Paralelización y mejoramiento de la métrica *clustering*.

Muñiz Morales, Erick. erickmuniz@ciencias.unam.mx
Departamento de Matemáticas Aplicadas. Facultad de Ciencias, UNAM.

18 de agosto de 2021

Resumen: El tiempo computacional requerido para las métricas básicas de análisis de redes es bastante tardío. El siguiente texto se enfoca en la paralelización de la métrica *clustering* o coeficiente de agrupamiento. En específico, hacemos una optimización para el cálculo del número de triángulos. Posterior a este paso, se emplea una paralelización sobre los productos interiores a calcular para mejorar los tiempos. La redes de prueba son redes binomiales, aunque el algoritmo es robusto para redes no dirigidas ni pesadas; basta con su matriz de adyacencia. Se obtiene un rendimiento bastante bueno, mejorando por más de 10 veces el implementado en Python con la función *clustering()*.

Palabras clave: redes complejas, paralelización, optimización del algoritmo.

1. Introducción

Comprender la difusión de un ente, tangible o intangible, en un sistema es un problema de índole complejo. Los sistemas dinámicos no lineales son de las primeras aproximaciones a esta problemática. Si bien dichos modelos introducen una variable temporal con varios estados relacionados, tiene una sutil problemática al omitir la estructura topológica de los individuos del sistema así como su importancia e interacción. En resumen, se tiene un modelo simplista que supone homogeneidad, que generalmente no siempre es cierto, en beneficio de la obtención de resultados.

Para involucrar y recuperar dichas relaciones no intuitivas, el uso del análisis de las redes complejas permite recuperar lo perdido. Esta nueva modelación se basa en, principalmente, el uso de *grafos* o gráficas. Una gráfica es el par $G = (V, E)$ donde V es el conjunto de nodos y E el conjunto de enlaces o relaciones. Notemos que aquí se hace énfasis en la topología del sistema por la construcción de los enlaces.

Si bien esta esa una alternativa para aproximarnos a la cuantificación de la verdad de un sistema, resulta que tenemos un costo computacional alto. Diversas métricas se han definido en función del sistema en cuestión y, afortunadamente, muchas de estas tienen una misma abstracción y utilidad en distintos sistemas. Entre dichas métricas tenemos la centralidad intermedia la cual tiene un orden mayor al cúbico [1]. Entonces, dado que la redes de interés deseadas cuentan con una cantidad de nodos es superior a los 3000, es necesario una mejor implementación de los algoritmos.

En este documento, nos encargaremos del mejoramiento del algoritmo de la métrica de *clustering* o coeficiente de agrupamiento. La implementación y construcción de la misma se explicarán en las siguientes secciones así como se presenta un brebaje cultural respecto a la redes y sistemas complejos.

1.1. Sistemas complejos y redes.

Los sistemas complejos estudian la dinámica y forma de los componentes y su interacción entre sí; '*pueden espontáneamente auto-organizarse y presentar estructuras globales y comportamientos no-triviales a mayores escalas, sin intervención externa, autoridad central o líderes que determinen el comportamiento colectivo*'[2]. Si bien esto no es una definición formal, capta la importancia del estudio en estos sistemas: las relaciones.

Se han detectado propiedades en común sobre dichos sistemas. Como en el caso de su definición,

puede variar dependiendo del autor y el sistema a estudiar. Las propiedades que se consideran más importantes son cuatro: *emergencia*, *autoorganización* , *independencia* e *interdependencia*.

Las últimas dos propiedades muestran la robustez del sistema ante perturbaciones del mismo. En efecto, notaremos que existen relaciones clave en un sistema que, en la ausencia de estas, pueden colapsar todo el sistema. Así mismo, también encontraremos relaciones fuertes que, en la ausencia de estas, el sistema no sufrirá daño.

La *autoorganización* y la propiedad emergente destacan por la necesidad de cambio por la estabilidad o beneficio del sistema aún cuando no exista un líder o reglas formalmente escritas o explícitas. Pues, cada componente del sistema puede actuar por beneficio propio autoorganizándose y, a nivel global, generar un patrón de cambio repentino o un estado emergente.

