A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

Tesina di progetto

Signal Processing and Optimization for Big Data

Corso di Laurea Magistrale in Ingegneria Informatica e Robotica
Curriculum Data Science
A.A. 2024-2025
Dipartimento di Ingegneria

docente
Prof. Paolo Banelli

Implementazione algoritmo per il clustering spettrale di un grafo

studente

363551 **Andrea Tomassoni** andrea.tomassoni@studenti.unipg.it

Data ultimo aggiornamento: 21 maggio 2025

0. Indice

1. Introduzione

Lo scopo di questa tesina è l'implementazione di un'algoritmo per il clustering spettrale di un grafo. L'idea di fondo è quella di trovare uno strumento che tramite la clusterizzazione del grafo permetta di analizzare una rete internet per andare ad identificare eventuali porzioni della rete poco connesse.
L'algoritmo implementato è composto da due parti: nella prima parte si va ad identificare il numero di cluster ottimale per il grafo analizzato tramite un euristica (Eigengap Heuristic). Mentre in una seconda fase si sfrutta il dato precedente ottenuto per andare a clusterizzare il grafo.
L' algoritmo è stato implementato in Matlab ed è stato testato sia su dati sintetici, generati tramite un apposito script, sia su un grafo che rappresenta la rete del Consorzio GARR.

2. Analisi Teorica del problema

2.1 Eigengap Heuristic

Questa euristica viene utilizzare per calcolare il numero ottimale di cluster in cui dividere il grafo. Si basa sull'idea che i primi k autovalori del Laplaciano di un grafo siano piccoli, mentre al *k+1* esimo autovalore si verifichi un salto.
Nello specifico, il procedimento è il seguente:

- calcolare e ordinare gli autovalori del Laplaciano

$$\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_n$$

- calcolare l'ampiezza di tutti i *gap* tra gli autovalori.

$$gap_i = \lambda_{i+1} - \lambda_i \tag{2.1}$$

- trovare il *gap* massimo

$$k = \arg\max_i(gap_i) \tag{2.2}$$

2.2 Spectral Graph Clustering

L'obiettivo di questo procedimento è quello di dividere il grafo in k cluster tramite minimizzazione del problema di ottimizzazione k-cut.

2.2.1 k-Cut

Il problema è formalmente definito come segue.
Posta la dimensione totale del taglio (*total cut size*) come

$$C\left(A_1, A_2, .., A_k\right) = \frac{1}{2}\sum_{i=1}^{k} W\left(A, \bar{A}_i\right) \tag{2.3}$$

si vuole andare a minimizzare la dimensione del taglio, ovvero

$$\min_{A_1,\ldots,A_k} C\left(A_1, \ldots, A_k\right) \tag{2.4}$$

2.2.2 RatioCut

Per prevenire soluzioni che vanno ad associare ad un cluster un singolo nodo, è possibile introdurre il *RatioCut* R, il quale va a penalizzare i cluster a bassa cardinalità.

$$R\left(A_1, \ldots, A_k\right) = \frac{1}{2} \sum_{i=1}^{k} \frac{W\left(A_i, \bar{A}_i\right)}{|A_i|} \tag{2.5}$$

Nota
Il problema di minimizzazione di R è di tipo *NP-Hard*, ma si può ricorrere ad una versione rilassata del problema ponendo:

$$R\left(A_1, \ldots, A_k\right) = \frac{1}{2} \sum_{j=1}^{k} \underline{h}_j^{\top} L \underline{h}_j^{\top} \tag{2.6}$$

con

$$\left(h_j\right)_i = \begin{cases} \frac{1}{|A_j|^{1/2}}, \text{ if } v_i \in A_j \\ 0, \text{ otherwise} \end{cases} \tag{2.7}$$

$$i = 1, \ldots, n, \quad j = 1, \ldots, k$$

Il problema precedente si può riscrivere come

$$\min_{\{A_1, \ldots, A_k\}} R\left(A_1, \ldots, A_k\right) = \frac{1}{2} \text{ trace } \left(H^{\top} L H\right) \tag{2.8}$$

