Universidad de Cantabria
Facultad de Ciencias

Trabajo fin de grado

Algoritmos heurísticos para la búsqueda de grafos contrafactuales en redes cerebrales

Mónica Zhang Wang

Grado en Ingeniería Informática

Supervisado por

Camilo Palazuelos Calderón
DPTO. MATEMATICAS, ESTADISTICA Y COMPUTACION

Junio 2023

Agradecimientos

Quisiera expresar mi agradecimiento a todas las personas que me han ayudado y apoyado tanto físicamente como moralmente en el desarrollo de este trabajo.

Agradezco especialmente a mi familia y amigos, por el apoyo incondicional y comprensión, mostrando una gran confianza constante hacia mi durante todo el proceso y ayudarme a llevar a cabo este trabajo.

En segundo lugar, a mi supervisor Camilo Palazuelos Calderón, por su dedicación, orientación e inapreciable ayuda en todo momento. Gracias por compartir sus conocimientos y experiencias, y por animarme en momentos complicados.

Por último, a todos los profesores que me han aportado conocimientos imprescindibles para llegar a este punto de la carrera y a la propia Universidad de Cantabria por ofrecerme la oportunidad de realizar este trabajo.

Índice

1. Introducción **7**
1.1. Motivación 7
1.2. Objetivos 7
1.3. Entorno de trabajo 8

2. Conceptos teóricos **9**
2.1. Teoría de grafos 9
2.1.1. Grafo completo 10
2.1.2. Grafo complementario 10
2.1.3. Matriz de adyacencia y sus operaciones 10
2.2. Conceptos utilizados en los algoritmos 12
2.2.1. Grafo contrafactual 12
2.2.2. Diferencia simétrica 12
2.2.3. Centralidad de intermediación 13
2.3. Clasificador de grafos 14
2.4. Declaración formal del problema 14

3. Algoritmos de Abrate y Bonchi **15**
3.1. *Oblivious Forward Search* (OFS) 15
3.1.1. Diseño 15
3.1.2. Implementación 17
3.1.3. PICK de OFS 18
3.2. *Oblivious Backward Search* (OBS) 18
3.2.1. Diseño 18
3.2.2. Implementación 19
3.2.3. PICK de OBS 21

4. Algoritmos alternativos propuestos **22**
4.1. Diseño 22
4.2. Implementación 23

5. Análisis de resultados **26**
5.1. Conjunto de datos 26
5.2. Preprocesado 28
5.3. Redes procesadas por los algoritmos OFS1 y OFS2 29
5.4. Distancia de edición 31
5.4.1. Distribución de la distancia de edición 31
5.4.2. Distribución acumulada de la distancia de edición 33
5.4.3. Diferencia de distancia entre OFS1+OBS y OFS2+OBS . . 33

5.4.4. Significación estadística 34

6. Conclusiones y líneas futuras **36**
6.1. Conclusión 36
6.2. Líneas futuras 36
6.2.1. Utilización de otros clasificadores de grafos 36
6.2.2. Comportamiento entre OFS1 y OFS2 37

Índice de figuras

1. Grafo con 6 vértices y 7 aristas 9
2. Grafos completos 10
3. Grafo complementario 10
4. Matriz de adyacencia 11
5. Diferencia simétrica 13
6. Betweenness centrality 14
7. Clasificación de los grafos 27
8. Número de redes procesadas 30
9. Distancia de edición en densidad 32
10. Distancia de edición en proporción 33
11. Distancia total 34
12. Distancia de edición promedio 35

Resumen

En neurociencias, los grafos contrafactuales se utilizan para explicar las decisiones tomadas en la clasificación de redes cerebrales y los factores esenciales que influyen en los resultados de dicha clasificación.

El objetivo de este trabajo es implementar dos algoritmos heurísticos para la búsqueda de grafos contrafactuales en redes cerebrales. Nuestra propuesta parte de las ideas de los algoritmos diseñados por Abrate y Bonchi (2021). Estos algoritmos tienen como objetivo encontrar un grafo contrafactual a partir de un grafo inicial dado como entrada, utilizando una búsqueda heurística hacia adelante y hacia atrás respectivamente. La finalidad es encontrar un grafo que tenga una gran similitud estructural con el grafo original, pero que sea clasificado en una clase diferentes por un clasificador de caja negra.

El primer algoritmo añade y elimina aristas gradualmente para encontrar un primer grafo contrafactual. El segundo algoritmo realiza el mismo método para disminuir la distancia entre el grafo contrafactual encontrado y el grafo original, con el propósito de encontrar un grafo que minimice la distancia de edición con el grafo dado.

Finalmente, se analizan los algoritmos mencionados en términos de la distancia de edición, el número de redes procesados por ambos algoritmos y el promedio de las aristas entre los grafos procesados. Estos análisis proporcionan información sobre la eficiencia de los algoritmos implementados y la calidad de los grafos contrafactuales generados.

Abstract

In neuroscience, counterfactual graphs are used to explain the decisions made in a classification based on brain activities and the significant factors that influence the outcomes of such classification.

The core of this project involves implementing two heuristic algorithms for the search of counterfactual graphs in brain networks. Our proposal is based on the ideas of the algorithms designed by Abrate and Bonchi (2021). These algorithm aim to find a counterfactual graph from an initial graph provided as input using forward and backward heuristic search, respectively. The goal is to discover a graph that closely resembles the original one but is classified differently by the black-box classifier.

The first algorithm gradually adds and removes edges in order to obtain a contrafactual graph. The second one employs the same approach to decrease the distance between the contrafactual graph found and the original graph. Its purpose is to discover an optimal graph for the given input graph.

Finally, the mentioned algorithms will be analyzed in terms of edit distance, the number of networks processed by both algorithms, and the average number of edges between the processed graphs. These experiments will provide information about the efficiency of the implemented algorithms and the quality of the generated contrafactual graphs.

1. Introducción

1.1. Motivación

En los últimos años, la *conectómica cerebral*, una disciplina de la neurociencia que representa cómo las diferentes regiones del cerebro están interconectadas entre sí a través de redes cerebrales, ha surgido como un modelo con intención de ayudar a entender los procesos mentales y las enfermedades cerebrales [8].

La importancia de conocer el conectoma del cerebro es que permitirá entender cómo los estados funcionales del cerebro pueden surgir de la estructura subyacente y cómo se puede ver afectada por daños en la misma [8].

Gracias a la representación gráfica de redes cerebrales, varios problemas de la neurociencia han sido reformulados como problemas de minería de grafos. Esta estructura de los datos tiene un diversas aplicaciones en varios dominios como la química, la biología, las redes sociales, etc. Se trata de una herramienta potente para el aprendizaje automático en grafos. Asimismo, permite mostrar una visión clara y directa entre las redes cerebrales de dos pacientes, por ejemplo, de uno que padece de autismo y otro que no, para observar las diferencias que tienen ambas.

1.2. Objetivos

Este trabajo se centra en la búsqueda heurística de grafos contrafactuales basado en un clasificador binario de caja negra. Se trata de un problema computacional, propuesto por Abrate y Bonchi [1], que, dados un grafo G y un clasificador de grafos binario f, pretende encontrar un grafo H que sea clasificado en la clase contraria a G, es decir, $f(H) = 1 - f(G)$, y minimizar la distancia de edición con G.

El problema puede ser aplicado en la neurociencia, por ejemplo, en la clasificación de redes cerebrales. En este contexto, se dan dos grupos de individuos, caso y control de una enfermedad de bajo estudio. Cada individuo es representado mediante una red cerebral, un grafo sobre un conjunto de vértices semánticamente significativos (correspondientes a regiones del cerebro). Así, se pueden descubrir subestructuras asociadas con fenotipos cognitivos específicos o, dada una red cerebral, proporcionar información relevante para identificar características concretas que influyen en la decisión de clasificación.

Abrate y Bonchi [1] introdujeron los primeros algoritmos heurísticos para la búsqueda de grafos contrafactuales. El objetivo general de este Trabajo Fin de Grado es proponer algoritmos heurísticos alternativos que mejoren los enfoques existentes

al minimizar tanto la distancia de edición como el número de iteraciones necesario para obtener la solución. Con este fin, se establecen tres objetivos específicos:

1. Implementar los algoritmos de Abrate y Bonchi.
2. Modificar los algoritmos anteriores para guiar la búsqueda de grafos contrafactuales en función de la importancia de las aristas (en vez de al azar) e implementar los cambios.
3. Evaluar y comparar el desempeño de los algoritmos de los puntos previos en términos de la optimización en distancia de edición y el número de redes que son capaces de procesar.

Como enfoque en el objetivo 2, se propone partir del concepto de las medidas de centralidad, en concreto, del valor de intermediación de las aristas (*edge betweenness*). Respecto al análisis de estos algoritmos en el objetivo 3, se hace uso de un conjunto de 520 redes cerebrales reales, de las cuales 190 se corresponden con individuos con trastorno por déficit de atención con hiperactividad y 330 a controles.

Respecto a la estructura de la memoria, se explicarán los conceptos teóricos en el capítulo 2, donde estará contenido por la teoría de grafos en la sección 2.1. Además de algunos conceptos que serán utilizados en los algoritmos en la sección 2.2 como la definición de un grafo contrafactual. Una vez explicados los conceptos fundamentales, se procederá a exponer los algoritmos de Abrate y Bonchi en el capítulo 3 y nuestra propuesta de un algoritmo alternativo en el capítulo 4. En ambos apartados, se explicará en detalle el diseño y la implementación de los algoritmos. Finalmente, en el capítulo 5 se demostrarán los resultados obtenidos de los análisis realizados y concluir en el capítulo 6 planteando algunas posibles líneas futuras de investigación.

