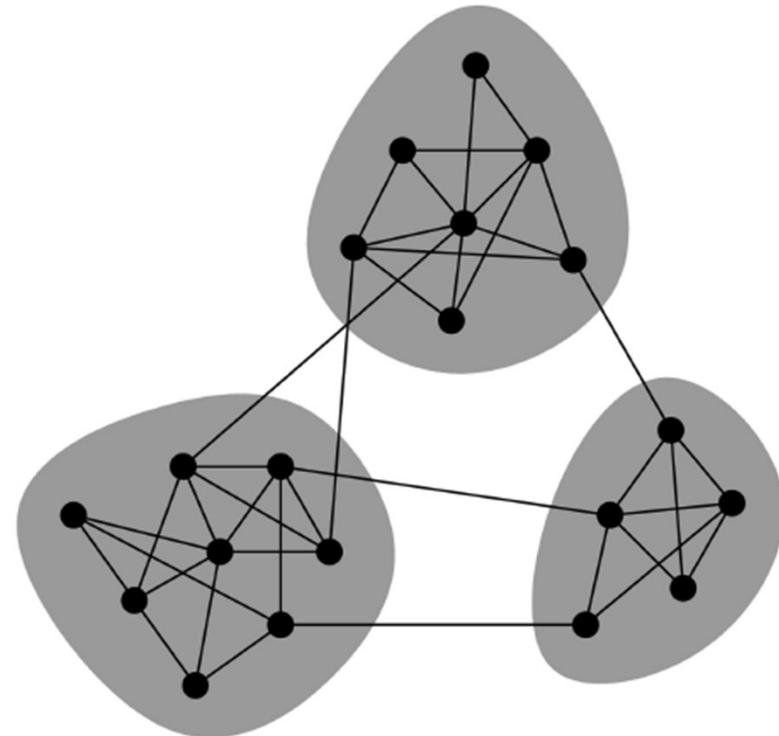


# Comunità nelle Reti

# Network e Comunità

- Spesso pensiamo alle network come qualcosa del genere:

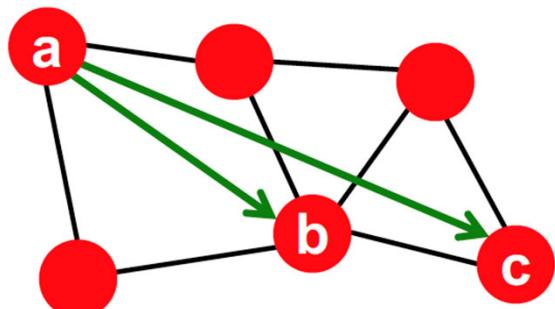


- Cosa ci porta a questa rappresentazione concettuale?

- **Quali sono i flussi di informazione in una rete?**
  - Quali ruoli hanno i nodi nella struttura della rete?
  - Quali ruoli differenti hanno i link (lunghi o corti)?
- **Come viene trovato il lavoro da una persona?**
  - Attraverso i contatti personali...
- **Ma: i contatti a volte sono solo dei conoscenti piuttosto che veri amici.**
  - **Sorprendente:** Ognuno di noi si aspetta che gli amici siano più dia aiuto dei conoscenti..
- **Ma allora perché i conoscenti sono più utili?**

# Granovetter 1973

- Due prospettive di amicizia:
  - **Strutturale:** l'amicizia si distribuisce in diverse parti della rete
  - **Interpersonale:** L'amicizia tra due persone può essere **forte** o **debole**
- **Ruolo strutturale delle Chiusure Triadiche**

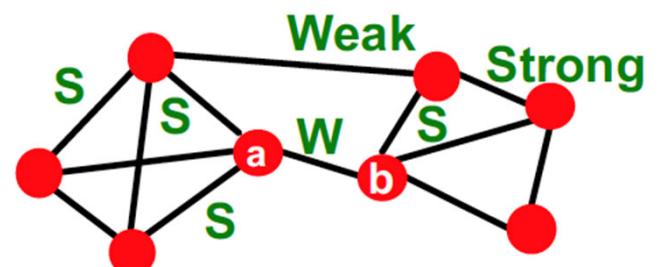


Quale arco è più verosimile  
a-b oppure a-c?

Se due persone in una rete hanno un amico in comune c'è una più alta probabilità che questi diventino amici

# Spiegazione di Granovetter

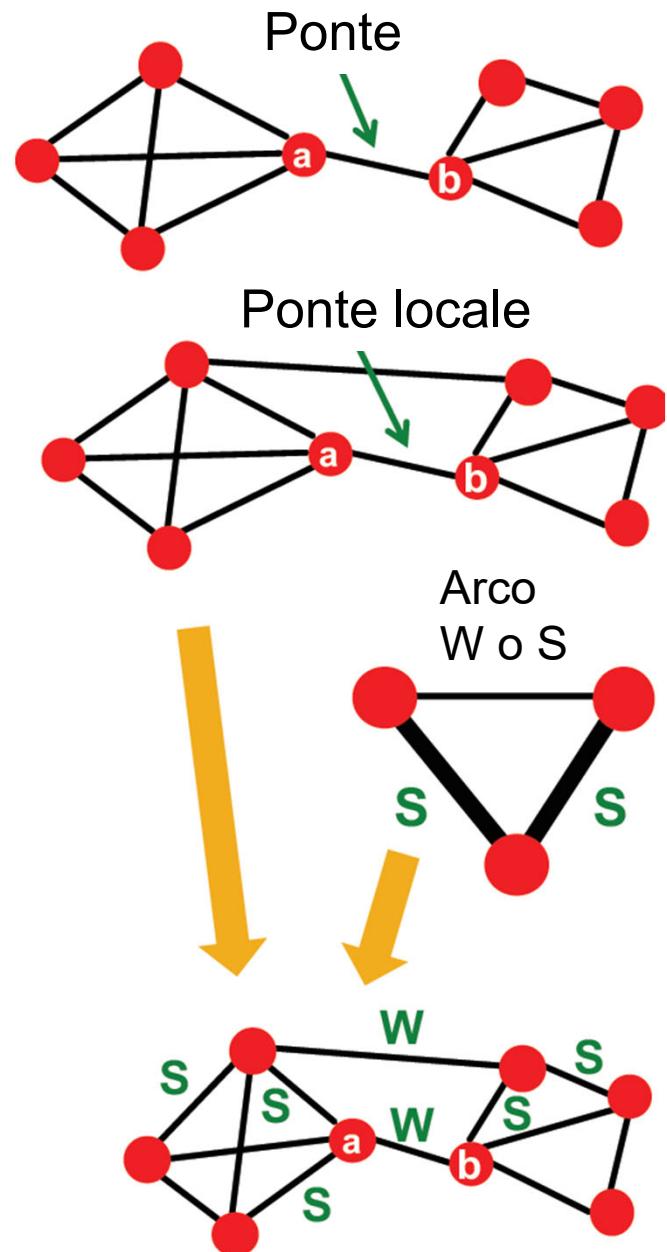
- **Connessione tra ruolo strutturale e sociale di un arco**
- **Primo punto: Struttura**
  - Archi all'interno di un struttura sono socialmente forti;
  - Archi lunghi che connettono diverse parti della rete sono socialmente deboli.
- **Secondo punto: Informazione**
  - Gli archi lunghi consentono di ottenere informazioni da diverse parti di una rete;
  - Gli archi all'interno di una struttura sono ridondanti in termini di accesso all'informazione.



# Chiusura Triadica

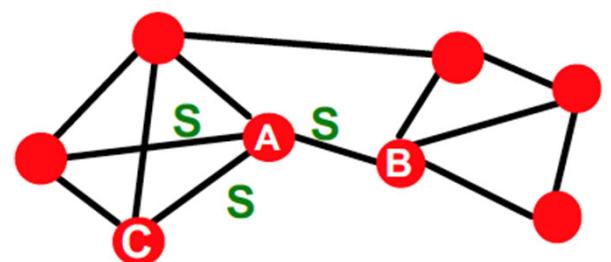
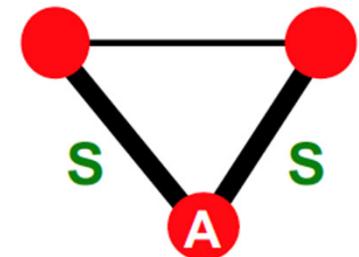
- **Chiusura triadica == Alto coefficiente di clustering**
- **Motivazioni**
- Se **B** e **C** hanno un amico **A** in comune, allora:
  - **B** verosimilmente incontrerà **C** (**A** spende del tempo con entrambi)
  - **A** è incentivato a legare **B** e **C**
- **Uno studio empirico di Bearman e Moody dimostra che le ragazze teenager con basso coefficiente di Clustering sono più inclini al suicidio**

- **Arco ponte:**
  - Se rimosso il grafo diventa disconnesso
- **Ponte locale:**
  - Arco con **Span >2**  
Span = distanza degli endpoint dell'arco se l'arco viene cancellato
- Due tipi di archi:
  - **Strong** (forti, amici), **Weak** (deboli, conoscenti)
    - Due legami forti implicano un terzo arco
- Fatto: Se la chiusura triadica è soddisfatta i ponti locali sono connessioni deboli



# Ponti locali e connessioni deboli

- Se un nodo **A** soddisfa una **Chiusura Triadica Forte** ed è coinvolto in almeno **due connessioni forti** allora un qualsiasi **ponte locale** adiacente ad A deve esser una **connessione debole**
- Per assurdo:
  - Assumiamo che **A** soddisfi una **Chiusura triadica Forte** ed abbia **due connessioni forti**
  - Sia **A—B** un **ponte locale** ed allo stesso tempo una connessione **forte**
  - Allora **B—C** deve esistere per la presenza della **chiusura triadica forte**
  - Allora **A—B non è un ponte locale** (assurdo)!



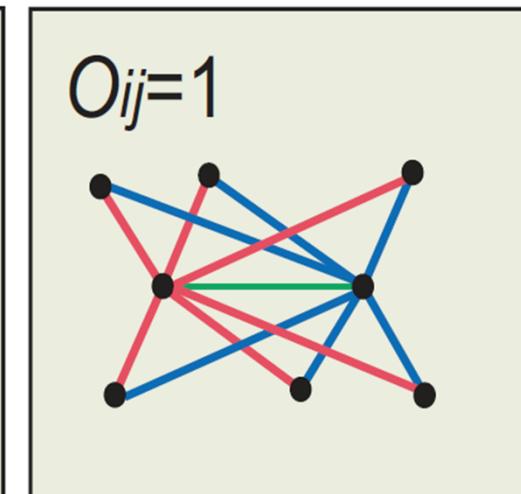
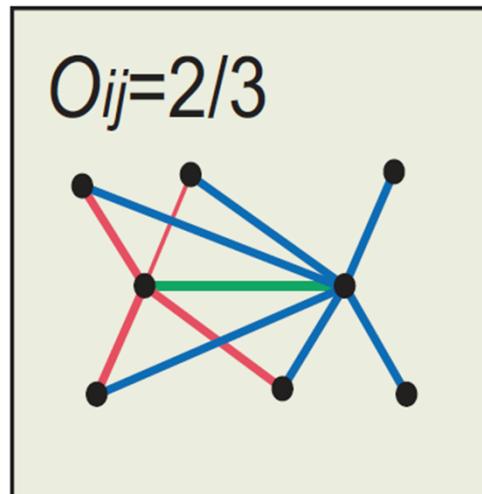
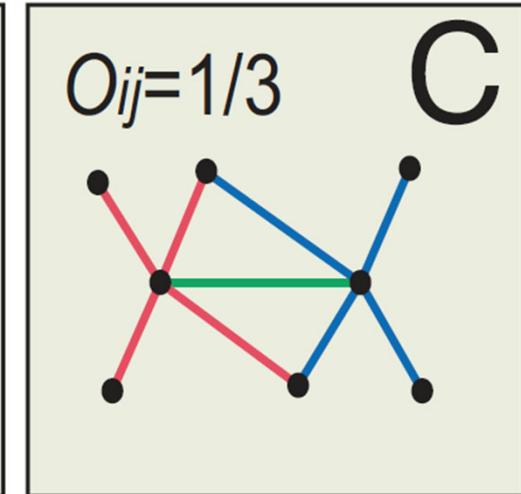
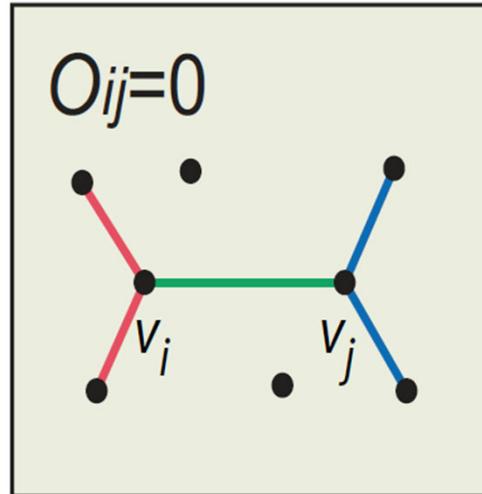
# Neighborhood overlapping

- Overlap di un arco:

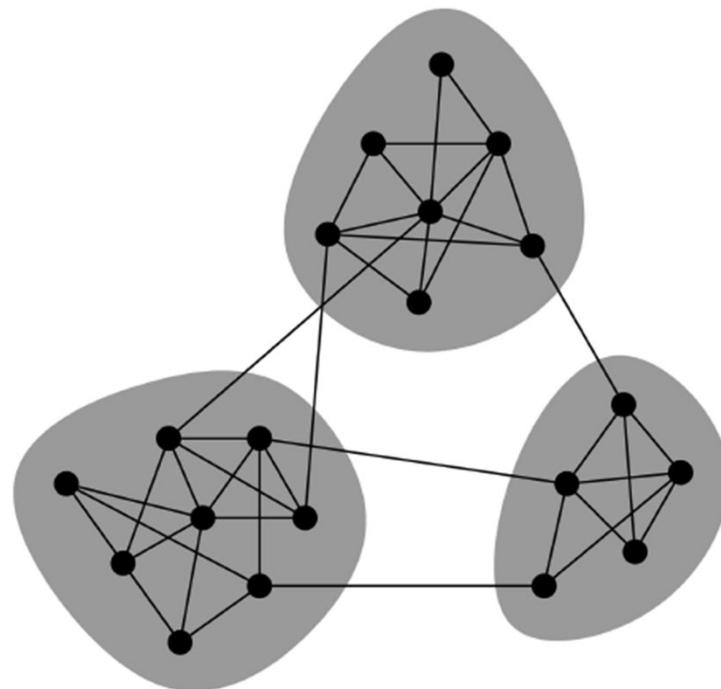
$$O_{ij} = \frac{N(i) \cap N(j)}{N(i) \cup N(j)}$$

Con  $N(i)$  insieme dei vicini di  $i$ .

**Overlap=0** quando un arco è un **ponte locale**

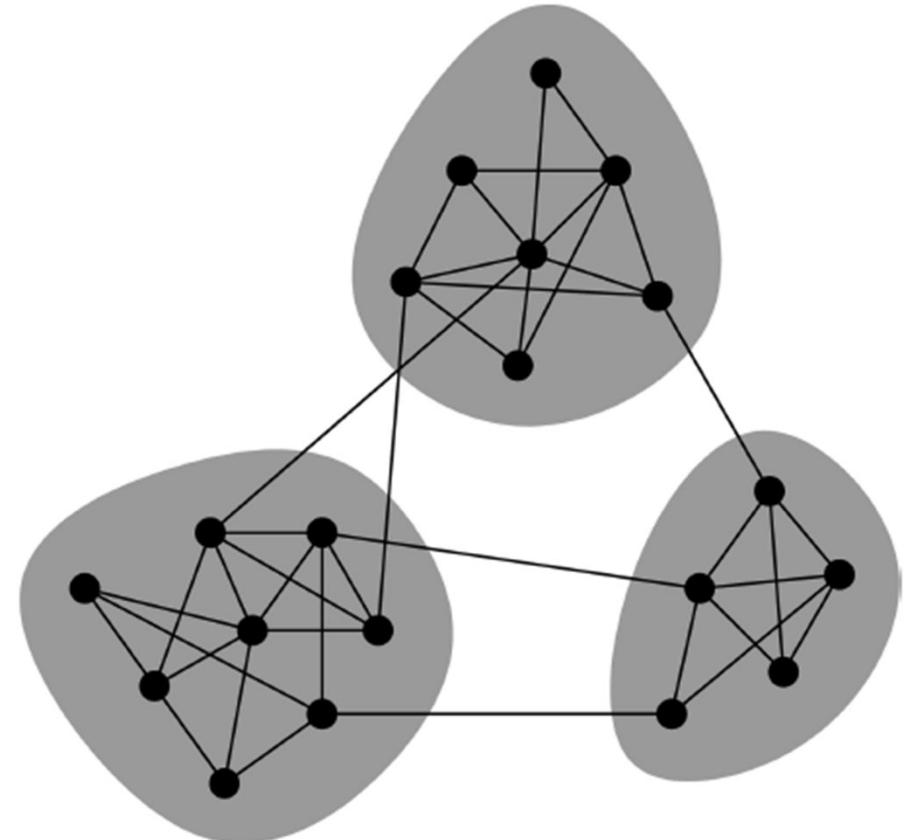


- La rimozione di archi in base al loro neighborhood overlapping causa la disconnessione della rete prima!



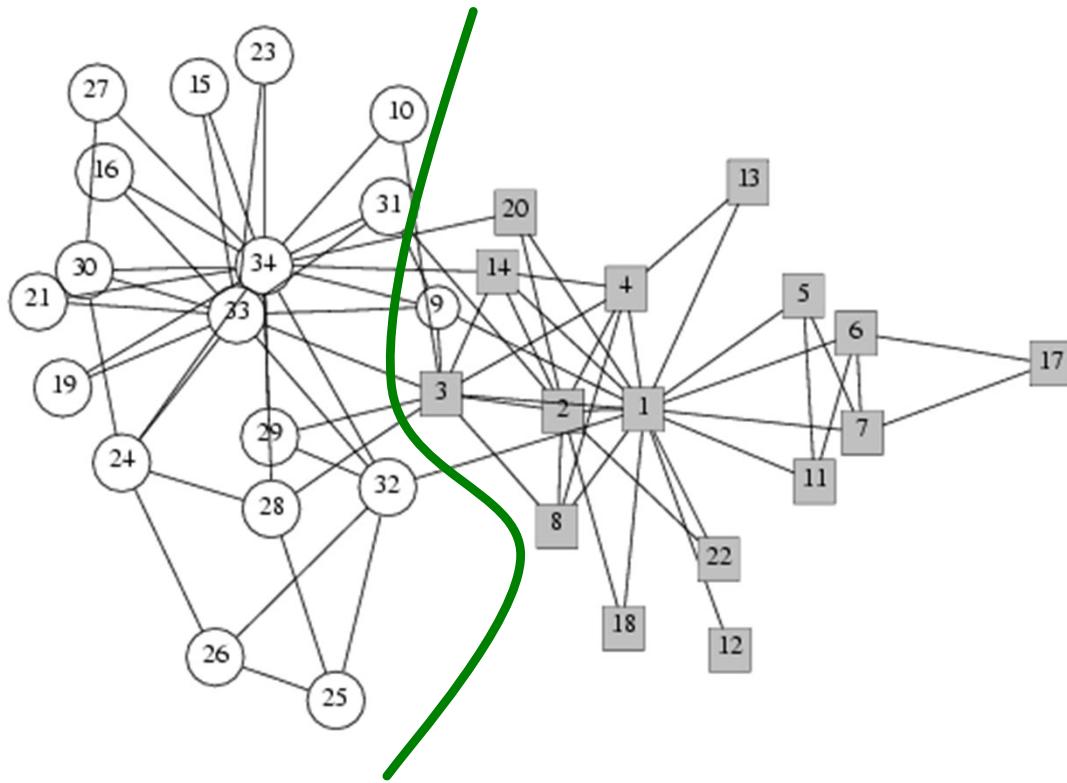
- La teoria di Granovetter suggerisce che le network sono composte da insiemi di nodi **molto connessi tra loro**.
- **Comunità**:
  - Insieme di nodi con **tante** connessioni all'interno e **poche** connessioni all'esterno (resto della rete)

