Experimentos con el algoritmo de Ford-Fulkerson en `python`

Yessica Reyna Fernández

Flujo en Redes

7 de mayo de 2018

1. Introducción

En esta práctica se presentan resultados de una red de nodos cuadricular, buscando analizar cómo es impactado el flujo que se pueda generar entre las diferentes conexiones de los nodos entre sí, por ejemplo, como serían las redes eléctricas y redes de transmisión de datos.
Para la realización del grafo se usa la estructura de los grafos creados anteriormente dentro del mismo repositorio [1] e implementando el algoritmo de Ford-Fulkerson.

2. Creación de grafo

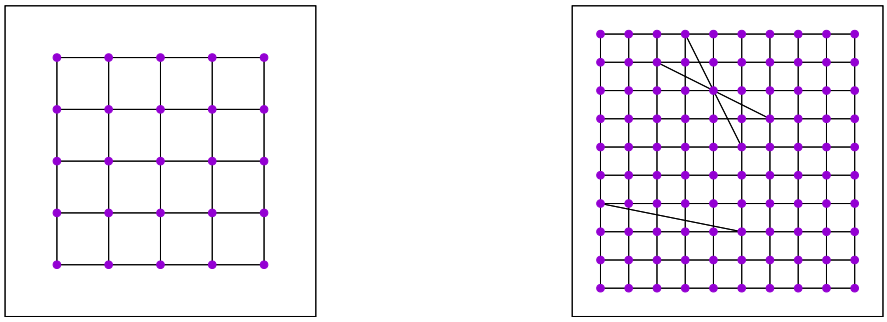
Para empezar, se determina la forma en la que se asignan las coordenadas a los nodos, ya que se busca formar una red de punto cuadrada, teniendo esto en cuenta se consideran las coordenadas de puntos enteros con una dimensión del tamaño deseado del grafo, definiendo en ese caso la variable k.
Asimismo, se considera otro para metro para la creación del grafo marcado con la variable l, con la cual se determinará la forma en la cual se conectan los nodos, es decir, esta variable determina la creación de las aristas en la red. Dependiendo del tamaño de l, es como se formarán las distancias, las cuales serán medidas en *distancias Manhattan* en donde la distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas.
Así pues, los nodos son ordenados de izquierda a derecha y creando las conexiones si las distancias son igual o por debajo del valor de l, además de que no se considera dirección en ellas, usando en este caso el par de aristas entre cada par de puntos, es decir, no se consideran iguales las aristas del nodo uno al nodo dos, que la arista del nodo dos al nodo uno. A continuación, se determina la ponderación de todas las aristas en el grafo, las que estén a una distancia Manhattan son dadas por una distribución normal con media de 5 y una desviación estándar de $5^{\frac{1}{2}}$, siendo definido lo ya dicho en las siguientes líneas de código.

```
    if i is not j and self.manhattan(i,j)<=l:
    x1=self.V[i][0]
    y1=self.V[i][1]
    x2=self.V[j][0]
    y2=self.V[j][1]
    self.pesos[(x1,y1),(x2,y2)]=self.pesos[(x2,y2),(x1,y1)]=floor(normalvariate(5,
        5**(1/2)))
    self.E[(i,j)]=self.E[(j,i)]=0
    self.vecinos[i].add(j)
    self.vecinos[j].add(i)
```

Acto seguido a esto se define la ponderación de las aristas aleatorias las cuales estarán dadas con una distribución exponencial con lambda de 0.1.

```
    if m is not w and (m,w) not in self.E:
    if random()< prob:
    x1=self.V[m][0]
    y1=self.V[m][1]
    x2=self.V[w][0]
    y2=self.V[w][1]
    self.pesos[(x1,y1),(x2,y2)]=self.pesos[(x2,y2),(x1,y1)]=ceil(expovariate(0.1))
    self.E[(m,w)]=self.E[(w,m)]=0
    self.vecinos[m].add(w)
    self.vecinos[w].add(m)
```

Algunos ejemplos de los grafos ya descritos son mostrados en la figura 1 para diferentes tamaños de la variable l.



