Aplicație Analiza Rețelei Facebook
Facebook Network Analysis

- **Graf Neorientat, no weights**

1.1 Vizualizare graf

- random_layout vs. spring_layout

1.2 Atribute topologice de bază

- nr.noduri (number_of_nodes), nr.total muchii (number_of_edges), gradul mediu nod (np.mean)
- ShortestPath nx.diameter, nx.average_shortest_path_length, nx.all_pairs_shortest_path_length(G), shortest_path_lengths[u][v] nx.excentricity, np.mean np.zeros, np.unique
- Densitate graf (nx.density(G))
- Număr componente graf nx.number_connected_components(G))

1.3. Centrality measures - Măsuri de centralitate

1. Degree Centrality, DC, Grad de centralitate
   nx.centrality.degree_centrality(G), degree_centrality.values()
2. Betweenness Centrality, BC, Centralitatea interioară (bridges)
   nx.centrality.betweenness_centrality(G), betweenness_centrality.values()
3. Clossenes Centrality, CC, Apropierea de Centralitate
   nx.centrality.closeness_centrality(G), closeness_centrality.values()
4. Eigenvector Centrality, EC Centralitatea vectorului propriu
   nx.centrality.eigenvector_centrality(G), eigenvector_centrality.items(), eigenvector_centrality.values()

1.4. Clustering Effects, CE

-CE Coeficient de grupare/clustering (și mediu)
nx.clustering(G).values(), nx.average_clustering(G)
- Inchidere triadică (nr. triunghiuri)
nx.triangles(G).values()
np.mean(triangles_per_node)
np.median(triangles_per_node)

1.5.Bridges inclusiv **local bridges**

1.6.Assortativity (Asortare, similaritate)

1.7.Network Communities

- semi-synchronous label propagation
- asynchronous fluid communities

Analiza Rețelei Facebook

Aplicația conține o analiză a rețelelor sociale rulată cu biblioteca NetworkX.

- Liste de prieteni facebook de 10 persoane vor fi examinate și examinate pentru a extrage diverse informații valoroase.
- Setul de date este la acest link: Stanford Facebook Dataset.
- O rețea facebook este **Neorientată** și **nu are greutate (no weights),** deoarece un utilizator poate deveni prieten cu un alt utilizator o singură dată.
- Din perspectiva analizei grafurilor: fiecare nod reprezintă un utilizator Facebook anonimizat care aparține uneia dintre cele 10 liste de prieteni.
- Fiecare muchie corespunde prieteniei 2 utilizatori Facebook ce aparțin acestei rețele.
- 2 utilizatori trebuie să devină prieteni Facebook pentru a fi conectați în rețea.

Spotlight nodes 0, 107, 348, 414, 686, 698, 1684, 1912, 3437, 3980

sunt nodurile a căror listă de prieteni va fi examinată.

- Aceste noduri sunt în centrul atenției acestei analize și sunt numite **spotlight nodes**.

Analiza Rețelei Facebook - Date și pachete

Import pachete

(Python libraries: pandas, numpy, pyplot)
(Librării grafice matplotlib, pyplot)

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from random import randint

%matplotlib inline
```

Analiză

Muchiile se încarcă din folderul de date, **data**, și salvate în dataframe.

- Cu ***read_csv*** se citesc datele **(comma separated values, csv)**

dar aici separatorul aici este specificat un **Spațiu (space) sep=' '**

- Datele sunt inițial comprimate **compression=gzip**
- Fiecare muchie (**Edge**) este un rând nou și pentru fiecare muchie există o coloană **start_node** și o coloană **end_node.**

```
facebook = pd.read_csv(
    "data/facebook_combined.txt.gz",
    compression="gzip",
    sep=" ",
    names=["start_node", "end_node"],
)
facebook
```

88234 rows × 2 columns

	start_node	end_node
0	0	1
1	0	2
2	0	3
3	0	4
4	0	5
...	...	...
88229	4026	4030
88230	4027	4031
88231	4027	4032
88232	4027	4038
88233	4031	4038

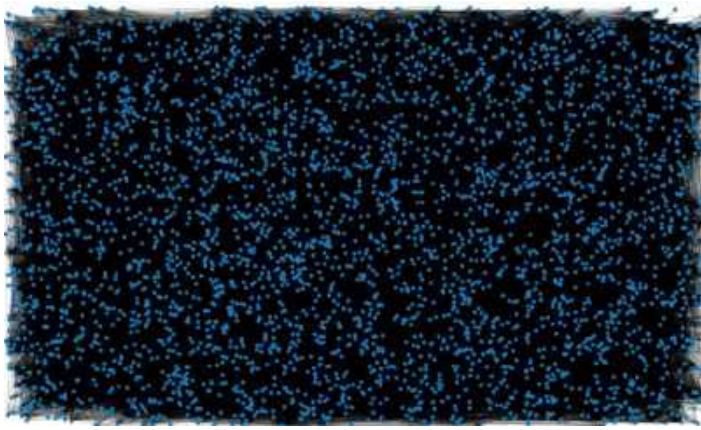
Graful este creat din dataframework **facebook** al muchiilor:

```
G = nx.from_pandas_edgelist(facebook, "start_node", "end_node")
```

Analiza Rețelei Facebook – 1.1. Vizualizare graf

• Se începe explorarea cu vizualizarea grafului.
• Vizualizarea are un rol central în analiza exploratorie a datelor pentru a ajuta la obținerea de **date calitative**.

1. Vizualizare random_layout



Pentru vizualizare graf se utilizează **random_layout,**
o funcție rapidă de aspect.

• Imaginea rezultată nu este foarte utilă analizei.

Vizualizările grafului sunt numite „hairballs”
din cauza **muchiilor (edges) suprapuse**

• **figsize**=dimensiune cadru de afișare figură.
• **ax.axis(off**)= nu afișează axele X, Y ale figurii
• **plot.options**=opțiuni desen, nod dimensiune 10, fără etichetare noduri (False),spațiu 0.15
• **nx.draw_networkx**=G graf de lucru, poziționarea funcția random, axele=off, **opțiuni plotare

```
fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
plot_options = {"node_size": 10, "with_labels": False, "width": 0.15}
nx.draw_networkx(G, pos=nx.random_layout(G), ax=ax, **plot_options)
```

Concluzia este obligația de **îmbunătățire a structurii** asupra poziționării, vizualizării, pentru o **analiză corectă a datelor**.

Analiza Rețelei Facebook 1.1 Vizualizare graf

2. Vizualizare spring_layout

• Pentru îmbunătățire vizualizare se utlizează funcția **spring_layout**, funcția aspect implicită pentru modulul de desen NetworkX.

• **Avantaj spring_layout =** se consideră și noduri și muchii în calculul locațiilor nodurilor.

• **Dezavantaj =** proces **f. costisitor computațional**; lent pt. grafuri mari (ex.100 noduri, 1000 muchii)

•**Facebook** are peste **80.000** muchii, **limităm nr. iterații** (**iterations=15**) pt. reducere timp calcul.

•**seeds= sa include links la spring layout si la toate direct pe networkx**

•Salvează aspectul calculat (**pos**) pentru vizualizări ulterioare.

```
pos = nx.spring_layout(G, iterations=15, seed=1721)
fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
nx.draw_networkx(G, pos=pos, ax=ax, **plot_options)
```

Această vizualizare este mult mai utilă decât random_layout.

Avantaje

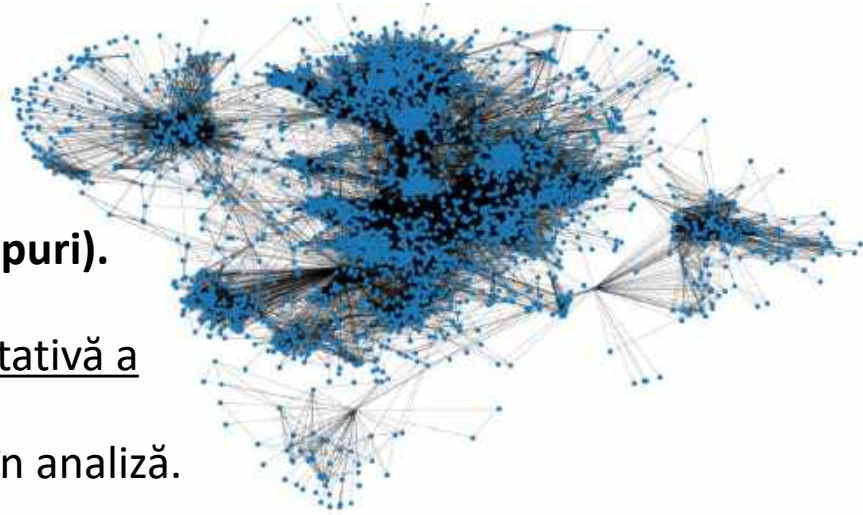
- Deja se poate clarifica parțial structura rețelei;

Ex. multe noduri par a fi foarte conectate,
așa cum ne-am aștepta la o rețea socială.

• Se observă că nodurile tind să formeze **clustere (grupuri).**

• **spring_layout** oferă un sens calitativ al grupării,
dar nu este conceput pentru o analiză repetabilă, calitativă a grupării (**dezavantaj**).

• Se va revizui evaluarea grupării rețelei mai departe în analiză.



Analiza Rețelei Facebook 1.2.Basic topological attributes Atribute topologice de bază

Numărul total de noduri în rețea, **number_of_nodes** (4039)

Numărul total de muchii, **number_of_edges** (88234)

Total number of nodes in network:

