Dense Subgraph Problem

Definición general:

En teoría de grafos, la densidad de un grafo es una propiedad que determina la proporción de aristas que posee. Un grafo denso es un grafo en el que el número de aristas es cercano al número máximo de aristas posibles, es decir, a las que tendría si el grafo fuera completo. Al contrario, un grafo disperso es un grafo con un número de aristas muy bajo, es decir, cercano al que tendría si fuera un grafo vacío.
https://es.wikipedia.org/wiki/Densidad_(teor%C3%ADa_de_grafos)

Definición formal:

Sea G = (V, E), un grafo simple (sin bucles) con n = |V| vértices y m = |E| aristas. Si el grafo es no dirigido, su densidad se define formalmente como:

La densidad varia entre 0, para grafos vacíos con m = 0, y 1, para grafos completos con el máximo número de aristas posibles, a saber, m = n(n-1)/2. Si el grafo es dirigido, su densidad se define como:

En este caso, la densidad alcanza valor 1 para el máximo número de aristas posibles en un grafo dirigido completo, a saber, m=n(n-1). Si el grafo es además ponderado, sean W = {w_1, …, w_m} los pesos de las aristas, entonces la densidad del grafo se define como el promedio de los valores asignados a las aristas:

Problema del grafo denso en las ciencias de la computación:

https://en.wikipedia.org/wiki/Dense_subgraph

En ciencias de la computación la noción de subgrafos altamente conectados aparece con frecuencia. Esta noción se puede formalizar de la siguiente manera. Dejar G = (E, V) sea un grafo no dirigido y sea ser un subgrafo de G. Entonces la densidad de S se define como

El problema del subgrafo más denso es el de encontrar un subgrafo de máxima densidad. En 1984, Andrew V. Goldberg desarrolló un algoritmo de tiempo polinomial para encontrar el subgráfico de densidad máxima utilizando una técnica de flujo máximo . Esto ha sido mejorado por Gallo, Grigoriadis y Tarjan en 1989 [1] para funcionar en tiempo.

SUBGRAFO MAS DENSO:

Hay muchas variaciones en el problema del subgrafo más denso. Uno de ellos es el problema del subgrafo k más denso, donde el objetivo es encontrar el subgrafo de máxima densidad en exactamente k vértices. Este problema generaliza el problema del Clique y, por lo tanto, es NP-difícil en gráficos generales. Existe un algoritmo polinomial que aproxima el subgrafo k más denso dentro de una proporción de $n^(\fract{1}{4}+c)$ para cada $e > 0$ si bien no admite una $n^(\fract{1}{poly \log \log n})$-aproximación en tiempo polinomial a menos que la hipótesis del tiempo exponencial sea falsa. Bajo una suposición más débil de que , no existe PTAS(Polynomial-time approximation scheme) para el problema.

El problema sigue siendo NP-hard en grafos bipartitos y grafos cordales, pero es polinomial para árboles y grafos divididos . Está abierto si el problema es NP-duro o polinomial en gráficos de intervalos (propios) y gráficos planos ; sin embargo, una variación del problema en el que se requiere conectar el subgrafo es NP-hard en grafos planos.

APLICACIÓN:

Para grafos dirigidos, Zeleny (1941) definió el índice de asociación de un nodo como la diferencia entre la densidad o media de la «intensidad» total de la red, y el grado de salida o «elecciones» realizadas por el nodo. Denotemos Is(v) el índice de asociación del nodo v. Entonces lo anterior se define formalmente como:

La densidad se utiliza en análisis de redes sociales desde sus inicios en los años 1950 al menos a partir de Kephart (1950) y Proctor y Loomis (1951) (estos últimos responsables de la centralidad de grado). La densidad es una propiedad relevante para las redes sociales representadas como grafos, que se puede considerar como una medida de centralización de la red. Bott (1990) la propuso como una forma de cuantificar los niveles de «entrelazado» de una red, y Barnes (1969) para determinar la «estrechez» de las uniones de redes empíricas, importante en modelos de bloque y otras técnicas algebraicas de análisis. Además, la densidad de un subgrafo sirve para evaluar la cohesión de subgrupos dentro de una red social.

