafos ponderados, los valores de la matriz podrían ser los pesos. A continuación, se muestra cómo sería un grafo con su correspondiente matriz de adyacencia.



$$A_{graph} = \begin{array}{c} \begin{matrix} A & B & C & D & E \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \\ E \end{matrix}$$

Figura 2-7. Grafo no dirigido y matriz de adyacencia.

Fuente: elaboración propia

En un grafo dirigido, la matriz de adyacencia es simétrica tal y como se puede observar en la matriz A, y esto se debe a que no hay distinción entre los nodos de origen y destino de las aristas pues no hay una dirección específica entre ellas. Al ser simétrica, la diagonal de esta matriz serán todos ceros pues un nodo no puede estar conectado consigo mismo en un grafo no dirigido.

Una vez que tenemos la matriz de adyacencia, podemos realizar diversas operaciones y análisis en nuestro grafo de una manera eficiente como, por ejemplo, determinar si existe una arista entre dos vértices, obtener los vecinos de un vértice ya que, viendo la fila o columna correspondiente de un nodo específico, los valores que estén a 1 están conectados a este por lo que sería su vértice.

Otras de las operaciones fundamentales de la matriz de adyacencia, es que se puede utilizar para aplicar los dos principales algoritmos de búsqueda y recorrido del grafo de una forma ordenada, aplicados para encontrar el camino entre dos nodos, como son los algoritmos de búsquedas de caminos o pathfindings, caracterizados por ser algoritmos en el cual su principal función es encontrar los caminos más cortos entre dos nodos en función del menor número den saltos o menor peso en su camino, y los principales algporitms para ello son el de búsqueda en profundidad o Depth First Search (DFS) y el de búsqueda en anchura o Breath First Search (BFS), véase [6] y [7].

El algoritmo de búsqueda en profundidad es un algoritmo para recorrer un grafo y su funcionamiento es el siguiente. Comienza en un nodo inicial y se va expandiendo por cada uno de los nodos que va encontrándose de forma recurrente, es decir, de un nodo mayor a uno menor, y cuando ha realizado todo el camino y no encuentra más nodos, realiza el mismo proceso, pero con cada vecino del nodo. En la siguiente figura, como bien se indica en [7], se puede ver cómo se va realizando la búsqueda de una forma recurrente. Uno de los ejemplos más claros para los que se utilizaría este algoritmo es para obtener la solución de un laberinto.

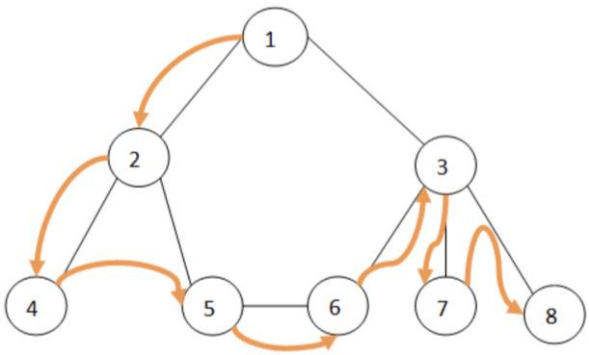


Figura 2-8: Algoritmo de búsqueda en profundidad

Fuente: [7]

En cambio, en el algoritmo de búsqueda en anchura, que también es un algoritmo para recorrer los nodos de un grafo, comienza seleccionando un nodo cualquiera del grafo, al que se le llama raíz y lo indicamos con un 0 y así cada salto a los vecinos se incrementará en 1 el valor. A continuación, y se exploran todos los vecinos de este nodo y se les pone un 1, y a continuación, se buscan los vecinos adyacentes de este nodo vecino del raíz hasta haber recorrido el grafo de forma completa y haber encontrado el camino más corto de i a j. En el caso que no hubiese camino de i a j, la distancia entre esos dos nodos es infinita.

Uno de los ejemplos más claros de uso de este algoritmo es para obtener el camino más corto entre dos lugares.

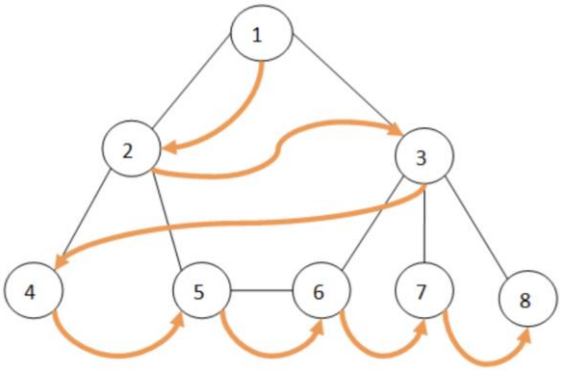


Figura 2-9: Algoritmo de búsqueda en anchura

Fuente: [7]

Es importante entender el uso de la matriz de adyacencia ya que proporciona una representación eficiente y fácil de entender de las conexiones en un grafo, permite realizar operaciones básicas y algoritmos avanzados en grafos de manera eficiente, lo que es esencial en muchos campos, como la teoría de redes, la planificación de rutas, o la optimización.

El grado de un nodo se refiere al número de aristas que inciden en ese nodo en el caso que sea un grafo no dirigido, y está definido por la fórmula

$$\sum_j k_j = \sum_{i,j} A_{i,j} = 2L, \tag{2,4}$$

en el cual, la variable L representa al número de aristas y se multiplica por dos ya que al ser no dirigido el número de extremos es doble, por ello se calcula como el número de enlaces por 2, y para el cálculo del grado medio del grafo haremos uso de la fórmula siguiente,

$$< k > = \frac{2L}{N}, \tag{2,5}$$

y en el caso de un grafo dirigido, el grado se diferencia en dos tipos, primero en grado entrante, que es el número de aristas que inciden en ese nodo, y en grado saliente que es el número de aristas que salen de ese nodo y la N representa el número total de enlaces que hay en dicho grafo. En el caso de las aristas, el grado es el número de nodos que están conectados por una arista, que en el caso de un grado no dirigido será 2 y en el de uno dirigido puede ser 1 o 2, pues puede que haya conexión mutua entre dos nodos, la siguiente fórmula demuestra este cálculo en el cual ya no está el 2 multiplicando previamente pues un entre dos nodos no tienen por qué incidir mutuamente,

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

Por último, se explica la densidad de un grafo, que es una medida que indica qué tan conectado está un grafo, es decir, cuántas aristas hay en relación con el número total de nodos. Se calcula dividiendo el número total de aristas en el grafo por el número máximo de aristas posibles en un grafo completo. En un grafo no dirigido, la densidad está entre 0 y 1, mientras que, en un grafo dirigido, la densidad puede ser mayor a 1 debido a la posibilidad de aristas dirigidas entre nodos. Un grafo es más denso cuanto mayor es su densidad, véase [3], la siguiente fórmula demuestra cómo calcular la densidad de un grafo de orden N,

$$D = \frac{size}{MaxSize} = \frac{L}{\binom{N}{2}} \tag{2,7}$$

Siendo MaxSize el máximo número de nodos por N nodos, calculado de la siguiente forma,

$$MaxSize = \binom{\mathrm{N}}{2} = \frac{N(N-1)}{2}. \tag{2,8}$$

2.5 Tipos de grafos según estructura

En este apartado se explicará los principales tipos de grafos existentes que podemos encontrarnos según su estructura y su relación e interconexión de nodos.

En primer lugar, está el grafo aleatorio o también llamado Erdős-Rényi, es un tipo de modelo de grafo utilizado para representar redes aleatorias. Fue propuesto por los matemáticos Paul Erdős y Alfréd Rényi en la década de 1960.

En este modelo, se considera un conjunto de N nodos, y cada par de nodos tiene una probabilidad p de estar conectados por una arista. La generación del grafo aleatorio se realiza seleccionando cada par de nodos y conectándolos con probabilidad p de forma independiente.

La propiedad clave de los grafos de Erdös-Rényi es que la presencia o ausencia de cada arista se determina de manera completamente independiente de las demás aristas. Esto da como resultado una distribución de grado binomial, lo que significa que la probabilidad de que un nodo tenga k conexiones sigue una distribución binomial. Este tipo de grafo es de gran relevancia a la hora de encontrar la modularidad en un grafo, ya que pide la calidad de la partición de los nodos en comunidades o clusters y la relación con este tipo de grafos, es que no suelen tener una modularidad fuerte debido a la naturaleza aleatoria de las conexiones, lo más probable es que se distribuyan de forma uniforme lo que dificulta en gran medida la identificación de comunidades bien definidas, la modularidad tiende a ser baja a menos que la densidad de aristas sea muy alta. Esto se debe a que la modularidad se compara con una configuración nula esperada, y en el modelo de Erdős-Rényi, la configuración nula esperada es un grafo completamente aleatorio sin estructura modular. La probabilidad de conexión está normalizada por la siguiente fórmula

$$p_{uv} = \frac{k_u k_v}{2L-1}, \tag{2,9}$$

y en el caso de superar ese número, significa que hay indicios de comunidad, a continuación, se muestra un tipo de grafo aleatorio.

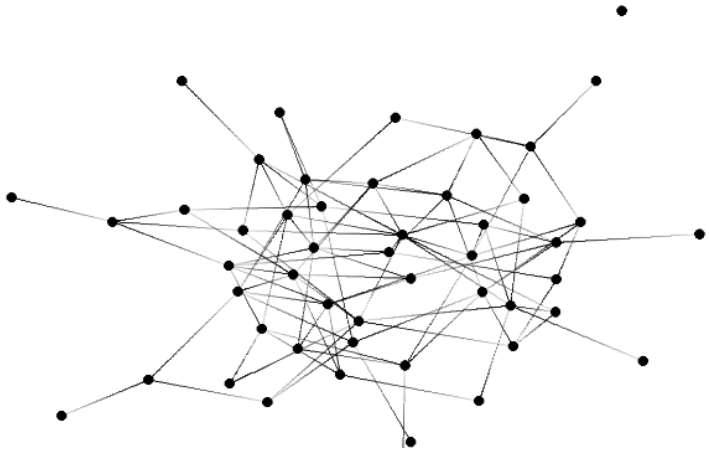


Figura 2-10: Grafo aleatorio

Fuente: Wikipedia

El siguiente tipo de grafo es el completo, también conocido como grafo completamente conectado, es un tipo de grafo en el cual todos los pares de nodos están conectados por una arista. En otras palabras, cada nodo del grafo completo está directamente conectado a todos los demás nodos.

Un grafo completo con N nodos tiene exactamente

$$\frac{N*(N-1)}{2} \tag{2,10}$$

aristas, ya que cada nodo se conecta con N-1 nodos distintos, y se divide por 2 para evitar contar las aristas duplicadas. Esta propiedad hace que los grafos completos sean bastante densos en términos de conexiones entre los nodos.

El grafo completo se denota comúnmente como K_n donde N representa el número de nodos en el grafo.

Una característica importante de los grafos completos es que son simétricos, lo que significa que, si el nodo A está conectado al nodo B, entonces el nodo B también está conectado al nodo A. Esto se debe a que las aristas en un grafo completo no tienen dirección, lo que implica una relación simétrica entre los nodos.

Los grafos completos son útiles en diversas áreas, como la teoría de grafos, o la geometría computacional.

Sin embargo, a medida que el número de nodos aumenta, la cantidad de aristas en un grafo completo crece rápidamente. Esto puede hacer que los grafos completos sean inmanejables en términos de almacenamiento y cálculo, especialmente cuando el número de nodos es grande. Por lo tanto, en la práctica, se utilizan otros tipos de grafos más esparcidos o con estructuras específicas para representar relaciones entre nodos en sistemas más complejos. A continuación, se muestra un ejemplo de grafo completo.

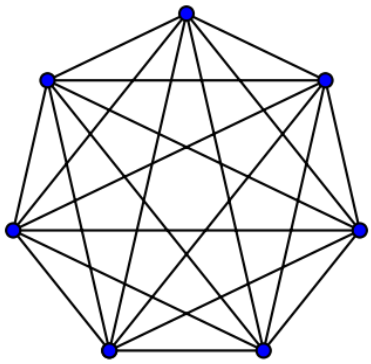


Figura 2-11: Grafo completo

Fuente: Wikipedia

Un grafo clique es un tipo especial de grafo completo en el cual todos los nodos están conectados directamente entre sí. Es decir, en un grafo clique, cada par de nodos está conectado por una arista. El grafo clique se puede definir como un subgrafo completo de un grafo. Esto significa que, si tomamos un subconjunto de nodos de un grafo completo y los conectamos entre sí, obtenemos un grafo clique.

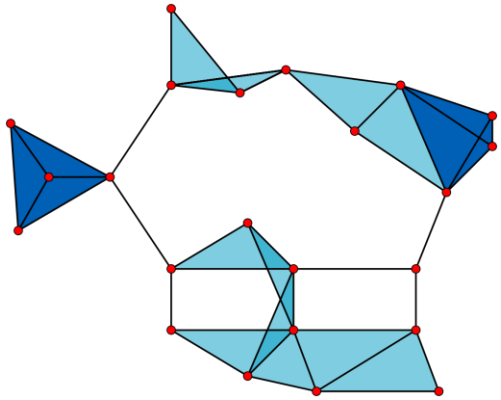


Figura 2-12: Grafo clique

Fuente: Wikipedia

Un grafo DAG (Grafo Acíclico Dirigido, por sus siglas en inglés) es un tipo de grafo dirigido en el cual no existen ciclos dirigidos, es decir, no se puede seguir una secuencia de aristas y llegar de vuelta al mismo nodo siguiendo la dirección de las aristas. En otras palabras, un DAG no tiene ciclos dirigidos.

En un grafo DAG, las aristas tienen una dirección que indica la relación de precedencia o dependencia entre los nodos. Esto significa que hay un orden parcial definido en los nodos, donde ciertos nodos deben preceder a otros en función de las aristas dirigidas.