```
G.number_of_nodes()
```

```
4039
```

1. Mean Degree (MD) Gradul mediu al unui nod, **np.mean**.

- În medie, un nod este conectat cu **aprox.44 noduri vecini** (43.69…)

Deci, utilizatorul Facebook are in medie 44 prieteni.

Calcul: creare liste cu toate (**for**) gradele **(d degree)** nodurilor

[d for _, d in G.degree] _ este orice indice/element

numpy.array (np) calculează media listei create (**np.mean**)

```
np.mean([d for _, d in G.degree()])
```

```
43.69101262688784
```

Total number of edges:

```
G.number_of_edges()
```

```
88234
```

2. ShortestPath

Proprietăți interesante legate de distribuția căilor prin graf:

- **Diametrul grafului** =cea mai lungă dintre cele mai scurte căi ce conectează orice nod la alt nod din graf (funcția **nx.diameter**)
- **Lungimea medie** a căii oferă o măsură a nr. mediu de muchii ce trebuie parcurse pentru a ajunge de la un nod la altul din rețea. (funcția **nx.average_shortest_path_length)**

Aceste analize necesită calcularea celei mai scurte căi între fiecare pereche de noduri din rețea: acest lucru poate fi destul de costisitor pentru rețele de dimensiune mare.

Analiza Rețelei Facebook 1.2.Basic topological attributes - ShortestPath

Shortestpath [all_pairs_shortest_path_length]

- **Motivația:** multe analize implică cel mai scurt drum pentru toate nodurile rețelei
- Se poate calcula o dată, se salvează într-un 'dicționar' (**dict**), se reutilizează pentru economisire timp de calcul
- **Calcul** cel mai scurt drum pentru toate perechile de noduri din rețea:

```
shortest_path_lengths = dict(nx.all_pairs_shortest_path_length(G))
```

nx.all_pairs_shortest_path_length returnează dict-of-dict
(a dictionary with string keys and dict values; a dictionary of dictionaries)
ce mapează un nod ***u*** la toate celelalte noduri din rețea,
maparea cea mai interioară returnează lungimea celei mai scurte căi dintre două noduri.

shortest_path_lengths[u][v] returnează cel mai scurt drum între oricare 2 noduri (***u,v)***:

```
shortest_path_lengths[0][42]  # Length of shortest path between nodes 0 and 42
```

```
1
```

Analiza Rețelei Facebook 1.2.Basic topological attributes - ShortestPath

Shortestpath [all_pairs_shortest_path_length]

• Se utilizează **shortest_path_lengths** pentru analiză, începând cu **diametrul *G***.

• Documentația arată că ***nx.diameter*** este echivalent cu excentricitatea maximă a grafului (maximum *eccentricity* of the graph)

• ***nx.excentricity*** are argument opțional ***sp*** ce include ***shortest_path_lengths*** precalculate pentru a reduce calculul suplimentar:

Echivalent cu ***`diameter = nx.diameter(G),*** eficient - reutilizează cele mai scurte drumuri precalculate

```
diameter = max(nx.eccentricity(G, sp=shortest_path_lengths).values())
diameter
```

Rezultat= 8 = pentru conectare de la un nod la oricare altul, se traversează 8 muchii sau mai puține.

Calcul lungimea medie a drumului (np.mean)

În loc de ***nx.average_shortest_path_length,*** este eficient ***shortest_path_length*** deja calculat:

Calculați lungimea medie a drumului cel mai scurt **(average shortest path length**) pentru fiecare nod **np.mean**
Media pentru toate nodurile **np.mean**

```
average_path_lengths = [
    np.mean(list(spl.values())) for spl in shortest_path_lengths.values()
]
# The average over all nodes
np.mean(average_path_lengths)
```

```
3.691592636562027
```

Rezultat=3.6= media lungimii celui mai scurt drum pentru toate perechile de noduri: pentru a ajunge de la un nod la altul, vor fi parcurse în medie aproximativ 3,6 muchii.

Analiza Rețelei Facebook 1.2.Basic topological attributes - ShortestPath

Shortestpath [all_pairs_shortest_path_length]

• Cu măsurile prezentate se obțin informații utile despre rețea, metrici ex.valoarea medie reprezintă doar **un moment al distribuției**; dar este util **să privim distribuția în sine**.

Distribuția în sine - Construim o **nouă vizualizare** a distribuției celor mai scurte drumuri (shortest path lengths) precalculate din dict-of-dicts.

Cunoscând lungimea maximă a drumului cel mai scurt (**diametrul),** se crează o matrice (**path_lengths**) de stocare valori de la 0 până la (și inclusiv) diametru

```
path_lengths = np.zeros(diameter + 1, dtype=int)
```

Se **extrage frecvența** (cnts) **celor mai scurte lungimi de drum** între două noduri

```
for pls in shortest_path_lengths.values():
    pl, cnts = np.unique(list(pls.values()), return_counts=True)
    path_lengths[pl] += cnts
```

Se exprimă **distribuția frecvenței** (freq_percent) ca procent % (ignoră lungimile drum valoare 0)

```
freq_percent = 100 * path_lengths[1:] / path_lengths[1:].sum()
```

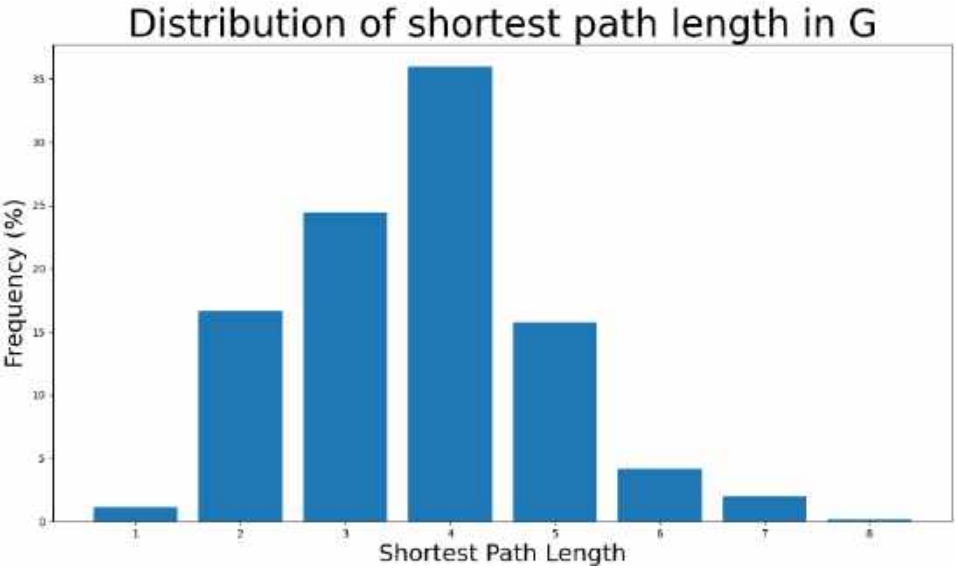
Utilizare funcția **Plot** în reprezentarea distribuției frecvenței (ignoră lungimile nule 0) ca procent %

```
fig, ax = plt.subplots(figsize=(15, 8))
ax.bar(np.arange(1, diameter + 1), height=freq_percent)
ax.set_title(
    "Distribution of shortest path length in G", fontdict={"size": 35}, loc="center"
)
ax.set_xlabel("Shortest Path Length", fontdict={"size": 22})
ax.set_ylabel("Frequency (%)", fontdict={"size": 22})
```

```
Text(0, 0.5, 'Frequency (%)')
```

Analiza Rețelei Facebook 1.2.Basic topological attributes. Densitate, Număr Componente

Basic topological attributes-Atribute topologice de bază



Rezultat Shortestpath

Majoritatea celor mai scurte drumuri sunt de la **2** la **5** lungimea muchiilor
(axa OX grafic cu Shortest Path Length)

Este f. puțin probabil ca o pereche de noduri să aibă o cale cea mai scurtă de lungime 8 (lungimea diametrului), deoarece probabilitatea este mai mică de 0.1%.

CALCUL Densitate graf cu funcția **nx.density(G)**

Rezultat densitate = 0,0108199635 **graful** este **foarte rar (very sparse) densitate < 1**

CALCUL Număr componente graf funcția: **nx.number_connected_components(G**)

Rezultat nr. componente = **1**

După cum era de așteptat, rețeaua **facebook** constă din 1 **o componentă** gigantică.

1.3. Centrality measures - 1. Degree Centrality, DC

1. Degree Centrality, DC - Grad de centralitate

DC atribuie un scor de importanță bazat doar pe numărul de legături deținute de fiecare nod.

- În analiză înseamnă: cu cât gradul de centralitate al unui nod este mai mare, cu atât mai multe muchii sunt conectate la nod și, cu atât mai multe noduri vecini (prieteni FB) are acest nod..
- DC al unui nod este fracția de noduri la care este conectat.
- DC este procentul din rețea la care este conectat nodul, „friend with“.

Calculăm nodurile cu cel mai înalt grad de centralitate.

Funcție – **nx.centrality.degree_centrality(G)**

În cod sunt noduri sortate descrescător (reverse=T) cu 8 centralități **[:8]** de cel mai înalt grad DC.

(# salvare rezultate în variabila **degree_centrality**)

```
degree_centrality = nx.centrality.degree_centrality(
    G
)  # save results in a variable to use again
(sorted(degree_centrality.items(), key=lambda item: item[1], reverse=True))[:8]
```

Rezultat – Nodul 107 are **cel mai înalt grad de centralitate,** 0.259, adică utilizatorul FB este prieten cu aprox. **26%** din întreaga rețea.

- Nodurile 1684,1912,3437 și 0 au centralităţi de grad foarte înalt.
- Era de așteptat, deoarece sunt **spotlight nodes** examinate aici.

