Experimentos con el algoritmo de Ford-Fulkerson en Python

Gabriela Sánchez Y.

Introducción

Analizamos el algoritmo de Ford-Fulkerson visto el la práctica tres [1] que determina el flujo máximo entre un nodo fuente s y un nodo destino t.

El objetivo de esta práctica es crear un tipo especial de grafo: una rejilla; percolar aristas y vértices y observar el efecto que ésto tiene en el flujo máximo. Además se analiza experimentalmente la complejidad asintótica del algoritmo de Ford-Fulkerson, midiendo el tiempo de ejecución del mismo.

Rejilla

Trabajamos con una rejilla uniforme de $k \times k$ nodos en la cual los nodos se encuentran uniformemente espaciados. La rejilla tiene dos tipos de aristas: simétricas y de largo alcance.

Las aristas simétricas se crean con base en la distancia Manhattan que hay entre los pares de nodos. Por ejemplo, si $l = 1$ se unirán todos aquellos pares de nodos que se encuentren a una distancia Manhattan igual a uno, si $l = 2$ se unirán todos los pares de nodos que se encuentren a distancia Manhattan $d \leq 2$. Este tipo de aristas serán ponderadas y no dirigidas. La ponderación de las aristas simétricas se elije de acuerdo a una distribución normal con media μ y desviación estándar σ.

Las aristas de largo alcance se crean en relación con una probabilidad p de manera aleatoria entre pares de nodos. Si ya hay una arista simétrica entre el par de nodos elegido, no se crea ningún cambio. Las aristas de largo alcance son dirigidas y su ponderación se obtiene de una distribución exponencial con parámetro λ.

La función `rejilla(k, l, p, mu, sigma, lamb)` de la clase `grafo.py` es la encargada de crear la rejilla. Recibe los parámetros k que determina el tamaño por lado de la rejilla, l para la creación de las aristas simétricas, la probabilidad p de crear aristas de largo alcance, la media μ y la desvación estándar σ de la distribución normal que determina las capacidades de las aristas simétricas y el parámetro λ de la distribución exponencial que define las capacidades de las aristas de largo alcance.

```
def rejilla(self, k, l, p, mu, sigma, lamb):
    et = 0 # etiqueta nodo
    for i in range(k):
        for j in range(k):
            x = j
            y = i
            r = 0.1 # radio del nodo
            self.nodo(et, x, y, r)
            et += 1
    for v in self.V: # aristas simetricas
        for w in self.V:
            d = self.man(v, w)
            if d <= l and (v, w) not in self.E and v != w:
                pond = random.normalvariate(mu, sigma)
                self.arista(v, w, False, pond)
    for v in self.V: # aristas de largo alcance
        for w in self.V:
            if random() < prob and (v, w) not in self.E:
                pond = random.expovariate(lamb)
                self.arista(v, w, True, pond)
```

Estaremos calculando el flujo máximo en este tipo de grafos por lo que es necesario definir el nodo fuente y el nodo destino. El nodo fuente s es el nodo superior izquierdo de la rejilla, mientras que el nodo destino t es el nodo inferior derecho, tal y como se muestra en la figura 1.

Percolación de aristas

El procedimiento que se sigue para la percolación de aristas es el siguiente: del conjunto de aristas, se elije de forma totalmente al azar una de ellas y se elimina.

Si la arista (u, v) a eliminar es simétrica, se debe eliminar también la arista que está en sentido contrario, esto es, la arista $(v.u)$. Si se trata de una arista de largo alcance no es necesario realizar este paso, ya que éstas son dirigidas.

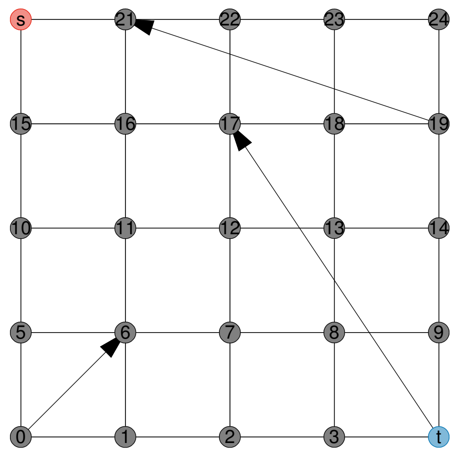

Figura 1: Rejilla con $k = 5$, $l = 1$ y $p = 0.005$. Nodo fuente s de color rojo y nodo destino t de color azul.

Cada que una arista es eliminada se calcula el flujo máximo en el nuevo grafo y se observan los efectos que tiene la percolación en el mismo. Se repite el procedimiento mientras haya un flujo existente del nodo s al nodo t.

```
def per_aris(self,cual): # percolacion aristas
    (u,v) = cual[0]
    (diri, pond) = self.E[(u,v)]
    if diri:
        self.E.pop((u,v))
    else: # si es arista simple
        self.E.pop((u,v))
        otro = (v, u)
        self.E.pop(otro)
```

Se realizaron cinco réplicas del experimento. Los resultados obtenidos se observan en la figura 2. Los resultados se obtienen de la percolación en una rejilla con $k = 10$, $l = 3$, $p = 0.005$, $\mu = 5$, $\sigma = 2$ y $\lambda = 5$.

