Diseño de los grafos

1484166

18 de marzo de 2019

Introducción

La importancia de los grafos es que con ellos se puede modelar ciertas procesos; encontrar los caminos mas cortos entre un nodo y otro pueden representar las conexiones sociales que existen entre una persona y otra en el caso que el grafo sea una red social. Para este y otros métodos más, la librería de `Networkx` incluye alguna variedad de algoritmos.

En este documento se implementan 5 algoritmos de `Networkx` y se mide su tiempo de ejecución.

Especificaciones técnicas

La computadora en la que se corrió los algoritmos es una Macbook Pro(13-inch,2017,Two Thunderbolt 3 ports) cuyo procesador es un Intel Core i5 de 2.3 GHz. Tiene una memoria RAM de 8 GB 2133 MHz LPDDR3

Algoritmos

Los algoritmos implementados fueron los siguientes.

`dfs_tree()`

Este algoritmo recibe como entrada un un grafo no orientado y te devuelve un árbol orientado hacia el retorno construido a partir de una fuente de búsqueda en profundidad.

`maximum_flow()`

Lo que hace este algoritmo es recibir un grafo dirigido o no dirigido que tenga capacidades, para poder devolver el valor del flujo máximo que va del nodo fuente al nodo sumidero y también devuelve un diccionario en el cual se especifica todos los arcos y el valor del flujo que corre por cada arco.

`all_shortest_pats()`

Este algoritmo proporciona el camino más corto de un grafo dirigido o no dirigido, entre un nodo inicial y un nodo final; devuelve todos los caminos con la misma cardinalidad que el del camino más corto.

`strongly_connected_components()`

En los grafos dirigidos, los componentes fuertemente conectados son subconjuntos de nodos en los cuales ir de un nodo a otro, es posible a través de cualquier otro nodo del subconjunto. Lo que hace este algoritmo es tomar grafos dirigidos y generar una lista que tiene todos los subconjuntos de componentes fuertemente conectados ordenados de la cardinalidad mas grande a la mas chica.

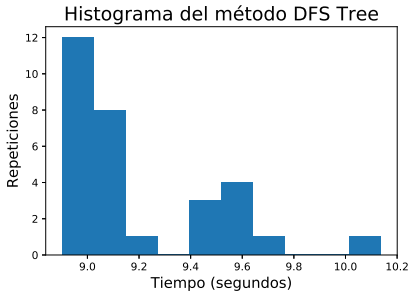
`treewidth_min_degree()`

Devuelve una descomposición del ancho de árbol usando la heurística de Grado Mínimo.

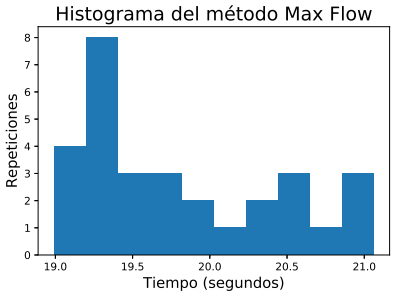
Metodología

Lo que se hizo para cada algoritmo es correr 5 grafos, cada uno con diferente cantidad de nodos y diferente cantidad de aristas, durante un tiempo de 5 segundos, a estas réplicas se les obtuvo la media, y lo mismo se hizo durante 30 veces, obteniendo así 30 datos de la media.

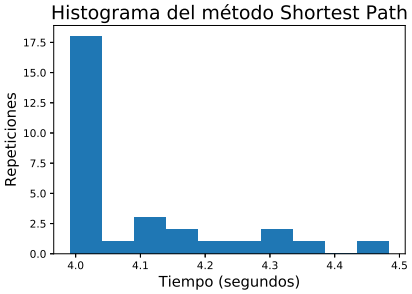
Con esa información se obtuvo los 5 histogramas de la figura 1



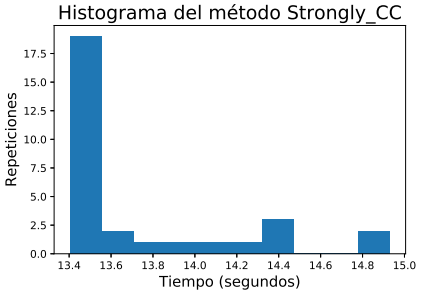
(a) Búsqueda a profundidad



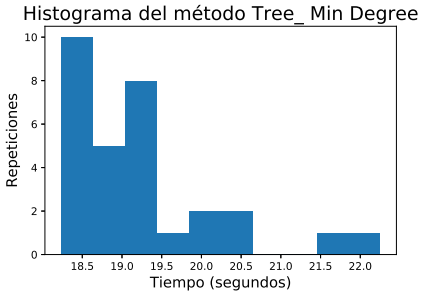
(b) Flujo máximo



(c) Caminos mas cortos



(d) Componentes fuertemente conectados



(e) Descomposición del ancho de árbol

Figura 1: Histogramas de tiempos promedios por algoritmo

Una vez guardada la información en una variable tipo diccionario, el código que da paso a los histogramas es el siguiente

```
for i in ('DFS Tree','Max Flow','Shortest Path','Strongly_CC','
    Tree_ Min Degree'):
    plt.xlabel('Tiempo (segundos)', size=14)
    plt.ylabel('Repeticiones', size=14)
    plt.title('Histograma del m todo '+i,size=18)
    plt.hist(dicrio[i]['medias_repeticiones'])
    plt.savefig(i+'.eps', format='eps', dpi=1000)
    plt.show()
```

Algoritmos_grafos.py

de donde se puede apreciar que no se aproximan a una gráfica normal.