```
[(107, 0.258791480931154),
 (1684, 0.1961367013372957),
 (1912, 0.18697374938088163),
 (3437, 0.13546310054482416),
 (0, 0.08593363051015354),
 (2543, 0.07280832095096582),
 (2347, 0.07206537890044576),
 (1888, 0.062902426944031 7)]
```

Dar și nodurile 2543, 2347, 1888 au unele din primele 8 centralități de cel mai înalt grad, chiar dacă nu le investigăm direct, nu sunt **spotlight nodes.** Acestea sunt f. **populare** in cercurile examinate acum, au cei mai mulți prieteni pe FB, în afară de spotlight nodes.

1.3. Centrality measures - 1. Degree Centrality, DC

NetworkX Funcția Degree cu apel **G. Degree** pe graful **G**

Acum se văd și **numărul de vecini** pentru **nodurile cu cel mai mare grad de centralități**:

```
(sorted(G.degree, key=lambda item: item[1], reverse=True))[:8]
```

```
[(107, 1045),
 (1684, 792),
 (1912, 755),
 (3437, 547),
 (0, 347),
 (2543, 294),
 (2347, 291),
 (1888, 254)]
```

Rezultat

Cum era de așteptat, nod 107 are DC=1045 prieteni Facebook, care este cel mai mult pe care îl are orice utilizator de Facebook în această analiză.

- Nodurile 1684 și 1912 au mai mult de 750 prieteni Facebook in această rețea, (DC=792, DC=755)

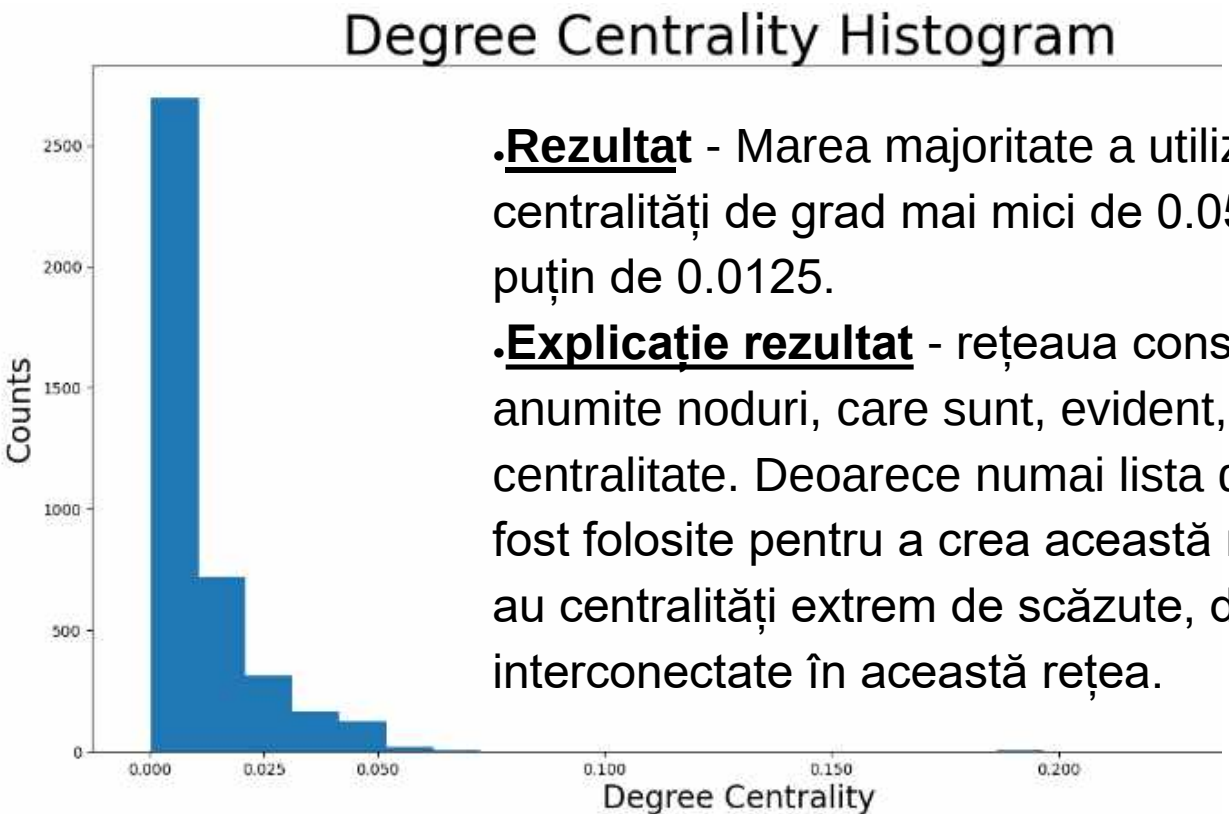
- Nodurile 3437 și 0 au următorul număr cel mai mare de prieteni Facebook în această rețea cu DC=547 și DC=347.

- Cei mai populari doi prieteni ai nodurilor spotlight au în jur de 290 prieteni Facebook in această rețea.

1.3. Centrality measures - 1. Degree Centrality, DC

Reprezentarea grafică – Histograma cu **distribuția centralităților (Degree Centrality values)**

```
plt.figure(figsize=(15, 8))
plt.hist(degree_centrality.values(), bins=25)
plt.xticks(ticks=[0, 0.025, 0.05, 0.1, 0.15, 0.2])  # set the x axis ticks
plt.title("Degree Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Degree Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

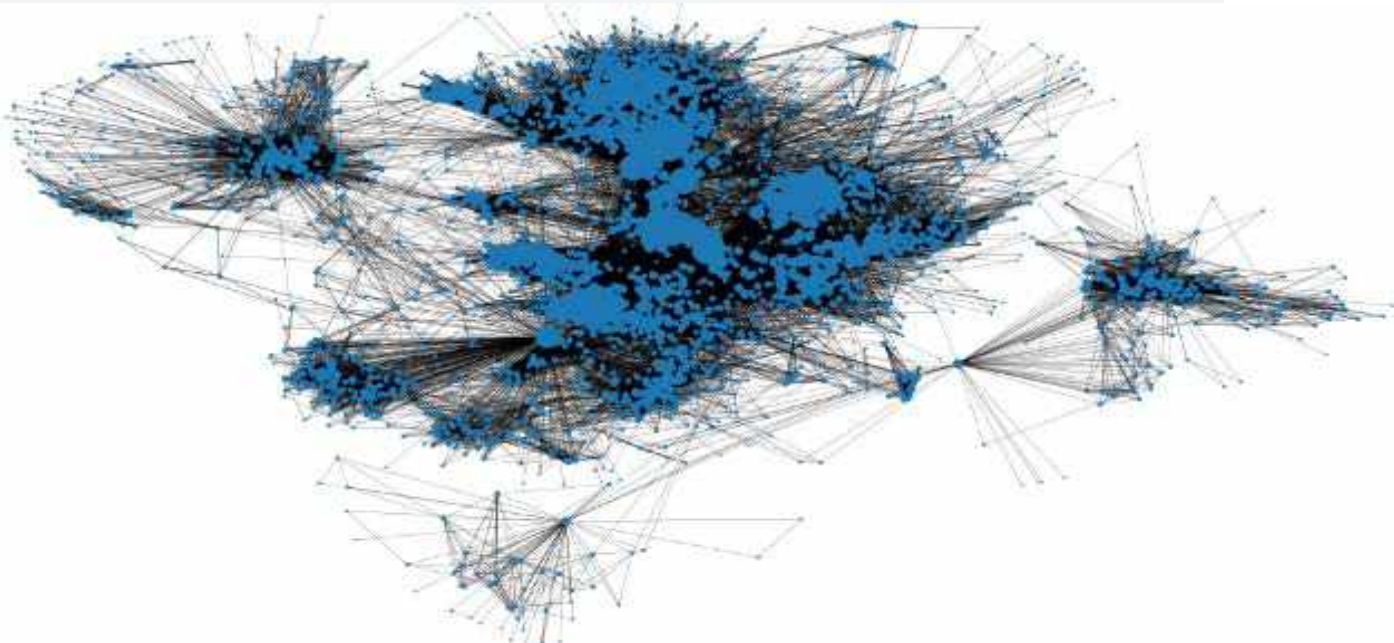


- **Rezultat** - Marea majoritate a utilizatorilor de Facebook au centralități de grad mai mici de 0.05, majoritatea are de fapt mai puțin de 0.0125.
- **Explicație rezultat** - rețeaua constă din liste de prieteni cu anumite noduri, care sunt, evident, cele cu cel mai înalt grad de centralitate. Deoarece numai lista de prieteni a anumitor noduri au fost folosite pentru a crea această rețea particulară, multe noduri au centralități extrem de scăzute, deoarece nu sunt foarte interconectate în această rețea.