Con base en estas ideas, la teoría de gráficas es objeto matemático ideal para el análisis de este tipo de sistemas [4]. En efecto, dado que existe una importancia a la relaciones entre los componentes del sistema, nace un nueva teoría que abarca esta necesidad: Teoría de redes.

Una **red** es una gráfica con interpretación. Aunque se conozcan métricas generales para tratar cualquier tipo de red, lo importante es la interpretación de las mismas. De estas métricas destacan métricas de centralidad local (como el coeficiente de agrupamiento *clustering)* y métricas globales que abstraen la topología de una red (como el coeficiente de asortatividad o modularidad) [5].

1.2. Sobre la matriz de adyacencia y la métrica *clustering*.

Toda la construcción de las herramientas usadas en el análisis de redes se basan en análisis de gráficas. Así, toda herramienta en gráficas es heredada a la la teoría de redes. Sin pérdida de generalidad, hagamos esta descripción de la red para una red simple; esto eso, una gráfica sin rizos y sin pesos. Notemos que esta condición no es limitante pues la casi todos los sistemas sociales tiene esta propiedad.

Principalmente, el objeto más útil ha sido la matriz de adyacencia (A). Esta matriz es la representación más simple de una red donde cada columna se puede abstraer como el estado de un nodo respecto a los demás. Matemáticamente, se define como

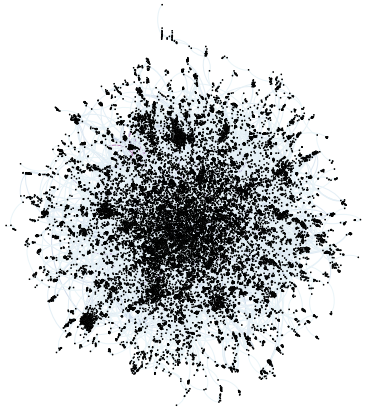

Figura 1: Ejemplo de una red social de usuarios interactuando en *Twitter*. Notemos que es importante generar un algoritmo que permita calcular métricas elementales sin tanto tiempo computacional.

$$A = \begin{cases} 1, & \text{Si } (i,j) \in E \\ 0, & \text{Si } (i,j) \notin E \end{cases}$$

Claramente, esta matriz es **simétrica** dado que consideramos una gráfica simple. Notemos que esta matriz se puede generar en función de la adyacencia de un nodo con los otros; esto eso, por cada nodo, tener un vector con entradas similares. De manera formal,definamos el vector columna $\Delta_u \in \mathcal{N}^{n\times 1}$ para algún nodo $u \in V$,

$$(\Delta_u)_j = \begin{cases} 1, & \text{Si } (u,j) \in E \\ 0, & \text{Si } (u,j) \notin E \end{cases}$$

En consecuencia de esta definición, tenemos que

$$\delta_u = \Delta_u^T \Delta_u \tag{1}$$

donde δ_u es el grado del nodo u. Así, la matriz de adyacencia se puede ver como

$$A = \begin{pmatrix} \Delta_{u_1} & \Delta_{u_2} & \cdots & \Delta_{u_n} \end{pmatrix}$$

Por otro lado, veamos el significado de la métrica en cuestión. Esta métrica de centralidad es local; esto es, vemos el *valor* del nodo en función de su primera vecindad. En esencia, calcula la razón de triángulos que están en la gráfica contra los ideales; pues, si queremos un nodo bien relacionado con sus vecinos, esperamos que los vecinos sean bien relacionados. Esta métrica tiene diversas interpretaciones donde la mayoría son de índole social [4, 6, 3].