1.3. Entorno de trabajo

En este apartado se describirá el entorno de trabajo para el desarrollo de este proyecto, tanto las partes del hardware como del software utilizadas. Comenzando por la infraestructura hardware, la parte física del equipo, se ha utilizado un computador con las siguientes características:

- Windows 10 Pro de 64 bits
- Procesador Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz con 6 cores principales y lógicos
- 16 GB de memoria RAM

- Red Ethernet 100 Mbps

A través de este computador, se utiliza el software *Visual Studio Code* como entorno de desarrollo integrado para el desarrollo de los algoritmos que se programarán en el lenguaje de Python. Se ha elegido Python fundamentalmente por la amplia variedad de librerías disponibles en su biblioteca estándar, además de su predominancia en el ámbito de la Inteligencia Artificial, que corresponde con el tema central de nuestro trabajo.

2. Conceptos teóricos

En este capítulo se abordan los conceptos fundamentales de la teoría de grafos que han sido aplicados en este Trabajo Fin de Grado y resultan esenciales para comprender el diseño de los algoritmos.

2.1. Teoría de grafos

Para entender los grafos contrafactuales, es importante comprender qué es un grafo. En pocas palabras, es un conjunto de nodos (o vértices) que se interconectan mediante aristas para representar relaciones binarias entre elementos de un conjunto [5].

Formalmente, un grafo G es definido por $G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de aristas del grafo. $|V|$ refiere al número de vértices de un grafo, llamado orden, y el grado de un vértice es el número de aristas en común con otros vértices. Una arista puede representarse de la forma (i, j) donde i y j son los nodos que están conectados entre sí.

En la figura 1 se muestra un ejemplo de un grafo para comprender mejor la teoría. Este grafo contiene seis vértices y siete aristas, por tanto $|V| = 6$ y el grado del vértice 5 por ejemplo, sería 3 al tener tres aristas entrantes y salientes, pues este ejemplo se trata de un grafo.

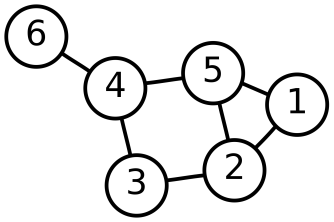
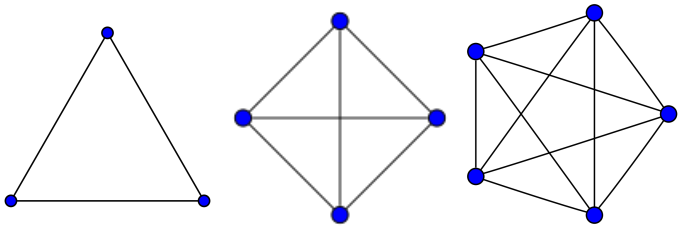


Figura 1: Grafo con 6 vértices y 7 aristas

2.1.1. Grafo completo

Asimismo, en la teoría de grafos pueden incluirse los grafos completos, en los que cada par de nodos debe estar conectado entre sí, es decir, cada vértice debe estar unido con todos los demás. Un grafo completo con n vértices tiene $\frac{n(n-1)}{2}$ aristas y se denota como K_n [5].



(a) K_3 (b) K_4 (c) K_5

Figura 2: Ejemplos de grafos completos [7]

2.1.2. Grafo complementario

Dado un grafo $G = (V, E)$, su grafo complementario es $G_c = (V, K \setminus E)$, donde K es el conjunto de todas las aristas posibles. G_c está formado por el mismo conjunto de vértices, y el conjunto de aristas $K \setminus E$ consta de todas aquellas aristas que no están en G.

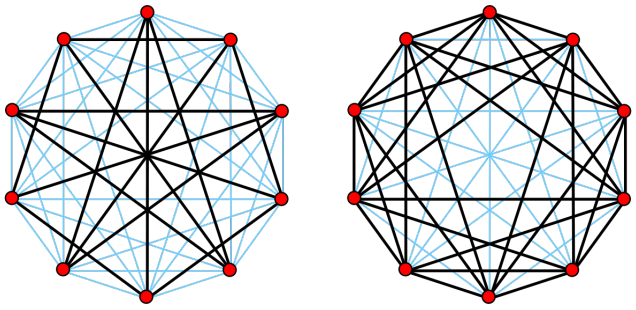
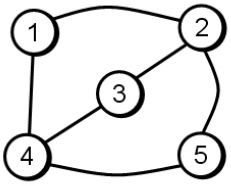


Figura 3: Grafo de Petersen (izquierda) y su grafo complementario (derecha) [7]

2.1.3. Matriz de adyacencia y sus operaciones

Una parte importante de nuestro trabajo es que los grafos están representados por matrices de adyacencia. Una matriz de adyacencia es una matriz cuadrada, es decir, el número de filas y columnas es igual, que se utiliza normalmente para la representación de grafos [5].

La matriz de adyacencia debe de ser simétrica, pues si dos nodos están conectados mediante una arista, el valor de la celda (i, j) debe ser el mismo que el de la celda (j, i).



M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Figura 4: Un grafo (izquierda) y su matriz de adyacencia (derecha) [4]

Respecto a la representación de la conexión entre dos nodos, por ejemplo, los nodos 1 y 2, el valor de las celdas $(1, 2)$ y $(2, 1)$ es 1 para indicar que están conectados. Asimismo, en este contexto no estaríamos permitiendo que un nodo tenga como extremo a sí mismo, y en consecuencia, se establecen todos los elementos de la diagonal de la matriz a valor 0.

Las operaciones fundamentales que se realizan sobre las matrices de adyacencia son añadir y eliminar las aristas del grafo representado. Básicamente, es modificar el valor de una celda (i, j) de 0 a 1 si se quiere añadir una arista y en caso contrario, eliminar. Este paso es esencial en la implementación de algoritmo porque es la única forma en la que nos desplazamos hacia el grafo contrafactual desde el grafo original.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

En este ejemplo podemos observar que la matriz de adyacencia, empezando a contar la primera fila o columna, ha tenido modificaciones en las celdas $(2, 4)$ y $(3, 4)$ (y $(4, 2)$ y $(4, 3)$ por mantener la simetría de la matriz). Por tanto, originalmente $(2, 4) = 1, (3, 4) = 0$ pasa a ser $(2, 4) = 0, (3, 4) = 1$, es decir, se ha eliminado la primera arista y se ha añadido la segunda al grafo correspondiente. Y así, es como se va a trabajar con los grafos durante todo el proceso de la implementación del algoritmo.

2.2. Conceptos utilizados en los algoritmos

En este apartado se explicarán los conceptos teóricos fundamentales necesarios para entender cómo se ha enfocado la implementación de los algoritmos y por qué se han utilizado unas funciones determinadas para llevar a cabo este proceso.

2.2.1. Grafo contrafactual

Dados un grafo G y un clasificador de grafos binario f, un grafo contrafactual H es aquel que f clasifica en la clase contraria a G, es decir, $f(H) = 1 - f(G)$. Cuando se refiere al contexto de la clasificación de redes cerebrales, un grafo contrafactual es creado mediante la modificación de enlaces (aristas) de un grafo inicial dado. Simplemente añadiendo una nueva arista o eliminando una existente, se podría obtener un nuevo grafo similar al original, pero clasificado diferentemente por el clasificador f.

En este Trabajo Fin de Grado, cada red es definida como un grafo no dirigido y sin ponderar (sin peso en las aristas). Además, solamente se consideran aquellos que están definidos sobre un mismo conjunto de vértices y es identificado mediante el conjunto de aristas $E \subseteq V^2$.

Respecto a la implementación de los algoritmos, se utiliza un concepto fundamental que es la diferencia simétrica en OBS y como objetivo del proyecto, introducimos un nuevo enfoque de utilizar *betweenness centrality* para desarrollar el algoritmo OFS.

2.2.2. Diferencia simétrica

La diferencia simétrica entre dos conjuntos A y B es la unión de ambos menos su intersección, es decir,

$$A \Delta B = (A \cup B) \setminus (A \cap B),$$

lo que da como resultado otro conjunto con todos los elementos excepto aquellos que tienen en común A y B, que se representa en la figura 5 en color azul.

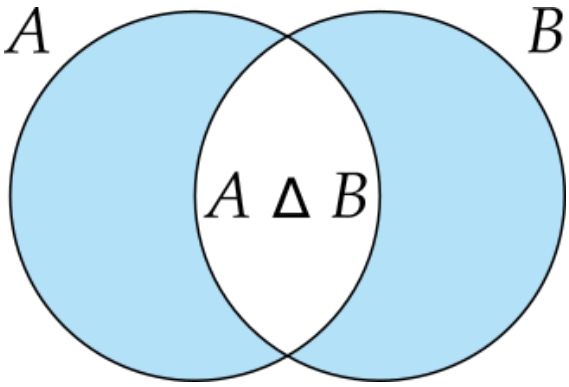


Figura 5: Diferencia simétrica entre dos conjuntos [6]

En el contexto de grafos, la diferencia simétrica entre dos grafos $E \Delta E_c$ es el conjunto de aristas que se necesitan para ir de E a E_c. Esta operación es ideal para encontrar el grafo contrafactual más cercano al grafo original, pues se añaden aquellas aristas que están en E y se eliminan las que están en E_c para ir disminuyendo la distancia (aristas diferentes en ambos grafos) en cada iteración.