El flujo disminuye lentamente al eliminar aristas, en ocasiones se mantiene constante, esto puede deberse a que en algún punto hay aristas que no son

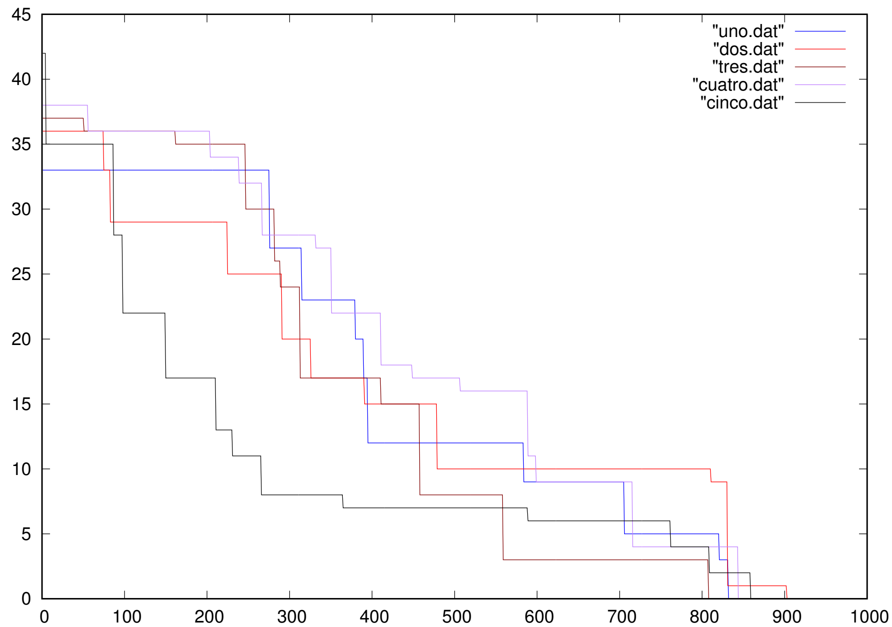

Figura 2: Efectos en el flujo debidos a la percolación de aristas.

relevantes en el camino que sigue el flujo.

Percolación de vértices

Del conjunto de vértices del grafo se elije uno al azar y se elimina; es claro que deben eliminarse todas las aristas que estén conectadas a él, ya sean simétricas o de largo alcance, utilizando el procedimiento descrito en la sección anterior.

En la figura 3 se visualiza una rejilla al realizarse la percolación de nodos y aristas, con $k = 3$, $l = 3$ y $p = 0.002$

Al igual que en el caso de la percolación de aristas, se calcula el flujo máximo usando el algoritmo de Ford-Fulkerson. Ya que el objetivo es observar el efecto de la percolación de vértices en la cantidad de flujo del nodo s al nodo t, es obvio que estos nodos no podrán ser eliminados.

```
def per_nodo(self,cual): # percolacion nodos
        cual = cual[0]
        vec = self.vecinos[cual]
        self.V.pop(cual)
```

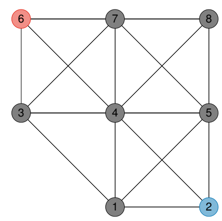

Figura 3: Ejemplo de rejilla con percolación de vértices y aristas.

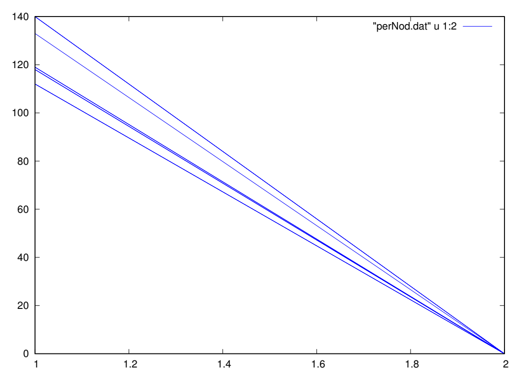

Figura 4: Efectos en el flujo debidos a la percolación de vérties.

```
for a in vec:
    quien = [(cual,a)]
    if (cual,a) in self.E:
        self.per_aris(quien)
```

Los resultados obtenidos se muestran en la figura 4 con $k = 10$, $l = 4$, $p = 0.002$, $\mu = 10$, $\sigma = 2$ y $\lambda = 5$.

Referencias

[1] Gabriela Sánchez Yepez. Práctica 3: Medición experimental de la complejidad asintótica con python y gnuplot. https://github.com/Saphira3000/Flujo_Redes/tree/master/tarea3.