https://iq.opengenus.org/approximate-algorithms-for-np-problems/

https://www.designofapproxalgs.com/book.pdf

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1184&rep=rep1&type=pdf

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.

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 Δ se define formalmente como:

$$\Delta = \frac{m}{n(n-1)/2} = \frac{2m}{n(n-1)}$$

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:

$$\Delta = \frac{m}{n(n-1)}$$

En este caso, la densidad alcanza valor 1para 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:

$$\Delta = \frac{\sum_{i=1}^{m} w_i}{n(n-1)}$$

Problema del grafo denso en las ciencias de la computación:

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 $S = (E_S, V_S)$ ser un subgrafo de G. Entonces la densidad de S se define como $d(S) = \frac{|E_S|}{|V_S|}$.

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 $O(nm\log(n^2/m))$ 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 $\mathrm{NP} \not\subseteq \bigcap_{\varepsilon>0} \mathrm{BPTIME}(2^{n^\varepsilon})$, 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:

$$I_S(v) = \Delta - g^+(v) = \frac{m}{n(n-1)} - |\{u \in V | (v,u) \in E\}|$$

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.

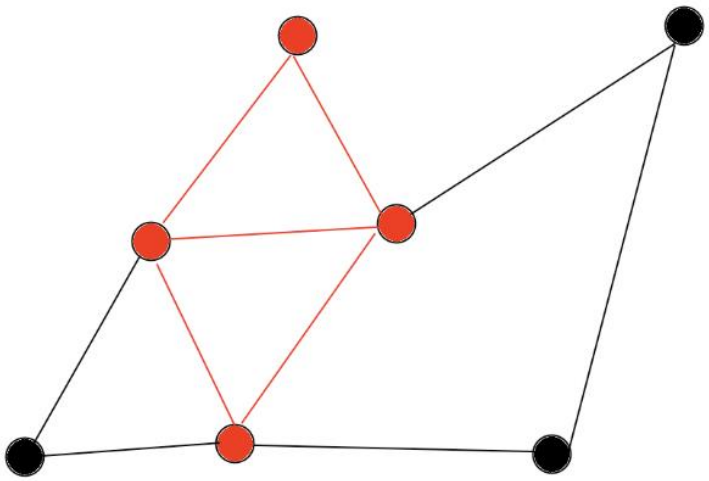
3. Corrección: Se restringó el rango del valor de entrada b tal que (k¦2) con valor como (k * (k – 1))/2.

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

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.

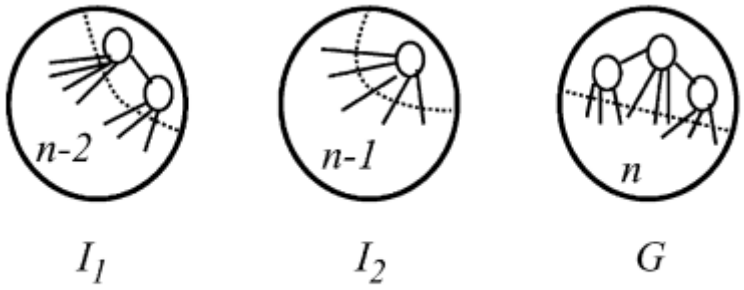


Fig. 1. Proof of Lemma 4.1.

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

Algorithm 1: 2-approximation greedy algorithm

```
Result: H
H = G;
while G contains at least one edge do
    Let v be the node with minimum degree δ_G(v) in G;
    Remove v and all its edges from G;;
    if ρ(G) > ρ(H) then
        H ← G;
    else
end
```

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.

```
while True:
   node = L_degrees[d_min].pop()#remove node with min degree
   while  d_min<d_max and not L_degrees[d_min] : #update min degree
        d_min+=1
    if binary_removed[node]==0:#if 1, already removed !
        break

removed.append(node) #add node to removed nodes
binary_removed[node]=1   #add node to removed nodes
```

- 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.

```
k=0 #count the number of edges removed to update the density
for neighbour in L_adjacent[node]:
        if not binary_removed[neighbour] :
            k+=1.0
         L_degrees[deg[neighbour]-1].append(neighbour)
            if deg[neighbour]== d_min : d_min-=1
            deg[neighbour]-=1
            e_removed+=[[neighbour,node]]
```

- 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.

```
#update E and V
    V-=1.0
    E-=k
    if V!=0 and float(E/V) > d  :

        #at the end, e_removed[m:] = H
        m=len(e_removed)
        d=E/V
```

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://www.google.com/search?q=approximation+greedy+algorithm&oq=approximation+greedy+algorithm+&aqs=chrome..69i57j69i61.10998j0j1&sourceid=chrome&ie=UTF-8

K CENTER PROBLEM: https://github.com/jesgadiaz/k-center-in-C#1

El problema del vértice k -centro es un problema clásico NP-difícil en ciencias de la computación. Tiene aplicación en ubicación de instalaciones y agrupación. Básicamente, el problema del vertex k-center modela el siguiente problema real:

Dada una ciudad con n instalaciones, encuentra las mejores k instalaciones donde construir estaciones de bomberos. Dado que los bomberos deben atender cualquier emergencia lo más rápido posible, la distancia desde la instalación más lejana hasta su estación de bomberos más cercana debe ser la menor posible. En otras palabras, la posición de las estaciones de bomberos debe ser tal que todo posible incendio sea atendido lo más rápido posible.

El problema del vertex k-center Fue propuesto por primera vez por Hakimi en 1964. [3] Formalmente, el problema del k -centro del vértice consiste en: dado un grafo no dirigido completo G = (V, E) en un espacio métrico y un entero positivo k, encontrar un subconjunto C c(rayita abajo) V tal que |C|<=K y la función objetivo r(C) = $\max_{v \in V}\{d(v, C)\}$ e minimiza. La distancia d(v,C). se define como la distancia desde el vértice v a su encuentro cercano en C.