Modelos de Interacciones Sociales: Taller 4

Facultad de Economía, Universidad de los Andes.

Noviembre 30 de 2022

Instrucciones del trabajo:

El presente taller busca que se apliquen los conceptos vistos durante las clases Magistral y Complementaria de las semanas 12-16 del curso. Adicional a esto, con los ejercicios propuestos se busca desarrollar una mejor comprensión del uso de Python, de los paquetes vistos en clase y fomentar un razonamiento lógico que lleve a los estudiantes a comprender e implementar de una buena manera los tópicos tratados.

El taller se debe desarrollar en grupos de **4 personas (sin excepciones)**. Las respuestas al mismo deben ser entregadas en formato pdf, con el patrón de marcado: Cod1_Cod2_Cod3_Cod4.pdf. Por otro lado, se debe adjuntar el archivo .py que contenga las implementaciones correspondientes. Este último archivo debe venir marcado con el formato: Cod1_Cod2_Cod3_Cod4.py. En esta ocasión solo se recibirán archivos .py que tengan la estructura de librería, i.e. consten únicamente de paquetes cargados y las funciones que ustedes construyan, las cuales deben tener la estructura solicitada (inputs y outputs en la forma y orden que se piden). También se pueden incluir funciones auxiliares que ayuden a la implementación de las funciones principales. La fecha de entrega es el **09 de Diciembre de 2022 a las 23:59** a través del enlace publicado en Bloque Neó y **NO se extenderá la fecha de entrega**. Los archivos .pdf y .py deben estar en una carpeta comprimida (.zip) marcada así: T4_MIS_Cod1_Cod2_Cod3_Cod4.zip.

Con respecto a los factores que pueden afectar la nota se encuentran: (1) los códigos que no estén ordenados, debidamente comentados, marcados según el formato especificado, sin los nombres solicitados para los módulos y funciones o que en las funciones solicitadas no tengan los argumentos solicitados, serán penalizados con 0.5 unidades por cada fallo; (2) aquellos scripts o fragmentos de script que no corran o no ejecuten la rutina solicitada anulan los puntos obtenidos por respuesta correcta; (3) los archivos .pdf que no sean legibles, no estén ordenados o no estén marcados con el formato solicitado tendrán la misma penalización que en el punto (1); (4) se dará un bono de 0.5 unidades a aquellos archivos .pdf creados con LaTeX. Como último elemento de estas instrucciones, la copia o plagio en este trabajo está totalmente prohibida e

incurrir en esta práctica conlleva a una nota de cero (0) en el taller, así como a las sanciones correspondientes tenidas en cuenta en el reglamento de estudiantes de la universidad.

Finalmente, los scripts deben estar organizados por puntos y subpuntos. Para ser consistentes con la organización, cada implementación, i.e. cada subpunto, debe contener una sección de funciones relevantes y posterior a esta la sección de implementación en la cual se ejecutan las funciones creadas para dar respuesta a lo solicitado. En este taller, los puntos que no sean demostraciones serán testeados y si las soluciones no son las esperadas se pondrá una nota de cero.

Puntos a Desarrollar:

0.1 Caminos más Cortos *(3.33 puntos.)*.

De acuerdo al planteamiento expuesto en las presentaciones de la solución al problema de encontrar caminos más cortos en una red, construya un algoritmo que use la ecuación de Bellman para encontrar este camino. La función debe retornar el camino más corto (un camino más corto en caso de tener múltiples) y el costo del camino. La estructura de la función es bellman_caminos_cortos(grafo:nx_DiGraph,source:node,target:node)→list, donde grafo es un grafo dirigido que contiene los nodos "source" y "target", los cuales representan el nodo desde donde se quiere ir (source) y el nodo al cuál se quiere llegar (target). Además, note que los enlaces deben tener un costo para ser transitados, así que el grafo tiene un atributo de enlace, llamado "costos", con dichos valores de costo (todos mayores o iguales a cero). Se retorna una lista con dos componentes: [0] el camino más corto entre source y target (si no existe, None) y [1] con el costo de dicho camino (si no existe, np.infty). El camino que se presenta debe ser una lista Python.

0.2 Juegos de Transporte *(3.34 puntos.)*.

Suponiendo que se está jugando un juego de transporte, donde un conjunto de agentes van de un lugar A a un lugar B, los cuales están conectados por al menos dos caminos distintos, se quiere verificar al menos un EN. Así, cree una función que reciba una red dirigida sin loops, un conjunto de N jugadores (N par), un listado con dos nodos (nodo inicial y terminal), y un diccionario que contenga llaves (los enlaces de la red) y valores (funciones de costo sobre los enlaces). Note que los nodos de la red son puntos en la red de transporte, luego no los confunda con los agentes. Esta función debe agregar el diccionario como atributo de enlaces a la red (llámelo 'costos') y luego de ello correr un algoritmo para encontrar un EN. La función debe encontrar un EN, el cuál será representado por un diccionario con la siguiente estructura: las llaves son los agentes que se introdujeron y los valores son los caminos (un único camino por jugador) que toman los agentes (estos deben presentarse como un listado de

Python). Para hacerse una idea, en el ejemplo de las diapositivas y con un conjunto de agentes dado por el listado $[0, 1, 2, 3]$, se debe retornar lo siguiente: en el caso de la red con dos caminos posibles la respuesta debe ser del estilo $\{0 : [A; C; B], 1 : [A; C; B], 2 : [A; D; B], 3 : [A; D; B]\}$, mientras que en el caso de la red con tres caminos donde un enlace tiene costo cero la respuesta debe ser $\{0 : [A; C; D; B], 1 : [A; C; D; B], 2 : [A; C; D; B], 3 : [A; C; D; B]\}$. Así mismo, en cada iteración del algoritmo que corran se debe calcular la energía potencial de la red y guardar esto en una lista Python (i.e. un listado de números que representan la energía potencial de TODA la red). Con esto, la función debe retornar un listado Python con dos componentes: [0] el diccionario que representa el EN y [1] el listado con los valores de energía potencial que se obtienen en cada iteración. *Hint: Inicialice el algoritmo de búsqueda de EN asignando 1/k agentes a cada uno de los posibles caminos entre el nodo source y target, siendo k el número de caminos entre estos nodos. Tenga cuidado con las asignaciones NO enteras.*

0.3 Flujos de Transporte *(3.33 puntos.)*.

A continuación implemente una función que reciba un grafo con pesos (atributo de enlaces llamado "costos") de ir de un conjunto de nodos "O" (oferentes) a un conjunto de nodos "D" (demandantes), donde los conjuntos son disjuntos. Además, el grafo contiene la clasificación del nodo (atributo de nodo llamado "tipo", el cuál es "O" u "D") y un valor de Oferta/Demanda (atributo de nodo llamado "valor"). Note que se entiende que el nodo tiene una oferta/demanda determinada por el atributo "tipo" y valuada según el atributo "valor". La función debe retornar un grafo que tenga como únicas conexiones las que se establecen entre oferentes que venden una cantidad positiva a un demandante (en un atributo de enlace llamado "transferencia" y debe ser un atributo extra a los atributos originales del grafo). La estructura de la función es transporte_optimo(grafo:nx.DiGraph)→nx.DiGraph. Finalmente, para que esta implementación funcione asegúrese que la oferta agregada es mayor o igual a la demanda agregada. Si este no es el caso, su función debe levantar una excepción que diga "Imposible realizar la optimización".