

Proyecto de Modelos de Matemática Aplicada.

Construcción y graficación de grafos causales

Dennis Fiallo Muñoz

Grupo C-311

DENNIS.FIALLO@ESTUDIANTES.MATCOM.UH.CU

Lauren O. Guerra Herández

Grupo C-312

LAUREN.GUERRA@ESTUDIANTES.MATCOM.UH.CU

Tutor(es):

Dr. Ferando Rodríguez Flores *MatCom*

Lic. Ania Mesa Rodríguez , *MatCom*

Resumen

El presente proyecto se encarga de resolver un problema práctico para matemáticos que realizan estudios sobre el cerebro humano. Es una herramienta creada específicamente para facilitar la visualización de modelaciones del cerebro y sus conexiones a partir de grafos causales. Para esto a partir de modelaciones en matrices previamente realizadas se construyen grafos con ayuda de la librería gravis y se visualizan cómodamente en una interfaz visual web.

Para más información sobre la interfaz visual del proyecto ver el manual de usuario en: [link](#)

Para acceder al código fuente del proyecto ir a: [link](#)

Abstract

This project is responsible for solving a practical problem for mathematicians who carry out studies on the human brain. It is a tool created specifically to facilitate the visualization of models of the brain and its connections from causal graphs. For this, based on matrix modeling previously carried out, graphs are built with the help of the gravis library and are conveniently displayed in a visual web interface.

For more information about the visual interface of the project, see the user manual at: [link](#)

To access the source code of the project go to: [link](#)

Palabras Clave: grafos causales, librerías, módulos, conexiones causales, conexiones contemporáneas, python

Introducción

Se necesita para investigaciones en el Centro de Neurociencias de Cuba una aplicación que ayude a los investigadores a representar en forma de grafo las de conexiones causales y contemporánea que ocurren en el cerebro humano para esto nos proponemos hacer un programa al cual se le pasen dos tensores de conectividad y una lista con los nombres de los nodos y con estos datos construir un grafo que admita las 360 conexiones necesitadas y además se vea de la mejor manera posible para apoyar su estudio y análisis.

El proceso de construcción de grafo causales se realiza a partir de dos tensores de 360x360xlags, siendo uno binario indicando si hubo conexión o no, y otro de pesos indicando la fuerza de las conexiones. Estos se construyen de la siguiente manera:

Conexiones contemporáneas: Se representan con una

línea recta. Se toma el lag 0(único lag simétrico) y donde aparezca 1 significa que hay conexión entre los nodos (i, j), mientras que el color de la conexión que se traza viene dado por el valor de la matriz de peso en la posición (i, j).

Lagged connections o causalidad: Se representan con flechas. Para el resto de lags se busca igualmente si hay un 1 en el tensor binario de lag $\neq 0$ y se construye una flecha del nodo i al j con la intensidad dada por el promedio de los valores de la matriz de pesos en los lags donde hubo conexión, además de especificar los lags con conexión.

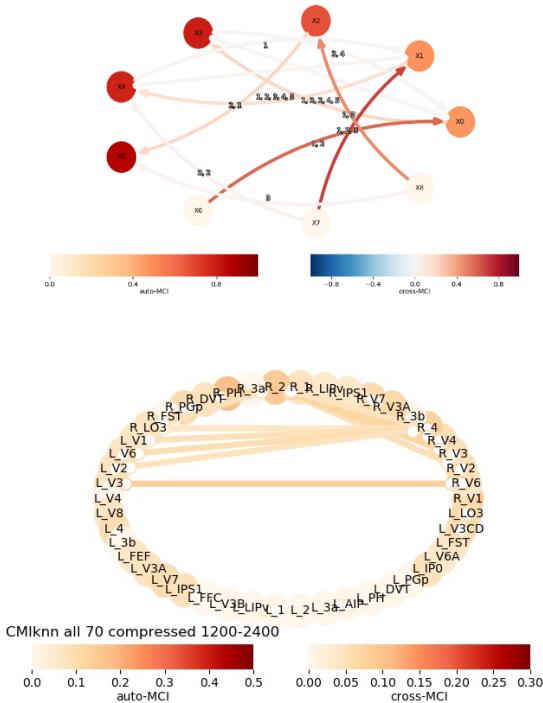


Figura 1: Ejemplo de graficación de grafos causales que se pueden ver con precisión

