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: 11 giugno 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 a identificare il numero di cluster ottimale per il grafo analizzato tramite un'euristica (Eigengap Heuristic), nella 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

Lo spectral graph clustering è un insieme di algoritmi che sfrutta le proprietà spettrali delle matrici associate a un grafo, in particolare la matrice laplaciana, per rilevare insiemi di nodi fortemente connessi internamente e debolmente connessi con il resto della rete. Differentemente da altri metodi utilizzati, il clustering spettrale non è basato su delle metriche legate allo spazio euclideo, ma si basa sullo studio degli autovalori e autovettori del grafo, mappando i nodi in uno spazio a dimensionalità ridotta in cui i cluster emergono in maniera più evidente. Quindi lo spectral graph clustering è basato sull'idea che se in un grafo è presente una struttura, questa dovrebbe emergere in modo naturale dallo spettro della matrice laplaciana.

Nello specifico, la matrice laplaciana è ottenuta come:

$$L = D - A \tag{2.1}$$

In cui A è la matrice di adiacenza del grafo e D la matrice dei gradi.

La matrice laplaciana dispone inoltre delle seguenti proprietà:

- è simmetrica e semi-definita positiva $f^T L f \geq 0$
- ha una base ortogonale di autovettori $u_1, \cdots, u_n$

Nell'ambito dello spectral graph clustering si utilizza la matrice laplaciana, in quanto rappresenta quanto ogni nodo differisca dai suoi vicini, più formalmente:

$$f^\top L f = \sum_{(i,j)\in E} (f_i - f_j)^2 \qquad \forall f \in \mathbb{R} \tag{2.2}$$

quindi, questo valore, definito come *Total Variation*, sarà piccolo quando i nodi connessi avranno valori simili. Per questo motivo si utilizza la matrice laplaciana per identificare i cluster.

Un'ulteriore considerazione sulle componenti spettrali di L va fatta sul più piccolo autovalore che sarà sempre 0, mentre il numero complessivo di autovalori nulli corrisponderà al numero delle componenti connesse del grafo. Quindi, nel caso di un grafo connesso avremo solamente $\lambda_1 = 0$.

Il problema della clusterizzazione di un grafo è quindi strettamente legato alla minimizzazione della *Total Variation* di L, in quanto è desiderabile che i nodi delle componenti

connesse abbiano i valori più simili possibili. La soluzione al problema della minimizzazione della *Total Variation* di L è il vettore di *fiedler*, ovvero l'autovettore associato al secondo autovalore più piccolo (λ_2).
Il vettore di fiedler tenderà ad assumere valori simili sui nodi fortemente connessi tra loro e valori diversi tra i nodi separati. Si avrà quindi che quando $\lambda_2 \approx 0$ il grafo sarà quasi disconnesso e risulterà quindi evidente la presenza di un taglio naturale tra i nodi del grafo. Mentre il valore di λ_2 sarà grande nel caso in cui il grafo sia fortemente connesso.

Questo legame tra autovalori e struttura del grafo è il fondamento dello spectral graph clustering.

2.1 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.

Nota
Per trovare il numero ottimo di cluster k è stata usata la *Eigengap Heuristic* (vedi 2.2)

2.1.1 k-Cut

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

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

si vuole andare a minimizzare la dimensione del taglio, ovvero

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

2.1.2 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:

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

con

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

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

Il problema precedente si può riscrivere come

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

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 cluster. Questo è possibile tramite l'utilizzo dell'algoritmo *k-means* sulle componenti spettrali.

Nello specifico, posto:

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

si andrà ad eseguire l'algoritmo *k-means* (vedi 2.1.4) sui vettori $\{\underline{y}_i\}$ per identificare i centrodi (cluster) e per assegnare ogni nodo al centroide a distanza minima.

2.1.3 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 della cardinalità dell'insieme dei nodi A_i, ma anche del grado dei nodi contenuti in quest'ultimo.

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* (vedi 2.1.4) 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.

Per fare ciò, il *NormalizedCut* utilizza il Laplaciano L_{sym} invece di L, in quanto l'utilizzo del Laplaciano L nelle porzioni di grafo in cui la distribuzione del grado dei nodi è molto disomogenea tenderà a dare risultati distorti che non rispecchiano la struttura naturale del grafo.
L_{sym}, invece, tiene conto del grado dei nodi vicini, risultando più robusta nel caso di grafi non regolari.

Formalmente, la minimizzazione della *Total Variation* di L_{sym} è pari alla minimizzazione della seguente funzione:

$$f^\top L_{\text{sym}} f = \sum_{(i,j)\in E} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2 \tag{2.13}$$

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.

2.1.4 k-means

L'idea su cui si basa k-means è quella di raggruppare i dati in k insiemi (cluster), in modo tale che ogni punto appartenga al centroide più vicino. Nello specifico, k-means si può applicare a dei dati distribuiti in uno spazio (es. spazio euclideo) e procede andando ad identificare dei nuovi punti dello spazio, chiamati centroidi, che riescano a rappresentare bene gruppi di punti simili.
Successivamente, si assegna ogni punto al centroide più vicino, in questo modo, un cluster sarà formato da tutti i nodi che sono associati ad un centroide. Come ultimo passo si va a ricalcolare la posizione del centroide facendo la media dei punti associati a quello stesso centroide. Questi passaggi vengono ripetuti finché la situazione non si stabilizza (convergenza o superamento soglia o numero massimo di iterazioni).

