

Community Algorithms

Algoritmi de Detectare a Comunității

Community Algorithms - Algoritmi de Detectare a Comunității

- Formarea comunității este comună în toate tipurile de rețele, iar identificarea acestora este esențială pentru evaluarea comportamentului de grup și a fenomenelor emergente.
- Principiul general în găsirea comunităților este ca membrii săi vor avea mai multe relații în cadrul grupului decât cu noduri din afara grupului lor.
- Identificarea seturilor asociate dezvăluie **grupuri de noduri, grupuri izolate și structura rețelei**.
- Aceste informații ajută la deducerea unui comportament sau preferințe similare ale grupurilor de colegi, la estimarea rezilienței, la găsirea de relații imbricate și la pregatirea datelor pentru alte analize.
- Algoritmii de detectare a comunității sunt utilizati în mod obișnuit și pentru a produce vizualizarea rețelei pentru inspecția generală.
- Se oferă detalii despre cei mai reprezentativi algoritmi de detectare a comunității:
 - 1) **Triangle Count & Clustering Coefficient**,
contorizare triunghiuri și a coeficientului de grupare pentru densitatea generală a relației.
 - 1) **Strongly Connected Components & Connected Components**,
componente puternic conectate și Componente conectate pentru a găsi clustere conectate.
 - 1) **Label Propagation**,
propagarea etichetelor pentru deducerea rapidă a grupurilor pe baza etichetelor nodurilor.
 - 1) **Louvain Modularity**
modularitatea Louvain pentru analiza calității grupării și a ierarhiilor

Community Algorithms - Algoritmi de Detectare a Comunității

- Se va explica modul de funcționare al algoritmilor.
- Dacă sunt algoritmi speciali, folosim relații ponderate pentru acești algoritmi, deoarece sunt de obicei folosiți pentru a surprinde semnificația diferitelor relații.
- Figura ofera o privire de ansamblu asupra diferențelor dintre algoritmii de detectare a comunității tratați aici.

Measuring Algorithms (Algoritmi de măsurare)

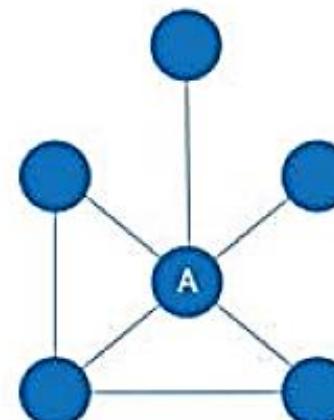
- **Triangle Count**, contorizare triunghiuri
numărul triunghiurilor ce trec printr-un nod,
ex. nodul A are 2 triunghiuri
- **Clustering coefficient (CC)** coeficientul de grupare
este probabilitatea ca vecinii nodului să fie conectați
intre ei.

Ex. nodul A are CC=0.2

Oricare 2 noduri conectate cu A au 20% =1/5
probabilitatea/șansa de a fi conectate între ele,
în total nodul A are 5 vecini.

Acstea măsuri pot fi contorizate/normalize global.

Measuring Algorithms



Triangle Count

The number of triangles that pass through a node. A has two triangles.

Clustering Coefficient

The probability that the neighbors of a node are connected to each other.

A has a 0.2 CC. Any 2 nodes connected to A have a 20% chance of being connected to each other.
These measures can be counted/normalized globally.

Community Algorithms - Algoritmi de Detectare a Comunității

Components Algorithms

- **Connected Components**, componente conectate

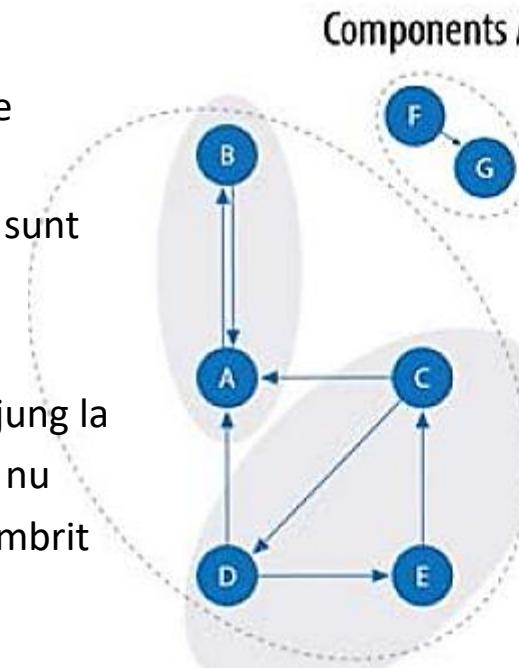
Mulțimile cu toate nodurile ce ajung la (reach) toate celelalte noduri, indiferent de direcție. ex. 2 mulțimi sunt ilustrate cu linii punctate $\{A,B,C,D,E\}$ și $\{F,G\}$

- **Strongly Connected Components**, componente puternic conectate. Mulțimile cu toate nodurile ce ajung la (reach) toate celelalte noduri, în ambele direcții, dar nu necesar în mod direct. ex. 2 mulțimi sunt ilustrate umbrit (shadow) $\{A,B\}$ și $\{C,D,E\}$

Label Propagation Algorithm

- Se dispersează etichetele la sau dinspre vecini **în căutare de clustere** (grupuri)
- Se rulează **multiple iterări**.

Greutățile (weights) ale relațiilor și/sau a nodurilor sunt de obicei utilizate pentru determinarea etichetei de "popularitate" într-un grup.



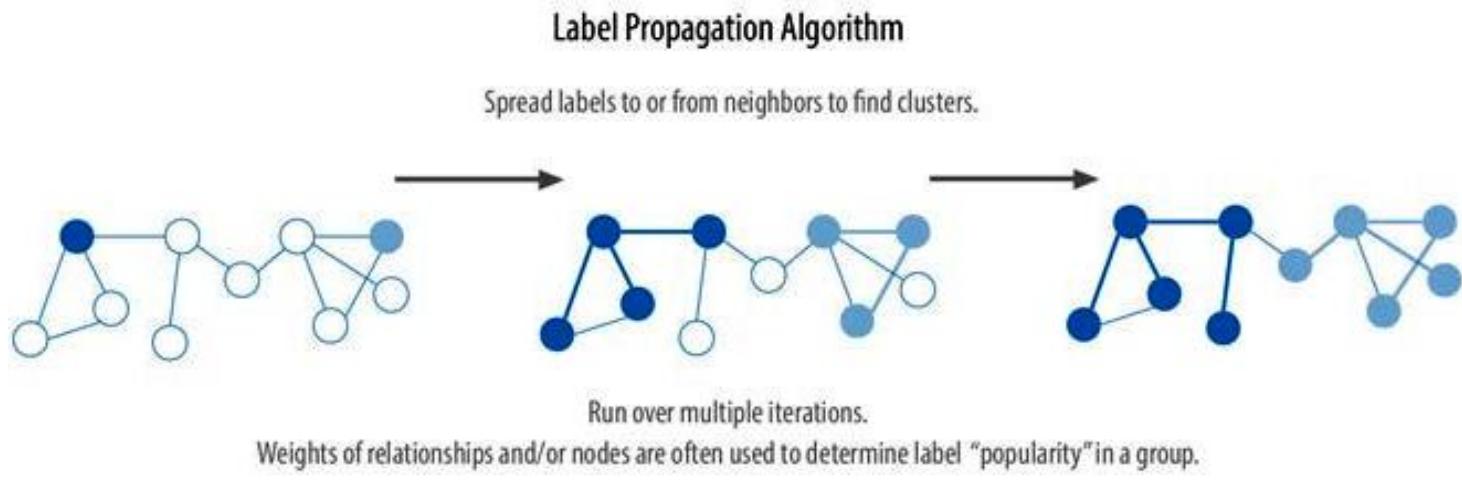
Components Algorithms

Connected Components

Sets where all nodes can reach all other nodes, regardless of direction.
2 sets shown with dashed outlines: $\{A,B,C,D,E\}$ and $\{F,G\}$.

Strongly Connected Components

Sets where all nodes can reach all other nodes in both directions, but not necessarily directly.
2 sets shown shaded:



Community Algorithms - Algoritmi de Detectare a Comunității

Observații Se utilizează termenii set (mulțime), partiție, cluster, grup și comunitate în mod interschimbabil. Sunt moduri diferite ce indică faptul că noduri similare pot fi grupate.

- Algoritmii de detectare a comunității sunt numiți **algoritmi de grupare și partiționare**.
- Se utilizează termenii cei mai importanți în literatură pentru un anumit algoritm.

Atenție la densitatea relațiilor

1) dacă graful este **foarte dens**, pot ajunge toate nodurile reunite în unul sau doar câteva grupuri.

Rezolvarea densității - prin filtrarea după grad, ponderi ale relațiilor sau valori de similitudine.

2) dacă graful este **prea rar cu** puține noduri conectate, se poate ajunge cu fiecare nod în propriul său cluster. **Rezolvare raritate** - încorporați tipuri de relații suplimentare ce să poarte informații mai relevante.

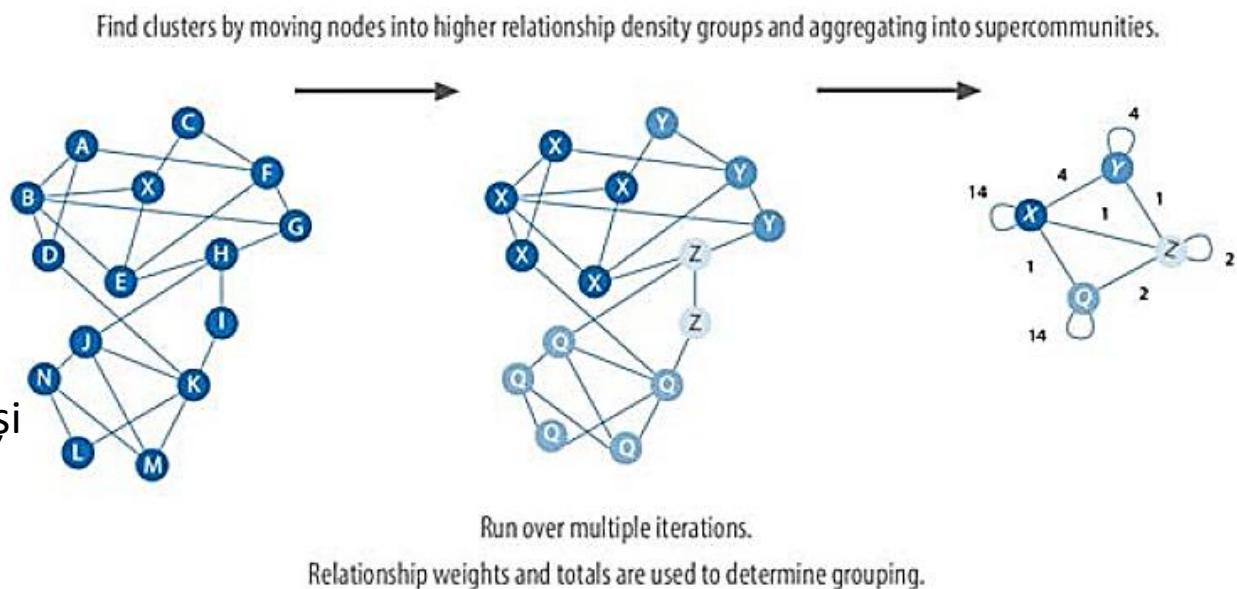
Louvain Modularity Algorithm

Găsește clustere (grupuri de noduri) în relații cu grad înalt de densitate și agregând în **supercomunități**.

(generalizare)

- Se rulează multiple iterații.
- Greutățile relațiilor (weights) și totalurile sunt utilizate în determinarea grupurilor.

