PROJET LARGE SCALE GRAPH MINING

DÉTECTION DE COMMUNAUTÉS PAR LA MÉTHODE DE LOUVAIN

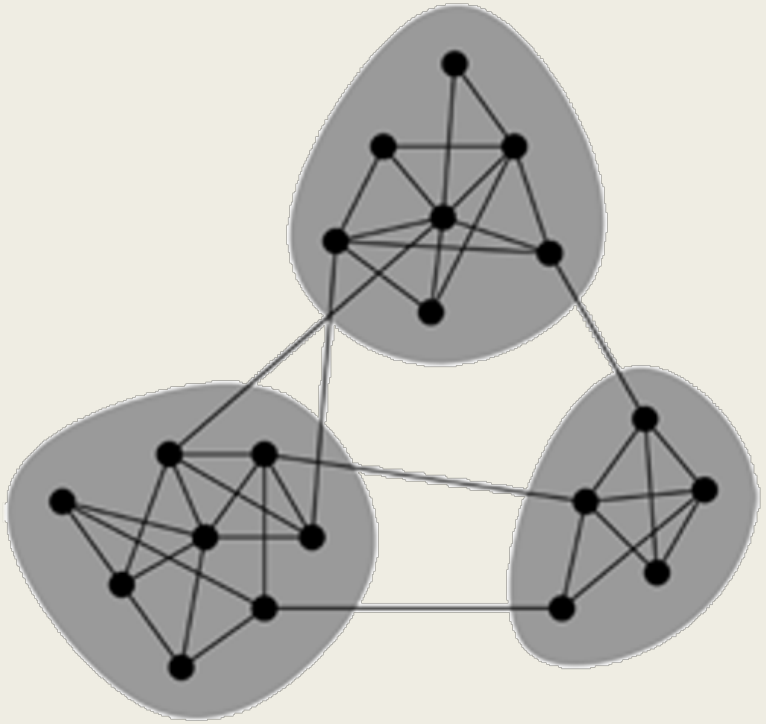
Grégoire Gambino
Jean-Baptiste Gourlet
16/04/2021

https://github.com/jbgour/Graph_Mining_Community_Detection

CentraleSupélec

Problème

- On se place dans le cadre de la théorie des graphes
- L'objectif est: étant donné un graphe *G*, on cherche extraire des communautés.
- Exemple pratique : pour un Graphe de relation sur un réseau social, on cherche à extraire les groupes de personnes en relation



Algorithme de Louvain : principe

- Objectif: Maximiser la modularité du graphe:
- $Q = \frac{1}{2m}\sum_{ij}[A_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j)$

Avec

- A_{ij} le poids de l'arête entre les nœuds *i* et *j.*
- k_i et k_j la somme des poids des arêtes liées aux nœuds *i* et *j*
- *2m* la somme de l'ensemble des poids des arrêtes du graphe
- c_i et c_j les communautés des nœuds
- δ une fonction delta : $\delta(c_i, c_j)$=*1* si *i* et *j* sont de la même communauté, 0 sinon

Algorithme de Louvain : calcul du gain en modularité

- 2 étapes pour calculer l'impact du changement de communauté d'un noeud:
  - On enlève le nœud de sa communauté initiale et on calcule le gain en modularité ΔQ_1
  - On ajoute le nœud à sa nouvelle communauté et on calcule le gain en modularité ΔQ_2. Avec par exemple $\Delta Q_2 = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$
    - Σ_{in} la somme des poids des arêtes à l'intérieur de la communauté de destination de i
    - Σ_{tot} la somme des poids de toutes les arrêtes liées aux nœuds de la communauté de destination de i
    - k_i le degré pondéré de i
    - $k_{i,in}$ la somme des poids des arrêtes entre i et les autres noeuds de la communauté de destination de i
    - 2m la somme de l'ensemble des poids des arrêtes du graphe
- Le changement est effectué si et seulement si $\boldsymbol{\Delta Q_2 - \Delta Q_1 > 0}$
- On itère sur tous les nœuds jusqu'à ce qu'aucune augmentation de modularité ne se produise.
- Une fois terminé, on construit le nouveau réseau où les nœuds sont les communautés de la phase précédente. On recommence l'algorithme et ainsi de suite jusqu'à avoir la profondeur souhaitée.

Outils utilisés

- Scala – GraphX
- objets spark.Graphx.graph
- Utilisation des vertices, edges, triplets
- Implémenation de map, filter
- Utilisation de collectNeighbors
- Implémentation de nos propres fonctions

Données utilisées

- Karate Graph: contient le réseau d'amitiés des membres d'un club de karaté d'une université américaine, graphe non orienté
- 34 sommets
- 78 arêtes

- Facebook Graph: contient un exemple de réseau d'amitiés Facebook, graphe non orienté
- 333 sommets
- 5038 arêtes

DEMO

Résultats

Karaté

- 5 itérations
- 45 secondes
- Passage à 6 communautés

Facebook

- Problème de gestion de mémoire:
  - *Retravailler le code*
  - *Utiliser un cluster*

améliorations possibles

- Paralléliser avec Pregel la deuxième itération (pour un nœud i, recherche de la meilleure communauté de destination)
- Gérer les graphes in memory, gestion du cache
- Représentations visuelles
- Itérer l'algorithme de Louvain sur les supers nœuds