- REDES SOCIALES: (https://www.cs.princeton.edu/~zdvir/apx11slides/charikar-slides.pdf)

Rastreando la Web en busca de ciber comunidades emergentes [KRRT '99]: las comunidades web se caracterizan por subgrafos bipartitos densos

- Biología computacional: Extracción de subgrafos densos coherentes en redes biológicas masivas para el descubrimiento funcional:

- el subgrafo de interacción de proteínas densas corresponde a un complejo de proteínas
- el subgrafo de coexpresión densa representa un grupo de coexpresión estrecho

PROBAR QUE UN SUBGRAFO DENSO ES NP COMPLETO:

(https://www.geeksforgeeks.org/prove-that-dense-subgraph-is-np-complete-by-generalisation/)

Problema: Dada la gráfica G = (V, E) y dos enteros a y b. Un conjunto de varios vértices de G tales que hay al menos b aristas entre ellos se conoce como subgrafo denso del grafo G.

Explicación: Para probar el problema del subgrafo denso como NP-completo por generalización, vamos a probar que es una generalización del conocido problema NP-completo. En este caso, vamos a tomar a Clique como el problema conocido que ya se sabe que es NP-completo, y se explica en Prueba de que Clique es un NP-Completo y necesitamos mostrar la reducción de Clique → Subgrafo denso.

Nota: Clique es un subconjunto de vértices de un gráfico no dirigido tal que cada dos vértices distintos en la camarilla son adyacentes.

Prueba:

1. Conversión de entrada: se requiere convertir la entrada de Clique a la entrada del Subgrafo denso.

Clique Input: An undirected graph **G(V, E)** and integer **k**.

Dense Subgraph Input : An undirected graph **G'(V, E)** and two integers **a** and **b**.

Se transforma la entrada de Clique para un sub grafo denso tal que:

- G’ = G(V, E)
- a = k
- b = (k * (k – 1))/2

Esta conversión tomará un tiempo O(1), por lo que es de naturaleza polinomial.

2. Conversión de salida: Se requiere convertir la solución de subgrafo denso a la solución del problema Clique.

La solución del grafo denso dará como resultado un conjunto a que sería un Clique de tamaño k cuando k = a. Por lo tanto, Clique puede utilizar la salida directa del grafo denso. Dado que no se requiere conversión, es nuevamente de naturaleza polinomial.

3. Corrección: Se restringó el rango del valor de entrada b tal que (k¦2) con valor como (k * (k – 1))/2.

Ahora estamos buscando un subgrafo que tenga k vértices y que estén conectados por al menos (k * (k – 1))/2 aristas.

Dado que en un grafo completo, n vértices pueden tener como máximo (n * (n – 1))/2 aristas entre ellos, podemos decir que necesitamos encontrar un subgrafo de k vértices que tengan exactamente (k * (k – 1)))/2 aristas, lo que significa que el gráfico de salida debe tener una arista entre cada par de vértices que no es más que Clique de k vértices.

De manera similar, un Clique de k vértices en un gráfico G(V, E) debe tener (k * (k – 1))/2 aristas que no es más que el subgrafo denso del gráfico G(V, E)

Entonces, esto significa que el subgrafo denso tiene una solución↔ Clique tiene una solución.

La reducción completa toma un tiempo polinomial y Clique es NP completo, por lo que el subgrafo denso también es NP completo.

Por lo tanto, podemos concluir que el subgrafo denso es NP Complete

HERE

PROBAR QUE UN SUBGRAFO DENSO ES NP COMPLETO: (https://www.tau.ac.il/~hassin/dense_02.pdf)

Se tiene lo siguiente:

Teorema 1. (k; f(k))-DSP es NP-completo para f(k) = \theta(k^{1+e}) donde e puede ser alguna constante positiva menor que uno.

Teorema 2. (k; f(k))-DSP es NP-completo para f(k) = ek^2/v^2(1 + O(v^{e −1})) donde e puede ser cualquier constante positiva menor que uno.