Formalmente:

dato un insieme di n punti $x_1, \cdots, x_n \in \mathbb{R}^m$ e un numero fissato di cluster $k \in \mathbb{N}$, l'algoritmo deve identificare k centroidi $\mu_1, \cdots, \mu_k \in \mathbb{R}^m$ e una partizione dei punti $C_1, \cdots, C_k$ che minimizzi:

$$\sum_{i=1}^{k} \sum_{x\in C_i} \|x - \mu_i\|^2 \tag{2.14}$$

Nello specifico, l'algoritmo (iterativo) eseguirà le seguenti fasi:

1. **Inizializzazione**: vengono scelti k centroidi (random o seguendo un criterio specifico)
2. **Assegnazione**: ogni punto viene assegnato al cluster con il centroide più vicino
3. **Aggiornamento**: le coordinate di ogni centroide vengono aggiornate facendo la media dei punti assegnati al centroide
4. **Iterazione**: ripetizione dei passi 2 e 3 fino al raggiungimento del criterio di stop (convergenza, superamento soglia o numero massimo di iterazioni)

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

Nel caso dello spectral graph clustering si utilizza *k-means* sulle componenti spettrali del grafo in quanto i primi k autovettori del Laplaciano, ovvero quelli associati ai k autovalori più piccoli, rappresentano una proiezione dei nodi in un dominio k-dimensionale.
In questo spazio ogni nodo verrà rappresentato come un punto $\in$ a $\mathbb{R}^k$.

Lo spazio ottenuto è costruito sfruttando le componenti spettrali del Laplaciano, le quali tendono a mettere in risalto la struttura naturale del grafo, se presente. In questo modo anche lo spazio k-dimensionale rifletterà la struttura del grafo.

Quindi, dopo aver proiettato i nodi nello spazio $\mathbb{R}^k$, sarà possibile applicare l'algoritmo *k-means* ai vettori ottenuti come righe della matrice degli autovettori (vedi Eq. 2.10), in modo da raggruppare i nodi in base alla loro rappresentazione spettrale, la quale riflette le connessioni nel grafo.

Un ulteriore prova di ciò è evidente se, preso un grafo diviso in k cluster tramite i passaggi precedentemente descritti, si calcola il k-esimo autovalore su ogni nodo; risulterà immediato vedere che, i nodi appartenenti allo stesso cluster avranno valori di λ_k simili, mentre nodi appartenenti a cluster diversi avranno valori diversi.

2.2 Eigengap Heuristic

Questa euristica viene utilizzata per calcolare il numero ottimale di cluster in cui dividere il grafo. Il fondamento teorico per cui questa euristica è ritenuta valida è strettamente legato alla teoria spettrale dei grafi. Nello specifico, se un grafo ha k componenti connesse disgiunte, la matrice di Laplace avrà i k autovalori più piccoli nulli (i.e. $\lambda_1 = \lambda_2 = \cdots = \lambda_k = 0$) e un salto all'autovalore *k+1*-esimo ($\lambda_{k+1} > 0$).
In caso reale, in cui i dati non formano cluster perfettamente disgiunti ma ci sono k gruppi di nodi fortemente connessi, i primi k autovalori risulteranno piccoli con un salto netto al λ_{k+1}.
Quindi un salto tra gli autovalori λ_k e λ_{k+1} indica che i primi k autovettori, ovvero quelli usati per il clustering, descrivono bene la struttura a k cluster. 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.

$$gap_i = \lambda_{i+1} - \lambda_i \tag{2.15}$$

- trovare il *gap* massimo

$$k = \arg\max_i(gap_i) \tag{2.16}$$

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 (`k_max`) 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 restituiti 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 ottenuti 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

