Graph Clustering and Community Detection

Matteo Pinna

Seminar: Large-scale Graph Processing and Graph Partitioning

TABLE OF CONTENTS

INTRODUCTION

Importance of graphs

- Graphs are a powerful tool for modeling, analyzing and understanding relationships between entities

- Graphs are widely used in several fields
    - **Computer science** (e.g. used as storage format in graph databases);
    - **Social sciences** (e.g. used to represent social ties in social networks);
    - **Physics** (e.g. used to model connections between particles in a fluid);
    - **Biology** (e.g. used to represent interactions between cells' components);
    - etc..

Graph clustering and Community detection

- ***GOAL****:* identify clusters of vertices that are connected, according to some measure or definition of similarity, and separate them from those with which they do not correlate.
- **CHALLENGES**
  - Real-world networks can be massive;
  - Networks may be dynamic (i.e. edge deletion);
  - Communities may overlap.

- No universally accepted definition for a “good” cluster, however in general
  - *good* means that nodes inside a community are cohesive;
  - and loosely connected w.r.t. nodes outside of the community.

- Typically no distinction between the two in the literature **[1]**

Graph clustering and Community detection

- **Def. Graph Clustering:** generally involves partitioning the vertices of a graph into groups based on some measure of similarity. **[2]**
- Similarity can be interpreted in different ways w.r.t.
  - vertex attributes;
  - vertices' behaviour;
  - etc..
- **Def. Community Detection:** focuses on identifying groups of vertices densely connected between each other w.r.t. the rest of the graph, based on the graph structure. **[3]**

STATE OF THE ART

State-of-the-art: an overview

- Several community detection algorithms, based on a variety of approaches, exist.

- The communities may be
  - *overlapping*, i.e. a node can belong to more than one community;
  - *non-overlapping,* i.e. a node can belong to at most one community.

- Popular techniques include
  - Quality metric optimization (e.g. modularity, conductance);
  - Random walks;
  - Spectral clustering;
  - Seed-set expansion;
  - etc..

The streaming model

- Community detection is a challenging topic due to the huge size of most real-world networks (e.g. million of vertices, billion of edges).

- Being able to scale up to massive graphs while maintaining performance is not easy.

- *Solution:* process the graph as a stream of edges (i.e. *data stream model*)
    - Insert-only streams: only edges insertion;
    - Dynamic streams: edges insertion and/or edges deletion.

SCoDA

SCoDA

- **SCoDA [3]** is a linear time, linear space, (insert-only) stream-based algorithm for community detection.

- ***IDEA:*** if an edge *e* is randomly chosen it is more likely to connect vertices of the same community (i.e. *intra-community edge).*

- If we consider a random permutation of the edges, it is expected for intra-community edges to arrive *early*, and for inter-community edges to arrive *late*

- Adjacent nodes of an early edge will be put in the same community, adjacent nodes of a late edge will be separated.

SCoDA

- ***Def.:*** an edge arrives *early* if the current degrees of its adjacent nodes *u* and *v*, accounting for previously arrived edges only, is low.

- This is implemented with a threshold parameter *D*, an edge is considered to be early if the degrees of its adjacent nodes are below the threshold.
- Different possibilities for tuning *D*:
  - **Average degree:** average degree value over the network;
  - **Median degree:** median of the degree value over the network;
  - **Mode of the degree distribution:** most common degree in the network excluding leaf nodes (i.e. nodes with degree 1).

SCoDA: pseudo-code and complexity

n: number of nodes
m: number of edges

- **Time complexity:** O(m) (m >> n)

- **Space complexity:** O(n)

UPDATE RULE

Algorithm 1: SCoDA

Input: (List of edges E between vertices $\{1, \ldots, m\}$, parameter $D \geq 1$, and the probability p)
Output: ($c = \{c_1, \ldots, c_n\}$)

begin
    `// Initialization`
    for $i = 1, \ldots, n$ **do**
        $d_i 0$ and $c_i \leftarrow i$;
    Shuffle the list of edges E;
    `// Processing`
    for $j = 1, \ldots, |E|$ **do**
        $(u, v) \leftarrow$ j-th edge of E;
        $d_u \leftarrow d_u + 1$ and $d_v \leftarrow d_v + 1$;
        if $d_u \leq D$ *and* $d_v \leq D$ **then**
            if $d_u < d_v$ **then**
                $c_u \leftarrow c_v$;
            else if $d_v < d_u$ **then**
                $c_v \leftarrow c_u$
            else
                $choice \leftarrow rand(0, 1)$ **if** $choice \geq p$ **then**
                    $c_u \leftarrow c_v$;
                else
                    $c_v \leftarrow c_u$;

CoEuS

CoEuS

- **CoEuS [4]** is a (insert-only) stream-based algorithm based on seed-set expansion.