2.2.3. Centralidad de intermediación

Proponemos el uso de esta nueva medida de centralidad porque los algoritmos originales están basados en búsquedas al azar. Aunque ambos estrategias son capaces de producir grafos contrafactuales destacados, nuestra intención es introducir este nuevo enfoque para orientar la selección de aristas a un patrón específico, en vez de forma aleatoria.

En la teoría de grafos (véase la sección 2.1), la medida de centralidad basada en los caminos más cortos del grafo, conocida como centralidad de intermediación, nos proporciona información sobre la importancia de una arista en términos de la influencia de un vértice en la comunicación con otros vértices de la red. La modificación que realizamos de los algoritmos originales basa en la selección de aristas en su centralidad de intermediación. Esta modificación permite tener en cuenta la importancia de las aristas en la búsqueda de un grafo contrafactual óptimo.

Para obtener un grafo contrafactual de forma eficiente, es crucial ir eliminando las aristas de mayor intermediación del grafo original e ir añadiendo aquellas de mayor intermediación del grafo complementario. Al priorizar las aristas con mayor intermediación, se busca maximizar el impacto de los cambios realizados en el grafo actual para que sea clasificado en la clase opuesta del original.

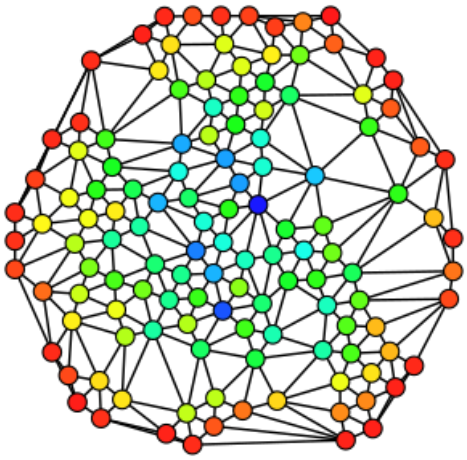


Figura 6: Representación de un grafo basado en la medida de centralidad de cada vértice desde menos significativo (rojo) hasta el más significativo (azul) [3].

2.3. Clasificador de grafos

Respecto a cómo diferenciar si el grafo encontrado es un grafo contrafactual, nos basamos en un clasificador de caja negra. Esta herramienta se proporciona como parámetro al algoritmo y su diseño, por lo tanto, queda fuera de los objetivos de este Trabajo Fin de Grado. Sin embargo, se proporciona a continuación una formalización de las características de este clasificador f.

El funcionamiento del clasificador es simple y se define de la siguiente manera:

$$f : G(V) \to \{0, 1\},$$

donde $G(V)$ es el conjunto de los posibles grafos cuyo conjunto de vértices es V.

Se asume que no se tiene ninguna información respecto f y sólo se pueden realizar llamadas a f, donde se le proporciona un grafo determinado y retorna la predicción de la clase.

2.4. Declaración formal del problema

Representaremos el grafo $G = (V, E)$ en función de su conjunto de aristas E. El problema planteado, mencionado anteriormente en la sección 1.2, es la búsqueda de un grafo E' que sea lo más próximo posible al grafo original E. Como distancia de edición, se define la siguiente función: $d(E, E') = |E \Delta E'|$, donde Δ denota la diferencia simétrica entre dos conjuntos (véase la sección 2.2.2).

Formalmente, se puede definir el problema de la siguiente forma: Dado un clasificador en caja negra $f : G(V) \to \{0, 1\}$ y un grafo $E \in G(V)$, encontrar un grafo contrafactual $E^* \in G(V)$ tal que:

$$E^* = \underset{\substack{E' \in G(V) \\ f(E') = 1 - f(E)}}{\operatorname{argmin}} \; d(E, E')$$

Es decir, lo deseable es encontrar un grafo contrafactual E' cuya distancia con E sea la mínima posible y que esté clasificado en la clase contraria al original E $f(E') = 1 - f(E)$. Para la resolución de este problema, se pone en práctica los algoritmos OFS y OBS, que se explicará en profundidad en el apartado 3.

3. Algoritmos de Abrate y Bonchi

Este capítulo constituye la parte práctica del trabajo, en el que se explicará con detalle el diseño de los algoritmos propuestos por los autores del artículo [1] y las decisiones tomadas en su implementación. Además, nuestra implementación alternativa al primer algoritmo en la que se intentan mejorar los resultados obtenidos.

Los algoritmos de Abrate y Bonchi conforman un método heurístico de búsqueda local bidireccional, por tanto, consta de dos fases que están ligadas entre sí, pues el segundo algoritmo utiliza el grafo contrafactual obtenido en el primer algoritmo como parámetro. El objetivo es encontrar un primer grafo contrafactual dado un grafo (OFS) y conseguir otro óptimo desde este grafo encontrado (OBS).

En estos algoritmos, como se ha mencionado anteriormente en la sección 2.1.3, los grafos son representados en matrices de adyacencia para trabajar de forma fácil computacionalmente, además, hacer que las ejecuciones sean eficientes en complejidad. A continuación, se procede a la explicación detallada del diseño e implementación de los algoritmos y finalmente, nuestra versión del OFS con el nuevo enfoque propuesto.

3.1. *Oblivious Forward Search* (OFS)

3.1.1. Diseño

El objetivo de este algoritmo es encontrar un primer grafo contrafactual E_c dado un grafo original E. Para ello, realiza iteraciones sobre el grafo E y va modificando sus aristas mediante las operaciones de adición y eliminación que son explicados anteriormente en la sección 2.1.3.

Este algoritmo depende del número máximo de iteraciones η para recorrer el bucle general hasta encontrar el grafo contrafactual, es decir, su condición de permanen-

cia es no superar este número y que ambos grafos sigan clasificados en la misma clase. En el bucle interno es donde se ejecutan las operaciones sobre el grafo E', dependiendo de una variable aleatoria para gestionar la operación que realiza. Esta operación se realiza k veces, que corresponde al número de aristas que se van a modificar en cada iteración. Básicamente se añaden o se eliminan aristas al grafo E' con el fin de encontrar el grafo contrafactual. En cada iteración, la arista debe ser añadida a una lista de "explorados" para no modificar una arista más de una vez y producir inconsistencias.

Algorithm 1 Oblivious Forward Search (OFS)

Require: $E \in G(V), \eta \in \mathbb{N}^+, k \in \mathbb{N}^+$, clasificador caja negra $f : G(V) \to \{0, 1\}$
Ensure: $E_c \in G(V)$ tal que $f(E_c) = 1 - f(E)$ o error
  $i \leftarrow 0$; $E' \leftarrow E$; $L \leftarrow \emptyset$
  while $f(E') = f(E)$ **and** $i < \eta$ **do**
    $i \leftarrow i + 1$, $j \leftarrow 0$
    while $j < k$ **do**
      $j \leftarrow j + 1$
      if RAND() $< 0{,}5$ **then**
        $edge \leftarrow \text{PICK}(1, V^2 \setminus (E \cup L))$
        $E' \leftarrow E' \cup edge$; $L \leftarrow L \cup edge$
      else
        $edge \leftarrow \text{PICK}(1, E \setminus L)$
        $E' \leftarrow E' \setminus edge$; $L \leftarrow L \cup edge$
      end if
    end while
  end while
  if $f(E') = 1 - f(E)$ **then**
    return E'
  else
    return *error*
  end if

En resumen, el algoritmo modifica una copia del grafo original E' mediante las operaciones de adición y eliminación de aristas. El objetivo principal es ir añadiendo aristas que no están presentes en el grafo original E e ir eliminando aquellas que sí están pero no han sido exploradas. Esto permite conseguir un grafo contrafactual de manera efectiva y devolverlo como resultado. Sin embargo, en caso de que no sea posible encontrar un grafo contrafactual en las iteraciones propuestas, el algoritmo devuelve un error.

3.1.2. Implementación

En esta sección se explicarán algunas decisiones tomadas en la implementación del algoritmo. Se reciben como parámetros de entrada el grafo inicial E y el clasificador f. Además de dos variables η y k que corresponden a la limitación del número máximo de iteraciones totales que ejecuta el algoritmo y al máximo número de aristas que se modifican por cada iteración, respectivamente.

Es de gran importancia que la copia que se realiza inicialmente en $E' \leftarrow E$ sea una copia profunda, porque no queremos realizar ninguna modificación sobre el grafo original, ya que podría afectar posteriormente en la selección de aristas.

Respecto a las variables i y L, estas indican el número de iteraciones que el algoritmo ha recorrido en el momento actual y la lista de aristas que han sido seleccionadas durante el proceso, respectivamente. Esta variable L se inicializa con un conjunto vacío $\emptyset$. Además, L debe de ser un conjunto (*set*), ya que necesitamos almacenar aristas únicas, no repetidas.

Una vez inicializadas todas las variables necesarias, se entra en un primer bucle, que es el bucle general para que el algoritmo vaya buscando el grafo contrafactual en las iteraciones indicadas η. Dentro de este bucle se define la variable j para saber el número de iteraciones que se han realizado y se seleccionan las k aristas, donde la operación que se realiza depende de un número entero aleatorio entre 0 y 1, que se obtiene mediante la función *random.randint*.

En el primer caso, cuando $rand = 0$, se selecciona una arista a partir de todas aquellas que no tiene el grafo E y la lista L. La nomenclatura V^2 se refiere a un grafo completo (véase la sección 2.1.1) y en $V^2 \setminus (E \cup L)$ estarán contenidas todas las aristas posibles quitando las aristas obtenidas mediante la unión de E y L. Esto es principalmente para añadir una arista desconocida y aumentar la probabilidad de que el grafo E' sea clasificado en la clase contraria. Por lo tanto, después de haber seleccionado la arista, se añade esta arista al grafo E' y a la lista de aristas "exploradas" L para evitar que se introduzcan aristas que se han agregado previamente en las siguientes iteraciones.