con

$$H = [\underline{h}_1, \ldots, \underline{h}_k]$$

Il problema rilassato da risolvere diventa:

$$\arg\min_{H=\{h_1, \cdots, h_n\}} \quad \frac{1}{2} \text{ trace } \left\{H^{\top} L H\right\} = \frac{1}{2} \sum_{i=1}^{k} h_i^{\top} L h_i \tag{2.9}$$

$$\text{s.t. } H^T H = I$$

Tramite il teorema di *Rayleigh-Ritz* è possibile dimostrare che la soluzione di questo problema sono i primi k autovettori del Laplaciano L.

La soluzione ottenuto va ora riportata ad una versione intera, in modo da assegnare ogni nodo ad un singolo cluster. Questo è possibile tramite l'utilizzo dell'algoritmo *k-means* sulle componenti spettrali. Nello specifico, posto:

$$\tilde{H} = [\underline{u}_1, \underline{u}_2, \ldots . \underline{u}_k] = \begin{bmatrix} \underline{y}_1^{\top} \\ \vdots \\ \underline{y}_n^{\top} \end{bmatrix} \tag{2.10}$$

andando ad eseguire l'algoritmo *k-means* sui vettori $\{\underline{y}_i\}$ vengono identificati i centrodi (cluster) alla distanza minima.

2.2.3 NormalizedCut

Nel caso di grafi non regolari, in cui il grado di alcuni nodi può differire molto da quello degli altri, è preferibile utilizzare il *NormalizedCut*, una versione modificata del *RatioCut* in cui si va normalizzare tenendo conto non solo dei nodi dell'insieme preso in considerazione, ma anche del grado di questi.

Il *NormalizedCut* è definito come:

$$\mathrm{N}\left(A_1, \ldots, A_k\right) = \sum_{i=1}^{k} \frac{\operatorname{cut}\left(A_i, \bar{A}_i\right)}{\operatorname{vol}\left(A_i\right)} \tag{2.11}$$

dove

$$\operatorname{vol}\left(A_i\right) = \sum_{v \in A_i} \deg(v)$$

Quindi, minimizzare N equivale a risolvere:

$$\arg\min_{H=\{h_1,\cdots,h_n\}} \quad \frac{1}{2} \operatorname{trace}\ \left\{H^{\top} L_{sym} H\right\} \tag{2.12}$$

$$\text{s.t. } H^T H = I$$

dove

$$L_{sym} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

Il problema risulta quindi analogo al caso del *RatioCut* e di conseguenza anche in questo caso si procederà prima con una soluzione rilassata del problema attraverso i primi k autovettori del Laplaciano (L_{sym}) e successivamente si andrà ad eseguire l'algoritmo *k-means* sui vettori $\{\underline{y}_i\}$

Nota
Nel *NormalizedCut*, gli autovettori $[\underline{u}_1, \underline{u}_2, \ldots . \underline{u}_k]$ vengono normalizzati come segue

$$\tilde{\mathbf{u}}_i = \frac{\mathbf{u}_i}{\|\mathbf{u}_i\|}$$

prima che venga eseguito *k-means*. Lo scopo di questo passaggio è quello di riportare ogni autovettore sulla sfera unitaria, in modo da confrontare solo la direzione del vettore e non la sua lunghezza. In questo modo si evita che i vettori la cui norma è maggiore dominino sugli altri, evitando così il rischio di creare cluster errati.

Il *NormalizedCut* quindi è preferibile nei casi reali in cui la distribuzione del grado dei nodi potrebbe non essere uniforme. Il *RatioCut* infatti non distigue se un nodo che viene tagliato è "importante" oppure no, con il rischio di creare cluster sbilanciati.