Louvain Modularity Algorithm



Community Algorithms - Algoritmi de Detectare a Comunității

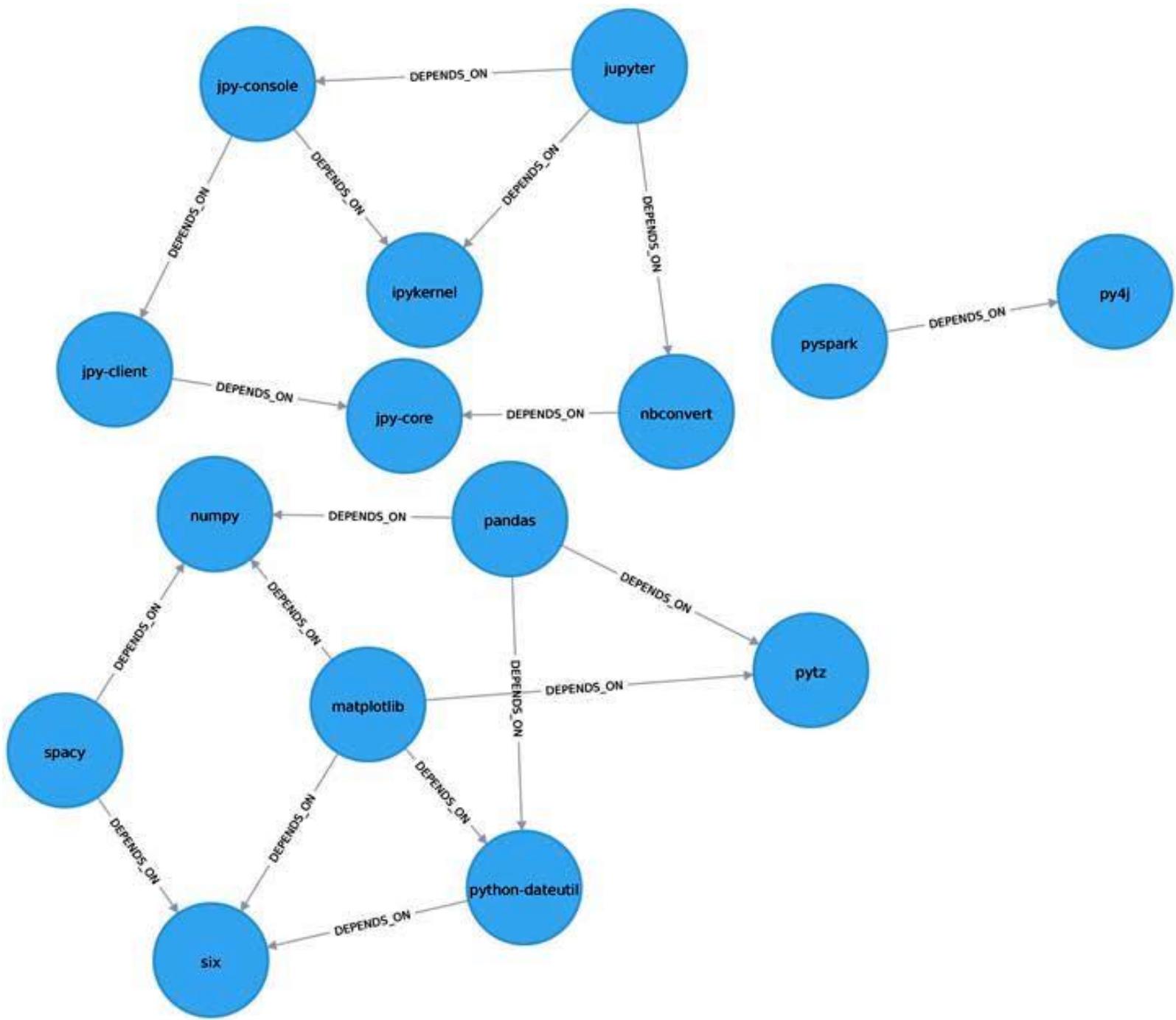
Tabelul ofera o referință rapidă cu privire la ceea ce calculeaza fiecare algoritm cu exemple de utilizare.

Tip Algoritm	Detalii	Exemple utilizare
Triangle Count & Clustering Coefficient	Măsoară câte noduri formează triunghiuri și gradul de la care nodurile tend să se grupeze în clustere.	Estimarea stabilității unui grup și dacă rețeaua are comportamente specifice "small-world" în grafuri cu clustere apropiat legate.
Strongly Connected Components	Găsirea de grupuri în care fiecare nod poate fi ajuns (reachable) din oricare alt nod în același grup bazându-se pe direcțiile relațiilor .	Realizarea de recomandări de produse bazându-se pe afilierea la un grup sau pe itemi similari.
Connected Components	Găsirea de grupuri în care fiecare nod poate fi ajuns (reachable) din oricare alt nod în același grup indiferent de direcțiile relațiilor .	Realizarea de grupări rapide pentru alți algoritmi și identificarea insulelor (noduri izolate)
Label Propagation	Deducerea clusterelor (grupurilor) prin dispersarea etichetelor bazându-se pe majoritatea vecinilor.	Înțelegerea consensului în comunități sociale sau găsirea combinațiilor periculoase de medicamente la prescrieri de alte medicamente.
Louvain Modularity	Maximizează acuratețea presupusă a grupărilor prin compararea greutății relațiilor (relationship weights) și a densităților la o estimare definită sau o medie (average).	Utilizare în analiza Fraudei , evaluare dacă un grup are doar câteva comportamente răuvoitoare sau acționează o fraudă continuă (fraud ring).

Example Graph Data: The Software Dependency Graph

- Grafurile de dependență sunt potrivite pentru a demonstra diferențele uneori subtile dintre algoritmii de detectare a comunității, deoarece acestea tend să fie mai conectate și mai ierarhice.
- Exemplile sunt executate pe un graf ce conține dependențe între bibliotecile Python, deși grafurile de dependență sunt utilizate în diferite domenii, de la software la rețele energetice.
- Acest tip de graf de dependență de software este folosit de dezvoltatori pentru a ține evidență interdependențelor tranzitive și a conflictelor din proiectele software.
- Se pot descărca nodurile și fișierele din depozitul GitHub `sw-nodes.csv`, `sw-relationships.csv` (<https://github.com/neo4j-graph-analytics/book/raw/master/data/>)
- Figura include graful ce va fi construit.
- Privind acest graf, vedem că există trei grupuri de biblioteci.
- Se vor folosi vizualizări pe seturi de date mai mici (ex. setul `matplotlib`) ca instrument pentru a ajuta la validarea clusterelor derivate de algoritmii de detectare a comunității.

src	dst	relationship	id
pandas	numpy	DEPENDS_ON	six
pandas	pytz	DEPENDS_ON	pandas
pandas	python-dateutil	DEPENDS_ON	numpy
python-dateutil	six	DEPENDS_ON	python-dateutil
pyspark	py4j	DEPENDS_ON	pytz
matplotlib	numpy	DEPENDS_ON	pyspark
matplotlib	python-dateutil	DEPENDS_ON	matplotlib
matplotlib	six	DEPENDS_ON	spacy
matplotlib	pytz	DEPENDS_ON	py4j
spacy	six	DEPENDS_ON	jupyter
spacy	numpy	DEPENDS_ON	jpy-console
jupyter	nbconvert	DEPENDS_ON	nbconvert
jupyter	ipykernel	DEPENDS_ON	ipykernel
jupyter	jpy-console	DEPENDS_ON	jpy-client
jpy-console	jpy-client	DEPENDS_ON	jpy-core
jpy-console	ipykernel	DEPENDS_ON	
jpy-client	jpy-core	DEPENDS_ON	
nbconvert	jpy-core	DEPENDS_ON	



Triangle Count & Clustering Coefficient

- Algoritmii de numărare a triunghiurilor și a coeficientului de grupare sunt prezentăți deodată, deoarece sunt foarte des folosiți împreună.

Triangle Count determină numărul de triunghiuri ce trec prin fiecare nod din graf.

- Un triunghi este un **set de 3 noduri**, unde fiecare nod are o relație cu toate celelalte noduri.
- Triangle Count poate fi rulat și la nivel global pentru evaluarea setului de date global.
- Rețelele (Netwokx) cu un numar mare de triunghiuri au mai multe șanse să prezinte structuri și comportamente de “small-world”.

Scopul algoritmului Clustering Coefficient (CC) coeficientului de grupare este de a măsura cât de strâns este grupat un grup în comparație cu cât de strâns ar putea fi grupat.

- Algoritmul folosește Triangle Count în calculele deoarece oferă un raport dintre triunghiurile existente și relațiile posibile.
- O valoare maxima 1 indică o clică (clique) în care fiecare nod este conectat la fiecare alt nod.

Există două tipuri de coeficienți de clustering: **clustering local și clustering global**.

Global Clustering Coefficient

- **Coeficientul global de clustering** este suma normalizată a coeficientilor de clustering locali.
- Coeficienții de grupare ne oferă un mijloc eficient de a găsi grupuri evidente, cum ar fi ciclele (cliques), în care fiecare nod are o relație cu toate celelalte noduri, dar se pot specifica și praguri pentru a seta niveluri (ex. unde nodurile sunt conectate în proporție de 40%).

Triangle Count & Clustering Coefficient

Local Clustering Coefficient

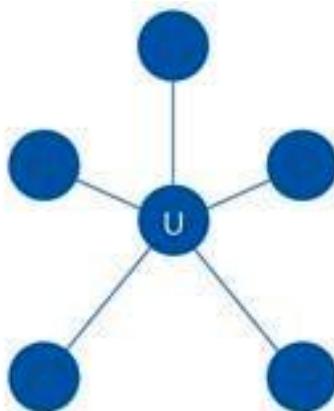
- Coeficientul de clustering local al unui nod este probabilitatea ca vecinii săi să fie și ei conectați.
- Calculul acestui scor implică numărarea triunghiurilor.
- Coeficientul de grupare al unui nod poate fi găsit înmulțind nr. triunghiuri ce trec prin nod cu 2 și apoi scufundându-l cu numarul maxim de relații din grup, gradul nodui, minus unu.
- Figura Ex. triunghiuri diferite și coeficienți de grupare pentru un nod cu 5 relații.
- Figura include un nod cu 5 relații ce face să pară că coeficientul de grupare va fi întotdeauna egal cu 10% din numarul de triunghiuri. Nu este cazul dacă modificăm numărul de relații.

Dacă schimbăm al doilea ex. cu 4 relații (și aceleași 2 triunghiuri), atunci coeficientul este 0.33.

Coeficientul de grupare pt. un nod folosește formula urmatoare

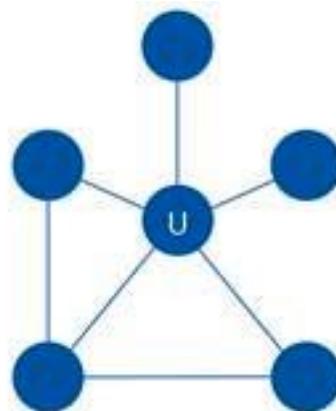
u = nod, $k(u)$ = gradul lui u , $R(u)$ = nr.relații prin vecinii lui u (folosind nr. triunghiuri ce trec prin u).

$$CC(u) = \frac{2R_u}{k_u(k_u - 1)}$$



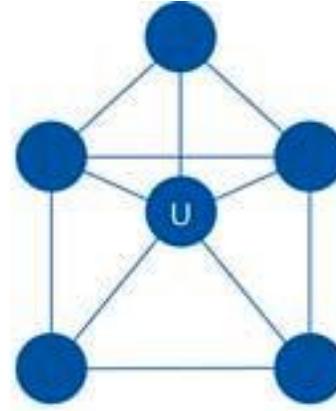
Triangles = 0
Clustering Coefficient = 0

$$CC(u) = \frac{0(2)}{5(5-1)}$$



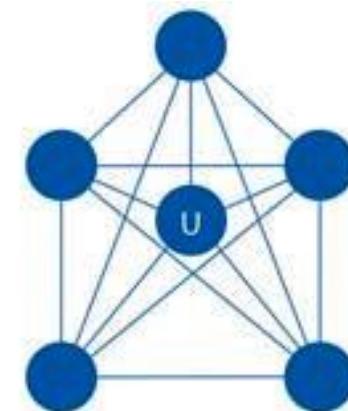
Triangles = 2
Clustering Coefficient = 0.2

$$CC(u) = \frac{2(2)}{5(5-1)}$$



Triangles = 6
Clustering Coefficient = 0.6

$$CC(u) = \frac{6(2)}{5(5-1)}$$



Triangles = 10
Clustering Coefficient = 1

$$CC(u) = \frac{10(2)}{5(5-1)}$$

Triangle Count & Clustering Coefficient

Când se utilizează Triangle Count & Clustering Coefficient?

