Progetto Scalable and Cloud Programming Community Detection

Giulia Giusti

Matteo Trentin

Dipartimento di Informatica
Università di Bologna

Obiettivi del progetto

- Implementazione di diverse versioni del Label Propagation Algorithm per la community detection
- Implementazione di diverse metriche utili per la valutazione delle community individuate dai differenti algoritmi
- Confronto tra gli algoritmi implementati e l'implementazione di LPA presente in libreria

Community Detection

- L'obiettivo di un algoritmo di community detection è quello di trovare gruppi di nodi di interesse all'interno di una rete, chiamati community.

- **Tipologie**:
  - Non-overlapping community detection:
    Ad ogni nodo del grafo viene attribuita una sola label rappresentante la community di appartenenza.
    $\rightarrow$ LPA e DLPA
  - Overlapping community detection:
    Ad ogni nodo del grafo viene attribuita una lista di label rappresentante le diverse community di appartenenza.
    $\rightarrow$ SLPA

Schema Label Propagation

Input: G(V,E)

Fase pre-processing

for $v \in V$ do
    label(v) $\leftarrow$ v

for (i $\leftarrow$ 0, i < maxStep, i++)
    for $v \in V$ do
        $\Gamma(v) \leftarrow$ neighbors di v
        labels = List[label(u)|u $\in \Gamma(V)$]
        Selezione della nuova label di v
        mediante la regola di aggiornamento

Output: G((V,Label),E)

Dataset e load del grafo

- **Dataset**
  - Link: https://www.kaggle.com/rozemberczki/musae-github-social-network
  - Tipologia rete: Grafo diretto
- **Load del grafo**
  - Componente GraphX di Spark

Struttura del progetto

1. Algoritmi implementati
   - Label Propagation Algorithm
   - Label Propagation Algorithm con Pregel
   - Directed Label Propagation Algorithm
   - Speaker-Listener Label Propagation Algorithm
   - Fase pre-processing: semplificazione grafo con SNN
2. Metriche di confronto
   - Density
   - Modularity for directed graphs
   - Separability
3. Test degli algoritmi e confronto
   - Confronto algoritmi
   - Risultati SLPA

Label Propagation Algorithm (LPA)

- Ad ogni nodo viene inizialmente assegnata una label che ne indica la community
- Dato un nodo con $id = x$, la sua label inizialmente è pari a x
- Ad ogni step dell'algoritmo viene effettuata una `map` per aggiornare le label dei nodi del grafo
- Ad ogni nodo viene assegnata la community a cui appartiene la maggioranza dei suoi nodi adiacenti
- In caso di parità, ne viene scelta una casuale tra quelle con numero massimo di occorrenze

- **LPA Shuffle**: versione alternativa di LPA nella quale l'estrazione dei nodi durante la fase di Label Propagation viene effettuata in modo casuale.

Label Propagation Algorithm con Pregel

- Pregel è un framework per l'elaborazione di grafi creato da Google per risolvere problemi difficili per i quali il framework MapReduce non è particolarmente performante.

- Pregel richiede l'implementazione dei seguenti metodi:
  - sendMessage: utilizzato da un nodo per informare i vicini della propria etichetta corrente
  - mergeMessage: utilizzato per fare il merge dei messaggi ricevuti da un nodo
  - vertexProgram: utilizzato per la selezione della nuova label del nodo

  e i seguenti parametri:
  - graph
  - initialMsg
  - maxIterations

Directed Label Propagation Algorithm (DLPA)

- Versione di LPA specifica per i grafi diretti
- Migliora la regola di aggiornamento associando un peso ad ogni vicino $\rightarrow$ *Preferential label frequency computation*
- Selezione della nuova label l_i^{new} per il nodo i:

$$l_i^{new} = argmax\ l \begin{cases} \sum_{j \in N^{l}_{(i^{in})}} [1 - \frac{k_i^{in} k_j^{out}}{k_i k_j}] \\ \sum_{j \in N^{l}_{(i^{out})}} [1 - \frac{k_i^{out} k_j^{in}}{k_i k_j}] \end{cases}$$

dove:

