Complejidad asintótica experimental

Mario Alberto Gutiérrez Carrales
1549273

2 de abril de 2019

Introducción

En esta práctica el enfoque central son los algoritmos de generación de grafos y las implementaciones de algoritmos de flujo máximo, utilizando la librería de NetworkX [1].

Se plantean dos fases, la primera consiste en generar una matriz de datos de tiempos que representan el tiempo requerido que toma al algoritmo de solución resolver el grafo generado por el algoritmo de generación para esto se consideran algunos factores como réplicas de grafos, el número de nodos del grafo y la pareja de nodos fuente-destino.

La segunda fase consiste en realizar un análisis de varianza de un modelo donde el tiempo es la variable dependiente y las independientes son el algoritmo generador, algoritmo de solución, orden logarítmico y la densidad del grafo con el objetivo de saber efectos simples o de interacción son estadísticamente significativos.

1. Algoritmos

Generadores de grafos. Cada generador crea un grafo dependiendo los parámetros que este recibe, es común que reciban un parámetro omitiendo los demás donde dicho parámetro indica el número de nodos, aunque no en todos los casos es así. En los siguientes puntos se habla a detalle de los generadores que se estudian en esta práctica.

`Grid graph`. Genera una malla n-dimensional rectangular, se específica al generador la dimensión que se desea y el numero de nodos para cada componente. En el código 7 se muestra como se utiliza la función y los parámetros que necesita para un caso en donde la dimensión es dos y de forma cuadrada, al dar el parámetro n crea un grafo de n^2 nodos y $2(n^2 - n)$ aristas.

Código 1: *Ejemplo generador Grid_graph*

```
1  G = nx.grid_graph(dim=[x1, x1])
```

`Complete graph`. Este generador crea un grafo muy utilizado en la práctica que es el grafo completo, que es aquel en el que para cada par de vértices hay un camino que los une. La función tiene como parámetro el número de nodos que se desea que tenga el grafo. En el código 2 se muestra un ejemplo de esto. Si a este generador se le da como parámetro un entero n, entonces crea un grafo con n nodos y $c(n, 2)$ aristas.

Código 2: *Ejemplo generador Complete_graph*

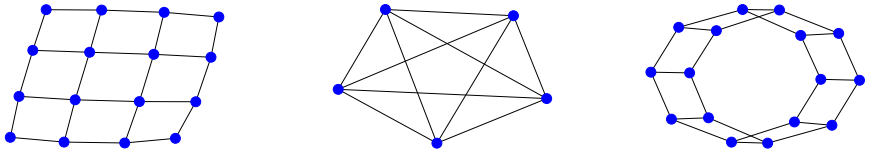
```
1  H = nx.complete_graph(x1)
```

`Circular ladder graph`. Al aplicar este generador se produce un grafo que consiste en dos ciclos de n nodos en donde cada uno de los n pares de nodos se unen por una arista. La función toma como parámetro un entero n y crea un grafo con $2n$ nodos y $3n$ aristas. En el código 3 se muestra como se lleva a cabo el uso de este generador.

Código 3: *Ejemplo generador Circular_ladder_graph*

```
1  W= nx.circular_ladder_graph(x1)
```

En la figura 1 se muestra un ejemplo de las visualizaciones para cada generador utilizado en esta práctica.



(a) Grid graph (b) Complete graph (c) Circular ladder graph

Figura 1: *Visualización de grafos creados con los algoritmos generadores.*

Implementaciones para el problema de máximo flujo. El problema de máximo flujo es un problema muy interesante en la optimización en redes, debido a ello se han desarrollado un gran número de implementaciones en diferentes softwares, en NetworkX hay una lista con una gran cantidad de implementaciones. En los siguientes puntos se describen aquellas que se utilizarán en esta práctica.

`Maximum flow`. Esta implementación retorna el valor de máximo flujo y además la cantidad que fluye por cada arco. En el código 4 se muestra un ejemplo de esta implementación.