1.3. Centrality measures - 1. Degree Centrality, DC

- Verificăm utilizatorii cu centralități de cel mai înalt grad din dimensiunea nodurilor lor:
- Nodurile v sunt inmulțite cu 1000 (v * 1000) pentru **o mai bună vizualizare**
- Se utilizează nodurile **v** din **degree_centrality.values()**
- **Dimensiunea nodului din reprezentarea vizuală** este dată de **valoarea gradului de centralitate (degree_centrality.values())**

```
node_size = [
    v * 1000 for v in degree_centrality.values()
]  # set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```



1.3. Centrality measures - 2. Betweenness Centrality (BC)

Betweenness Centrality (BC) Centralitatea interioară, măsoară de câte ori un nod se află pe calea cea mai scurtă între alte noduri, deci acționează ca o punte (bridge).

- BC a unui nod ***v*** este **procentul %** tuturor celor mai scurte căi dintre oricare două noduri (în afară de ***v***), care trec prin ***v***.

În graful Facebook, BC este asociată cu

- **capacitatea utilizatorului (nodului) de a-i influența pe ceilalți.**
- Un utilizator cu valoare **mare** a **BC** acționează ca o punte (**bridge**)

către mulți utilizatori care nu sunt prieteni,

deci poate influența prin transmiterea de informații

de exemplu, prin postarea sau partajarea unei postări sau conectare prin cercul utilizatorului.

1.3. Centrality measures - 2. Betweenness Centrality (BC)

Calcul BC pt. nodurile cu cele mai mari 8 centralități de întreținere

```
betweenness_centrality = nx.centrality.betweenness_centrality(
    G
)  # save results in a variable to use again
(sorted(betweenness_centrality.items(), key=lambda item: item[1], reverse=True))[:8]
```

Rezultate – nodul **107** are **BC=0.48**, adică se află pe aproape jumătate din totalul celor mai scurte căi între alte noduri. Combinând cunoștințele privind centralitatea gradului:

Nodurile **0, 107, 1684, 3437** au atât cel mai înalt **Degree Centrality (DC**) și **BC** și sunt **spotlight nodes**. (BC=0.146, BC=0.48, BC=0.337, BC=0.236)

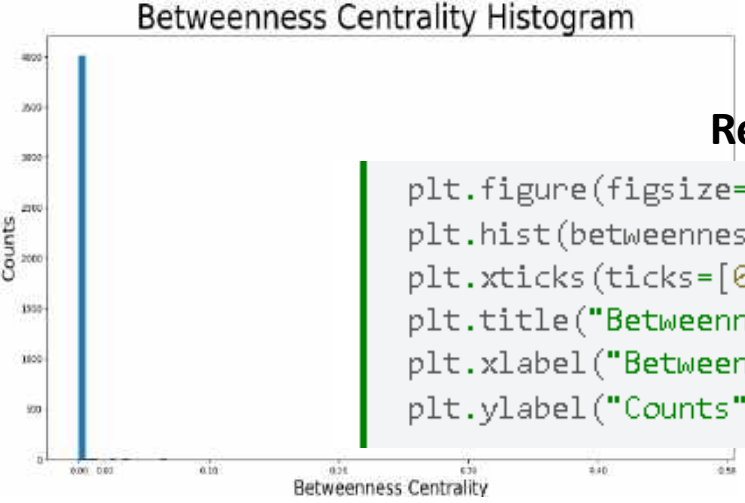
- Deci nodurile sunt cele mai populare și pot influența și răspândi informații în rețea.
- Dar sunt câteva dintre nodurile a căror listă de prieteni este chiar rețeaua cu care lucrăm și, deci este o descoperire așteptată.

. Nodurile **567 (BC=0.096), 1085 (BC=0.15)** nu sunt spotlight, dar au unele dintre cele mai înalte BC dar nu au DC de cel mai înalt grad. Deși nu sunt cei mai populari, au cea mai mare influență în rețea printre prietenii nodurilor spotlight în răspândirea informațiilor.

```
[(107, 0.4805180785560152),
 (1684, 0.3377974497301992),
 (3437, 0.23611535735892905),
 (1912, 0.2292953395868782),
 (1085, 0.14901509211665306),
 (0, 0.14630592147442917),
 (698, 0.11533045020560802),
 (567, 0.09631033121856215)]
```

.Nodul **698** (**BC=0.115)** este un **spotlight node** și are o centralitate f. mare, chiar dacă nu are centralități de cel mai înalt grad. Nu are o listă de prieteni f. mare, dar, întreaga sa listă de prieteni este parte a rețelei și, astfel, utilizatorul poate conecta diferite cercuri din această rețea fiind un **intermediar**.

1.3. Centrality measures – 2. Betweenness Centrality (BC)



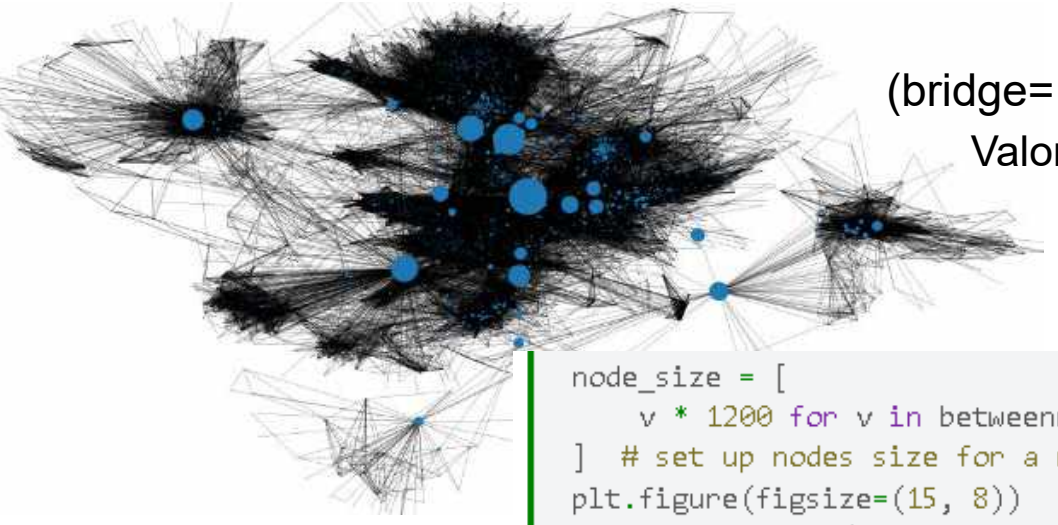
Reprezentarea grafică prin Histogramă a **distribuției BC:**

```
plt.figure(figsize=(15, 8))
plt.hist(betweenness_centrality.values(), bins=100)
plt.xticks(ticks=[0, 0.02, 0.1, 0.2, 0.3, 0.4, 0.5])  # set the x axis ticks
plt.title("Betweenness Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Betweenness Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

Rezultat - Marea majoritate BC < **0.01**, deoarece **graful** este **foarte rar,** deci majoritatea nodurilor nu acționează ca punți (**bridge**) pe ShortestPath.