- $N^{l}_{(i^{EdgeDirection})}$: nodi vicini di i con label l per cui esiste un arco orientato rispetto a *EdgeDirection* tra i e il vicino
- k_j^{in}: inDegree del nodo j
- k_j^{out}: outDegree del nodo j
- k_i: degree del nodo j

Speaker-Listener Label Propagation Algorithm (SLPA)

Input: G(V,E)

Fase pre-processing

for $v \in V$ do
    label(v)$\leftarrow$ v
for (i $\leftarrow$ 0, i < maxStep, i++)
 for $v \in V$ do
   1) Ogni vicino del nodo v invia a tale nodo una singola label secondo la speaking rule
   2) Selezione della nuova label di v dalla lista di label ottenuta dai vicini secondo la listening rule

Fase post-processing

Output: G((V,List[Label]),E)

Speaker-Listener Label Propagation Algorithm (SLPA)

SLPA è un algoritmo di overlapping community detection.

- **Speaking rule**: selezione casuale pesata di una label tra tutte le label della memoria del nodo speaker, la probabilità associata ad ogni label viene calcolata in base alla sua frequenza.
- **Listening rule**: selezione casuale tra le label più popolari dell'insieme che *v* ottenuto dai vicini.
- **Fase post-processing**:
  - Conversione della memoria di ogni nodo in una distribuzione di probabilità basata sulla frequenza di ogni label.
  - Eliminazione delle label con probabilità minore della soglia `r`.

Semplificazione grafo con SNN

- Semplificazione basata sull'assegnazione di un peso agli archi con approccio Shared Nearest Neighbors
- Ad ogni arco (i, j) si assegna peso pari al numero di nodi adiacenti sia ad i che a j
- Il grafo viene poi semplificato rimuovendo tutti gli archi di peso nullo
- Così facendo vengono eliminati gli archi singoli tra vari nodi, lasciando intatte le connessioni più forti
- Successivamente vengono eliminati i nodi isolati prodotti (che rappresenterebbero community singole)

Struttura del progetto

1. Algoritmi implementati
   - Label Propagation Algorithm
   - Label Propagation Algorithm con Pregel
   - Directed Label Propagation Algorithm
   - Speaker-Listener Label Propagation Algorithm
   - Fase pre-processing: semplificazione grafo con SNN
2. Metriche di confronto
   - Density
   - Modularity for directed graphs
   - Separability
3. Test degli algoritmi e confronto
   - Confronto algoritmi
   - Risultati SLPA

Density

- Metrica basata sull'intuizione secondo la quale una buona community sia ben connessa, cioè esistono molti archi che connettono i nodi appartenenti alla community.

- Tale metrica applica la seguente funzione ad ogni community ω rilevata dall'algoritmo:

$$d(\omega) = \frac{2\ |E_{\omega}^{in}|}{|\omega|\ (|\omega|-1)}$$

dove:

  - $|E_{\omega}^{in}|$: numero di archi tra i nodi all'interno della community ω.
  - $|\omega|$: dimensione della community ω.

Modularity for directed graphs

- Metrica utilizzata per calcolare la robustezza della suddivisione del grafo in community.

- I grafi con elevata modularità hanno molti archi tra i nodi all'interno delle community ma pochi archi tra nodi di community diverse.

- Tale metrica calcola la seguente funzione:

$$m = \frac{1}{m} \sum_{i,j} [A_{i,j} - \frac{d(i)^{out}\ d(i)^{in}}{m}]\ \ \delta_{\omega_i,\omega_j}$$

  dove:
  - $d(i)^{in}$ e $d(i)^{out}$: in-degree e out-degree del nodo i.
  - ω_i: community del nodo i.
  - $A_{i,j}$: 1 se l'arco tra i e j esiste, 0 altrimenti.
  - m: numero di archi
  - δ: Kronecker delta function che restituisce 1 sse ω_i=ω_j.

Separability

- Metrica basata sull'intuizione secondo la quale una buona community sia ben separata dal resto della rete, il che significa che ha relativamente pochi archi che partono da un nodo della community e raggiungono un nodo esterno ad essa.