- Utilizați Triangle Count la determinarea **stabilității unui grup** sau ca parte a calculului altor măsuri de rețea, cum ar fi coeficientul de grupare.
- Numărarea triunghiurilor este populară în **analiza rețelelor sociale**, unde este folosită pentru a detecta comunitățile.
- Coeficientul de clusterizare poate oferi probabilitatea ca nodurile alese aleator să fie conectate.
- Se poate utiliza pentru a **evalua rapid coeziunea unui anumit grup** sau a rețelei dvs.
- Împreună, acești algoritmi sunt utilizati pentru a **estima rezistența și pentru a căuta structuri de rețea**.

Exemple de cazuri de utilizare (Use case) includ:

- Identificarea caracteristicilor pentru **clasificarea** unui anumit **site web** ca conținut **spam**.
Detalii în “Efficient Semi-Streaming Algorithms for Local Triangle Counting in Massive Graphs” de L. Becchetti et al.
- **Investigarea structurii comunității a graf social al Facebook**, unde cercetatorii au găsit cartiere dense de utilizatori într-un graf global, altfel rar.
Detalii în „The Anatomy of the Facebook Social Graph”, de J. Ugander et al..
- **Explorarea structurii tematice a web-ului și detectarea comunităților de pagini cu subiecte comune pe baza legăturilor reciproce dintre ele.**
Detalii în „Curvature of Co-Links Descopes Hidden Thematic Layers in the World Wide Web”, de J.-P. Eckmann și E. Moses.

Triangle Count & Clustering Coefficient

Triangle Count Algorithm Exemplu

- Un triunghi în graful exemplu cu libraries Python, indică faptul că 2 dintre nodurile vecinilor sunt vecini. 6 libraries participă în triunghiuri.
 - Pentru a găsi ce noduri sunt în triunghiuri, intervine fluxul triunghiular.

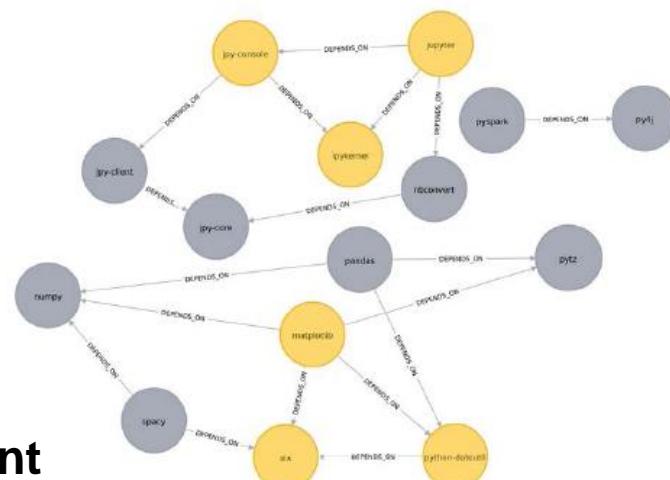
Rezultat initial

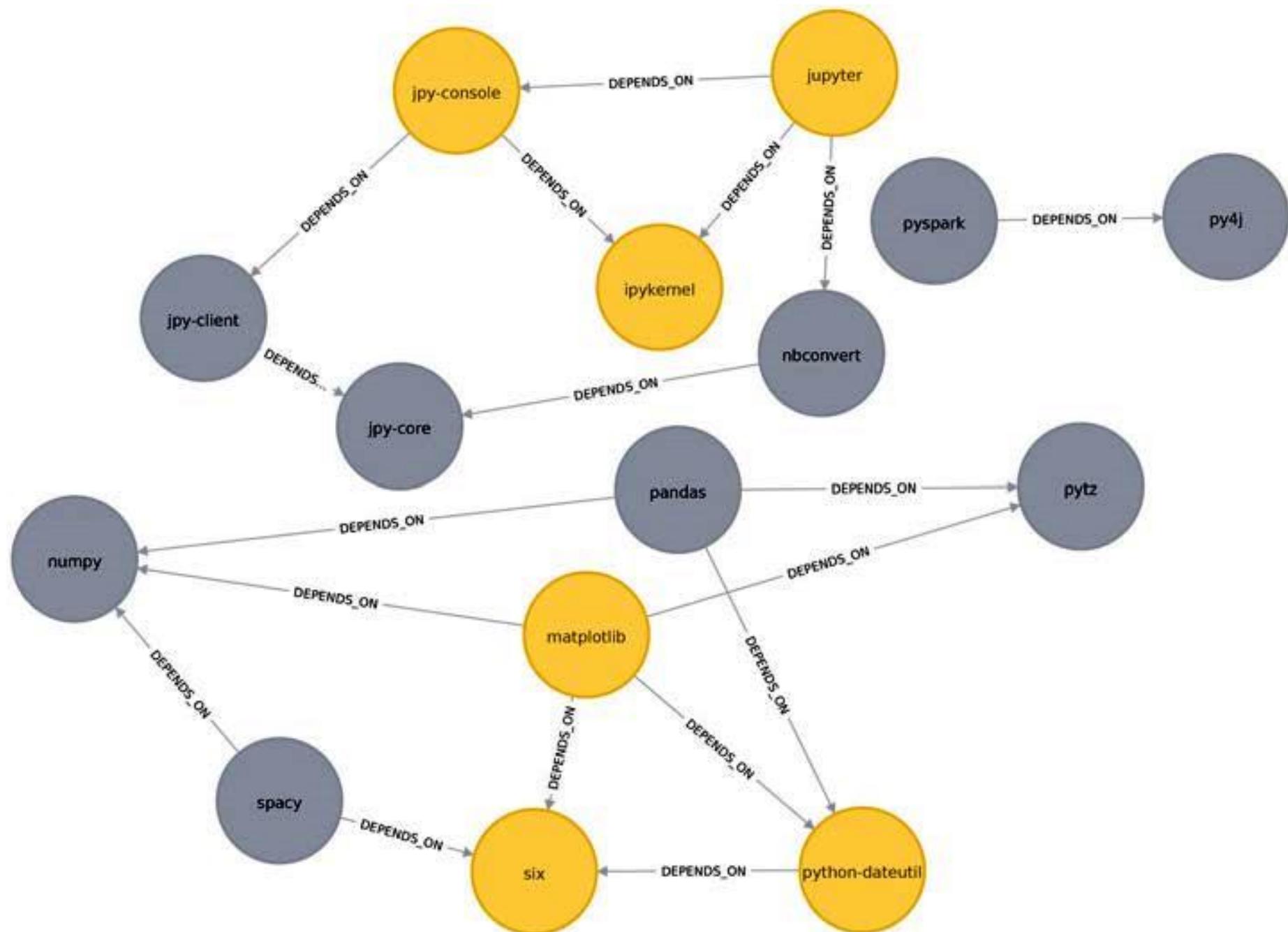
count	id
1	jupyter
1	python-dateutil
1	six
1	ipykernel
1	matplotlib
1	jpy-console

nodeA	nodeB	nodeC
matplotlib	six	python-dateutil
jupyter	jpy-console	ipykernel

Calcul Local Clustering Coefficient

- **ipykernel** are scor 1 adică toți vecinii **ipykernel** sunt vecini unul cu celalalt. Cu alte cuvinte, **comunitatea directă** din jurul **ipykernel** este **foarte coerentă**.
 - S-au filtrat nodurile cu scor de coeficient 0 în acest ex., dar nodurile cu coeficienți mici pot fi interesante.
 - Un scor scăzut găsește nodul gaură structurală - un nod bine conectat la noduri din diferite comunități care nu sunt altfel conectate între ele. Aceasta este metoda de gasirea punctilor (bridge) potențiale (Centrality).





Strongly Connected Components

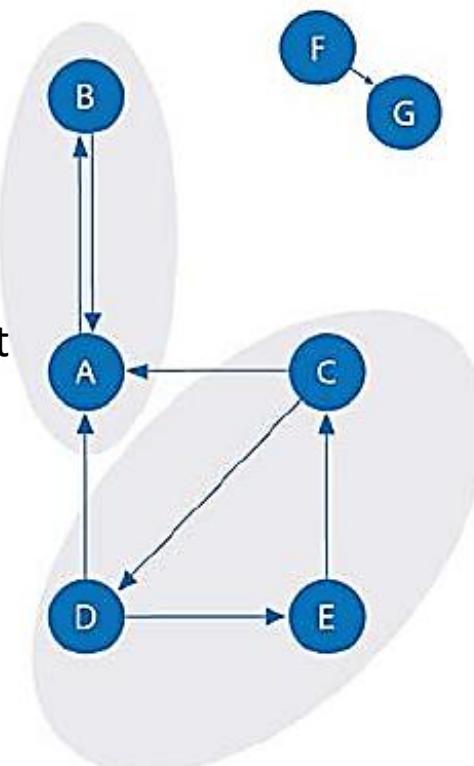
Componente puternic conectate

- Algoritmul Strongly Connected Components (SCC) este unul din cei mai vechi algoritmi.
- SCC găsește seturi de noduri conectate într-un graf direcționat în care fiecare nod este accesibil în ambele direcții de la orice alt nod din același set.
- Operațiunile sale de rulare se scalează bine, proporțional cu numărul de noduri.
- Figura ilustrează faptul că nodurile dintr-un grup SCC nu trebuie să fie vecine imediate, dar trebuie să existe căi direcționale între toate nodurile din set.
- Descompunerea unui graf orientat (direcționat) în componente sale puternic conectate este o aplicație clasică a algoritmului **Depth First Search (DFS)**.

Mulțimile cu toate nodurile ce ajung la (reach) toate celelalte noduri, în ambele direcții, dar nu necesar în mod direct.

ex. 2 mulțimi SCC sunt ilustrate umbrat (shadow) $\{A,B\}$ și $\{C,D,E\}$

În $\{C,D,E\}$ fiecare nod poate ajunge (reach) la celelalte noduri, dar uneori trebuie să treacă prin alte noduri mai întâi (ex. din E la A se trece prin C)



Strongly Connected Components

Sets where all nodes can reach all other nodes in both directions, but not necessarily directly.

2 sets of strongly connected components are shown shaded : $\{A,B\}$ and $\{C,D,E\}$

Note that in $\{C,D,E\}$ each node can reach the others, but in some cases they must go through another node first.

Strongly Connected Components (SCC)

Când se utilizează Strongly Connected Components (SCC) ?

- Utilizați SCC ca pas incipient în analiza grafului pentru a vedea cum este structurat un graf sau pentru a identifica grupuri strânse care ar putea justifica investigații independente.
- O componentă puternic conectată poate fi utilizată pentru a profila comportamente sau înclinații similare într-un grup pentru aplicații ex. **motoarele de recomandare**.
- Multii algoritmi de detectare a comunității, ex. SCC, sunt utilizati pentru a găsi și restrângere clustere în noduri unice pentru o analiză ulterioară între clustere.
- Se poate utiliza SCC pentru a vizualiza cicluri pentru analize, cum ar fi găsirea de procese ce s-ar putea bloca deoarece fiecare subproces așteaptă ca alt membru să ia masuri.

Exemple de cazuri de utilizare (Use Case):

- **Găsirea setului de firme în care fiecare membru deține** direct și/sau indirect **acțiuni** la fiecare alt membru, ca în „The Network of Global Corporate Control”, o analiză a corporațiilor transnaționale puternice de S. Vitali et al.
- **Calcularea conectivității diferitelor configurații de rețea** atunci când se măsoară performanța de rutare în rețelele fără fir multihop.
- **ACTIONEAZĂ CA PRIM PAS ÎN MULȚI ALGORITMI DE GRAF** ce funcționează numai pe grafuri puternic conectate. În rețelele sociale gasim multe grupuri puternic conectate. În aceste seturi, oamenii au adesea preferințe similare. SCC este folosit pentru a găsi astfel de grupuri și pentru a sugera pagini de apreciat sau produse de cumpărat persoanelor din grup ce nu au facut acest lucru.

Observații- Unii algoritmi au strategii pentru a scăpa de bucle infinite, dar dacă ne scriem proprii algoritmi sau găsim procese neterminative, putem folosi SCC pentru a verifica ciclurile.