Es obvio que (k; f(k))-DSP está en NP. Para probar que está en NP-hard, reducimos un problema Clique a (k; f(k))-DSP. Aquí consideramos el problema Clique restringido que pregunta si existe un grafo completo de n-vértices (n-clique) en un 2n-vértice dado grafo G = (V, E). Se puede demostrar fácilmente que este problema del Clique todavía es NP-completo por reducción del problema general k-clique de la siguiente manera:

Para un grafo de entrada I de n vértices, sumamos un grafo completo de n − k-vértices k_{n-k}, conexión bipartita completa entre I y k_{n-k} y k vértices aislados. El grafo consecuente tiene 2n vértices, y tiene un Clique n si y solo si existe un Clique k en I.

Sea f(k) = nm(2n − 1) + n(n − 1)=2, donde m es un polinomio en n y determinado

luego. Construya un gráfico H = (V', E') compuesto por una copia G de G = (V; E) y m

grafos completos, cada uno de los cuales tiene 2n vértices. H tiene |V'| = 2n(m + 1) vértices y

|E'| = |E| + nm(2n − 1) aristas en total. Luego establezca k = 2nm + n. Esta construcción de H

se puede hacer en tiempo polinomial obviamente.

Lema 6. Supongamos que hay m grafos completos $I_1, …, I_m$ de 2n vértices y un

(no necesariamente completo) grafo G de 2n vértices. Tome 2nm+n vértices entre esos

2nm + 2n unos. Entonces, el número de aristas inducidas se vuelve máximo cuando tomamos todos los vértices de 2nm de $I_1, …, I_m$ (y otros n de G).

Prueba: Supongamos que tomamos 2nm−d vértices de *$I_1, …, I_m$* y n+d vértices de G. (Vea la Fig. 1 para m = 2 y d = 3.) Entonces uno puede ver fácilmente que el número de aristas incluidas no disminuye si abandonamos (cualquiera) d (tres en la Fig. 1) vértices de G y tome d vértices de I_1 e I_2, ya que ambos completos. Esta declaración es obviamente cierto para m y d generales. Así sostiene el lema.

Este lema muestra que un subgrafo de k-vértice más denso de H consta de todos los Clique de 2n vértices y un subgrafo de n vértices de G. Si el número de aristas en este subgrafo es f(k), entonces el número de aristas en el subgrafo tomado de G debe ser n(n − 1)=2; es decir, debe ser una camarilla. Por el contrario, si G tiene una camarilla de tamaño n, entonces obviamente es posible tomar un subgrafo de k-vértices de f(k) aristas.

Para la demostración del Teorema 1, queda demostrar que dado 0< e <1, se puede elegir f(k) de modo que cumpla la condición f(k) = \theta(k^{1+e}). Para cualquier fijo 0 <e<1 dado, elegimos m = n^{1/e−1}, de modo que k = 2n^{1/e} +n y f(k)=2n^{1+1/e} −n^{1/e} +n(n−1)=2. Así, en términos generales, k^{1+e}/2^{e+1} < f(k) < k^{1+e}/2^{e−1}, es decir, f(k) =

\theta(k^{1+e}). Entonces, como f(k) = |E'|k^2/|V'|^2 (1 + O(m^{−1})) y m = O(|V'|^{ 1−e}), también obtenemos el Teorema 2 del Teorema 1.

ENCONTRAR SUBGRAFOS MÁS DENSOS EN TIEMPO LINEAL

Sabemos que encontrar el subgrafo óptimo más denso se puede calcular en tiempo polinomial. En este ejercicio, intentamos implementar un algoritmo que calcule una aproximación del subgrafo más denso en tiempo lineal de la entrada. Vamos a implementar un algoritmo codicioso de 2 aproximaciones.

2 IMPLEMENTACIÓN EN TIEMPO LINEAL DE UN ALGORITMO DE 2 APROXIMACIONES

2.1 El algoritmo

2.2 Implementación de tiempo lineal

Para mayor claridad, vamos a explicar la implementación en Python. Más adelante daremos una implementación en C++ que se ejecuta aún más rápido. El código se puede segmentar en 3 pasos, la inicialización que produce todas las variables y datos necesarios para el algoritmo greedy, el algoritmo descrito anteriormente y el almacenamiento de los datos.

3 JUSTIFICACIÓN

3.1 Paso 1

- inicialización Antes de ejecutar el Algoritmo 1, necesitamos procesar los datos.