- Tale metrica applica la seguente funzione ad ogni community ω rilevata dall'algoritmo:

$$s(\omega) = \frac{|E_\omega^{in}|}{|E_\omega^{out}|}$$

dove:

- $|E_\omega^{in}|$: numero di archi tra i nodi all'interno della community ω.
- $|E_\omega^{out}|$: numero di archi che da un nodo interno a ω puntano ad un nodo esterno ad essa.

Struttura del progetto

1. Algoritmi implementati
   - Label Propagation Algorithm
   - Label Propagation Algorithm con Pregel
   - Directed Label Propagation Algorithm
   - Speaker-Listener Label Propagation Algorithm
   - Fase pre-processing: semplificazione grafo con SNN
2. Metriche di confronto
   - Density
   - Modularity for directed graphs
   - Separability
3. Test degli algoritmi e confronto
   - Confronto algoritmi
   - Risultati SLPA

Test degli algoritmi e confronto

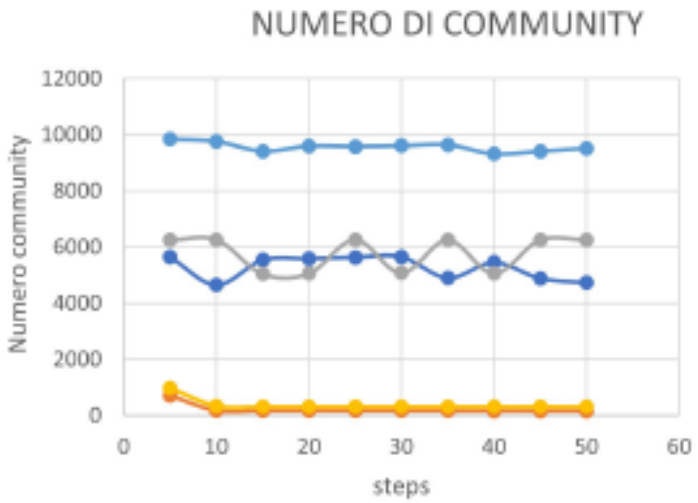
- Gli algoritmi di Non-Overlapping Community Detection sono stati confrontati in base alle seguenti caratteristiche:
  - Tempo di esecuzione
  - Numero di community individuate
  - Media modularity
  - Media separability
  - Media density

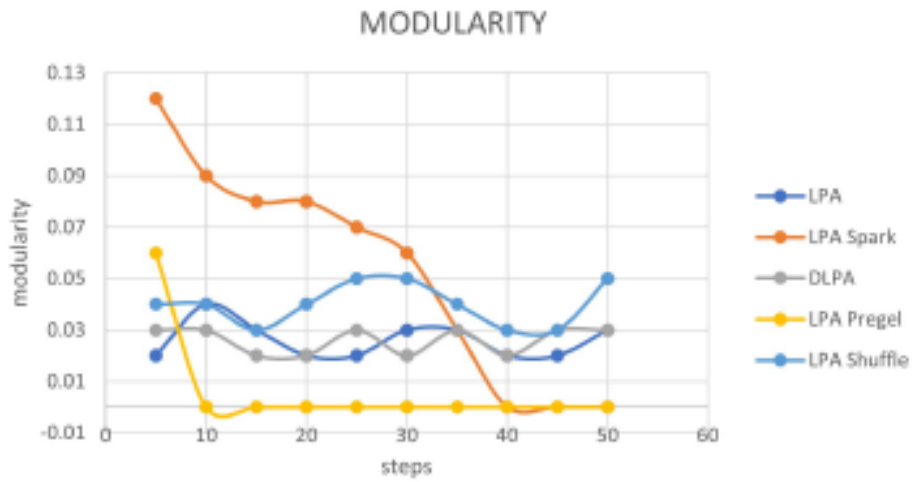
  al variare del numero di step.