Código 4: *Ejemplo generador Grid_graph*

```
1 Max_Flujo, Flujo_Arcos = nx.maximum_flow(G, (origen1
    ,origen2),(destino1,destino2), capacity='weight')
```

`Minimum cut value`. El valor de mínimo corte es retornado al llevar a cabo esta implementación. En el código 5 se muestra como se utiliza esta implementación.

Código 5: *Ejemplo generador Grid_graph*

```
1 cut_value = nx.minimum_cut_value(G, (origen1,origen2
    ),(destino1,destino2),capacity='weight')
```

`Maximum flow value`. Se retorna solamente el valor de máximo flujo. Se muestra un ejemplo en el código 6.

Código 6: *Ejemplo generador Grid_graph*

```
1          flow_value = nx.maximum_flow_value(G, (origen1,
    origen2),(destino1,destino2),capacity='weight')
```

En los tres casos la función recibe como parámetros el nodo fuente y destino, así como el atributo que tiene el papel de la capacidad. En las implementaciones para el problema de máximo flujo, si un arco no tiene una capacidad asignada se considera como infinita.

Se puede tener acceso a todos los algoritmos generadores de grafos y las implementaciones para el problema de máximo flujo a traves de los siguientes *links*:

Generadores.
`https://networkx.github.io/documentation/stable/reference/generators.html`

Implementaciones.
`https://networkx.github.io/documentation/stable/reference/algorithms/flow.html#module-networkx.algorithms.flow`

2. Ambiente computacional

Para llevar a cabo la experimentación se utiliza `python` [2] y para poder utilizar las funciones requeridas se necesitan las siguientes librerías:

1. `NetworkX`. Como se mencionó anteriormente, esta librería es de gran ayuda para la manipulación de grafos.
2. `Matplotlib` [3]. Esta librería contiene funciones que sirven para visualizar gráficas y guardar imágenes en distintos formatos, en particular el eps.
3. `Time` [4]. Contiene la función que determina el tiempo (en segundos) que uno o varios procesos tardan, dicho tiempo se calcula con una diferencia entre el tiempo inicial y el final.
4. `Numpy` [5]. Entre las funciones que proporciona esta librería, se encuentra aquella que calcula la media y desviación estándar de una lista, en este caso será necesaria para calcular tiempos promedios y desviaciones estándar entre los tiempos de algoritmos.
5. `Pandas` [6]. Una de las funciones con las que cuenta esta librería es la lectura de archivos con extensión csv y la manipulación de *data frames* facilitando el uso de tablas.
6. `Seaborn` [7]. Se utiliza para la creación del diagrama de caja y bigotes.
7. `Statmodels` [8]. Contiene varios modelos estadísticos y entre ellos esta el `ols` que sirve para ajustar formulas y entre los modelos a usar esta el anova_lm y ols.

Código 7: *Librerias utilizadas en la elaboración del código*

```
1 from statsmodels.formula.api import ols
2 import matplotlib.pyplot as plt
3 import statsmodels.api as sm
4 import networkx as nx
5 import seaborn as sns
6 import pandas as pd
7 import numpy as np
8 import time
```

Para guardar imagenes en formato eps se utiliza el comando que proporciona el fragmento de código 8.

Código 8: *Guardar imágenes en formato eps*

```
1       plt.savefig("Corrida" + str(contador) + ".eps")
```

Para evitar que los bordes de una imagen se recorten al momento de guardarla se le aplica el proceso que se muestra en el código 9.

Código 9: *Aumentar bordes para evitar cortes en el grafo*

```
1       plot_margin = 0.05
2
3       x0, x1, y0, y1 = plt.axis()
4
5       plt.axis((x0 - plot_margin,
6                 x1 + plot_margin,
7                 y0 - plot_margin,
8                 y1 + plot_margin))
```

El objetivo de la experimentación es realizar una comparación entre los tiempos de ejecución tomando en cuenta varios factores para resolver un problema de máximo flujo, para eso, se utiliza una laptop con sistema operativo de 64 bits y un procesador amd A9-9410 radeon R5, 5 compute cores 2C+3G 2.90 GHz.