Nello specifico, *NormalizedCut* utilizza il Laplaciano L_{sym} invece di L in quanto il Laplaciano L nelle porzioni di grafo in cui la distribuzione del grado dei nodi è molto disomogenea tenderà a favorire tagli che separano nodi di grado basso anche se questi non rappresentano i veri cluster.
E' stato inoltre dimostrato[1] che, dal punto di vista della teoria della convergenza, il Laplaciano L non è consistente al crescere della mole di dati.

[1]Von Luxburg, U. (2007): "A Tutorial on Spectral Clustering"

3. Implementazione

3.1 Eigengap Heuristic

3.1.1 Algoritmo

L'algoritmo utilizzato per il calcolo del numero di cluster ottimale prende in input la matrice di adiacenza A ed esegue i seguenti passaggi:

- Calcolo matrice dei Gradi D
- Calcolo della matrice $D^{-\frac{1}{2}}$
- Calcolo del Laplaciano L_{sym}
- Calcolo degli autovalori di L_{sym}
- Ordinamento degli autovalori calcolati
- Calcolo dei *gap*
- Selezione del *gap* e del relativo autovalore

In output viene restituito il k ottimo trovato tramite questa euristica.

3.1.2 Codice Matlab

```
function k_opt = k_eigengap_heuristic(A, max_k)
    D = diag(sum(A, 2));
    D_inv_sqrt = diag(1 ./ sqrt(diag(D) + eps));
    L_sym = eye(size(A)) - D_inv_sqrt * A * D_inv_sqrt;
    [~, Lamdba_vals] = eigs(L_sym, max_k + 1, 'smallestreal');
    eigvals = sort(diag(Lambda_vals));
    gaps = diff(eigvals);
    [~, k_opt] = max(gaps);
end
```

Nota: Nel codice matlab è presente un secondo parametro in ingresso alla funzione che deve essere oggetto di tuning in base alla cardinalità del problema.

3.2 Spectral Normalized k-Cut

L'algoritmo per la clusterizzazione del grafo prende in input la matrice di adiacenza A del grafo e il numero di cluster k ottimale calcolato precedentemente. I passaggi sono i seguenti:

- Calcolo matrice dei Gradi D
- Calcolo della matrice $D^{-\frac{1}{2}}$
- Calcolo del Laplaciano L_{sym}
- Calcolo dei k autovettori più piccoli di L_{sym}
- Normalizzazione degli autovettori $\{\underline{u}_i\}$ (righe della matrice U) a norma 1
- Calcolo dei cluster tramite l'algoritmo k-means che prende in input i vettori $\{\underline{y}_i\}$ (colonne della matrice U)

In output vengono restuiti i k cluster calcolati.

3.2.1 Codice Matlab

```
function clusters = spectral_k_norm_cut(A, k)
    D = diag(sum(A, 2));
    D_inv_sqrt = diag(1 ./ sqrt(diag(D) + eps));
    L_sym = eye(size(A)) - D_inv_sqrt * A * D_inv_sqrt;
    [U, ~] = eigs(L_sym, k, 'smallestreal');
    U_norm = U ./ vecnorm(U, 2, 2);

    labels = kmeans(U_norm, k, 'Replicates', 10);
    clusters = cell(1, k);
    for i = 1:k
        clusters{i} = find(labels == i);
    end
end
```

Nota
Il codice completo dell'implementazione è disponibile alla repository Github del progetto

4. Dataset

L'algoritmo implementato è stato testato su due diversi dataset: uno sintetico, utilizzato principalmente in fase di sviluppo e debug del codice. Il secondo dataset invece rappresenta una rete internet reale.

4.1 Dati Sintetici

Il dataset sintetico è stato costruito dinamicamente tramite un'apposita funzione Matlab[1] che permette di generare un grafo indicando i seguenti parametri:

- Numero di cluster,
- Numero di nodi per cluster,
- Probabilità di connessione dei nodi intra-cluster (2 nodi appartenenti allo stesso cluster),
- Probabilità di connessione dei nodi inter-cluster (2 nodi appartenenti a cluster differenti),
- Pesi degli archi intra-cluster,
- Pesi degli archi inter-cluster.