(a) $n = 5$. (b) $n = 10$.

Figura 1: Distancia con $l = 1$.

3. Percolación

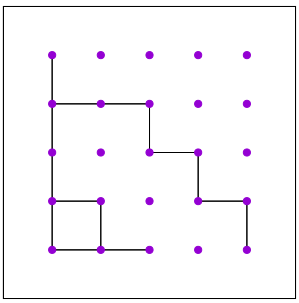
En relación con todo lo descrito con la sección anterior, ahora en lugar de buscar la creación de conexiones, se trata de eliminarlas, siendo esto para el uso de la implementación del algoritmo de Ford-Fulkerson, de lo cual se hablará sobre su propósito en la siguiente sección, se le dirá *percolación* a la eliminación ya sea de nodos (en otras palabras, dejar el nodo sin conexiones en la red) o de aristas, todo esto seleccionado aleatoriamente de la red.
Para la percolación de nodos se elimina uno a la vez, y en cuanto a las aristas

en la función implementada se garantiza la eliminación de 4 aristas seleccionadas aleatoriamente. A continuación, se muestra la función para lo ya descrito dentro del código, con la variable *percolacion* para determinar si se quiere hacer percolación en el grafo o no, y con las variables *nodo* y *arista* para determinar la percolación de nodos o aristas:

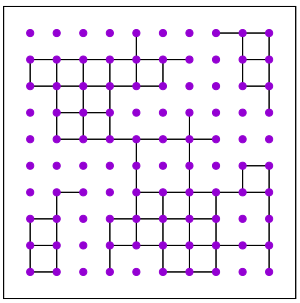
```
    arista=True
    nodo=False
    percolar=True
    G.percolacion(nodo,arista,l)
    if nodo is True:
    m=randint(1,k)
    del self.vecinos[m]
    if ((x1,y1),(x2,y2)) in self.pesos:
    del self.E[(m,b)] #donde b recorre los índices a los cuales está conectado m
    del self.E[(b,m)]
    del self.pesos[(x1,y1),(x2,y2)]
    del self.pesos[(x2,y2),(x1,y1)]
```

4. Flujo en la red

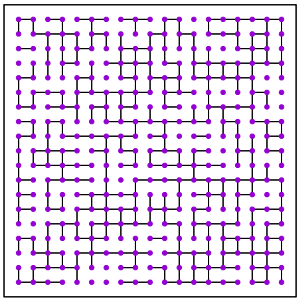
Usando la implementación ya realizada en un reporte anterior [2] del algoritmo de Ford-Fulkerson, con los grafos ya descritos se busca encontrar el flujo máximo dentro de la red ya sea que no se realice alguna percolación en ella o se haga alguno de los tipos de percolación, ya sea de nodos o de aristas. A continuación, se muestran algunos ejemplos de las redes formadas habiendo aplicado la función de percolación y el algoritmo de Ford-Fulkerson.



(a) $n = 5$.



(b) $n = 10$.