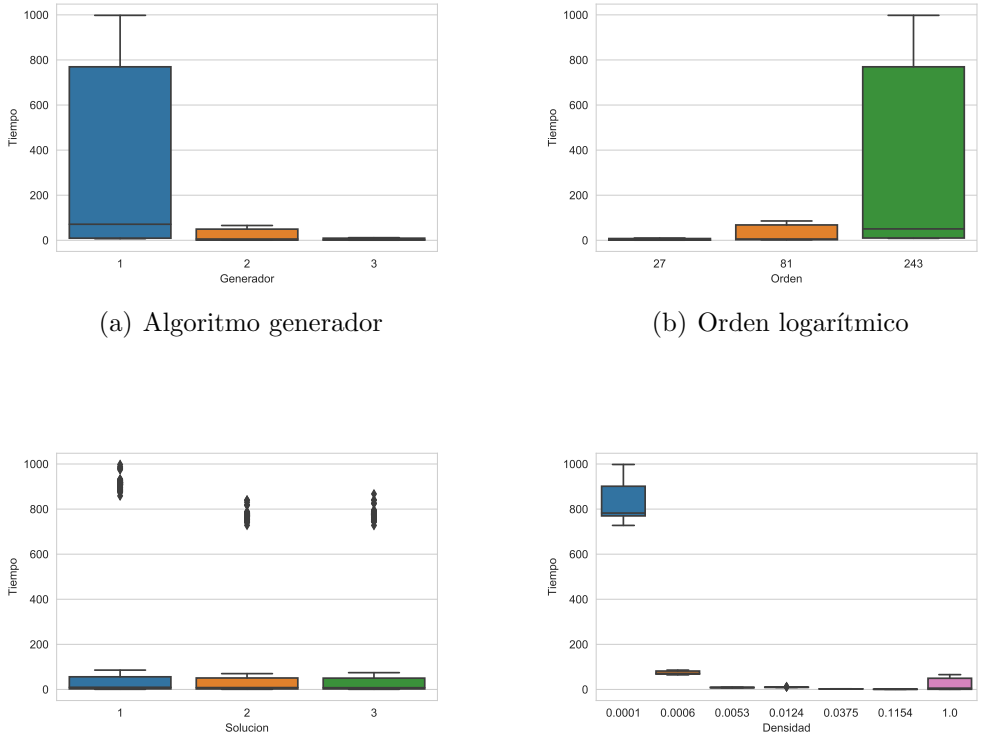
Para realizar la experimentación se consideran los siguientes parámetros:

- 3 Generadores (Grid, Complete y Circular ladder).
- 3 ordenes (27, 81 y 243).
- 10 Grafos distintos.
- 3 Implementaciones (Maximum flow, Minimum cut value y Maximum flow value).
- 5 parejas fuente-destino (aleatorias).

Se les asigna la variable aleatoria P a las capacidades de los arcos tal que $P \sim N(15, 5)$ y además para evitar tiempos nulos se consideran 100 repeticiones.

3. Experimentación

Una vez que se genera la matriz de datos con la caracterización de cada observación y su respectivo tiempo de ejecución se procede a la siguiente fase que es la experimentación estadística. En la figura 2 se observa como se comporta el tiempo de ejecución (en segundos) para los distintos grupos y niveles de cada grupo.



(a) Algoritmo generador

(b) Orden logarítmico

(c) Algoritmo de solución

(d) Densidad del grafo

Figura 2: *Variación del tiempo de ejecución por nivel de los diferentes grupos.*

En la figura 2 (a) se observa que el generador 1 es el que consume mayor tiempo y es un resultado que se esperaba puesto que este generador se aplica para crear grafos en 2 dimensiones, en (b) se hace énfasis a que entre mayor orden, mayor tiempo de ejecución, en (c) podría parecer que el efecto del algoritmo es nulo, es decir que no aportan variación en el tiempo de ejecución y en (d) se observa que a menor densidad mayor tiempo de ejecución aunque cuando la densidad es uno, tambien aumenta un poco el tiempo.