En caso contrario, se selecciona una arista entre las aristas del grafo original E que no esté seleccionada anteriormente para ser eliminada del grafo E'. Utiliza la función *pick* con el parámetro de entrada $E \setminus L$ para indicar las aristas deseadas. Esta arista se elimina del grafo E' y también se añade a la lista L por el mismo motivo mencionado.

Finalmente, si el algoritmo es capaz de encontrar un grafo contrafactual, sale del bucle general y al cumplir la primera condición del if $f(E') = 1 - f(E)$, retorna el grafo E' generado. En caso contrario, sale igualmente del bucle al llegar el número

máximo de iteraciones del bucle η y retorna error, que en caso de Python, sería *None*, es decir, un valor nulo para indicar que no se ha podido encontrar el grafo contrafactual dado el grafo original E.

3.1.3. Pick de OFS

Esta función realiza la selección de aristas en el algoritmo, donde se le indican las aristas a seleccionar y un grafo representado en matriz de adyacencia en la que se proporciona una lista de aristas candidatas para elegir. En nuestro caso, hemos definido como parámetro de entrada, una matriz de adyacencia cualquiera G y la operación *op* que se va a realizar, eliminar (0) o añadir (1).

Lo primero que se realiza es elegir de forma aleatoria una celda (i, j) como punto de comienzo, y se va recorriendo la matriz de adyacencia, en filas desde el mismo. En el caso de que se llega a la última fila y columna de la matriz y no ha sido posible encontrar una arista existente, entonces comienza nuevamente desde el principio hasta el punto de comienzo. En el caso de que tampoco haya podido encontrar una posible arista en este segundo bucle, el método retorna vacío finalmente.

3.2. *Oblivious Backward Search* (OBS)

3.2.1. Diseño

Este segundo algoritmo se basa en el grafo contrafactual encontrado del algoritmo anterior OFS, que es denotado como E_c^1. El objetivo principal es desplazar hacia el grafo original E y aproximarse lo máximo posible, pero sin llegar a situarse en la misma clasificación. Es decir, se intenta conseguir un grafo contrafactual óptimo en términos de distancia de edición, mediante modificaciones en sus aristas de la misma manera que OFS.

En este caso, el algoritmo va recorriendo el bucle mientras no se haya superado el número máximo de iteraciones η o no se haya terminado de procesar la lista de aristas candidatas. Estas aristas son obtenidas mediante la diferencia simétrica (véase la sección 2.2.2) entre los grafos E y E_c.

Para cada iteración, E_c^i mantiene el mejor grafo contrafactual encontrado hasta el momento y se van añadiendo o eliminando aristas del grafo E_c para generar este grafo candidato E_c^i. En la ejecución del algoritmo, pueden ocurrir tres casos en las que se puede situar cada iteración:

Condición	Descripción
$f(E_c^i) = 1 - f(E)$	El grafo producido en la iteración (grafo candidato) sigue sin cruzar el borde de la clasificación, por tanto, se incrementa k para la siguiente iteración y se actualizan $E_c \leftarrow E_c^i$ y $E_d \leftarrow E \Delta E_c$.
$k > 1$	El grafo ha cruzado el borde, es decir, ya se ha situado en la clase opuesta y por tanto, es necesario reducir k para la siguiente iteración y volver a probar E_c^i con otra selección de aristas.
$k = 1$	Se deja de decrementar k y se eliminan las aristas que ya han sido analizadas de la lista de candidatos E_d, pues al estar probando las aristas una por una, no tendría sentido comprobar múltiples veces una misma.

Algorithm 2 Oblivious Backward Search (OBS)

Require: $E, E_c^1 \in G(V), \eta, k \in \mathbb{N}^+$, clasificador en caja negra $f : G(V) \to \{0, 1\}$
Ensure: $E_c \in G(V)$ tal que $f(E_c) = 1 - f(E)$
  $E_c \leftarrow E_c^1$; $i \leftarrow 0$
  $E_d \leftarrow E \Delta E_c$
  while $i < \eta$ **and** $|E_d| > 0$ **do**
    $i \leftarrow i + 1$, $k \leftarrow \text{MIN}(k, |E_d|)$
    $E_d^k \leftarrow \text{PICK}(k, E_d)$
    $E_c^i \leftarrow E_c \Delta E_d^k$
    if $f(E_c^i) = 1 - f(E)$ **then**
      $k \leftarrow k + 1$; $E_c \leftarrow E_c^i$; $E_d \leftarrow E \Delta E_c$
    else if $k > 1$ **then**
      $k \leftarrow k - 1$
    else
      $E_d \leftarrow E_d \backslash E_d^k$
    end if
  end while
  return E_c

3.2.2. Implementación

El enfoque es justificado por la observación fundamental de que al modificar E_c^1 mediante la adición y eliminación de aristas a partir de la diferencia simétrica con E, se garantiza que se obtiene un grafo E' con una distancia menor. En otras

palabras, se cumple la siguiente desigualdad: $d(E', E) < d(E_c^1, E)$ e implica que el grafo contrafactual E' está más cercano al grafo original E.

La copia del grafo $E_c \leftarrow E_c^1$ es también una copia profunda, de la misma manera que OFS, para no modificar los valores del E_c^1 original. En este caso no se define una lista de aristas seleccionadas L, sino que se trabaja con una lista de candidatos E_d inicializado mediante la diferencia simétrica $E \Delta E_c$. Estas aristas son las que van a ser seleccionadas posteriormente para añadir o eliminar del grafo a través de la función *pick*.

El algoritmo tiene un único bucle que itera sobre un grafo candidato E_c^i y se van sacando aristas de la lista E_d. En cada iteración, en E_c se almacena siempre el grafo contrafactual óptimo actual, pues la idea es ir modificando el grafo candidato. Si el grafo cruza la línea y pasa a clasificarse en la clase del grafo original E, vuelve al estado del grafo E_c para intentar con diferentes aristas. En caso contrario, es decir, si consigue un grafo contrafactual mejor que el actual, se actualiza E_c al grafo candidato encontrado ($E_c \leftarrow E_c^i$).

Profundizando, lo primero que se realiza es definir el número de aristas que se van a seleccionar, que es el mínimo entre la variable k introducida como parámetro y la longitud de la lista E_d. En la mayoría de los casos, $k = min(k, |E_d|)$ porque lo más probable es que E_d tenga almacenada una gran cantidad de aristas candidatas. Este número k se le introduce luego a la función *pick* para seleccionar las k aristas de la lista E_d y realizar la diferencia simétrica con E_c.

La diferencia simétrica $E_c \Delta E_d^k$ funciona de diferente manera computacionalmente, porque es en realidad entre una matriz de adyacencia y una lista de aristas, pero la idea es la misma. Se define un grafo E_c^i y, para cada arista de la lista E_d^k que no esta presente el grafo E_c, se añade al grafo E_c^i; en caso contrario, se elimina. Es decir, E_c^i contiene todas las aristas que no están en E_c y elimina aquellas que están.

Respecto a la diferencia simétrica entre E y E_c en el primer cuerpo del bloque condicional, se ha implementado una función auxiliar que recibe como parámetro dos grafos representados en matrices de adyacencia. En este caso ambos son grafos, pero necesita que se le devuelva una lista de aristas E_d y no en formato de matriz de adyacencia como antes en E_c^i. Por lo tanto, la función implementada utiliza *abs(graph1-graph2)* para obtener una matriz de adyacencia con todas las aristas que tienen en diferencia los dos grafos, que es justo la definición de diferencia simétrica (véase en la sección 2.2.2). Como consiguiente, se recorre esta matriz en fila y columna para añadir las aristas existentes en una lista y retornarlo finalmente.

Finalmente, el algoritmo finaliza cuando la lista E_d se queda vacía o cuando se

alcanza el número máximo de iteraciones η que es introducido como parámetro y retorna el grafo contrafactual óptimo E_c que ha podido conseguir como resultado.

3.2.3. Pick de OBS

En este algoritmo, la función *pick* se encarga de seleccionar k aristas de forma aleatoria de una lista de candidatos, llamada E_d. Genera un número entero aleatorio entre 0 y la longitud de E_d, el cual se utiliza como índice para acceder a la propia lista. La función selecciona las k aristas correspondientes y las devuelve como resultado final.

En detalle, se emplea un bucle *while* que para la longitud de la lista *edges* sea menor que el número de aristas a seleccionar k, vaya calculando un número entero aleatorio mediante la función *randint(a,b)* que proporciona la librería *random*. Los dos números a y b indican los dos extremos del número aleatorio y es utilizado como índice. Al ser E_d una lista, se añade (add) fácilmente a *edges* (conjunto local para retornar en la función) la arista que se obtiene al acceder $E_d[idx]$. Esta variable *edges* es definido como conjunto simplemente para asegurarnos de que no se añada una arista más de una vez, y lo único que se necesitaría es retornar este conjunto en forma de lista mediante *list(edges)* para mantener la consistencia con el código. Pues este valor es utilizado en E_d^k para calcular la diferencia simétrica con E_c en la siguiente instrucción del algoritmo.

4. Algoritmos alternativos propuestos

En este capítulo se introduce el algoritmo que se ha implementado a partir del concepto de centralidad de intermediación mediante modificaciones al algoritmo original OFS propuesto por Abrate y Bonchi. A continuación, se explicarán el diseño e implementación que se ha llevado a cabo para desarrollar esta versión alternativa, con el fin de superar los resultados obtenidos.