**Comunità, cluster,  
gruppi, moduli**

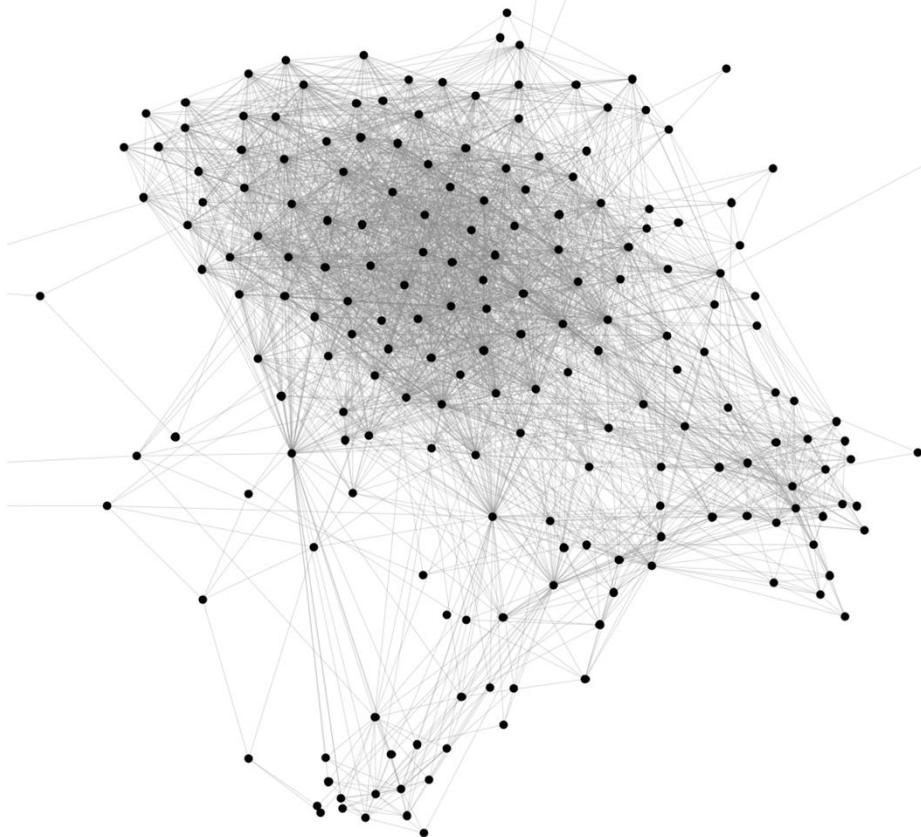


# Identificare le comunità

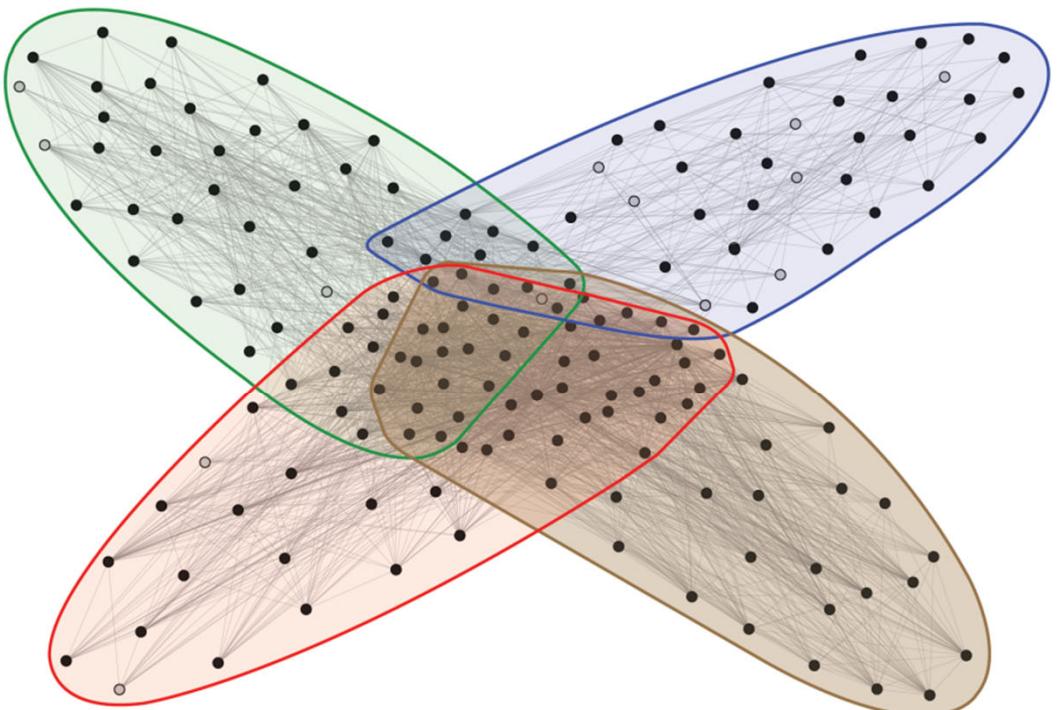
- Come identificare i gruppi di nodi che sono connessi in modo denso?
- Idealmente i cluster identificati automaticamente dovrebbero essere gruppi reali.
- Esempio:
- Zachary Karate club Net



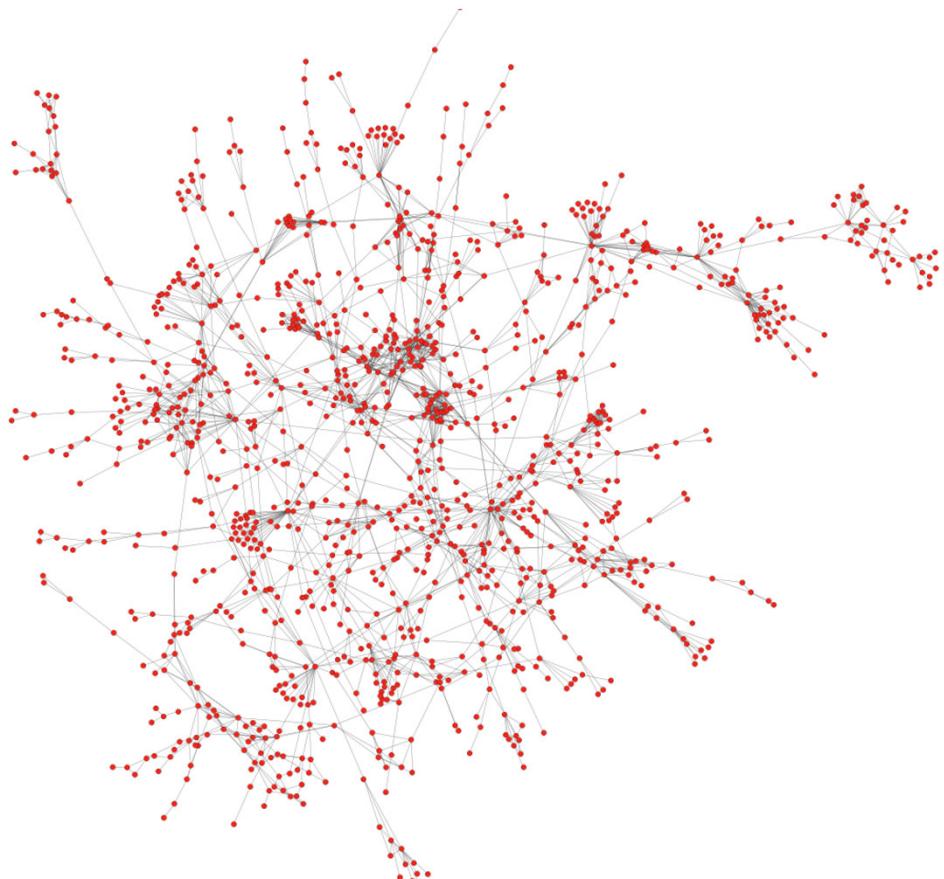
# Facebook Ego-network



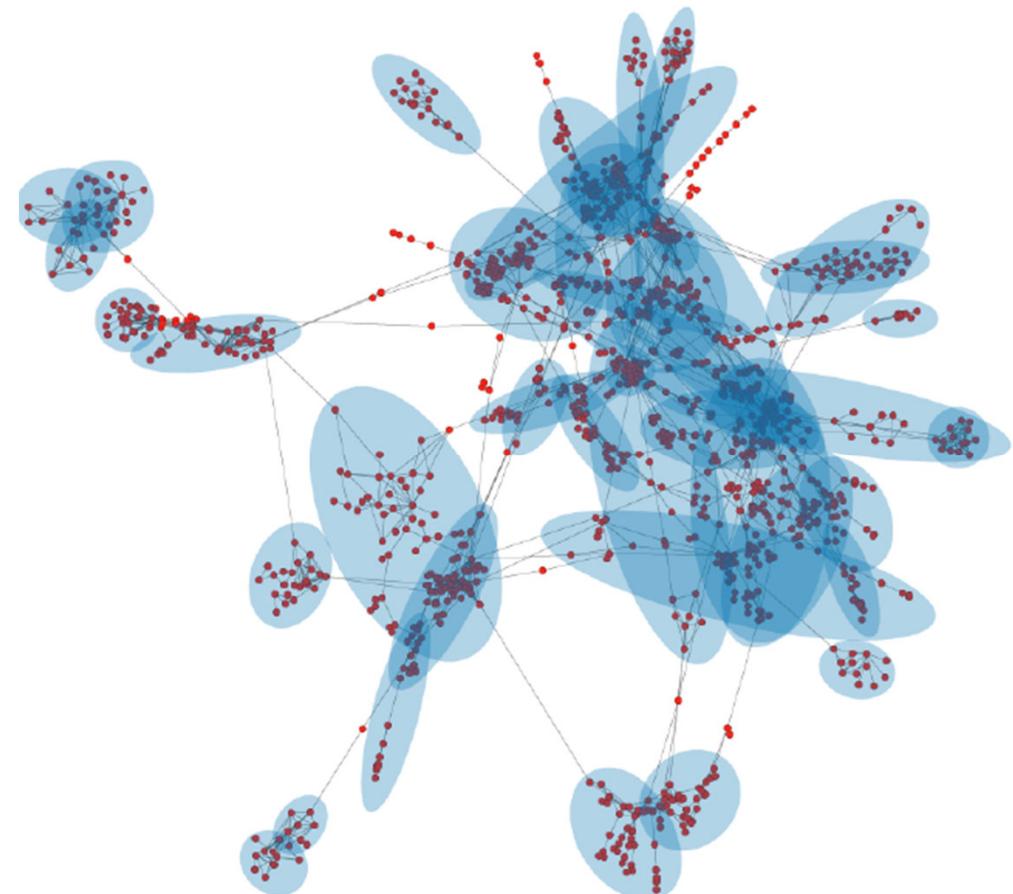
Possiamo identificare comunità Sociali?



# Protein-Protein interaction Net

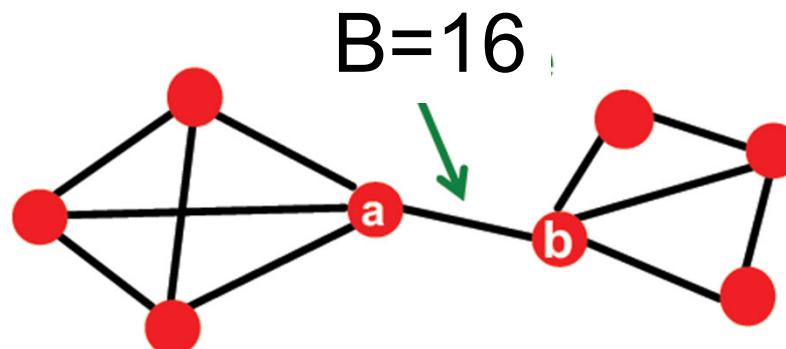


Possiamo identificare moduli funzionali?



# Algortimi per la Community Detection

- Come identificare le community?
  - Lavoriamo su grafi non diretti non pesati
  - **Edge betweenness:** Numero di shortest path che passano da un arco



- L'edge betweenness è una misura che quantifica l'importanza di un arco all'interno di una rete, in termini di numero di cammini minimi che attraversano quell'arco. Sia  $e \in E$  di  $G = (V, E)$

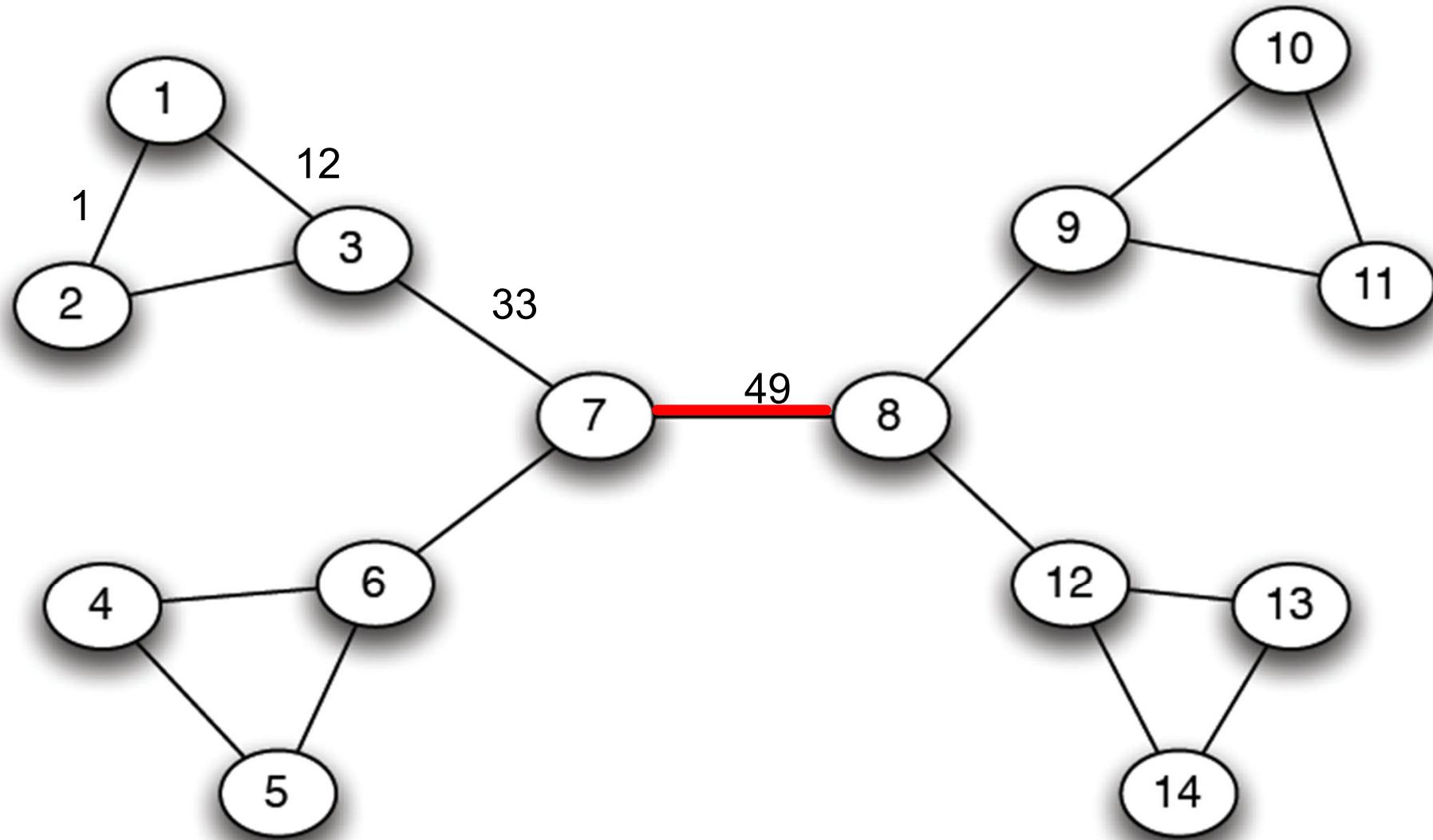
$$EB(e) = \frac{\sum_{s \neq t \in V} \sigma_{st}(e)}{\sigma_{st}}$$

- $\sigma_{st}$  numero totale di cammini minimi tra s e t
- $\sigma_{st}(e)$  numero totale di cammini minimi tra s e t che passano per l'arco e

# Algoritmo di Girvan-Newman

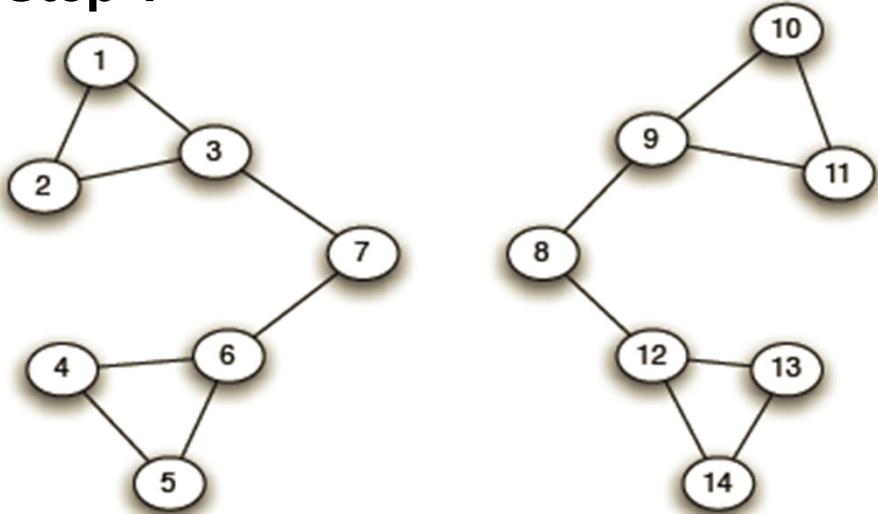
- Algoritmo di clustering gerarchico basato sulla nozione di edge **betweenness**
- **Girvan-Newman**
  - Grafo non diretto non pesato
- **Ripeti fino a quando non ci sono archi:**
  - Calcola la betweenness degli archi
  - Rimuovi gli archi con più alta betweenness
- Le componenti connesse sono le community
- Restituisce una decomposizione gerarchica della rete

# Esempio

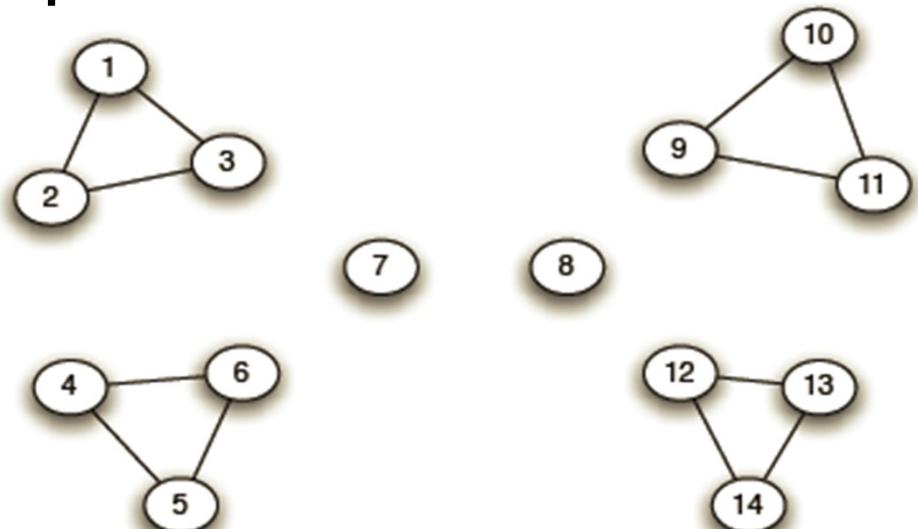


Bisogna ricalcolare la betweenss ad ogni step

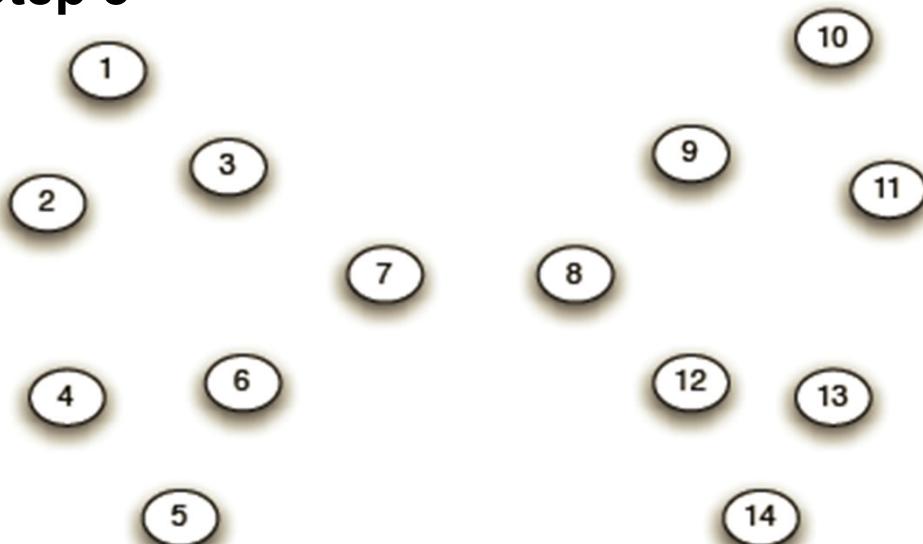
### Step 1



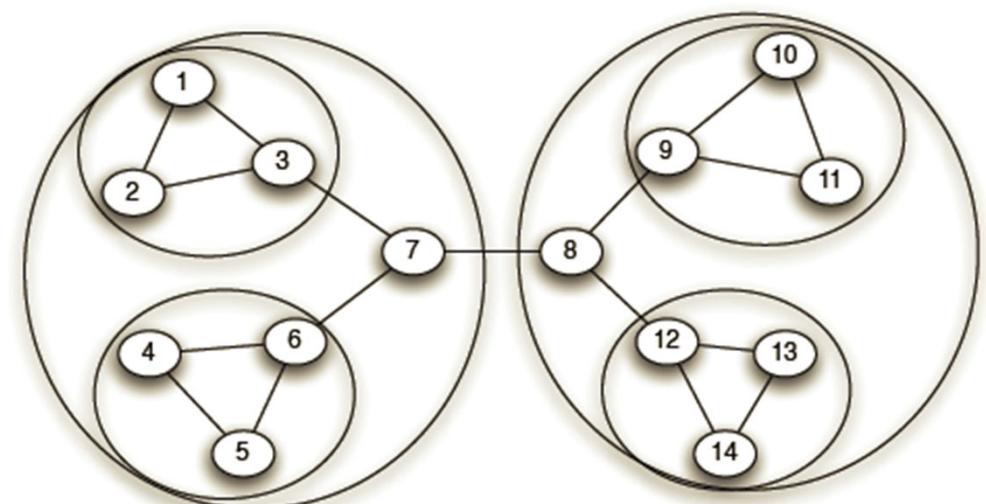
### Step 2



### Step 3



### Clustering gerarchico della rete

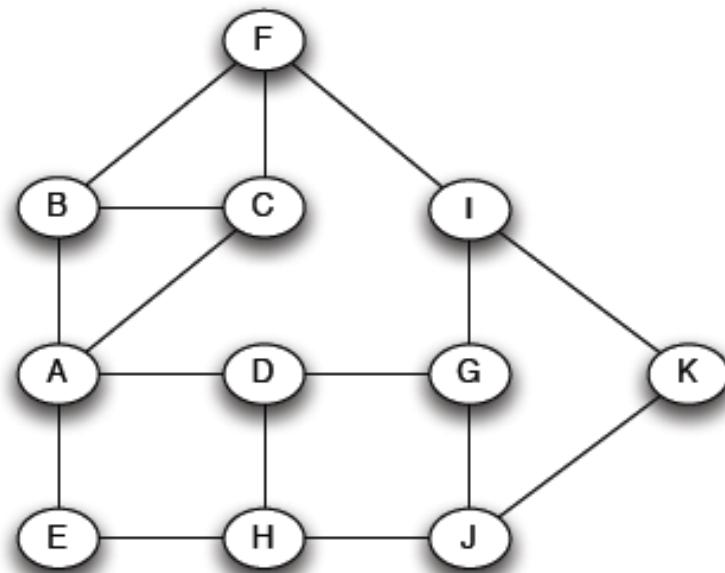


# Due punti da risolvere:

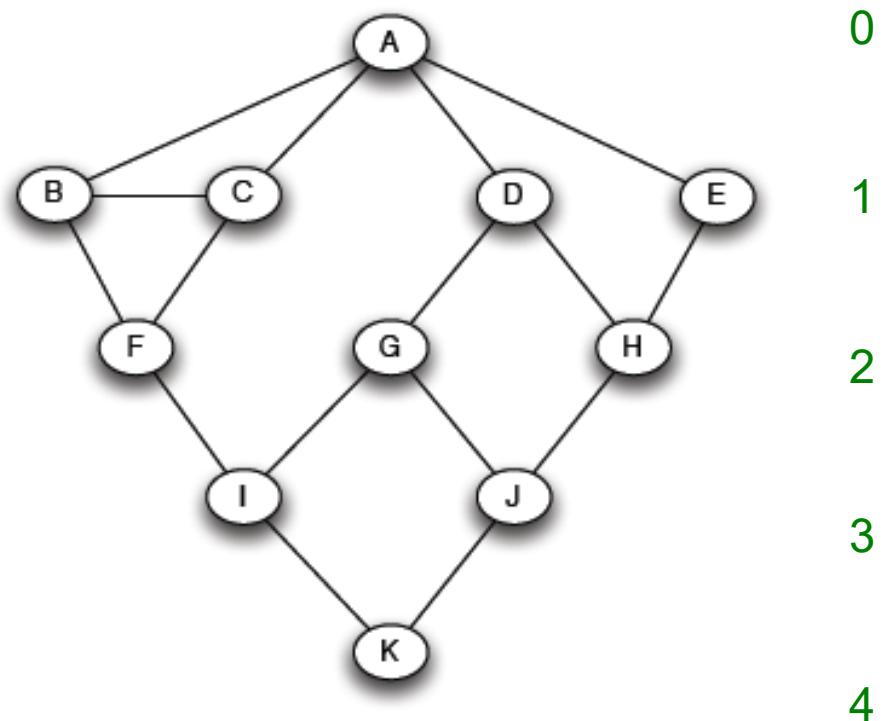
1. Come calcoliamo le betweenness?
2. Come selezioniamo il numero di cluster?