Strongly Connected Components

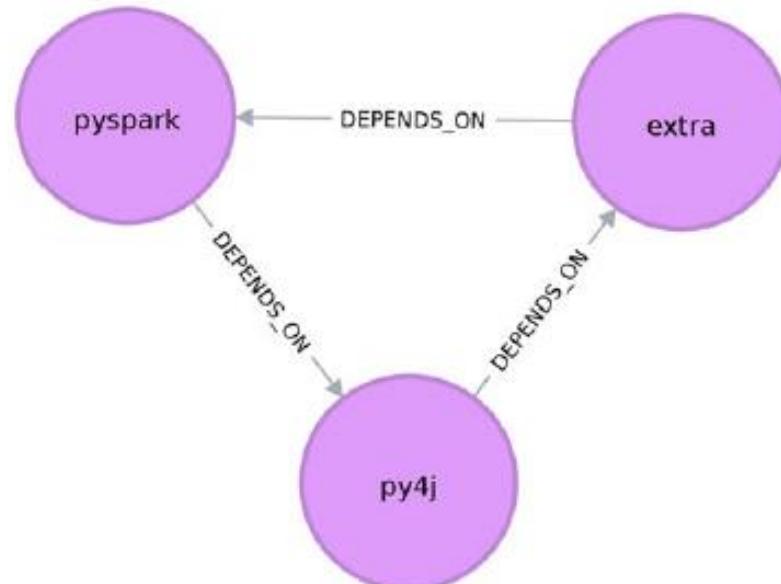
Exemple

- **Tabel 1-Inițial (dreapta)** fiecare nod din exemplul de graf este în propria sa partitie (ca rezultat al SCC)
- Până acum algoritmul a dezvăluit doar ca bibliotecile noastre Python se comportă foarte bine

- Să cream o dependență circulară în graf pentru a face lucrurile mai interesante.
- Adică vom ajunge cu niște noduri în aceeași partitie.
- Se va adauga o biblioteca suplimentară (**extra**) ce creează o dependență circulară între **py4j** și **pyspark**.

partition	libraries
1	[pyspark, py4j, extra]
8	[ipykernel]
11	[six]
2	[matplotlib]
5	[jupyter]
14	[numpy]
13	[pandas]
7	[nbconvert]
10	[jpy-core]
9	[jpy-client]
3	[spacy]
15	[python-dateutil]
6	[jpy-console]
0	[pytz]

Tabel (stânga) și Figura - pyspark, py4j și extra
 fac toate parte din aceeași partitie, iar SCC-urile ne ajuta să găsim dependența circulară.



partition	libraries
8	[ipykernel]
11	[six]
2	[matplotlib]
5	[jupyter]
14	[python-dateutil]
13	[numpy]
4	[py4j]
7	[nbconvert]
1	[pyspark]
10	[jpy-core]
12	[pandas]
3	[spacy]
6	[jpy-console]

Connected Components

- **Algoritmul Connected Components** (numit uneori **Union Find** sau **Weakly Connected Components**) găsește seturi de noduri conectate într-un graf NEORIENTAT (nedirecționat) în care fiecare nod este accesibil de la orice alt nod din același set.
- Difera de algoritmul SCC deoarece are nevoie doar de o cale pentru a exista între perechi de noduri într-o direcție, în timp ce SCC are nevoie de o cale pentru a exista în ambele direcții.
- Bernard A. Galler, Michael J. Fischer au descris pentru prima data acest algoritm în lucrarea lor din 1964, „An Improved Equivalence Algorithm”.

Când se utilizează Connected Components?

- **Similar cu SCC** Componentele conectate sunt adesea folosite la începutul unei analize pentru a înțelege structura unui graf.
- Deoarece se scalează eficient, se utilizează **pentru grafurile ce necesită actualizari frecvente**.
- Poate arăta rapid **noi noduri comune** între grupuri, ceea ce este util pentru analize, ex. detectarea fraudelor.
- Este obișnuit a rula Alg. Componente conectate pentru a testa dacă un graf este conectat ca pas prezentator pentru analiza generală a grafului.
- Efectuarea acestui test rapid poate **evita rularea accidentală a algoritmilor pe o singură componentă deconectată a unui graf și obținerea de rezultate incorecte**.

Exemple de cazuri de utilizare (Use case):

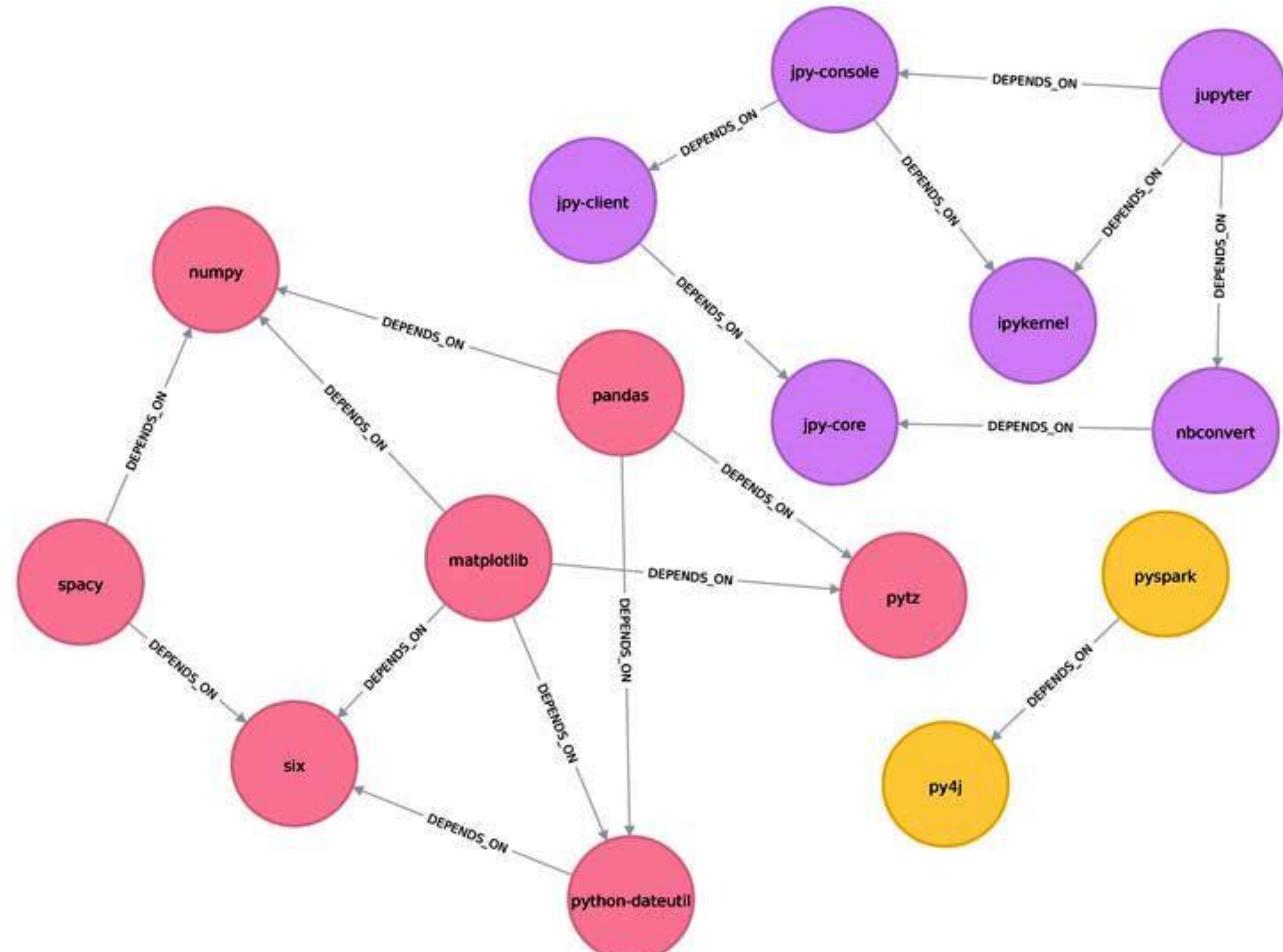
- **Urmărirea grupurilor de înregistrari de baze de date**, ca parte a procesului de deduplicare. Deduplicarea este o importantă în aplicațiile de gestionare a bazelor de date.
Detalii în „An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records”, de A. Monge și C. Elkan.
- **Analiza rețelelor de citare**. Se folosește Alg. Componente conectate pentru a afla cât de bine este conectată o rețea și apoi pentru a vedea dacă conectivitatea rămâne dacă nodurile „hub” sunt mutate din graf.
Detalii în „Characterizing and Mining Citation Graph of Computer Science Literature” de Y. An et al.

Connected Components

Exemplu rulare Componente Conectate

Table de rezultate și Figura - există 3 componente conectate. Algoritmul se utilizează și pentru grafuri de dimensiune mare.

component	libraries
180388626432	[jpy-core, nbconvert, ipykernel, jupyter, jpy-client, jpy-console]
223338299392	[spacy, numpy, six, pandas, pytz, python-dateutil, matplotlib]
936302870528	[pyspark, py4j]



- Ambii algoritmi de detectare a comunității pe care i-am acoperit pâna acum sunt **determiniști**, adică returnează aceleași rezultate de fiecare dată când îi rulăm.
- Urmatorii doi algoritmi sunt exemple de algoritmi **nedeterminiști**, unde putem vedea rezultate diferite dacă le rulam de mai multe ori, chiar și pe aceleași date.

Label Propagation Algorithm (LPA)

- Algoritmul de propagare a etichetei (LPA) este un algoritm **rapid pentru găsirea comunităților într-un graf**.
- În LPA, nodurile își **selectează grupul în funcție de vecinii lor direcți**.
- Procesul este potrivit pentru rețelele în care grupările sunt mai puțin clare și ponderile se folosesc pentru a ajuta nodul să determine comunitatea în care să se plaseze.
- Este eficient și pentru învățarea semisupravegheată (**semisupervised learning**) , deoarece se poate însemna procesul cu etichete de noduri orientative prealocate.
- Intuiția din spatele acestui algoritm este că **o singură etichetă poate deveni rapid dominantă** într-un grup de noduri dens conectat, **dar va avea probleme în a traversa o regiune slab conectată**.
- Etichetele sunt prinse într-un grup de noduri dens conectat, iar nodurile ce ajung cu aceeași etichetă atunci când algoritmul se termină sunt considerate parte a aceleiași comunități.
- LPA **rezolvă suprapunerile**, în care nodurile sunt potențial parte din mai multe clustere, prin atribuirea apartenenței la vecinatatea etichetei cu cea mai mare relație combinată și greutatea nodului.
- LPA este un algoritm relativ nou propus în 2007 în “Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks” de U. N. Raghavan et al.

Label Propagation Algorithm (LPA)

- Există două variante ale Propagării etichetelor, o metoda simplă de **push** și metoda de **pull** mai tipică care se bazează pe ponderile relațiilor.
- Metoda **pull** se pretează bine paralelizării.

Etapele pentru metoda - Label Propagation **pull**:

1. Fiecare nod este inițializat cu o etichetă unică (un identifier) și, optional, pot fi utilizate etichete preliminare „**seeds**”
2. Aceste etichete se propagă prin rețea.
3. La fiecare iterație de propagare, fiecare nod își actualizează eticheta pentru a se potrivi cu cea cu greutatea maxima, care este calculată pe baza greutăților nodurilor vecine și a relațiilor lor. Legaturile sunt rupte uniform și aleator.
4. LPA ajunge la convergență atunci când fiecare nod are eticheta majoritară a vecinilor săi.

Pe măsură ce etichetele se propagă, grupurile de noduri dens conectate ajung rapid la un consens asupra unei etichete unice.

La sfârșitul propagării, ramân doar câteva etichete, iar nodurile cu aceeași etichetă aparțin aceleiași comunități.

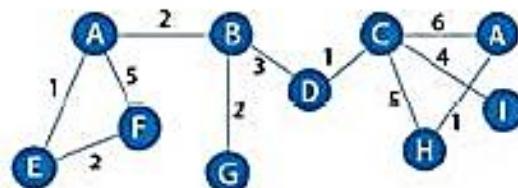
Label Propagation Algorithm (LPA)

Label Propagation-PULL Deplasează etichetele dinspre vecini bazându-se pe greutățile relațiilor (weights) pentru a găsi clustere (grupuri)

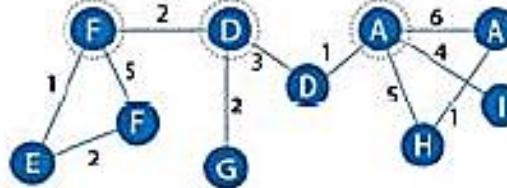
- 1- Exemplu cu **2 noduri cu aceeași etichetă (A)**. Toate celelalte au etichete unice. Se ignora greutățile nodurilor 1.
- 2- **Nodurile sunt aranjate random, A,B,C devin F, D, A (A devine F (max (5,2,1)=5, B devine D (max (2,2,3)=3, C devine A, max(1,4,5,6)=6) cu cel mai mare total al greutății.**
- 3- Vechile noduri F,D,A nu își modifică eticheta deoarece nodurile cu cele mari weights au aceeași denumire cu ele.
- 4- Se continuă etichetarea celorlalte noduri cu **F, D, A (E devine F, G devine D, I și H devin A, max. weights)**
- 5- Sunt **identificate 3 clustere**. Etichetele F,D,A nu au în acest ex. nici o semnificație.