Matemáticamente, se escribe de la siguiente manera. Sea $n \in \mathcal{N}$ el número de nodos. Sea $u \in E$ un nodo en la gráfica. Sea $\delta_u \in \mathcal{N}$ el grado del nodo (la cantidad de vecinos de u) y sea $\tau(u) \in \mathcal{N}$ el número de triángulos que contienen al nodo u. Entonces, el coeficiente de agrupamiento es

$$c_u = \frac{\tau(u)}{\binom{\delta_u}{2}} \tag{2}$$

$$= \frac{2\tau(u)}{(\delta_u)(\delta_u - 1)} \tag{3}$$

Este valor tiene como cotas 0 y 1. La división por $\binom{\delta_u}{2}$ es el ideal de triángulos. En particular, notemos que dicha métrica considera el conteo de triángulos. Cabe recordar que un triángulo es un ciclo de longitud 3. Así, una propiedad interesante de la matriz A es que la diagonal de A^3 nos dará el doble de estos valores ya que consideramos una red simple; entonces si tenemos un camino de idea, inmediatamente tenemos otro pero de regreso. Esta virtud será conveniente por lo que se ve en la fórmula (3).

Si definimos a $diag(A^3) \in \mathcal{N}^{n\times 1}$ como un vector, tenemos que (3) se puede definir como

$$c_u = \frac{diag(A^3)_u}{(\delta_u)(\delta_u - 1)} \tag{4}$$

Así, la ecuación (4) nos da un cálculo directo *clustering*.

Existen ya implementaciones de este algoritmo. Entre estos, el más conocido es la función ***nx.clustering(G)*** de la paquetería Networkx de Python. Dicha función, leyendo el código fuente, se tiene que esta no usa la matriz adyacencia como en el algoritmo antes mostrado. Al contrario, literalmente, busca estos estos triángulos como si *lo estuviéramos viendo*; aplica un bucle *for* para iterar sobre los nodos, busca sus vecinos y, sobre los vecinos, vuelve a buscar vecinos y encuentra los triángulos a partir de la coincidencia con ellos. Si bien es una implementación trivial y sencilla de leer, esta nos involucra un gasto mayor en tiempo computacional. A experiencia propia, pues los cálculos eran bastante tardíos y se necesitaba una mejora del mismo algoritmo.

n	Tiempo (segundos)
10	0.000249
100	0.0461
400	1.9275
800	15.4956

Cuadro 1: Segundo en el cálculo del coeficiente de agrupamiento para una red de n nodos. Notemos cómo a partir de cierto valor, el tiempo computacional se incrementa demasiado.

Notemos del cuadro (1), como a partir de 500 nodos, el tiempo se incrementa de manera acelerada. Así, esta es la raíz de la problemática y sobre esto se presenta el algoritmo que mejorará el tiempo de ejecución.

Como resumen, se coloca el pseudocódigo del algoritmo en (4), en su versión secuencial.

2. Metodología

Con base en lo visto anteriormente, aquí se presentará la versión mejorada del algoritmo. Esta versión no sólo será mejorada en su versión secuencial, sino que será optimizada a través del uso de la interfaz *OpenMP*. Toda la implementación se hará en C. Primeramente, presentemos la complejidad del método en (4).

Recordemos que calcular el producto de matrices cuadradas implica $nn(2n-1)$ operaciones, lo cual implica que este cálculo es de orden cúbico $\mathcal{O}(n^3)$. Así, al volver a multiplicar esta matriz resultante tenemos otra vez el mismo orden. Para el

Algorithm 1 Obtención del *clustering* en secuencial.

Require: Matriz de adyacencia A de dimensión $n \times n$

Definimos Vector_grados, Vector_diagonal, vector_clust.

for nodo **do**

Sumamos la entradas del vector Δ_{nodo}.

Guardamos la suma en vector_grados.

end for

Calculamos A^3 tal cual la definición formal con un hilo. ▷ Esta es la parte más tardada.

for nodo **do**

Extraemos el valor de la diagonal de A^3 asociado al nodo.

Guardamos el valor en vector_diagonal.

end for

for nodo **do**

Extraemos los valores en vector_grados (δ_{nodo}) y vector_diagonal asociados a el nodo (D_{nodo}).

Calculamos $\frac{D_{nodo}}{(\delta_{nodo})(\delta_{nodo}-1)}$.

Guardamos el valor en el vector vector_clust.

end for

último cálculo, llamámese la división en (4), tenemos que son n divisiones y n productos y $n-1$ sumas. Así, esto es de orden lineal $\mathcal{O}(n)$. Entonces, claramente, el orden de complejidad de (4), será de $\mathcal{O}(n^3)$.