# Come calcolare la Betweenness?

- Vogliamo calcolare la betweenness dei path che partono dal nodo A



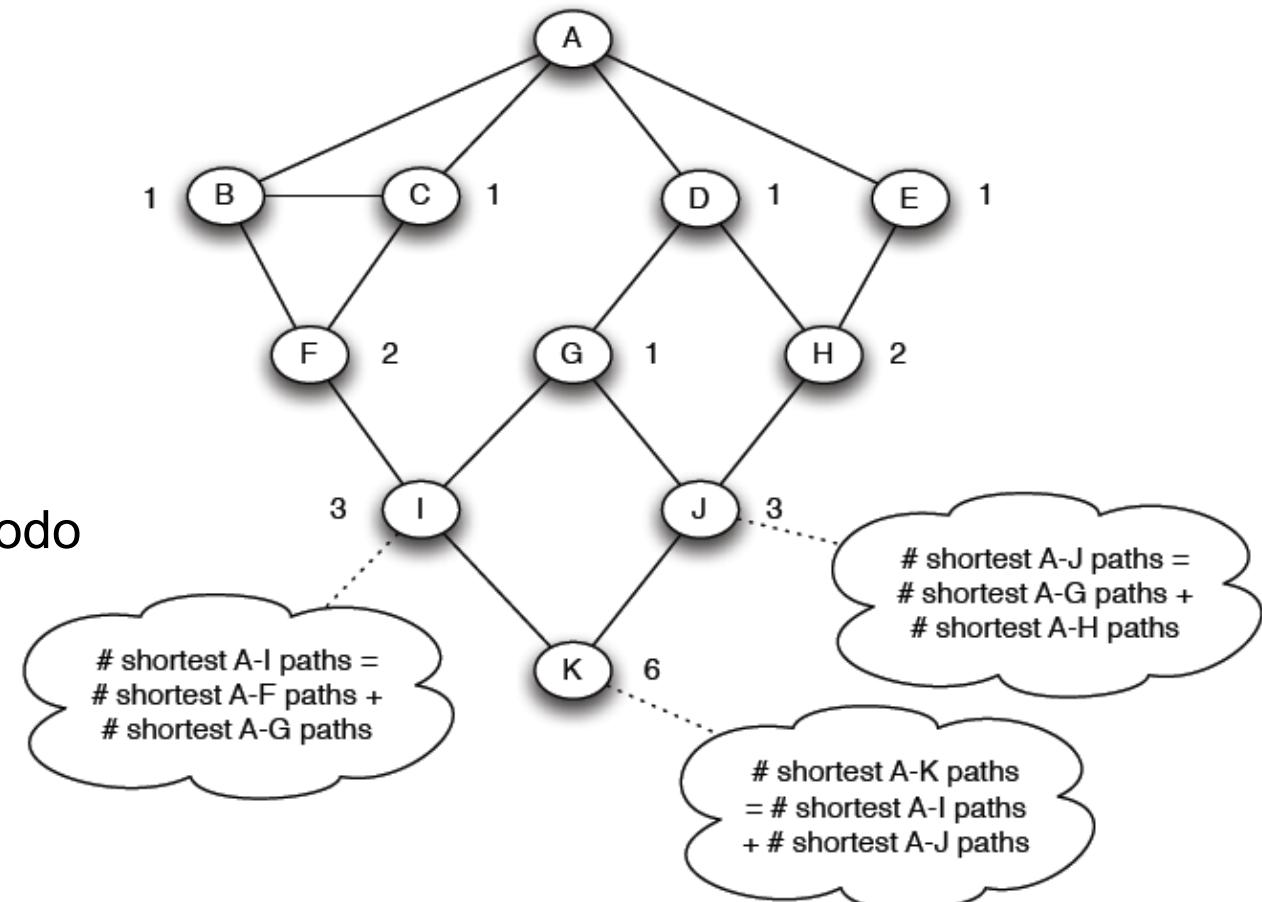
- Breath first search dal nodo A:



# Come calcolare la Betweenness?

- Contare il numero di shortest path da A a tutti gli altri nodi nella network:

Il numero di shortest path ad ogni altro nodo è la **somma** del numero di shortest path a tutti gli altri nodi direttamente sopra il nodo



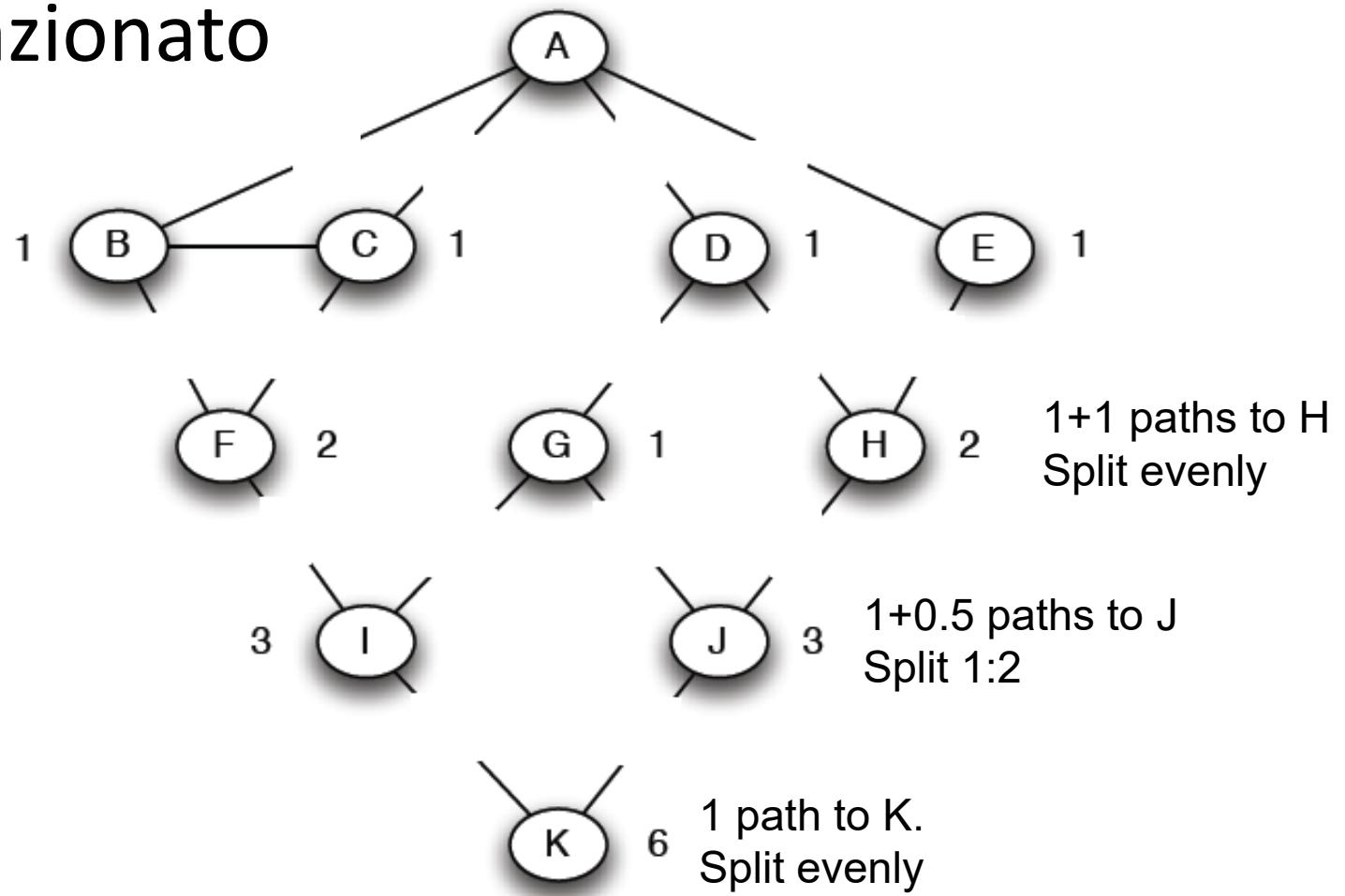
# Come calcolare la Betweenness?

## ■ Calcolare la betweenness lavorando

**sull'albero:** Se ci sono più path contare questi in modo frazionato

### Algoritmo:

- edge flow:
  - node flow =  
 $1 + \sum \text{archi figlio}$
  - dividi il flusso in base al valore del genitore
- Ripeti la BFS per ogni nodo  $U$



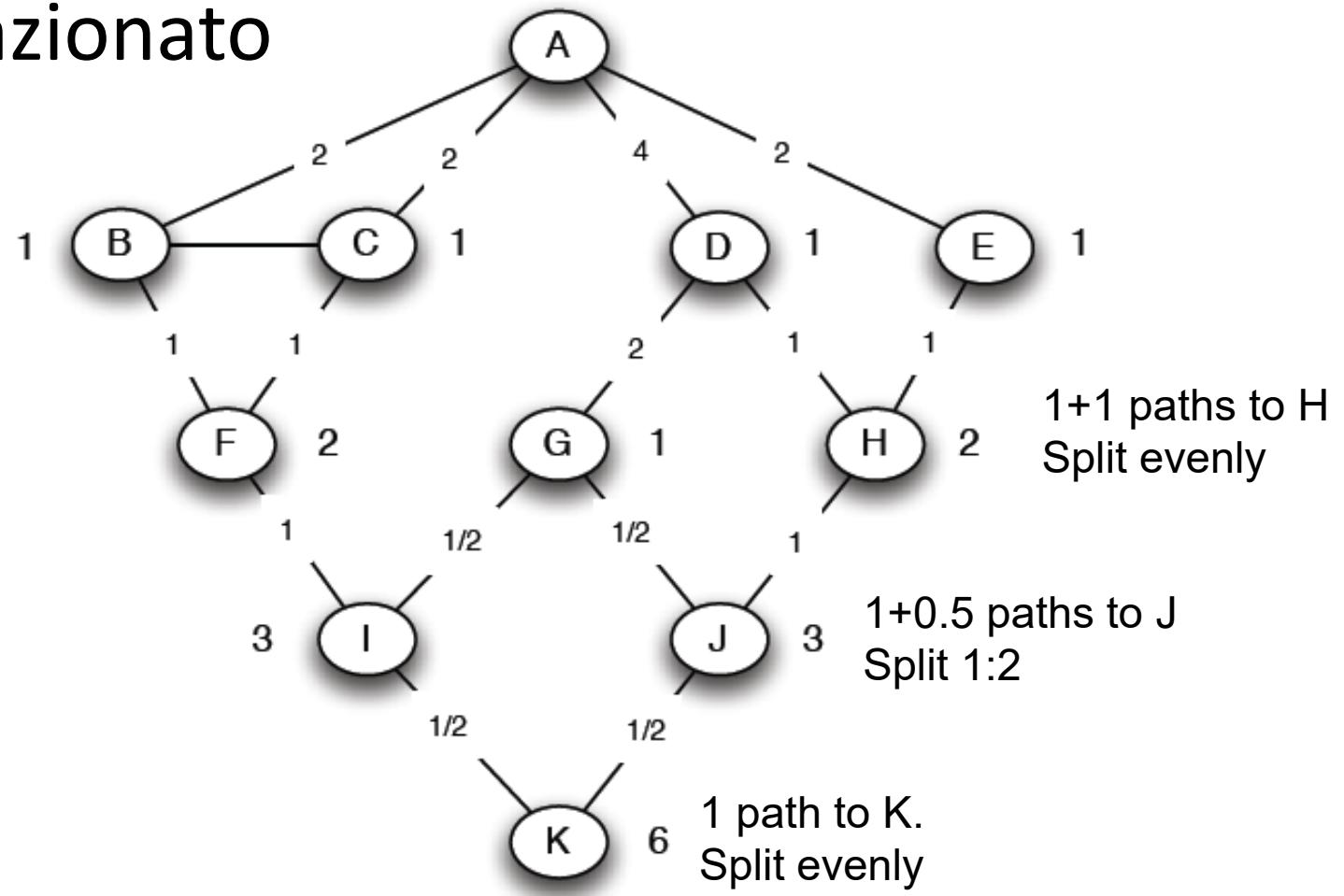
# Come calcolare la Betweenness?

## ■ Calcolare la betweenness lavorando

**sull'albero:** Se ci sono più path contare questi in modo frazionato

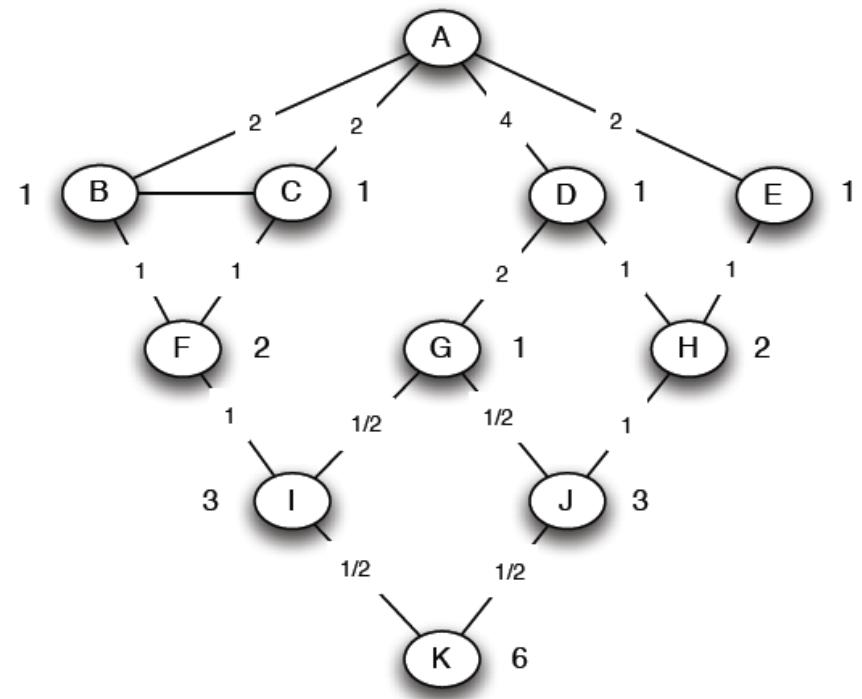
### Algoritmo:

- edge flow:
  - node flow =  
 $1 + \sum \text{archi figlio}$
  - splitta il flusso in base al valore del genitore
- Ripeti la BFS per ogni nodo  $U$



# Metodo

- Procedi **bottm up**
- Ad ogni nodo X
  - aggiungere tutto il flusso che arriva **dagli** archi direttamente **sotto** X, più 1 per il flusso destinato per X stesso
  - **Dividi** questo per gli archi che conducono verso l'alto da X, in **proporzione** al **numero di shortest path** che vengono da ciascuno



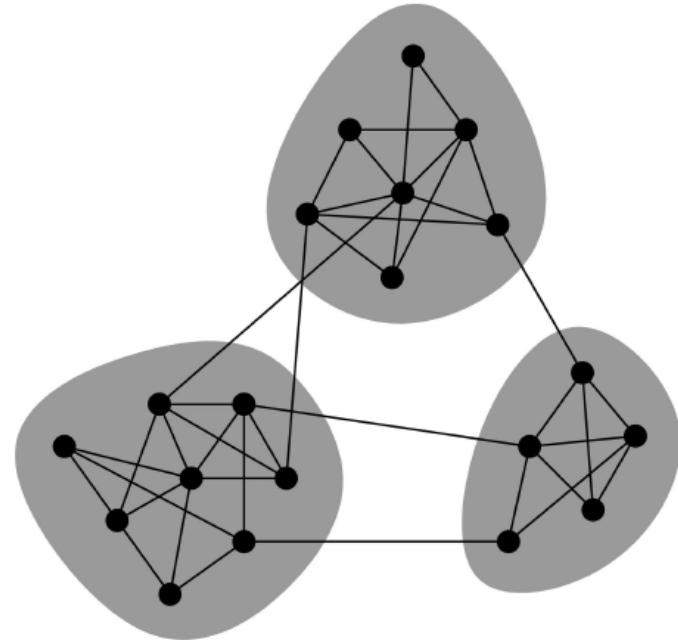
# Due punti da risolvere:

1. Come calcoliamo le betweenness?
2. Come selezioniamo il numero di cluster?

# Network Community

- **Community:** set di nodi **connessi in modo forte**
- Definizione: **Modularità  $Q$** 
  - Una misura di come la rete è ben partizionata in comunità
  - Data la partizione della rete in gruppi  $s \in S$ :

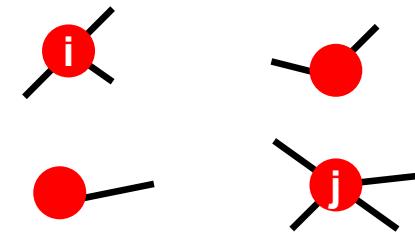
$$Q \propto \sum_{s \in S} [ (\# \text{ edges within group } s) - \underbrace{(\text{expected } \# \text{ edges within group } s)}_{\text{Necessità di un modello nullo!}} ]$$



# Null Model: Configuration Model

- Data una rete reale  $G$  con  $n$  nodi ed  $m$  archi, costruire una rete random  $G'$

- Stessa degree distribution ma connessioni random



- Considerare  $G'$  come un multigrafo

- Il numero di archi atteso tra due nodi

$i$  e  $j$  di degree  $k_i$  e  $k_j$  è uguale a:  $k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$

- Il numero atteso di archi in (multigrafo)  $G'$ :

- $= \frac{1}{2} \sum_{i \in N} \sum_{j \in N} \frac{k_i k_j}{2m} = \frac{1}{2} \cdot \frac{1}{2m} \sum_{i \in N} k_i (\sum_{j \in N} k_j) =$

- $= \frac{1}{4m} 2m \cdot 2m = m$

Nota:  
 $\sum_{u \in N} k_u = 2m$

# Modularità

## ■ Modularità della partizione $S$ di $G$ :

- $Q \propto \sum_{s \in S} [ (\# \text{ edges within group } s) - (\text{expected } \# \text{ edges within group } s) ]$

- $$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left( A_{ij} - \frac{k_i k_j}{2m} \right)$$

Costo normalizzazione:  $-1 < Q < 1$

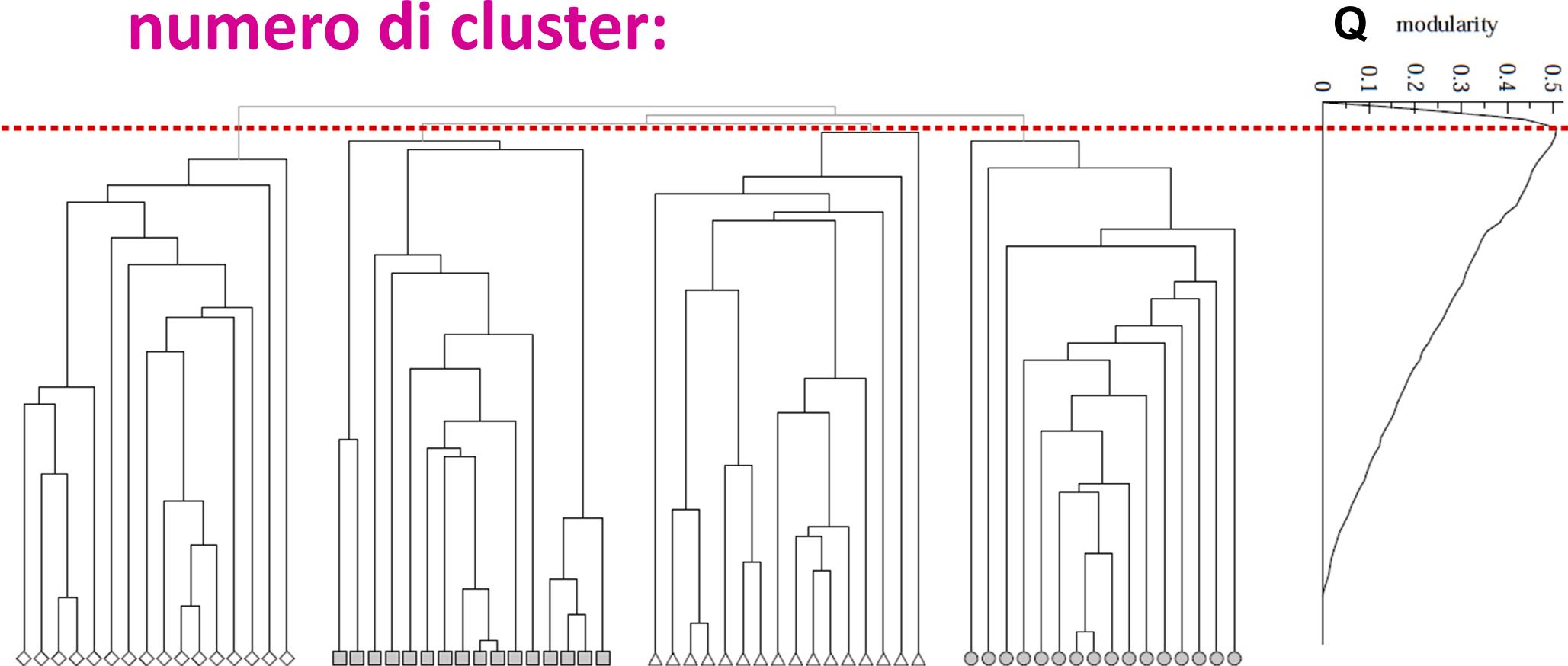
$A_{ij} = 1$  if  $i \rightarrow j$ ,  
0 else

## ■ Modularità assume valori nel range $[-1,1]$

- E' positiva se il numero di archi nel gruppo eccede il valore atteso
- $0.3 < Q < 0.7$  indica una significativa community structure

# Modularità: Numero di cluster

- La modularità è utile per selezionare il numero di cluster:



Si può ottimizzare la modularità in maniera diretta?