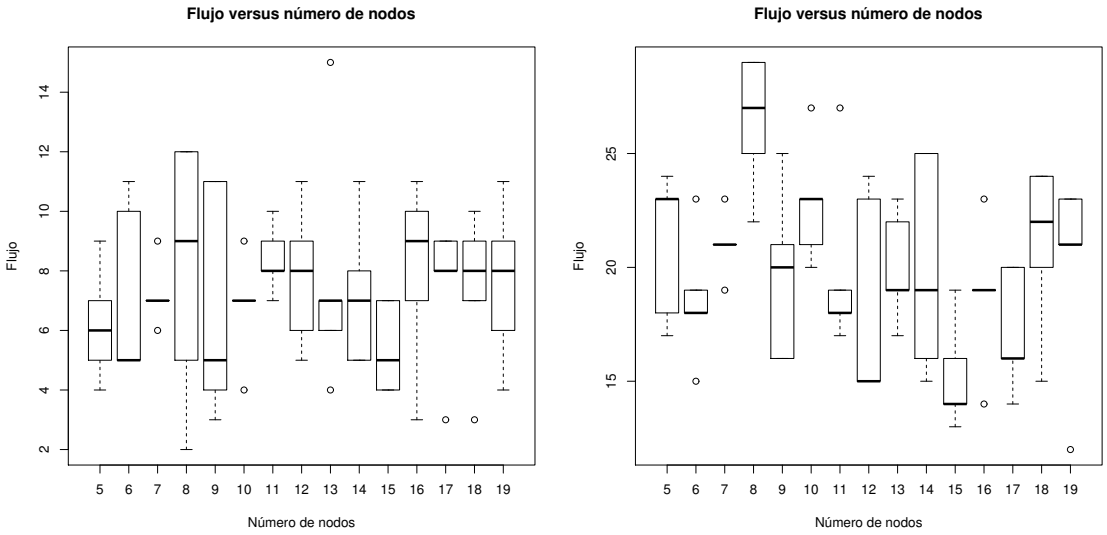


(c) $n = 19$.

Figura 2: Distancias en $l = 1$ con percolación de nodos.

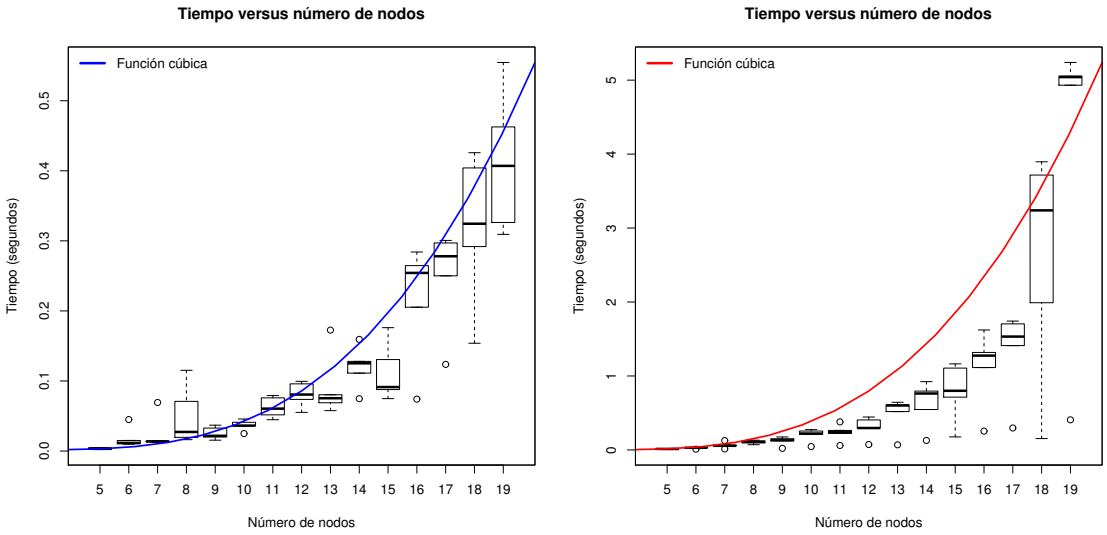
5. Resultados

Para finalizar se hablará sobre los resultados obtenidos de los flujos y los tiempos computacionales para tres tamaños diferentes de grafos, y además de dos diferentes variaciones de distancias de conexiones con $l = 1$ y $l = 2$.
Haciendo referencia hacia los tiempos, se puede observar que el comportamiento de los tiempos en los grafos que no percolaban tiene una tendencia cubica, debido a que entre más grande sea el grafo, es decir, la dimensión de la variable k definida como tamaño de dimensión aumenta, el tiempo aumenta de una forma polinomial, en particular con una tendencia cubica para estos resultados.
Mientras tanto en relación con los flujos, los datos reportados cada vez disminuyen poco a poco por la eliminación de un nodo cada vez que se introducía la percolación y limitaba el valor de flujo máximo dentro de la red por la desconexión de un nodo en ella, así mismo para las aristas eliminando no una sino 4 aristas por cada percolación, afectando de manera considerable el flujo por cada vez que se percolaba la red, donde todo esto puede ser mostrado en las figuras , donde las mismas son obtenidas a partir del manejo de la base de datos con el uso de `R`.