Este paso requiere leer el archivo txt o csv que contiene todas las aristas del grafo, cada arista está en una línea diferente. Almacenamos todos los bordes en una lista, siendo cada borde un par de nodos. Al iterar a través de las líneas, también almacenamos el nodo con la identificación máxima, asumiendo que el gráfico contiene todos los nodos entre 0 y el nodo con la identificación máxima. Cada paso es constante en el tiempo, por lo tanto, una complejidad de tiempo lineal $O(|E|)$.

list_adjacent(edges,max_node)

Esta función calcula la lista de adyacencia creando una lista de longitud el número de nodos y recorriendo las aristas en tiempo constante en cada paso. Así, una complejidad de tiempo en $O(|V|) + O(|E|) = O(|E|)$.

grade_function(list_adj)

Esta función da una lista con el grado de cada nodo. Devuelve D donde D[i] es el grado del nodo i. Itera la lista de adyacencia con tiempo constante en cada paso. Por lo tanto, una complejidad de tiempo en $O(|E|)$.

list_nodes_by_degree(D)

Esta función devuelve L donde L[i] es una lista de nodos de grado i. Crea una lista de longitud por debajo de |V| e itera a través de la lista de grados D con cada paso que se ejecuta en tiempo constante. Por lo tanto, una complejidad de tiempo en $O(|E|)$. También creamos una lista de aristas y nodos eliminados, también almacenamos $|V|$ y $|E|$ que se actualizará. En conclusión, el paso de inicialización se ejecuta en un tiempo lineal $O(|E|)$.

3.2 Paso 2 - Algoritmo greedy

•mientras G contiene al menos una arista:

- El criterio de la parada se puede reemplazar comprobando si se han eliminado todos los nodos. while number_nodes != len(eliminados): Esto se puede hacer en tiempo constante verificando si el número de nodos eliminados es igual al número inicial de nodos $|E\_G|$.

•Sea v el nodo con grado mínimo en G:

- Luego eliminamos un nodo con grado mínimo actualizando las variables necesarias en tiempo constante. Para esto hacemos aparecer un nodo con grado mínimo en la lista de nodos por grado. Tenga en cuenta que se agregaran nuevos nodos.

- solo agregamos un nodo cuando eliminamos un borde. Entonces, el número total de nodos agregados al final del algoritmo greedy está en $O(|E|)$. Por lo tanto, el bucle aún mantiene una complejidad lineal.

- Luego actualizamos el grado de todos los vecinos del nodo seleccionado. Actualizamos la lista de nodos por grados agregando los vecinos a las listas de sus grados-1 que es su nuevo grado. Tenga en cuenta que luego aparecerán varias veces, pero debido a que solo consideramos nodos con grado mínimo, no cambia el resultado. También actualizamos el grado mínimo y agregamos los bordes que contienen el nodo eliminado a la lista de aristas eliminados.

- Todo dentro del bucle for se ejecuta en tiempo constante. Dado que, en cada iteración del algoritmo greedy, se selecciona un nodo diferente y debido a que la

lista de adyacencia tiene varios nodos en $O(|E|)$, este bucle for mantiene el tiempo lineal.

•Si la densidad de G es mayor que H, H se convierte en G Actualizamos el número de nodos, aristas y la densidad. El resultado final son todas las aristas eliminadas (= todos los bordes iniciales porque eliminamos todo al final) menos los bordes eliminados en el último paso donde se actualiza H.

El truco clave aquí es que no necesitamos guardar H actualizada cuando se encuentra una mayor densidad. Solo necesitamos saber qué aristas se han eliminado, que es lo mismo que recordar la longitud de la arista eliminada. Esta parte final también está en tiempo constante, por lo que el algoritmo greedy se ejecuta en tiempo lineal $O(|E|)$

La representación como matriz de adyacencia es más común en grafos densos, cuando el número de aristas es grande.

Verificar si dos nodos están conectados es más sencillo en una matriz de adyacencia que en una lista de adyacencia.