# Metodo 2: ottimizzazione della Modularità

- Supponiamo di voler splittare il grafo in 2 comunità
- Ottimizziamo direttamente la modularità

$$\max_S Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left( A_{ij} - \frac{k_i k_j}{2m} \right)$$

- Definiamo un vettore di appartenenza alla community  $s$

- $s_i = 1$  se il nodo è nella comunità 1  
 $= -1$  se il nodo  $i$  è nella comunità -1

$$\frac{(s_i s_j + 1)}{2} = 1 \text{ se } s_i = s_j \\ \text{o altrimenti}$$

- $$Q(G, s) = \frac{1}{2m} \sum_{i \in N} \sum_{j \in N} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \frac{(s_i s_j + 1)}{2} = \\ \frac{1}{4m} \sum_{i, j \in N} \left( A_{ij} - \frac{k_i k_j}{2m} \right) s_i s_j$$

# Matrice di modularità

## ■ Definizione

- **Matrice di modularità:**  $B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$
- **Appartenenza:**  $s=\{-1,+1\}$
- **Ne segue:**
  - $$Q(G, s) = \frac{1}{2m} \sum_{i \in N} \sum_{j \in N} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \frac{(s_i s_j + 1)}{2} = \frac{1}{4m} \sum_{i,j \in N} (B_{ij}) s_i s_j$$
  - $= \frac{1}{4m} \sum_i s_i \sum_j B_{ij} s_j = \frac{1}{4m} s^T B s$
  - **Task:** Trovare  $s$  in  $\{-1,1\}^n$  che massimizza  $Q(G,s)$

- Se la matrice A è simmetrica
  - è semidefinita positiva  $A = U \lambda U^T$
- Consideriamo le soluzioni  $\lambda, x$  all'equazione  $A x = \lambda x$ 
  - Autovettori  $x_i$  ordinati in maniera decrescente rispetto al proprio autovalore  $\lambda_i$
  - $x_i$  ortonormali e formano un sistema di coordinate
  - Se A è semidefinita positiva gli autovalori  $\lambda_i$  sono positivi
- **Teorema dell'autodecomposizione:** Possiamo riscrivere la matrice A in termini dei suoi autovettori e autovalori:

$$A = X \Lambda X' = \sum_i x_i \lambda_i x_i^T$$

- Riscriviamo  $Q(G, s) = \frac{1}{4m} s^T B s$  **in termini dei suoi autovettori e autovalori**

$$s^T \left[ \sum_{i=1}^n x_i \lambda_i x_i^T \right] s = \sum_{i=1}^n s^T x_i \lambda_i x_i^T s = \sum_{i=1}^n (\mathbf{s^T x_i})^2 \lambda_i$$

- Se non ci sono altri vincoli su  $s$  per massimizzare  $Q$  dobbiamo rendere  $s = x_n$ 
  - Perché gli autovalori sono ordinati dal più grande al più piccolo
  - Tutti gli altri termini  $s^T x_i$  valgono 0 per l'ortonormalità.

# Come trovare il vettore s

- Consideriamo solo il primo termine della somma:

$$\max_s Q(G, s) = \sum_{i=1}^n (s^T x_i)^2 \lambda_i \approx (s^T x_n)^2 \lambda_n$$

Massimizzare  $\sum_{j=1}^n s_j \times x_{n,j}$  dove  $s_j \in \{-1, +1\}$

Settiamo:

$$s_j = \begin{cases} +1 & \text{se } x_{n,j} \geq 0 \\ -1 & \text{se } x_{n,j} < 0 \end{cases}$$

Continuiamo la bisezione in modo gerarchico

- Algoritmo veloce per l'ottimizzazione della modularità
  - Trovare l'autovettore  $x_n$  di modularità di  $B$
  - Dividere i nodi in base al segno degli elementi di  $x_n$
  - Ripetere gerarchicamente fino:
    - A quando lo split non causa un incremento della modularità e si restituisce un solo cluster
    - Tutte le comunità non sono piu' divisibili
  - **Per trovare  $x_n$  usiamo il Power method!!**

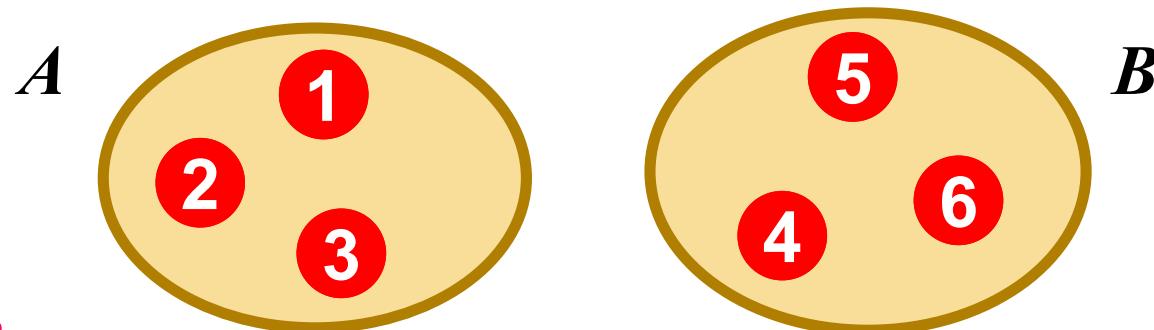
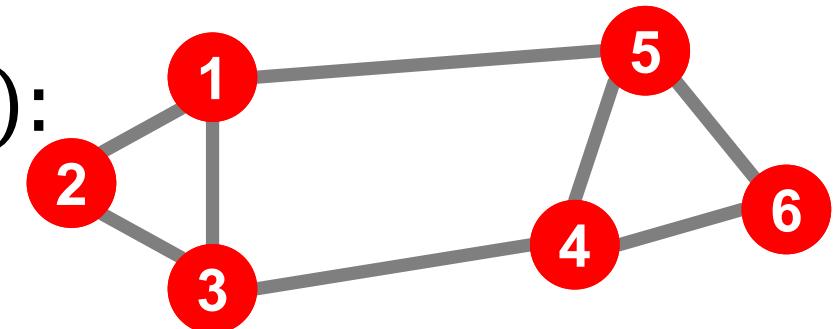
# Modularità

- **Girvan-Newman:**
  - Basato sul concetto di connessioni deboli e forti
- **Modularità**
  - Usato per determinare il numero di comunità
- **Algoritmo veloce per l'ottimizzazione della modularità**
  - Trasformare il problema della modularità in ricerca degli autovalori

# Spectral Clustering

# Graph Partitioning

- Grafo non direzionato  $G(V, E)$ :
- Bi-partitioning task:
  - Dividere i vertici in due gruppi disgiunti  $A, B$

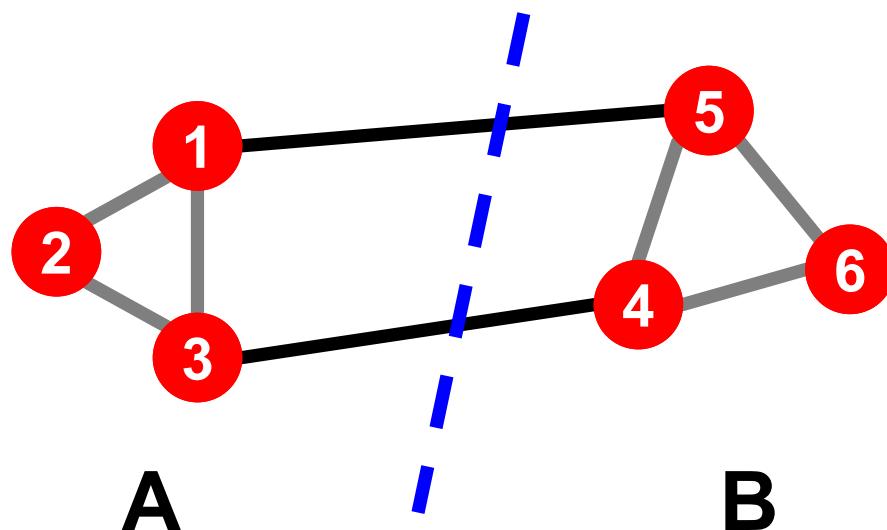


- Domande:
  - Come definiamo una “buona” partizione di  $G$ ?
  - Come possiamo trovare tale partizione in modo efficiente?

# Graph Partitioning

## ■ Cosa rende buona una partizione?

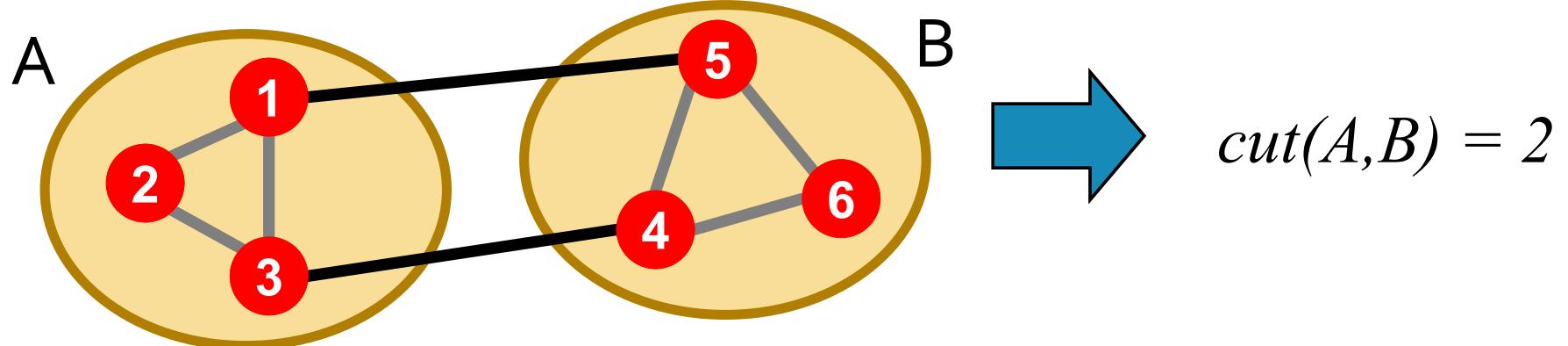
- Massimizzare il numero di connessioni within-group (intra gruppo)
- Minimizzare il numero di connessioni between-group (inter gruppo)



# Graph Cut

- Esprimere l'obiettivo della partizione come una funzione dell’“edge cut” della partizione
- **Cut:** Insieme di archi con un vertice in A e l'altro in B

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$



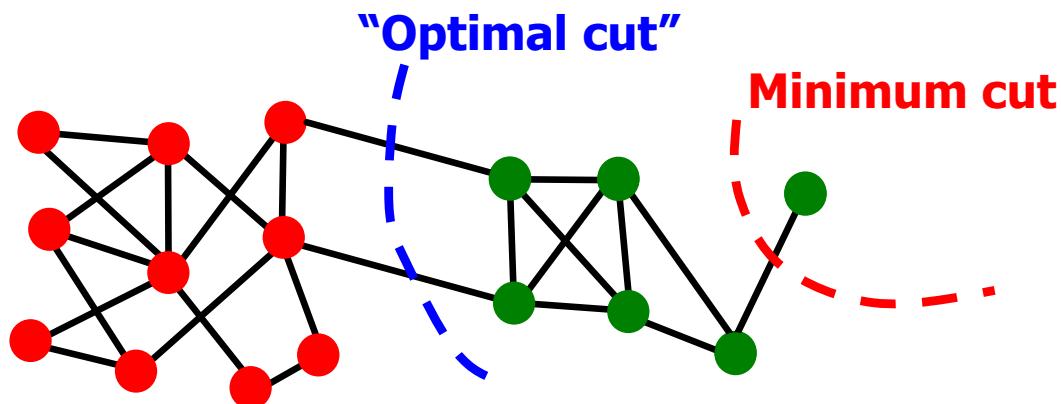
# Graph Cut Criterio

## ■ Criterio: Minimum-cut

- Minimizzare il peso delle connessioni intergruppo

$$\arg \min_{A,B} cut(A,B)$$

## ■ Caso limite:



## ■ Problema:

- Considera solo le connessioni esterne
- Non considera la connettività interna

# Graph Cut Criterio

## ■ Criterio: Normalized-cut [Shi-Malik, '97]

- Connattività tra i due gruppi relativa alla densità dei gruppi

$$ncut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(A, B)}{vol(B)}$$

$vol(A)$ : peso totale degli archi con almeno un endpoint in  $A$ :  $vol(A) = \sum_{i \in A} k_i$

## ■ Perché usare questo criterio?

- Produce partizioni bilanciate

## ■ Come trovare una buona partizione?

- Problema: Trovare il taglio ottimale è un problema NP-hard

# Spectral Graph Partitioning

- $A$ : Matrice di Adiacenza di  $\mathbf{G}$ 
  - $A_{ij} = 1$  se  $(i, j)$  è un arco,  $0$  altrimenti
- $x$  vettore in  $\mathbb{R}^n$  con  $(x_1, \dots, x_n)$  componenti
  - Pensiamolo come etichette/valori di ogni nodo in  $G$
- **Cosa significa  $A \cdot x$ ?**

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad y_i = \sum_{j=1}^n A_{ij} x_j = \sum_{(i,j) \in E} x_j$$

- **Entry  $y_i$  somma di tutte le etichette  $x_j$  dei vicini di  $i$**

# Cosa significa $Ax$ ?

## ■ $j^{th}$ coordinata di $A \cdot x$ :

- Somma dei valori  $x$  dei vicini di  $j$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- Nuovo valore per  $j$

## ■ Spectral Graph Theory:

$$A \cdot x = \lambda \cdot x$$

- Analizzare lo “spettro” della matrice che rappresenta  $G$

- **Spettro:** Autovettori  $x_i$  del grafo, ordinati per la magnitudo (forza) dei corrispondenti autovalori  $\lambda_i$ :

$$\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$$

$$\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$$

# Intuizione geometrica

- Gli **autovettori** definiscono le "direzioni" in cui il grafo può essere "separato" con una decomposizione minima della struttura.
- Gli **autovalori** misurano la "forza" o l'"importanza" di queste direzioni.
- **Nel clustering:**
  - È probabile che i nodi con i componenti degli autovettori simili appartengano allo stesso cluster.

# Esempio: d-regular graph

- Supponiamo che in  $G$  tutti i nodi abbiano degree  $d$  e che  $G$  sia connesso
- **Quali sono gli autovalori/vettori of  $G$ ?**

$$A \cdot x = \lambda \cdot x \quad \lambda? x?$$

- Proviamo :  $x = (1, 1, \dots, 1)$
- Quindi:  $A \cdot x = (d, d, \dots, d) = \lambda \cdot x$ . segue:  $\lambda = d$
- Trovata la coppia di  $G$ :  $x = (1, 1, \dots, 1)$ ,  $\lambda = d$

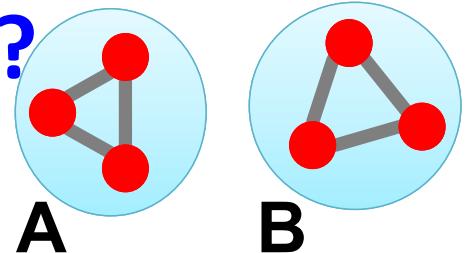
$$y = A \cdot x:$$

$$y_i = \sum_{j=1}^n A_{ij}x_j = \sum_{(i,j) \in E} x_j$$

# Esempio: Grafo con due componenti

## ■ Cosa succede se $G$ non è连通的?

- $G$  ha 2 componenti , ognuna  $d$ -regular



## ■ Gli autovettori?

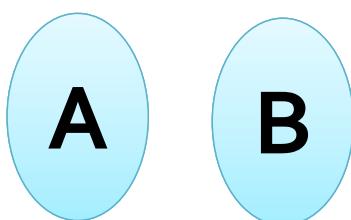
- $x = 1s$  su  $A$  and  $0$  su  $B$  o vice versa

- $x' = (\underbrace{1, \dots, 1}_{|A|}, \underbrace{0, \dots, 0}_{|B|})$  quindi  $A \cdot x' = (d, \dots, d, 0, \dots, 0)$

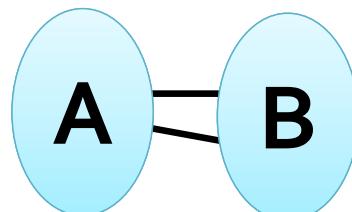
- $x'' = (0, \dots, 0, 1, \dots, 1)_{|B|}$  quindi  $A \cdot x'' = (0, \dots, 0, d, \dots, d)$

- In entrambi i casi abbiamo che  $\lambda = d$

## ■ Intuizione:



$$\lambda_n = \lambda_{n-1}$$

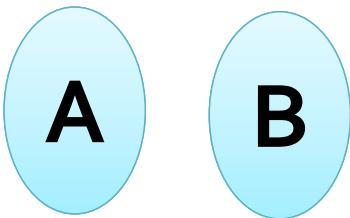


$$\lambda_n - \lambda_{n-1} \approx 0$$

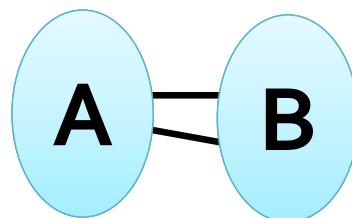
secondo  
autovalore  
 $\lambda_{n-1}$  ha valore  
molto vicino a  $\lambda_n$

# Intuizione

## ■ Ancora:



$$\lambda_n = \lambda_{n-1}$$



$$\lambda_n - \lambda_{n-1} \approx 0$$

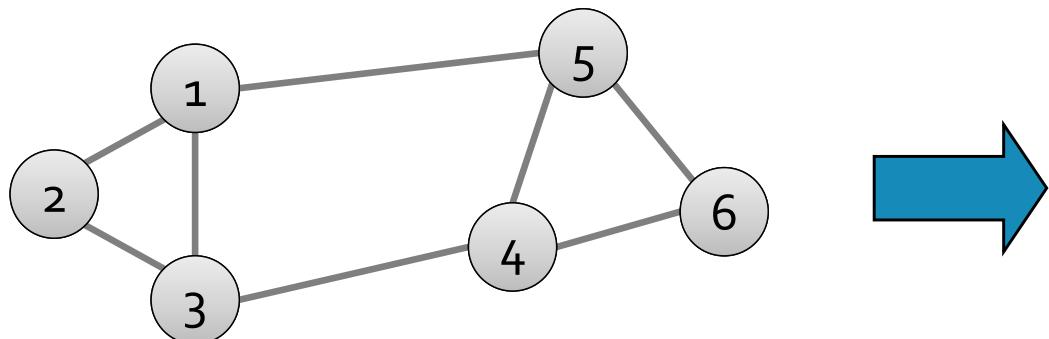
2<sup>nd</sup> secondo  
autovalore.  
 $\lambda_{n-1}$  ha valore  
molto vicino a  $\lambda_n$

- Se il grafo è connesso (destra) sappiamo che  $x_n = (1, \dots, 1)$  è un autovettore
- Gli autovettori sono ortogonali quindi  $x_n \cdot x_{n-1} = 0$ 
  - Ma  $x_n \cdot x_{n-1} = \sum_i x_n[i] \cdot x_{n-1}[i]$
- Prendiamo il 2<sup>nd</sup> autovettore e inseriamo i nodi positivi in A e quelli negativi in B.