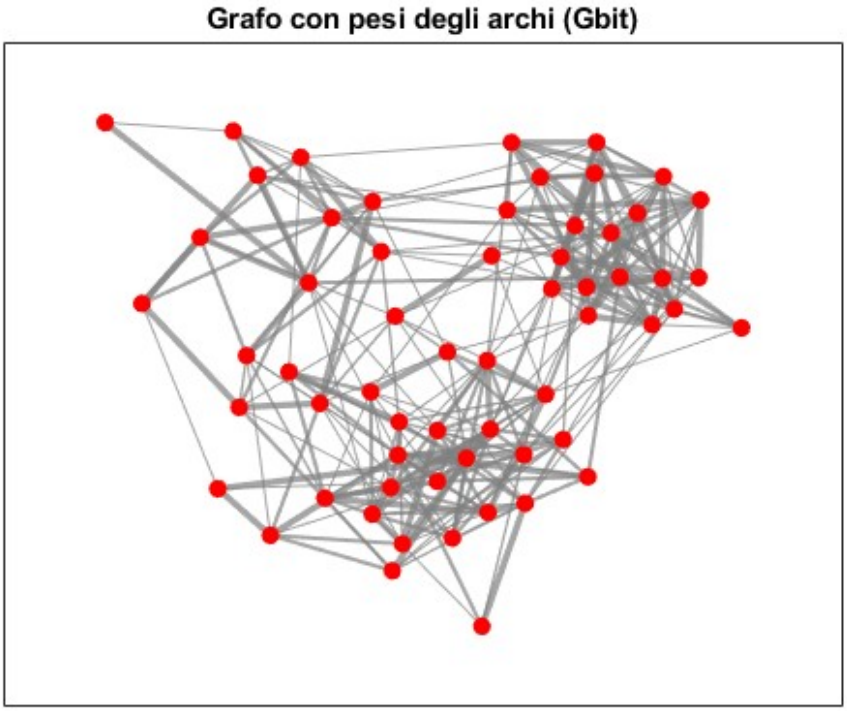
5.1.1 Normalized Cut

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

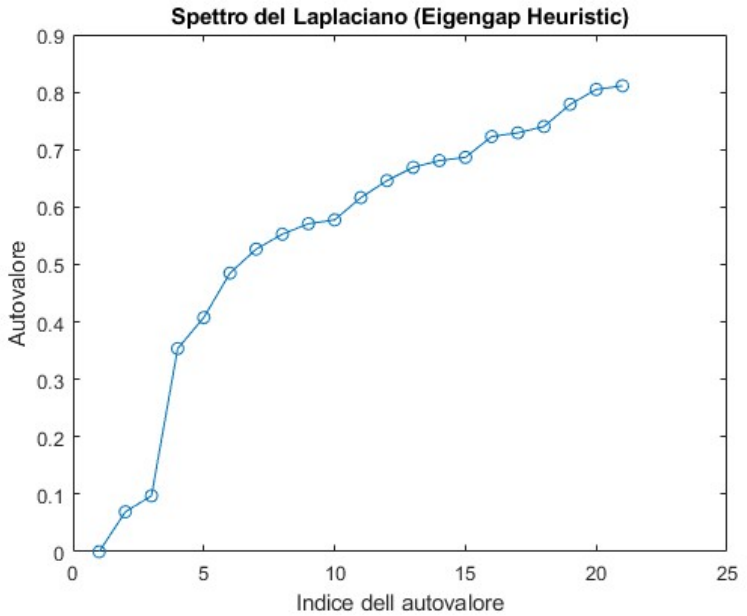
Parametri

```
% 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]}
```

