Experimentos con el algoritmo aproximado de contracción para el corte mínimo en Python

Gabriela Sánchez Y.

Introducción

Hemos visto anteriormente [3] que la complejidad del algoritmo de Ford-Fulkerson, cuando las capacidades de las aristas son reales, depende del número de nodos n y aristas m en el grafo y de una cota para todas las capacidades U. Esto es, el tiempo de ejecución del algoritmo es del orden $\mathcal{O}(mnU)$. De manera que, para calcular el flujo en grafos con un número grande de nodos y aristas, se requerirá un tiempo de ejecución elevado.

El objetivo de la práctica es analizar una alternativa para calcular el flujo máximo en un grafo. De acuerdo al teorema de Flujo máximo - Mínimo corte [1], el valor del flujo máximo del nodo fuente s al nodo destino t en una red capacitada es igual a la mínima capacidad de todos los cortes del nodo s al nodo t.

Algoritmo aproximado de contracción

El algoritmo aproximado de contracción resuelve el problema de mínimo corte, que se basa en contraer aristas [2]. Al contraer la arista (u, v), se reemplazan los nodos u y v por un nodo nuevo w. La arista que se contrae desaparacere y todas las aristas (u, s) con $s \neq v$ se reemplazan por (s, w). De manera análoga con las aristas (s, v), $s \neq u$. Después de las contracciones, obtenemos dos conjuntos $\mathcal{C}$ y $\mathcal{C} \setminus \mathcal{V}$, que corresponden a los dos nodos que quedan en la última iteración. El corte mínimo corresponde a las aristas que cruzan del conjunto $\mathcal{C}$ al conjunto $\mathcal{C} \setminus \mathcal{V}$.

La implementación se realiza de la siguiente manera: cada que una arista es contraída el vértice u "absorve" al vértice v creando el vértice $u - v$, la arista contraída es eliminada, las aristas (u, r) son reemplazadas por $(u - v, r)$ y las

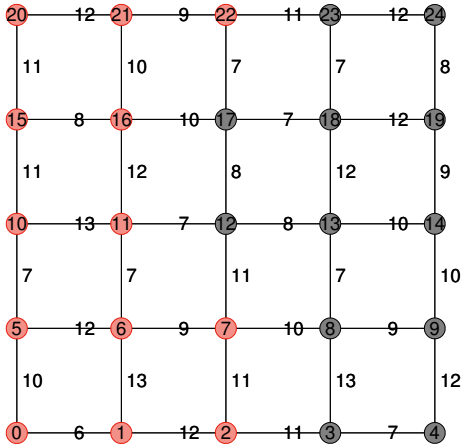

Figura 1: Resultado ejemplo del algoritmo de aproximación.

aristas $(u - v, d)$ son reemplazadas por $(u - v, d)$, si la arista ya existe, las capacidades se suman.

La elección de la primera arista a contraer se realiza de manera al azar. Ya que nos interesa encontrar una cota mínima para el corte mínimo las siguientes se contraen de acuerdo a su capacidad, es decir, se contraen las aristas con mayor capacidad. En cada paso se verifica que sea posible contraer la arista, ya que el nodo fuente s y el nodo destino t no deben pertenecer al mismo conjunto. Si la arista no puede contraerse, se elige otra de manera al azar hasta que ya no haya conflicto.

Al terminar el algoritmo nos dará una cota para el corte mínimo, el objetivo es estudiar el número de iteraciones necesarias para que la cota obtenida iguale al flujo máximo que es previamente calculado con el algoritmo de Ford-Fulkerson. La implementación se encuentra en el archivo `practica6.py`. La figura 1 muestra un ejemplo de un corte para un grafo con $k = 5$.

Los experimentos se realizan en grafos tipo malla como los creados en la práctica anterior [4], con $l = 1$, $k = 10, 15, 20$ y sin aristas de largo alcance. En las figuras 2-4 se muestran los resultados de algunos ejemplos.

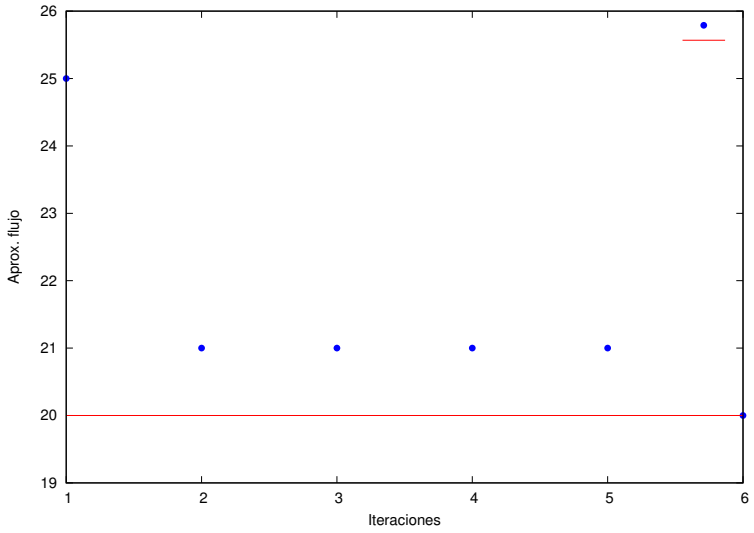

Figura 2: Cotas para el flujo máximo para un grafo con $k = 10$.