# Rappresentazione Matriciale

## ■ Matrice di Adiacenza ( $A$ ):

- $n \times n$
- $A = [a_{ij}]$ ,  $a_{ij} = 1$  se c'è un arco tra  $i$  e  $j$



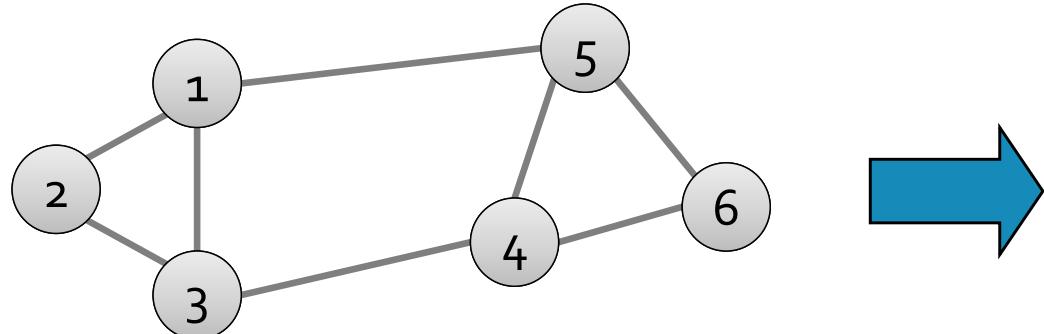
	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

## ■ Proprietà:

- Matrice simmetrica
- Autovettori reali ed ortogonali

# Rappresentazione Matriciale

- **Degree matrix (D):**
  - $n \times n$  matrice diagonale
  - $D = [d_{ii}]$ ,  $d_{ii}$  = degree del nodo  $i$

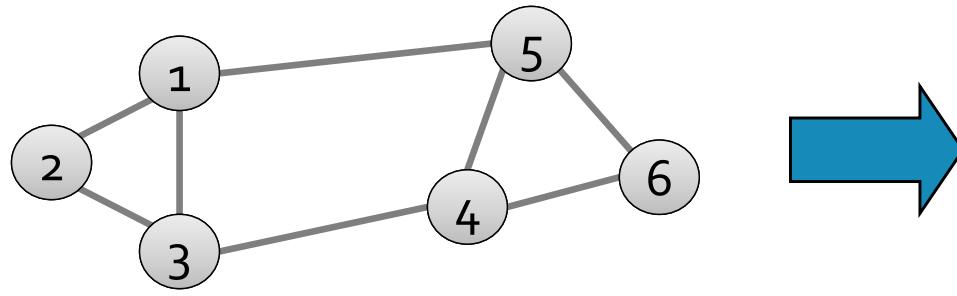


	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

# Rappresentazione Matriciale

## ■ Matrice Laplaciana ( $L$ ):

- $n \times n$  simmetrica



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

## ■ Qual è una autocoppia banale? $L = D - A$

- $x = (1, \dots, 1)$  quindi  $L \cdot x = 0$   $\lambda = \lambda_1 = 0$

## ■ Proprietà:

- **Autovalori** valori reali non negativi
- **Autovettori** Reali e ortogonali

# Matrice di Incidenza

- La matrice di Incidenza di un grafo orientato  $G$  è una matrice  $N$  di dimensione  $n \times m$ , dove  $m$  è il numero di archi del grafo, in cui si rappresentano le orientazioni degli archi nella rete:

$$N_{ij} = \begin{cases} -1 & \text{se } v_i \text{ è la coda dell'arco} \\ 1 & \text{se } v_i \text{ è la testa dell'arco} \\ 0 & \text{altrimenti} \end{cases}$$

- Consideriamo  $E=\{(1,2),(3,1)\}$

$$N^T = \begin{bmatrix} -1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}$$

- Per definire il Laplaciano possiamo utilizzare la matrice di incidenza.
- Dato un grafo non orientato  $G$ , dando un orientamento arbitrario agli archi si ha che

$$L = N^T N$$

# Proprietà del Laplaciano $L$

- (a) Tutti i suoi autovalori sono  $\geq 0$
- (b)  $x^T L x = \sum_{ij} L_{ij} x_i x_j \geq 0$  per ogni  $x$
- (c)  $L = N^T \cdot N$

- $L$  Semi-definita positiva
- **Dimostrazione:**
  - (c) $\Rightarrow$ (b):  $x^T L x = x^T N^T N x = (xN)^T (Nx) \geq 0$
  - (b) $\Rightarrow$ (a): Sia  $\lambda$  un autovalore di  $L$ . Dalla (b) ne segue  $x^T L x \geq 0$  quindi  $x^T L x = x^T \lambda x = \lambda x^T x \Rightarrow \lambda \geq 0$

# $\lambda_2$ come un problema di ottimizzazione

- Proprietà: Per una matrice simmetrica  $M$ :

$$\lambda_2 = \min_x \frac{x^T M x}{x^T x}$$

- Qual è il significato di  $x^T L x$  su  $G$ ?

- $x^T L x = \sum_{i,j=1}^n L_{ij} x_i x_j = \sum_{i,j=1}^n (D_{ij} - A_{ij}) x_i x_j$
- $= \sum_i D_{ii} x_i^2 - \sum_{(i,j) \in E} 2x_i x_j$
- $= \sum_{(i,j) \in E} (\underbrace{x_i^2 + x_j^2}_{\text{Node } i \text{ has degree } d_i} - 2x_i x_j) = \sum_{(i,j) \in E} (x_i - x_j)^2$

Nodo  $i$  ha degree  $d_i$ . Quindi , il valore  $x_i^2$  deve essere sommato  $d_i$  volte.  
Ogni arco  $(i,j)$  ha due endpoint quindi abbiamo bisogno di  $x_i^2 + x_j^2$

# $\lambda_2$ come un problema di ottimizzazione

## ■ Cos'altro sappiamo di $x$ ?

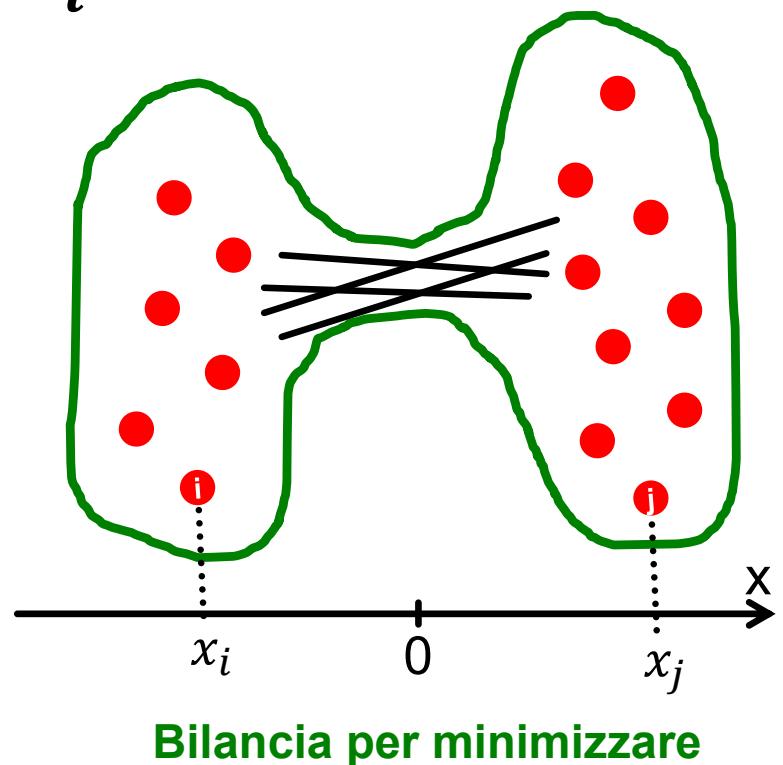
- $x$  è un vettore unità:  $\sum_i x_i^2 = 1$
- $x$  ortogonale al primo autovettore  $(1, \dots, 1)$  quindi:  $\sum_i x_i \cdot 1 = \sum_i x_i = 0$

## ■ Ricordiamo che:

$$\lambda_2 = \min \frac{\sum_{(i,j) \in E} (x_i - x_j)^2}{\sum_i x_i^2}$$

Labeling dei  
nodi  $i$  tale  
che  $\sum x_i = 0$

Vogliamo assegnare valore  $x_i$  ai nodi  $i$  tale  
che pochi archi attraversano lo 0.  
(vogliamo sottrarre  $x_i$  e  $x_j$ )



# Trovare il taglio ottimale [Fiedler'73]

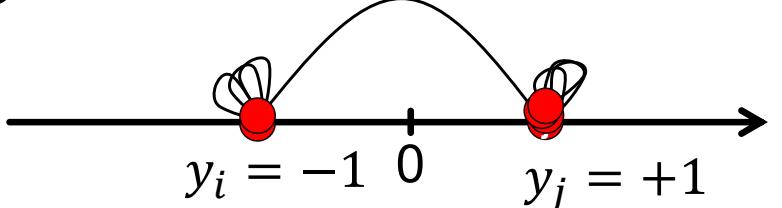
- Ritorniamo al taglio ottimale
- Esprimiamo la partizione  $(A, B)$  come un vettore

$$y_i = \begin{cases} +1 & \text{if } i \in A \\ -1 & \text{if } i \in B \end{cases}$$

- Minimizzare il taglio della partizione  
identificando un vettore non banale  $y$  che  
**minimizza**:

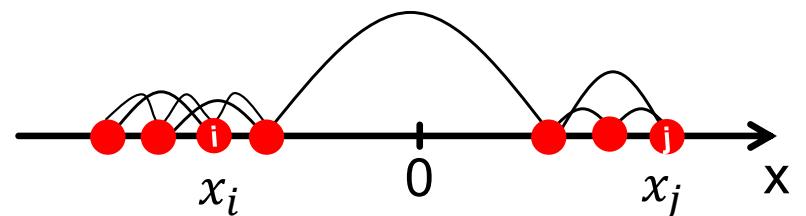
$$\arg \min_{y \in [-1, +1]^n} f(y) = \sum_{(i,j) \in E} (y_i - y_j)^2$$

Non possiamo risolverlo in modo esatto.  
Rilassiamo  $y$  ammettendo valori reali.



# Teorema di Rayleigh

$$\min_{y \in R^n} f(y) = \sum_{(i,j) \in E} (y_i - y_j)^2 = y^T L y$$



- $\lambda_2 = \min_y f(y)$ : il minimo di  $f(y)$  è dato dal secondo più piccolo autovalore  $\lambda_2$  del Laplaciano  $L$
- $\mathbf{x} = \arg \min_y f(y)$ : la soluzione ottimale per  $y$  è data dal corrispondente autovettore  $\mathbf{x}$ , chiamato **vettore di Fiedler**

# Approssimazione garantita dello Spectral Clustering

- Supponiamo di avere una partizione di  $\mathbf{G}$  in  $\mathbf{A}$  a  $\mathbf{B}$  dove  $|A| \leq |B|$ , s.t.  $\alpha = \frac{(\# \text{ edges from } A \text{ to } B)}{|A|}$  allora  $2\alpha \geq \lambda_2$ 
  - Questa approssimazione ci dice che ciò che troviamo con lo spectral clustering è al più 2 volte peggiore dello score ottimale  $\alpha$ .

## ■ Dimostrazione:

- Sia:  $a=|A|$ ,  $b=|B|$  con  $e= \# \text{ archi da } A \text{ a } B$
- Sufficiente per scegliere  $x_i$  basato su  $\mathbf{A}$  e  $\mathbf{B}$  tale che:

$$\lambda_2 \leq \underbrace{\frac{\sum (x_i - x_j)^2}{\sum_i x_i^2}}_{\lambda_2 \text{ è solo più piccolo}} \leq 2\alpha \quad (\text{inoltre } \sum_i x_i = 0)$$

# Approssimazione garantita dello Spectral Clustering

## ■ Dimostrazione (continuazione):

■ 1) Impostiamo:  $x_i = \begin{cases} -\frac{1}{a} & \text{if } i \in A \\ +\frac{1}{b} & \text{if } i \in B \end{cases}$

■ Verifichiamo che  $\sum_i x_i = 0$ :  $a\left(-\frac{1}{a}\right) + b\left(\frac{1}{b}\right) = 0$

■ 2) Allora:  $\frac{\sum(x_i - x_j)^2}{\sum_i x_i^2} = \frac{\sum_{i \in A, j \in B} \left(\frac{1}{b} + \frac{1}{a}\right)^2}{a\left(-\frac{1}{a}\right)^2 + b\left(\frac{1}{b}\right)^2} = \frac{e \cdot \left(\frac{1}{a} + \frac{1}{b}\right)^2}{\frac{1}{a} + \frac{1}{b}} =$

$$e \left(\frac{1}{a} + \frac{1}{b}\right) \leq e \left(\frac{1}{a} + \frac{1}{a}\right) \leq e \frac{2}{a} = 2\alpha$$

e ... numero di archi tra A e B

Costo dello spectral  
Clustering è due volte del  
costo OPT

# Approssimazione garantita dello Spectral Clustering

## ■ Mettendo tutto assieme:

$$2\alpha \geq \lambda_2 \geq \frac{\alpha^2}{2k_{max}}$$

- dove  $k_{max}$  max degree nel grafo
  - Dimostrata solo la 1<sup>st</sup> parte:  $2\alpha \geq \lambda_2$
  - Non  $\lambda_2 \geq \frac{\alpha^2}{2k_{max}}$
- $\lambda_2$  da sempre un buon bound

# Fino ad ora...

- **Come definire una “buona” partizione del grafo?**
  - Minimizzare un criterio di graph cut
- **Come trovare tale partizione in modo efficiente?**
  - Approssimazione basata su autovettori e autovalori del grafo
- **Spectral Clustering**

# Algoritmi per lo Spectral Clustering

## ■ Tre strategie di base:

### ■ 1) Pre-processing

- Costruire una matrice che rappresenta il grafo

### ■ 2) Decomposizione

- Calcolare autovalori e autovettori della matrice
- Mappare ogni punto a una rappresentazione a dimensione più bassa basata su uno o più autovettori

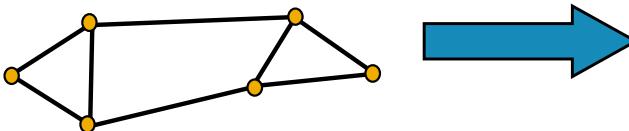
### ■ 3) Grouping

- Assegnare i punti a due o più cluster usando questa nuova rappresentazione

# Algoritmo Spectral Partitioning

## ■ 1) Pre-processing:

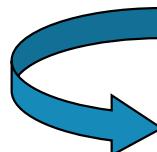
- Costruire la matrice Laplaciana  $L$  del grafo



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

## ■ 2) Decomposizione:

- Identificare gli autovalori  $\lambda$  e gli autovettori  $x$  di  $L$



0.0
1.0
3.0
3.0
4.0
5.0

X =

0.4	0.3	-0.5	-0.2	-0.4	-0.5
0.4	0.6	0.4	-0.4	0.4	0.0
0.4	0.3	0.1	0.6	-0.4	0.5
0.4	-0.3	0.1	0.6	0.4	-0.5
0.4	-0.3	-0.5	-0.2	0.4	0.5
0.4	-0.6	0.4	-0.4	-0.4	0.0

- Mappare ogni vertice alle componenti corrispondenti di  $\lambda_2$

1	0.3
2	0.6
3	0.3
4	-0.3
5	-0.3
6	-0.6



Come troviamo i cluster?

# Spectral Partitioning

## ■ 3) Grouping:

- Ordinare i punti 1-dimensionali del vettore

- Identificare i cluster spittando in due

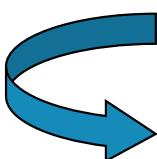
## ■ Come scegliere il punto di split?

- Approccio Naïve :

- Split a 0 oppure al valore mediano

- Approcci più sofisticati:

- Cercare di minimizzare il cut normalizzato a 1-dimensione  
Scorrere l'ordine indotto dei nodi



1	0.3
2	0.6
3	0.3
4	-0.3
5	-0.3
6	-0.6



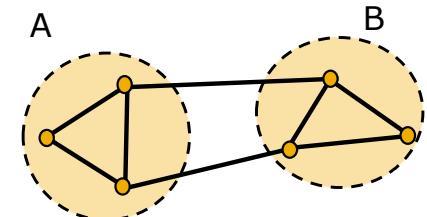
**Split a 0:**

**Cluster A:** Punti Positivi

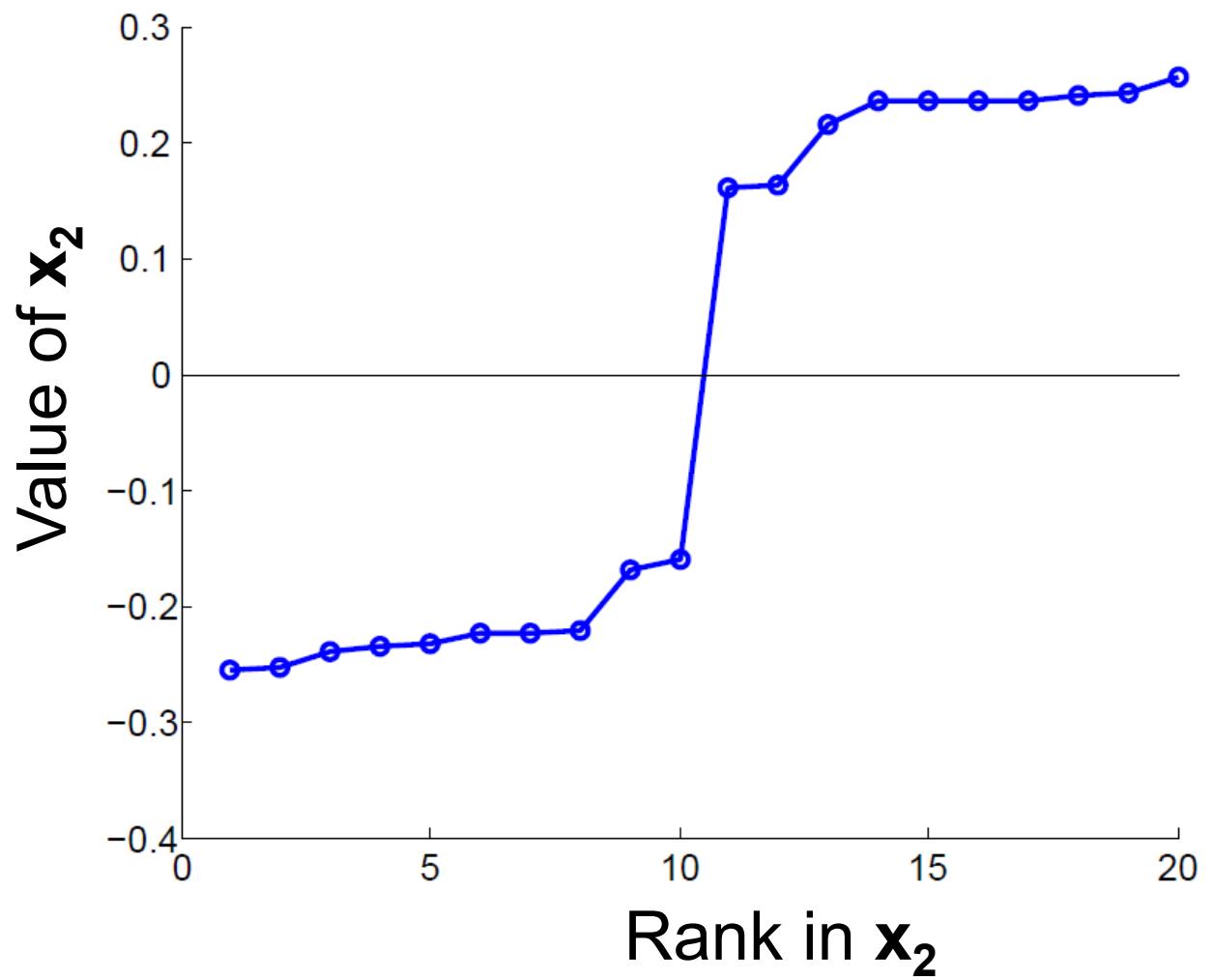
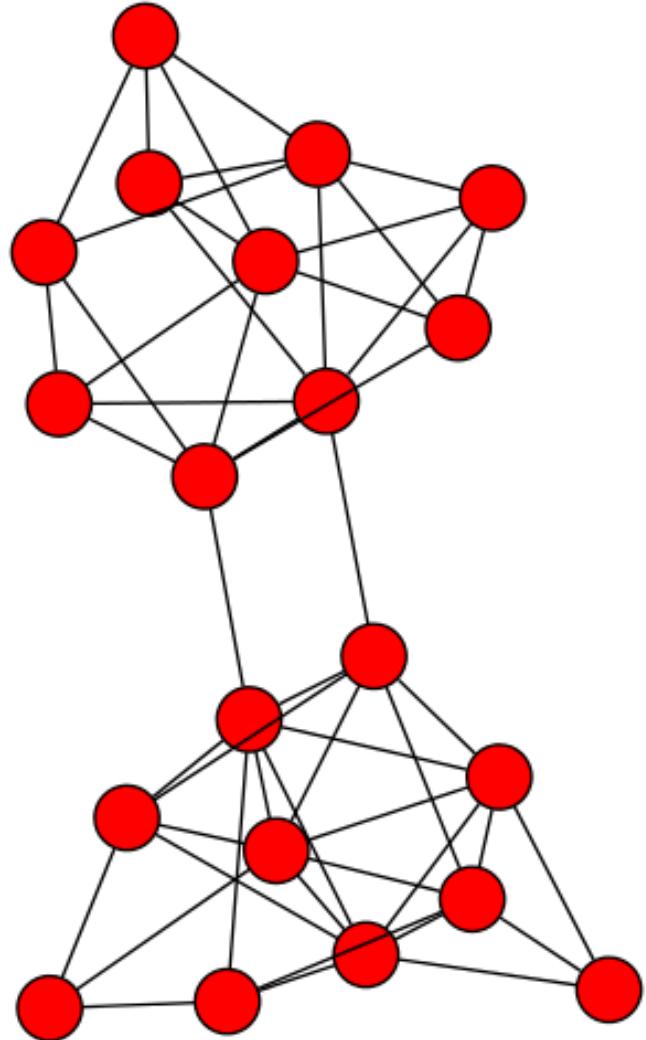
**Cluster B:** Punti Negativi