4.1. Diseño

Este algoritmo incluye el concepto de la centralidad de intermediación, para que la selección de aristas esté orientada por este valor de medida y no de forma aleatoria como se ha explicado en los algoritmos anteriores. Sin embargo, sigue manteniendo el objetivo principal de OFS, que es el de encontrar un grafo contrafactual a partir de un grafo inicial E con la ayuda de un clasificador de caja negra.

Se hace uso del grafo complementario del grafo inicial E para obtener la intermediación de las aristas de ambos grafos ordenadas de mayor a menor. Estos conjuntos de aristas serán las seleccionadas para modificar al grafo E'. El diseño es similar al algoritmo original, pero con algunas modificaciones, por ejemplo, el bucle general ahora depende de la longitud de las listas de aristas y no del número máximo de iteraciones.

El bucle interno permanece idéntico, se añaden o eliminan k aristas en cada iteración del bucle general y, en este caso, la probabilidad es distinta por la existencia de una gran diferencia de tamaño entre las dos listas de aristas y depende de las mismas. A diferencia del algoritmo original, que depende de un número binario aleatorio, ahora la dependencia es de un número decimal. Y además, la selección de aristas no se hace uso de la función *pick* sino que se accede directamente mediante un índice a la lista de aristas ordenadas por valor de intermediación.

Tras realizar todas las operaciones necesarias, se retorna finalmente el grafo contrafactual encontrado si se sigue clasificando en la clase opuesta al grafo original o error si no es capaz de encontrarlo.

Las modificaciones realizadas sobre el algoritmo OFS son resaltadas en color rojo en el algoritmo OFS2.

Algorithm 3 Oblivious Forward Search 2 (OFS2)

Require: $E \in G(V), k \in \mathbb{N}^+$, clasificador en caja negra $f : G(V) \to \{0, 1\}$
Ensure: $E' \in G(V)$ tal que $f(E_c) = 1 - f(E)$ o error

```
i ← 0; E_c ← E
E_comp ← COMPLEMENT(E)
edgesE ← BETWEENNESS(E), edgesEComp ← BETWEENNESS(E_comp)
idx1 ← 0; idx2 ← 0
while idx1 < |edgesE| and idx2 < |edgesEComp| and f(E) = f(E') do
  i++; j ← 0
  if idx2 + k > |edgesEComp| then
    k ← 1
  end if
  while j < k do
    j++;
    rand ← RANDOM()
    if idx1 = |edgesE| then
      rand ← 0
    end if
    prob ← |edgesE|/|edgesE| + |edgesEComp|)
    if rand < prob then
      edge ← edgesEComp[idx2]
      E' ∪ {edge}
    else
      edge ← edgesE[idx1]
      E' \ {edge}
    end if
  end while
end while
if f(E') = 1 − f(E) then
  return E'
else
  return error
end if
```

4.2. Implementación

Para llevar a cabo la implementación de este algoritmo, se ha comenzado con la inicialización de las variables i y E', siguiendo el mismo proceso que se describe en la sección 3.1. La diferencia radica en la definición de un grafo complementario y de las listas de aristas ordenados de mayor a menor por intermediación de los

grafos E y E_{comp}. Este grafo complementario se define a partir del grafo original E. La función *complement* se encarga recorrer la matriz de adyacencia de E y elimina las aristas existentes en el grafo complementario, mientras que añade las aristas que no existen en el grafo original.

Respecto a la ordenación por la medida de centralidad de las aristas de los grafos E y E_{comp} se ha definido otra función llamada *betweenness*. Esta función hace uso de la librería *networkx*, donde es capaz de convertir una matriz de adyacencia, que es la estructura con la que se trabaja, a un grafo *networkx* mediante la función *from_ numpy_ array*. Tras convertirlo en grafo, se permite utilizar la función *edge_ betweenness_ centrality* para conseguir la intermediación de las aristas en estructura de diccionario donde la arista (i, j) es la clave y el valor su intermediación.

Sin embargo, este diccionario no está ordenado, por lo tanto, es necesario ordenar las aristas (*sorted*) de mayor a menor en función del valor. Tras ordenarlo, es importante mantener la consistencia que se ha mencionado durante todo el proceso de la implementación de los algoritmos, que no existan aristas repetidas. Es decir, si hay una arista (20, 192), que no exista otra (192, 20). Asimismo, añadir un control en las aristas para que el valor de la fila sea siempre menor que la columna $i < j$.

Una vez obtenidas las listas de aristas ordenadas por intermediación, se inicializan dos variables $idx1$ y $idx2$ para recorrer estas dos listas posteriormente, pues de ellas se seleccionarán las aristas a modificar del grafo E'. El algoritmo mantiene la estructura de un bucle general y otro bucle interno que se encargará de seleccionar las k aristas en cada iteración. Este primer bucle general terminará sólo cuando hayamos encontrado un grafo contrafactual $f(E) \neq f(E')$ o cuando unos de los índices haya llegado a la máxima longitud de su lista correspondiente.

Antes de entrar en detalle en la implementación del segundo *while*, es importante destacar la funcionalidad del if del bucle general, donde si $idx2$ sumado de k es mayor que la longitud de la lista de aristas de $edgesEComp$, $k = 1$. Esto es debido a que las aristas del grafo complementario, va a ser siempre mucho más extenso que el grafo original, por tanto, en la mayoría de los casos $idx1$ termina antes de recorrer la lista. Por ejemplo, para los valores $|edgesEComp| = 1500$, $idx2 = 1497$ y $k = 5$, sigue realizando una iteración porque no se ha excedido de la lista todavía, sin embargo, se estará intentando acceder a elementos fuera del rango en la selección de la cuarta arista y como resultado, excepción.

Una vez implementado este caso de error, se procede a la selección de las k aristas y la operación a realizar dependerá de un número aleatorio entre 0 y 1, en este caso, decimal y no entero. Esto es porque, como se ha mencionado anteriormente, el grafo complementario contiene muchas más aristas que el grafo original. De hecho,

todos los grafos del repositorio contienen 1796 aristas y su complementario 16159. Al existir una gran diferencia entre el número de aristas de ambos grafos, no sería ideal que tengan la misma probabilidad de añadir una arista de *edgesEComp* que la de eliminar una arista de *edgesE* porque se estaría agotando las aristas del grafo original más rápido que las del complementario. Como consecuencia, se establece como limitador entre ambos la probabilidad calculada de la siguiente manera:

$$\frac{|edgesEComp|}{|edgesE| + |edgesEComp|}$$

Así, cuando el número aleatorio sea menor que esta probabilidad, obtenido mediante la función *random.random()*, se selecciona la arista con máximo *betweenness* actual del grafo complementario, se añaden al grafo E' y procede a acumular $idx2$ para seguir con la siguiente arista. En caso contrario, se selecciona la arista con mayor *betweenness* del grafo original E para eliminarlo al grafo contrafactual posible E' y de la misma manera, aumentar $idx1$. Este proceso se realizan k veces, que son las aristas que se modificarán en el grafo E' en la iteración correspondiente.

Finalmente, cuando una de las condiciones del bucle general no se cumple, el algoritmo termina y devuelve el resultado obtenido: el grafo contrafactual si se ha encontrado, o *None* si no ha sido posible encontrarlo.

En resumen, este algoritmo sigue la misma idea de implementación que el OFS1 original, que es modificar el grafo original mediante una copia E' a través de la adición de aristas del grafo complementario y la eliminación de aristas del grafo original, que es la forma más eficiente para conseguir un grafo contrafactual. Sin embargo, en este caso no es necesario controlar la duplicación de aristas mediante un conjunto L, ya que accedemos a la lista mediante índices y, una vez aumentado el valor, la arista actual no será seleccionada nuevamente en siguientes iteraciones.

5. Análisis de resultados

En este apartado se demuestran los resultados obtenidos con los algoritmos implementados fundamentalmente en función de el número de redes que son capaces de procesar y la distancia de edición.

5.1. Conjunto de datos

Con la finalidad de evaluar y comparar el rendimiento de los algoritmos implementados, se utilizará el siguiente conjunto de datos: *Attention Deficit Hyperactivity Disorder* (ADHD), extraído de USC Multimodal Connectivity Database (USCD) [2]. Lo que se trata es evaluar si a partir de un primer grafo contrafactual generado por los algoritmos OFS1 y OFS2 (OFS1 es el algoritmo original y OFS2, nuestra propuesta alternativa), OBS es capaz de reducir a un grafo de mayor calidad con uno u otro.

Esta base de datos contiene 190 individuos en el grupo de condición ADHD y 330 en el grupo de control TD, en total 520 que proviene de diferentes fuentes de recursos, por ejemplo, *Kennedy Krieger Institute* (KKI) o *NYU Langone Medical Center* (NYU).

Cuadro 1: Número de redes disponibles del conjunto de datos

Estudios	ADHD	TD
KKI	20	58
NeuroIMAGE	22	17
NYU	91	96
Peking_1	23	61
Peking_2	34	32
Pittsbugh	0	66
Total	190	330

Con este conjunto de datos que se ha utilizado en el desarrollo de este trabajo, todos los grafos son constantes en términos del número de aristas, por tanto, la probabilidad que se obtiene en el algoritmo OFS2 descrito en la sección 4.2 es la siguiente:

$$\frac{16159}{1796 + 16159} = 0{,}89997215$$

Se trata de un número amplio en comparación con la probabilidad de 50 % del algoritmo original OFS1, y se ha introducido con la intención de que se seleccionen

más aristas del grafo complementario que del grafo original por la existencia de una gran desigualdad.