- *Seed-set expansion:* communities are expanded based on their corresponding initialized seed-set (i.e. set of node ids)

- Seed-sets initialization is crucial for the algorithm’s outcome.

- No restrictions on the order in which edges arrive (i.e. shuffling is not needed)

CoEuS: community participation

- Communities are periodically pruned each time *W* (window size) elements have been processed.

- The pruning aims at filtering out irrelevant nodes from communities and is based on community participation scores.

- *Def.:* the *community participation* value of a node *u* in a community *C* can be define as

$$cp(u) = \frac{|\{(u,v) \in E : v \in C\}|}{|\{(u,v) \in E\}|}$$

i.e., ratio between the number of its adjacent nodes that are part of the community, and the number of linked nodes in the rest of the graph.

- Nodes with which community paticipation scores are highly relevant in the given community.

CoEuS: pseudo-code and complexity

n: number of nodes
m: number of edges
c: number of ground-truth communities

- **Time complexity:** O(m*c)

- **Space complexity:** O(n*(c + 1))

UPDATE RULE

Algorithm 2: CoEuS(S, K')

Input: (A graph stream S, and a set of community seed-sets K')
Output: (A set of communities C')

```
begin
    // Initialization (seed-sets)
    foreach K ∈ K' do
        C ← {};
        foreach k ∈ K do
            C[k] = 1;
        C'.put(C);
    // Processing
    while ∃(u, v) ∈ S do
        degree_V[u] += 1];
        degree_V[v] += 1];
        foreach C ∈ C' do
            if u ∈ C then
                degree_C[v] += 1;
                C.put(v);
            if v ∈ C then
                degree_C[u] += 1;
                C.put(u);
        processedElements += 1;
        // Pruning
        if (processedElement % W) == 0 then
            foreach C ∈ C' do
                C ← pruneComm(C, s, degree_V, degree_C;
```

CoEuS: *pruneComm*

Algorithm 3: pruneComm

Function pruneComm$(C, s, degree_V, degree_C)$:
- $minheap \leftarrow []$;
- **foreach** $c \in C$ **do**
  - $cp(c) = \frac{degree_C[c]}{degree_V[c]}$;
  - **if** $minheap.size() < s$ **then**
    - $minheap.push(c, cp(c))$;
  - **else if** $cp(c) > minheap[0]$ **then**
    - $minheap.pop()$;
    - $minheap.push(c, cp(c))$;
- **return** $set(minheap)$;

- Min-heap used to prune communities based on their community scores

- *s* is a threshold that represent the community size a community should have

CoEuS: *edgeQuality* optimization

Algorithm 4: CoEuSedgeQuality(S, K')

Input: (A graph stream S, and a set of community seed-sets K')

Output: (A set of communities C')

begin

// Seed-sets initialization

foreach $K \in K'$ **do**

$C \leftarrow \{\}$;

foreach $k \in K$ **do**

$C[k] = 1$;

$C'.put(C)$;

// Stream processing

while $\exists(u, v) \in S$ **do**

$degree_V[u] += 1]$;

$degree_V[v] += 1]$;

// Edge quality optimization

$addToCommByEdgeQuality()$; EQ UPDATE RULE

$processedElements += 1$;

// Pruning

if $(processedElement \,\%\, W) == 0$ **then**

foreach $C \in C'$ **do**

$C \leftarrow pruneComm(C, s, degree_V, degree_C$;

Algorithm 5: addToCommByEdgeQuality

Procedure `addToCommByEdgeQuality():`

foreach $u \in C$ **do**

if $u \in C$ **then**

$degree_C[v] += \frac{degree_C[u]}{degree_V[u]}$;

$C.put(v)$;

if $v \in C$ **then**

$degree_C[u] += \frac{degree_C[v]}{degree_V[v]}$;

$C.put(u)$;