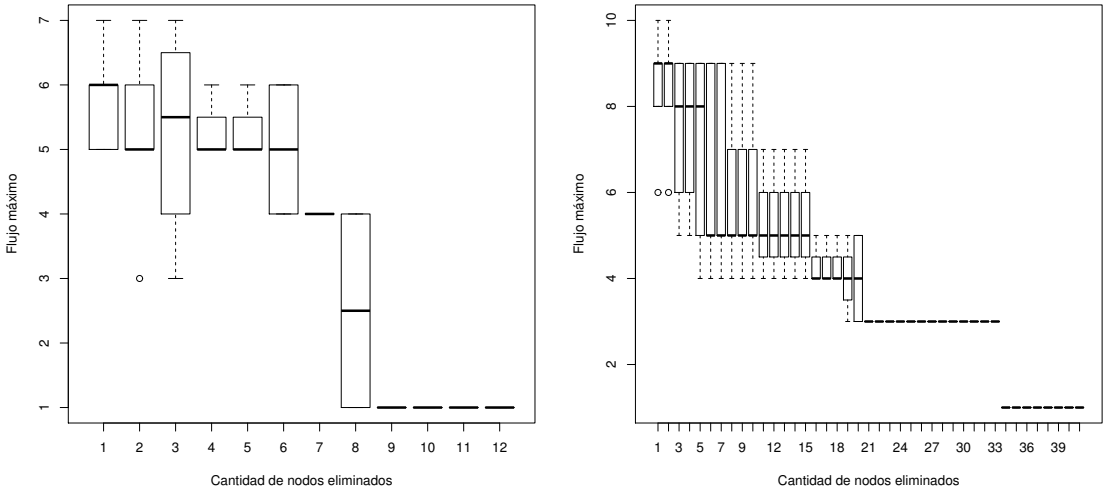
(a) $l = 1$. (b) $l = 2$.

Figura 3: Variaciones de flujo máximo obtenidos para diferentes tamaños de grafo sin percolación.



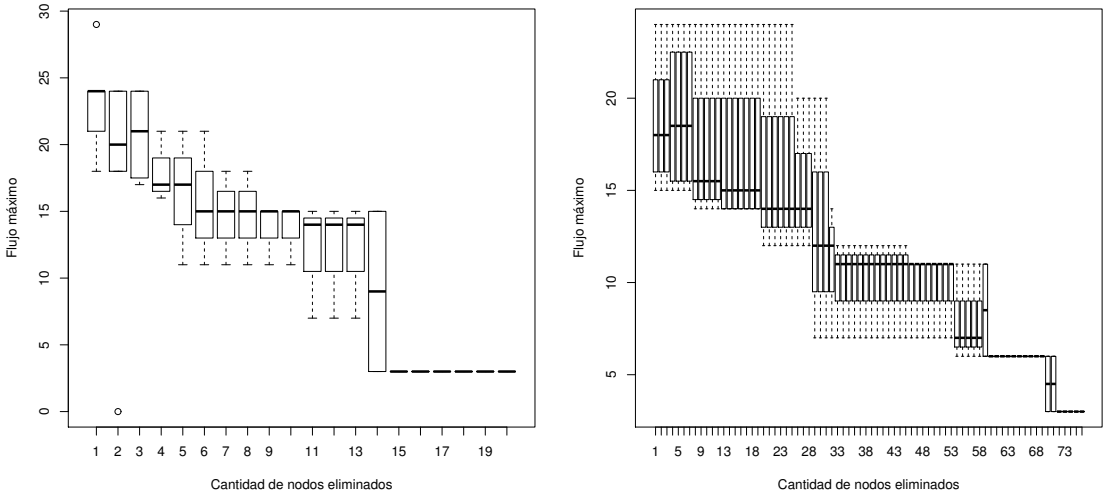
(a) $l = 1$. (b) $l = 2$.