Similarmente, el algoritmo presentado en *nx.clustering* de la paquetería de Python, tiene un orden de complejidad similar al cúbico. En efecto, si iteramos sobre los n nodos, verificamos sus vecinos; de ellos, verificamos los vecinos de los vecinos se hace costo computacional proporciona la n^3. Notemos a proporcional ya que, internamente, Python realiza operaciones son conjuntos que implican un costo mayor.

Entonces, la meta de este proyecto es reducir la complejidad y el costo computacional del algoritmo antes visto. Para ello, analicemos bien cómo es la forma de la matriz de adyacencia. Notemos que en ningún momento se usa el hecho que la matriz A sea simétrica y, dado que la potencia es pequeña, permite ver un uso directo. Por ejemplo, $(AA)^T = A^T A^T = AA$; esto es que la segunda potencia es simétrica. Entonces, tenemos, *apriori*, sólo se requieren $\frac{n(n+1)}{2}$ productos interiores.

$$A^2 = \begin{pmatrix} \delta_{u_1} & \Delta_{u_1}^T\Delta_{u_2} & \Delta_{u_1}^T\Delta_{u_3} & \dots & \Delta_{u_1}^T\Delta_{u_n} \\ \dots & \delta_{u_2} & \Delta_{u_2}^T\Delta_{u_3} & \dots & \Delta_{u_2}^T\Delta_{u_n} \\ \dots & \dots & \delta_{u_3} & \dots & \Delta_{u_3}^T\Delta_{u_n} \\ \dots & \dots & \dots & \delta_{u_{n-1}} & \Delta_{u_{n-1}}^T\Delta_{u_n} \\ \dots & \dots & \dots & \dots & \delta_{u_n} \end{pmatrix} \tag{5}$$

Por otro lado, notemos que la diagonal de A^2 (véase (5)) es el grado de cada nodo. Así, sólo debemos calcular $\frac{n(n-1)}{2}$ productos interiores (la triangular superior) ya que la diagonal principal se obtiene en el primer paso ya que son los grados. Estas $\frac{n(n-1)}{2}$ entradas serán calculadas por medio de una estrategia de paralelización; la cual es entre los hilos disponibles repartir estos productos interiores. Su complejidad, hasta este momento, es de orden $\mathcal{O}(\frac{n(n-1)}{2}(2n-1)) = \mathcal{O}(n^3)$; si bien tenemos una misma complejidad a un producto matricial usual, es claro que reducimos al mitad de los cálculos.

A partir de esto, tenemos el cálculo de A^2; nos falta A^3. Sin embargo, notemos que de A^3 úni-

camente necesitamos la diagonal. Entonces, claramente, sólo requerimos n productos interiores de vectores y columnas que ya se conocen y son independientes entre sí. Así, esta parte también será paralelizada distribuyendo estos n productos interiores entre los hilos que nos proveerá la máquina. Como parte extra, es claro que tendremos un complejidad cuadrática $\mathcal{O}(n^2)$ puesto que son n productos interiores.

Notemos que el último paso es restar, multiplicar y dividir nuestros cálculos como en la (4). Evidentemente, también se paralelizará distribuyendo estos n operaciones. Así, la complejidad del mismo es de orden lineal $\mathcal{O}(n)$.

Así, si bien esta nueva forma de ver el algoritmo pertenece al orden cúbico, es esperable que podamos bajar su tiempo computacional al repartir los productos interiores entre los hilos que nos proveé la máquina. En resumen, el pseudocódigo de nuestro algoritmo mejorado y paralelizado está en el recuadro de algoritmo (2).

3. Resultados y discusión

Para hacer un análisis de los resultados obtenidos, se usarán redes binomiales. Una red binomial es una red donde la existencia de las aristas dependen de un parámetro de probabilidad. Especifícamente, si dos nodos cuales quiera, hay una probabilidad p de que exista este enlace en la red. Si bien es un caso particular en el mundo de las redes, no es pérdida de generalidad para ver el rendimiento de la implementación del algoritmo. Pues, sólo ocupamos su matriz de adyacencia. Además, diversos análisis de redes son de redes simples. Aún así, notemos que la única limitante es sobre el peso de la red; sin embargo, con esta perspectiva, se necesita un análisis más particular para ver si este algoritmo le es útil. .