Dar sunt și **noduri cu centralități extrem de mari** ex.nod 107, BC=0.48 și nod 1684,BC=0.34

- Se obține o imagine asupra nodurilor cu cele mai mari BC și unde sunt situate în rețea.
- Evident acestea sunt punți de legătură (**bridges**) de la o comunitate la alta



(bridge= puncte ce leagă grupurile, ex, dreapta jos)

Valorile sunt inmulțite cu 1200 pt. bună vizualizare funcție=**betweenness_centrality.values()**

```
node_size = [
    v * 1200 for v in betweenness_centrality.values()
]  # set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```

1.3. Centrality measures – 3. Closeness Centrality, CC

3. Clossenes Centrality, CC, Apropierea de Centralitate măsoară fiecare nod pe baza „apropierii” lor de toate celelalte noduri din rețea.

Centralitatea de apropiere a nodului **v** măsoară

distanța medie față de toate celelalte noduri.

Cu cât este **mai mare CC** a nodului ***v,***,

cu atât ***v***, este situat **mai aproape de centrul rețelei.**

Măsura centralității apropierii este foarte importantă pentru monitorizarea răspândirii informațiilor (de exemplu, știri false) sau a virușilor, link-uri rău intenționate ce câștigă controlul contului de Facebook).

Exemplu știri false (fake news)

- Dacă utilizatorul cu **cea mai mare** valoare **CC** ar răspândi informații de știri false (prin partajare sau postări), întreaga rețea ar fi informată greșit cât mai rapid posibil.
- Dacă un utilizator cu **Clossenes Centrality** foarte scăzută ar încerca același lucru, răspândirea informațiilor greșite în întreaga rețea ar fi mult mai lentă.
- Deci, informațiile false dacă ajung mai întâi la un utilizator cu centralitate mare de apropiere (**Clossenes Centrality, CC)**, le răspândește rapid în multe părți diferite ale rețelei.

1.3. Centrality measures – 3. Closeness Centrality, CC

CALCUL – noduri cu **cele mai mari Closness Centrality,CC (centralități de apropiere**)

Funcție **nx.centrality.closeness_centrality(G)**

```
closeness_centrality = nx.centrality.closeness_centrality(
    G
)  # save results in a variable to use again
(sorted(closeness_centrality.items(), key=lambda item: item[1], reverse=True))[:8]
```

```
[(107, 0.45969945355191255),
 (58, 0.3974018305284913),
 (428, 0.3948371956585509),
 (563, 0.3939127889961955),
 (1684, 0.39360561458231796),
 (171, 0.37049270575282134),
 (348, 0.36991572004397216),
 (483, 0.3698479575013739)]
```

Rezultat - Inspectând utilizatorii cu cele mai mari valori CC, centralități de apropiere, înțelegem că nu există un decalaj uriaș între ei, în contrast cu metricile anterioare.

Nodurile 107, 1684, 348 sunt singurele **spotlight nodes** găsite în cele cu cele mai mari centralități de apropiere.

Deci **<u>un nod cu mulți prieteni nu este necesar aproape de centrul rețelei</u>**.

Distanța medie (average distance) a unui anumit nod **v** la orice alt nod poate fi găsit cu formula:

$$\frac{1}{closeness\,centrality(v)}$$

```
1 / closeness_centrality[107]
```

```
2.1753343239227343
```

Deci distanța medie de la nodul ***107*** la un nod aleator este 2.175... aproximativ de 2 hops.

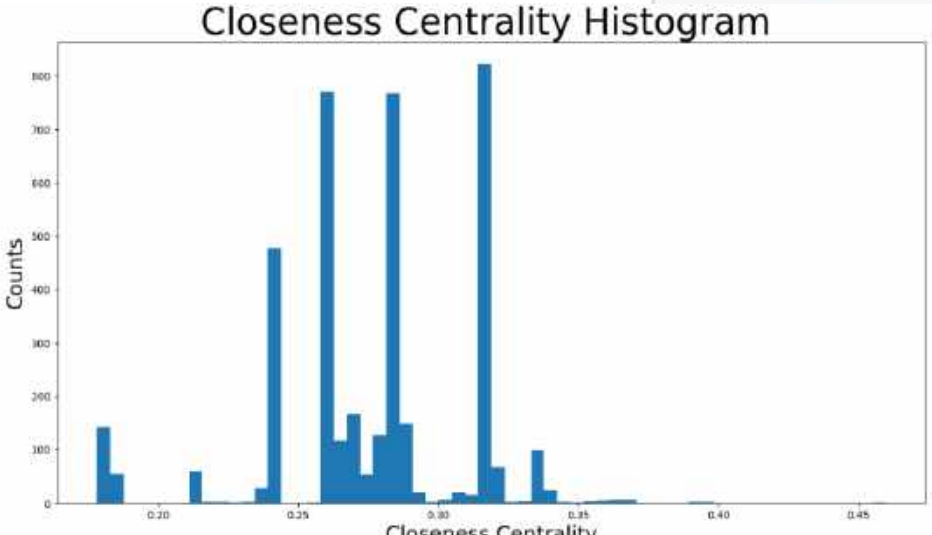
1.3. Centrality measures – 3. Closeness Centrality, CC

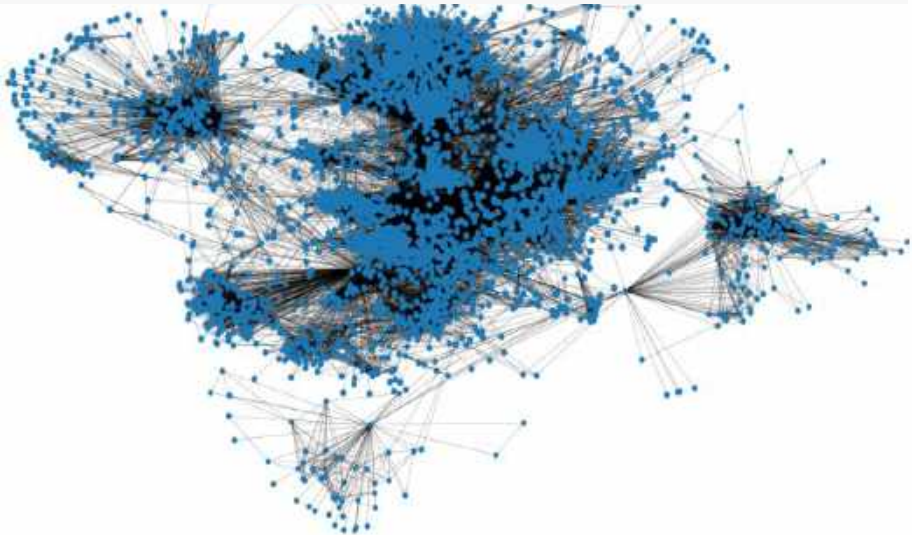
Reprezentare Histograma Closeness Centrality, CC, distribuția centralităților de apropiere:

```
plt.figure(figsize=(15, 8))
plt.hist(closeness_centrality.values(), bins=60)
plt.title("Closeness Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Closeness Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

- Centralitățile de apropiere sunt distribuite pe diverse valori de la 0.17 la 0.46
- Majoritatea nodurilor sunt **relativ aproape de centrul rețelei (CC** între **0.25** și **0.3**) și aproape de alte noduri în general.
- Dar există și comunități situate mai departe, cu noduri ce ar avea centralități minime de apropiere

```
node_size = [
    v * 50 for v in closeness_centrality.values()
]  # set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```





1.3. Centrality measures – 4. Eigenvector Centrality, EC

4. Eigenvector Centrality, EC, Centralitatea vectorului propriu

EC, Centralitatea vectorului propriu este metrica ce arată cât de conectat este un nod la alte noduri importante din rețea.

EC măsoară influența unui nod în funcție de cât de bine este conectat în rețea și de câte legături au conexiunile sale și așa mai departe.

EC identifică nodurile cu cea mai mare influență asupra întregii rețele.

Nod cu centralitate **EC mare/înaltă** a vectorului propriu înseamnă că nodul este conectat la alte noduri care au ei înșiși centralități ridicate cu vectori proprii.

În **analiza Facebook**, **măsura EC este asociată cu capacitatea utilizatorilor de a influența întregul graf (întreaga rețea FB) și astfel utilizatorii cu cele mai mari EC centralități de vector propriu sunt cele mai importante noduri din această rețea.**

1.3. Centrality measures – 4. Eigenvector Centrality, EC

Nodurile cu **cele mai mari centralități EC** de vector propriu vor fi examinate.

[:10] deci 10 cele mai mari valori/măsuri EC, funcția **nx.centrality.eigenvector_centrality(G)**

```
eigenvector_centrality = nx.centrality.eigenvector_centrality(
    G
)  # save results in a variable to use again
(sorted(eigenvector_centrality.items(), key=lambda item: item[1], reverse=True))[:10]
```

Rezultate:

- Nodul **1912 are cea mai mare centralitate de vector propriu, EC=0.095**
- 1912 este și **spotlight node** și poate fi considerat cel mai important nod din această rețea, cu influență generală asupra întregii rețele.

Nodul 1912 are și DC și BC de grad înalt, făcând utilizatorul foarte popular și influent pt. alte noduri.