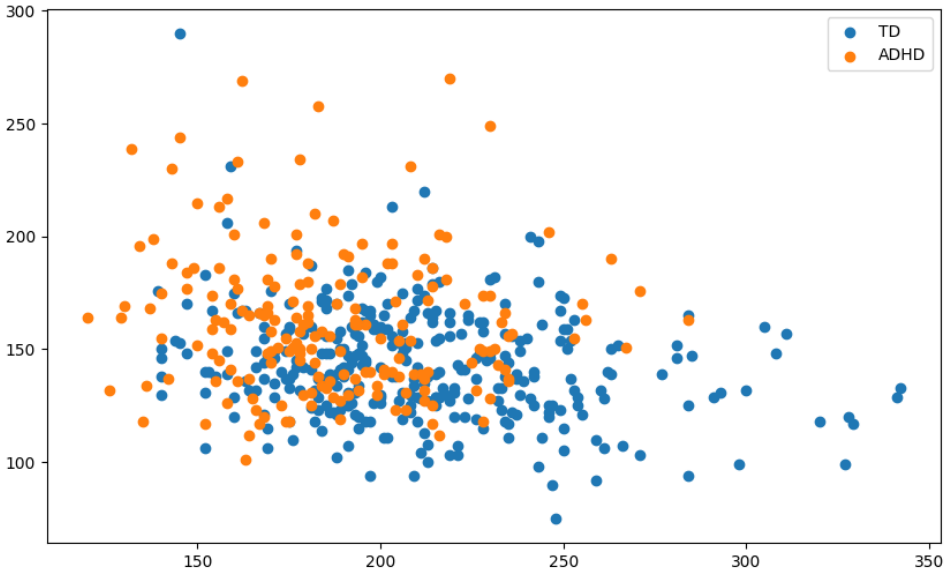


Figura 7: Representación de las redes cerebrales clasificadas en espacio 2D

Sin embargo, es importante tener en cuenta que no se han considerado todos los grafos, ya que existen casos en las que el algoritmo no es capaz de procesarlos, por tanto, no sería relevante incluirlo a nuestro análisis general. Por ejemplo, si el algoritmo OFS1 no logra obtener un grafo contrafactual, pero OFS2 sí, entonces este grafo no sería analizado debido a la falta de datos para comparar ambas ejecuciones.

En consecuencia, se representa la tabla 2 que especifica en detalle cuántas redes tomadas como grafo inicial de los diferentes estudios para el algoritmo *Oblivious Forward Search* (OFS), ha podido conseguir un grafo contrafactual correspondiente como resultado.

No se ha realizado un estudio sobre las iteraciones que se ejecutan para el proceso completo porque se estarían añadiendo casos en las que el número es elevado al tardar en encontrar un grafo contrafactual, pues los algoritmos son parcialmente aleatorios y dependen del grafo original proporcionado y el clasificador. En algunos grafos consigue mejores resultados que otros, e incluso otros que no es capaz de obtener. La gran variedad del número de iteraciones entre grafos hace que el análisis

Cuadro 2: Número de redes procesados por los algoritmos

Estudios	ADHD	TD
KKI	19	36
NeuroIMAGE	22	16
NYU	84	44
Peking_1	23	53
Peking_2	31	30
Pittsbugh	0	61
Total	179	240

no sea totalmente precisa y fiable justo por esos casos concretos que aumenta el promedio.

En consecuencia, tendría mayor relevancia orientarse por la evaluación del número de redes procesadas y no depender del número de iteraciones, sino por el resultado final que devuelven los algoritmos, es decir, la probabilidad de éxito de conseguir un grafo contrafactual dados i intentos.

Aun así, lo primero de todo es ver cómo se ha realizado el preprocesado de los datos y se ha definido el clasificador para que seas posibles estos análisis. Además, se han establecido los siguientes valores por defecto en la llamada de los algoritmos: el número máximo de iteraciones $\eta = 2000$ (en OFS2 no se usaría este parámetro); y selección de $k = 5$ aristas en cada iteración.

5.2. Preprocesado

El preprocesado de estos datos transforma desde las matrices de correlación en matrices de adyacencia como se ha mencionado anteriormente. A partir de los conjuntos de datos que se proporcionan, se leen las matrices de correlación de cada red cerebral. Estos datos son almacenados en un diccionario *data* para realizar cálculos posteriormente, definiendo como clave el nombre del fichero y como valor su matriz. Además de este diccionario, se inicializa otro llamado *map_dict* para que se almacene todos los nombres (valor) de los ficheros en una lista en función de un índice (clave).

Este diccionario se utiliza principalmente para relacionar el grupo (ADHD o TD) con cada grafo mediante la información proporcionada por el fichero *subject_labels_list*. Por lo tanto, sólo es necesario leer cada línea de este fichero y si la línea empieza por una A, entonces se añade (*append*) en el valor de *map_dict* a un 1, por ser el grupo de condición y en caso contrario a 0. Presentaría el siguiente formato: {0: ['KKI_1018959', 0]}

Con la ayuda de *map_ dict*, se puede completar el diccionario *data* con todos los datos necesarios, donde cada elemento contendrá una tupla del grupo al que pertenece y la matriz de correlación correspondiente. Esto es posible porque *map_ dict* tiene el nombre de fichero y es necesario para acceder a cada elemento de *data*.

A continuación, después de haber relacionado el grafo con su respectivo grupo y matriz de correlación, se calcula el percentil de cada grafo, que es lo que convierte la matriz de correlación en matriz de adyacencia. Este grafo es guardado en un diccionario *graphs* con el nombre de fichero como clave y la tupla de etiqueta y matriz de adyacencia como valor. Esta variable es fundamental para la definición del clasificador y la realización de los análisis.

El propósito del proyecto es la implementación de los algoritmos, por tanto, la definición del clasificador es proporcionado por el repositorio y no está incluido en nuestro trabajo. Es decir, la calidad de nuestro código depende en parte del clasificador, pues cuanto mejor sea este, mejor serán los grafos contrafactuales obtenidos.

Finalmente, una vez que se tiene el clasificador definido, ya se puede proceder a la interpretación de los algoritmos y su posterior evaluación. Pues tendríamos todo lo necesario para la ejecución de los algoritmos, un grafo inicial E y el clasificador de caja negra.

5.3. Redes procesadas por los algoritmos OFS1 y OFS2

En primer lugar, se ha realizado un análisis del número de redes que son capaces de procesar los algoritmos OFS1 y OFS2, para observar qué tan acertado es cada uno. A partir de los 520 redes que proporciona el conjunto de datos, se han ejecutado los algoritmos OFSs para observar el número de redes que son capaces de procesar ambos.

La experimentación se ha basado en la paralelización de dos ejecuciones, los grafos ADHD y los grafos TD, en la que se realizan $k = 20$ llamadas a OFS1 y OFS2 para cada red. En el caso de que se haya encontrado un grafo contrafactual para el grafo dado, se acumula en 1 el número de veces exitosas, es decir, que ha podido procesar correctamente, este mismo grafo para el algoritmo correspondiente. Por lo tanto, lo que se almacena en el fichero de texto para la posterior representación gráfica es el nombre del grafo y el número de veces que ha podido procesar este grafo ambos algoritmos.

En la figura 8 se puede observar que el algoritmo original OFS1 tiende a disminuir cuando se incrementa la frecuencia de éxito. Aproximadamente 300 grafos son procesados en la primera llamada de los algoritmos, y no llega a los 100 grafos para

la probabilidad total, es decir, que sea capaz de encontrar un grafo contrafactual siempre. Por otro lado, el algoritmo propuesto OFS2 logra una probabilidad de éxito bastante alta, siendo capaz de procesar aproximadamente 450 grafos para las k llamadas sin casos de fallo.

Las tendencias que siguen los algoritmos son bastantes diferentes. En el caso de OFS1, a medida que aumenta el valor de k, disminuye la cantidad de grafos que puede procesar, lo que indica una mayor probabilidad de fallo en comparación con OFS2. Por otra parte, OFS2 mantiene una tendencia constante para las 450 redes, lo que significa que es un algoritmo que o siempre falla o siempre encuentra un grafo contrafactual.

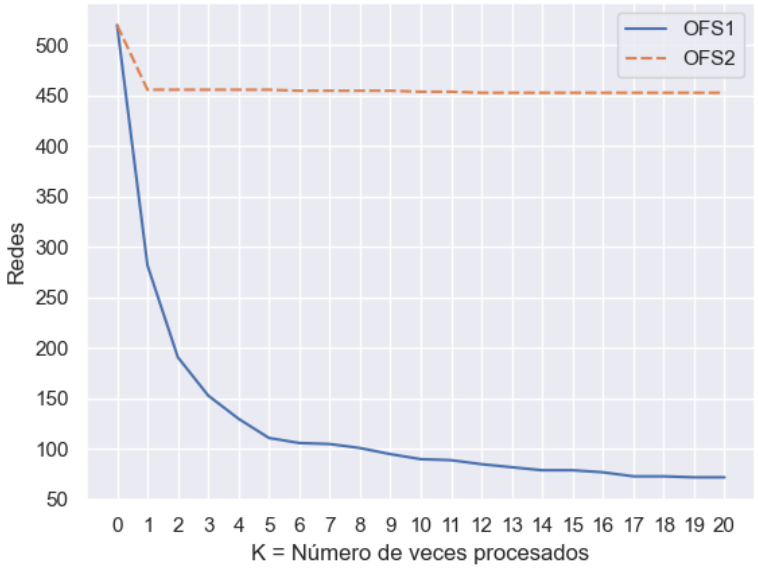


Figura 8: Número de redes que han podido procesar OFS1 y OFS2 en función de K

Al revisar los datos del análisis, se ha observado un patrón interesante que vale la pena destacar. Para aquellas redes en las que OFS2 no es capaz de procesar, OFS1 siempre logra encontrar un grafo contrafactual para la misma en las k llamadas. Esto explica la disminución en la tendencia en OFS2 desde $k = 0$ a $k = 1$, que corresponde precisamente a esas redes falladas. Del mismo modo, es por la misma razón que OFS1 llega hasta $k = 20$ con ese número de redes, ya que se refiere a los mismos grafos en los que OFS2 ha fallado.