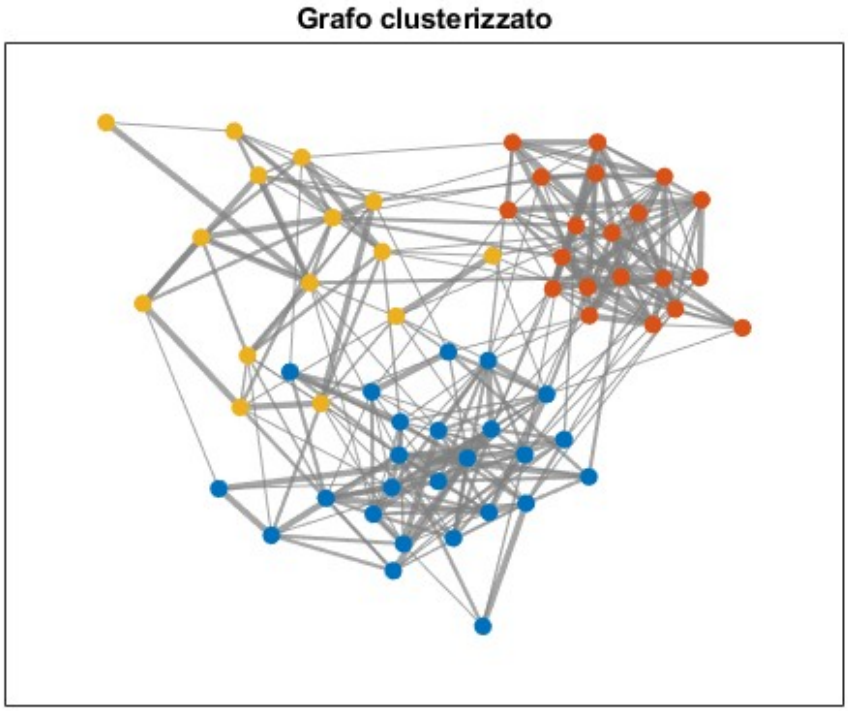
Ed è stata ottenuta la rete in figura

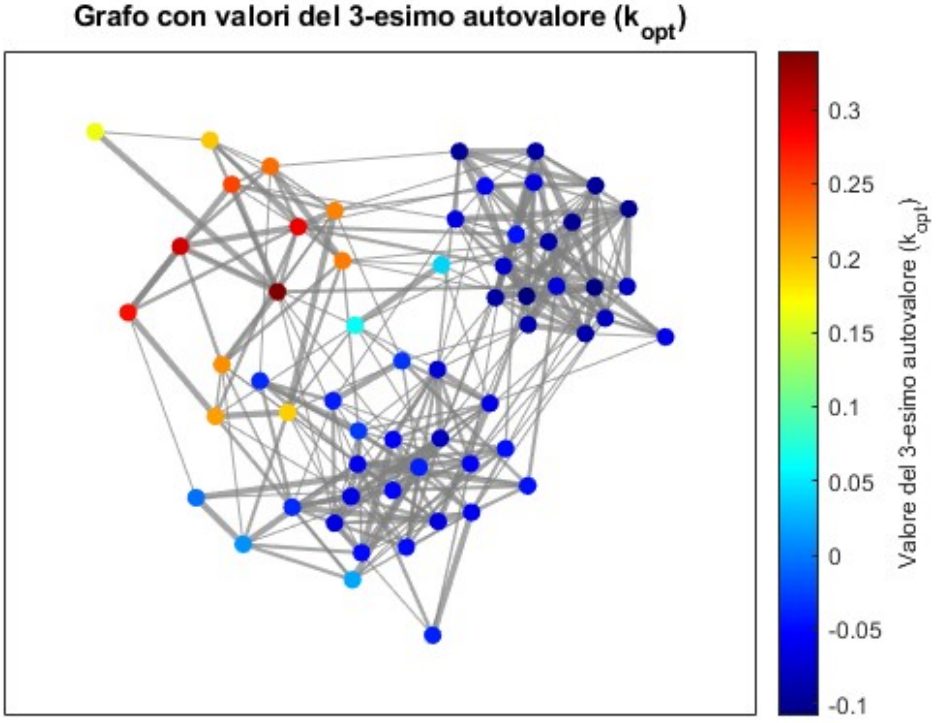


Successivamente sono state applicate la Eigengap Heuristic e la funzione di clusterizzazione.



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)

5.1.2 Confronto metodi

Nella seguente sezione viene mostrato il confronto tra l'esecuzione dei tre metodi descritti nel capitolo 2 per la clusterizzazione di un grafo. Nello specifico, sono stati applicati i tre diversi algoritmi al variare della probabilità di connessione inter-cluster; il motivo di questa scelta è dovuto al fatto che, come precedentemente affermato, queste tecniche di clusterizzazione del grafo utilizzano le componenti spettrali della matrice per far venire fuori la struttura naturale del grafo. Il risultato atteso è che, all'aumentare della probabilità di presenza di archi inter-cluster, questa struttura del grafo venga meno, con un conseguente aumento di errore nella classificazione dei nodi. Come metrica è stata utilizzata la percentuale di corretta classificazione dei nodi.

Nota
Lo script utilizzato per questo test è disponibile nella repository github, con il nome `"confronto_sint.m"`

Mentre i parametri utilizzati per la simulazione sono i seguenti

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 = [0.5, 0.5, 0.5];

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

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

% Probabilità di connessione dei nodi inter-cluster
p_inter_values = 0:0.05:1;

% Numero di iterazioni
num_iterations = length(p_inter_values);