Questa funzione permette di definire un grafo come l'unione di più cluster. L'idea di fondo era quelle di riprodurre una rete internet e quindi diversi AS (Autonomous System) collegati tra loro.

Nota
Questa tipologia di struttura del grafo permette, tarando i parametri, di andare a controllare la corretta esecuzione e funzionamento dell'algoritmo implementato.

[1] graphN.m - disponibile nella repository del progetto

4.2 Dataset GARR

Il secondo dataset utilizzato rappresenta l'infrastruttura di rete del consorzio GARR, il quale si occupa di fornire connessioni a banda ultralarga a università, centri di ricerca e musei. La scelta è ricaduta su questa rete in quanto rende accessibile a tutti le informazioni riguardanti l'infrastruttura, la posizione dei nodi e le specifiche di connettività delle rete[2]. E' stato quindi possibile scaricare l'elenco dei nodi della rete, le connessioni tra i nodi e anche la banda di queste connessione.

I dati estratti ottenuto sono stati manualmente corretti ed elaborati in modo da renderli comprensibili da matlab, come nell'esempio sottostante:

Nodes.txt

```
rl1.fi04 [1625,1775]
rl1.fi05 [1675,1825]
...
rl1.pg00 [525,1375]
rl1.rm02 [625,1325]
```

Links.txt

```
rl1.fi04 rl1.fi02 10000
rl1.fi05 rl1.fi02 10000
...
rl1.pg00 rl2.pg00 20000
rl1.pg00 rl1.rm02 10000
```

Nota
Per quanto riguarda questo dataset, ne sono state testate due versioni differenti: la prima contenente solamente i nodi della rete GARR; mentre una seconda è comprensiva anche degli Endpoint[3] esterni alla rete GARR.

[2]GARR Weather Map
[3]Es. Google, Microsoft GEANT, ...

5. Prova Sperimentale

Di seguito vengono riportati i risultati di un'esecuzione dell'algoritmo implementato, sia nel caso di rete generata da dataset sintetico che della rete GARR.

Nota
Nelle seguenti visualizzazione, le etichette degli archi che riportano i pesi sono state rimosse per una migliore visibilità della figura statica.

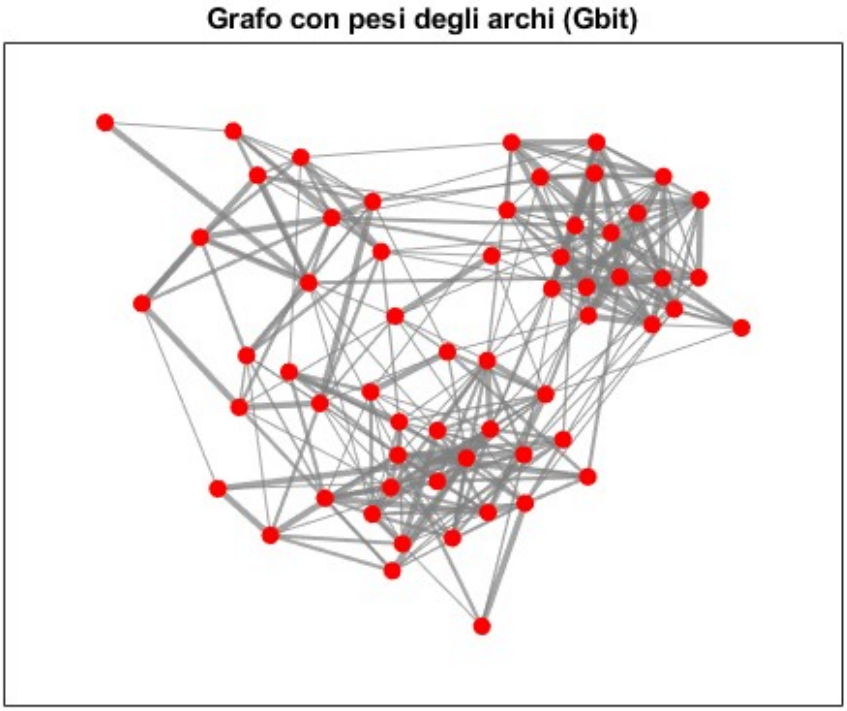
5.1 Rete Sintetica

Per la generazione della rete sintetica sono stati utilizzati i seguenti parametri:

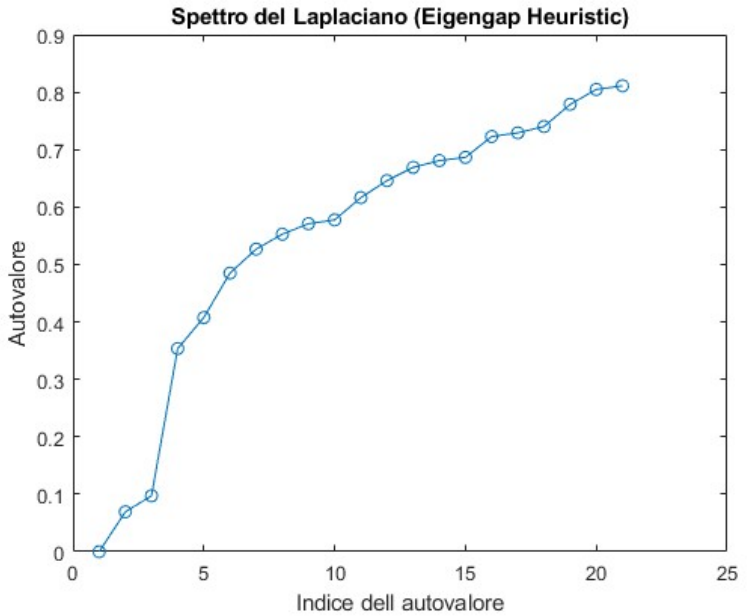
Parametri

```
% Numero di cluster
clusters_num = 3;

% Numero di nodi per singolo cluster
nodes = [15, 25, 20];

% Probabilità di connessione dei nodi intra-cluster
p_intra = [0.4, 0.3, 0.5];

% Probabilita di connessione dei nodi inter-cluster
p_inter = [0 0.05 0.05; 0.05 0 0.05; 0.05 0.05 0];

% Pesi degli archi intra-cluster
weights_intra = {[10, 25, 50, 100]};

% Pesi degli archi inter-cluster
weights_inter = {[1, 5, 10, 25, 50]}
```