En general, a pesar de este comportamiento mencionado, es importante remarcar

que afecta a una minoría de los grafos. En la mayoría de los casos, OFS2 demuestra tener una alta probabilidad de éxito, mientras que OFS1 va variando entre 0 y k. Por lo tanto, OFS2 tiene una mayor probabilidad de acierto y una mayor efectividad en comparación con OFS1.

5.4. Distancia de edición

En este análisis se va a mostrar el resultado obtenido de unos de los principales objetivos propuestos en el trabajo, que es la de intentar encontrar un grafo contrafactual mejor con OFS2 que OFS1, y esto es medido en función de la distancia de edición. La distancia de edición, como se ha explicado anteriormente, es el número de aristas diferentes entre dos grafos.

5.4.1. Distribución de la distancia de edición

El análisis que se ha realizado en este apartado es la comparación entre OFS1+OBS y OFS2+OBS, pues la combinación de los dos algoritmos consigue el grafo contrafactual óptimo E_c dado un grafo E y lo que se quiere comparar es qué combinación de los dos obtiene una menor distancia de edición con el grafo original. Además con las redes cerebrales utilizadas en este experimento, se han utilizado aquellos grafos en las que ambos algoritmos han podido conseguir un grafo contrafactual, en este caso, exactamente 419 redes de los 520 originales.

En este sentido, el desarrollo del análisis se ha basado fundamentalmente en la ejecución de los algoritmos OFS1 y OFS2 junto con OBS para 5 iteraciones. Concretamente, los resultados son las distancias de edición finales de ambas ejecuciones con el grafo original dado, así, calculando el promedio se obtiene la figura 9. La ejecución se ha paralelizado con intención de aumentar la eficiencia, se ha separado diferentes ejecuciones en función de la fuente de estudio y del grupo al que pertenece (ADHD o TD).

Como se ha explicado en la sección 5.3, OFS1 es un algoritmo que tiene una probabilidad alta de fallo y por tanto, se ha establecido un número de intentos (20) para que proceda a buscar el siguiente grafo. En el caso de OFS2, al ser un algoritmo que siempre falla o acierta, se ha establecido un número de intentos mucho menor (5), pues en el caso de que falle en la primera iteración, es poco probable que consiga un grafo contrafactual en las restantes.

Entonces, después de ejecutar los algoritmos y conseguir un grafo contrafactual, se hace uso de la función *edit_ distance* para obtener la distancia entre dos grafos. Por ejemplo, *edit_ distance(E_c, E)* calcularía la distancia de edición entre un grafo contrafactual E_c con el grafo E. Estas distancias son almacenadas en una lista

para su posterior cálculo y escritura en un fichero de texto con la finalidad de poder manipular los datos fácilmente, pues en el caso de que ocurra algún error en la ejecución, se perderían todos los resultados calculados hasta el momento.

A continuación se procede a la representación gráfica de la distancia de edición entre los grafos contrafactuales generados por la combinación de los dos fases y el grafo original de la que se parte. En el eje X se indica la distancia entre ambos y en el eje Y la densidad de los datos, es decir, cuanto mayor sea este valor, mayor son el número de casos que se obtienen dichos resultados.

Se puede observar en la figura 9 que el comportamiento del OFS2 junto con OBS es ligeramente superior a la de OFS1 con OBS, pues consigue unas distancias de edición menores y con más concentración en valores menores que 50, que se refleja en el punto máximo de la curva. Esto es lo deseable, pues cuanto menor sea la distancia de edición, mejor es el resultado obtenido.

Además, a pesar de que nuestro algoritmo consiga un primer grafo contrafactual no eficiente en términos de distancia de edición, al ejecutar OBS sobre ese grafo contrafactual, se obtiene un grafo final con menor distancia de edición con el original, que si se hubiera ejecutado OFS1 con el mismo grafo.

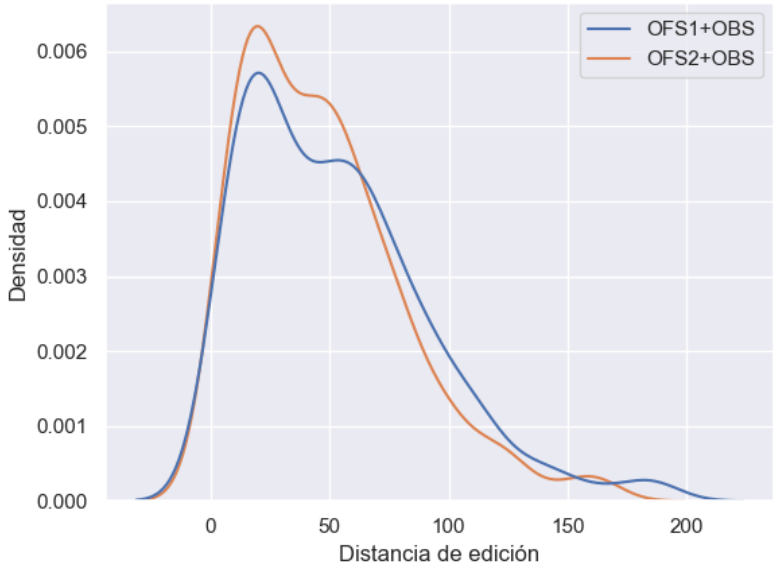


Figura 9: Distancia de edición de OFS1+OBS y OFS2+OBS en densidad

5.4.2. Distribución acumulada de la distancia de edición

La gráfica anterior se basa en la densidad de los datos obtenidos, en este caso, se ha generado esta otra para proporcionar otra perspectiva con la misma intención. Al utilizar los mismos resultados obtenidos que en la figura 9, un punto de interés es precisamente en la línea vertical $X = 50$, donde se puede apreciar que el 60 % de los grafos obtenidos mediante OFS2 tienen una distancia de edición menor que X. Mientras que en el caso contrario, desciende a un 50 % aproximadamente.

Como se ha mencionado anteriormente, esta curva ascendente que se obtiene se asemeja a los datos que se adquieren en la gráfica anterior, existiendo una alta concentración de información alrededor de la distancia con valor 50 y es similar a esta gráfica, pues este mismo valor es el que establece la diferencia de eficiencia entre ambos algoritmos.

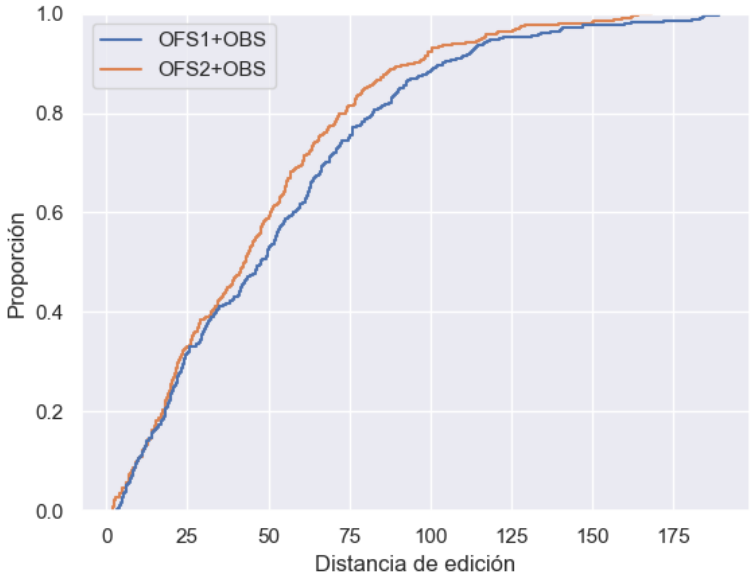


Figura 10: Distancia de edición de OFS1+OBS y OFS2+OBS en proporción

5.4.3. Diferencia de distancia entre OFS1+OBS y OFS2+OBS

No obstante, otro análisis interesante que se ha realizado es la representación gráfica de la diferencia entre los grafos generados. Con ello se justifica que los algoritmos OFSs generan grafos contrafactuales totalmente diferentes, pues uno está basado en la selección aleatoria de aristas y otro en función valor intermediación entre las aristas. De esta manera, se resuelven posibles incertidumbres sobre la similitud en-

tre $G(OFS1+OBS)$ y $G(OFS2+OBS)$, definidos de esta forma para representar los grafos generados con la combinación de los algoritmos.

Se puede observar que los grafos contrafactuales generados son bastantes diferentes, con distancias de edición de hasta 600 y con mayor concentración en el rango $(50, 100)$. Esta distancia puede definirse como alta, pues si se comparan estos resultados con los resultados obtenidos de la distancia entre E_c y E, es aún mucho más elevado.