[https://youtu.be/IP0TOZPtjqM]

https://en.wikipedia.org/wiki/Dense_subgraph

https://es.wikipedia.org/wiki/Densidad_(teor%C3%ADa_de_grafos)

https://www.cs.princeton.edu/~zdvir/apx11slides/charikar-slides.pdf

https://shinerightstudio.com/posts/prove-np-complete-by-restriction/ ->prueba

http://www.cs.ucc.ie/~gprovan/CS4407/2013/HW-NP-complete_2013-SOLUTIONS.pdf -> prueba pag 4

https://www.tau.ac.il/~hassin/dense_02.pdf -> prueba mas completa

https://www.openu.ac.il/lists/mediaserver_documents/academic/cs/TheDensekSubgraphProblem.pdf --> prueba mirar

https://geeksforgeeks.org/prove-that-dense-subgraph-is-np-complete-by-generalisation/ -> prueba

https://sci-hub.hkvisa.net/10.1007/s004530010050

https://www.cs.upc.edu/~mjserna/docencia/grauA/T18/Greedy-approx.pdf aproximación con greedy

https://github.com/anhvung/Densest-Subgraph-Algorithm ---> revisar

https://github.com/giannisnik/k-clique-graphs-dense-subgraphs ---> revisar tmb

https://hal.archives-ouvertes.fr/hal-00874586/document

https://github.com/tsourolampis/Densest-subgraph-peeling-algorithm

https://github.com/btsun/kclistpp

https://github.com/varungohil/Densest-Subgraph-Discovery

//////// CÓDIGOS QUE PUEDEN SERVIR

https://github.com/anhvung/Densest-Subgraph-Algorithm

https://github.com/chenhao-ma/SIGMOD2020DDS

https://github.com/tsourolampis/Densest-subgraph-peeling-algorithm

https://github.com/sawlani/densest-subgraph

/////// extras

https://github.com/aepasto/densest-subgraph

https://github.com/RapidsAtHKUST/RapidMatch

// AQUI EMPIEZA LO DE MASSA ////////////////////////////////////

MASSA PART

[1]Ravi Kannan and V Vinay. 1999. Analyzing the structure of large graphs. Rheinische Friedrich-Wilhelms-Universität Bonn Bonn
[2]Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In International Workshop on Approximation Algorithms for Combinatorial Optimization. Springer, 84–95
[3]Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In International Colloquium on Automata, Languages, and Programming. Springer, 597–608.

https://scholar.google.es/scholar?hl=es&as_sdt=0%2C5&q=Efficient+Algorithms+for+Densest+Subgraph+Discovery+on+Large+Directed+Graphs&btnG=

Chenhao Ma
,
Yixiang Fang
,
Reynold Cheng
,
Laks V.S. Lakshmanan
,
Wenjie Zhang
,
Xuemin Lin
Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V.S. Lakshmanan, Wenjie Zhang, Xuemin Lin

Ma, C., Fang, Y., Cheng, R., Lakshmanan, L. V., Zhang, W., & Lin, X. (2020, June). Efficient algorithms for densest subgraph discovery on large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (pp. 1051-1066).

[25] GT Heineman, G Pollice, and S Selkow. 2008. Network Flow Algorithms.
Algorithms in a Nutshell. (2008).

INTRODUCCIÓN

Kan nan y Vinay [1] fueron los primeros en introducir la noción de densidad y el problema del subgrafo más denso en grafos dirigidos (es decir el problema DDS). Charikar [2] desarrolló un algoritmo exacto para este problema, que completa el costo de tiempo polinomial al resolver O(n) linealmente.
Khuller y Saha [3] propusieron un algoritmo basado en flujo, que también toma el costo de tiempo polinomial. algoritmos exactos. Sin embargo,todos estos algoritmos exactos son computacionalmente costosos para grafos grandes, por lo que los investigadores han recurrido al desarrollo de algoritmos de aproximación eficientes.

Kannan y Vinay propusieron un algoritmo de aproximación O(logn) para calcular el subgrafo más denso. Charikar diseñó un algoritmo de 2 aproximaciones con una complejidad temporal de O(n · (n + m)). Khuller y Saha presentaron un algoritmo de aproximación lineal y afirmaron que logra una aproximación de 2.