1	0.3
2	0.6
3	0.3

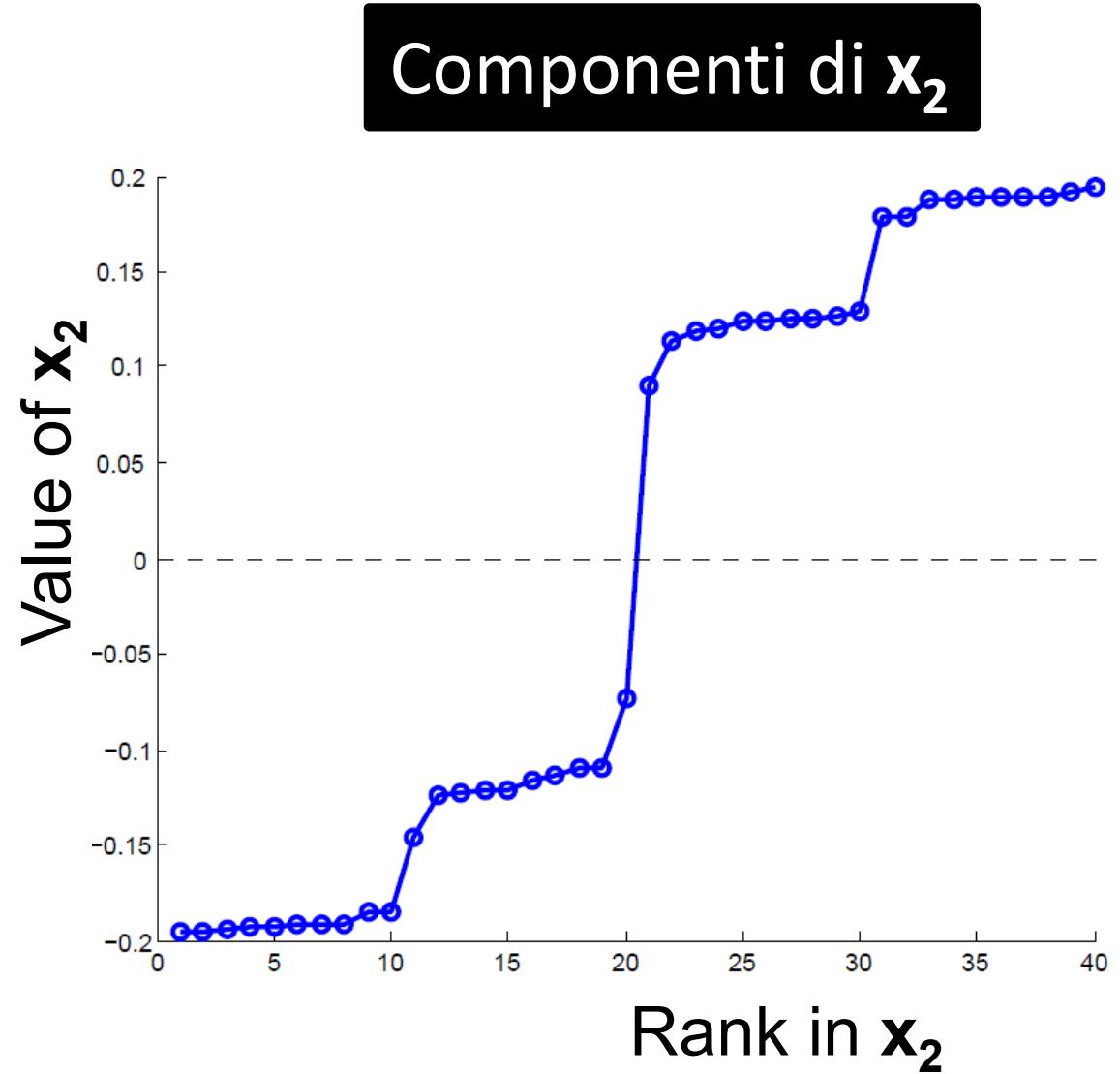
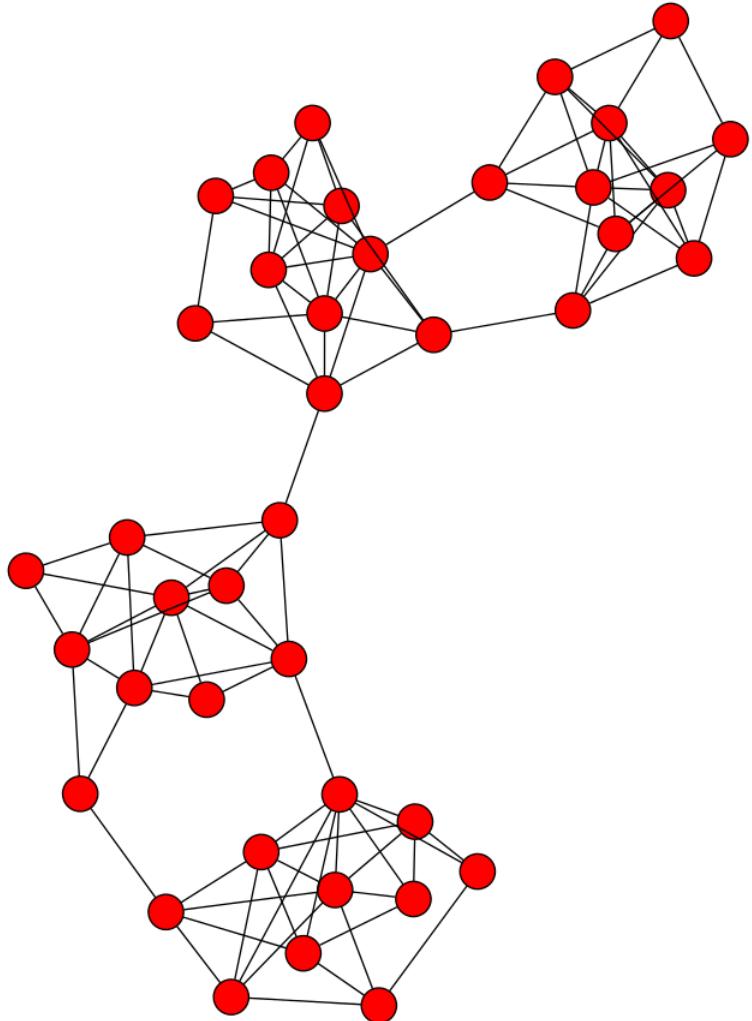
4	-0.3
5	-0.3
6	-0.6



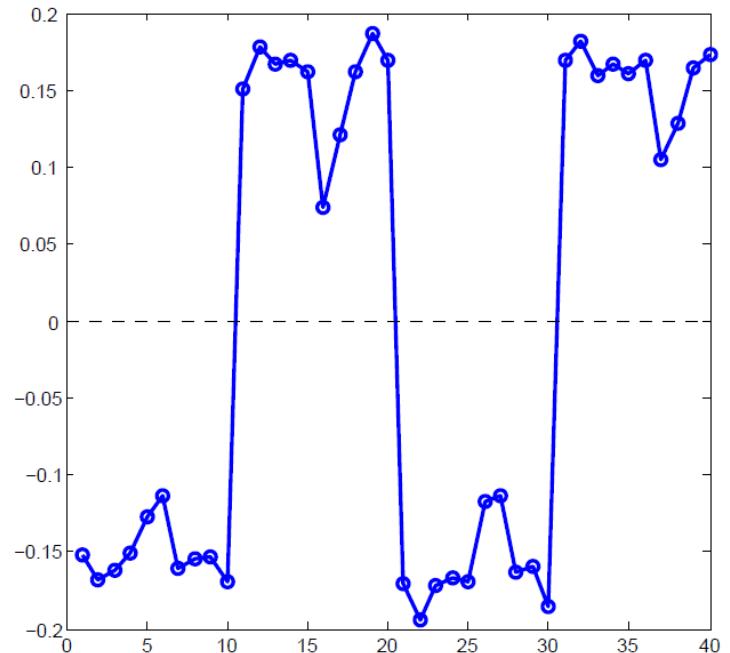
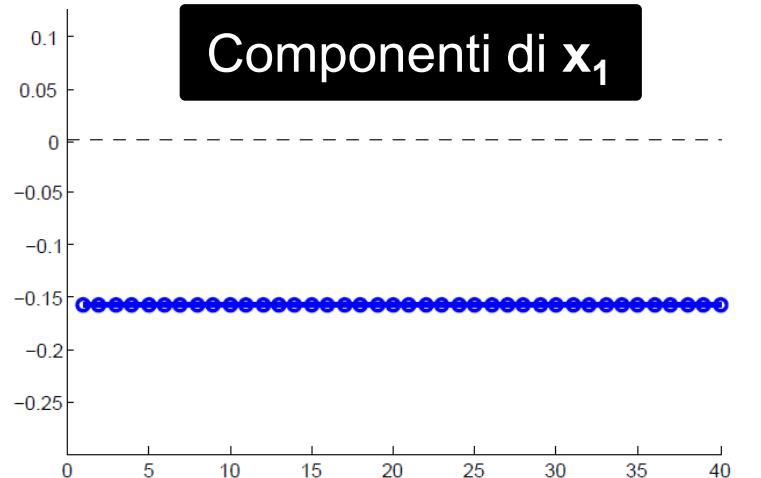
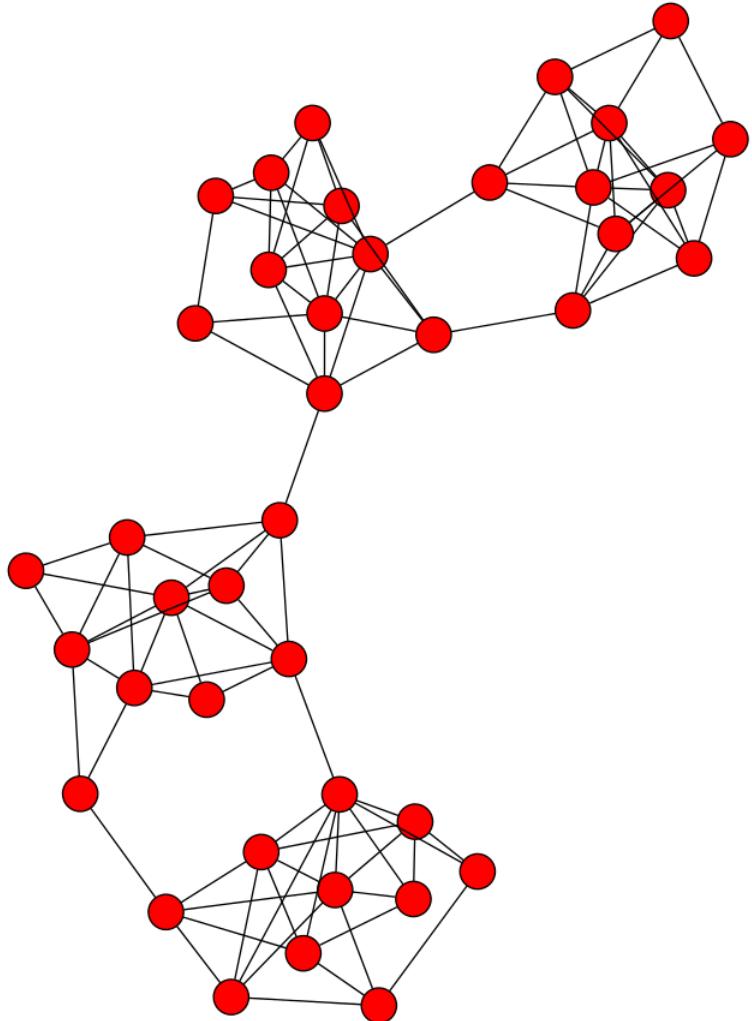
# Esempio: Spectral Partitioning



# Esempio: Spectral Partitioning



# Esempio: Spectral partitioning



Componenti di  $\mathbf{x}_3$

# k-Way Spectral Clustering

- Come partizioniamo in  $k$  cluster?
- Due approcci di base:
  - Recursive bi-partitioning [Hagen et al., '92]
    - Ricorsivamente applica la bipartizione e suddividi i cluster in modo gerarchico
    - Inefficiente e instabile
  - Cluster multiple eigenvector [Shi-Malik, '00]
    - Costruisci uno spazio ridotto con più autovettori
    - Comunemente usato in diversi lavori
    - Approccio preferibile...

# Perché usare diversi autovettori?

- **Approssimare il cut ottimale** [Shi-Malik, '00]
  - Usato per approssimare il k-way normalized cut
- **Sottolinea i cluster coesivi**
  - Aumenta la disuguaglianza nella distribuzione dei dati
  - Le associazioni tra punti simili vengono amplificate, le associazioni tra punti dissimili vengono attenuate
- **Spazio ben separato**
  - Trasforma i dati in un nuovo "spazio incorporato" Costituito da vettori di base ortogonali
  - Gli autovettori prevengono l'instabilità a causa della perdita di informazioni

# Sweep

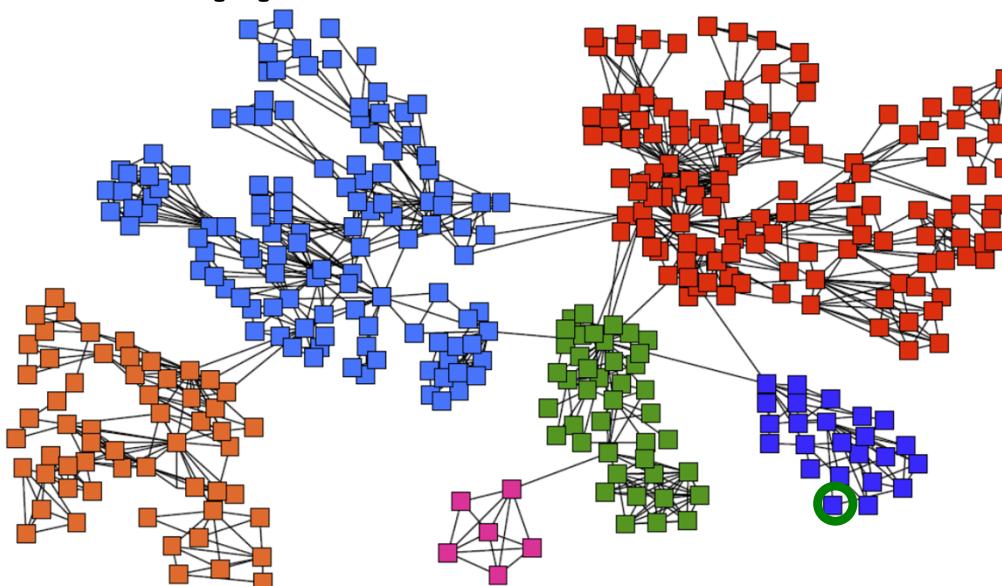
# Setting

- **Il grafo è grande**
  - **Entra in memoria primaria**
    - Es. 200M nodi e 2 Miliardi di archi
  - Il grafo è troppo grande per pensare di eseguire un algoritmo che richieda piu' di un tempo lineare
- Introdurremo un algoritmo basato su PageRank per trovare cluster densi.
  - Il running time sarà proporzionale alla dimensione dei cluster e non del grafo!

# Idea: nodi seme

## ■ Scoprire cluster a partire dai nodi seme

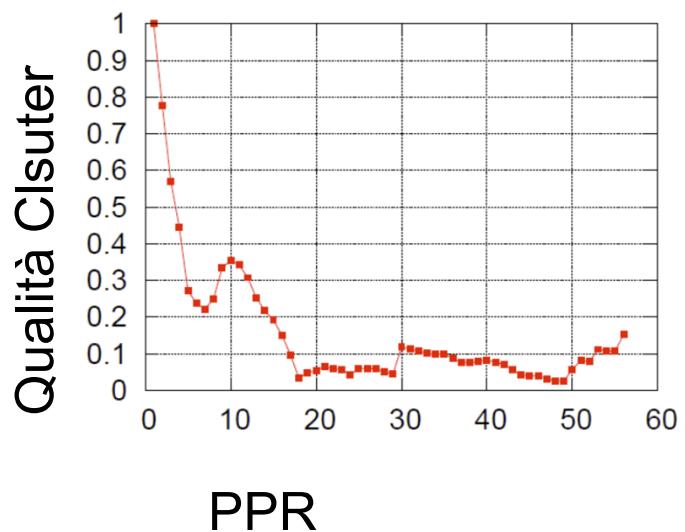
- Dato: Nodo seme  $s$
- Calcola il **Personalized PageRank (PPR)** approssimato attorno al nodo  $s$  (telerpot set =  $\{s\}$ )
- Se  $s$  appartiene ad un cluster la random walk rimarrà **intrappolata** nel cluster



# Intuizione

## ■ Outline dell'algoritmo

- Scegliere un nodo  $s$  di interesse
- Eseguire il **PPR** con il teleport set  $\{s\}$
- Ordinare i nodi in modo decrescente rispetto allo **score PPR**
- **Sweep** sui nodi per trovare cluster buoni



# Criterio per il partizionamento

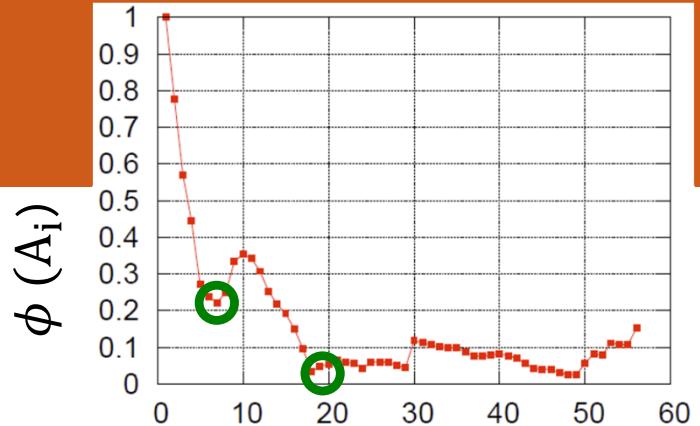
## ■ Conduttanza

- Connettività nel gruppo rispetto al resto della rete  
relativa alla densità del gruppo

$$\phi(S) = \frac{|\{(x, y) \in E \mid x \in S, y \notin S\}|}{\min(Vol(S), 2m - Vol(S))}$$

- Dove  $Vol(S) =$  numero archi in  $S +$  archi uscenti da  $S$
- $Vol(V) = 2m$ , dove  $m = |E|$
- Perché usare questo criterio?
  - Produce partizionamenti più bilanciati

# Algoritmo Sweep



PPR

- Outline
  - Scegliere un nodo  $s$  di interesse
  - Eseguire PPR con  $\text{teleport}=\{s\}$
  - Ordinare i nodi in modo decrescente rispetto allo score PPR
  - Sweep (spazzare) sui nodi per trovare cluster buoni
- Sweep
  - Ordinare i nodi in modo decrescente rispetto allo score PPR  $r_1 > r_2 > \dots > r_n$
  - Per ogni  $i$  calcolare  $\phi(A_i = \{u_1, u_2, \dots, u_i\})$
  - Il minimo locale di  $\phi(A_i)$  corrisponde ad un buon cluster!

# Calcolare lo Sweep

- La curva di Sweep può essere calcolata in **tempo lineare**
  - Iteriamo sui nodi,
  - Manteniamo una tabella hash dei nodi in  $A_i$
  - Per calcolare  $\phi(A_{i+1}) = \frac{Cut(A_{i+1})}{Vol(A_{i+1})}$  con
    - $Vol(A_{i+1}) = Vol(A_i) + d_{i+1}$
    - $Cut(A_{i+1}) = Cut(A_i) + d_{i+1} - 2\#(\text{archi da } u_{i+1} \text{ ad } A_i)$

Implementare una funzione che consente il calcolo incrementale della conduttanza

# PageRank Approssimato

- Chiamato anche PageRank-Nibble
  - Metodo veloce per calcolare il Personalized Page Rank PPR con teleport set  $\{s\}$
  - $\text{ApproximatePageRank}(s, \beta, \epsilon)$ 
    - $s$  nodo seme
    - $\beta$  parametro teleport
    - $\epsilon$  parametro per l'errore di approssimazione

# PPR approssimato

- PPR approssimato su grafi non direzionali
  - **Lazy random walk**, variante della random walk, stai fermo con probabilità  $\frac{1}{2}$  ad ogni step, e cammini nel vicinato nell'altra metà dei casi

$$r_u^{(t+1)} = \frac{1}{2} r_u^{(t)} + \frac{1}{2} \sum_{i \rightarrow u} \frac{1}{d_i} r_i^{(t)}$$

- Teniamo traccia degli **score PPR residui**  $q_u = p_u - r_u^{(t)}$
- Residuo  $q_u$  quanto lo score  $p_u$  è ben approssimato
- $r_u^{(t)}$  la stima del PPR all'istante  $t$
- Se il residuo  $q_u$  del nodo  $u$  è troppo grande  $\frac{q_u}{d_u} \geq \epsilon$  allora cammina ancora (distribuisci alcuni dei residui  $q_u$  al vicinato di  $u$  altrimenti non toccare il nodo)

# Arrivare al PPR approssimato

- Un modo diverso di vedere il PageRank

$$p_\beta(a) = (1 - \beta)a + \beta p_\beta(M * a)$$

- $p_\beta(a)$  PageRank con parametro di teletrasporto  $\beta$  e vettore di teletrasporto  $a$
- $p_\beta(M * a)$  PageRank con parametro di teletrasporto  $\beta$  e vettore di teletrasporto  $M * a$ 
  - $M$  matrice stocastica del page rank
  - $M^*a$  uno step del random walk

- $p_\beta(a) = (1 - \beta)a + \beta p_\beta(M * a)$ 
  - Spezziamo la probabilità in due casi:
    - walk di lunghezza 0
    - walk di lunghezza  $> 0$
  - Con probabilità  $(1 - \beta)$  finiamo dove abbiamo iniziato
  - Con probabilità  $\beta$  la walk inizia alla distribuzione, fa un passo  $(M * a)$  e prende il resto della random walk con distribuzione  $p_\beta(M * a)$

# Operazione «Push»

- Idea
  - $r$  PageRank approssimato,  $q$  residual PageRank
  - Iniziamo con una approssimazione banale:  $r=0$  e  $q=a$
  - Iterativamente facciamo un push del PageRank da  $q$  ad  $r$  fino a quando  $q$  è piccolo
- **Push 1 step di lazy random walk dal nodo  $u$**

$Push(u, r, q)$ :

$$r' = r, q' = q$$

$$r'_u = r_u + (1 - \beta)q_u$$

$$q'_u = \frac{1}{2}\beta q_u$$

Per ogni  $v$  tale che  $u \rightarrow v$ :

$$q'_v = q_v + \frac{1}{2}\beta \frac{q_u}{d_u}$$

return  $r'$  e  $q'$

# Intuzione

- Se  $q_u$  è grande significa che stiamo sottostimando l'importanza di u
- Dobbiamo prendere alcuni dei residui ( $q_u$ ) e darli
- Teniamo  $\frac{1}{2}\beta q_u$  e il resto lo diamo ai vicini.  
Questo corrisponde a dividere  $\frac{1}{2}\beta q_u/d_u$

# PPR approssimato

*ApproxPageRank*( $S, \beta, \epsilon$ )

Set  $r = [0, \dots, 0]$  e  $q = [0, \dots, 0, 1, 0, \dots, 0]$

while  $\max_{u \in V} \frac{q_u}{d_u} \geq \epsilon$

scegli un qualsiasi vertice  $u$  tale che  $\frac{q_u}{d_u} \geq \epsilon$

*Push*( $u, r, q$ ):

$$r' = r, q' = q$$

$$r'_u = r_u + (1 - \beta)q_u$$

$$q'_u = \frac{1}{2}\beta q_u$$

Per ogni  $v$  tale che  $u \rightarrow v$ :

$$q'_v = q_v + \frac{1}{2}\beta \frac{q_u}{d_u}$$

$$r = r' \text{ (normalizzato)}, q = q'$$

Return  $r', q'$

# Esempio $\beta = 0,5$

**Init:**

**r** = [0, 0, 0, 0]

**q** = [1, 0, 0, 0]

**Push(s, r, q):**

**r** = [.5, 0, 0, 0]

**q** = [.25, .08, .08, .08]

**Push(s, r, q):**

**r** = [.62, 0, 0, 0]

**q** = [.06, .10, .10, .10]

**Push(a, r, q):**

**r** = [0.62, .05, 0, 0]

**q** = [.09, .03, .10, .10]

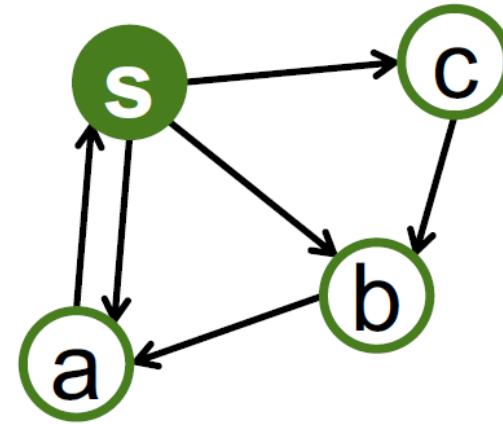
**Push(b, r, q):**

**r** = [0.62, .05, .05, 0]

**q** = [.09, .05, .03, .10]

....