Label Propagation—PULL

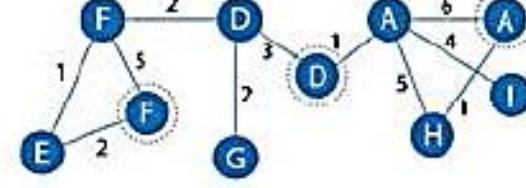
Pulls Labels from Neighbors Based on Relationship Weights to Find Clusters



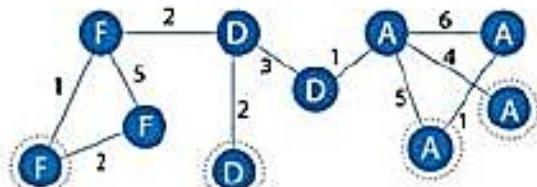
Example with 2 nodes with same label, "A." All others are unique. Node weights default to 1 and in this example are ignored.



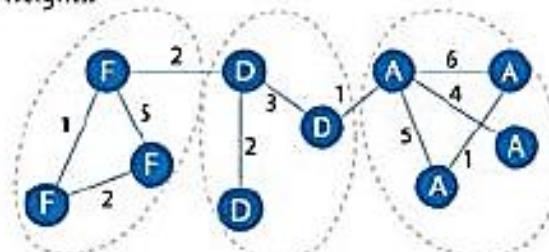
Nodes are shuffled for processing order and each node considers its direct neighbors' labels (3 are highlighted above). Nodes acquire the label matching the total highest relationship weights.



Note that these 3 nodes don't change labels because their highest weight relationships in this step have the same label.



This continues until all nodes have updated their labels.



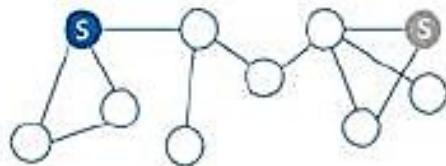
3 clusters are identified. The labels themselves have no meaning.

Label Propagation Algorithm (LPA)

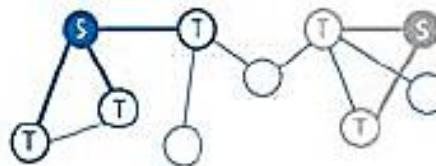
- Label Propagation-PUSH** Deplasează etichetele către vecini pentru a găsi clustere (grupuri)
- 1- Exemplu de 2 noduri date cu etichete seeds (S)
 - 2- Cele 2 Seeds caută vecinii din imediata apropiere ca target (T) pentru a dispersa etichetele lor.
 - 3- Dacă nu există nici un conflict, etichetele (labels) se dispersează în noile noduri vecini.
 - 4- Noile noduri etichetate (S) sunt activate ca noi种子
 - 5- Conflicturile (ex. același vecin vizitat de noduri diferite) se rezolvă pe baza unui set de măsuri, de exemplu pe baza greutăților relațiilor (weights) (ex. greutatea mare/mică).
 - 6- Procesul continuă până cand sunt updateate toate nodurile. Aici sunt identificate 2 clustere. LPA poate fi rulat cu etichete seeds + noduri neetichetate sau fiecare nod pornind de la o etichetă unică. Cu cât etichetele sunt mai unice cu atât se folosește mai mult rezolvarea conflictelor.

Label Propagation—PUSH

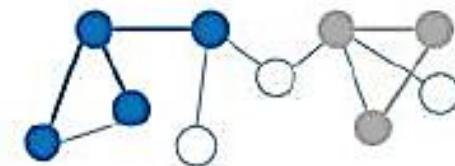
Pushes Labels to Neighbors to Find Clusters



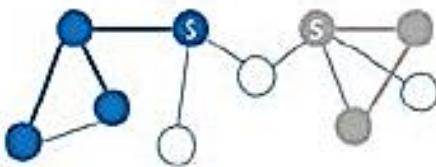
Example of 2 nodes given seed labels.



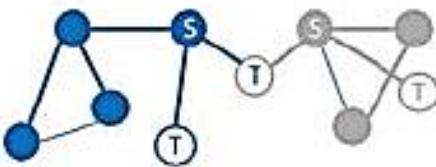
They look for immediate neighbors as targets to spread their labels to.



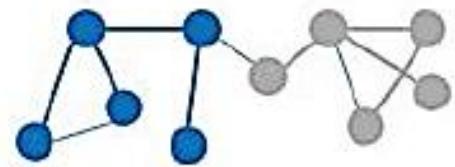
Where there is no conflict the label spreads.



The recently labeled nodes now activate like new seeds.



Conflicts are resolved based on a set measure such as relationship weights.



The process continues until all nodes are updated. 2 clusters are identified.

LPA can be run with seed labels plus unlabeled nodes or each node starting with a unique label.

The more unique labels, the more conflict resolution used.

Label Propagation

Semi-Supervised Learning & Seed Labels- Învățare semi-supervizată și etichete pentru semințe

- Spre deosebire de alți algoritmi, Label Propagation poate returna structuri de comunitate diferite atunci când este rulat de mai multe ori pe același graf.
- **Ordinea în care LPA evalueaza nodurile poate avea o influență asupra comunităților finale** pe care le returnează.

Gama de soluții este restrânsă atunci când unor noduri li se dă etichete preliminare (adica, etichete de seeds), în timp ce celelalte sunt neetichetate.

- Este mai probabil ca nodurile neetichetate să adopte etichetele preliminare.
- Aceasta utilizare a Label Propagation poate fi considerată o metodă de învățare semi-supervizată pentru a găsi comunități.
- Învățarea semi-supervizată este o clasă de sarcini și tehnici de învățare automată care funcționează pe o cantitate mică de date etichetate, împreună cu o cantitate mai mare de date neetichetate.
- Se poate rula algoritmul în mod repetat pe grafuri pe măsură ce acestea evoluează.
- În cele din urmă, LPA uneori nu converge către o singură soluție.
- În această situație, rezultatele comunității se vor schimba continuu între câteva comunități remarcabil de similare, iar algoritmul nu s-ar finaliza niciodată.
- **Etichetele semințelor (seeds) ajută la ghidarea acesteia către o soluție.**

Label Propagation

Când se utilizează Label Propagation?

- Utilizați Propagarea etichetelor în rețele la scară largă pentru detectarea inițială a comunității, mai ales atunci când sunt disponibile ponderi.
- Acest algoritm poate fi **paralelizat** și, deci extrem de rapid la partaționarea grafurilor.

Exemple de cazuri de utilizare (Use case) includ:

- **Atribuirea polarității tweet-urilor ca parte a analizei semantice.**

În acest scenariu, etichetele de seeds pozitive și negative dintr-un clasificator sunt utilizate în combinație cu graful de următori Twitter.

Detalii “Twitter Polarity Classification with Label Propagation over Lexical Links and the Follower Graph”, de M. Speriosu et al..

- **Găsirea de combinații potențial periculoase de posibile medicamente prescrise concomitent**, pe baza similarității chimice și a profilurilor efectelor secundare.

Detalii în “Label Propagation Prediction Interactions Based on Clinical Side Effects”, de P. Zhang et al..

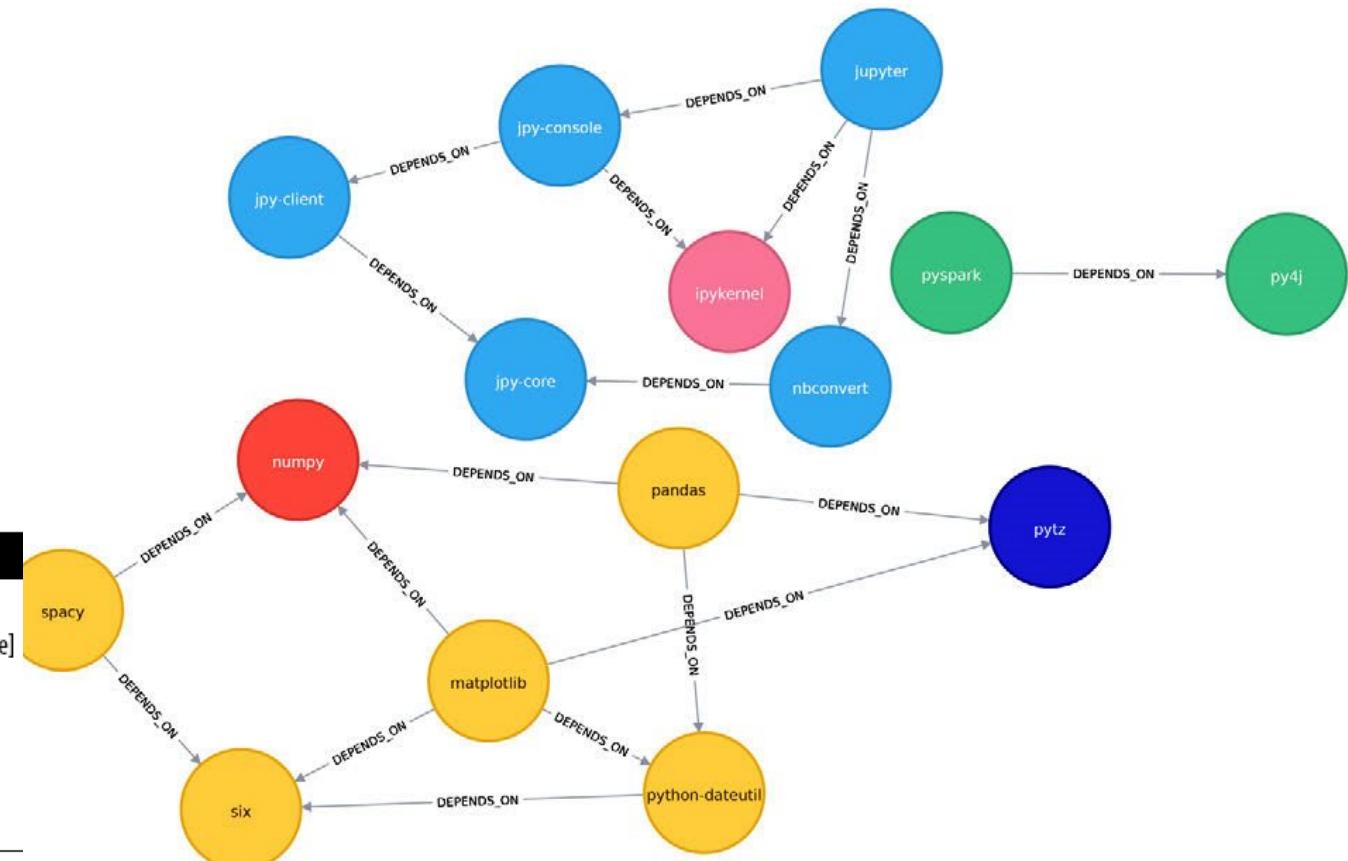
- **Deducerea caracteristicilor de dialog și a intenției utilizatorului** pentru un model de învățare automată.

Detalii „Feature Inference Based on Label Propagation on Wikidata Graph for DST”, de Y. Murase et al..

Label Propagation

Exemplu Graf orientat

- În comparație cu Componentele conectate, rezultă mai multe grupuri de biblioteci în exemplu.
- LPA este mai puțin strict decât Componentele conectate în ceea ce privește modul în care determină clusterele.
- Doi vecini (noduri conectate direct) pot fi găsite în grupuri diferite utilizând Propagarea etichetei.
- Folosind Componente conectate, un nod ar fi totdeauna în același cluster cu vecinii săi, deoarece alg. bazează gruparea strict pe relații.
- În exemplul, diferența evidentă este că bibliotecile Jupyter au fost împărțite în 2 comunități - una cu părțile de bază ale bibliotecii (label 11) și cealaltă cu instrumente orientate client (label 10).