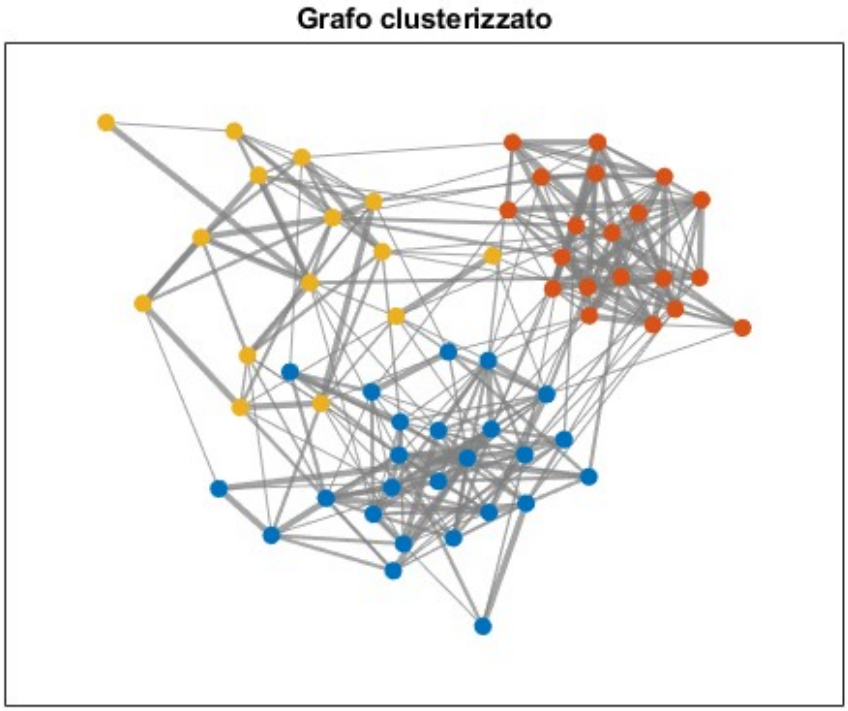
Ed è stata ottenuta la rete in figura

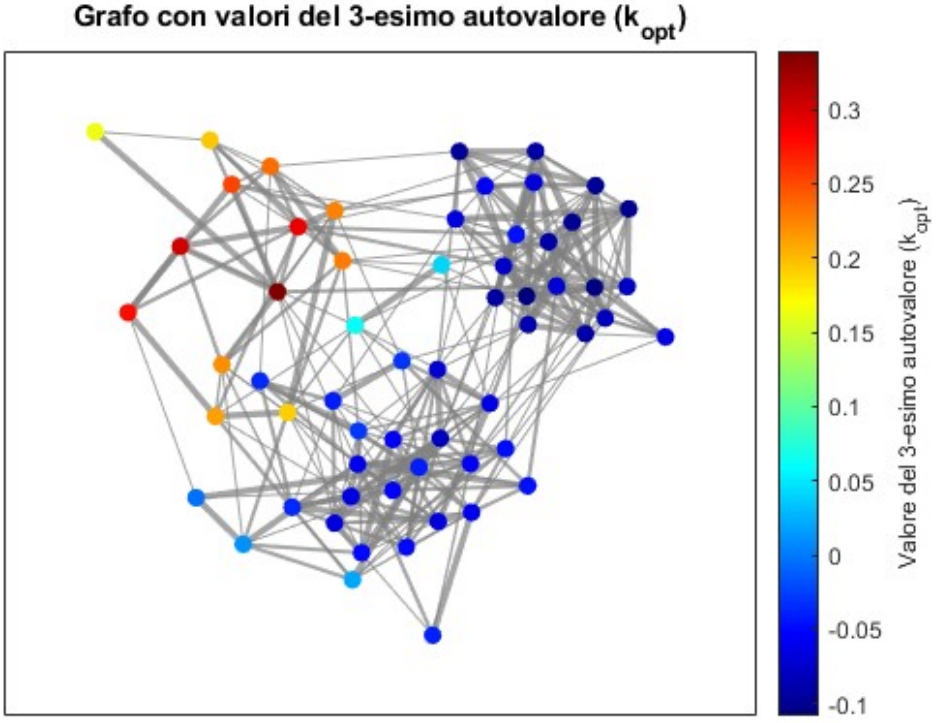


Successivamente sono state applicate la Eigengap Heuristic e la funzione di clusterizzazione.