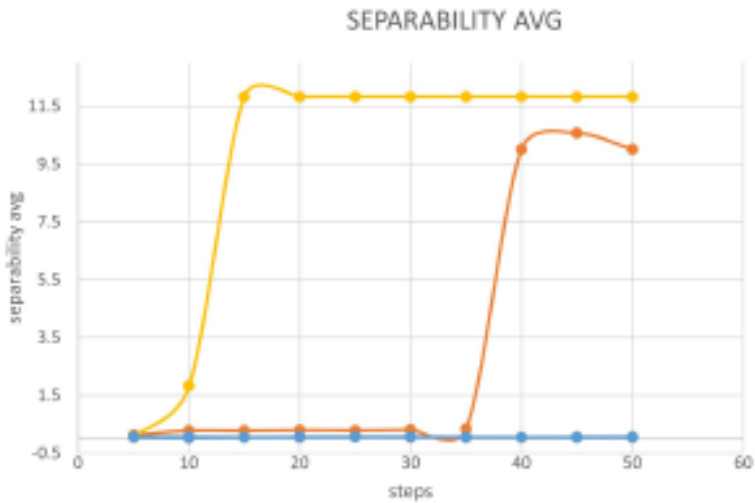
- L'algoritmo SLPA di Overlapping Community Detection è stato testato su diversi valori di threshold r al variare del numero di step.

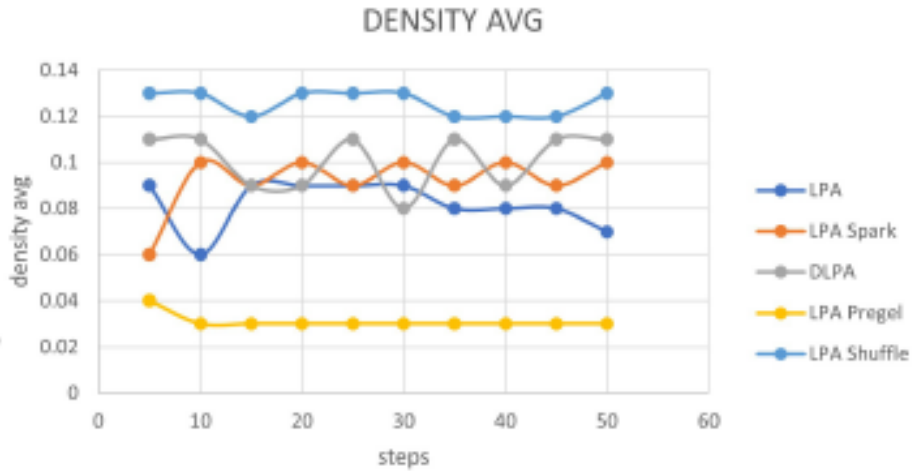
- Tali test sono stati effettuati sia sul grafo di base sia sul grafo semplificato.