Si revisamos el algoritmo exacto de última generación(es decir el algoritmo propuesto por Kan nan y Vinay) [1] y los algoritmos de aproximación (Charikar [2] y Khuller y Saha [3]) para el problema DDS.
Observamos que para los algoritmos de aproximación, tanto los algoritmos (Khuller y Saha [3] y Charikar [2]) lograron un raito de aproximación de 2, pero el primero se ejecuta mucho más rápido que el último. Tenga en cuenta que en este documento, la relación de aproximación se define como la relación de la densidad máxima sobre la densidad del subgrafo devuelto.

EL ALGORITMO EXACTO Y ALGORITMO DE APROXIMACION

El algoritmo exacto de última generación [3] calcula el DDS resolviendo un problema de flujo máximo, que generalmente tiene el paradigma de encontrar los subgrafos más densos en grafos no dirigidos. Denotamos este como algoritmo Exacto. Una red de flujo es un gráfico dirigido donde hay un nodo fuente s, un nodo sumidero t y algunos nodos intermedios; cada borde tiene una capacidad y la cantidad de flujo en un borde no puede exceder la capacidad del borde.
El flujo máximo de una red de flujo es igual a la capacidad de su corte mínimo, $\langle S, T \rangle$, que divide el conjunto de nodos en dos conjuntos disjuntos, S y T, de modo que $s \in S$ y $t \in T$.

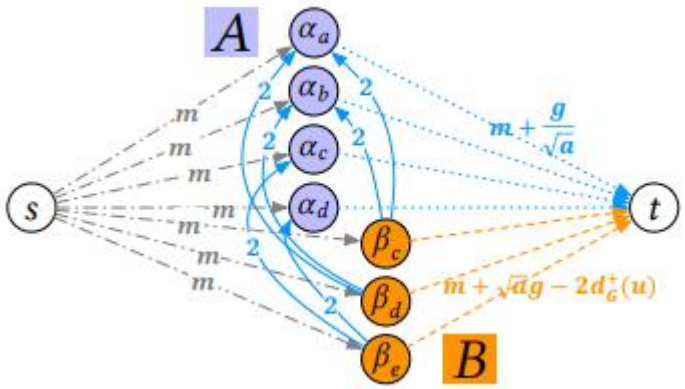

figura.Ilustración de red de flujo

Entonces en nuestro Algoritmo 1:

1. Enumeramos los posibles valores de $a=\frac{|S|}{|T|}$ (línea 2).
2. A Través de la búsqueda binaria adivinamos el valor de g de la densidad máxima por cada a.(líneas 3-5).
3. Para cada par de a y g, se construye una red de flujo y ejecuta el algoritmo de flujo máximo para calcular el corte de st mínimo ⟨S, T ⟩ (líneas 6-11).
4. Si existe un subgrafo inducido <S,T> (es decir $S\backslash\{s\} \neq \emptyset$ tal que su densidad sea al menos g) y que sea mayor a su densidad ρ^*; entonces actualizamos el DDS D y su densidad correspondiente ρ^* (línea 11).
5. Crea un conjunto VF de nodos (líneas 14-15), y luego agrega bordes dirigidos con diferentes capacidades entre estos nodos (líneas 16-20).

Algorithm 1: Exact [32]

Input : $G=(V, E)$
Output : The exact DDS $D=G[S^*, T^*]$

$\rho^* \leftarrow 0$;
foreach $a \in \{\frac{n_1}{n_2} \mid 0 < n_1, n_2 <= n\}$ **do**
  $l \leftarrow 0, r \leftarrow \max_{u \in V}\{d_G^+(u), d_G^-(u)\}$;
  while $r - l \geq \frac{\sqrt{n}-\sqrt{n-1}}{n\sqrt{n-1}}$ **do**
    $g \leftarrow \frac{l+r}{2}$;
    $F=(V_F, E_F) \leftarrow$ BuildFlowNetwok(G, a, g);
    $\langle \mathcal{S}, \mathcal{T} \rangle \leftarrow$ Min-ST-Cut(F);
    if $\mathcal{S}=\{s\}$ **then** $r \leftarrow g$;
    else
      $l \leftarrow g$;
      if $g > \rho^*$ **then** $D \leftarrow G[\mathcal{S} \cap A, \mathcal{S} \cap B]$, $\rho^* = g$;
return D;
Function BuildFlowNetwok$(G = (V, E), a, g)$:
  $A \leftarrow \{\alpha_u | u \in V\}, B \leftarrow \{\beta_u | u \in V\}, E_F \leftarrow \emptyset$;
  $V_F \leftarrow \{s\} \cup A \cup B \cup \{t\}$;
  for $\alpha_u \in A$ **do** add (s, α_u) to E_F with capacity m;
  for $\beta_u \in B$ **do** add (s, β_u) to E_F with capacity m;
  for $\alpha_u \in A$ **do** add (α_u, t) to E_F with capacity $m + \frac{g}{\sqrt{a}}$;
  for $\beta_u \in B$ **do** add (β_u, t) to E_F with capacity $m + \sqrt{a}g - 2d_G^+(u)$;
  for $(u, v) \in E$ **do** add (β_v, α_u) to E_F with capacity 2;
  return $F = (V_F, E_F)$