```
[(1912, 0.09540696149067629),
 (2266, 0.08698327767886552),
 (2206, 0.08605239270584342),
 (2233, 0.08517340912756598),
 (2464, 0.08427877475676092),
 (2142, 0.08419311897991795),
 (2218, 0.0841557356805503),
 (2078, 0.08413617041724977),
 (2123, 0.08367141238206224),
 (1993, 0.0835324284081597)]
```

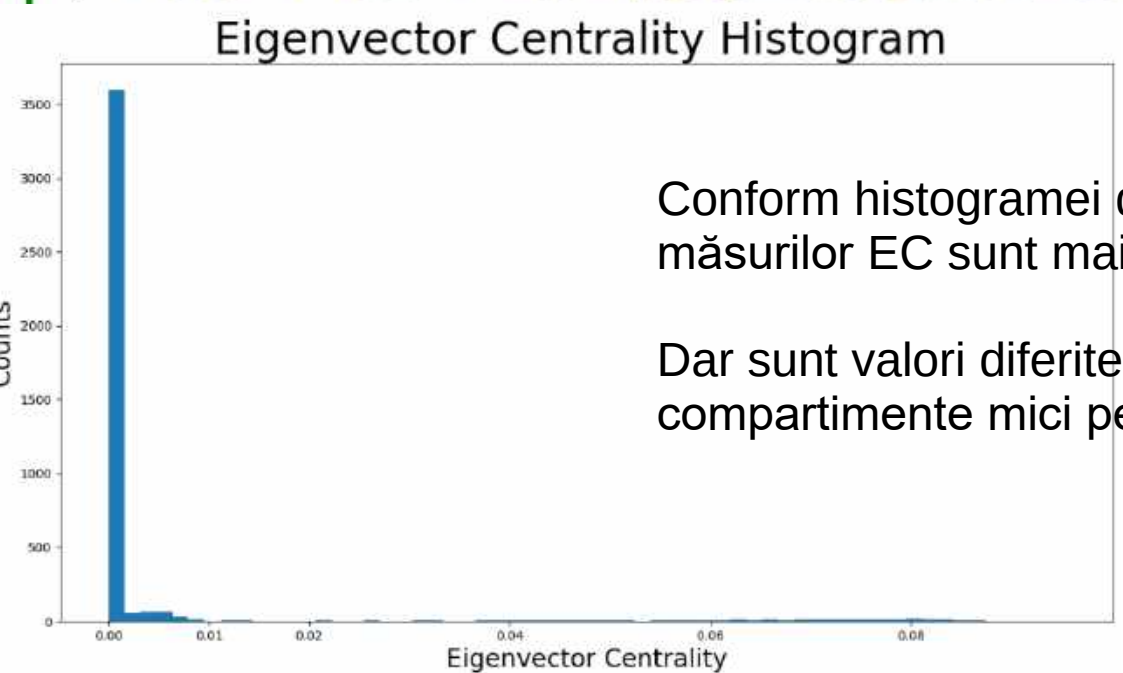
- Nodurile 1993, 2078, 2206, 2123, 2142, 2218, 2233, 2266, 2464 chiar dacă nu sunt noduri spotlight, au unele dintre cele mai înalte centralități de vector propriu (EC $0.83 - 0.87$)
- Toate acele noduri sunt identificate pentru prima dată, adică **nu** au cel mai înalt grad DC, BC, CC, centralități în graf.
- **O concluzie posibilă (ipoteză)** ar fi că aceste noduri sunt **foarte conectate cu nodul 1912** si astfel au scor de centralitate EV foarte mare.

1.3. Centrality measures – 4. Eigenvector Centrality, EC

Verificare Ipoteză dacă nodurile 1993, 2078, 2206, 2123, 2142, 2218, 2233, 2266, 2464 sunt conectate la cel mai important nod 1912 (**neighbors_1912**)

Funcții **eigenvector_centrality.items()** Rezultatul este **True,** deci ipoteza este corectă.

```
high_eigenvector_centralities = (
    sorted(eigenvector_centrality.items(), key=lambda item: item[1], reverse=True)
)[
    1:10
]  # 2nd to 10th nodes with heighest eigenvector centralities
high_eigenvector_nodes = [
    tuple[0] for tuple in high_eigenvector_centralities
]  # set list as [2266, 2206, 2233, 2464, 2142, 2218, 2078, 2123, 1993]
neighbors_1912 = [n for n in G.neighbors(1912)]  # list with all nodes connected to 1912
all(
    item in neighbors_1912 for item in high_eigenvector_nodes
)  # check if items in list high_eigenvector_nodes exist in list neighbors_1912
```



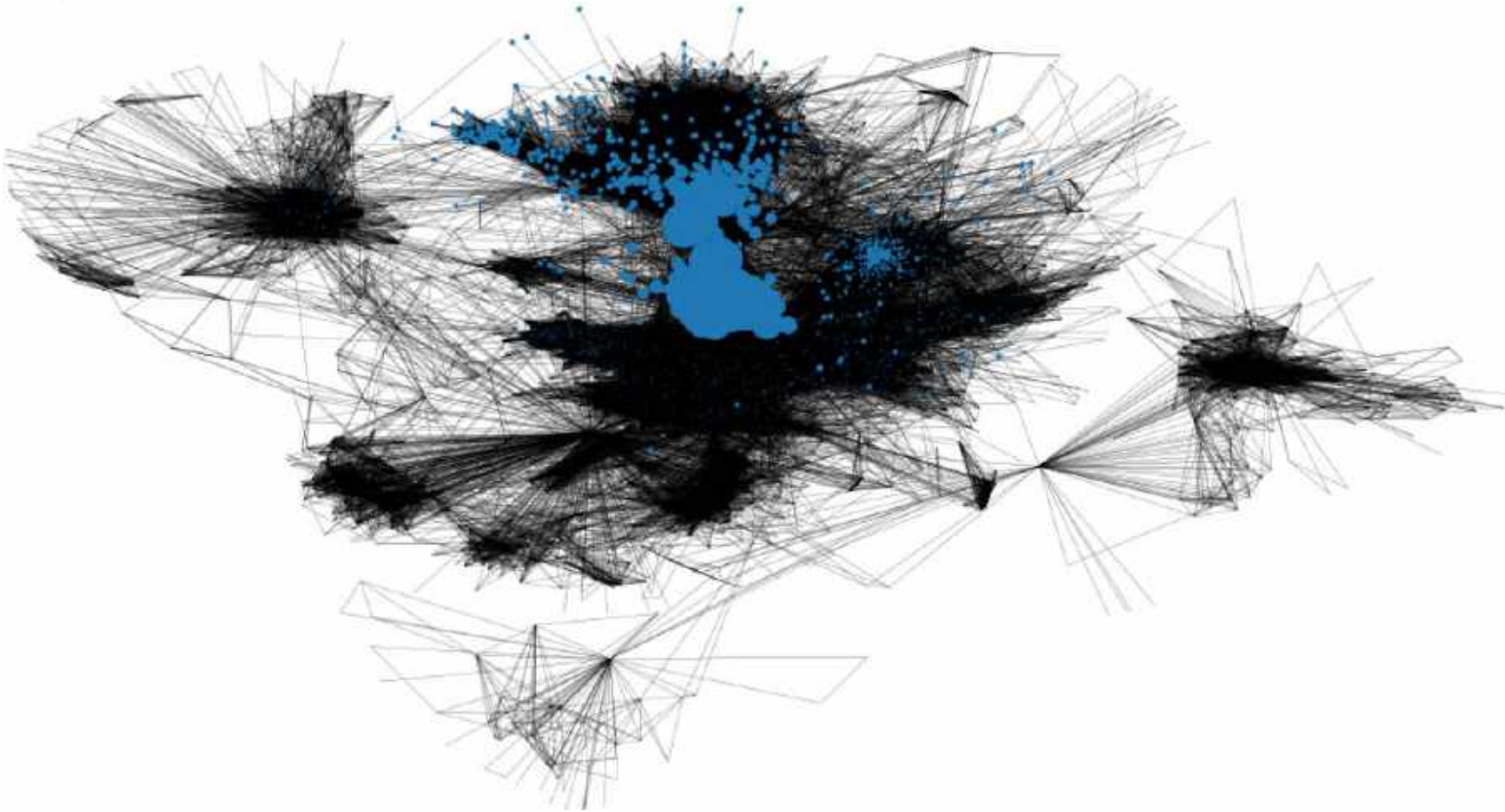
Conform histogramei de distribuției marea majoritate a măsurilor EC sunt mai jos/mici de 0.005 și aproape de 0.

Dar sunt valori diferite ale EC, deoarece există compartimente mici pe toată axa x.

1.3. Centrality measures – 4. Eigenvector Centrality, EC

Se pot identifica EC ale nodurilor pe baza dimensiunii lor în următoarea reprezentare: (**eigenvector_centrality.values()**)

```
node_size = [
    v * 4000 for v in eigenvector_centrality.values()
]  # set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```



1.4. Clustering Effects, CE - Coeficientul de grupare al unui nod

CE, Coeficientul de grupare al unui nod v este definit ca probabilitatea ca 2 prieteni aleși aleatoriu ai nodului ***v*** sunt prieteni unul cu celălalt.

Coeficientul mediu de clustering este media coeficienților de clustering ai tuturor nodurilor.

Funcția NetworkX aplicată grafului G **nx.average_clustering(G)**

. Cu cât coeficientul mediu de grupare este **mai aproape** de valoarea **1**, cu atât graful va fi **mai complet**, deoarece există o singură componentă gigantică în rețeaua Facebook..