Las aplicaciones con las que cuenta el usuario al pasarle un gráfico de más de 180 series de tiempo(en la práctica se necesitan 360), el programa pone 180 nodos sobre una circunferencia y los restantes los sobrepone a estos en la misma circunferencia, haciendo que no sea posible ver las conexiones.

También otro problema que presenta el usuario es que guarda cada uno de los pares de tensores que se utilizan para la contrucción de los grafos en formato matlab, donde cada archivo .mat que contiene un par de tensores ocupa un espacio de más de 3.0 mb, por lo cual al tener cientos de archivos de este tipo para su investigación ocupan un espacio considerable, por esto además nos propusimos que una vez conformado el grafo, sea guardado en un archivo que pese mucho menos y luego este pueda ser cargado desde nuestra misma aplicación.

Primeras ideas

Al comienzo de la realización de este proyecto se tenía en mente la implementación de métodos para visualizar grafos que lograran una buena distribución de nodos y aristas en el espacio bidimensional utilizando los conocimientos adquiridos en asignaturas como Matemática Discreta II, Estructura de Datos y Algoritmos II y Modelos de Optimización, esto dibujando el grafo en su forma maximal planar y luego ir añadiendo las aristas que falten por dibujarse minimizando la cantidad de aristas que serían dibujadas cortando a las que ya estuvieran colocadas.

Esto tuvo grandes inconvenientes porque los grafo causales, sobre todo el de conexiones contemporáneas, están muy lejos de ser planares, tienen muchas aristas y a la hora de acomodar los nodos y dibujar las aristas que no formaban parte del grafo maximal planar tomado como base no se podían garantizar distribuciones de nodos y aristas que dieran lugar a la correcta observación del grafo.

Buscando en el estado del arte no existía nada como esto, ninguna distribución de nodos fija que se hiciera a un grafo resultaba buena en todos los casos, para todo tipo de grafos, probamos con la librería networkx las diferentes distribuciones que esta tiene implementadas y ninguna de ellas cumplía nuestro objetivo, por lo que llegamos a la conclusión que crear un algoritmo que graficara cualquier grafo de la mejor manera posible para él es un ejercicio muy complicado por lo cual no se encontraba a nuestro alcance por razones de tiempo y experiencia.

Por esto nos propusimos utilizar las mejores implementaciones de visualización de grafos que estuvieran disponibles en librerías de python para poder resolver el problema del usuario y lograr su satisfacción y comodidad.

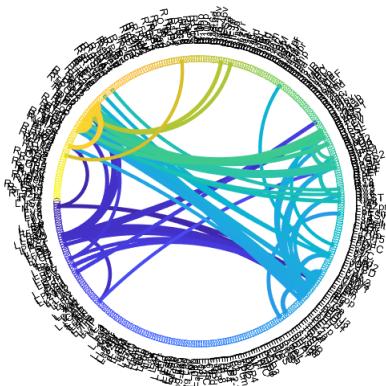


Figura 2: Ejemplo de graficación en que los nodos se superponen y no se pueden observar los resultados

Por esto nuestro objetivo es hacer un programa al cual se le pasen dos tensores de conectividad y una lista con los nombres de los nodos y construir un grafo como se explicó anteriormente que admita las 360 necesarias y en general la cantidad de conexiones que se necesiten representar en cada caso y se puedan ver correctamente los nodo, las conexiones y la fuerza de estas.