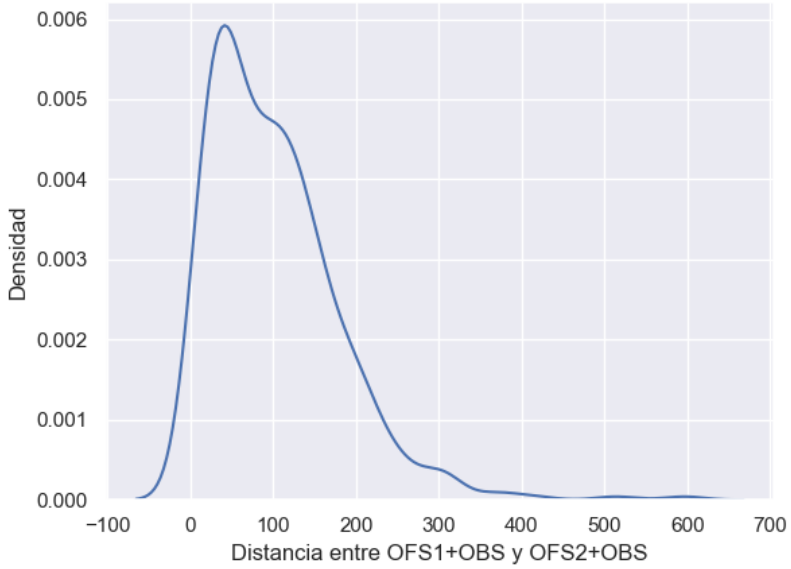


Figura 11: Distancia total entre los grafos contrafactuales generados

5.4.4. Significación estadística

Después de realizar los análisis anteriores, se puede observar que existe una diferencia considerable entre ambos algoritmos. Sin embargo, es importante evaluar si esta diferencia es significativa y si vale la pena considerarla en términos de la relación coste-eficiencia. Dicho esto, es necesario realizar un análisis estadístico para determinar la magnitud de la diferencia y evaluar si los beneficios obtenidos con el algoritmo más efectivo (OFS2) justifican los posibles costos adicionales en términos de recursos y tiempo de ejecución.

Con la prueba t de Student para datos emparejados, para determinar si existe una diferencia significativa entre los algoritmos OFSs se obtiene un valor $p = 2{,}64 * 10^{-40} < 0{,}05$. Lo cual significa que hay suficientes indicios para rechazar la

hipótesis de que los dos métodos producen grafos contrafactuales con la misma distancia de edición. Además, el promedio de distancia de edición entre todos las redes procesadas es de aproximadamente 5.5, que sustenta lo anteriormente expuesto.

Como se había expuesto previamente, la distancia de edición promedio es de 5.5 entre todos los grafos procesados, que se puede observar fácilmente en el punto máximo de la curva. Además, esta distancia se ha calculado con la diferencia entre cada grafo generado por OFS1 y OBS con el generado por OFS2 y OBS. Por lo tanto, que el punto máximo esté situado a la derecha del eje de ordenada tiene gran significado, demostrando que $G_i(OFS1 + OBS) > G_i(OFS2 + OBS)$ en la mayoría de los casos y justifica que los grafos obtenidos con OFS2 son mejores.

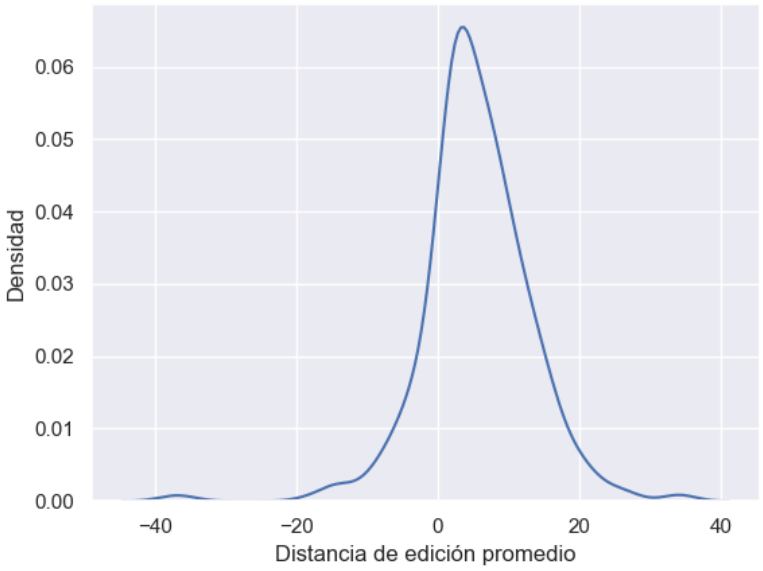


Figura 12: Promedio entre las aristas de los grafos

6. Conclusiones y líneas futuras

6.1. Conclusión

El propósito general de este Trabajo Fin de Grado era la implementación de los algoritmos de Abrate y Bonchi, y fundamentalmente, el desarrollo de un algoritmo alternativo que sea capaz de superar los resultados obtenidos con estos algoritmos. Nuestro enfoque fue la de introducir la centralidad de intermediación en la implementación para que la selección de aristas estuviese basada en un patrón y no al azar.

Después de la implementación, se han realizado estudios para observar la calidad de los grafos contrafactuales obtenidos con ambos (OFS1+OBS y OFS2+OBS) y podemos concluir que hemos alcanzado nuestro objetivo principal, consiguiendo unos resultados mejores. Esto es logrado tanto en distancia de edición como en la probabilidad de acierto a la hora de encontrar un primer grafo contrafactual.

6.2. Líneas futuras

Cabe destacar que en la actualidad, no hay muchas implementaciones para la búsqueda de grafos contrafactuales, ni se han realizado muchos experimentos y análisis sobre bases de datos reales. Por lo tanto, es necesario realizar una investigación exhaustiva en este campo, lo que da lugar a posibles líneas de continuación, y entre ellas resaltamos las siguientes.

6.2.1. Utilización de otros clasificadores de grafos

Este experimento adicional es de gran importancia para observar el comportamiento de nuestros algoritmos, pues la calidad de los grafos contrafactuales que encuentran dependen en gran medida del clasificador que se le proporciona. En este proyecto, el clasificador que se utiliza es de caja negra, donde sólo es capaz de acceder a la predicción resultante, es decir, la explicación del proceso realizado es desconocido.

Además, este clasificador no es determinista, es decir, los grafos son clasificados en una clase u otra distintamente en cada ejecución. Esto puede afectar a los grafos contrafactuales obtenidos con los algoritmos por la gran dependencia a la precisión del clasificador.

Sería interesante utilizar un clasificador de caja blanca, como la regresión lineal, para comparar nuestros grafos contrafactuales obtenidos con el óptimo, siempre y cuando sea posible identificarse.

6.2.2. Comportamiento entre OFS1 y OFS2

Esta línea de investigación es también interesante por el comportamiento de ambos algoritmos que realizan a la hora de buscar un primer grafo contrafactual. Este comportamiento se ha explicado anteriormente en el análisis del número de redes procesadas en la sección 5.3, básicamente eran aquellas redes en la que OFS2 no es capaz de encontrar un grafo contrafactual E_c, OFS1 presenta una probabilidad de acierto máximo y completo para el mismo. Sin embargo, tras visualizar los resultados obtenidos con los $G(OFS2 + OBS)$, OFS2 se trata de un algoritmo "booleano", en el sentido de que para un número determinado de llamadas, si es capaz procesar en el primer intento, entonces siempre acertará para las llamadas restantes. En caso contrario, nunca será capaz de encontrar un grafo contrafactual.

Es necesario resaltar la relación que tiene este comportamiento con la calidad del clasificador para la predicción de las redes neuronales, pues está ligado principalmente en la clasificación que es obtenido. Por lo tanto, realizar investigaciones por esta dirección sería posible determinar precisamente esta influencia y además de perfeccionar nuestro algoritmo para que sea capaz de procesar un número de redes más amplio.

Referencias

[1] Carlo Abrate y Francesco Bonchi. «Counterfactual Graphs for Explainable Classification of Brain Networks». En: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining*. KDD '21. Virtual Event, Singapore: Association for Computing Machinery, 2021, págs. 2495-2504. ISBN: 9781450383325. DOI: 10.1145/3447548.3467154. URL: https://doi.org/10.1145/3447548.3467154.

[2] Jesse Brown et al. «The UCLA multimodal connectivity database: a web-based platform for brain connectivity matrix sharing and analysis». En: *Frontiers in Neuroinformatics* 6 (2012). ISSN: 1662-5196. DOI: 10.3389/fninf.2012.00028. URL: https://www.frontiersin.org/articles/10.3389/fninf.2012.00028.

[3] Wikimedia Commons. *Centralidad de intermediación*. https://commons.wikimedia.org/wiki/File:Graph_betweenness.svg. Accedido en 16-06-2023.

[4] Wikimedia Commons. *Matriz de adyacencia*. https://commons.wikimedia.org/wiki/File:Matriz_de_adyacencia.jpg. Accedido en 16-06-2023.

[5] Mark Newman. *Networks*. Oxford University Press, Inc., 2018 2ª ed. ISBN: 0198805098.

[6] Alfredo Sánchez. *Teoría de conjuntos*. https://aprendeconalf.es/analisis-manual/01-teoria-conjuntos.html. Accedido en 16-06-2023.

[7] D. A. Holton, J. Sheehan. *The Petersen Graph*. Cambridge University Press. ISBN: 0-521-43594-3. DOI: 10.2277/0521435943.

[8] J. Torres. *Conectomas neuronales*. https://www.ugr.es/~jtorres/Tema_5_conectomes.pdf. Accedido en 16-06-2023. 2019.