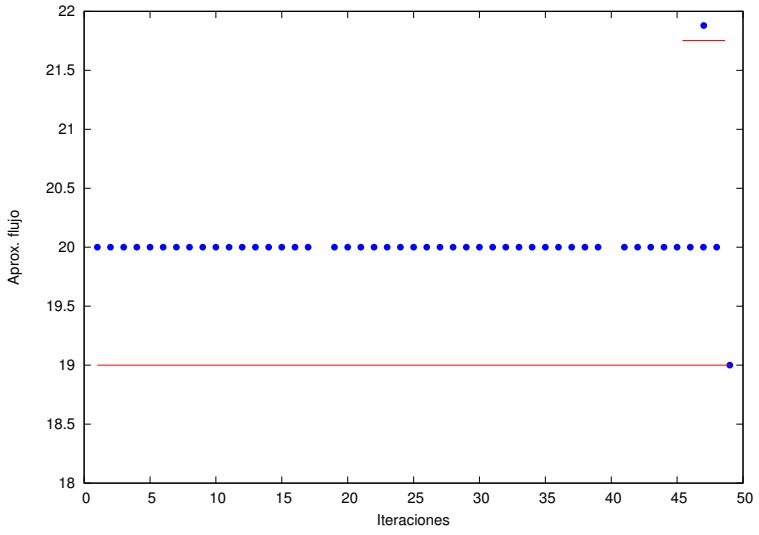

Figura 3: Cotas para el flujo máximo para un grafo con $k = 15$.

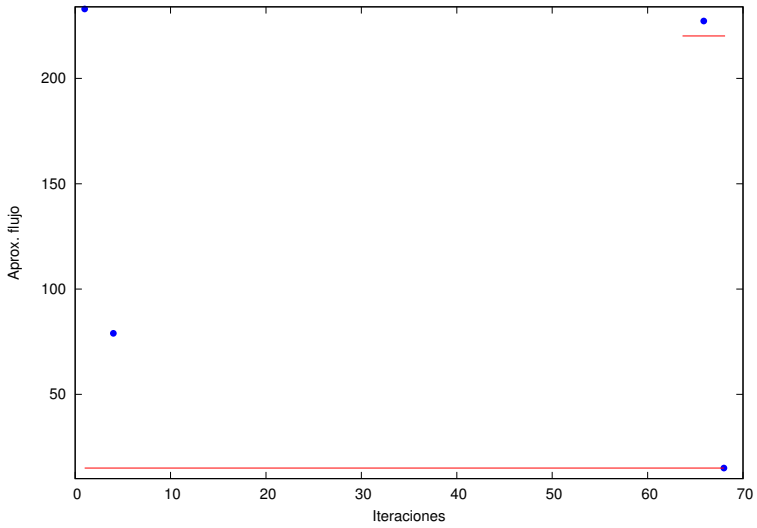

Figura 4: Cotas para el flujo máximo para un grafo con $k = 20$.

Diferencias en los tiempos de ejecución

Dependiendo del tamaño del grafo fijamos un margen de error en la cota para el flujo máximo. Si $25 < k < 50$ permitimos una diferencia entre la cota y el flujo de tres unidades. Una vez que establecido lo anterior, el siguiente objetivo es analizar la diferencia en los tiempos de ejecución del algoritmo de Ford-Fulkerson y el algoritmo aproximado de contracción.

La figura 5 muestra las diferencias obtenidas en los tiempos de ejecución de ambos algoritmos para encontrar el valor del flujo máximo. El experimento se realizó para grafos con $k = 5, 10, 15, 20, 25$ y 30. Si la diferencia es negativa, significa que el algoritmo de Ford-Fulkerson encontró el valor en un menor tiempo, si es positivo el algortimo de aproximación fue mejor.

Referencias

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] Elisa Schaeffer. Complejidad computacional de problemas y el anĺisis y diseño de algoritmos. https://elisa.dyndns-web.com/teaching/aa/pdf/aa.pdf.

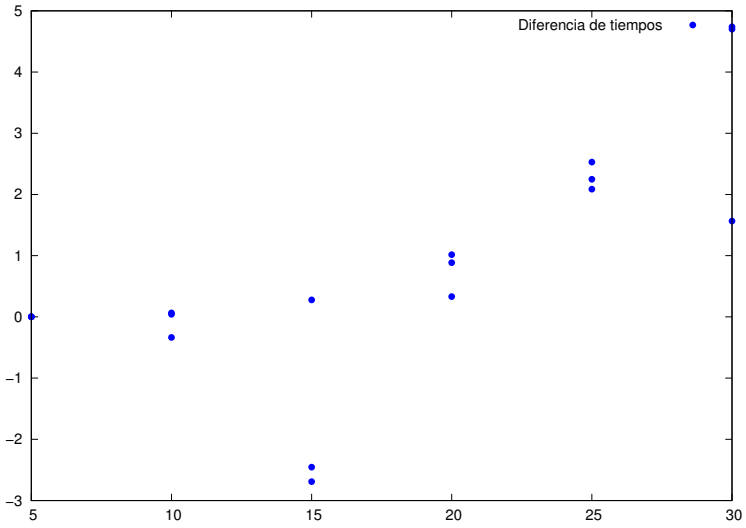

Figura 5: Diferencia en los tiempos de ejecución para encontrar el flujo máximo.

[3] Gabriela Sánchez Yepez. Práctica 3: Medición experimental de la complejidad asintótica con python y gnuplot. https://github.com/Saphira3000/Flujo_Redes/tree/master/tarea3.

[4] Gabriela Sánchez Yepez. Práctica 5: Experimentos con el algoritmo de ford-fulkerson en python. https://github.com/Saphira3000/Flujo_Redes/tree/master/tarea5.