Dicho lo anterior, la prueba se hará sobre una computadora con tres cpu's lógicos, uno físico y tres lógicos,donde cada uno nos permite un hilo. Por lo cual, tenemos tres hilos como la cantidad máxima para usar en el algoritmo visto en la metodología con *OpenMP*.

Finalmente, obtenemos los resultados generales expresados en el siguiente cuadro (2). Notemos que si bien cuando la cantidad de nodos es muy pequeña los tiempos entre los algoritmos nos

Algorithm 2 Optimización y mejoramiento del algoritmo 1.

Require: Matriz de adyacencia A de dimensión $n \times n$
Definimos matriz_a2[n][n], vector_clust[n].
Empezamos una región de paralelización. ▷ Notemos que aquí distribuimos estos cálculos entre los hilos permitidos.
for nodo u_i hasta $nodo_n$ **do**
  for nodo $v_j = u_i + 1$ hasta $nodo_n$ **do**
    if $u_i == v_j$ **then**
      Obtenemos su grado
      Sumamos los valores de Δ_{u_i}
      Lo guardamos en la entrada (i, i) del arreglo matriz_a2
    else if $u_i \neq v_j$ **then**
      Calculamos $\Delta_{u_i}^T \Delta_{v_j}$
      Lo guardamos en la entrada (i, j) y (j, i) del arreglo matriz_a2
    end if
  end for
end for
Terminamos una región de paralelización.
Empezamos una región de paralelización.
for nodo u_i hasta $nodo_n$ **do**
  Calculamos el producto interior de Δ_{u_i} con el renglón i del arreglo matriz_a2. Definimos este valor como D_{u_i}.
  Obtenemos su grado con el valor de la diagonal de matriz_a2 asociado a u_i. Digamos a este valor como δ_{u_i}.
  Calculamos $\frac{D_{u_i}}{(\delta_{u_i})(\delta_{u_i}-1)}$ y lo guardamos en vector_clust.
end for
Terminamos una región de paralelización.

distan demasiado, se nota la mejoría del algoritmo cuando tenemos una n más grande. Esto resultados son esperables dada la construcción antes mencionada. Además, notemos que, si bien el algoritmo 1 nunca se paraleliza ni se optimiza, aún le sigue ganando al proporcionado en Python.

n	Python	Algo. 1 (sequencial)	Alg. 2 (Paralelo)
5	0.000138	0.000005	0.000099
85	0.02650	0.005055	0.001149
105	0.04152	0.01153	0.002175
505	3.45675	1.274484	0.214452
785	14.23974	4.925218	0.706059

Cuadro 2: Tiempo en segundos en calcular el coeficiente de agrupamiento para una red binomial con n nodos.

Ahora si bien tenemos que sólo se ha considerado los cálculos hasta menos de 800 nodos. Sin embargo, por el análisis anterior estas eficacias entre los algoritmo mantendrán el mismo orden. De forma visual, lo podemos ver en el gráfico 2

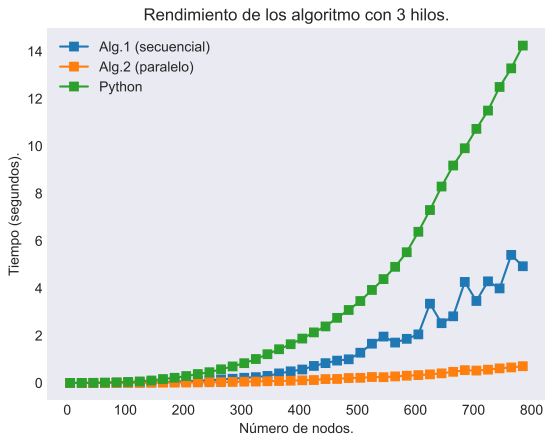

Figura 2: Comparación entre los tiempo de ejecución de los algoritmos. Entre más abajo esté la gráfica es mejor. Notemos que el algortimo 2 (paralelo) es el mejor.