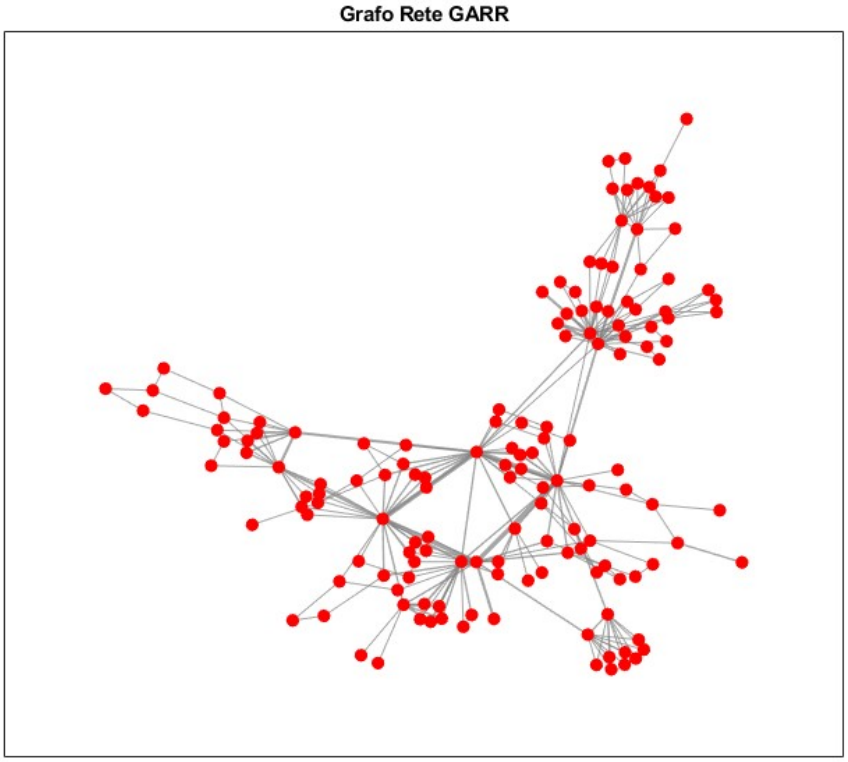
Numero di cluster: 3

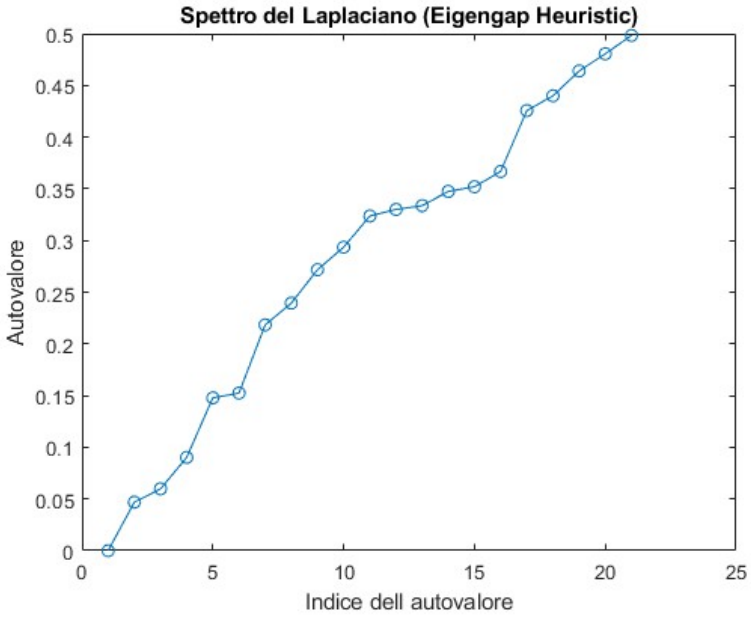

Grafo clusterizzato


Grafo con valori del 3-esimo autovalore (k_opt)
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
Valore del 3-esimo autovalore (k_opt)