Figura 4: Tiempos de ejecución para flujos máximos sin percolación.



(a) $n = 5$ y $l = 1$. (b) $n = 10$ y $l = 1$.

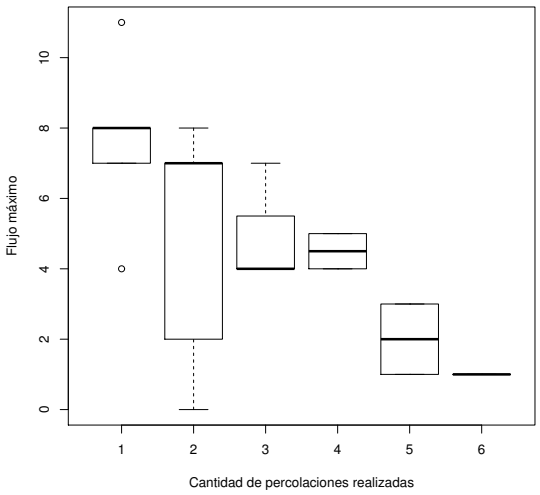
Figura 5: Variaciones de flujo máximo obtenidos para diferentes tamaños de grafo con percolaciones de nodos.



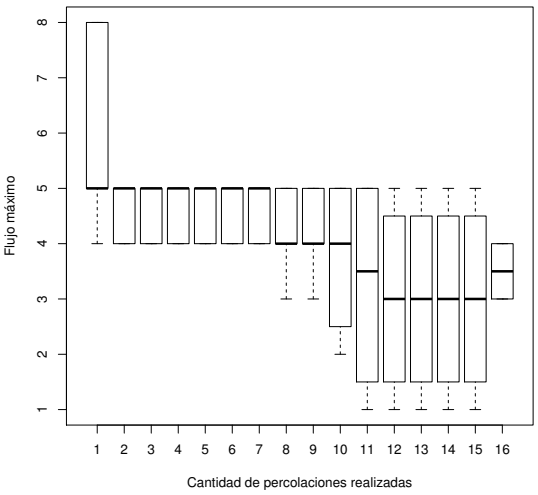
(a) $n = 5$ y $l = 2$.

(b) $n = 10$ y $l = 2$.

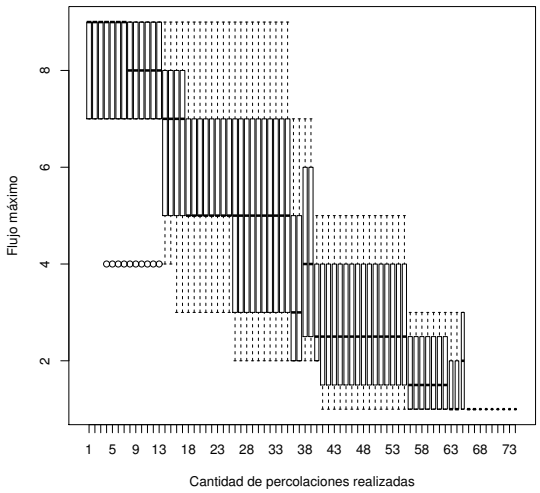
Figura 6: Variaciones de flujo máximo obtenidos para diferentes tamaños de grafo con percolaciones de nodos.



(a) $n = 5$ y $l = 1$.