Deci este un semn de **închidere triadică**, deoarece cu cât graful este mai complet, cu atât vor apărea de obicei mai multe triunghiuri.

```
nx.average_clustering(G)
```

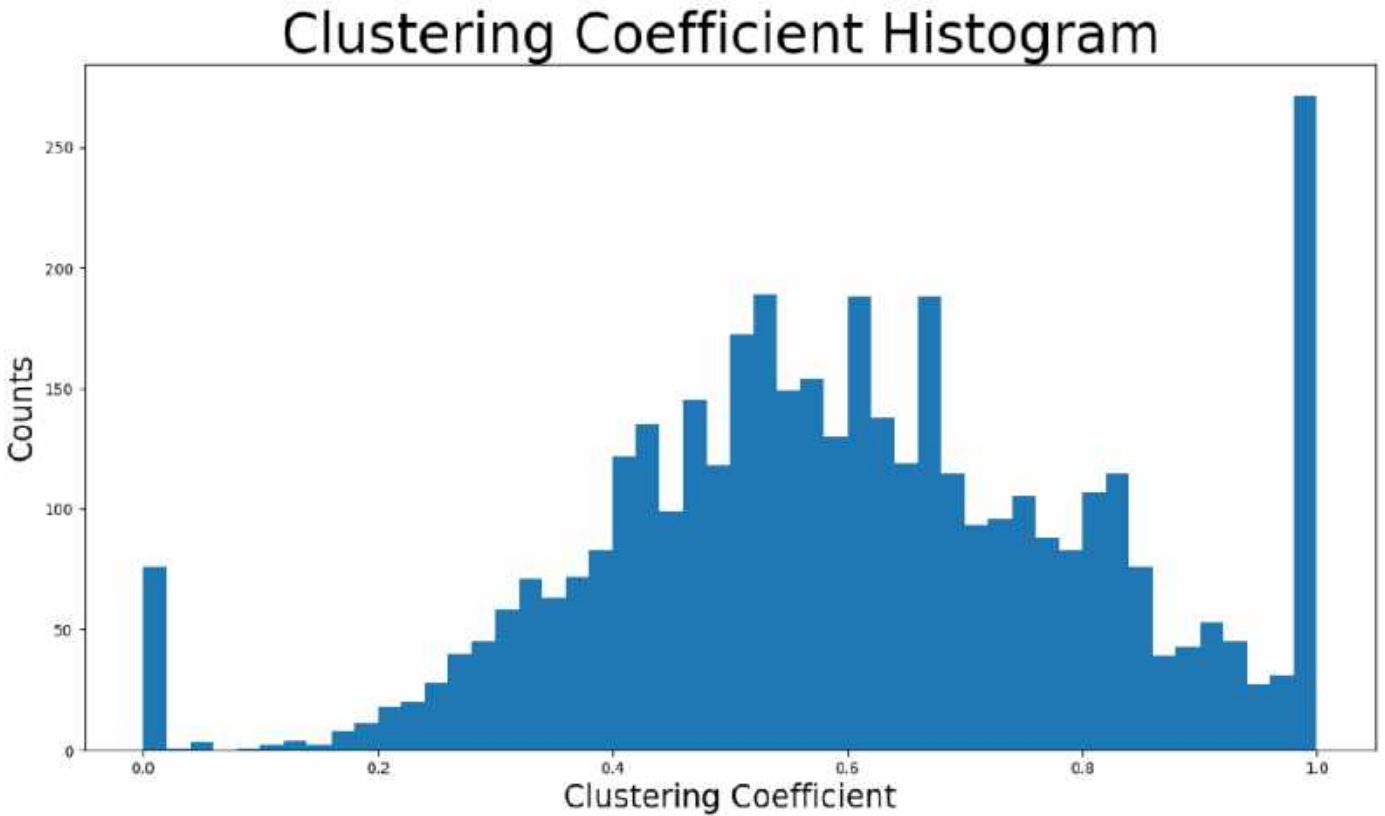
Rezultatul este 0.6055467186200862, aproape de valoarea 1.

Analiza Rețelei Facebook 1.4. Clustering Effects, CE

Reprezentarea Histogramei cu distribuția CE, Clustering Effects a coeficienților de grupare:
Se folosesc **50 bins/grupări (metodă supervizată, specifică nr. grupări) ,** funcția **nx.clustering(G).values()**

```
plt.figure(figsize=(15, 8))
plt.hist(nx.clustering(G).values(), bins=50)
plt.title("Clustering Coefficient Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Clustering Coefficient", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

- **Binul (grupul) cu cele mai mari valori CE** se referă la noduri cu coeficientul de grupare aproape de valoarea **1** deoarece există mai mult de **250 de noduri în acel bin (grup)**.
- Bins/grupuri cu coeficient de clustering **CE între 0.4 și 0.8** conțin majoritatea nodurilor.



1.4. Clustering Effects – Număr Triunghiuri

Calcul număr de **triunghiuri unice** din rețea. Rezultat = 1612010

```
triangles_per_node = list(nx.triangles(G).values())
sum(
    triangles_per_node
) / 3  # divide by 3 because each triangle is counted once for each node
```

Funcție **nx.triangles(G).values()**
se împarte la 3 pt. ca fiecare triunghi să fie contorizat o singură dată pt. fiecare nod

Numărul mediu de triunghiuri din care face parte un nod:

- **np.mean(triangles_per_node)** **Rezultat** **1197**.3334983906907

Se aplică metrica **Median**, deoarce unele noduri aparțin unui număr mare de triunghiuri

- **np.median(triangles_per_node)** **Rezultat** 161

Valoarea mediană este de 161 triunghiuri, când media este aprox. 1197 triunghiuri din care face parte un nod.
Deci majoritatea nodurilor din rețea aparțin unor extrem de puține triunghiuri, în timp ce unele noduri fac parte din multe triunghiuri (ce au valori extreme CE și măresc media)

Concluzie - coeficientul mediu mare de clustering (**high average clustering coefficient**) împreună cu numărul mare de triunghiuri sunt semne ale închiderii triadice.

- Închiderea triadică înseamnă că pe măsură ce timpul trece, noi muchii (**edges**) tind să se formeze între doi utilizatori care au unul sau mai mulți prieteni comuni.
- Se explică prin faptul că Facebook sugerează de obicei noi prieteni unui utilizator atunci când există mulți prieteni comuni între utilizator și noul prieten care trebuie adăugat.
- Dar, există o sursă de stres latent. De exemplu, dacă nodul A este prieten cu nodul B și C, se acumulează ceva tensiune dacă B și C nu sunt prieteni unul cu celălalt.

1.5. Bridges - Punți/poduri

O muchie ce **unește două noduri** A și B în graf este considerată o punte (bridge),
dacă ștergerea muchiei ar face ca A și B să se afle în două componente diferite.

Se verifică dacă există punți (bridges) în această rețea cu funcția: **nx.has_bridges(G)**
Rezultat True
Deci, există punți în rețea.

Muchiile ce sunt poduri (bridges) se salvează în listă(bridges) și numărul lor (**75**) se afișează

```
bridges = list(nx.bridges(G))
len(bridges)
```

- Existența atâtor poduri (**75**) se datorează faptului că această rețea conține doar **spotlight nodes** și prietenii acestora. Ca rezultat, unii prieteni ai nodurilor reflectoare sunt conectați doar la un **spotlight node** făcând din acea muchie o punte.
- Muchiile ce sunt punți locale sunt salvate într-o listă și numărul lor este tipărit.
- În detaliu, o muchie ce unește două noduri ***C*** și ***D*** într-un graf este o punte locală, dacă punctele sale finale ***C*** și ***D*** nu au prieteni în comun.

Foarte important o muchie ce este un pod (bridge) este și un pod local (ocal bridge).
Lista va conține în total **78** bridges, incluzînd toate podurile (bridges) de mai sus