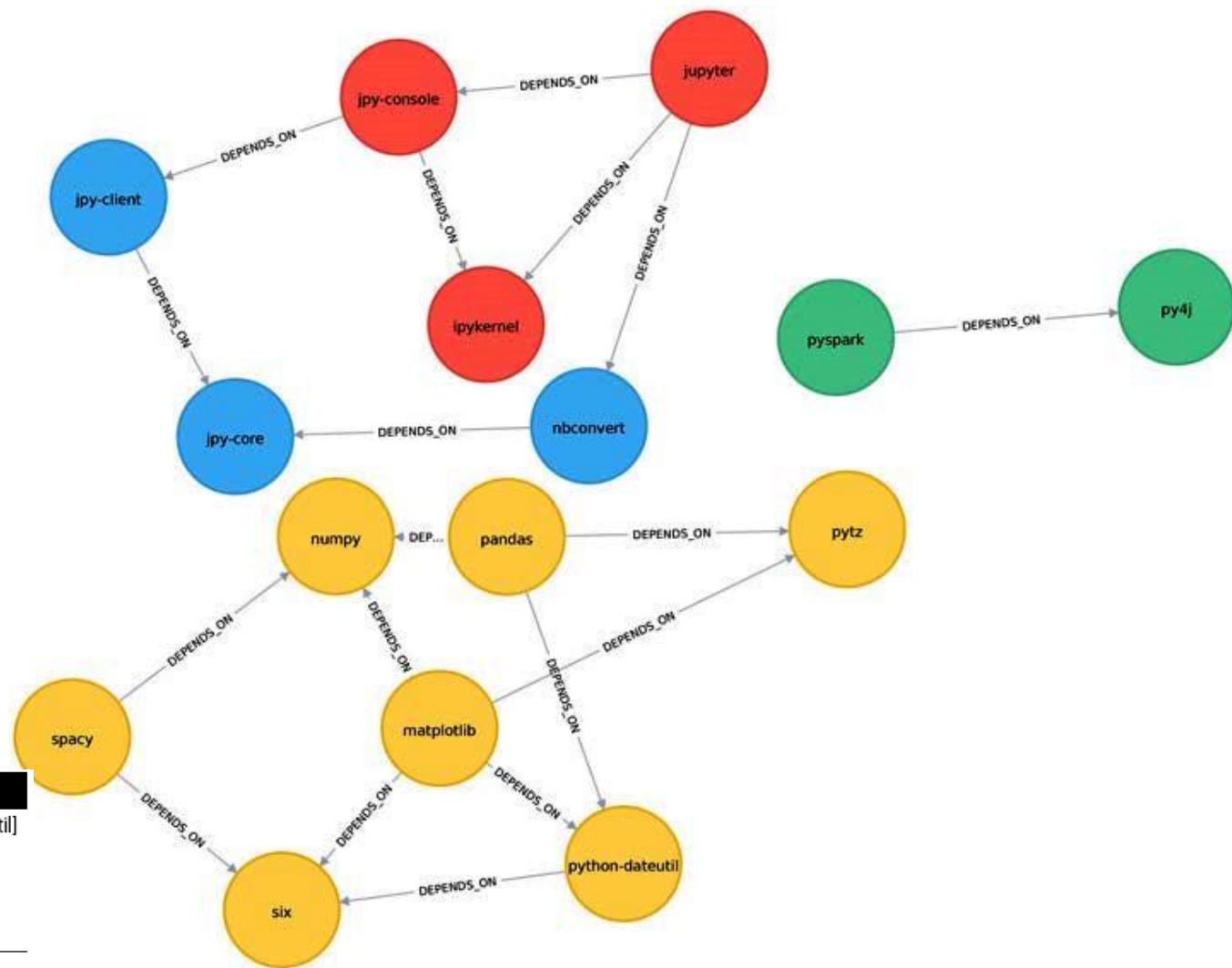


label	libraries
11	[matplotlib, spacy, six, pandas, python-dateutil]
10	[jupyter, jpy-console, nbconvert, jpy-client, jpy-core]
4	[pyspark, py4j]
8	[ipykernel]
13	[numpy]
0	[pytz]

Label Propagation

Graf Neorientat, Label Propagation nr. de clustere a scăzut de la 6 (graf orientat) la 4 (graf neorientat), iar toate nodurile din partea **matplotlib** a grafului sunt acum grupate împreună.

- Label Propagation pe grafuri complexe se văd diferențe mai semnificative între grafuri orientate-neorientate
- Ignorarea direcției face ca nodurile să încearcă să adopte mai multe etichete, indiferent de sursa relației.



label libraries

label	libraries
11	[pytz, matplotlib, spacy, six, pandas, numpy, python-dateutil]
10	[nbconvert, jpy-client, jpy-core]
6	[jupyter, jpy-console, ipykernel]
4	[pyspark, py4j]

Louvain Algorithm -Modularitatea Louvain

- Algoritmul Louvain Modularity găsește clustere comparând densitatea comunității, deoarece atribuie noduri diferitelor grupuri.
- Este similar cu analiza „ce ar fi dacă” pentru a încerca diverse grupuri cu scopul de a ajunge la un optim global.
- Propus în 2008, algoritmul Louvain este unul dintre cei mai **rapizi** algoritmi bazați pe modularitate.
- Pe lângă detectarea comunităților, dezvaluie și o ierarhie a comunităților la diferite scări/nivele de ierarhie.
- Acest lucru este util pentru înțelegerea structurii unei rețele la diferite niveluri de granularitate.
- Louvain **cuantifică cât de bine este alocat un nod unui grup** analizând densitatea conexiunilor din cluster în comparație cu un eșantion mediu sau aleator.
- Aceasta *măsura a atribuirii comunității* se numește **modularitate**.

Grupare bazată pe calitate prin modularitate (Quality-based grouping via modularity)

- Modularitatea este o tehnică de descoperire a comunităților prin împărțirea unui graf în module mai grosiere (sau clustere) și apoi măsoără puterii grupărilor.
- Metoda compară densitățile relațiilor din clustere date cu densitățile dintre clustere.
- **Măsura calității acestor grupări** se numește **modularitate**.
- Algoritmii de modularitate optimizează comunitățile la nivel local și apoi global, folosind mai multe iterații pentru a testa diferite grupări și pentru a mări clusterul.
- Aceasta strategie **identifică ierarhiile comunității** și oferă o înțelegere largă a structurii generale.

Dezavantaje ale tuturor algoritmilor de modularitate :

1. **Unesc comunități mai mici în altele mai mari.**
2. **Un platou** poate apărea acolo unde sunt prezente mai multe opțiuni de partitie cu modularitate similară, formând **maxime locale și împiedicând progresul**.

Detalii în „The Performance of Modularity Maximization in Practical Contexts”, de B. H. Good et al.

Louvain algorithm - Modularitatea Louvain

Calcularea modularității

- Un calcul simplu al modularității se bazează pe fracția dintre relațiile din cadrul grupurilor date minus fracția așteptată dacă relațiile au fost distribuite la întâmplare între toate nodurile.
- Valoarea este întotdeauna între 1 și -1, valorile pozitive indicând o densitate mai mare a relației decât v-ați așteptat întâmplator, iar valorile negative indicând o densitate mai mică.
- Figura ilustrează mai multe scoruri de modularitate diferite bazate pe grupări de noduri.
- Formula pentru modularitatea unui grup este:

- L este numarul de relații din întregul grup.
- L_c este numarul de relații dintr-o partitură.
- k_c este gradul total de noduri dintr-o partitură

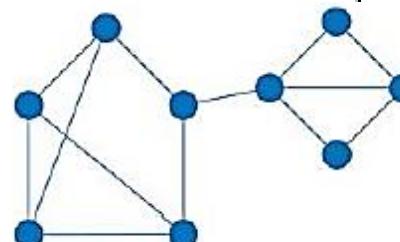
Calculul pentru partitură optimă din partea de sus a figurii:

$$\text{Partițiile dark} \quad \left(\frac{7}{13} - \left(\frac{15}{2(13)} \right)^2 \right) = 0.205$$

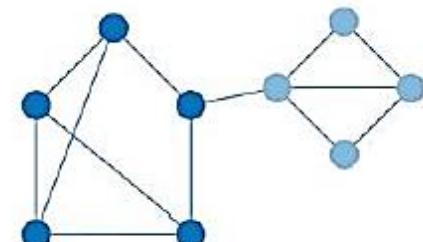
$$\text{Partiția light} \quad \left(\frac{5}{13} - \left(\frac{11}{2(13)} \right)^2 \right) = 0.206$$

- **Optimal Partition**=se adună p.dark+p.light
 $M = 0.205 + 0.206 = 0.41$

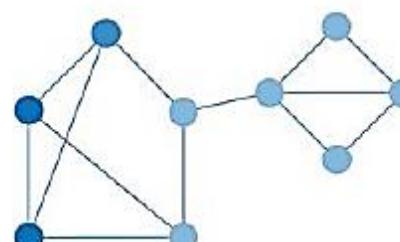
$$M = \sum_{c=1}^{n_c} \left[\frac{L_c}{L} - \left(\frac{k_c}{2L} \right)^2 \right]$$



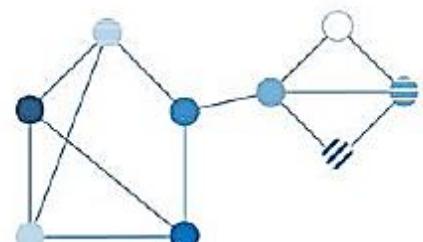
Random Baseline
(Single Community)
 $M = 0.0$



Optimal Partition
 $M = 0.41$



Suboptimal Partition
 $M = 0.22$



Negative Modularity
 $M = -0.12$

Louvain algorithm - Modularitatea Louvain

Alg. Louvain constă în aplicarea repetată a doi pași (Figura)

Pass1: Atribuire „greedy” a nodurilor către comunități, favorizând optimizările locale ale modularității.(Step 0. Alege nod start (X) **Evaluează opțini modularitate** pt. a intra în comunități cu vecinii cei mai apropiati. Step 1. Nodul de start se grupează cu nodul cu cea mai mare schimbare de modularitate. Procesul continuă: toate nodurile corespondente. Step 2. Creare super-comunități cu suma weights.)

Pass 2 Definire rețea mai grosieră bazată pe comunitățile găsite în Pass1. Această rețea cu granulație grosieră va fi utilizată în următoarea iterație a algoritmului.

Pass 1 & 2 se repetă până când nu mai sunt posibile reatribuirile ulterioare ale comunităților ce cresc modularitatea sau s-a ajuns la nr. de iterări specificate. **Formula Louvain**

m este ponderea totală a relației de-a lungul întregului graf

(**2m** valoare de normalizare comună în formule modularitate)

$A_{uv} - \frac{k_u k_v}{2m}$ este puterea relației dintre u și v

în comparație cu ce ne-am așteptă la o alocare aleatorie (tinde spre medii) a acelor noduri din rețea. — A este ponderea relației dintre u și v.

— k_u (k_v) este suma ponderilor relațiilor pt. u (v)

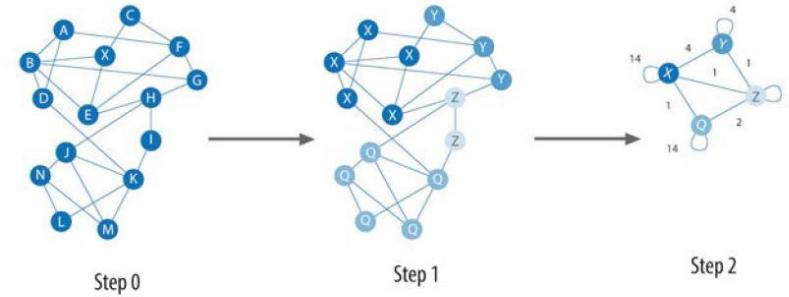
$\delta(c_u, c_v) = 1$ dacă u și v sunt alocate aceleiași comunități, și
 $= 0$ dacă acestea nu sunt.

O alta parte a acestui prim pas evaluează schimbarea modularității dacă un nod este mutat într-un alt grup.