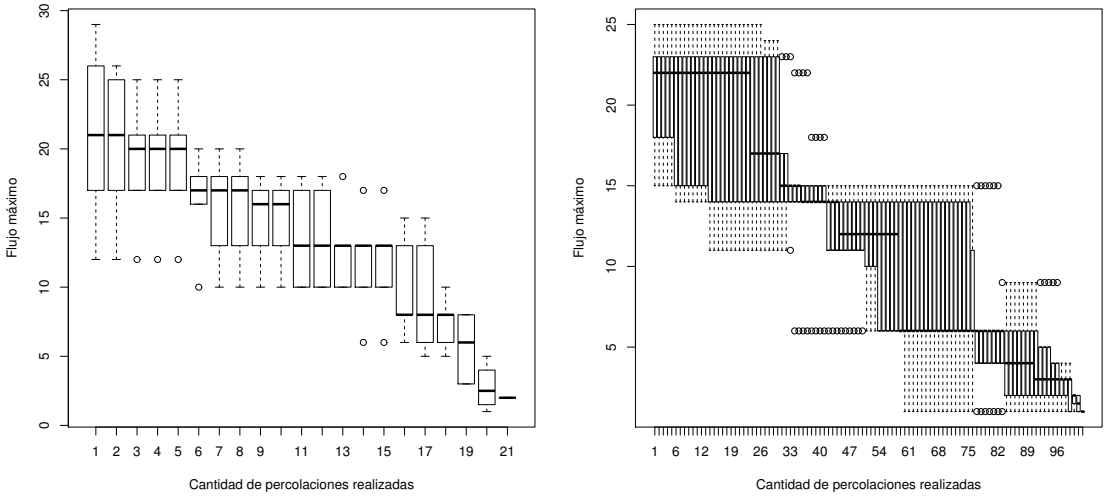


(b) $n = 10$ y $l = 1$.



(c) $n = 19$ y $l = 1$.

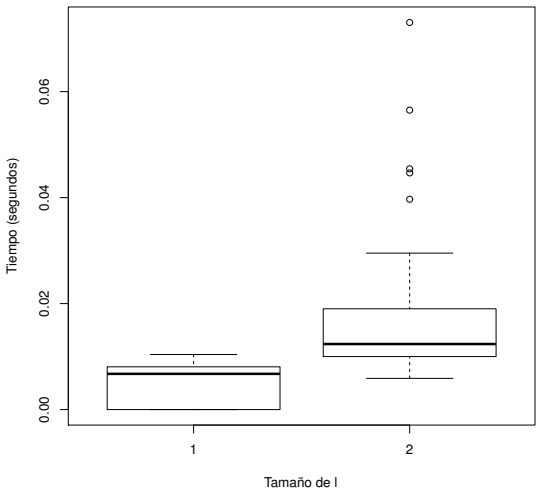
Figura 7: Variaciones de flujo máximo obtenidos para diferentes tamaños de grafo con percolaciones de aristas.



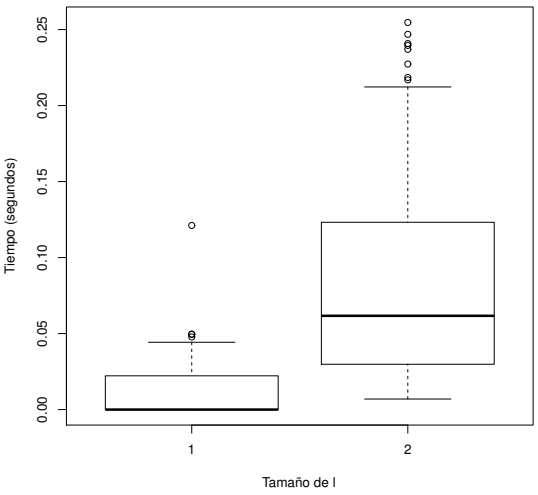
(a) $n = 5$ y $l = 2$.

(b) $n = 10$ y $l = 2$.