```
local_bridges = list(nx.local_bridges(G, with_span=False))
len(local_bridges)
```

Analiza Rețelei Facebook 1.5. Bridges

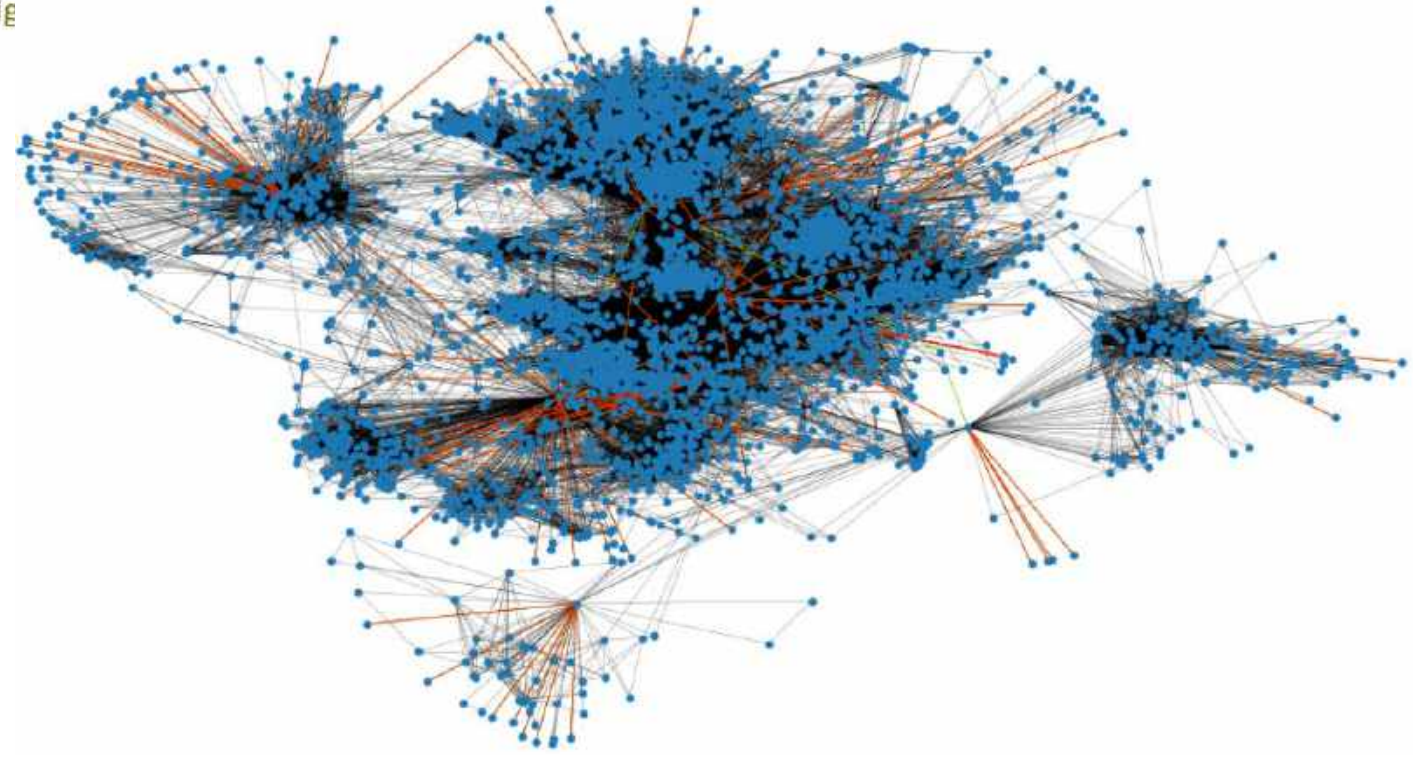
Reprezentarea bridges din rețea

Podurile au culoarea roșu și podurile locale au culoarea **verde**.

Muchiile negre nu sunt nici poduri locale, nici poduri.

Toate podurile se referă la noduri ce sunt conectate doar la un **spotlight node** (au grad valoare **1**)

```
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=10, with_labels=False, width=0.15)
nx.draw_networkx_edges(
    G, pos, edgelist=local_bridges, width=0.5, edge_color="lawngreen"
)  # green color for local bridges
nx.draw_networkx_edges(
    G, pos, edgelist=bridges, width=0.5, edge_color="r"
)  # red color for bridg
plt.axis("off")
```



1.6. Assortativity – Asortarea (similaritatea) nodurilor

Asortarea descrie **preferința ca nodurile unei rețele să se atașeze la altele care sunt similare** într-un fel.

Asortarea în ce privește gradele nodurilor (nodes degrees) se găsește în două moduri:

nx.degree_assortativity_coefficient(G) **Rezultat** 0.0635772291856494

nx.degree_pearson_correlation_coefficient(G) **Rezultat** 0.0635772291856491 8

mai rapid cu funcția scipy.stats.pearsonr

Coeficientul de asortare este coeficientul de **corelație Pearson** de grad între perechile de noduri legate (valori în intervalul [-1, 1]).

- Coeficientul de asortare **pozitiv >0,** indică o **corelație între nodurile de grad similar**,
- Coeficientul **negativ <0,** indică o **corelație între nodurile de grade diferite**.

Concluzie

- Datele FB, coeficientul de asortarea rezultat este în jur **0.064,** care este **aproape 0**.
- Deci, **rețeaua** este **almost non-assortative** și **nu se pot corela nodurile legate în funcție de gradele lor.**
- Nu se pot trage concluzii cu privire la numărul de prieteni ai unui utilizator din numărul de prieteni ai prietenilor acestuia (gradul de prieteni), deoarece folosim doar lista de prieteni a **spotlight nodes**, nodurile ce nu sunt spotlight nodes au tendința de a avea mult mai puțini prieteni.

Analiza Rețelei Facebook 1.7 Network Communities

O comunitate este **un grup de noduri,** astfel încât **nodurile din interiorul grupului** sunt **conectate cu mult mai multe muchii decât între grupuri.**

Doi algoritmi diferiți vor fi utilizați **pentru detectarea comunităților** din această rețea

1.semi-synchronous label propagation method – **propagare a etichetei** semi-sincronă

•Funcția **determină prin ea însăși (**implementarea tehnicii**) numărul de comunități** ce vor fi detectate.

•Comunitățile vor fi iterate și o listă de culori va fi creată pentru a conține aceeași culoare pentru nodurile care aparțin aceleiași comunități.

•Funcția utilizată **nx.community.label_propagation_communities(G)**

•Se tipărește **numărul de comunități** (counter):

```
colors = ["" for x in range(G.number_of_nodes())]  # initialize colors list
counter = 0
for com in nx.community.label_propagation_communities(G):
    color = "#%06X" % randint(0, 0xFFFFFF)  # creates random RGB color
    counter += 1
    for node in list(
        com
    ):  # fill colors list with the particular color for the community nodes
        colors[node] = color
counter
```

Rezultat 44 comunități detectate (**nu specifică de la început nr. comunități)**

2. asynchronous fluid communities – algoritmul comunităților fluide asincrone [2].

•**Se specifică exact numărul de comunități care trebuie detectate**.

•De exemplu se caută exact un număr de **8** comunități.

•Comunitățile vor fi iterate și o listă de culori va fi creată pentru a conține aceeași culoare pentru nodurile care aparțin aceleiași comunități.

•Funcția utilizată **nx.community.asyn_fluidc(G, 8, seed=0)** graf G, nr.comunități=8, seed=0

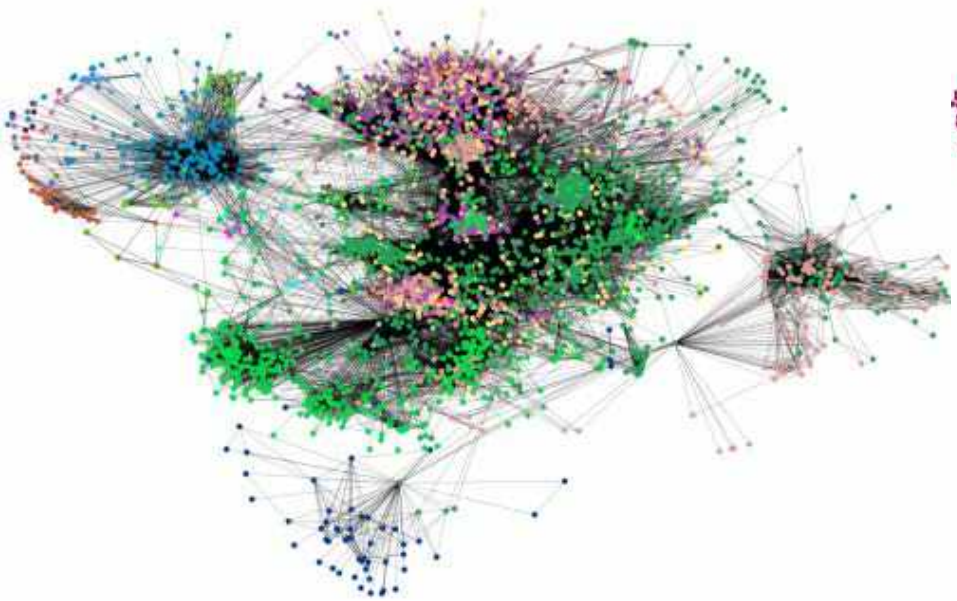
```
colors = ["" for x in range(G.number_of_nodes())]
for com in nx.community.asyn_fluidc(G, 8, seed=0):
    color = "#%06X" % randint(0, 0xFFFFFF)  # creates random RGB color
    for node in list(com):
        colors[node] = color
```

Rezultatul este graful cu 8 comunități (**specifică de la început nr. comunități)**

Reprezentare - fiecare comunitate cu o culoare diferită, iar nodurile sale sunt de obicei situate aproape unul de celălalt

```
plt.figure(figsize=(15, 9))
plt.axis("off")
nx.draw_networkx(
    G, pos=pos, node_size=10, with_labels=False, width=0.15, node_color=colors
)
```

1. **semi-synchronous label propagation:**
Facebook network grupat în 44 comunități



2. **asynchronous fluid:**
Facebook network grupat în 8 comunități