Los grafos DAG son utilizados para representar estructuras de datos o relaciones en las que se establece una jerarquía o dependencia entre los elementos. Algunos ejemplos comunes de aplicaciones de los grafos DAG incluyen:

1. Representación de dependencias de tareas: Un grafo DAG puede utilizarse para modelar un conjunto de tareas donde ciertas tareas dependen de otras y deben realizarse en un orden específico.
2. Análisis de dependencias de software: En la ingeniería de software, los grafos DAG se utilizan para representar las dependencias entre módulos, bibliotecas o componentes de un sistema, lo que ayuda a entender las relaciones y facilita el análisis de impacto de cambios o la resolución de conflictos.
3. Programación de proyectos: Los grafos DAG se utilizan en la gestión de proyectos para representar las

tareas y actividades, así como las restricciones y dependencias entre ellas. Esto permite una programación eficiente y la identificación de caminos críticos en el proyecto.

Una propiedad importante de los grafos DAG es que se pueden ordenar topológicamente. El orden topológico de un DAG es una asignación lineal de los nodos de tal manera que, para cada arista dirigida, el nodo de origen precede al nodo de destino. El orden topológico es muy útil para resolver problemas de planificación, optimización y secuenciación en contextos donde existen dependencias [3].

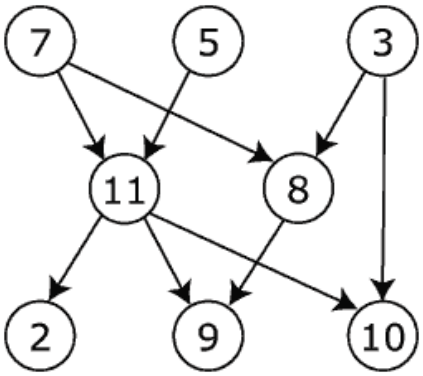


Figura 2-13: Grafo DAG

Fuente: Wikipedia

Un grafo tree (árbol) es un tipo especial de grafo dirigido o no dirigido que consiste en un conjunto de nodos conectados por aristas, donde se cumple la siguiente condición, hay exactamente un camino único entre cualquier par de nodos en el grafo, es decir, un grafo tree es un grafo conectado en el cual no existen ciclos. Cada nodo del árbol, excepto uno llamado "raíz", tiene un único nodo padre, mientras que el nodo raíz no tiene un padre. Además, cada nodo puede tener múltiples hijos.

Las características clave de un grafo tree son la conectividad, pues todos los nodos están conectados a través de una ruta única desde cualquier nodo hasta la raíz. Esto significa que se puede llegar a cualquier nodo desde cualquier otro nodo del árbol. La jerarquía ya que la estructura del árbol impone una relación de jerarquía entre los nodos, es decir, los nodos hijos están directamente vinculados a sus respectivos nodos padres, formando una estructura de ramificación y la ausencia de ciclos, en un grafo tree, no existen ciclos, lo que implica que no se puede seguir una secuencia de aristas y volver al mismo nodo.

Es importante destacar que un grafo tree es un caso especial de grafo acíclico. Si se trata de un grafo dirigido acíclico (DAG) en el que existe un único nodo fuente, también se le conoce como árbol enraizado o rooted tree.

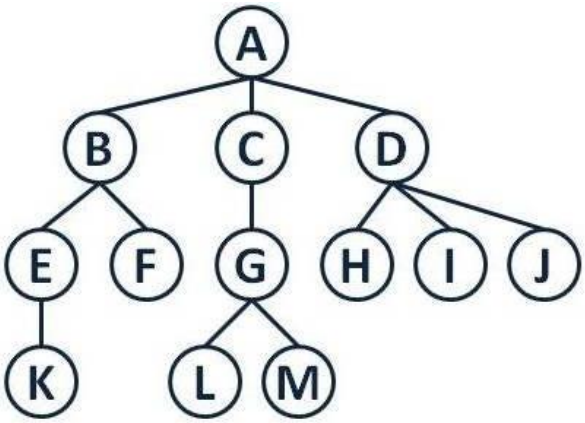


Figura 2-14: Grafo Tree

Fuente: Wikipedia

Por último, y muy importante debido a su relación con el algoritmo que posteriormente se explicará, el de Page Rank, están los grafos bipartitos.

Un grafo bipartito es un tipo de grafo en el cual los nodos pueden ser divididos en dos conjuntos distintos, de manera que todas las aristas del grafo conectan nodos de un conjunto con nodos del otro conjunto. En otras palabras, no hay aristas que conecten nodos dentro del mismo conjunto.

Formalmente, un grafo bipartito G = (U, V) se define como un grafo donde el conjunto de nodos completamente distintos como son U y V, todos los nodos de U están conectados a V mediante algún nodo.

Un grafo bipartito se puede representar gráficamente como dos conjuntos de nodos dispuestos en columnas, donde las aristas atraviesan las columnas, conectando nodos de una columna con nodos de la otra columna. Esta estructura de partición en dos conjuntos es lo que hace que el grafo sea bipartito.

Los grafos bipartitos se utilizan para modelar relaciones entre dos conjuntos de entidades. Por ejemplo, en redes sociales, se pueden usar para representar las conexiones entre usuarios y eventos o entre usuarios y grupos.

En los grafos bipartitos existen las denominadas proyecciones que son subgrafos que se obtienen al "proyectar" las aristas del grafo original en alguno de los conjuntos de nodos. Específicamente, hay dos tipos de proyecciones en un grafo bipartito: la proyección en el conjunto de nodos izquierdo y la proyección en el conjunto de nodos derecho.

La proyección en el conjunto de nodos izquierdo se obtiene al eliminar todas las aristas que conectan nodos dentro del mismo conjunto de nodos izquierdo, y conservando únicamente las aristas que conectan nodos de distintos conjuntos, es decir, las que van desde nodos izquierdos hacia nodos derechos, así mismo pasa con las proyección en el conjunto de nodos derecho.

Al proyectar un grafo bipartito, se obtiene un nuevo grafo que muestra las conexiones y relaciones entre los nodos de un conjunto específico, sin tener en cuenta las conexiones dentro del mismo conjunto. Esto puede ser útil para analizar y comprender las relaciones entre los elementos de un conjunto en particular [3].

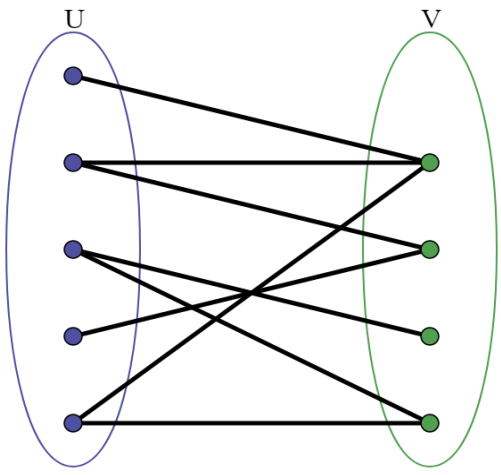


Figura 2-15: Grafo bipartito

Fuente: Wikipedia

3 ALGORITMOS DE ESTUDIO

Los algoritmos de grafos se utilizan para calcular métricas de grafos, nodos o relaciones. Pueden proporcionar información sobre entidades relevantes en el grafo o estructuras como comunidades.

Muchos algoritmos de grafos son algoritmos iterativos que recorren con frecuencia el grafo para realizar el cálculo mediante paseos aleatorios, búsquedas de amplitud o profundidad, o concordancia de patrones.

Afortunadamente, existen algoritmos optimizados que utilizan determinadas estructuras del grafo, memorizan partes ya exploradas y paralelizan las operaciones. Siempre que ha sido posible, hemos aplicado estas optimizaciones.

3.1 Algoritmo de Betweenness Centrality

El algoritmo Betweenness Centrality es la puntuación que calcula el camino más corto (ponderado) que hay entre dos nodos de un grafo conectado pasando por el nodo en el que estamos obteniendo su puntuación, utilizando el algoritmo de búsqueda breadth-first. A cada nodo se le puntúa basándose en el número de caminos más cortos, es decir, el número de caminos más cortos entre cada par de nodo, que pasan por ese nodo. Los nodos que se encuentran con más frecuencia en estos caminos más cortos tendrán una puntuación de centralidad entre nodos más alta.

Gracias al algoritmo de Betweenness Centrality, es posible detectar cómo de influyente es un nodo sobre el flujo de información en un grafo, y uno de los usos más comunes es para encontrar el nodo que sirve de puente entre dos partes de un grafo, identificar cuellos de botella o detectar las vulnerabilidades de un grafo. En el caso que estos nodos que actúan como puente, fuesen eliminados, se desconectaría parte del grafo, de ahí la importancia de este algoritmo [4].

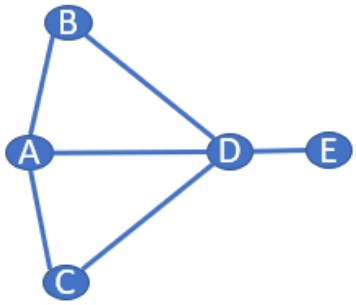
El funcionamiento de este algoritmo funciona de la forma siguiente:

- Primero, para cada nodo, encuentra los caminos más cortos hacia todos los demás nodos.
- En el segundo paso, para cada nodo se divide el número de shortest paths por cada nodo origen y destino que pasan por dicho nodo entre la suma de todos los caminos más cortos de todos los nodos.
- Un vez que se ha hecho el paso anterior, los nodos identificados como puente o cuello de botella son aquellos con mayor puntuación de centralidad de interrelación.

La fórmula para hallar esta puntuación es:

$$B(u) = \sum_{s \neq u \neq t} \frac{n\_shortest\_path_u(s,t)}{Total\_n\_shortest\_path(s,t)}, \tag{3,1}$$

donde $n_{\_shortest\_path_u}(s,t)$ es el número de caminos más cortos desde un nodo source (S) hacia otro target (T) pasando por U, y posteriormente lo normalizamos por el número total de caminos que hay entre el origen y el destino, que eso es lo que representa $Total\_n_{\_shortest\_path}(s,t)$ [3]. A continuación, se demuestra este algoritmo con un ejemplo.



Cálculo nodo D		
Parejas con caminos más cortos en nodo D	Total de posibles caminos más cortos en D	% del total a través de D (1/Total)
A,E	1	1
B,E	1	1
C,E	1	1
B,C	2	0.5
Betweenness Score		3.5

Figura 3-1. Cálculo de puntuación BC nodo D.

En este caso, el primer paso como aplica la fórmula, se calcula el número total de caminos más cortos que pasan por D como indica la tabla y posteriormente dividimos inversamente ese número de caminos más cortos posibles y sumamos todos los valores obtenidos por cada par de nodos y obtenemos la puntuación del algoritmo de Betweenness Centrality en ese nodo.

Unos de los mayores inconvenientes de utilizar este algoritmo suceden cuando el grafo es de gran tamaño, pues el tiempo que conllevaría en obtener las puntuaciones de los nodos es muy alto y podrían aparecer errores, es por ello, que se usan diferentes algoritmos para obtener una aproximación para que se ejecute con mayor velocidad y también poder obtener incluso, una información más útil.

Este algoritmo es el llamado "RA-Brandes", es el algoritmo más conocido para obtener una aproximación de la puntuación de la centralidad de interrelación ya que, en vez de calcular los caminos más cortos entre todos los nodos, sólo la obtiene de un conjunto de nodos, y la elección de estos conjuntos de nodos se hace mediante dos estrategias existentes que son la Random y la de Degree.

El método random de selección de nodos en el algoritmo de Betweenness Centrality es una técnica que se utiliza para estimar la centralidad de intermediación de los nodos en un grafo. La centralidad de intermediación mide la importancia de un nodo como puente en las rutas más cortas entre otros nodos en el grafo.

El proceso de selección de nodos aleatorios implica lo siguiente:

1. Selección de un conjunto de nodos de manera aleatoria del grafo. El número de nodos seleccionados se determina según las necesidades y la capacidad computacional disponible.
2. Para cada par de nodos seleccionados, se calculan las rutas más cortas que los conectan y se contabiliza la presencia de cada nodo en esas rutas.
3. Se repite el proceso un número suficiente de veces para obtener una muestra representativa de los caminos más cortos en el grafo.
4. Finalmente, se promedian los resultados de todas las ejecuciones para obtener una estimación de la centralidad de intermediación de los nodos.

Es importante tener en cuenta que la estimación obtenida utilizando el método random de selección de nodos puede tener un cierto grado de error y depende de la calidad y el tamaño de la muestra seleccionada. Cuanto mayor sea la muestra y más iteraciones se realicen, más precisa será la estimación.

Sin embargo, es importante destacar que el método random de selección de nodos es una técnica aproximada y puede no capturar todas las características de la centralidad de intermediación en el grafo.

El método degree (grado) de selección de nodos en el algoritmo de Betweenness Centrality es una estrategia utilizada para calcular la centralidad de intermediación de los nodos en un grafo. Este método se basa en el grado de los nodos, es decir, en el número de aristas que están conectadas a cada nodo.

El algoritmo de Betweenness Centrality calcula la importancia de un nodo como intermediario en las rutas más cortas entre otros nodos en el grafo. El método degree de selección de nodos se puede aplicar de la siguiente manera:

1. Seleccionar los nodos con mayor grado en el grafo. Estos son los nodos que tienen la mayor cantidad de conexiones con otros nodos en el grafo.
2. Calcular la centralidad de intermediación para cada uno de los nodos seleccionados utilizando los caminos más cortos que pasan por ellos.
3. Repetir el proceso para un número suficiente de nodos, dependiendo de los requerimientos específicos del problema.

El método degree de selección de nodos se basa en la idea de que los nodos con un mayor número de conexiones tienen más probabilidades de ser importantes en la centralidad de intermediación. Esto se debe a que los nodos con mayor grado actúan como "puentes" entre diferentes partes del grafo y tienen más posibilidades de estar involucrados en las rutas más cortas entre otros nodos.