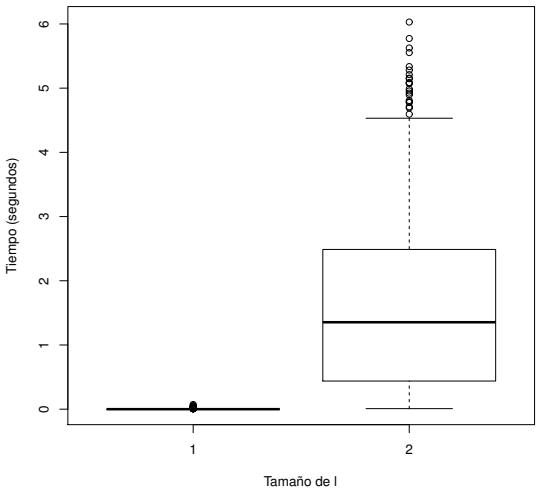
Figura 8: Variaciones de flujo máximo obtenidos para diferentes tamaños de grafo con percolaciones de aristas.



(a) $n = 5$.

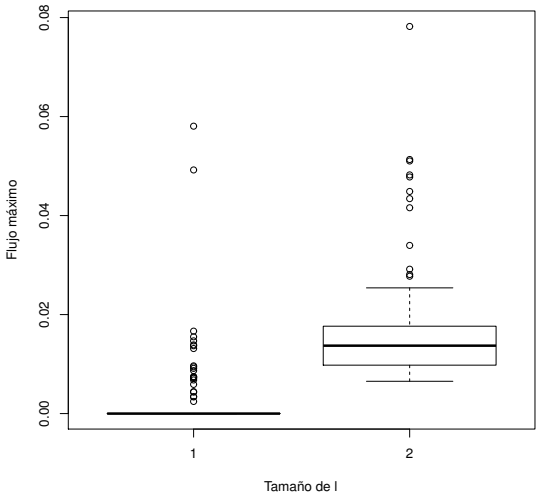


(b) $n = 10$.

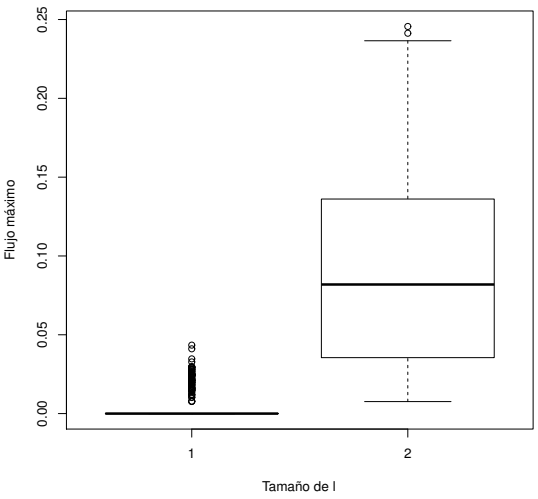


(c) $n = 19$.

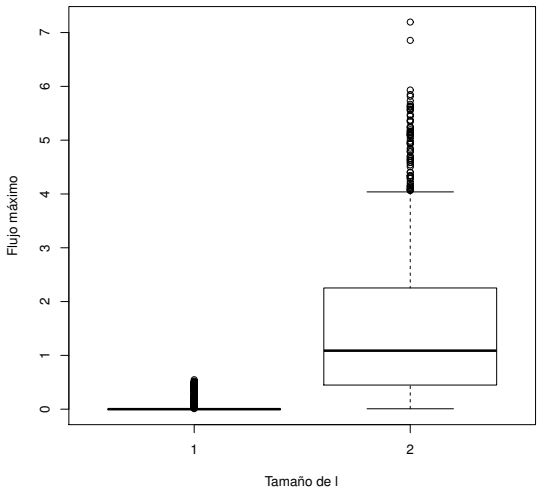
Figura 9: Tiempos de ejecución para percolaciones de nodos.



(a) $n = 5$.



(b) $n = 10$.



(c) $n = 19$.

Figura 10: Tiempos de ejecución para percolaciones de aristas.

Referencias

[1] Yessica Reyna Fernández. 2018. https://github.com/YessicaFer/FlujoenRedes

[2] Yessica Reyna Fernández. Tarea 3. 2018. https://github.com/YessicaFer/FlujoenRedes/blob/Tarea-1/Tarea%203/Tarea_3_Yessica.pdf