Solución

El proyecto se encuentra programado en su totalidad en Python. Para la graficación de los grafos causales se usa el módulo gravis [1]; su nombre significa **graph visualization**, visualización de grafos, y como su

nombre indica, está destinado a la creación de grafos interactivos en 2d y 3d. Usa tecnologías como HTML, CSS y JS para representar los grafos, apoyándose en gran medida de módulos como `d3.js`, `vis.js` y `three.js` (las representaciones visuales basadas en estos módulos se presentan en el proyecto como **Graph**, **Complex Graph** y **3D Graph** respectivamente). Para mostrar los resultados se usa en la interfaz visual se usa el módulo `streamlit` [2], tomando la información proporcionada por `gravis` como HTML.

Para la representación de los grafos en el proyecto se usa el mismo formato que se usa en `gravis`, `gravis` JSON Graph Format(gJGF), el cual se define como `dict` en Python generándose como se muestra a continuación:

```

1 {
2     "graph": {
3         "metadata": {},
4         "directed": bool,
5         "nodes": {},
6         "edges": []
7     }
8 }
```

Para la representación de los grafos se puede indicar desde el inicio las posiciones espaciales de cada nodo, como es el caso del **3D Graph** que presentamos con ubicaciones reales a escala de las áreas estimuladas del cerebro. Sin embargo en el resto de los casos se usan algoritmos apoyados en las físicas para representar los nodos.

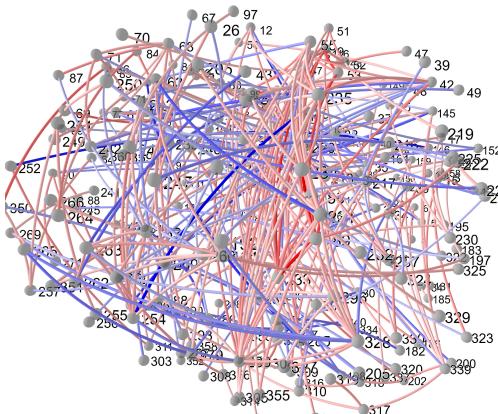


Figura 3: Ejemplo de representación de grafo en tres dimensiones con los nodos distribuidos según la forma del cerebro

Usando la opción de **Many-Body Force** existen distintos tipos de fuerzas e interacciones entre los nodos y las aristas, las cuales se pueden modificar, entre ellas: **Links Force** la cual es la fuerza que ejercen las aristas sobre los nodos enlazados, **Collision Force** que es la fuerza de repulsión entre nodos, **Centering Force** esta atrae a los nodos hacia

el centro de la gráfica, **x Force** y **y Force** que son las fuerzas que atraen a los nodos hacia el eje señalado.

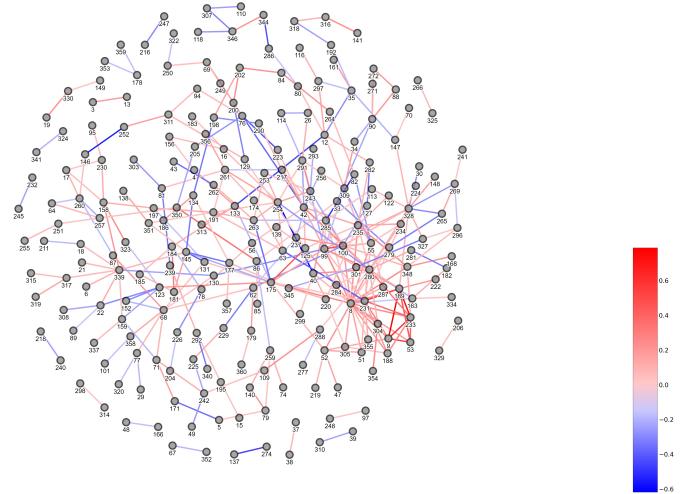


Figura 4: Representación de un grafo que representa las conexiones contemporáneas y las fuerzas de los enlaces con los colores de las aristas

