Grafos en red, distancia Manhattan y percolación de vecindades

José Alberto Benavides Vázquez

7 de mayo de 2018

1. Introducción

Esta práctica se realizó a partir de un programa en desarrollo para flujo en redes alojado en `https://github.com/jbenavidesv87/FlujoRedes`. El código de esta práctica puede consultarse en `https://github.com/jbenavidesv87/FlujoRedes/tree/master/ejemplos/08Rejilla` [1].

En esta simulación se generan grafos con $k \times k, k = \{3, 4, 5, 6, 7, 8, 9, 10\}$ nodos dispuestos en forma de red, conectados con vecinos a una distancia Manhattan $l = \{1, 2, 3\}$ y con con una probabilidad $p = \{0, 0.02, 0.04, 0.06, 0.08, 0.1\}$ de conectarse de manera aleatoria con otro nodo elegido al azar entre los nodos disponibles que no formen parte de su vecindad, todo esto con el fin de estudiar el comportamiento del flujo máximo entre un nodo inicial y uno final ocasionado por la percolación de aristas en grafos así definidos.

2. Descripción de la implementación del modelo

La disposición de grafos en forma de red se logró mediante la implementación del método `Cuadrado` a la clase `Grafo` del programa inicialmente mencionado:

```
1 def Cuadrado(self, nodos):
2   N = len(nodos)
3   lado = floor(sqrt(N))
4   for i in range(N):
5     n = nodos[i]
6     n.radio = 1 / N
7     n.posicion = ((i %lado) / (lado - 1), 1 - int(i / lado) / (
      lado - 1))
8
```

Esta función obtiene la longitud de una lista de nodos (línea 2), calcula el piso de la raíz cuadrada para obtener cuántos nodos ha de haber por lado (línea 3) y cada nodo de la lista se sitúa en una posición dada por esa cantidad de nodos por lado (línea 7).

Las conexiones en Manhattan se realizaron por el método `PasoManhattan` integrado a la clase `Grafo`:

```
def PasoManhattan(self, n, v, paso, nuevasVecindades = None):
  if self.dirigido:
    for  v1 in self.vecinos[v]:
      self.ConectarNodos(n, v1)
      if paso > 1:
        self.PasoManhattan(n, v1, paso - 1)
  else:
    for v1 in self.vecinos[v]:
      if n != v1:
        nuevasVecindades.append((n, v1))
      if paso > 1:
        self.PasoManhattan(n, v1, paso - 1, nuevasVecindades)
    return nuevasVecindades

for i in range(k - 1):
  if i %lado != lado - 1:
    G.ConectarNodos(G.nodos[i], G.nodos[i + 1])

  if int(i / lado) < lado and i < lado ** 2 - lado:
    G.ConectarNodos(G.nodos[i], G.nodos[i + lado])

a = [] # Nuevas vecindades
  for n in G.nodos:
    for v in G.vecinos[n]:
      a = G.PasoManhattan(n, v, l - 1, a)
  for par in a:
    G.ConectarNodos(par[0], par[1])

```

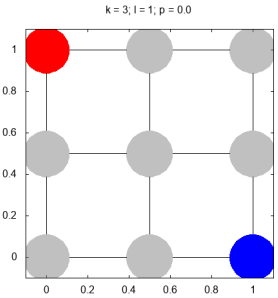
Este método da por cada nodo l pasos en Manhattan. Cada paso que da implica una conexión con un vecino, luego, de manera recursiva, los nodos siguientes dan $l-1$ pasos y se conectan con los nodos adyacentes a esa distancia Manhattan.

Los grafos resultantes para estos parámetros y con estas especificaciones se graficaron y se almacenaron en una animación `gif` que puede consultarse en `https://goo.gl/YEQLiz`. Asímismo, se muestran ejemplos de estos grafos en la figura 1 (p. 3).

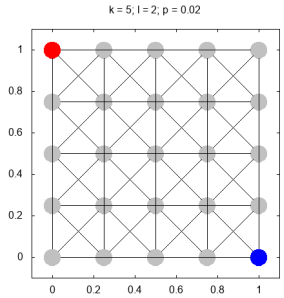
3. Implementación del código de percolación de aristas

El código de percolación de aristas consiste en una función que elimina arcos de manera aleatoria en los nodos creados de la manera descrita en la sección anterior. Para eliminar estas aristas es necesario eliminar de la clase `Grafo` las vecindades y pesos que se establecen de manera predeterminada al conectar nodos entre sí. Este proceso se repite mientras haya caminos que lleven flujo del nodo inicial al final. Tal procedimiento se codificó en las siguientes líneas:

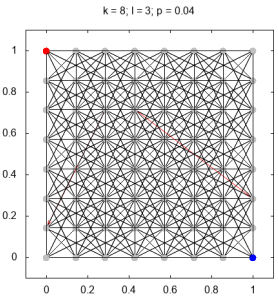
```
llega = True
  a = 0 # Cantidad de arcos eliminados
  while llega:
    tiempo = clock()
```



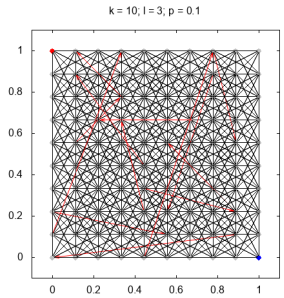
(a) Grafo con 9 nodos y $l = 1$ vecindades en distancia Manhattan.