Sin embargo, es importante tener en cuenta que el método degree de selección de nodos puede no capturar todas las características de la centralidad de intermediación en el grafo. En algunos casos, los nodos con menor grado pueden tener un impacto significativo en las rutas más cortas y, por lo tanto, en la centralidad de intermediación [4].

La centralidad de la interrelación (Betweenness Centralito) de cada vértice es el número de estos caminos dentro del vértice. La centralidad de la interrelación tiene amplias aplicaciones en la teoría de redes: representa el grado en que los nodos se interponen entre sí. Por ejemplo, en una red de comunicación, un nodo con mayor puntuación tendrá más control sobre la red porque fluirá más información a través de ella. El centro de los nodos se establece como punto central.

Se aplica a muchos problemas diferentes en la teoría de redes, incluidos los relacionados con las redes sociales, la biología, el transporte y las tecnologías de la comunicación [8].

3.2 Algoritmo de PageRank

En 1999, Larry Page y su amigo y compañero de Standford Sergey Brin, desarrollaron el algoritmo de Page Rank. Un algoritmo que está muy ligado al Science Citation Index (SCI), que mide en función de la influencia y relevancia en base al número de publicaciones y sus respectivas citaciones y referencias.

Previo a la creación y diseño de este algoritmo, un usuario realizaba una pregunta en un ordenador o buscador en internet y el ordenador busca todos los documentos o páginas web que contienen que esa palabra, e internamente el ordenador tiene una tabla en la que ve con cuanta frecuencia salen esos términos que aparecen en tu búsqueda y mira cuantos documentos hacen referencia a esos términos, pero en vez de tener un documento con esas palabras, obtienes las palabras que apuntan a esos documentos y a partir de ahí, se crea un grafo bipartito como previamente se ha explicado, en la que las columna U serían los términos que apuntan a documentos o páginas webs que sería la columna V, para posteriormente obtener las proyecciones del grafo V para así tener las relaciones de documentos y las conexiones entre otros documentos o páginas webs. Por lo que aplicando esta forma de búsqueda se convierte en inviable pues el resultado sería la relación entre mas de 10000 documentos o webs, por lo que a la hora de ofrecerlo al usuario se debe realizar un ranking y es el momento en el que se creó el algoritmo de PageRank, que obtiene el grafo de la proyección de V teniendo en cuenta esos enlaces y mide la importancia de cada web para así ofrecer al usuario un número menor de páginas relevantes. La siguiente imagen demuestra el proceso realizado con anterioridad a la creación de este algoritmo [3].

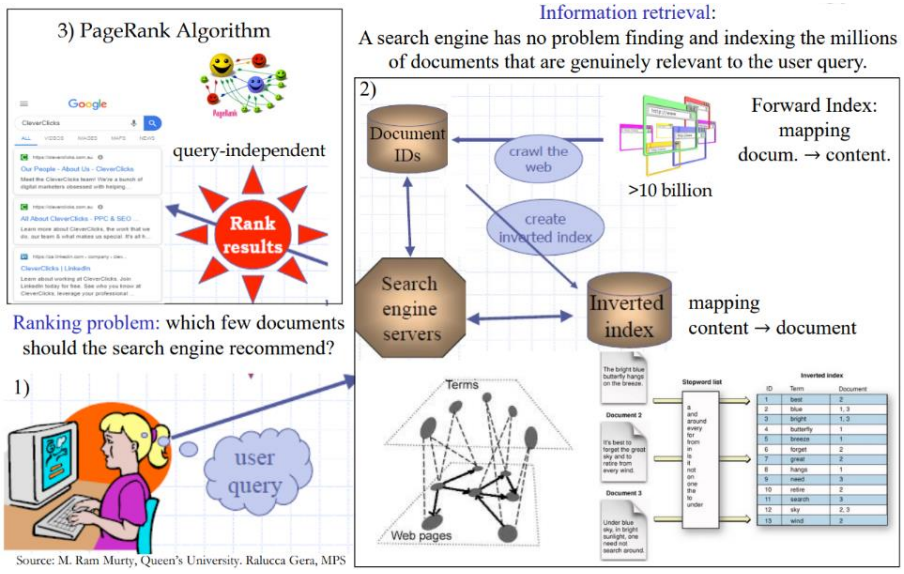


Figura 3-2. Algoritmo de búsqueda previo a PageRank.

Fuente: [3]

Page Rank es un algoritmo muy parecido, pero en el mundo de Google, que fue creado para darle una importancia en base a una puntuación de una página web, tratando a las web como un grafo dirigido no ponderado.

El propósito del diseño de este algoritmo fue probar y mejorar los resultados del motor de búsqueda de Google para los sitios web en los que trabajaron. Antes de la publicación de dicho trabajo, como se ha explicado previamente, los buscadores se basaban en heurísticas como el número de apariciones de una palabra clave a partir de una búsqueda, o el número de enlaces a esa página.

Sin embargo, las clasificaciones basadas en este tipo de estrategia se pueden usar fácilmente para intentar

posicionarse más alto en los motores de búsqueda. Por ejemplo, una página web basada en la repetición de una misma palabra comenzará a aparecer en la clasificación en función del número de apariciones de la palabra clave. En el caso de programas basados en la cantidad de enlaces a una página determinada, puede ser más fácil utilizar los resultados creando una gran cantidad de páginas web con enlaces a la página de destino para mejorar su clasificación.

PageRank es un algoritmo que mide la influencia transitiva, o direccional, de los nodos, es decir, mide la probabilidad de que de forma aleatoria cliquees en enlace y llegue a una página web particular. Todos los demás algoritmos de centralidad miden la influencia directa de un nodo, mientras que PageRank tiene en cuenta la influencia de tus vecinos y sus vecinos, es decir, la reputación de cada nodo respecto a todos los demás. Por ejemplo, tener unos pocos amigos influyentes puede aumentar tu PageRank más que tener muchos amigos poco influyentes.

El PageRank se calcula distribuyendo iterativamente el rango de un nodo (basado originalmente en el grado) entre sus vecinos o recorriendo el grafo aleatoriamente y contando la frecuencia con la que se alcanza cada nodo durante estos recorridos. [4]

También, existe el algoritmo de PageRank simplificado o también llamado normalized eigenvector centrality scoring, que genera una puntuación de rango, es decir, una distribución de probabilidad, que es la probabilidad en el cual una persona haya clics aleatoriamente en una página y llegue finalmente a otra determinada.

Previo a la explicación del algoritmo simplificado de Page Rank se debe entender cómo se transfiere la probabilidad o influencia entre nodos, se explicará usando de ejemplo y demostración el siguiente grafo con sus probabilidades o influencia.

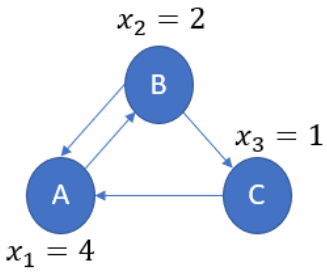


Figura 3-3. Grafo de ejemplo

En este caso, para transferir influencia entre B y C al nodo A, debemos sumar las influencias de cada uno hacia el nodo deseado, es decir, sumar lo que conecta cada uno y con A, y esto se realiza de forma matemática gracias a la matriz de adyacencia y el vector x de los valores de los nodos. EL vector de los valores de los nodos y la matriz de adyacencia del grafo quedaría de la siguiente manera:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}, A^T = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \quad (3,2)$$

A continuación, para realizar la transferencia de influencia del grafo debemos hacer la siguiente operación, que sería A traspuesta por el vector x, y de forma matemática se representa a continuación,

$$y_i = A^T \mathrm{x} = \sum_{(j,i)\in\varepsilon} x_j = \begin{pmatrix} 3 \\ 4 \\ 2 \end{pmatrix}. \quad (3,3)$$

La operación que se realiza en y, es una suma de los vecinos x_j que enlazan con el nodo i, es decir, que tienen enlaces de i a j, entonces si $A_{ji,}$ que es la matriz de adyacencia traspuesta, tiene un 1, transfiere el valor de su enlace y como resultado obtienes el vector y que demuestra como se realiza la acumulación de influencias pues el nodo A obtiene 3 ya que recibe 2 de B y 1 de C, el nodo B tiene 4 pues recibe esos 4 de A y el C es 2 ya que sólo recibe de B.

Una vez definido lo anterior, el algoritmo de PageRank funciona prácticamente de la misma forma que acabamos de explicar pues la influencia se acumula, con la diferencia que las influencias, son las probabilidades de estar en cada nodo, a eso se le llama el Pr(A), la influencia de cliquear sobre ese nodo, y la probabilidad total debe ser 1.

Para transferir esta probabilidad, debemos tener en cuenta que si de B salen dos enlaces como en el ejemplo anterior se muestra, esa probabilidad se está dividendo por 2, o el número de enlaces salientes de ese nodo, y no podría ser acumulativa sino la suma de las probabilidades de los nodos sería mayor que 1, así que esta influencia se transfiere, pero de forma normalizada entre los nodos salientes. La siguiente fórmula expresa como se transfiere la influencia de cada nodo entre los demás nodos

$$\Pr(v_i) = \sum_{v_i \in vecino(v_i)} \frac{\Pr(v_j)}{k_{vj}^{out}}. \quad (3,4)$$

La anterior fórmula podría definirse como la influencia de que el nodo i es la suma de las influencias de los vecinos que este caso son los nodos j, pero divididos por cada número de enlaces salientes que tiene cada vecino, por lo que si estamos obteniendo el PageRank de A, estaríamos dividiendo la influencia de B entre 2, que son sus enlaces salientes más la influencia de C entre 1. La fórmula anterior en la que se suman sólo de los vecinos es equivalente a hacer la suma pero de todos los vecinos y multiplicándolo posteriormente por $A_{ji,}$ que es la matriz de adyacencia del grafo, ya que si es 1, significa que es vecino pero si es 0, significa que no es vecino. Esta operación es similar a lo antes explicado, pues para obtener el Page Rank de todos los nodos es igual a la matriz A traspuesta para dividirlo por una matriz diagonal de los degrees de cada nodo, que es el número de enlaces salientes, multiplicado por el Page Rank de los nodos, quedando la siguiente operación con matrices. La siguiente operación se realiza de forma iterativa hasta llegar al punto de convergencia en el cual la y es prácticamente igual a la x, es decir, que el resultado coincida con el valor de partida y por mucho que hagas la operación no se realiza ningún cambio, y como solución significa que has obtenido un autovector dominante de la solución de esta ecuación

$$\begin{pmatrix} \Pr(A) \\ \Pr(B) \\ \Pr(C) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} k_A^{out} & 0 & 0 \\ 0 & k_B^{out} & 0 \\ 0 & 0 & k_C^{out} 0 \end{pmatrix}^{-1} \begin{pmatrix} \Pr(A) \\ \Pr(B) \\ \Pr(C) \end{pmatrix}. \quad (3,5)$$

La ecuación anterior podría definirse como una operación de autovalores, pero la dificultad de esto es muy grande debido al gran tamaño de las matrices por lo que haría falta muchísimos recursos para obtener un resultado de un grafo tan grande.

A continuación, se muestra matemáticamente el cálculo de esta probabilidad por cada nodo aplicando esta fórmula

$$\Pr(A) = \frac{\Pr(B)}{2} + \frac{\Pr(C)}{1},\ \Pr(B) = \frac{\Pr(A)}{1},\ \Pr(C) = \frac{\Pr(B)}{2}. \quad (3,6)$$

El sumatorio de la anterior fórmula debe dar siempre 1, pues la suma de todos los nodos de un grafo es siempre 1.

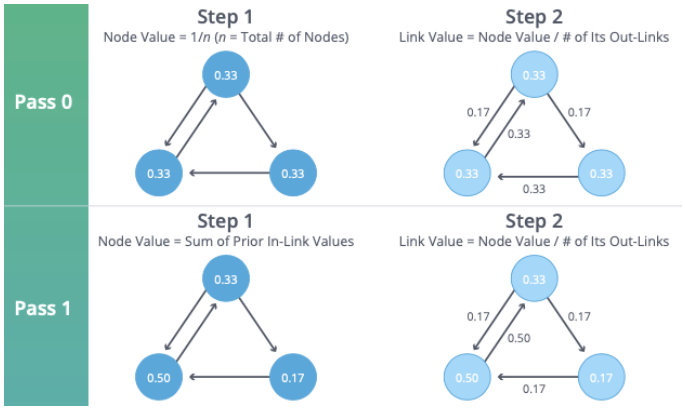


Figura 3-4. Ejemplo de iteraciones del algoritmo

En la anterior imagen se ve el funcionamiento de este algoritmo según se ha explicado mediante las fórmulas anteriores. En el primer paso cada nodo tiene asignada una influencia por lo que comienza el algoritmo y se produce la primera iteración y se ve como el nodo superior divide entre 2, pues sólo tiene dos enlaces, su probabilidad y la manda a sus nodos vecino, así lo hace también el nodo derecho e izquierdo, pero ninguno divide su influencia pues sólo tienen un nodo vecino cada nodo. En el siguiente paso, se ve que la probabilidad de cada nodo se ha modificado y el nodo izquierdo tiene la mayor pues recibió la mitad de la probabilidad del nodo superior y la probabilidad completa del nodo derecho, y a continuación se vuelve a producir la misma operación y así iterativamente hasta que las puntuaciones coinciden con las obtenidas previamente a esa iteración y se puede decir que ha convergido.

Uno de los problemas del uso de este algoritmo son las llamadas 'spider traps', que ocurre cuando en una página web no existen enlaces que te redirijan a otras páginas de ese grupo, lo que provoca que el usuario que está en esa página web acabe atrapado y la solución sea moverse en círculos entres las páginas de esa misma web, por lo que los votos, o la puntuación, no se redistribuye como debería según el algoritmo y la influencia se quedaría atrapada [9]. Las siguientes imágenes muestran el grafo de los nodos atrapados y la demostración de cómo su influencia queda atrapada.