% Numero di ripetizioni per iterazione
num_repeats = 100;
```

Come si può notare, oltre al valore `p_inter` (probabilità di creazione arco inter-cluster) che varia da 0 a 1 con salti di 0.05.

Vengono eseguite 100 ripetizioni per ogni step, ciò è dovuto al fatto che il grafo viene generato in maniera randomica e per rendere la statistica significativa si è reso necessario

ripetere la misurazione della metrica per ogni iterazione più volte.

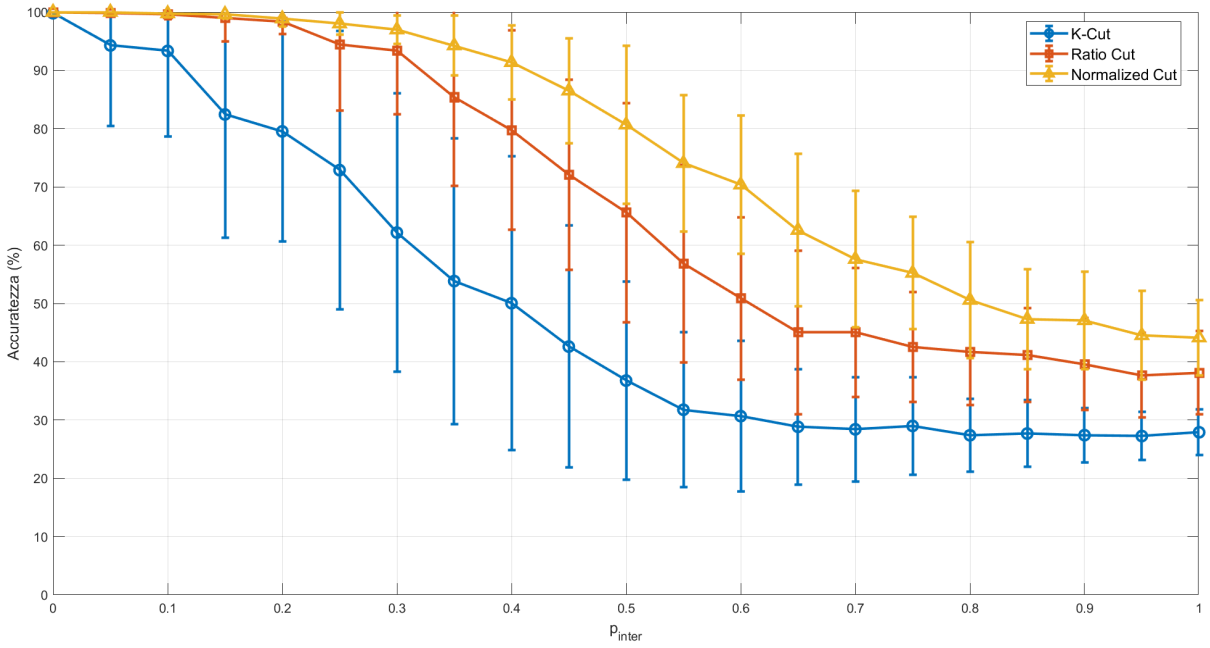


Figura 5.1: Percentuale accuratezza al variare di `p_inter`

I risultati ottenuti sono visibili in figura ed è subito evidente come il metodo Normalized cut e il Ratio cut forniscano dei risultati nettamente migliori all'aumentare delle connessioni tra i cluster.

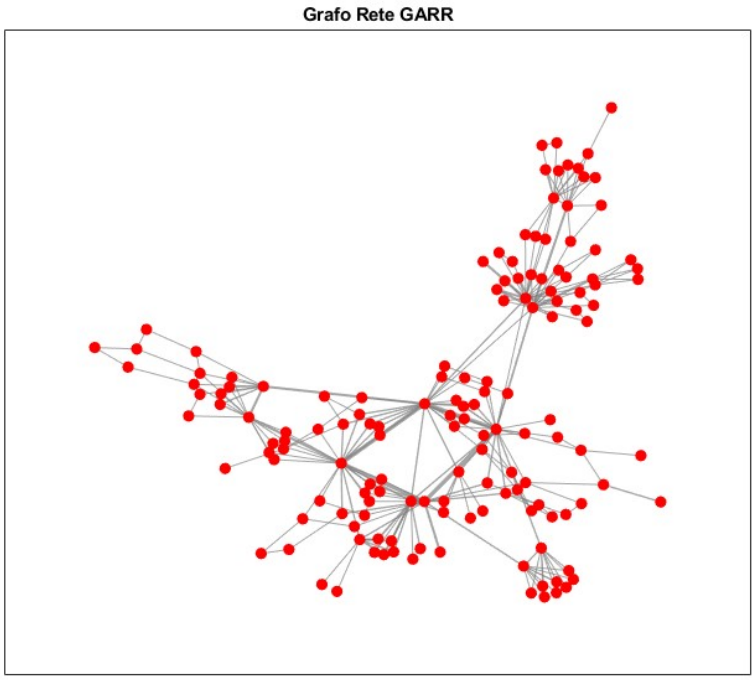
Inoltre, è doveroso fare un'ulteriore considerazione per ciò che riguarda la probabilità di connessione intra-cluster, quella utilizzata per la simulazione è pari al 50% ed è quindi poco significativo andare a confrontare le misurazioni in cui la probabilità inter-cluster supera quella intra-cluster, in quanto la struttura del grafo che si sta cercando di analizzare risulta quasi del tutto assente, avendo le stesse probabilità di connessione tra nodi che appartengono e non appartengono allo stesso cluster.

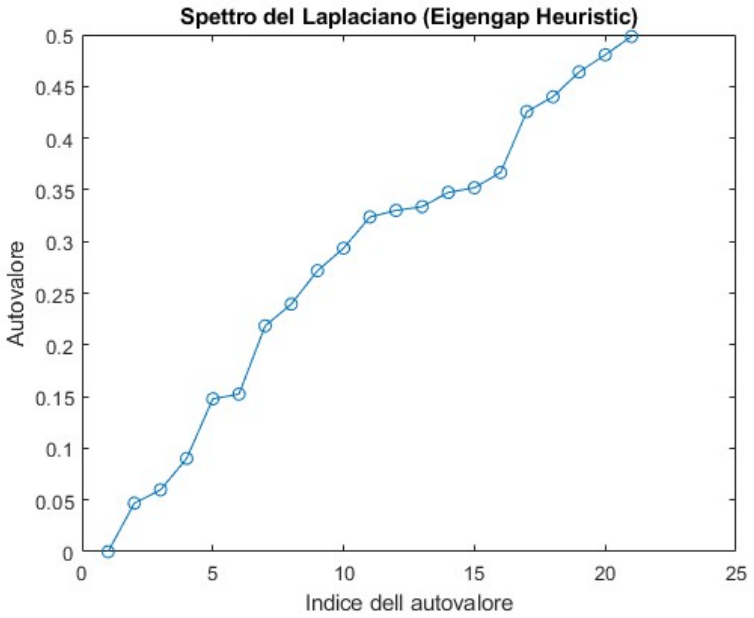
E' importante anche notare che in questa simulazione gli archi intra-cluster hanno pesi maggiori rispetto agli archi inter-cluster, e di conseguenza, le prestazioni non si degradano troppo velocemente rispetto ad un grafo non pesato. Infatti, tutti gli algoritmi visti si basano sulla minimizzazione del taglio e di conseguenza verranno prima tagliati i nodi inter-cluster che hanno in media un peso minore.

