Proyecto de Modelos de Matemática Aplicada. Construcción y graficación de grafos causales

Dennis Fiallo Muñoz DENNIS.FIALLO@ESTUDIANTES.MATCOM.UH.CU
Grupo C-311

Lauren O. Guerra Herández LAUREN.GUERRA@ESTUDIANTES.MATCOM.UH.CU
Grupo C-312

Tutor(es):
Dr. Ferando Rodríguez Flores *MatCom*
Lic. Ania , *MatCom*

Resumen

Se necesita para investigaciones en el Centro de Neurociencias de Cuba una aplicación que ayude a los investigadore a representar en forma de grafo las de conexiones causales en el cerebro humano 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 se puedan ver correctamente los nodo, las conexiones y la fuerza de estas.

Abstract

The English abstract must have have 100 to 200 words, and present the essentials of the article content in a clear and concise form.

Palabras Clave: grafos causales, librerías, python

1. Introducción

Se necesita para investigaciones en el Centro de Neurociencias de Cuba una aplicación que ayude a los investigadore 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 contrucción 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≠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.

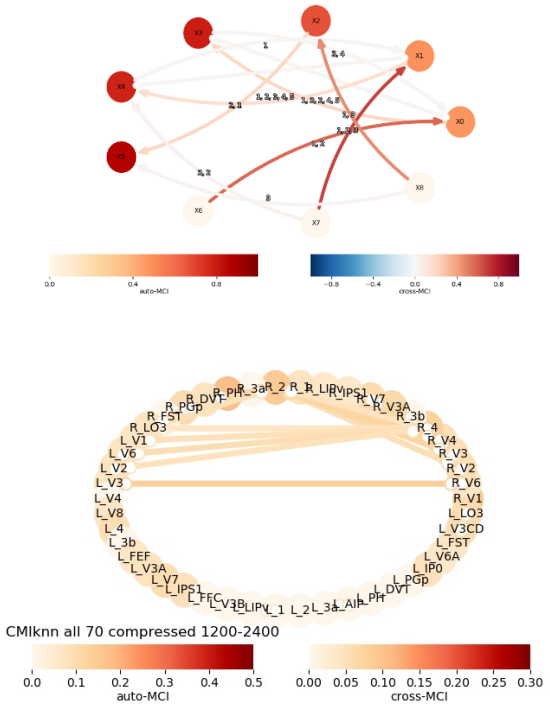

Figura 1: Ejemplo de graficación de grafos causales que se pueden ver con precisión

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.

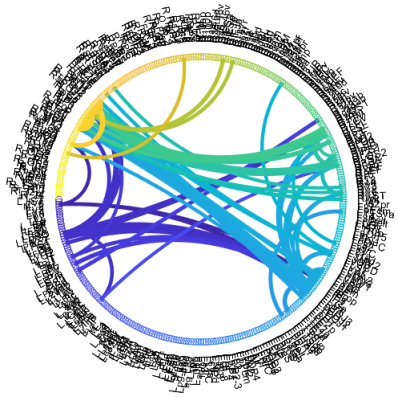

Figura 2: Ejemplo de graficación en que los nodos se superponen y no se pueden observar los resultados

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.

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 dibujar grafos que lograran una buena ditribución 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 distrubuciones 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 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 poible para él es un ejercicio muy complicado por lo cual no se encontraba a nuestro alcance.

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

CODE

El proyecto se encuentra programado en su totalidad en Python. Para la graficación de los grafos causales se usa el módulo gravis; 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 interfáz visual se usa el módulo streamlit, 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:

```
{
        "graph":{
                "metadata": {},
                "directed": bool,
                "nodes": {},
                "edges": []
        }
}
```

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 con ubicaciones reales a escala de las areas estimuladas del cerebro. Sin embargo en el resto de los casos se usan algoritmos apoyados en las físicas para representar los nodos. Usando la opción de `Many-Body Force` existen distintos tipos de fuerzas e interacciones entre los nodos y las arista, 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. En el caso de las representaciones de `Complex Graph` además de estas físicas anteriores, tambien 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 logar optimizar en tiempo de ejecución y procesamiento, pues los algoritmos que se usan tienen una complejidad temporal máxima de $O(n \log n)$.

Solución

1.1 Listas y Descripciones

Para producir listas enumeradas, utilice el siguiente estilo:

1. Primer Elemento
2. Segundo Elemento
   a) Segundo Elemento - Subítem Uno
   b) Segundo Elemento - Subítem Dos

Para producir descripciones, use el siguiente estilo:

Primer Elemento con su respectiva descripción.

Segundo Elemento también con su respectiva descripción.

1.2 Figuras

Para producir cuerpos flotantes (figuras o tablas), asegúrese de numerar y etiquetar correctamente cada figura. Las referencias a las figuras deben estar correctamente etiquetadas. Por ejemplo, véase la Fig. 3...

	Método 1	Método 2
A		
B		
C		

Figura 3: Figura de ejemplo. Recuerde especificar el origen de los datos que se muestran.

1.3 Código Fuente

Para producir código fuente, envuélvalo en una figura flotante y etiquételo correctamente. Por ejemplo, en la Fig. 4 se muestra un código bastante conocido...

```
int main(int argc, char** argv)
{
    // Imprimiendo "Hola Mundo".
    printf("Hello, World");
}
```

Figura 4: Código fuente de ejemplo.

2. Conclusiones

En esta sección puede incluir las conclusiones de su investigación y las ideas sobre la continuidad del trabajo, en el caso que aplique.

3. 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 curvear 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.