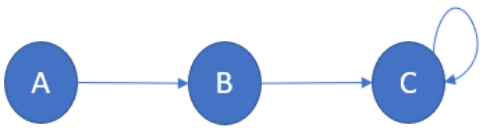


Figura 3-5. Ejemplo de grafo con spider trap

Y la siguiente fórmula demuestra la acumulación de influencia y cómo acabaría el nodo C con el Pr = 1, ya que se acumularía todo al final.

$$\Pr(A) = 0,\ \Pr(B) = \Pr(\mathrm{A}),\ \ \Pr(C) = \Pr(B) + \Pr(A) \tag{3,7}$$

Otro de los problemas son los dead ends, destruyen la influencia ya que no hay loop pues la acumulan y la destruyen y no permiten normalizar la probabilidad ya que la siguiente iteración no vas a ningún lado, pero tampoco te quedas por lo que esa probabilidad desaparece.

Figura 3-6. Ejemplo de grafo con dead end

En este caso, la influencia es destrozada y desaparece al no haber loop, o bucle, al final del grafo, por lo que la fórmula matemática quedaría de la siguiente forma:

$$\Pr(A) = 0,\ \Pr(B) = \Pr(\mathrm{A}),\ \ \Pr(C) = \Pr(B) \tag{3,8}$$

La solución a esos dos problemas previamente explicados es la teleportación, que es cuando un usuario se queda atrapado en una trampa de araña o en un camino sin salida, el algoritmo de PageRank asigna una pequeña probabilidad de teletransportarse a cualquier página web de forma aleatoria, y esto lo arregla porque en el caso que se viese un usuario atrapado en estos dos casos, podría teleportarse con esa probabilidad [3].

Gracias a la D, llamado dumping, es la probabilidad de seguir clicando sobre el grafo y (1-D) es la probabilidad de cansarte y teleportarte y comenzar en otro sitio, ya que podrías empezar en otro sitio o saltártelo. Y está definido por la siguiente fórmula

$$\Pr(v_i) = (1 - \mathrm{d})\frac{1}{N} + d\sum_{v_i \in vecino(v_i)} \frac{\Pr(v_j)}{k_{vj}^{out}} \tag{3,9}$$

La fórmula anterior, la primera parte es la probabilidad de la teleportación mientras que la segunda parte es la probabilidad de seguir clicando, y la suma de esta probabilidad total no debe superar el 1. La N de la fórmula representa la probabilidad de teleportación a alguno de los N nodos.

La iteración de la fórmula anterior de forma matricial, cuantas más iteraciones menos resultados vamos obteniendo hasta conseguir el número óptimo de resultados.

A continuación, se muestra el código en Python la fórmula explicada de forma que el usuario puede definir la matriz de adyacencia deseada, el número de iteraciones y su factor dumping o D.

```
def pagerank(M, num_iterations: int= 100, d: float = 0.85):
  N = M.shape[1]
  v = np.ones(N)/N
  M_hat = (d*M+(1-d)/N)
  for i in range(num_iterations):
    v = v @ M_hat
    return v
```

Figura 3-7. Código algoritmo de PageRank de forma matemática

La primera línea define la función pagerank que toma tres argumentos, M, que representa la matriz de transición del grafo, num_iterations que indica el número de iteraciones a realizar, y d, que es el factor de amortiguación (por defecto es 0.85). M es la traspuesta de la matriz de adyacencia y M_hat es el resultado de las iteraciones de este algoritmo.

Por último, se realiza un buble "for" para realizar todas las iteraciones en el cual, el vector v se multiplica con el resultado de la iteración realizada previamente [3].

A continuación, se explica un ejemplo sobre el funcionamiento de este algoritmo aplicándolo en la red social Instagram.

Piense que una red social como Instagram (que puede entenderse como un grupo de usuarios conectados por los seguimiento, por lo que puede considerarse como un grafo no ponderado, donde un grupo de usuarios serían los vértices y el conjunto de relaciones asociadas al seguimiento serían las aristas) un usuario normal, con pocos seguidores, comienza a seguir a un amigo que tampoco tiene muchos seguidores.

Instagram envía una notificación a todos los seguidores del usuario de que el usuario ha comenzado a seguir a su amigo. Dado que su número de seguidores es bajo, esta acción no tendrá mucha relevancia y es probable que nadie más comience a seguir al amigo de la universidad. Sin embargo, suponiendo que nuestro usuario, en lugar de tener un pequeño grupo de seguidores, sea un famoso en las redes sociales seguido por millones, las notificaciones de nuevos seguidores llegarán a más usuarios y tal vez el amigo vea que su número de seguidores está creciendo drásticamente.

Esta es básicamente la idea detrás del algoritmo PageRank, es decir, la puntuación de un vértice de grafo se basa en la relevancia de los vértices que contienen aristas que apuntan hacia el vértice. Para hallar la puntuación de los vértices de los grafos, se usa la fórmula definida cómo,

Algunos ejemplos de uso de este algoritmo son, el uso para la recomendación de personas en redes sociales como Instagram según tus seguidores y quien te sigue a ti, para ver la afluencia en las calles y así detectar cuáles son las principales para los usuario y también se usa dentro de una empresa, para la detección de comportamientos diferentes comparador con los demás empleados, se obtiene una puntuación y el trabajador que marca la diferencia es el objeto a revisar [4].

3.3 Detección de Comunidades

La detección de comunidades es la parte de la teoría de grafos que se centra en identificar agrupaciones o subconjuntos de nodos que están más conectados entre sí que con el resto del grafo, es decir, se trata de encontrar estructuras en un grafo que sean altamente cohesivas internamente y relativamente aisladas de las demás partes del grafo. Una de las principales características es que las comunidades no son necesariamente mutuamente excluyentes, es decir, un nodo puede pertenecer a más de una comunidad o ser el nexo entre dos comunidades.

En este contexto de las comunidades, son de especial relevancia la estructura de la matriz de adyacencia pues suelen tener unas estructuras muy específicas que reflejan las conexiones entre los nodos y las comunidades identificadas. Algunas de las características más importantes de la matriz de adyacencia en las comunidades son las siguientes. En muchas situaciones, las comunidades se representan como bloques diagonales en la matriz de adyacencia. Esto significa que los nodos dentro de una comunidad tienen conexiones más fuertes entre sí que con los nodos de otras comunidades. En una matriz de adyacencia con estructura de bloques diagonales, los valores en los bloques diagonales serán más altos o distintivos, mientras que los valores fuera de estos bloques serán más bajos o cercanos a cero. Los nodos dentro de una comunidad tienden a tener una alta densidad de conexiones entre sí. Esto se refleja en la matriz de adyacencia como una mayor concentración de valores no nulos dentro de los bloques correspondientes a las comunidades. Estas regiones tendrán más conexiones entre los nodos de la comunidad en comparación con las conexiones con nodos de otras comunidades.

Cuando los nodos de las comunidades están desordenados, características como los bloques en la matriz de adyacencia no son posible verlos fácilmente y es cuando se realiza un reordenamiento de los nodos para así obtener la matriz de adyacencia permutada.

Esta operación se realiza para poder agrupar nodos similares o que estén juntos para así facilitar el análisis y detección de comunidades en un grafo ya que, al agrupar nodos similares en una región específica de la matriz de adyacencia permutada, es más probable que aparezcan patrones o estructuras de comunidad en forma de bloques diagonales. A continuación, se muestra cómo sería una matriz desordenada y cómo quedaría después de aplicarle la siguiente operación

$$A' = PAP^T \tag{3,4}$$

a la matriz A, siendo la matriz de permutación P [10].

Después de haber aplicado la anterior operación para obtener la matriz de adyacencia permutada, podemos observar el siguiente cambio en la ordenación de nodos en la matriz

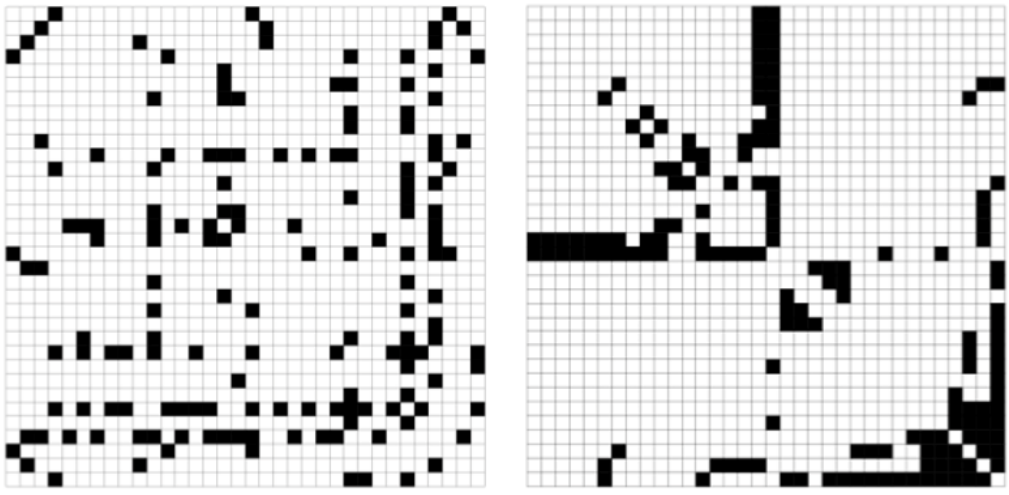


Figura 3-9. Matriz A y matriz A' permutada

Fuente: [3]

Las comunidades en un grafo pueden entenderse como grupos de nodos que tienen cierta similitud o afinidad en su comportamiento o en su función dentro del sistema que representa el grafo. Por ejemplo, en una red social, los usuarios que comparten intereses similares pueden formar una comunidad. En una red de transporte público, las estaciones que están más conectadas entre sí pueden formar una comunidad, aunque luego estén conectadas a más estaciones.

Los algoritmos más utilizados para la detección de comunidades son el algoritmo de Louvain, que es en el que se prestará más atención a continuación, el algoritmo de componentes débilmente conectados (WCC) y el algoritmo de propagación de etiquetas (LPA). Y los principales ejemplos para los que se usan los algoritmos de detección de comunidades es para la segmentación de imágenes, optimización de redes de transporte o la identificación de grupos de genes relacionado.

3.3.1 Algoritmo de Louvain

EL algoritmo de Louvain es un algoritmo como se explicó antes, para la detección de comunidades, y es usado principalmente en grandes redes o grafos, pues es uno de los algoritmos más rápidos en encontrar comunidades en estas redes. Aparte de detectar las comunidades, nos muestra la jerarquía de las comunidades a diferentes escalas [3].

Su principio de funcionamiento es maximizando la puntuación de modularidad para cada comunidad en cada iteración para así evaluar cómo de densamente está conectada esta comunidad comparándolos con los de una red aleatoria y conseguir una partición óptima del grafo en comunidades. Este algoritmo nos ayuda a identificar patrones y estructuras que no son posible a simple vista [4].

Uno de los principales problemas que se puede encontrar al utilizar este algoritmo es que, muchas veces, un nodo funciona como puente o de intermediario para la comunidad y sin ese nodo, la comunidad se desconectaría y con el funcionamiento de este algoritmo, que cada iteración los nodos se van moviendo entre comunidades vecinas, si ese nodo puente de una comunidad se mueve hacia otra comunidad, la conectividad de esta comunidad se rompería y con este algoritmo no se puede hacer nada para solucionar lo que sería el mayor de los problemas [11].

La modularidad es la medida que identifica cómo de bien se ha dividido los grupos en clústeres, es decir, la calidad en las divisiones del grafo, y compara las relaciones del grupo obtenido con lo que se esperaría si fuese un número de conexiones aleatorias. Se podría definir como la diferencia entre la cantidad de enlaces que hay en una comunidad y el número de enlaces esperados al azar, por lo que una alta modularidad indican que el número de comunidades detectadas tienen más enlaces que si las esperásemos al azar, que esto quiere decir que están bien elegidas las comunidades y de una forma compacta.

La modularidad está comprendida entre los valores -0.5 y 1, y su fórmula es

$$M = \frac{1}{2L}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2L}\right]\delta(c_i, c_j) \tag{3,4}$$

Dónde

- A_{ij}es la matriz de adyacencia que son los nodos i y j.
- $k_i k_j$son el grado del nodo que como se ha explicado antes, son el número de aristas que salen.
- L es el número de enlaces que hay en el grafo.
- c_i, c_j son las comunidades a las que pertenecen los nodos i y j.
- δ es la delta, que en el caso que los nodos pertenezcan a la misma comunidad será un 1 y sino un 0, por

lo que esta ´formula es útil en el caso que los nodos pertenezcan a la misma comunidad, sino no tendría sentido.

Se tiene que maximizar la modularidad para encontrar la comunidad de mayor calidad, es por eso, que el primer término debe ser lo mayor posible y el segundo lo más bajo, para que así se maximice [2].

Figura 3-10. División de comunidades.

El algoritmo de Louvain está dividido en varios pasos o fases.

1. Se identifica cada nodo como una única comunidad, es decir, el nodo A sería la comunidad 1 y el nodo B la comunidad 2, y así con todo el grafo, de ahí el nombre algoritmo aglomerativo, pues los nodos comienzan formando comunidades únicas y posteriormente se van agrupando.
2. Por cada nodo, se une a sus vecinos individualmente y se queda donde obtenga mayor puntuación de modularidad, y para ello se aplica la siguiente fórmula entre los dos nodos a unir,

$$M = E_{U,V}\left[A_{U,V} - \bar{A}_{U,V} \mid C_U = C_V\right] \tag{3,5}$$

en la cual se está calculado la media del número de conexiones que existiría entre dos nodos con respecto a la conexión media si ese grafo fuera la red aleatoria suponiendo que los nodos pertenecen a la misma comunidad, es decir, que la comunidad de U sea igual que la de V, por lo que se repetiría esa operación por cada par de nodos hasta obtener $\Delta M = 0$, que significaría que no se producen más aumentos de modularidad y se habrá obtenido una primera partición del grafo.