Otra forma de visualizar los rendimientos entre los algoritmos, es el cálculo de los dos *speed up*. Esto es la razón de rendimiento entre un dos algoritmos. Matemáticamente, es la división del tiempo que le costó un algoritmo en realizar la rutina entre el tiempo que le costo al otro algoritmo en realizar la misma rutina. Aquí el orden es importante para tener una mejor interpretación. En este sentido, sólo haremos este cálculo para compara los algoritmos presentados contra el de Python y la diferencia entre los algoritmos; esto es, el *speed up* entre el algoritmo de Python cotra el algoritmo 1 (secuencial) , algoritmo de Python cotra el algoritmo 2 (paralelo) y el algoritmo 1 contra el algoritmo 2.

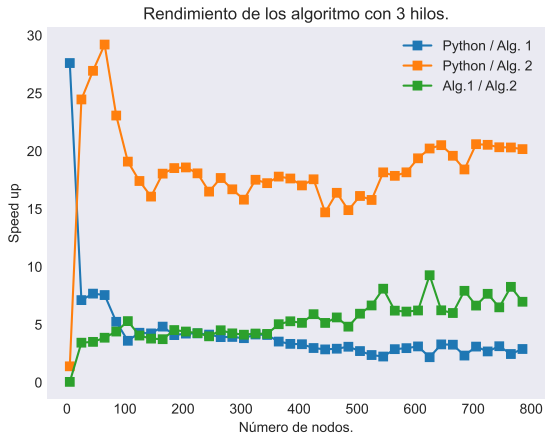

Figura 3: *SpeedUp* entre los algoritmos. Entre mayor sea el valor es mejor para el algoritmo de la derecha. Notemos que existe un ruido para menos de 100 nodos.

Del gráfico 3, podemos ver como el algoritmo 2 (paralelo) resulta ser el mejor de todos. Esto en consecuencia de su optimización y de la paralelización del mismo. Si bien tenemos que el algoritmo paralelo muestra un mejor rendimiento (5 a 8 veces más) que el algoritmo 1 (secuencial), el mejor rendimieto del algoritmo paralelo se lo lleva contra Python. Pues, el algoritmo 2 (paralelo) tiene es 15 veces más rápido que el algoritmo incorporado en Python.

4. Conclusión

El análisis de los algoritmos siempre nos permite mejorar aun cuando no lo paralelizamos. Y aunque paralelizar no lo es todo, siempre mejoramos el algoritmo si se hace un análisis previo del mismo.

En este documento, se logró el objetivo de me-

jorar el rendimiento del algoritmo en Python al menos 10 veces. Y, para realmente hablar de una comparación, se mejora el algoritmo secuencial al menos 5 veces.

Si bien esto datos son bastante buenos, siempre está en duda la calidad de la codificación mostrada para hacer aun más eficiente el uso de recursos; notemos que usamos una matriz simétrica. Además, si bien es un algoritmo particular para el análisis de redes, se necesitarán pocas modificaciones para obtener alguna métrica en particular. Por lo cual, es flexible en su entendimiento y fácil utilidad.

Así como conclusión, siempre debe ser importante analizar un algoritmo antes de paralelizarlo aún cuando la paralelización siempre será buena. Esto último, ya que implica un uso más eficiente de los recursos.

Además, esto induce a que las métricas relacionadas con redes sean definidas en función de matrices; ya que la matriz de adyacencia no la única forma de representarlas.

Referencias

[1] Ulrik Brandes. A faster algorithm for betweenness centrality$*.TheJournalofMathematicalSociology, 25(2) : 163 - -177, June2001.$

[2] Manlio De Domenico and Hiroki Sayama. Complexity explained. 2019.

[3] Srijan Kumar, William L. Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. *CoRR*, abs/1803.03697, 2018.

[4] Melanie Mitchell. *Complexity : a guided tour*. Oxford University Press, Oxford England New York, 2009.

[5] M. E. J. Newman. *Networks*. Oxford University Press, Oxford, 2018.

[6] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.