IMPLEMENTATION

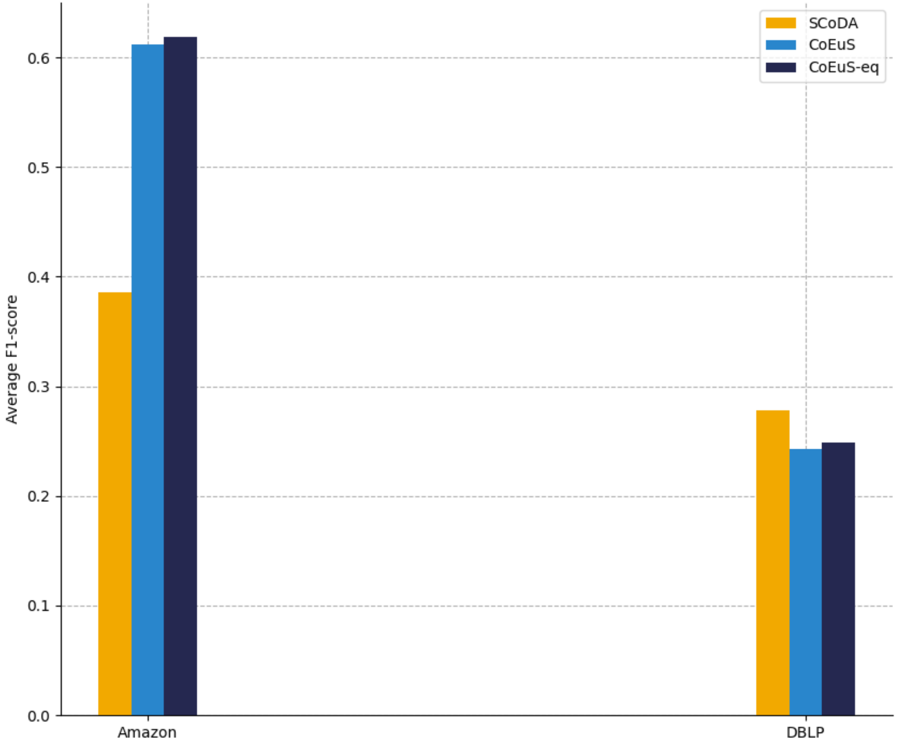
Implementation choices

- Implementation of *SCoDA* and *CoEuS* in Java.

- Stream processing through Java input/output streams (i.e. *BufferedReader* and *BufferedWriter*).

- *SCoDA* random permutation with *Fisher Yates Shuffling* algorithm.

- *CoEuS* seed-sets initialized with <MAX_SEEDS> ramdom nodes from ground-truth communities.



DATASETS:
Datasets from *Stanford Large Network Dataset Collection* [5].
•*Amazon*
•*DBLP*

- Top 5000 ground-truth communities with at least three (> 3)

github.com/NennoMP/community-detection

CONCLUSIONS

Conclusions

- Community detection is a challenging research topic due to massive real-world networks.

- The streaming model is a viable approach to solve scalability issues in large-scale networks.

- SCoDA requires a random permutation of the edges list before processing the stream.
- SCoDA is not able to execute in an online-fashion, as the shuffling requires the entire graph to be available at once.

- CoEuS space complexity is non-trivial for massive networks with a high number of nodes and communities.
- CoEuS does not require a pre-processing shuffling step, and would be able execute in an online-fashion.

BIBLIOGRAPHY

[1] *Santo Fortunato. 2010. Community detection in graphs. Physics reports.*

[2] *Isa Inuwa-Dutse, Mark Liptrott, and Ioannis Korkontzelos. 2021. A multilevel clustering technique for community detection. Neurocomputing.*

[3] *Alexandre Hollocou, Julien Maudet, Thomas Bonald, and Marc Lelarge. 2017. A linear streaming algorithm for community detection in very large networks. CoRR (2017).*

[4] *Panagiotis Liakos, Alexandros Ntoulas and Alex Delis. 2017. COEUS:community detection via seed-set expansion on graph streams. In 2017 IEEE International Conference on Big Data (Big Data).*

[5] *Jure Leskovec and Andrej Krevl. 2014. Snap Datasets: Standford Large Network Dataset Collection.*

Thanks!

ANY QUESTIONS?

Average F1-score

Given a set of detected communities $C' = \{c_1, \dots, c_K\}$;

Given a set of ground-truth communities $C = \{c_1, \dots, c_l\}$;

Precision(c', c) = $\frac{|c' \cap c|}{|c'|}$ **Recall(c', c)** = $\frac{|c' \cap c|}{|c|}$

F1(c', c) = $\frac{Precision(c', c) * Recall(c', c)}{Precision(c', c) + Recall(c', c)}$

F1(C', C) = $\frac{1}{K} \sum_{k=1}^{K} max_{1 \le l \le L}\ F1(C'_l, C_k)$

F1-avg(C, C') = $\frac{F1(C', C) + F1(C, C')}{2}$