3. Por último, la comunidad resultante del anterior paso pasa a ser agregada a un único nodo por cada comunidad donde los ejes que hay entre las nuevas comunidades, es decir, los nuevos nodos, son la suma de los ejes entre comunidades previa a la agrupación y los enlaces que hay dentro de la agrupación del paso anterior se ponen como selfloop, donde el peso de los selfloops es el número de enlaces que había dentro de la agregación multiplicado por 2.
4. De nuevo, se vuelve a aplicar el algoritmo de Louvain al grafo obtenido en el paso 3, se vuelve al paso 1.

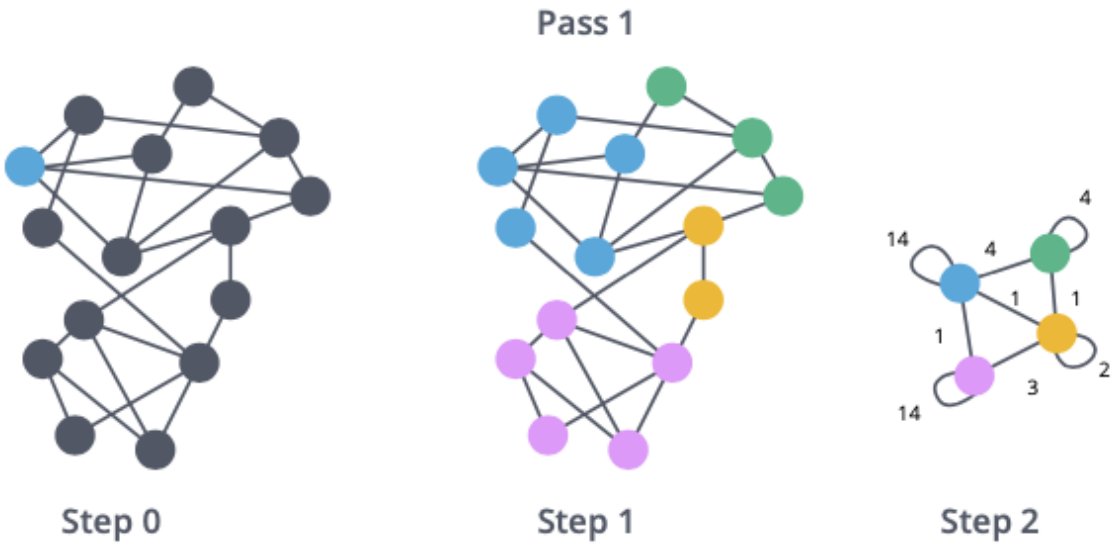


Figura 3-11. División de comunidades de Louvain primer paso.

La siguiente imagen muestra el resultado de haber aplicado el paso 1 de nuevo, posterior a la agrupación obtenida previamente, en la que se visualiza una nueva agrupación de nodos y la detección de comunidades más reducida.

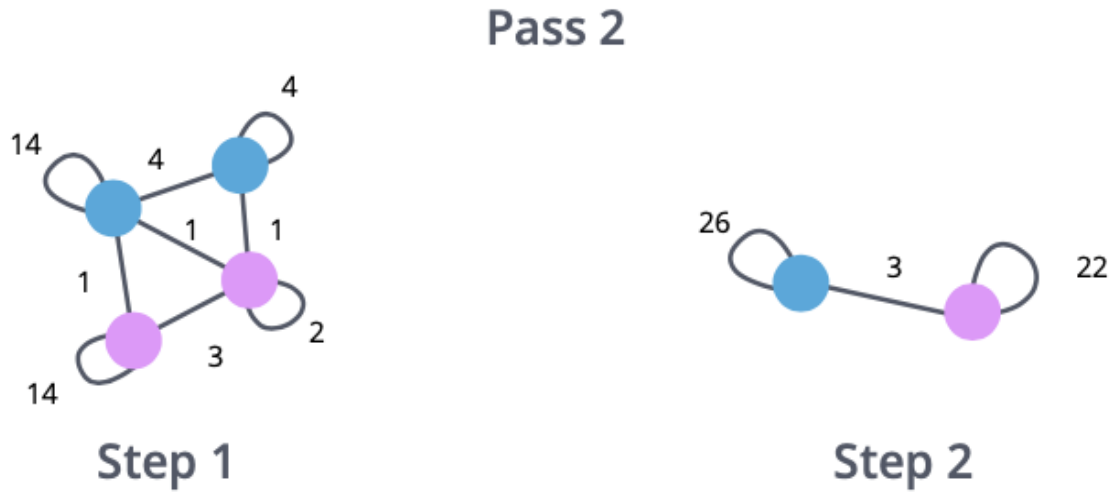


Figura 3-12. División de comunidades de Louvain segundo paso.

Gracias al parámetro resolución, que pertenece al algoritmo de Louvain, podemos obtener comunidades más grandes, o más pequeñas y específicas [12] [13] [2].

4 IMPLEMENTACIÓN Y RESULTADOS

En los puntos anteriores de este documento se han descrito los conceptos relacionado con los grafos desde su historia hasta sus características principales, y todo ello para entender el análisis que a continuación se explicará respecto a la ciudad de Sevilla, usando para ellos los algoritmos previamente descritos. Las ideas anteriores se han descrito desde una perspectiva teórica, es decir, su funcionamiento de análisis y la fórmula que define el algoritmo, sin haber considerado cuestiones prácticas ni implementaciones.

Sin embargo, el enfoque a seguir en este capítulo pretende ser completamente diferente, centrándose en las aplicaciones prácticas y dejando de lado los textos matemáticos. Por tanto, se espera explicar el código creado de acuerdo con su estructura, ya que incluso desde la implementación trata de explicar los conceptos descritos en el documento, prestando especial atención a la calidad del informe, para guardar y volver a utilizar el código en cualquier momento.

En este trabajo, se ha tratado de implementar los diferentes algoritmos para analizar el grafo de la ciudad de Sevilla gracias a Google Colab, que es un producto de Google Research donde se permite a cualquier usuario con conexión a internet escribir y ejecutar cualquier línea de código en el lenguaje de programación Python, un lenguaje de programación muy especializado en el análisis de datos y diversas técnicas de aprendizaje automático.

Para el análisis de grafos especialmente, se ha decidido implementar la librería de datos NetworkX, que como ellos definen se trata de un paquete de datos para la creación, manipulación, y estudio de las diferentes estructuras y funciones de redes complejas, que es nuestro caso sería el estudio de grafos.

El porqué de la elección de NetworkX para la realización de este trabajo, es que, gracias a esta librería, la facilidad para la lectura y posterior análisis de grafos se facilita en gran parte y es el más usado por los científicos de datos hoy en día.

4.1 Implementación

4.1.1 Descripción de librerías

4.1.1.1 Numpy

NumPy es una librería de Python diseñada para el procesamiento numérico de arrays multidimensionales y matrices. Proporciona una gran cantidad de herramientas para la manipulación de datos y el cálculo científico, lo que la convierte en una de las librerías fundamentales para la computación científica en Python.

En cuanto a su aplicación en el ámbito de los grafos y la ingeniería, NumPy permite realizar operaciones vectoriales y matriciales de una manera eficiente, lo que es esencial en la mayoría de las tareas relacionadas con

el procesamiento de datos y la simulación de sistemas.

Algunas de las aplicaciones más comunes de NumPy en estos campos son la representación de grafos, pues los grafica en función de su matriz de adyacencia y matriz de incidencia, lo que facilita su manipulación y análisis.

Otros usos muy importantes para los que se usa la librería Numpy es para el procesamiento de señales y audio, y de imágenes, gracias a su manipulación de matrices con píxeles.

4.1.1.2 SciPy

SciPy es una librería de Python que proporciona herramientas para realizar cálculo científico y matemático en Python. Se construye sobre NumPy, y proporciona una variedad de módulos para la computación científica, incluyendo integración numérica, optimización, álgebra lineal, transformadas de Fourier, procesamiento de señales e imágenes y análisis estadísticos.

Esta librería se usa mayoritariamente en el ámbito de la ingeniería para la resolución de problemas que requieren el procesamiento numérico y computacional, así como para el análisis y visualización de datos.

4.1.1.3 Matplotlib

Matplotlib es una librería de Python que permite la creación de grafo y visualizaciones de datos de una manera sencilla y eficiente. Se utiliza ampliamente en la visualización de datos en el ámbito de la ciencia, la ingeniería y la estadística, así como en otras disciplinas que requieren el análisis y la representación gráfica de datos.

Dentro del ámbito de los grafos y la ingeniería, Matplotlib es una herramienta muy útil para la visualización de datos obtenidos a partir de mediciones. Algunas de las aplicaciones más comunes de Matplotlib en estos campos son los gráficos de barras o histogramas pues permite la creación y representación de este tipo de datos, grafo de líneas como por ejemplo una línea de tendencia o los grafo de dispersión o 3 dimensiones.

4.1.1.4 OSMnx

OSMnx es una librería de Python que permite descargar, analizar, visualizar y modelar redes de calles y carreteras a partir de datos de OpenStreetMap (OSM). Esta librería se utiliza ampliamente en el análisis de sistemas de transporte y la planificación urbana.

OSMnx es una herramienta muy útil para la modelación y análisis de redes de transporte, ya que permite la descarga de datos de OpenStreetMap y la construcción de grafos de calles y carreteras a partir de estos datos, siendo las carreteras las aristas y los cruces los nodos.

Los principales usos para los que se usa esta herramienta son para el análisis de la red vial, pues gracias a OSMnx podemos ver las características de su conectividad, su topología y la centralidad de las carreteras según el mapa cargado. También es muy útil realizar análisis sobre la seguridad vial, pues se podría analizar cuál son las carreteras más importantes según el tráfico y la afluencia.

4.1.1.5 Community

Community es un subpaquete dentro de la librería de datos de NetworkX utilizada exclusivamente para la detección de comunidades de un grafo y todo tipo de análisis que tenga relación con la detección de comunidades, en este caso se ha utilizado para aplicar el algoritmo de detección de comunidades de Louvain.

```
[ ] !pip install --upgrade scipy==1.8.0 networkx==2.8 matplotlib==3.5.1 numpy==1.23 taxicab==0.0.3 osmnx community

[ ] import networkx as nx
    import matplotlib.pyplot as plt
    import osmnx as ox
    import taxicab as tc
    import community.community_louvain as community_louvain
    ox.settings.log_console=True
```

Figura 4-1. Fragmento para la instalación de librerías.

4.1.2 Diseño mapa de Sevilla

Después de importar las librerías y paquetes necesarios para poder realizar la ejecución de estos algoritmos, el primer paso es el diseño del mapa de Sevilla, que como se ha explicado antes, gracias a la librería OSMnx es posible cargar el mapa de Sevilla utilizando este comando. En el comando se especifica que el tipo de red que queremos es la red de tráfico de Sevilla, y posteriormente especificamos que el color de los nodos sea de color rojo. Al usar este comando, el mapa que se carga no tiene ninguna zona separada del resto del mapa, pues el comando especifica que todo el grafo debe estar conectado.

```
[ ] # Descargar el mapa de Sevilla desde OpenStreetMap
    sevilla = ox.graph_from_place('Seville, Spain', network_type='drive') #Seleccionamos el mapa de carreteras de Sevilla

    fig, ax = ox.plot_graph(sevilla, node_color = 'r', node_size = 1)
```

Figura 4-2. Fragmento creación mapa de Sevilla.

Como resultado se obtiene un grafo dirigido con las calles que son las aristas del grafo, y los puntos de color rojos son los cruces que serían nuestros nodos del grafo. Una vez que tenemos este modelo, podemos empezar a moldearlo para realizar la aplicación de los algoritmos descritos.

Figura 4-3. Mapa de Sevilla como grafo dirigido.

Una vez que el grafo de la ciudad de Sevilla se ha cargado correctamente, usamos el comando Folium para la visualización del mapa de una forma interactiva en la que poder ver bien que zonas de Sevilla se están cargando.

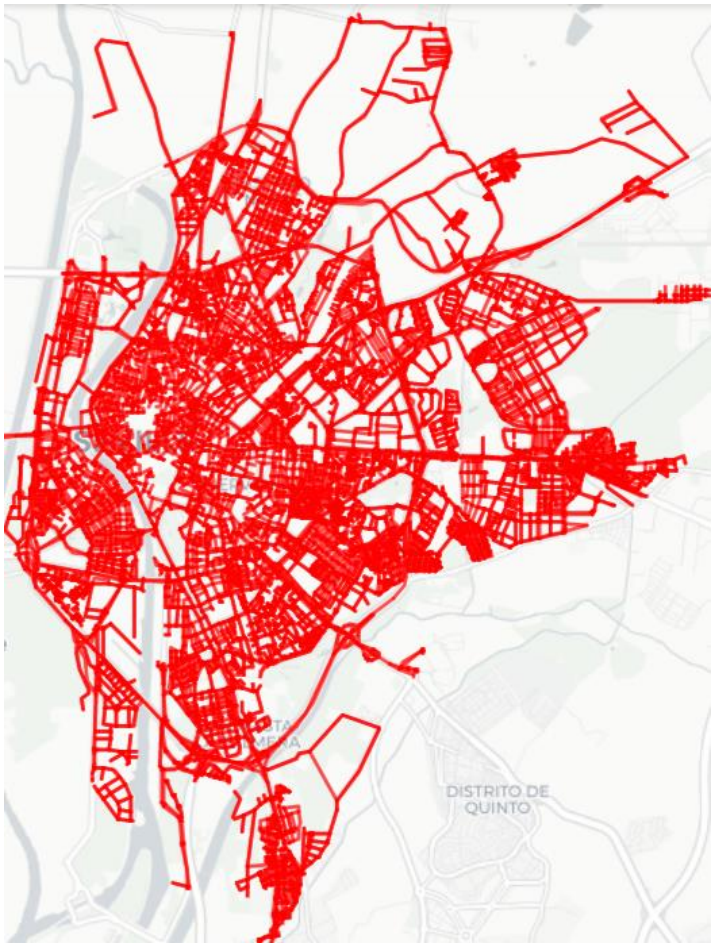


Figura 4-4. Mapa de Sevilla interactivo.