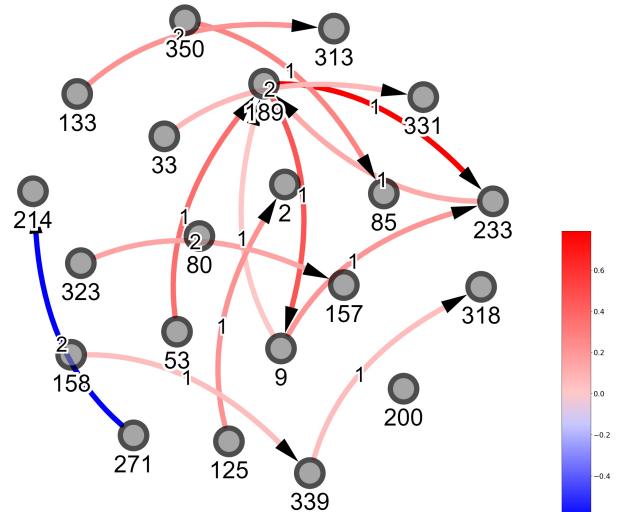


Figura 5: Representación de un grafo dirigido que representa las conexiones causales

En el caso de las representaciones de **Complex Graph** además de estas físicas anteriores, también se permiten seleccionar distintos algoritmos para la representación, como es el caso del algoritmo de Barnes-Hut, este divide recursivamente los n cuerpos en grupos almacenándolos en octábolos, mientras que el espacio se subdivide recursivamente en octantes hasta que cada subdivisión tenga 0 o 1 cuerpos.

Aplicando estas representaciones con físicas se logra una graficación más factorizada, logrando facilitar al usuario las visualización de los resultados, además de lograr optimizar en tiempo de ejecución y procesa-

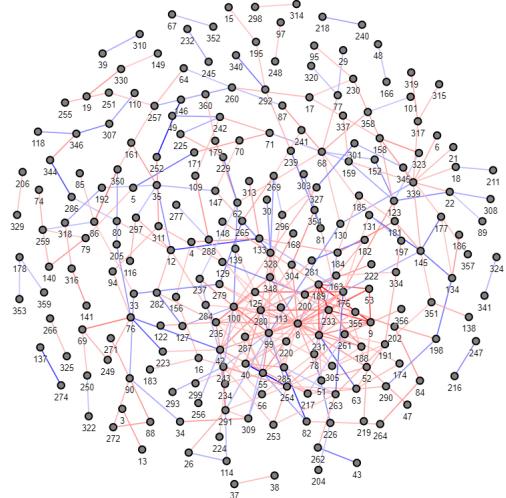


Figura 6: Ejemplo de representación de tipo Complex Graph

miento, pues los algoritmos que se usan tienen una complejidad temporal máxima de $O(n \log n)$.

1. Conclusiones

En las investigaciones para la realización de este proyecto pudimos adentrarnos en todo lo que es la representación de grafos en python y las mejores formas de visualización que se han implementado en este lenguaje. Las formas interactivas de graficación son bastante mejores y ofrecen una mejor visualización que las diferentes formas de distribución fija de nodos que se hacen hoy en día.

Por esto con la utilización de módulos de python que representan grafos de forma interactiva en el espacio bidimensional se logra bastante bien el objetivo de este proyecto de visualizar con comodidad grafos causales y esperamos sea una herramienta de mucha utilidad para las investigaciones que se apoyen en él.

2. Recomendaciones

Para mejorar este proyecto y que resulte aún más útil se recomienda representar en un solo grafo las conexiones contemporáneas y las conexiones causales, lo cual no pudimos llevar a cabo porque esto necesita tener un grafo con aristas dirigidas y no dirigidas al mismo tiempo, lo que nos imposibilitaría explotar las bondades de librerías como gravis [1] que hizo todo el proceso de distribuir el grafo en el espacio bidimensional. Por lo que esta mejora llevaría un gran trabajo.

Igualmente en los gráficos en tres dimensiones, los cuales se organizan para tener forma cerebral, una mejora visual sería que curvar las aristas de forma que simule mejor la forma del cerebro y añadir una imagen del cerebro debajo para que se vea realmente que zona

de este representa cada nodo.

Referencias

- [1] Documentación de la librería de python gravis <https://robert-haas.github.io/gravis-docs>.
- [2] Documentación de la librería de python streamlit <https://docs.streamlit.io>.