(b) Grafo con 25 nodos y $l = 2$ vecindades en distancia Manhattan.



(c) Grafo con 64 nodos y $l = 3$ vecindades en distancia Manhattan.



(d) Grafo con 100 nodos y $l = 3$ vecindades en distancia Manhattan.

Figura 1: Ejemplos de grafos en red generados a partir de parámetros k, l, p donde los nodos inicial y final aparecen en la esquina superior izquierda en azul y en la esquina inferior derecha en rojo respectivamente.

```
    ford = G.Ford_Fulkerson(G.nodos[0], G.nodos[k - 1])
    tiempo = clock() - tiempo
    if ford == 0:
      llega = False

    n = sample(set(G.nodos), 1)[0]
    candidatos = set(G.vecinos[n])
    if len(candidatos) > 0:
      v = sample(set(G.vecinos[n]), 1)[0]
      if v:
        del(G.pesos[(n, v)])
        G.vecinos[n].remove(v)
        if n in G.vecinos[v]:
          del(G.pesos[(v, n)])
          G.vecinos[v].remove(n)
        a = a + 1

```

4. Ford–Fulkerson

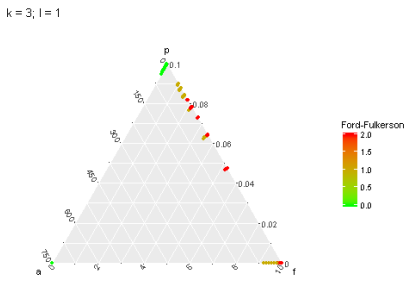
Se midió el flujo máximo entre el nodo inicial y el final de cada grafo generado con base en los parámetros destritos en este reporte, los resultados se graficaron para cada k, l, p en diagramas ternarios en los que se tomaron como ejes la probabilidad de establecer vecindades al azar p, el flujo máximo entre el nodo inicial y el final f y la cantidad de arcos eliminados a. Una muestra de los diagramas ternarios generados se muestra en la figura 2 (p. 5). Una animación de la secuencia de estos diagramas puede consultarse en `https://goo.gl/VFCgkr`.

Estas gráficas muestran que los flujos son máximos conforme aumenta la probabilidad p de establecer vecindades al azar y se mantiene en el mínimo la cantidad a de arcos eliminados, del mismo modo que aumentan cuando la distancia Manhattan para establecer vecindades aumenta también.

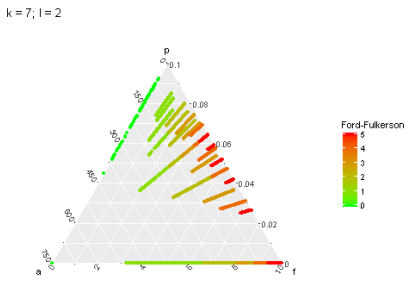
Para valores por encima de los 64 nodos con $l = 3$, la cantidad de arcos removidos debe ser al menos de un tercio del total de nodos para poner eliminar por completo el flujo entre el nodo incial y final de los grafos como los definidos en este reporte.

Referencias

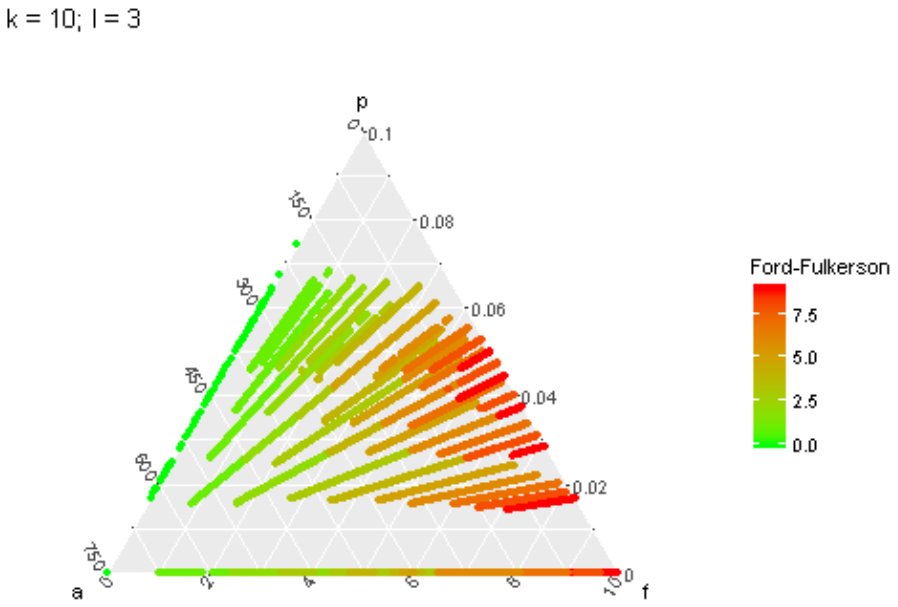
[1] José Alberto Benavides Vázquez. Grafos con python para flujo en redes. https://github.com/jbenavidesv87/FlujoRedes.



(a) Diagrama con $k = 3, l = 1$.



(b) Diagrama con $k = 7, l = 2$.



(c) Diagrama con $k = 10, l = 3$.

Figura 2: Ejemplos de diagramas ternarios donde se muestra la relación entre la probabilidad de establecer vecindades al azar p, el flujo máximo entre el nodo inicial y el final f, y la cantidad de arcos eliminados a.