4.1.3 Aplicación algoritmo de Betweenness Centrality

Uno de los algoritmos a aplicar es el de Betweenness Centrality al grafo de la ciudad de Sevilla que antes se ha obtenido.

La centralidad de intermediación o Betweenness Centrality es una forma de detectar cómo de influyente es una arista, en nuestro caso, sobre el flujo de información que recorre un grafo, y gracias a la librería contenida en Python, NetworkX, podemos aplicar este algoritmo, sobre el grafo del mapa de Sevilla obtenido anteriormente.

```
# Cálculo la centralidad de intermediación de los nodos
bc = nx.betweenness_centrality(G)
```

Figura 4-4. Fragmento de código aplicación algoritmo.

El siguiente paso, ordenamos los nodos según su puntuación de centralidad intermedia y obtenemos los 15 nodos con mayor puntuación para posteriormente crear un grafo a partir de esos 10 nodos y graficarlos de nuevo en un mapa añadiéndole un marcador para así facilitar su visualización, todo ello se aplica en el fragmento de código siguiente.

```
[ ]  # Obtener los 15 nodos más importantes
     top_nodes = sorted(bc.keys(), key=lambda x: bc[x], reverse=True)[:15]

     # Crear un subgrafo con los 15 nodos más importantes
     H = G.subgraph(top_nodes)

     # Crear un mapa de folium con los nodos más importantes
     map = ox.plot_graph_folium(H)

     # Agregar un marcador en cada uno de los nodos
     for node in top_nodes:
         folium.Marker(location=[G.nodes[node]['y'], G.nodes[node]['x']]).add_to(map)

     # Mostrar el mapa
     map
```

Figura 4-5. Fragmento de código orden y visualización.

Al ejecutar este fragmento de código, podemos ver los 10 nodos más influyente del grafo según su conectividad con los demás nodos y obtenemos este resultado:

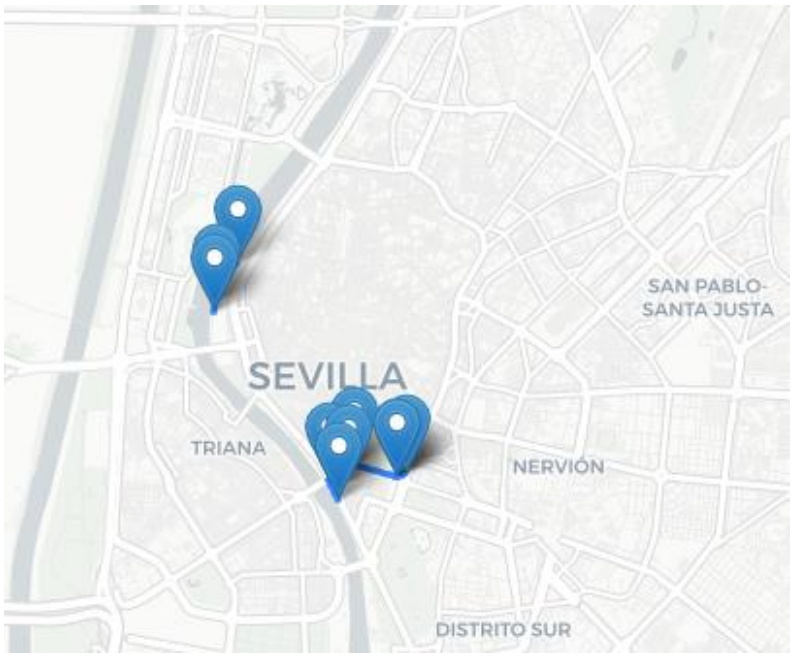


Figura 4-6. Mapa con nodos más importantes.

Como primeras conclusiones, podemos ver claramente cómo las avenidas alrededor del Centro Histórico de Sevilla son las que tienen mayor puntuación de centralidad intermedia, es decir, por los nodos donde mayor cantidad de tráfico de forma segura pasaría y habría que poner más importancia en esas zonas pues son de alta conexión con diferentes carreteras.

A continuación, se ha obtenido el histograma según la puntuación obtenida al aplicar el algoritmo de Betweenness Centrality usando el fragmento de código que a continuación se muestra. Primero, decidimos el tamaño de nuestra gráfica, pues al ser un histograma, preferiblemente era mejor tener una gráfica amplia, para

posteriormente obtener el valor de cada puntuación de centralidad y formar el histograma siguiente.

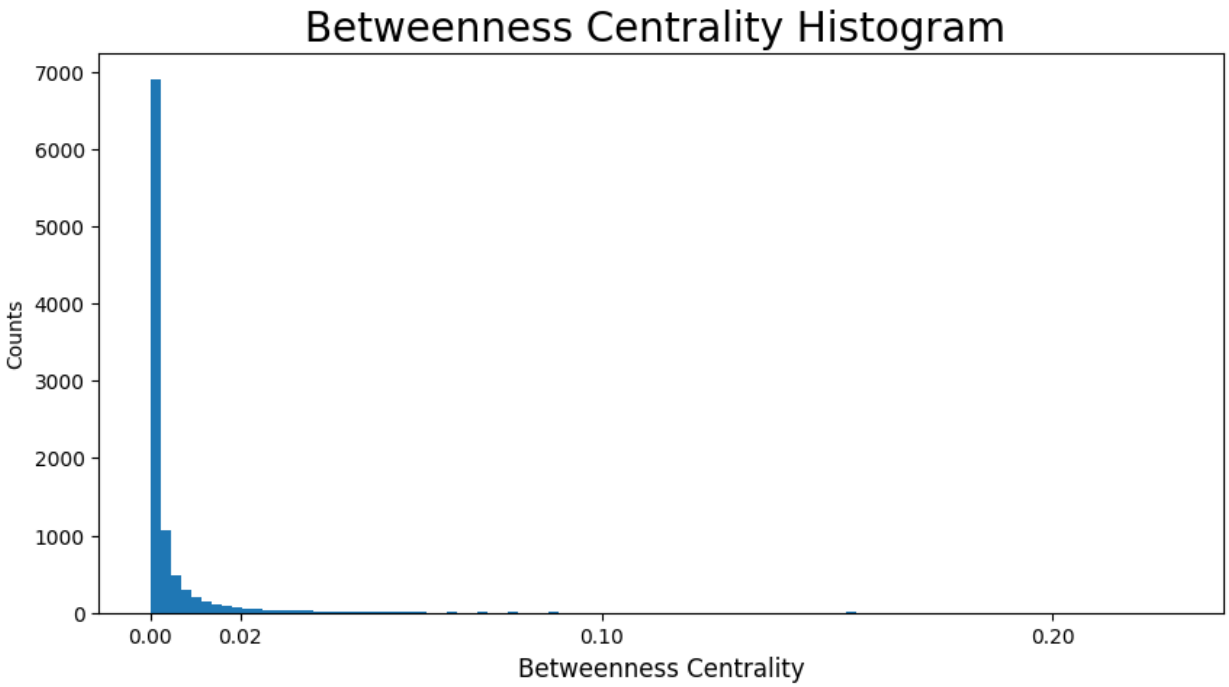


Figura 4-7. Histograma Betweenness Centrality.

Gracias a este histograma, en cuanto a las barras en la zona izquierda, podemos ver como la gran mayoría de nodos tienen una baja puntuación, que podríamos considerarlo como menos influyentes considerándolos en términos de intermediación, pero, sin embargo, no deberíamos de descartarlo por completos ya que podrían tener otros roles como por ejemplo la conexión con otras comunidades específicas.

En cuanto a la distribución de las puntuaciones, es claramente sesgada hacia el lado izquierdo, es decir, la gran mayoría de nodos con menor importancia, pero con algunos nodos con mayor puntuación que actúan como puentes o puntos de paso más críticos conectando diferentes zonas del grafo.

También, podemos llegar a la conclusión que no es un grafo que pueda dividirse en diferentes comunidades, pues de ser así, veríamos diferentes agrupaciones de barras, pero en este caso no es así.

Para finalizar el análisis de este algoritmo en la ciudad de Sevilla, se ha procedido a graficar el mapa completo y con el color de las aristas, en nuestro caso las calles, según su puntuación del algoritmo de centralidad de intermediación. Para ello, se ha procedido a aplicar el algoritmo de Betweenness Centrality de la librería de NetworkX, pero en este caso, comparado con el anterior, se ha aplicado a las aristas, posteriormente según su puntuación se ha asignado un color a la aristas, este color pertenece al mapa de color ‘tab20’.

Figura 4-8. Colormap tab20.

A continuación, se incluye el fragmento de código con el que se ha hallado lo anterior explicado.

```
[ ]  edge_dc = nx.edge_betweenness_centrality(G)
     nx.set_edge_attributes(G, edge_dc, "edge_dc")

[ ]  ec = ox.plot.get_edge_colors_by_attr(G, "edge_dc", cmap='tab20')#summer
     fig, ax = ox.plot_graph(G, edge_color=ec, edge_linewidth=1, node_size=0)
```

Figura 4-9. Fragmento de código.

El mapa de Sevilla hallado con este código muestra claramente las carreteras más influyentes en la ciudad. Podemos reconocer fácilmente que todas las carreteras diferentes de color azul son avenidas que rodean los barrios, que se pueden diferenciar por la agrupación de nodos muy pegados, por ello, las avenidas o carreteras al exterior de esos barrios serían la conexión entre diferentes agrupaciones de nodos y cualquier fallo en esas carreteras podría suponer grandes cortes de tráfico por ejemplo o servir de ayuda para una organización del transporte público en la ciudad para mejorar la accesibilidad y la movilidad en la ciudad entre diferentes barrios.

También se podría considerar estas carreteras con una alta puntuación de centralidad de intermediación para la construcción de carriles bici, pues como se ha explicado antes, sirven de conexión entre diferentes zonas de la ciudad, y cualquier fallo en estas carreteras podría crear grandes problemas de movilidad.

Figura 4-10. Mapa de Sevilla según puntuación.

4.1.4 Diseño usando algoritmo de Louvain

Una vez que tenemos las librerías y paquetes necesarios para obtener el grafo de la ciudad de Sevilla, en el que las calle como previamente se ha explicado, son las aristas del grafo y los nodos son los cruces entre esas

carreteras, el siguiente paso necesario para poder aplicar el algoritmo de detección de comunidades de Louvain es transformar el grafo de dirigido a no dirigido, es decir, analizaremos la distribución y la interconexión de las carreteras en función de su posición y no de su directividad.

```
sevilla_und = sevilla.to_undirected()
```

Figura 4-11. Fragmento código cambiar la directividad.

Una vez que tenemos el grafo no dirigido de la ciudad de Sevilla, se procede a aplicar el algoritmo de Louvain gracias a la librería NetworkX incluida en Python.

```
[ ]  # Aplicar el algoritmo de detección de comunidades de Louvain
     louvain = community_louvain.best_partition(sevilla_und)
```

Figura 4-12. Fragmento código que ejecuta el algoritmo de Louvain.

Ya aplicado el algoritmo de detección de comunidades de Louvain, tenemos esas comunidades, y se procede a colorear los nodos que pertenecen a cada comunidad de un color, para así facilitar su visualización aplicando el mapa de color 'viridis'.

```
# Asignar un color a cada comunidad detectada
colors = [louvain[n] for n in sevilla_und.nodes()]
cmap = plt.cm.get_cmap('viridis', max(colors) + 1)
```

Figura 4-13. Fragmento de código para asignar color a la gráfica.

Figura 4-14. Colormap viridis.

A continuación, se muestra la imagen obtenida con el mapa de color 'viridis' aplicado por comunidades.

Figura 4-15. Mapa dividido por comunidades según nodos.

Cómo se puede observar en la siguiente imagen, el graficado del grafo dividido en comunidades dándole color a los nodos no es del todo preciso, por eso, en el siguiente paso se ha graficado de nuevo el mapa de Sevilla dividido en comunidades según el algoritmo de Louvain, habiéndose pasado primero a un formato en línea, es decir, que no salen los nodos y sólo las carreteras, pero esta vez se ha coloreado los nodos en función de sus aristas, para así ver mejor el resultado y de forma más clara, como se muestra en la siguiente imagen.

Para la obtención de este graficado con estos colores, se ha decidido usar el mapa de color 'Spectral' como en la siguiente imagen se muestra.

Figura 4-16. Colormap Spectral.