Louvain folosește o variantă mai complicată a acestei formule și apoi determină cea mai bună repartizare a grupului.

$$Q = \frac{1}{2m} \sum_{u,v} \left[A_{uv} - \frac{k_u k_v}{2m} \right] \delta(c_u, c_v)$$

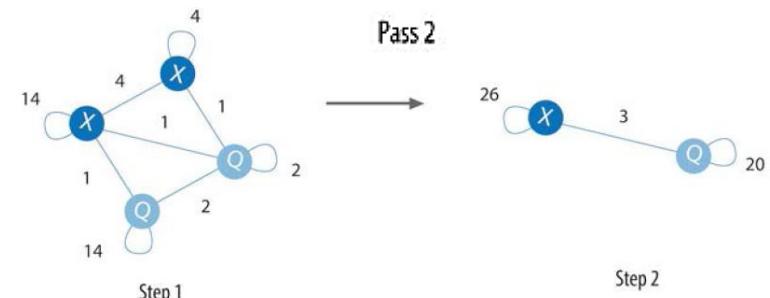
Pass 1



Choose a start node (X above) and calculate the modularity options for joining different communities with each of their immediate neighbors.

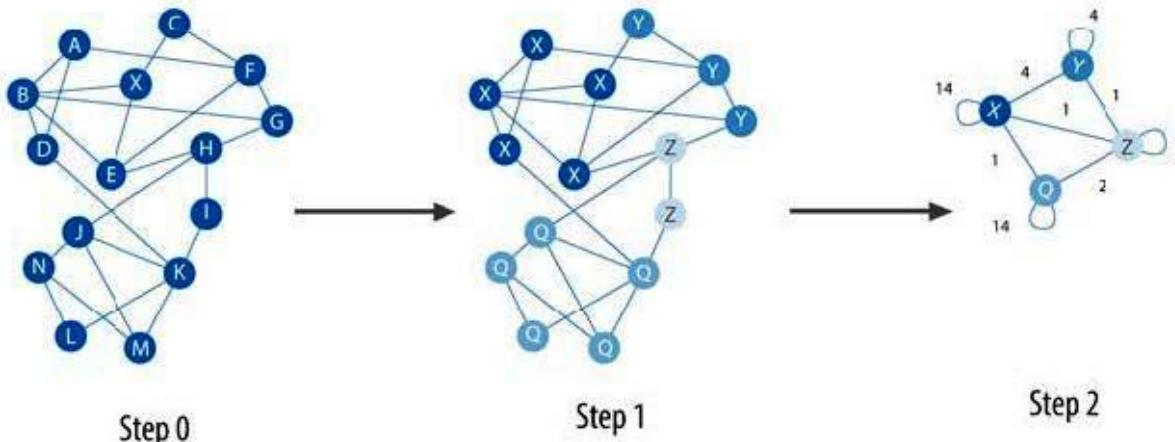
The start node joins the node with the highest modularity change. The process is repeated for each node with the above communities formed.

Communities are aggregated to create supercommunities and the relationships between these super nodes are weighted as a sum of previous links. (Self-loops represent the previous relationships in both directions now hidden in the super node.)



Steps 1 and 2 repeat in passes until there is no further increase in modularity or a set number of iterations have occurred.

Pass 1

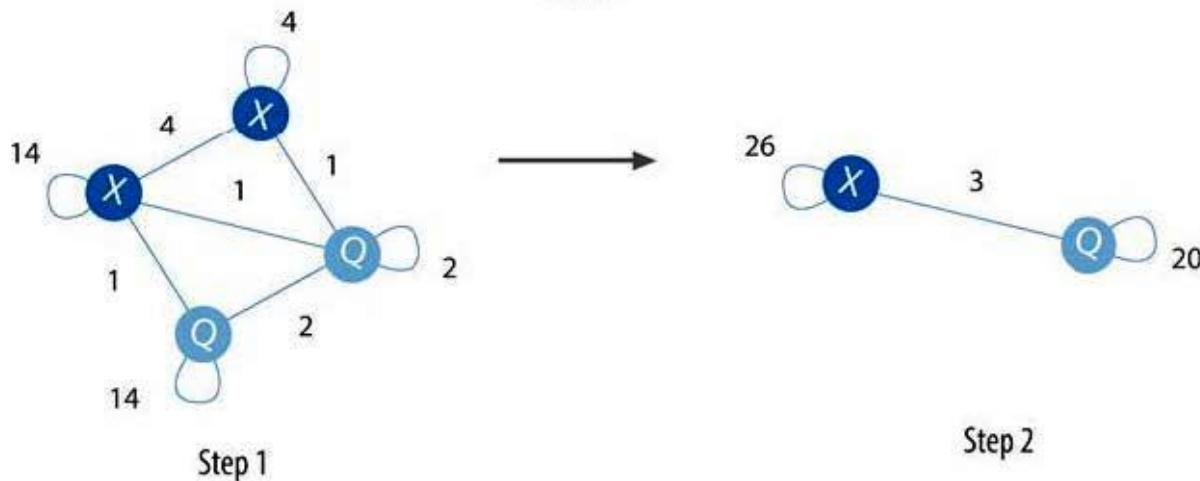


Choose a start node (X above) and calculate the modularity options for joining different communities with each of their immediate neighbors.

The start node joins the node with the highest modularity change. The process is repeated for each node with the above communities formed.

Communities are aggregated to create supercommunities and the relationships between these super nodes are weighted as a sum of previous links. (Self-loops represent the previous relationships in both directions now hidden in the super node.)

Pass 2



Steps 1 and 2 repeat in passes until there is no further increase in modularity or a set number of iterations have occurred.

Louvain algorithm

Când se utilizează algoritmul Louvain?

- Utilizați modularitatea Louvain pentru a **găsi comunități în rețele vaste**.
- Alg. Louvain aplică o modularitate **euristică**, spre deosebire de cea exactă computațional costisitoare.
- Alg. Louvain poate fi utilizat pe **grafuri mari** în care alg. standard de modularitate pot avea probleme.
- Louvain este f. util și pentru **evaluarea structurii rețelelor complexe**, la descoperirea multor niveluri de ierarhie, cum ar fi ce ați găsi într-o organizație (ex. nivel ierarhic firme, departamente, grupuri de lucru etc.)
- Algoritmul poate oferi rezultate ce pot fi **detaliate pe diferite niveluri de granularitate** și se pot găsi subcomunități în subcomunități din subcomunități.

Exemple de cazuri de utilizare (Use Case) :

- **Detectarea atacurilor cibernetice.** Algoritmul Louvain a fost utilizat în 2016 de S. V. Shanbhag privind detectarea rapidă a comunității în rețelele cibernetice la scară largă pentru aplicații de securitate cibernetică. Odata ce comunitățile au fost detectate, pot fi folosite pentru detecția atacurilor cibernetice.
- **Extragerea subiectelor de pe platformele sociale online**, ex. Twitter și YouTube, pe baza co-apariției termenilor în documente, ca parte a procesului de modelare a subiectelor.

Detalii în „Topic Modeling Based on Louvain Method in Online Social Networks” de G. S. Kido et al.

- **Găsirea structurilor comunitare ierarhice în cadrul rețelei funcționale a creierului.**

Detalii în “Hierarchical Modularity in Human Brain Functional Networks” de D. Meunier et al.

Observații - Algoritmii de optimizare a modularității, inclusiv Louvain, au două mari **probleme (issues)**

1. Algoritmii **pot ignora comunitățile mici** din rețele mari; **recomandare** - revizuirea pașilor intermediari de consolidare.
2. În grafuri mari cu comunități suprapuse, optimizatorii de modularitate pot **determina incorect maximele globale; recomandare** – utilizați algoritmii de modularitate doar ca ghid pentru estimarea brută, dar nu pentru determinare scor de acuratețe completă.

Louvain Algorithm testare

libraries	communities
pytz	[0, 0]
pyspark	[1, 1]
matplotlib	[2, 0]
spacy	[2, 0]
py4j	[1, 1]
jupyter	[3, 2]
jpy-console	[3, 2]
nbconvert	[4, 2]
ipykernel	[3, 2]
jpy-client	[4, 2]
jpy-core	[4, 2]
six	[2, 0]
pandas	[0, 0]
numpy	[2, 0]
python-dateutil	[2, 0]

Initial alg. Louvain Modularity optimizează modularitatea local pe toate nodurile, care găsește comunități mici; apoi fiecare comunitate mică este grupată într-un nod de conglomerat mai mare și Pass 1 se repetă până ajungem la un optim global.

- Coloana **communities** descrie comunitatea în care se încadrează nodurile la 2 niveluri.
- Ultima valoare din tabel este comunitatea finală, cealaltă este comunitate intermedieră
- Numerele atribuite comunităților intermediere și finale sunt etichete fără sens măsurabil (indică nodurile comunității cărora le aparțin, cum ar fi „apartine unei comunități etichetată 0”, „o comunitate etichetată 4” s.a.m.d.)

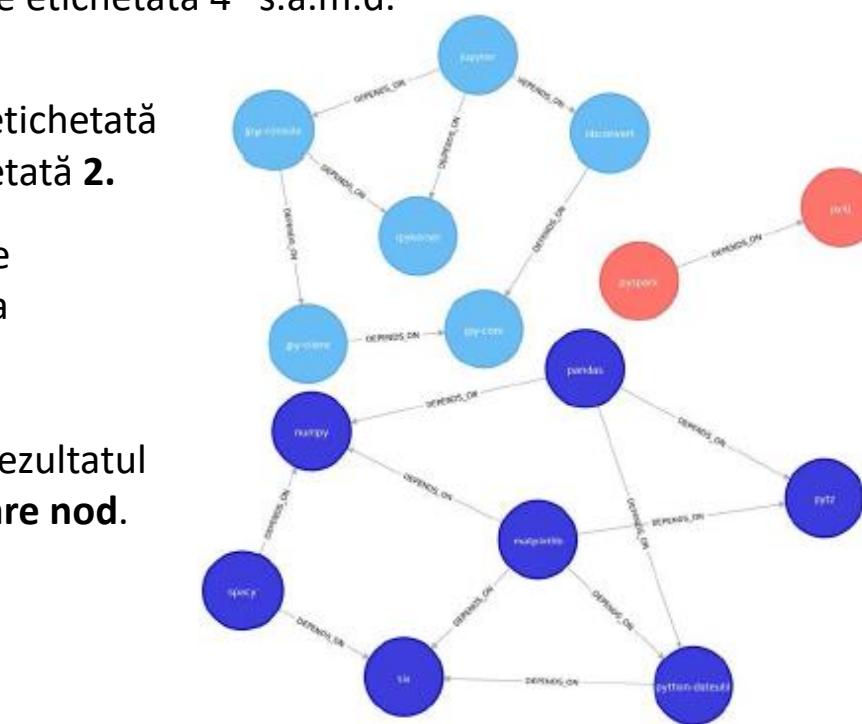
Ex. **matplotlib** are rezultat [2,0], adică comunitatea **finală** a lui **matplotlib** este etichetată **0**, iar comunitatea sa **intermediară** etichetată **2**.

Mai facilă vizualizarea dacă stocam aceste comunități folosind versiunea de scriere a algoritmului și apoi o interogăm.