5.2 Rete GARR

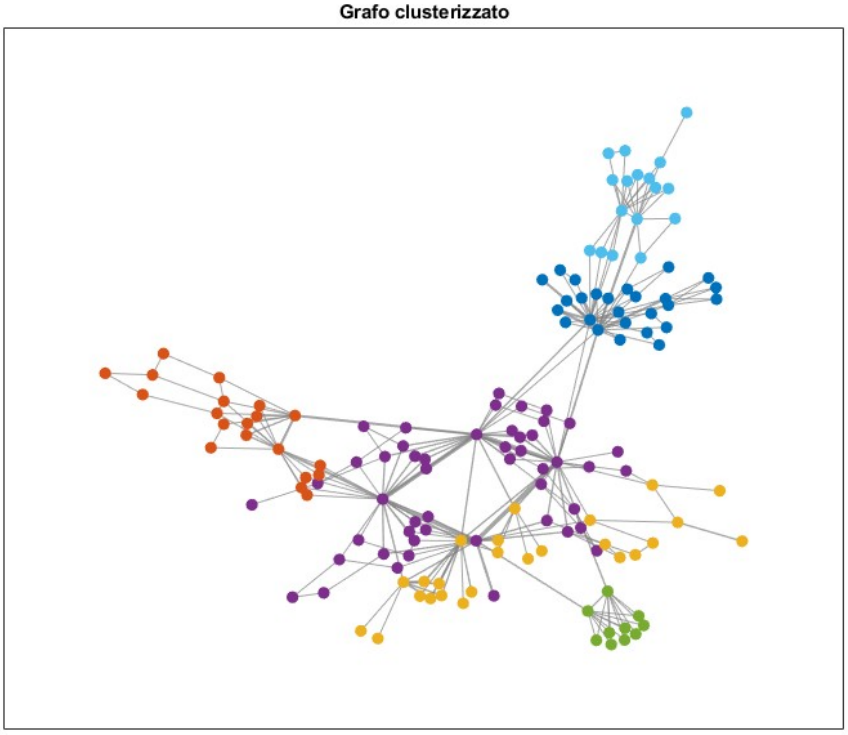
5.2.1 Normalized Cut

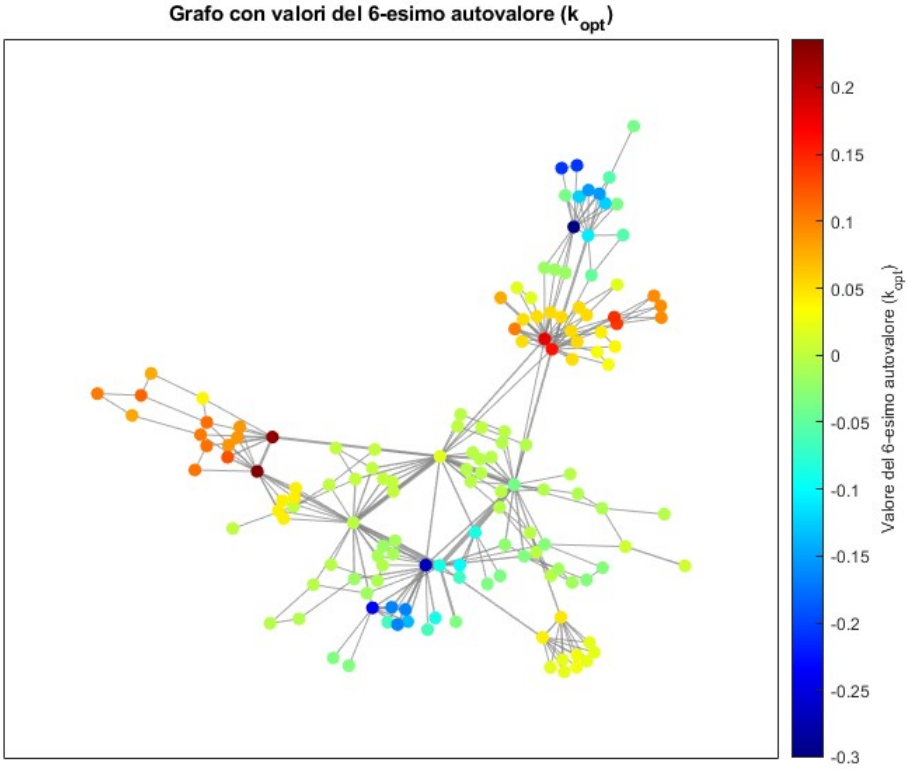
Per quanto riguarda il dataset della rete GARR verrà mostrata l'esecuzione dell'algoritmo utilizzando il dataset che comprende anche gli Endpoint esterni.





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)

5.2.2 Confronto metodi

Come per la prova sperimentale con dati sintetici anche con il dataset della rete GARR è stato svolto un test per confrontare i tre algoritmi illustrati nei capitoli precedenti e giustificare l'utilizzo del Normalized Cut rispetto al Ratio Cut e al k-Cut

Nota
Il live script utilizzato per questo test è disponibile nella repository github, con il nome `"confronto_garr.mlx"`

Come già visto in precedenza, tutti i metodi utilizzati si basano sullo studio delle componenti spettrali della matrice laplaciana, quindi accanto ad ogni plot del grafico clusterizzato viene mostrato anche il valore del k-esimo autovalore su tutti i nodi del grafo. Nello specifico, questa simulazione utilizza i dati del dataset GARR, compensivo di Endpoint esterni, ed è stato stimato un $k_{opt} = 6$ tramite la *Eigengap Heuristic*.

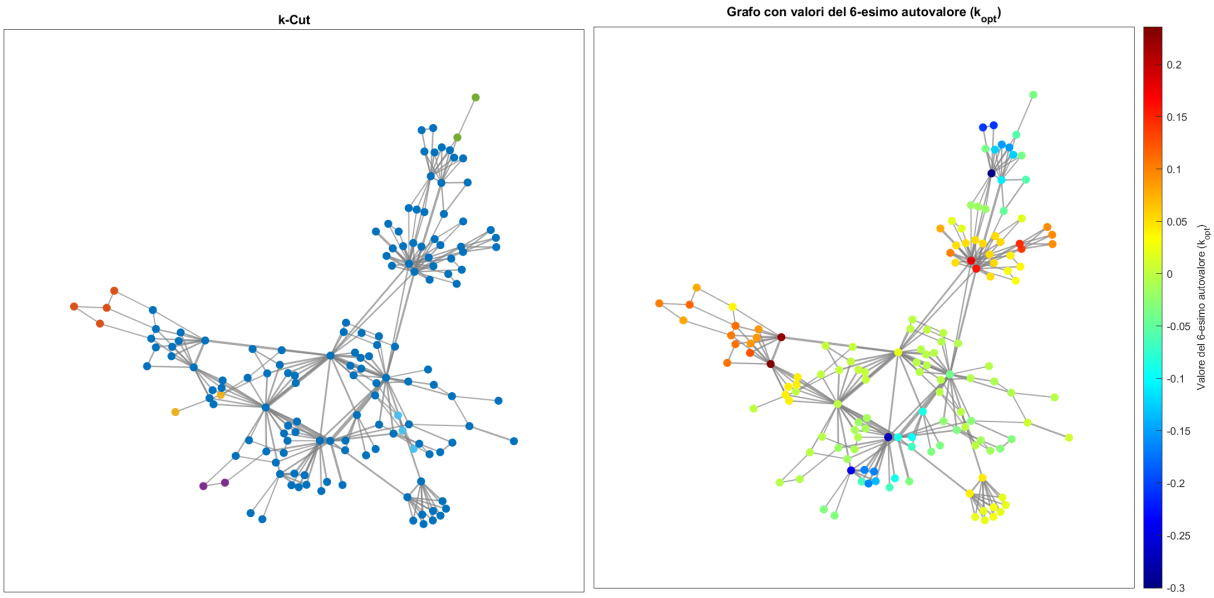


Figura 5.2: Confronto k-Cut e k-esimo autovalore

Nel primo caso, quello in cui è stato utilizzato k-Cut, è evidente come la clusterizzazione tenda solamente a minimizzare il taglio portando alla formazione di un cluster molto grande, mentre i restanti cinque cluster sono formati al più da quattro nodi. E' inoltre evidente come la clusterizzazione tenga poco conto del valore di λ_6 (k-esimo autovalore).