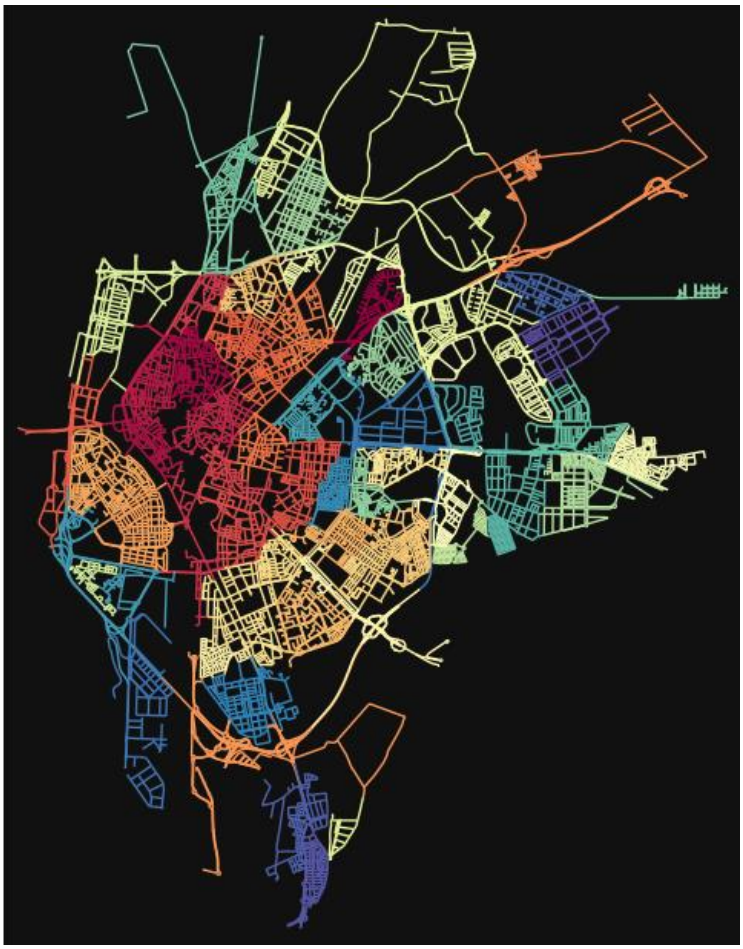


Figura 4-17. Mapa dividido por comunidades según ejes.

Una vez que se tiene el mapa de Sevilla dividido por comunidades gracias al algoritmo de Louvain como se muestra en la imagen superior, aplicando el mismo comando para obtener el grafo de la ciudad de Sevilla en forma de nodos y aristas, se ha usado el mismo para obtenerlo para los barrios más importante de la ciudad como son el casco histórico, los Remedios, Triana, la zona norte y la zona sur, para así demostrar cómo los barrios de Sevilla en estructura, distribución y conexión es un claro ejemplo de división por comunidades.

Figura 4-18. Grafo barrio de Triana.

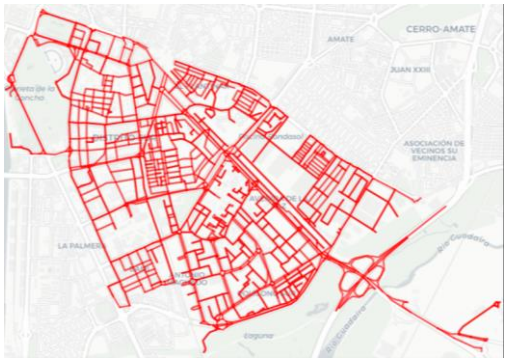


Figura 4-19. Grafo barrio de zona Sur.

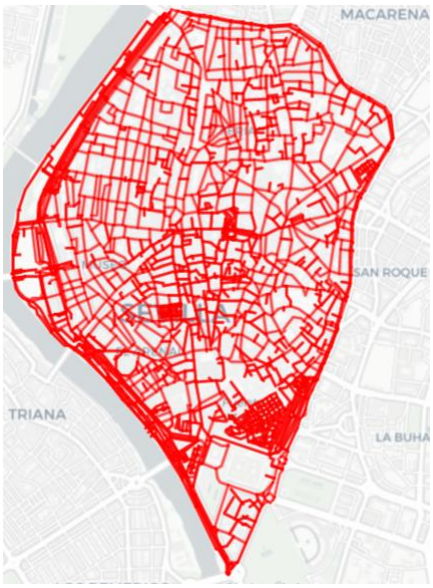


Figura 4-20. Grafo barrio de zona centro histórico.

En primer lugar y de forma más clara, vemos como el mapa del centro histórico está perfectamente dividido como una comunidad en el grafo de la ciudad de Sevilla, en la siguiente imagen aparece la zona con el barrio identificado.

Figura 4-21. Grafo barrio de zona centro histórico superpuesto.

La zona Sur de Sevilla es otra zona que está perfectamente detectada como una comunidad, pero en este caso sale delimitada por dos colores diferentes al ser una zona de Sevilla y no un barrio, pero superponiendo ambas imágenes se ve claramente su perfecta delimitación y el parecido de los colores debido a su cercanía.

Figura 4-22. Grafo barrio de zona sur superpuesto.

El barrio de Triana demuestra claramente el funcionamiento de este algoritmo con los grafos, pues es un barrio el cual se podría dividir en 3 subgrafos, como así lo ha dividido el algoritmo de Louvain, que hay 3 zonas en las que se produce una acumulación de nodos y a cada una de las 3 se le ha asignado un color diferente.

Figura 4-23. Grafo barrio de zona Triana superpuesto.

Como conclusión, unas de las principales a la que puede ayudar usar este algoritmo en diferentes aspectos sería para ver cómo son las interacciones sociales en cada comunidad o que patrón destaca en cada una. También para poder diferenciar habitantes en cada zona divididas por los intereses o actividades similares en esa zona, o, por ejemplo, para saber que comunidades tienen más relación entre ellas y así poder obtener un patrón común de comportamiento o pensamiento.

4.1.5 Diseño usando algoritmo PageRank

El algoritmo de PageRank, como bien se ha explicado antes, se desarrolló para clasificar la importancia de las páginas web, pero también se puede aplicar a relaciones entre amigos o grupos de amigos en una red social.

En este contexto, el algoritmo puede ayudar a identificar a las personas más influyentes o populares dentro de la red, en lugar de considerar enlaces entre páginas web en este caso se consideran conexiones o interacciones sociales entre amigos, podríamos pensar, por ejemplo, cuando se siguen en una red social.

Cada persona en la red social se representa como un nodo en el grafo, y las relaciones entre ellos serían las aristas de nuestro grafo. Cuando aplicamos este algoritmo sobre la estructura de este gafo se busca determinar qué usuarios tienen más influencia en función de cuántos y qué tipo de enlaces tienen.

Ahora vamos a demostrar que una persona puede tener una alta puntuación de Page Rank si tiene muchos amigos que también son populares y tienen muchos seguidores, por lo que al aplicar este algoritmo obtendremos la información para ver quién son las personas más relevantes, es decir, con más popularidad dentro de la red

En primer lugar, debemos importar las librerías y paquetes necesarios para poder realizar el ejemplo anteriormente descrito, en este caso serán las librerías NetworkX, para aplicar el algoritmo de PageRank a nuestro grafo, y matplotlib, necesario para graficar estos.

```
[8]  import networkx as nx
     import matplotlib.pyplot as plt
     plt.rcParams.update(plt.rcParamsDefault)
     plt.rcParams.update({'figure.figsize': (7.5, 5)})
```

Figura 4-24. Fragmento de código para importar las librerías.

Una vez que están las librerías y subpaquetes importadas, se crea un grafo dirigido, y es dirigido, por que una persona o usuario de esta red social, puede seguir a otra persona pero que no sea de forma recíproca, como por ejemplo los famosos, tienen grandes cantidades de seguidores, pero no de personas que le siguen.

Los datos de las relaciones de personas son introducidos a partir de un fichero .txt con formato de cada fila (source, target), que para ello se procede a leer gracias al siguiente fragmento de código.

```
[3]  # Crear un grafo dirigido
     G = nx.DiGraph()
     # Agregar aristas al grafo
     with open('datos1.txt', 'r') as file:
         for line in file:
             source, target = line.strip().split(',')
             G.add_edge(source, target)

     # Crear un grafo dirigido
     H = nx.DiGraph()
     # Agregar aristas al grafo
     with open('datos2.txt', 'r') as file:
         for line in file:
             source, target = line.strip().split(',')
             G.add_edge(source, target)
```

Figura 4-25. Fragmento de código para crear los grafos.

Se han creado los grafos G y H pues la intención es que cada grafo sea un grupo de amigos con sus distintas relaciones de seguimiento entre ellos, pero todos tienen en común que siguen a un famoso.

El siguiente paso, es la unión entre las dos gráficas y posteriormente aplicar el algoritmo de PageRank, con un Alpha, que es el factor de amortiguación que es utilizado en nuestro caso para la probabilidad que un usuario siga a otra persona sin haber venido de otro amigo, sino que lo ha encontrado por su cuenta, esto se ha introducido para que no se produzcan bucles infinitos en los grafos, de 0.85 pues es el más común para todas las pruebas.

```
[4]  # Unir las gráficas G y H en una gráfica combinada
     comb_graph = nx.compose(G, H)

     # Aplicar el algoritmo de PageRank
     pr = nx.pagerank(comb_graph, alpha=0.85)
```

Figura 4-26. Fragmento de código para unir grafos y aplicar algoritmo.

A continuación, añadimos el código utilizado para el graficar las conexiones del grafo. Para el tamaño de los nodos, cuanto mayor puntuación de PageRank, de mayor tamaño será el nodo. Para el color de las aristas, se utilizará el mismo procedimiento, pues cuanto más oscura, mayor puntuación de PageRank habrá entre dos nodos.

```
[4]  # Graficar el grafo con los nodos de mayor a menor puntuación de PageRank
     pos = nx.kamada_kawai_layout(comb_graph)
     node_sizes = [5000 * pr[n] for n in comb_graph.nodes()]
     node_colors = [pr[n] for n in comb_graph.nodes()]
     edge_widths = [5 * pr[e[1]] for e in comb_graph.edges()]

     plt.figure(figsize=(8, 6))
     nx.draw(comb_graph, pos, node_size=node_sizes, node_color=node_colors,width =edge_widths,  cmap=plt.cm.Greens, with_labels=True)
     plt.title("Grafo algoritmo de PageRank")
     plt.show()
```

Figura 4-27. Fragmento de código para graficar.

Por último, es necesario saber la puntuación de cada nodo par así proceder a un análisis más exacto que si nos fijásemos sólo en el tamaño de los nodos, para ello se han ordenado de mayor a menor y se imprime por pantalla la puntuación con el formato: Nodo nombre: valor.

```
[5] # Obtener los nodos y sus puntuaciones de PageRank ordenados de mayor a menor
    sorted_pr = sorted(pr.items(), key=lambda x: x[1], reverse=True)

    # Imprimir los nodos y sus puntuaciones de PageRank
    print("Nodos y sus puntuaciones de PageRank:")
    #print(pr)
    for node, score in sorted_pr:
        print(f"Nodo {node}: {score}")
```

Figura 4-28. Fragmento de código para ordenar puntuaciones.

4.1.5.1 Caso 1

En el primer caso, dos personas se siguen mutuamente por lo que la puntuación de PageRank es la misma, pues los dos reciben entre los dos.

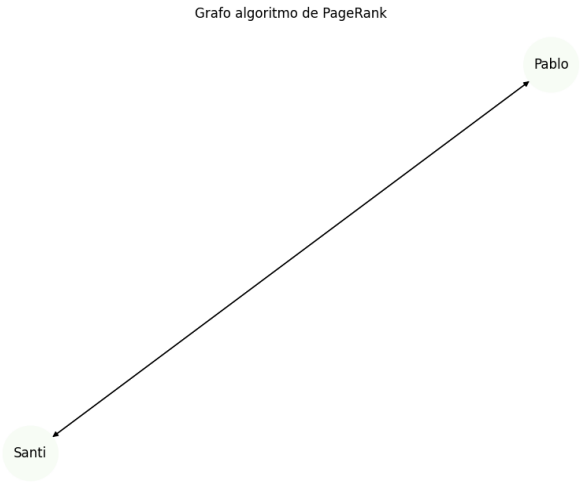


Figura 4-28. Gráfica análisis PageRank caso 1.

Nodos	Puntuación PR
Nodo Santi	0,5
Nodo Pablo	0,5

Tabla 4-1. Resultados caso 1.

4.1.5.2 Caso 2

En el caso 2 se ha añadido una nueva conexión.

Juan es compañero de Santi por lo que ambos se siguen, y a su vez, Santi presenta a Pablo su amigo Juan, por lo que Juan le sigue, pero Pablo a Juan no, por lo que los resultados varían según el caso anterior. El que más puntuación es Santi, pues le siguen Juan y Pablo y a su vez sigue a Pablo y Juan. En segunda posición, está Pablo pues como dice el algoritmo de Page Rank, se da más importancia a que te sigan enlaces importantes a tener muchos seguidos.

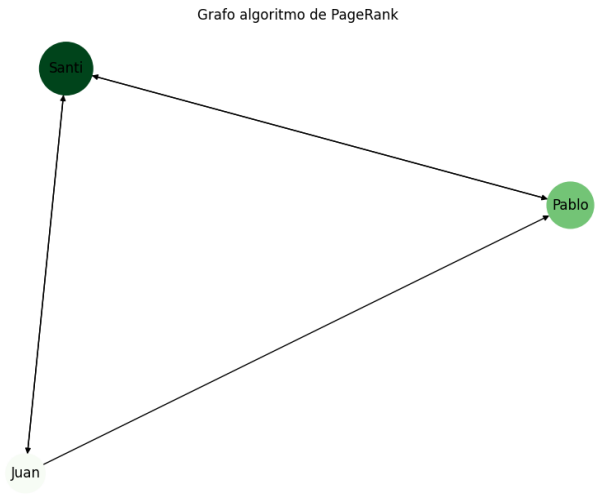


Figura 4-29. Gráfica análisis PageRank caso 2.

Nodos	Puntuación PR
Nodo Santi	0.43274880303664615
Nodo Pablo	0.33333333333333326
Nodo Juan	0.23391786363002037

Tabla 4-2. Resultados caso 2.

4.1.5.3 Caso 3

Los 3 amigos tienen novia y se siguen entre el novio y la novia y obtenemos los siguientes resultados.

Como podemos comprobar, la novia con mayor puntuación es la de Pablo, pues, aunque Santi tenga la mayor puntuación, los enlaces que recibe Pablo son de gran importancia y calidad, pues recibe de Santi que es el que más puntuación tiene, al ser la persona que más seguidores y seguidos tiene.