Confronto algoritmi-Grafo di base

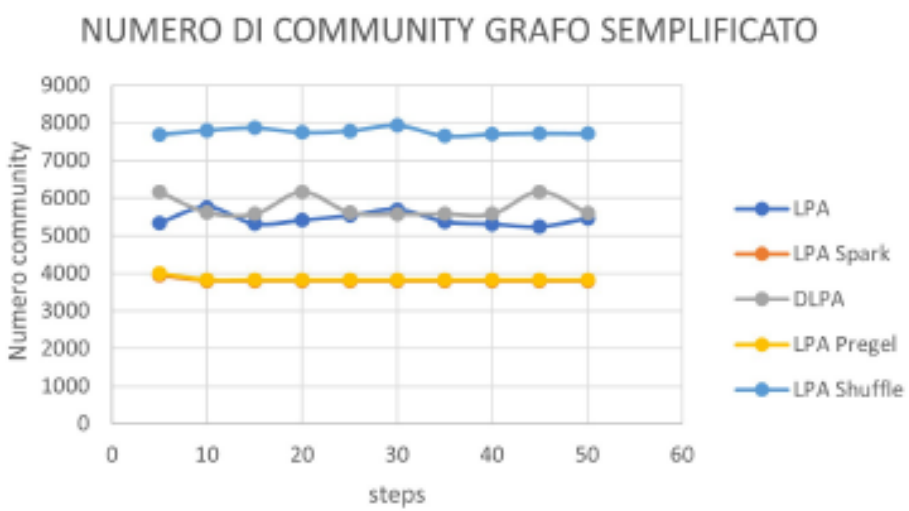


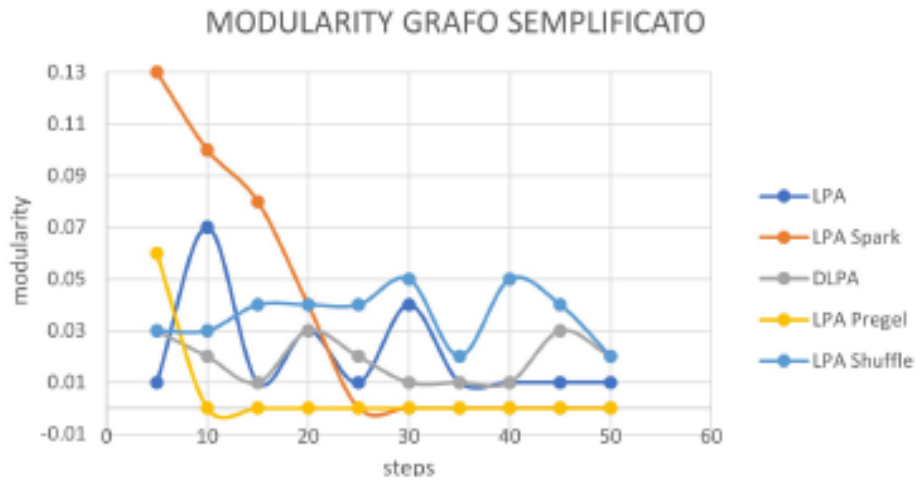


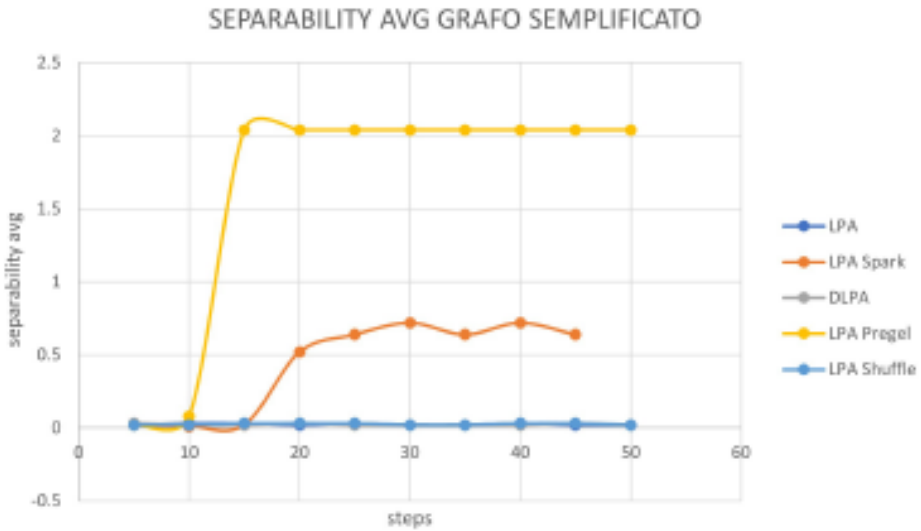


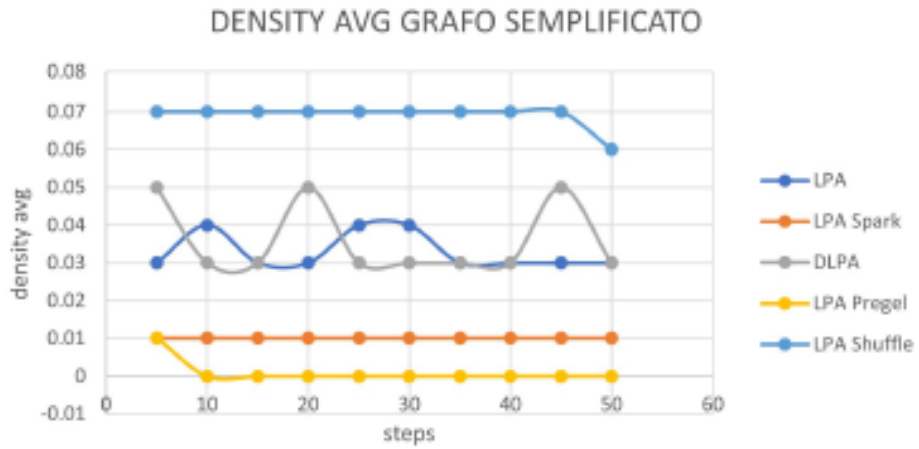


Confronto algoritmi-Grafo semplificato

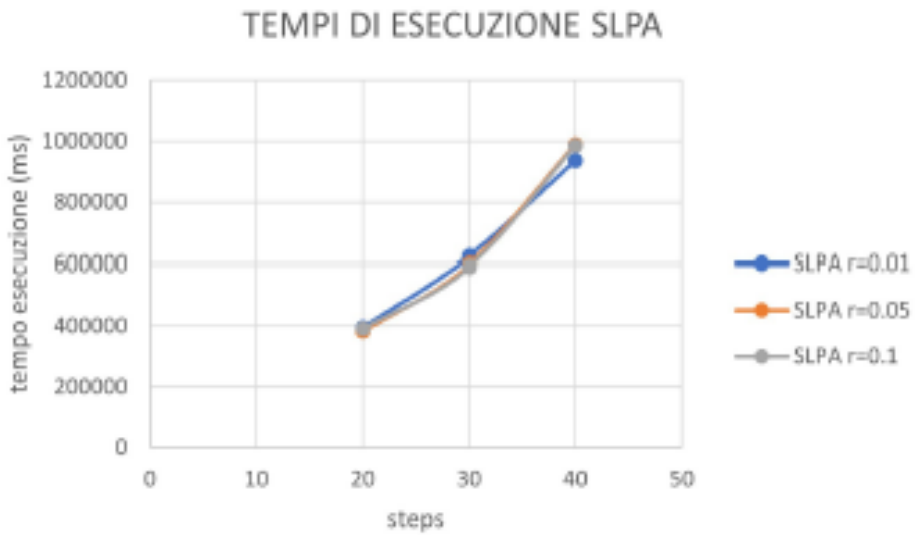








Risultati SLPA



Conclusioni

- L'implementazione che porta ai risultati migliori è, prevedibilmente, quella di libreria
- Le implementazioni alternative hanno mostrato risultati meno soddisfacenti in termini di community e metriche, seppur in tempi sensibilmente inferiori
- La semplificazione basata su SNN non ha mostrato particolari vantaggi, se non nei tempi di esecuzione degli algoritmi non basati su Pregel

Sviluppi futuri

- Miglioramento dei risultati ottenuti dagli algoritmi basati su Map
- Esplorazione di diverse strategie di selezione e di aggiornamento delle label
- Calcolo di più metriche, anche per Overlapping Community Detection
- Ottimizzazione in termini di tempo del calcolo delle metriche
- Test e confronto su più dataset

Riferimenti

- **Community Detection e LPA**
  - S. E. Garza and S. E. Schaeffer, Community detection with the Label Propagation Algorithm: A survey, 2019
  - U.N. Raghavan, R. Albert and S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, 2007
- **DLPA**
  - X. Li, Directed LPA: Propagating labels in directed networks, 2019.
- **SLPA**
  - J. Xie, B. K. Szymanski, X. Liu, SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process, 2011.

Riferimenti

- **Metriche**
  - T.Chakraborty, A. Dalmia, A. Mukherjee and N. Ganguly, Metrics for Community Analysis: A Survey, 2016
- **GraphX**
  - R. S. Xin et al., GraphX: A resilient distributed graph system on Spark, 2013.
  - Documentazione GraphX di Spark
- **Pregel**
  - Documentazione Pregel di GraphX.
  - GraphX - Pregel internals and tips for distributed graph processing.
  - R. Ramamonjison, Apache Spark Graph Processing, 2015.
  - Slide del corso *Distributed Algorithms and Optimization* della Stanford University