**r** = [0.57, 0.19, 0.14, 0.09]

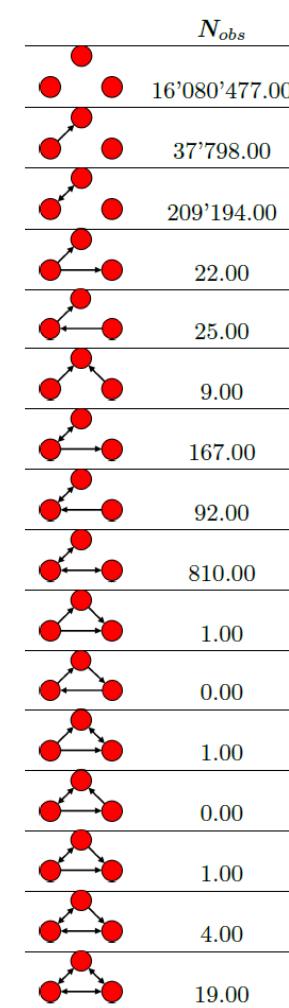


- PageRank approssimato calcola il PPR in un tempo pari a  $\left(\frac{1}{\epsilon(1-\beta)}\right)$ 
  - Power iteration richiederebbe tempo  $\left(\frac{\log n}{\epsilon(1-\beta)}\right)$
- Approssimazione garantita
  - Se esiste un taglio con una conduttanza  $\phi$  e volume  $k$  allora il metodo trova un taglio di conduttanza  $O\left(\sqrt{\frac{\phi}{\log k}}\right)$

# Motif-Based Clustering

# Clustering spettrale basato su Motivi

- Potremmo voler costruire i cluster rispetto a pattern diversi dagli archi
- Sottografi (motif o graphlet) building block delle reti

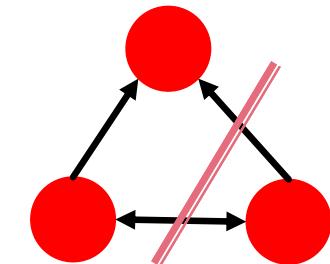


# Ridefiniamo la conduttanza per i motivi!

Edge cut



motif cut



$$V(S) = \#\text{endpoint in } S$$

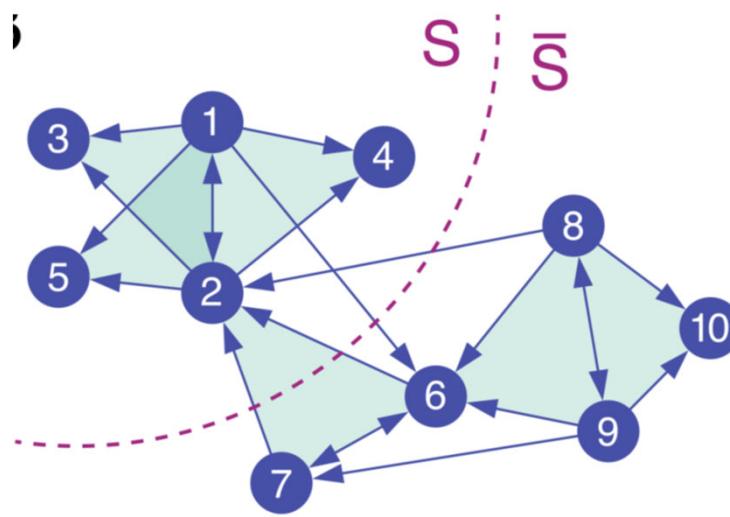


$$Vol_M(S) = \#\text{ motif endpoint in } S$$

$$\phi(S) = \frac{\#\text{(edge cut)}}{Vol(S)}$$



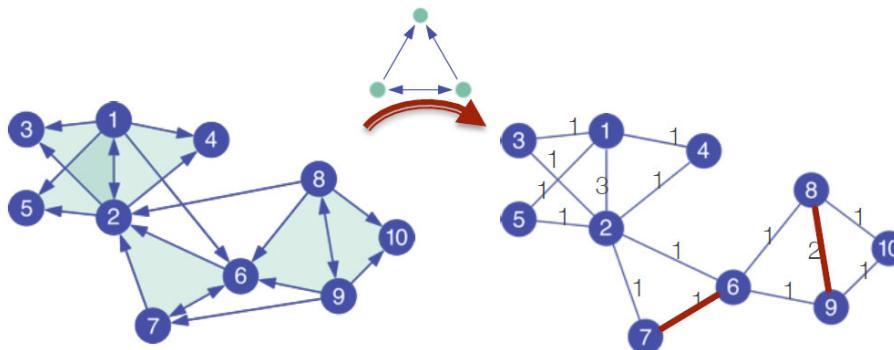
$$\phi_M(S) = \frac{\#\text{(motif cut)}}{Vol_M(S)}$$



## ■ Passi

### ■ Pre-processing

- $W_{ij}^{(M)} = \text{volte in cui } (i, j) \text{ partecipa ad un motif}$



### ■ Approximate PageRank

- Come quello visto ma sulla matrice pesata  $W^{(M)}$

### ■ Sweep

- Come visto

# Ottimizzazione Modularità Louvain

# Algoritmo Louvain

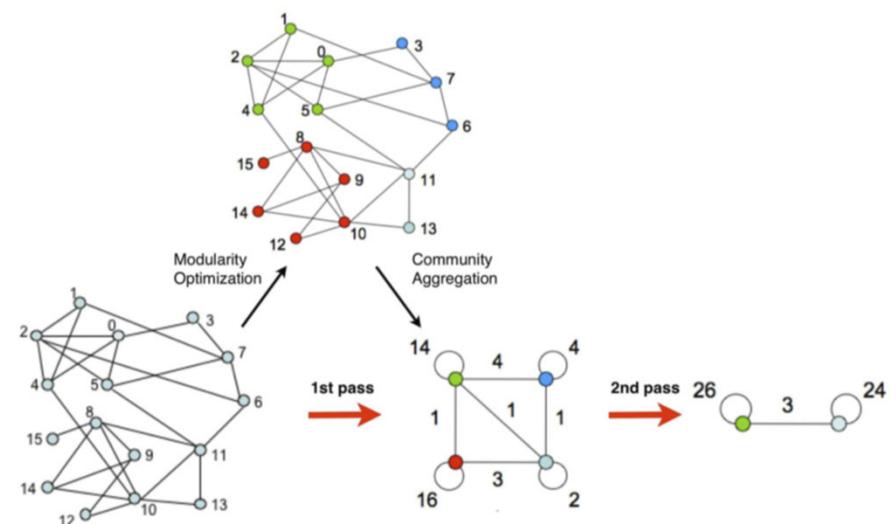
- Algoritmo Greedy per la community detection
  - $O(n \log n)$  running time
  - Funziona con grafi pesati
  - Comunità gerarchiche
  - Ampiamente usato per large network:
    - Veloce
    - Converge velocemente
    - Alta modularità

# Modularità

$$Q = \frac{1}{2m} \sum_{i,j \in N} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

Dove  $c_i$  e  $c_j$  sono i cluster di appartenenza dei due nodi i e j rispettivamente  
 $\delta(c_i, c_j)$  funzione indicatore che vale 1 se i cluster coincidono 0 altrimenti

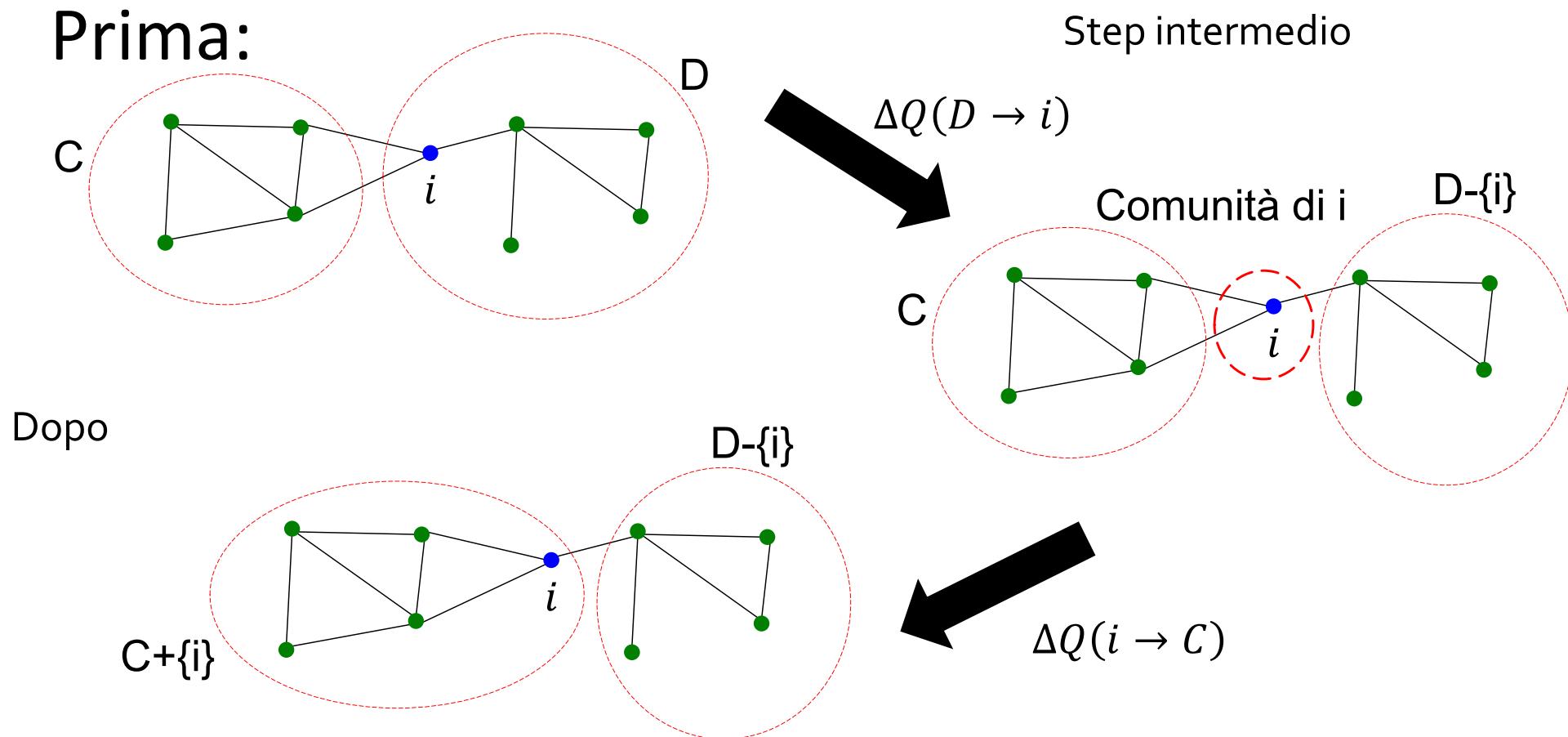
- Massimizzare in modo greedy la modularità
- Ogni passo è fatto da due fasi:
  - **Fase 1:** la modularità è ottimizzata consentendo solo cambiamenti locali alle comunità dei nodi
  - Fase 2: Le comunità identificate sono aggregate in supernodi per costruire una nuova rete
  - Ritorna alla fase 1



# Louvain: Fase 1 (partizionamento)

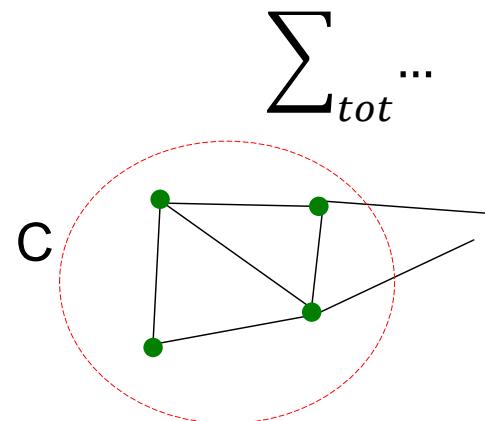
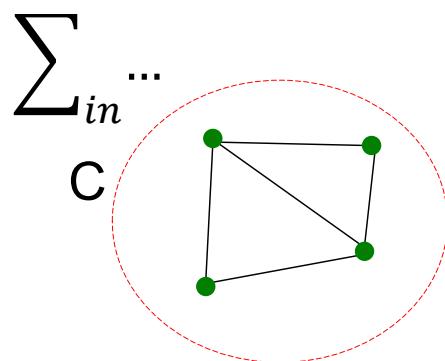
- Inserisci ogni nodo del grafo in una **community distinta**
- Per ogni nodo  $i$ , l'algoritmo effettua due calcoli:
  - Calcola la modularità delta ( $\Delta Q$ ) quando inserisci il nodo  $i$  nella comunità di un vicino  $j$
  - Sposta il nodo  $i$  nella comunità del nodo  $j$  che da il maggiore guadagno in  $\Delta Q$
- **La fase 1 finisce quando non ci sono movimenti che possano aumentare il gain**
  - L'output dell'algoritmo dipende dall'ordine in cui i nodi sono processati
  - La fase finisce con un massimo locale della modularità

$$\Delta Q(D \rightarrow i \rightarrow C) = \Delta Q(i \rightarrow C) + \Delta Q(D \rightarrow i)$$



# Deriviamo $\Delta Q(i \rightarrow C)$

- $\sum_{in} \dots = \sum_{i,j \in C} A_{i,j}$  è la somma dei pesi dei link tra i nodi in  $C$
- $\sum_{tot} \dots = \sum_{i \in C} k_i$  è la somma dei pesi di tutti i link dei nodi in  $C$

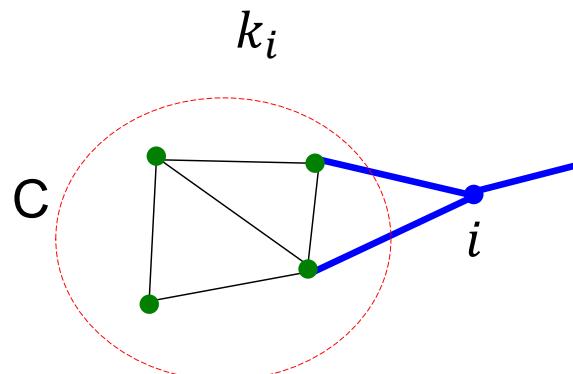
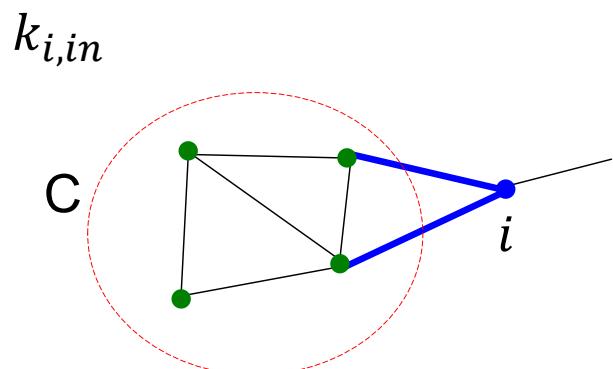


# Deriviamo $\Delta Q(i \rightarrow C)$

- $\sum_{in} \dots = \sum_{i,j \in C} A_{i,j}$  è la somma dei pesi dei link tra i nodi in  $C$
- $\sum_{tot} \dots = \sum_{i \in C} k_i$  è la somma dei pesi di tutti i link dei nodi in  $C$
- $$Q(C) = \frac{1}{2m} \sum_{i,j \in C} \left[ A_{i,j} - \frac{k_i k_j}{2m} \right] = \frac{\sum_{i,j \in C} A_{i,j}}{2m} - \frac{(\sum_{i \in C} k_i)(\sum_{j \in C} k_j)}{(2m)^2} = \frac{\sum_{in}}{2m} - \left( \frac{\sum_{tot}}{2m} \right)^2$$

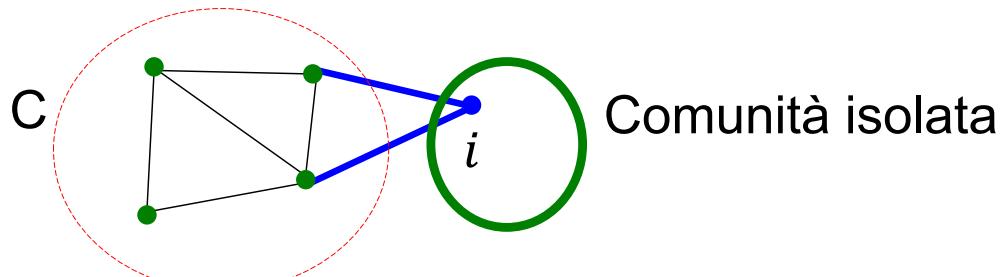
# Deriviamo $\Delta Q(i \rightarrow C)$

- $k_{i,in} = \sum_{j \in C} A_{i,j} + \sum_{j \in C} A_{j,i}$  è la somma dei pesi dei link tra il nodo i e C
- $k_i$  = è la somma dei pesi del nodo i (grado di i)

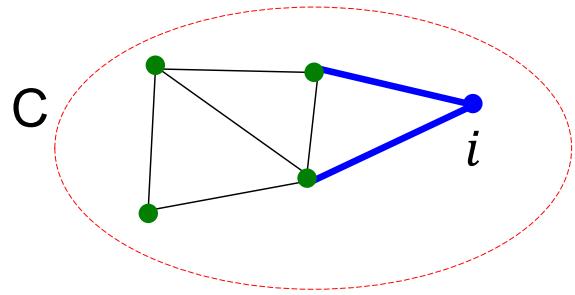


# Deriviamo $\Delta Q(i \rightarrow C)$

## ■ Prima del merging



## ■ Dopo il merging



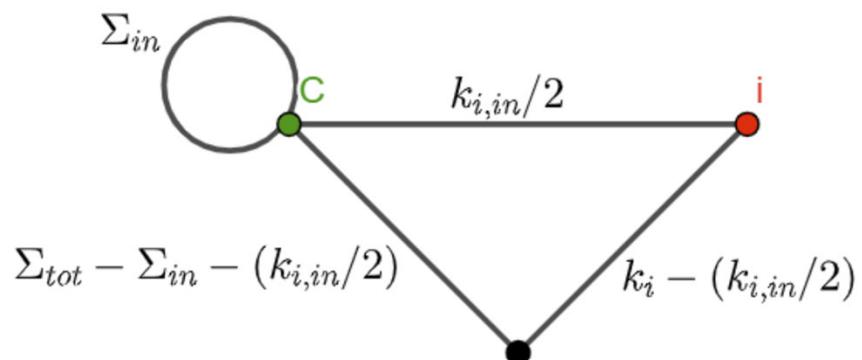
$$\begin{aligned}Q_{prima} &= Q(C) + Q(\{i\}) \\&= \left[ \frac{\sum_{in}}{2m} - \left( \frac{\sum_{tot}}{2m} \right)^2 \right] \\&\quad + \left[ 0 - \left( \frac{k_i}{2m} \right)^2 \right]\end{aligned}$$

$$\begin{aligned}Q_{dopo} &= Q(C + \{i\}) \\&= \left[ \frac{\sum_{in} + k_{i,in}}{2m} - \left( \frac{\sum_{tot} + k_i}{2m} \right)^2 \right]\end{aligned}$$