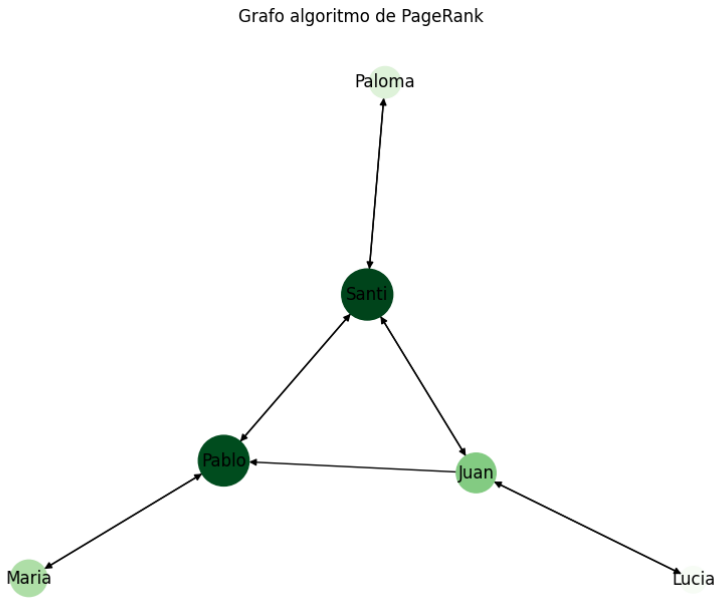


Figura 4-30. Gráfica análisis PageRank caso 3.

Nodos	Puntuación PR
Nodo Santi	0.2680786300744857
Nodo Pablo	0.26272438123776226
Nodo Juan	0.16097428005258876
Nodo María	0.1366576032734455
Nodo Paloma	0.10095615326614654
Nodo Lucía	0.070608952095571

Tabla 4-3. Resultados caso 3.

4.1.5.4 Caso 4

En este cuarto caso, todos los amigos siguen a un famoso, que se le ha puesto de nombre MC, al ser la persona que más conexiones sociales tiene, es decir, el más seguido, aunque esa persona no siga a nadie tendrá la puntuación más alta.

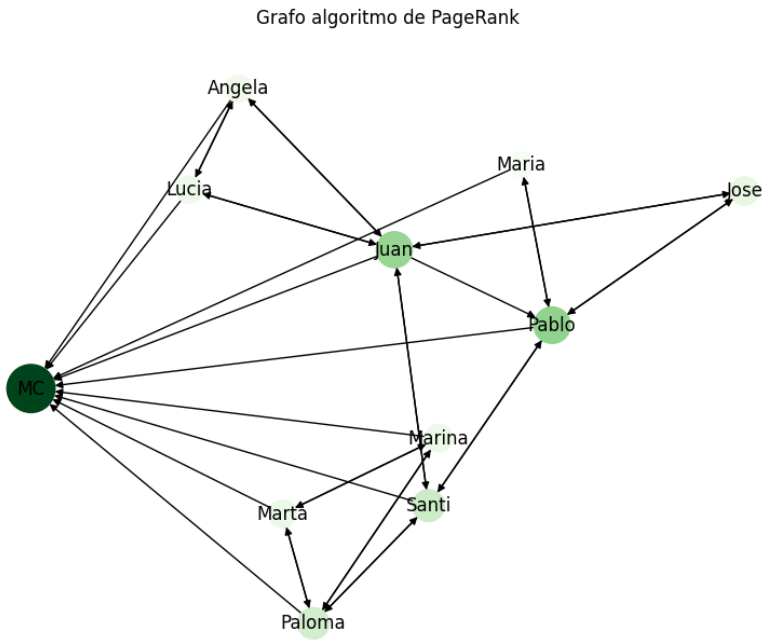


Figura 4-31. Gráfica análisis PageRank caso 4.

Nodos	Puntuación PR
Nodo MC	0.2052942099249399
Nodo Pablo	0.11799299158852745
Nodo Juan	0.1147230290070561
Nodo Santi	0.08915806726167273
Nodo Paloma	0.08626623213924643
Nodo José	0.07082636028178423
Nodo Marina	0.06674190128451109
Nodo Marta	0.06674190128451109
Nodo Lucía	0.06384077828712253
Nodo Ángela	0.06384077828712253
Nodo María	0.05457375065350591

Tabla 4-4. Resultados caso 4.

4.1.5.5 Caso 5

Por último, este caso incluye a dos grupos de amigos con la única persona en común entre ellos es Lucía, pero todos siguen también a MC y también aparece el nombre de Juan Carlos como otra persona aislada a los grupos de amigos antes descritos que sigue a MC y MC sigue también a Juan Carlos.

Al aplicar el algoritmo de PageRank, MC sigue siendo la persona con mayor puntuación pues es la persona que más enlaces recibe, pues recibe de todos. Pero en este caso, al seguir MC también a Juan Carlos, éste tendrá la segunda posición porque, aunque sólo reciba un seguimiento comparado con los demás, es de gran calidad y como se define el algoritmo, una persona con muchas conexiones sociales, se vuelve muy influyente en el grafo y contribuirá a aumentar la puntuación a aquellos a los que esté conectado.

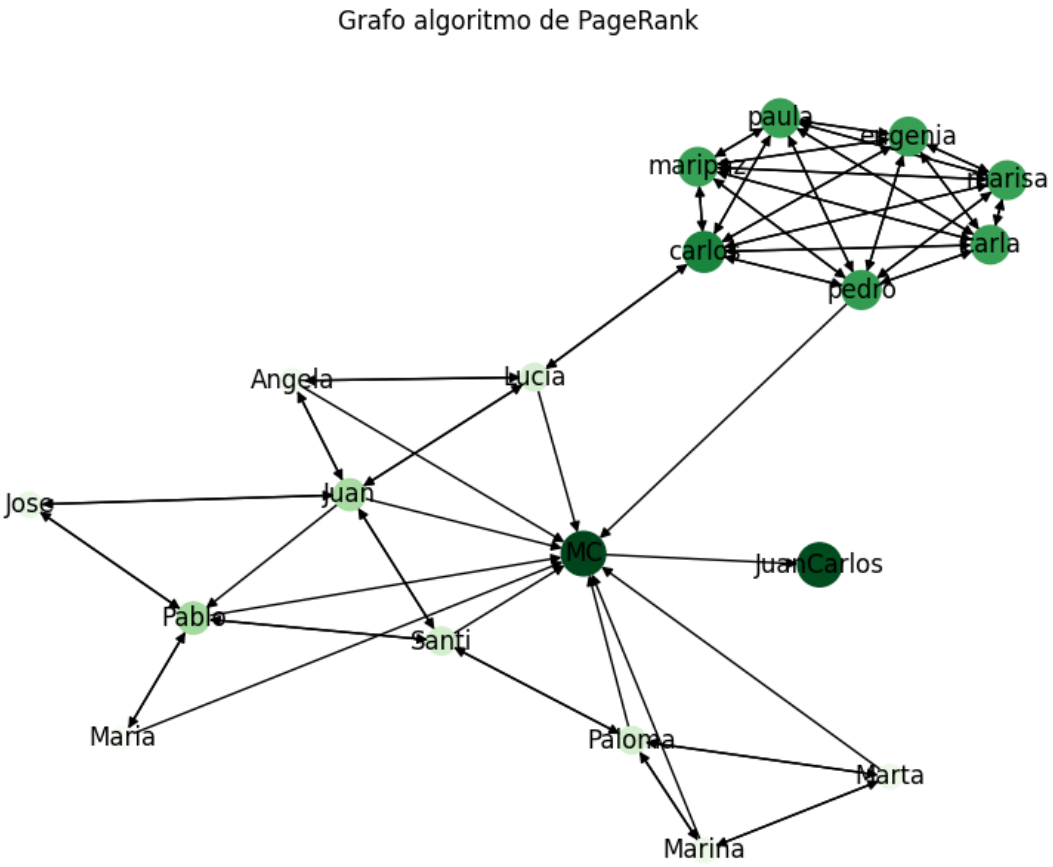


Figura 4-32. Gráfica análisis PageRank caso 5.

Nodos	Puntuación PR
Nodo MC	0. 09164635814745827
Nodo Juan Carlos	0.08981342075798239
Nodo Carlos	0.07624361537872262
Nodo Pedro	0.0695868482751419
Nodo Carla, Marisa, Eugenia, Paula, Maripaz	0.0683533150026475
Nodo Pablo	0.04772570625892311
Nodo Juan	0.046578757128580714
Nodo Santi	0.03605951428852401
Nodo Lucía	0.03512968825832029
Nodo Paloma	0.034851312401414224
Nodo José	0.028653468364513955
Nodo Marina	0.02695669477491288
Nodo Marta	0.02695669477491288
Nodo Ángela	0.025976643664587476
Nodo María	0.022054702512767387

Tabla 4-5. Resultados caso 5.

5 CONCLUSIONES

En este trabajo, se ha investigado y aplicado los considerados más importantes algoritmos de análisis de grafos, que son PageRank, Betweenness Centrality y el algoritmo de detección de comunidades de Louvain. Estos algoritmos han demostrado ser herramientas muy poderosas para analizar y comprender el funcionamiento y estructura dinámica de los grafos, tanto para el análisis del grafo de una ciudad en cuanto a su distribución en comunidades como para hallar los puntos más importantes según su posición, como para entender la importancia de las relaciones en las redes sociales según la influencia de los demás.

El algoritmo de PageRank ha sido ampliamente utilizado para medir la importancia y relevancia de los nodos en una red, pues gracias a su algoritmo nos ha permitido identificar los nodos más influyentes y populares en una red. Al aplicar este algoritmo en nuestro ejemplo, hemos obtenido una clasificación de nodos basados en su importancia, obteniendo así la identificación de las personas más relevantes y destacadas en nuestro grafo.

Otro algoritmo fundamental que hemos explorado profundamente ha sido el de Betweenness Centrality, que mide la importancia de un nodo en función de su participación en la conexión de otros nodos dentro del grafo de la ciudad de Sevilla. Al calcular el Betweenness Centrality, hemos podido identificar aquellos nodos que actúan como puntos más críticos en la comunicación y estructura de la ciudad de Sevilla entre las diferentes partes del grafo, lo que nos ha permitido comprender mejor cómo es la distribución de la ciudad de Sevilla en cuanto a este aspecto.

Además de aplicar el algoritmo de Betweenness Centrality al grafo de la ciudad de Sevilla, hemos aplicado el algoritmo de detección de comunidades de Louvain para identificar las principales comunidades, es decir, las acumulaciones de nodos que están muy interconectados entre ellos por cada zona. Gracias a este algoritmo, hemos demostrado su efectividad en la identificación de comunidades en el grafo que eran los principales barrios de la ciudad.

En resumen, este estudio ha demostrado la utilidad y aplicabilidad de los algoritmos de análisis de grafos como son los algoritmos de PageRank, Betweenness Centrality y el de detección de comunidades de Louvain. Estos algoritmos nos han demostrado una visión más profunda y detallada de la estructura e importancia de la conexión de los nodos y aristas de los grafos, se ha identificado los nodos más importantes y destacados, identificando sus puntos más críticos en cuanto a su conexión y se ha obtenido las comunidades en el grafo de la ciudad de Sevilla.

Estos resultados son valiosos para comprender mejor las interacciones y las relaciones de nuestros grafos ya que pueden tener implicaciones significativas en campos como el análisis de redes sociales o la optimización del tráfico en una ciudad, por lo que los algoritmos de análisis de grafos se afirman como herramientas esenciales para obtener una mejor comprensión de estas estructuras interconectadas.

Como líneas futuras de la teoría de grafo, se propone el análisis de grafos de gran tamaño para así hallar la detección de comunidades, utilizando algoritmos más eficientes que reduzcan los tiempos de ejecución o el uso de menos recursos, y para ello, se podría usar diferentes técnicas de optimización como la eliminación de ramas innecesarias o la reestructuración del grafo de una forma más eficiente.

REFERENCIAS

[1] M. v. Bell, "Highlights from the History of Graph Theory," April 2015.

[2] F. C. y. J. Amat, "Detección de comunidades en grafos y redes con python," Abril 2023. [Online]. Available: https://www.cienciadedatos.net/documentos/pygml02-detecion-comunidades-grafos-redes-python.html. [Accessed 19 Abril 2023].

[3] S. Cruces, Graph and Network Analytics, Universidad de Sevilla, 2022.

[4] M. N. &. A. E. Hodler, A Comprehensive Guide to Graph Algorithms un Neo4j, neo4j, 2021.

[5] S. G. Prado, "Algoritmos para Big Data: Grafos y PageRank," Universidad de Valladolid, Valladolid, 2017.

[6] R. J. Gould, Graph Theory, 1988.

[7] M. L. Mamani, "DFS vs BFS. Ancora," 25 Mayo 2020. [Online]. Available: https://www.encora.com/es/blog/dfs-vs-bfs. [Accessed 31 Mayo 2023].

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

[9] O. C. García, "Redes y Sistemas Complejos," Universidad de Granada, 2019.

[10] O. C. García, "Modularidad, particionamiento y comunidades," in *Redes y sistemas complejos*, Granada, Ugr, 2019, p. 72.

[11] L. W. N. J. v. E. Vincent Traag, "Usando el algoritmo de Leiden para encontrar grupos bien conectados en redes," 21 11 2019. [Online]. Available: http://ars-uns.blogspot.com/2019/11/comunidades-el-algoritmo-de-leiden.html. [Accessed 16 04 2023].

[12] J. F. X. S. J. F. Jicun Zhang, "An Improved Louvain Algorithm for Community Detection," 2021. [Online]. Available: https://doi.org/10.1155/2021/1485592.

[13] D. &. Z. G. Perrin, "Recursive module extraction using Louvain and PageRank," F1000Research, 2018.