LIMITACIONES

1. El número de valores posibles de a es n, entonces el bucle while de búsqueda binaria tendrá O(logn) iteraciones. Calcular el corte de st mínimo de una red de flujo requiere un tiempo de O(nm) . En consecuencia, la complejidad de tiempo total es $O(n^3 m \log n)$, que por lo tanto es muy ineficiente incluso en gráficos pequeños. Se demostró que tarda más de 2 días en encontrar el DDS en un gráfico con ~1200 vértices y ~2600 aristas.
2. Esta ineficiencia se puede comprobar con todos los n^2 valores de a, lo cual es muy costoso.
3. La red de flujo F siempre se construye sobre el gráfico completo en cada iteración, mientras que el DDS es a menudo un pequeño subgrafo de G.

4. los límites inicial inferior y superior de ρ ∗ no son muy ajustados. Por lo tanto, hay espacio para mejorar su eficiencia.

HERE

Por otro lado, el algoritmo de aproximación publicado más preciso es BS-Approx [2], que es capaz de encontrar correctamente un resultado de 2 aproximaciones. Describimos sus pasos en el Algoritmo 2. Similar a Exacto, BS-Approx enumera todos los valores posibles de a=|S ||T |(línea 2), y para cada a específico, iterativamente elimina el vértice con el grado mínimo de S o T en función de una condición predefinida (línea 8), y luego actualiza S y T, así como el DDS De aproximado (líneas 4-9).

Algorithm 3: BS-Approx [10]

Input : G=(V, E)
Output: An approximate DDS $\widetilde{D}$

1 $\widetilde{\rho^*} \leftarrow 0, \widetilde{D} \leftarrow \emptyset$;
2 **foreach** $a \in \{\frac{n_1}{n_2} | 0 < n_1, n_2 <= n\}$ **do**
3 $\quad S \leftarrow V, T \leftarrow V$;
4 $\quad$ **while** $S \neq \emptyset \wedge T \neq \emptyset$ **do**
5 $\quad\quad$ **if** $\rho(S,T) > \widetilde{\rho^*}$ **then** $\widetilde{D} \leftarrow G[S,T], \widetilde{\rho^*} \leftarrow \rho(S,T)$;
6 $\quad\quad u \leftarrow \arg\min_{u \in S} d^-_G(u)$;
7 $\quad\quad v \leftarrow \arg\min_{v \in T} d^+_G(v)$;
8 $\quad\quad$ **if** $\sqrt{a} \cdot d^-_G(u) \leq \frac{1}{\sqrt{a}} \cdot d^+_G(v)$ **then** $S \leftarrow S \setminus \{u\}$;
9 $\quad\quad$ **else** $T \leftarrow T \setminus \{v\}$;
10 **return** $\widetilde{D}$;

LIMITACIONES

La complejidad temporal de BS-Approx es $O(n^2 \cdot (n + m))$, donde la sobrecarga principal proviene del ciclo de enumeración de todos los n^2 valores de a. Aunque es mucho más rápido que Exact, sigue siendo ineficiente para gráficos grandes. En un gráfico con unos 3000 vértices y 30 000 aristas, se tarda unos 3 días en calcular el DDS. Por lo tanto, es imperativo desarrollar algoritmos de aproximación más eficientes.