Posteriormente se efectúa un análisis de varianza (ANOVA) tipo dos para determinar los factores que tienen efecto sobre el tiempo, también es considerada la interacción entre los factores en el modelo para saber el tipo de efecto en la variable de respuesta.
Se considera el siguiente modelo matemático:

$$Y_{ijkl} = \mu_{...} + \alpha_i + \beta_j + \gamma_k + \delta_l + (\alpha\beta)_{ij} + (\alpha\gamma)_{ik} + (\alpha\delta)_{il} + (\beta\gamma)_{jk} + (\beta\delta)_{jl} + (\gamma\delta)_{kl} + \varepsilon_{ijkl}$$

Donde:

- $\mu_{...}, \alpha_i, \beta_j, \gamma_k, \delta_l, (\alpha\beta)_{ij}, (\alpha\gamma)_{ik}, (\alpha\delta)_{il}, (\beta\gamma)_{jk}, (\beta\delta)_{jl}, (\gamma\delta)_{kl}$ son parámetros que representan el efecto de cada nivel para cada factor simple y de interacción.
- $\varepsilon_{ijkl} \sim N(0, \sigma^2)$

y se realizan las siguientes pruebas estadísticas:

1. si cada factor tiene un efecto simple significativo en la variación del tiempo de ejecución
2. si cada pareja de factores tiene un efecto de interacción significativo sobre el tiempo de ejecución

En el cuadro 1 se muestran los resultados después de efectuar el ANOVA donde se destaca que dados los tiempos correspondientes se tiene que cada valor p (simple y de interacción) es menor a .05.

Cuadro 1: *Análisis de varianza del tiempo de ejecución.*

Fuente	**Suma de cuadrados**	**Grados de libertad**	**Razón F**	**P(>F)**
Generador	1.974622e+07	1	11894.0916	0.0000
Orden	3.012073e+07	1	18143.1582	0.0000
Solucion	7.021175e+04	1	42.291901	0.0000
Densidad	5.559616e+06	1	3348.82309	0.0000
Generador:Orden	1.292572e+07	1	7785.78226	0.0000
Generador:Solucion	8.767172e+04	1	52.808873	0.0000
Generador:Densidad	1.172786e+04	1	7.064253	0.0080
Orden:Solucion	9.736782e+04	1	58.649303	0.0000
Orden:Densidad	5.499527e+06	1	3312.62828	0.0000
Solucion:Densidad	2.271015e+04	1	13.679409	0.0002
Residual	2.222968e+06	1339	NA	NA

Del anova mostrado en el cuadro 1 podemos concluir que: se proporcionan valores p que son casi cero, por lo cual se puede decir que dada la elección de los párametros mencionados en el ambiente computacional se obtiene una gran variación del tiempo de ejecución dado cualquier factor en el modelo y también cualquier interacción, es decir, si se cambia de nivel en algún factor por si solo o combinación de factores es muy seguro que haya grandes cambios en el tiempo de ejecución.

Referencias

[1] NetworkX developers Versión 2.2. https://networkx.github.io.

[2] Python Software Foundation Versión 3.7.2. https://www.python.org/.

[3] The Matplotlib development team Versión 3.0.2. https://matplotlib.org/.

[4] Sulce A. PythonHow. https://pythonhow.com/measure-execution-time-python-code/.

[5] NumPy developers Versión 1.16.2. https://networkx.github.io.

[6] Augspurger T. and the pandas core team Versión 0.24.2. https://pandas.pydata.org/.

[7] Waskom M. Versión 0.9.0. Copyright 2012-2018. https://seaborn.pydata.org/.

[8] Perktold J. and Seabold S. Versión 0.9.0. Copyright 2012-2018. https://www.statsmodels.org/stable/index.html.