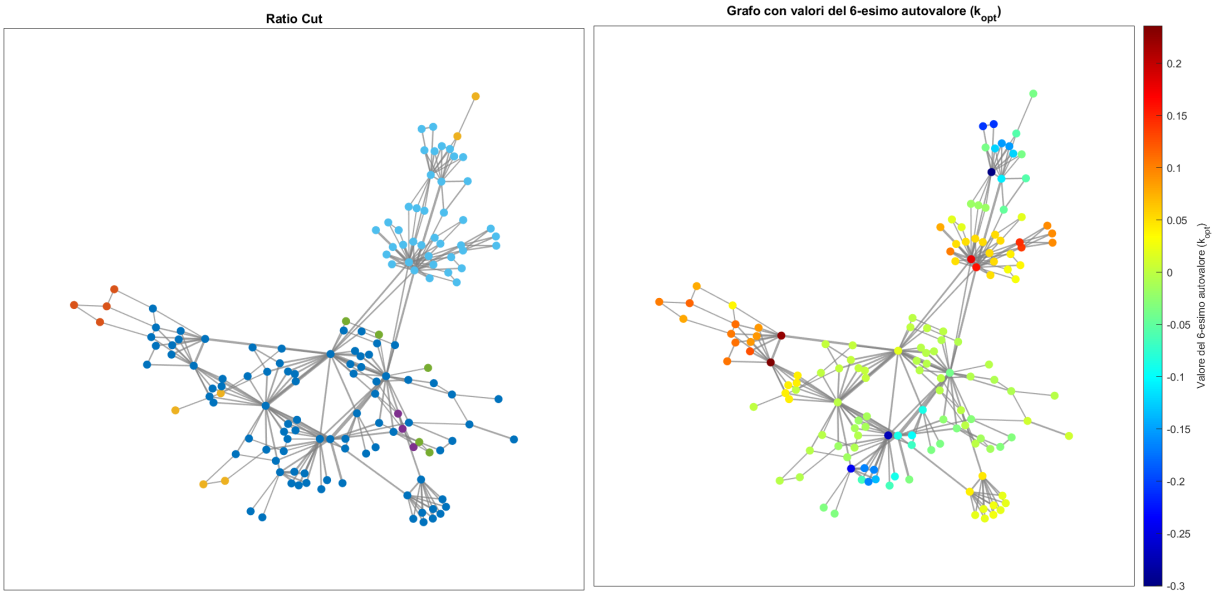


Figura 5.3: Confronto Ratio-Cut e k-esimo autovalore

Nel caso di utilizzo di Ratio Cut, invece, si scoraggia la creazione di di cluster piccoli e confrontando le due immagini in figura 5.3 risulta immediato come la clusterizzazione tramite l'algoritmo Ratio Cut rispecchi meglio di k-Cut la struttura che emerge dal grafo in cui viene mostrato il k-esimo autovalore (λ_6) su nodi del grafo.

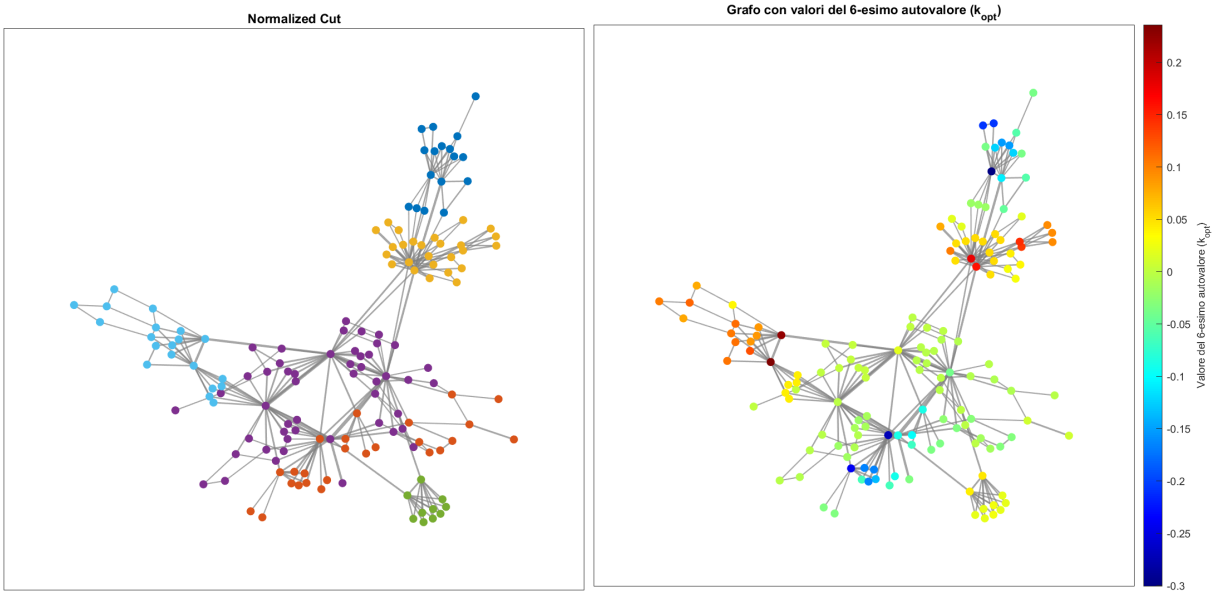


Figura 5.4: Confronto Normalized-Cut e k-esimo autovalore

Nell'ultimo caso, dove è stata utilizzato l'algoritmo Normalized Cut, questo costruisce dei cluster che approssimano molto bene la variazione del autovalore λ_6.

Nota
In questa simulazione si è utilizzato un valore k_{opt} calcolato tramite la *Eigengap Heuristic* che a sua volta è stata implementata utilizzando il Laplaciano L_{sym}. Per coerenza teorica sarebbe stato meglio implementare una versione alternativa dell'euristica che utilizzasse

un Laplaciano non normalizzato, da utilizzare per gli algoritmi k-Cut e Ratio-Cut, i quali utilizzano L e non L_{sym} per il processo di clusterizzazione.

Il motivo per cui si è preferito non utilizzare due algoritmi differenti per calcolare il k_{opt} è dovuto al fatto che, per questo dataset, la *Eigengap Heuristic* implementata con il Laplaciano L restituiva come risultato $k_{opt} = 1$, il che è in evidente conflitto con lo scopo del progetto, ovvero la divisione del grafo in cluster.

6. Conclusioni e sviluppi futuri

6.1 Conclusioni