Con las medias por algoritmo se graficó el tiempo promedio que se tardó en resolver las cantidades de nodos que se expresan como el eje y, como se puede observar en la figura 2.

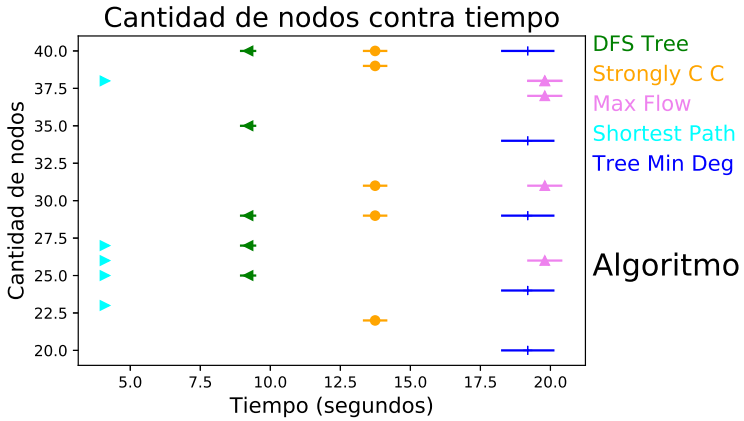


Figura 2: Error bar del tiempo promedio de ejecución vs la cantidad de nodos.

Al igual que se graficó el tiempo promedio que se tardó en resolver las cantidades de aristas que se expresan como el eje y, tal como se ilustra en la figura 3.

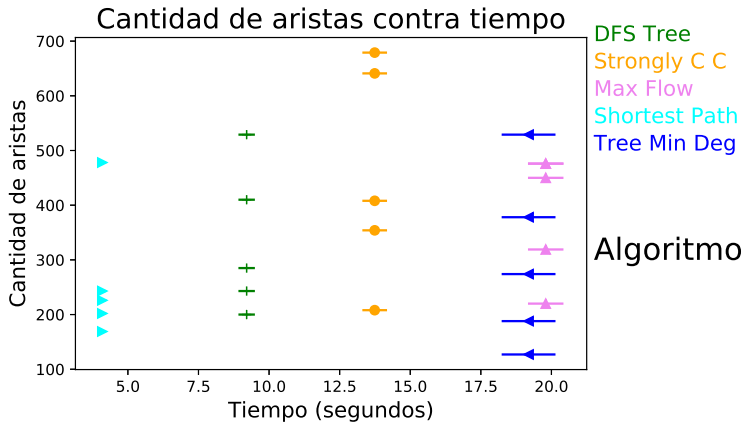


Figura 3: Error bar del tiempo promedio de ejecución vs la cantidad de aristas.

Conclusiones

Lo que se puede concluir es que el algoritmo mas rápido es el que encuentra los caminos mas cortos y el algoritmo mas complejo es el de flujo máximo.

Para graficar la error bar de la figura 3 se hace con el siguiente código.

```
y=[]
for i in ('DFS Tree','Max Flow','Shortest Path','Strongly_CC','
    Tree_ Min Degree'):
    for j in (1,2,3,4,5):
        y.append(dicrio[i][j]['aristas'])
for i in (1,2,3,4,5):
    if i==1:
        x=[dicrio['DFS Tree']['media_algoritmo']]
        xe=[dicrio['DFS Tree']['std_algoritmo']]
        plt.errorbar(5*x,y[0:5], xerr=5*xe, fmt='+',color='g',alpha
    =0.5)
    if i==2:
        x=[dicrio['Max Flow']['media_algoritmo']]
        xe=[dicrio['Max Flow']['std_algoritmo']]
        plt.errorbar(5*x,y[5:10], xerr=5*xe, fmt='^',color='violet'
    ,alpha=0.5)
    if i==3:
        x=[dicrio['Shortest Path']['media_algoritmo']]
        xe=[dicrio['Shortest Path']['std_algoritmo']]
        plt.errorbar(5*x, y[10:15],xerr=5*xe, fmt='>',color='aqua',
    alpha=0.5)
    if i==4:
        x=[dicrio['Strongly_CC']['media_algoritmo']]
        xe=[dicrio['Strongly_CC']['std_algoritmo']]
        plt.errorbar(5*x,y[15:20], xerr=5*xe, fmt='o',color='orange
    ',alpha=0.5)
    if i==5:
        x=[dicrio['Tree_ Min Degree']['media_algoritmo']]
        xe=[dicrio['Tree_ Min Degree']['std_algoritmo']]
        plt.errorbar(5*x,y[20:25], xerr=5*xe, fmt='<',color='blue',
    alpha=0.5)
plt.xlabel('Tiempo (segundos)', size=14)
plt.ylabel('Cantidad de aristas', size=14)
plt.title('Cantidad de aristas contra tiempo',size=18)
plt.text(21.5, 700, r'DFS Tree',color='g',size=14)
plt.text(21.5, 650, r'Strongly C C',color='orange',size=14)
plt.text(21.5, 600, r'Max Flow',color='violet',size=14)
plt.text(21.5, 550, r'Shortest Path',color='aqua',size=14)
plt.text(21.5, 500, r'Tree Min Deg',color='blue',size=14)
plt.text(21.5, 300, r'Algoritmo',color='black',size=20)
plt.savefig('scater2.eps', format='eps', bbox_inches='tight', dpi
    =1000)
```

Algoritmos_grafos.py

Referencias

[1] N. Developers, "Networkx documentation," 2012.