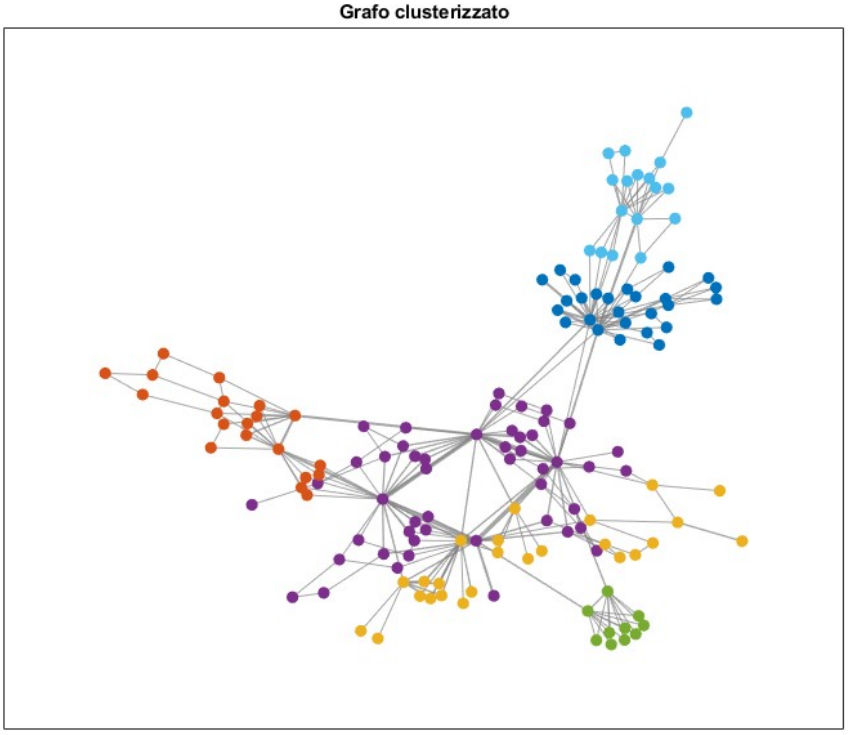
5.2 Rete GARR

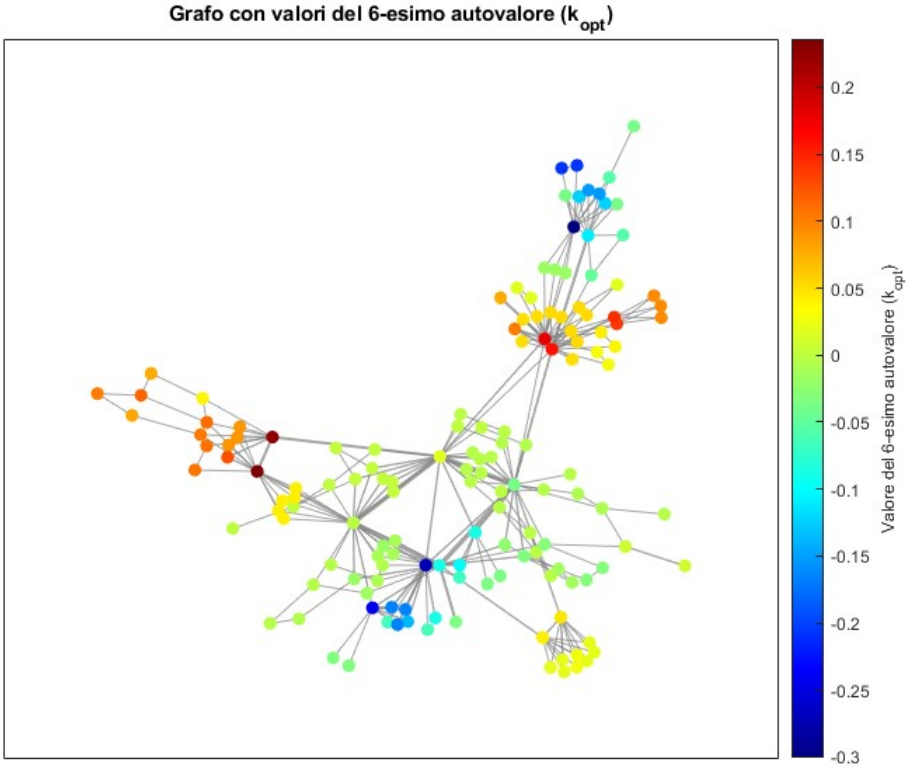
Per quanto riguarda il dataset della rete GARR verrà mostrata l'esecuzione dell'algoritmo utilizzando il dataset che comprende anche gli Endpoint esterni.





Numero di cluster: 6


Grafo clusterizzato


Grafo con valori del 6-esimo autovalore (k_{opt})
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
Valore del 6-esimo autovalore (k_{opt})