Contributo della modularità  
dopo l'inserimento del nodo i

Contributo della modularità  
prima dell'inserimento del  
nodo i

$$\Delta Q(i \rightarrow C) = \left[ \frac{\Sigma_{in} + k_{i,in}}{2m} - \left( \frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[ \frac{\Sigma_{in}}{2m} - \left( \frac{\Sigma_{tot}}{2m} \right)^2 - \left( \frac{k_i}{2m} \right)^2 \right]$$



# Louvain: guadagno modularità

- A cosa corrisponde  $\Delta Q$  quando muoviamo il nodo i nella comunità C?

$$\Delta Q(i \rightarrow C) = \left[ \frac{\sum_{in} + k_{i,in}}{2m} - \left( \frac{\sum_{tot} + k_i}{2m} \right)^2 \right] - \left[ \frac{\sum_{in}}{2m} - \left( \frac{\sum_{tot}}{2m} \right)^2 - \left( \frac{k_i}{2m} \right)^2 \right]$$

Dove:

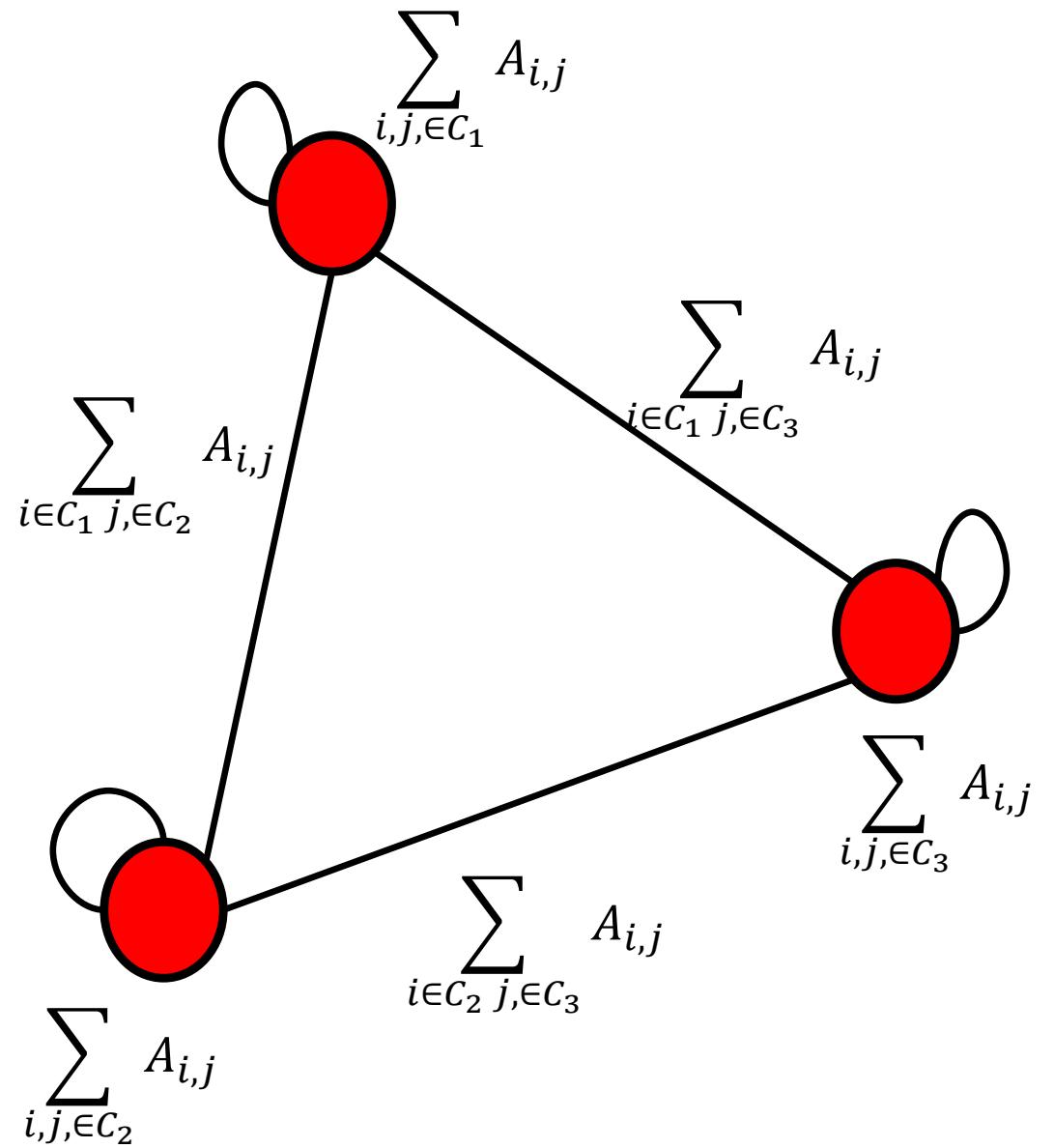
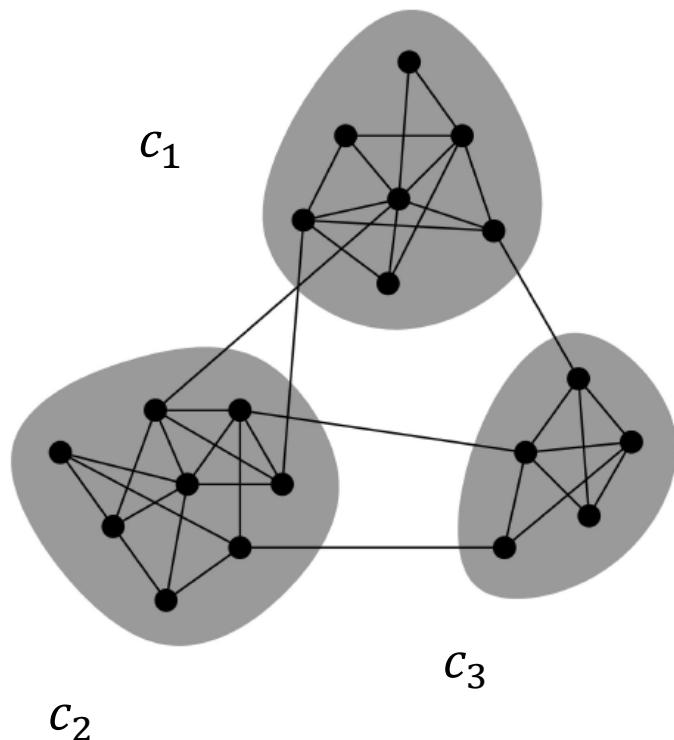
- $\sum_{in}$  ... è la somma dei pesi dei link tra i nodi in C
- $\sum_{tot}$  ... è la somma dei pesi di tutti i link dei nodi in C
- $k_{i,in}$  ... è la somma dei pesi dei link tra il nodo i e C
- $k_i$  ... è la somma di tutti i pesi (es. il grado) del nodo i
- Allo stesso modo bisogna derivare  $\Delta Q(D \rightarrow i)$  uscita del nodo i dalla comunità D
- $\Delta Q = \Delta Q(i \rightarrow C) + \Delta Q(D \rightarrow i)$



# Louvain: seconda fase

- Tutte le community ottenute nella prima fase sono contratte in super-nodi, e una rete è creata di conseguenza:
  - I super-nodi sono connessi se esiste almeno un arco che connette le due community
  - Il peso dell'arco tra due super-nodi è pari alla somma dei pesi degli archi dei nodi che connettono le community
- Ripeti la fase uno sulla rete di super-nodi

# Louvain – seconda fase



---

**Algorithm 1:** Sequential Louvain Algorithm

---

**Input:**  $G = (V, E)$ : graph representation.  
**Output:**  $C$ : community sets at each level;  
 $Q$ : modularity at each level.  
**Var:**  $\hat{c}$ : vertex  $u$ 's best candidate community set.

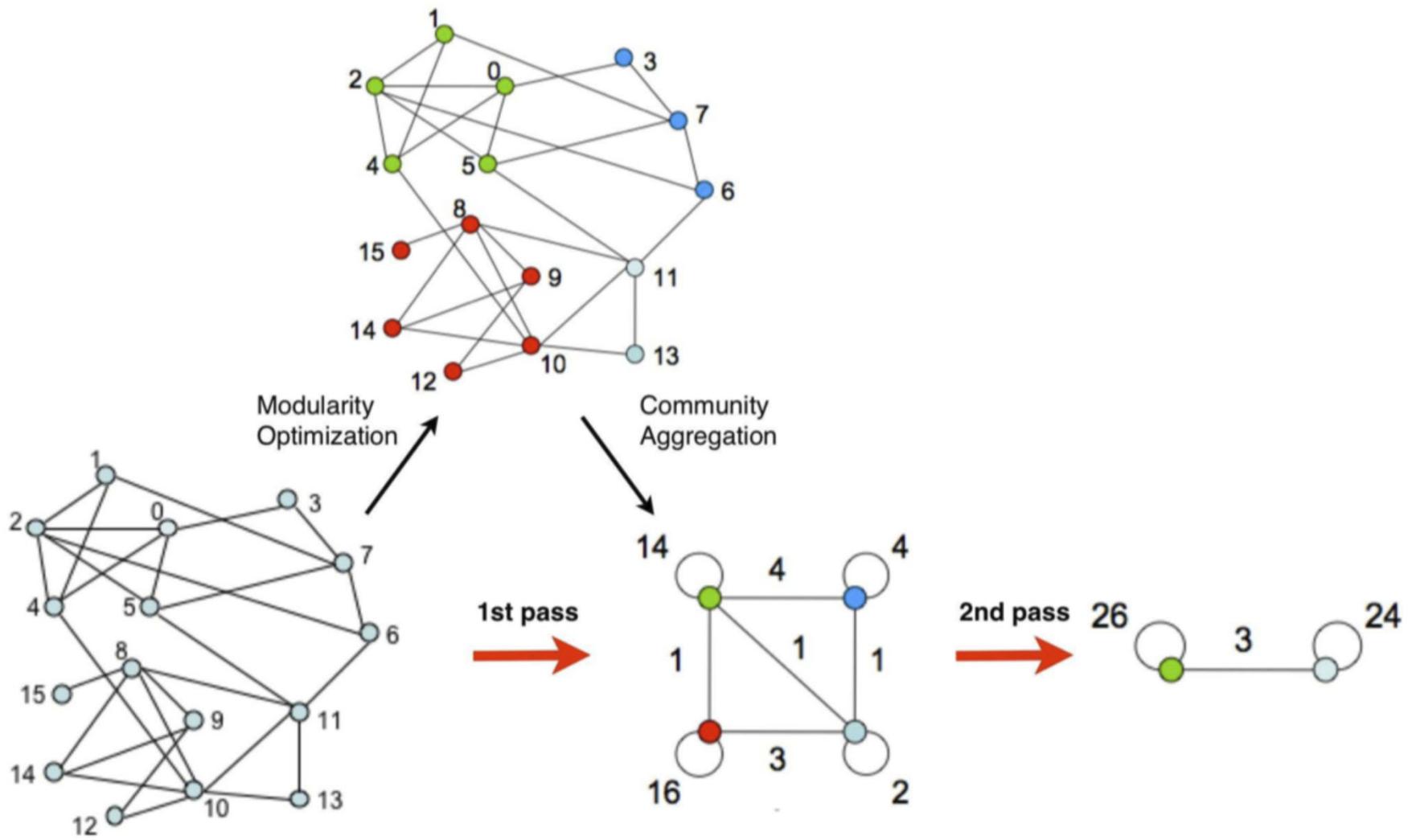
1 **Loop outer**

2     $C \leftarrow \{\{u\}\}, \forall u \in V$  ;  
    $\Sigma_{in}^c \leftarrow \sum w_{u,v}, e(u,v) \in E, u \in c \text{ and } v \in c$  ;  
    $\Sigma_{tot}^c \leftarrow \sum w_{u,v}, e(u,v) \in E, u \in c \text{ or } v \in c$  ;  
   // Phase I.

6    **Loop inner**

7      **for**  $u \in V$  and  $u \in c$  **do**  
       // Find the best community for vertex  $u$ .  
        $\hat{c} \leftarrow \operatorname{argmax}_{\forall c', \exists e(u,v) \in E, v \in c'} \Delta Q_{u \rightarrow c'}$  ;  
       **if**  $\Delta Q_{u \rightarrow \hat{c}} > 0$  **then**  
         // Update  $\Sigma_{tot}$  and  $\Sigma_{in}$ .  
          $\Sigma_{tot}^{\hat{c}} \leftarrow \Sigma_{tot}^{\hat{c}} + w(u)$  ;  $\Sigma_{in}^{\hat{c}} \leftarrow \Sigma_{in}^{\hat{c}} + w_{u \rightarrow \hat{c}}$  ;  
          $\Sigma_{tot}^c \leftarrow \Sigma_{tot}^c - w(u)$  ;  $\Sigma_{in}^c \leftarrow \Sigma_{in}^c - w_{u \rightarrow c}$  ;  
         // Update the community information.  
          $\hat{c} \leftarrow \hat{c} \cup \{u\}$  ;  $c \leftarrow c - \{u\}$  ;  
       **if** No vertex moves to a new community **then**  
         **exit inner Loop**;

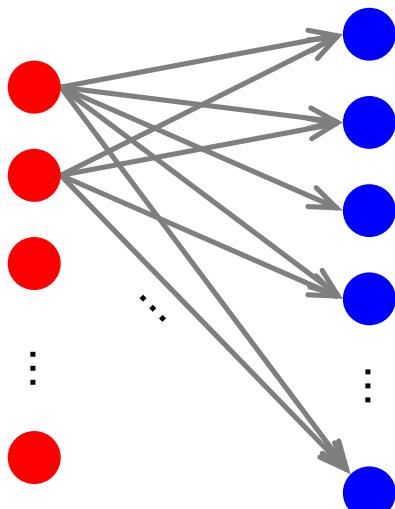
18 // Calculate community set and modularity.  
 19  $Q \leftarrow 0$  ;  
 20 **for**  $c \in C$  **do**  
    $Q \leftarrow Q + \frac{\Sigma_{in}^c}{2m} - \left(\frac{\Sigma_{tot}^c}{2m}\right)^2$  ;  
 22  $C' \leftarrow \{c\}, \forall c \in C$  ; print  $C'$  and  $Q$  ;  
 23 // Phase 2: Rebuild Graph.  
 24  $V' \leftarrow C'$  ;  
 25  $E' \leftarrow \{e(c, c')\}, \exists e(u, v) \in E, u \in c, v \in c'$  ;  
 26  $w_{c, c'} \leftarrow \sum w_{u, v}, \forall e(u, v) \in E, u \in c, v \in c'$  ;  
 27 **if** No community changes **then**  
   **exit outer Loop**;  
 29  $V \leftarrow V'$  ;  $E \leftarrow E'$  ;



# Analisi di grandi grafi: Trawling

# Trawling

- Identificare piccole comunità nel grafo del web
- Qual è la segnatura di una community / discussione nel Web graph?



Dense 2-layer graph

**Usato per definire i “topics”:**  
Quali sono le stesse persone  
a sinistra che a destra?

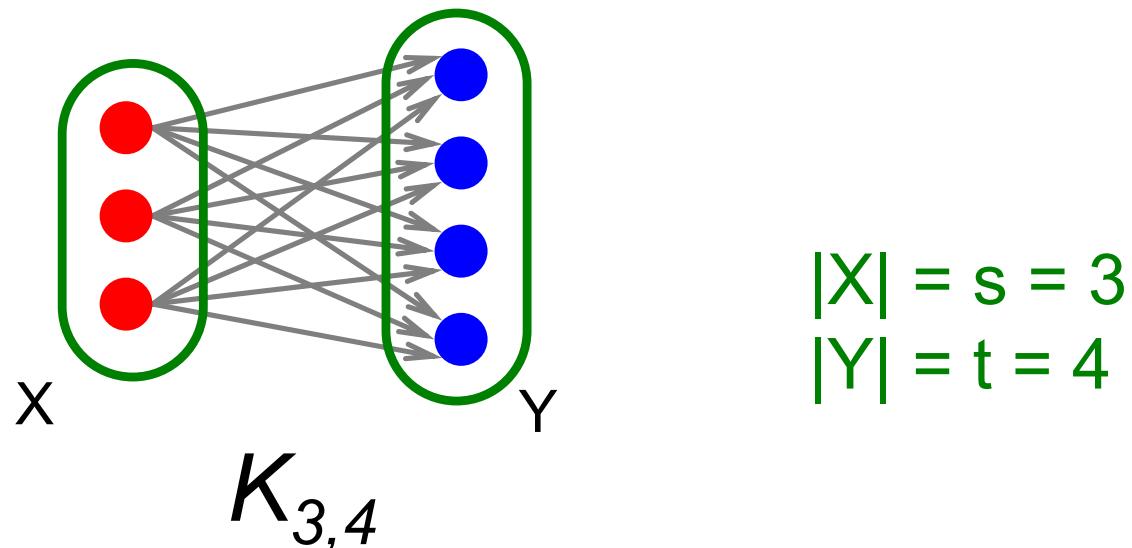
**Intuizione:** diverse persone parlano della stessa cosa

# Ricerca di piccole comunità

## ■ Problema ben definito:

Enumerare i sottografi completi bipartiti  $K_{s,t}$

- Dove  $K_{s,t}$  :  $s$  nodi a “sinistra” dove ognuno linka allo stesso nodo  $t$  sulla “destra”



Totalmente connesso

# Frequent Itemset Enumeration

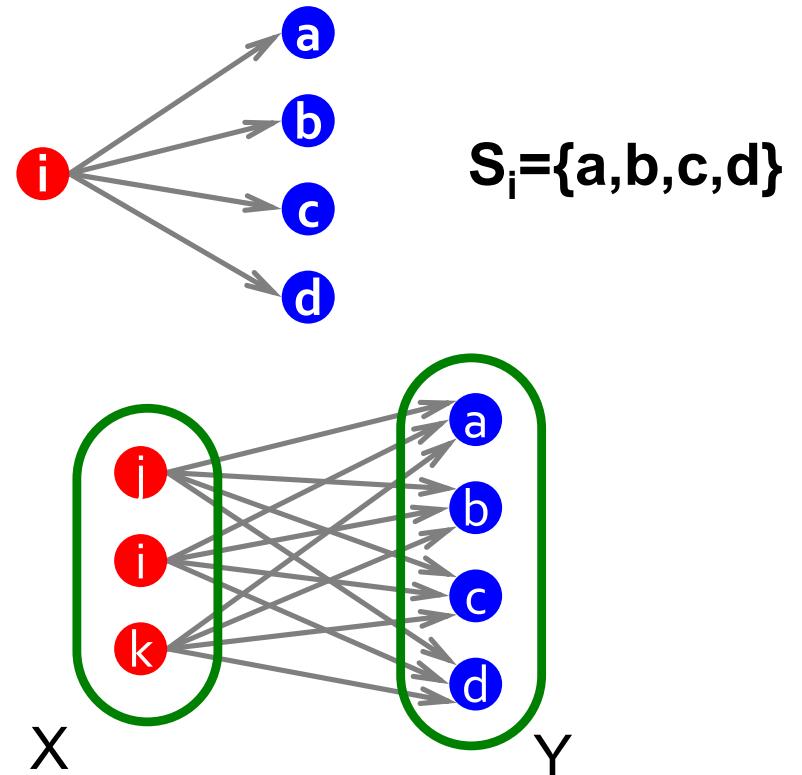
- **Market basket analysis.** Setting:
  - **Market:** Universo  $U$  di  $n$  item
  - **Basket:**  $m$  sottoinsiemi di  $U$ :  $S_1, S_2, \dots, S_m \subseteq U$   
( $S_i$  insieme di item acquistati da una persona)
  - **Supporto:** Threshold di frequenza  $f$
- **Goal:**
  - Trovare tutti i sottoinsiemi  $T$  tale che  $T \subseteq S_i$  in almeno  $f$  insiemi  $S_i$ ,  
(item in  $T$  acquistati almeno  $f$  volte)
- **Dove sta la connessione tra itemset mining e grafi bipartiti?**

# From Itemsets to Bipartite $K_{s,t}$

**Frequent itemset = complete bipartite graphs!**

## ■ Come?

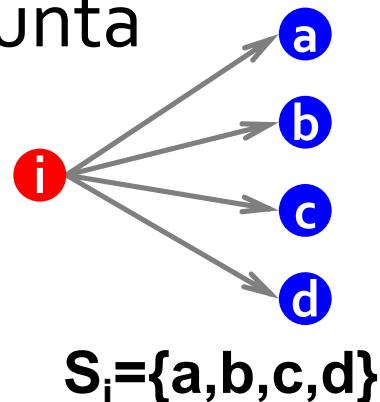
- Ogni nodo  $i$  come un insieme  $S_i$  di nodi a cui  $i$  punta
- $K_{s,t} = \text{Insieme } Y \text{ di size } t \text{ che si presenta in } s \text{ set } S_i$
- Cercare  $K_{s,t} \rightarrow$  insieme con frequenza minima  $s$  guardare al layer  $t$  – tutti gli insiemi frequenti di size  $t$



$s \dots$  minimum support ( $|X|=s$ )  
 $t \dots$  itemset size ( $|Y|=t$ )

# From Itemsets to Bipartite $K_{s,t}$

Ogni nodo  $i$  come un insieme  $S_i$  di nodi a cui  $i$  punta



Find frequent itemsets:  
 $s$  ... minimum support  
 $t$  ... itemset size

**Abbiamo trovato  $K_{s,t}$ !**

$K_{s,t} = \text{insieme } Y \text{ di size } t$   
 presente in  $s$  set  $S_i$

Troviamo un **insieme frequente**  $Y = \{a, b, c\}$  con supporto  $s$  Quindi, ci sono  $s$  nodi che linkano a tutti  $\{a, b, c\}$ :