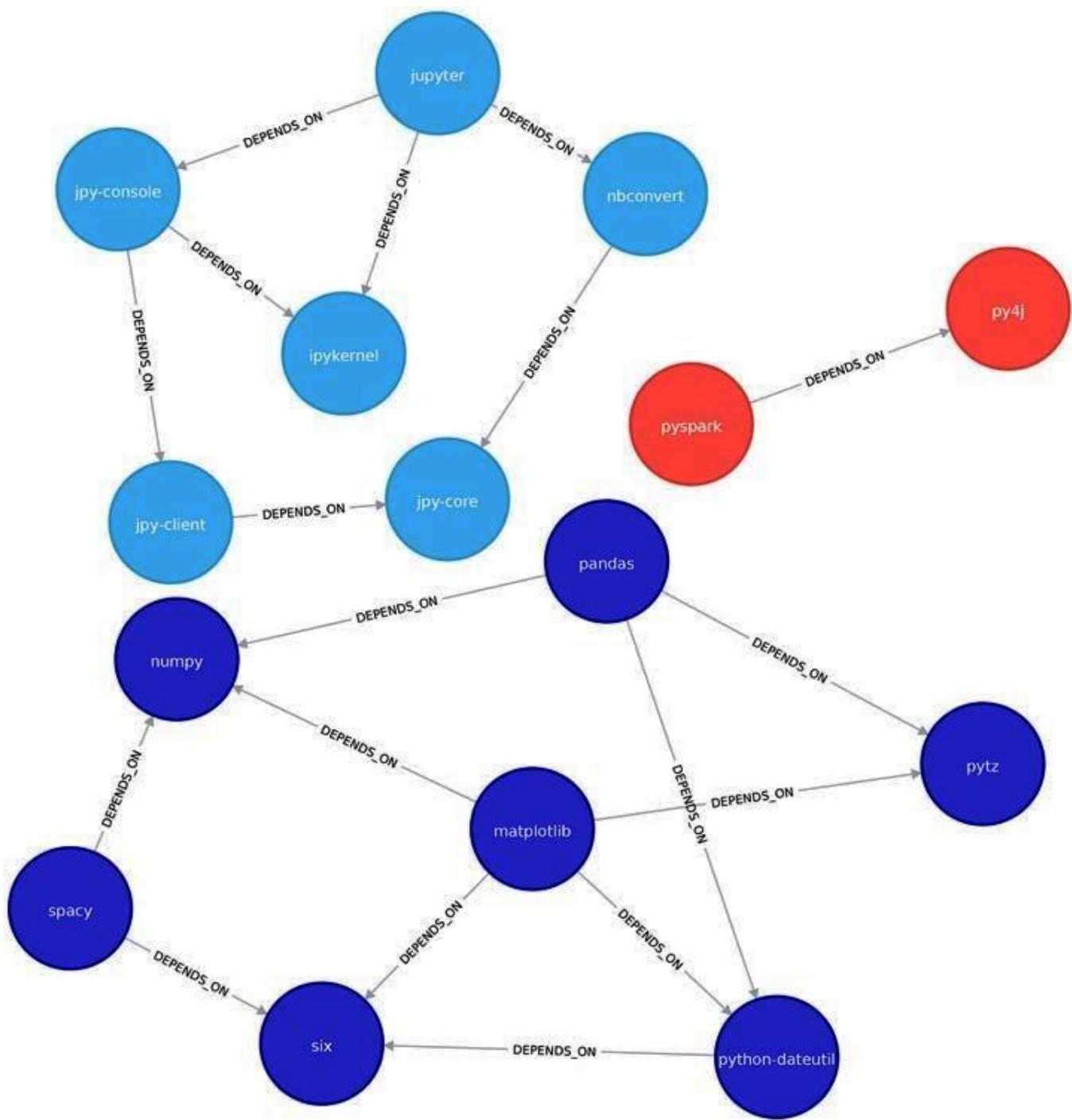
După Algoritmul Louvain se poate stoca rezultatul în **proprietațea comunităților de pe fiecare nod**.

Tabel cu clusterele finale

- Aceasta grupare este aceeași de la algoritm componentelor conectate.
- matplotlib** este într-o comunitate cu **pytz**, **spacy**, **six**, **panda**, **numpy** și **python-dateutil**
- Se poate vedea mai clar în Figura.

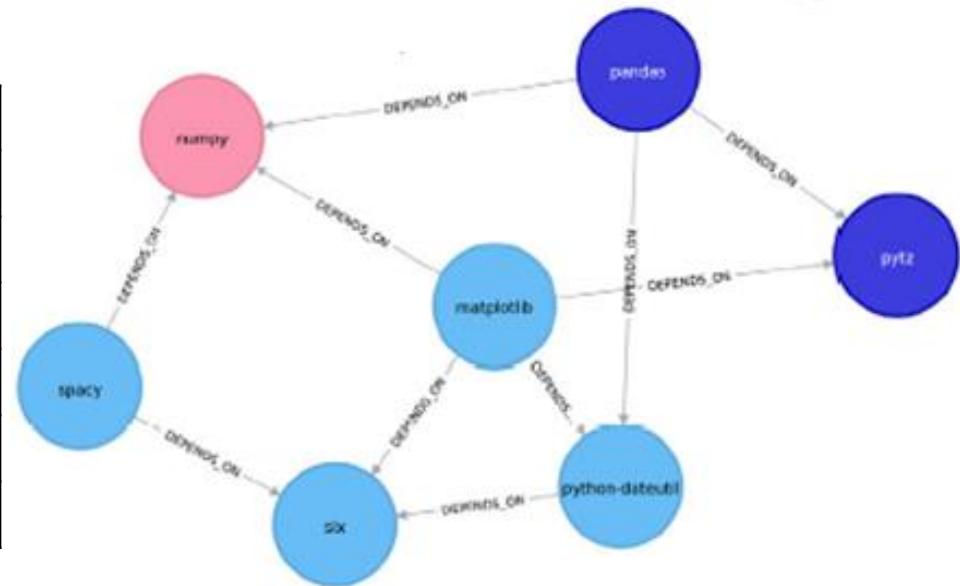


community	libraries
0	[pytz, matplotlib, spacy, six, pandas, numpy, python-dateutil]
2	[jupyter, jpy-console, nbconvert, ipykernel, jpy-client, jpy-core]
1	[pyspark, py4j]



Louvain Algorithm

community	libraries
2	[matplotlib, spacy, six, python-dateutil]
4	[nbconvert, jpy-client, jpy-core]
3	[jupyter, jpy-console, ipykernel]
1	[pyspark, py4j]
0	[pytz, panda]
5	[numpy]

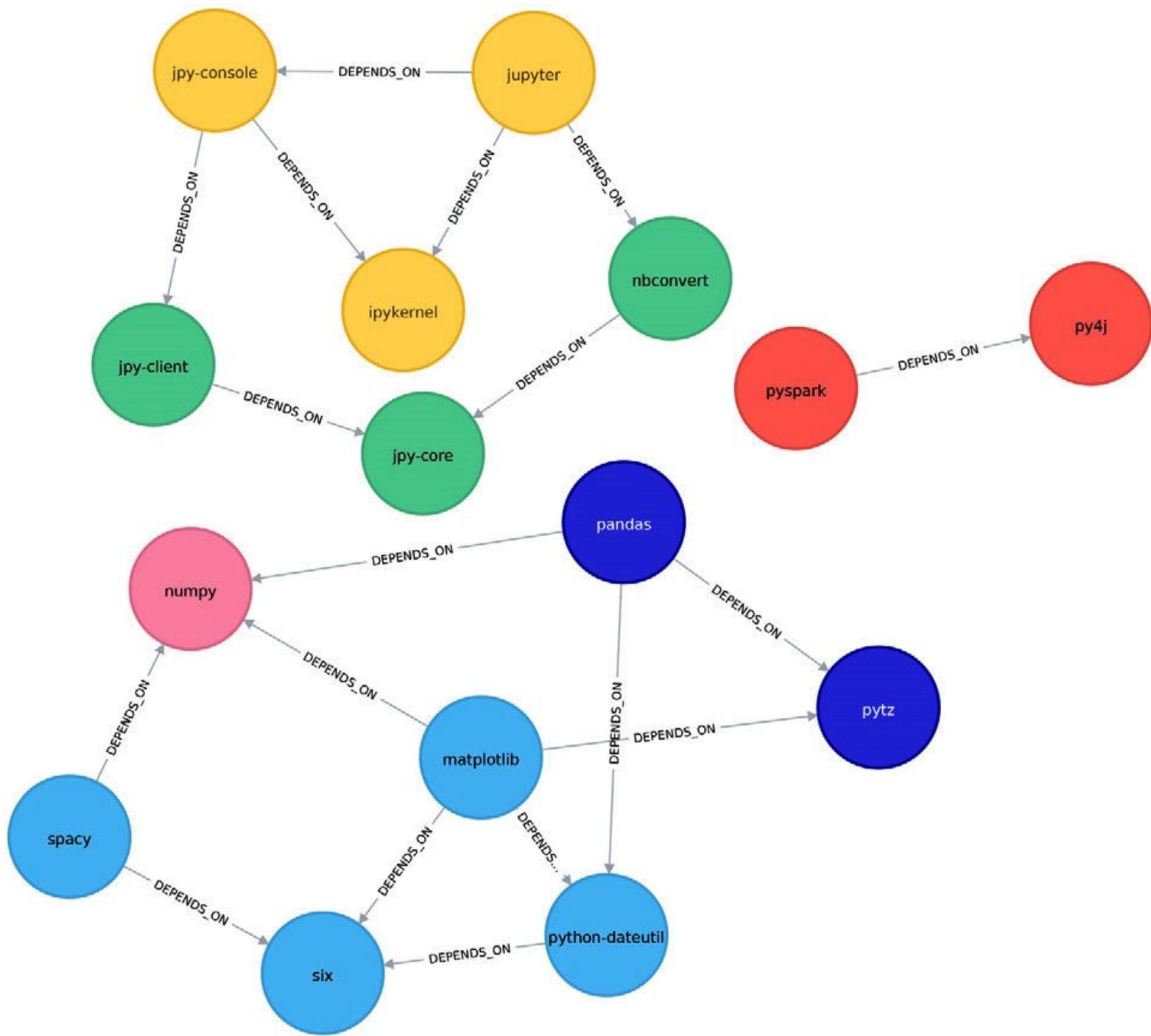


O caracteristică specială a Louvain algorithm este faptul că se vizualizează clustere intermediare. Aceasta permite vizualizarea cu o granulație mai fină a grupărilor decât a grupării finale.

Bibliotecile din comunitatea **matplotlib** s-au împărțit acum în trei comunități mai mici:

- 1) **matplotlib, spacy, six, python-dateutil** (**lightBlue**)
- 2) **pytz și panda** (**blue**)
- 3) **numpy** (**red**)

- Deși acest graf a arătat doar două straturi de ierarhie, dacă s-ar rula acest algoritm pe un graf mai mare, s-ar vedea o ierarhie mai complexă.
- Clusterele intermediare pe care le dezvaluie Louvain pot fi foarte utile pentru detectarea comunităților cu granulație fină care ar putea să nu fie detectate de alți algoritmi de detectare a comunității.



Validating Communities - Validarea comunităților

- Algoritmii de detectare a comunității au, în general, același scop: identificarea grupurilor.
- Cu toate acestea, deoarece diferiți algoritmi **încep cu ipoteze diferite**, ei pot descoperi comunități diferite.
- Acest lucru face ca **alegerea algoritmului potrivit pentru o anumita problemă să fie dificilă și necesită explorare**.
- Majoritatea algoritmilor de detectare a comunității se descurcă destul de bine **dacă densitatea relațiilor este mare** în cadrul grupurilor în comparație cu mediul înconjurător, dar rețelele din lumea reală sunt adesea mai puțin distințe.
- **Validarea acurateții comunităților** găsite se face **comparând rezultatele obținute cu un benchmark bazat pe date cu comunități cunoscute**.

State-of-art

- Exemple repere: **algoritmii Girvan-Newman (GN)** și **Lancichinetti-Fortunato-Radicchi (LFR)**.
- Rețelele de referință pe care le generează algoritmii sunt diferite:
 - GN generează o rețea aleatorie care este mai **omogenă**,
 - LFR creează un graf mai **eterogen** în care gradele nodurilor și dimensiunea comunității sunt distribuite conform unei legi de putere.
- Deoarece acuratețea testării depinde de benchmark-ul utilizat, este important **să potrivim benchmark-ul cu setul de date**. Pe cât posibil, căutați densități similare, distribuții de relații, definiții ale comunității și domenii înrudită.

Concluzii

- Algoritmii de detectare a comunității sunt utili pentru înțelegerea modului în care nodurile sunt grupate într-un graf
- S-au descris Triangle Count & Clustering Coefficient, algoritmii de numărare a triunghiului și coeficientul de grupare.

Algoritmi Determiniști

- S-au descris Strongly Connected Components & Connected Components, algoritmi determiniști de detectare a comunității:
Componente puternic conectate și Componente conectate.
Acești algoritmi au **definiții stricte** ale ceea ce constituie o comunitate
Sunt foarte utili pentru obținerea unei idei despre structura grafului la începutul procesului de analiză a grafului.

Algoritmi nedeterminiști

- Label Propagation și Louvain sunt doi algoritmi nedeterminiști capabili să detecteze **comunități mai fine**. Louvain a aratat și o ierarhie a comunităților la diverse scări.