Questa tesina ha analizzato il problema della clusterizzazione spettrale di un grafo e della sua implementazione pratica, ponendo un focus sull'implementazione dell'algoritmo per l'utilizzo su reti internet modellate con un grafo. Il problema è stato suddiviso in due ulteriori sottoproblemi: trovare il numero di cluster ottimo tramite l'Eigengap Heuristic e clusterizzare il grafo utilizzando l'algoritmo Normalized Cut.

I risultati delle simulazioni condotte sia su reti sintetiche che su dataset reali (rete GARR) hanno confermato la validità del metodo scelto. In particolare, rispetto ad altri algoritmi (k-Cut e Ratio Cut), Normalized Cut si è rivelato più robusto soprattutto nel caso di grafi non regolari, confermando le ipotesi teoriche fatte a priori.

Al netto di alcune limitazioni da un punto di vista teorico, per quanto riguarda i tipi di Laplaciano utilizzato, l'Eigengap Heuristic ha permesso di stimare efficacemente il numero di cluster.

Complessivamente, l'algoritmo scelto è stato in grado di rivelare efficacemente la struttura latente della rete, evidenziando parti debolmente connesse che potrebbero indicare aree critiche o colli di bottiglia. L'intera implementazione si è rivelata un buon punto di partenza per eventuali sviluppi futuri.

6.2 Sviluppi Futuri

Per quanto riguarda la rete GARR, un possibile sviluppo futuro potrebbe essere l'integrazione di sistema di recupero dati dinamico che permetta di utilizzare come pesi degli archi la percentuale di banda utilizzata su ogni collegamento tra due nodi al passare del tempo.

In questo modo si potrebbe fare un'analisi più accurata di quali sono le porzioni della rete più critiche e soggette a rallentamento o in cui il rischio di partizionamento è più alto.