

# Mining Graphs

## 1. Clustering

### Note

### DEF (Clustering)

- **INFORMALE** Il clustering è il processo di esaminare una collezione di "punti" e raggrupparli in "cluster" secondo una determinata misura di distanza. L'obiettivo è che i punti all'interno dello stesso cluster abbiano una distanza ridotta tra loro, mentre i punti appartenenti a cluster diversi presentino una distanza maggiore tra loro.
- **FORMALE** Il clustering consiste nel prendere un insieme  $\mathbb{U}$  di  $n$  oggetti  $p_1, p_2, \dots, p_n$  (che possono rappresentare foto, punti, documenti etc...) e classificarli in gruppi (cluster) coerenti. L'idea è dunque quella di dividere tali punti in gruppi  $C_1, C_2, \dots, C_k$  detti **cluster**, dove ciascun cluster rappresenta un sottoinsieme di oggetti simili.
  - $\mathbb{C} = \{C_1, C_2, \dots, C_k\}$
  - $C_i \subseteq \mathbb{U} \forall i = 1, \dots, k$
  - $C_i \cap C_j = \emptyset \forall i, j = 1, \dots, k$  tale che  $i \neq j$
  - $\bigcup_{i=1}^k C_i = \mathbb{U}$

### DEF ( $k$ - Clustering)

Si dice  $k$ -clustering una partizione dell'insieme  $\mathbb{U}$  in  $k$  sottoinsiemi non vuoti.

### Note

### DEF (Distance Function)

Per valutare la vicinanza tra due oggetti  $p_i$  e  $p_j$  viene definita una funzione di distanza ( $d(p_i, p_j)$ ) che associa ad ogni coppia di oggetti  $p_i$  e  $p_j$  un valore numerico.

## Proprietà

1. **Identità degli indiscernibili:**  $d(p_i, p_j) = 0$  se e solo se  $p_i = p_j$
2. **Non negatività:**  $d(p_i, p_j) \geq 0$
3. **Simmetria:**  $d(p_i, p_j) = d(p_j, p_i)$

### Note

## DEF (Spacing)

Lo spacing rappresenta la **distanza minima tra due punti appartenenti a cluster differenti**.

$$d(C_i, C_j) = \text{spacing}(C_i, C_j) = \min \{d(p_i, p_j) : p_i \in C_i, p_j \in C_j\}$$

Questo valore misura quanto sono *separati* due cluster.

## OBIETTIVO

Dato un intero  $k$ , l'obiettivo è trovare un  $k$ -clustering che massimizza lo spacing tra i cluster. In pratica, si cerca di garantire che i cluster siano il più possibile distanti tra loro.

Il problema del clustering si può modellare mediante l'utilizzo di grafi pesati  $G(V, E)$ .

Dati in input  $P = \langle p_1, p_2, \dots, p_n \rangle$ , la funzione di distanza  $d()$  e  $k$ , dobbiamo definire il nostro modello di grafo su cui poi andare a creare i cluster.

- $V = \mathbb{U} = \{p_1, \dots, p_n\}$  è l'insieme dei nodi (oggetti da clusterizzare).
- $E$  è l'insieme di tutte le coppie non ordinate di nodi, formando un grafo completo.
- Ogni arco  $e = (p_i, p_j)$  ha un costo  $\text{cost}(e) = d(p_i, p_j)$ , dove  $d(p_i, p_j)$  è la funzione di distanza tra i punti.

L'algoritmo che risolve questo problema è il **Single-Link k-Clustering Algorithm**, un algoritmo greedy che costruisce la soluzione passo dopo passo seguendo una strategia semplice ma efficace.

L'obiettivo è ottenere  $k$  gruppi (o cluster) di punti mantenendo una separazione massima tra i gruppi.

L'idea di base è quella di immaginare i punti come nodi di un grafo. Inizialmente, ogni punto è considerato un cluster separato. Quindi, abbiamo un grafo con  $n$  nodi e nessun arco. A questo punto, l'algoritmo procede in più passaggi:

1. A ogni iterazione, si cerca la coppia di punti appartenenti a cluster diversi che sono i più vicini tra loro. Questo significa che viene scelto l'arco con il costo più basso, dove il costo è la distanza tra i due punti. Questo arco viene aggiunto al grafo e i due cluster vengono fusi in uno solo.
2. Si continua a unire cluster seguendo lo stesso principio, ovvero scegliendo sempre la coppia di cluster più vicina. Questo processo si ripete  $n - k$  volte, dove  $k$  è il numero desiderato di cluster. Quando rimangono esattamente  $k$  cluster, l'algoritmo si ferma.
3. A questo punto, i cluster rimanenti sono il risultato finale. Essi rappresentano gruppi ben distinti, con la distanza minima (spacing) tra punti di cluster diversi che è stata massimizzata durante il processo.

Questo metodo è praticamente identico all'algoritmo di **Kruskal**, usato per trovare il **Minimum Spanning Tree (MST)** di un grafo. Tuttavia, c'è una differenza chiave: mentre Kruskal continua fino a costruire l'intero MST, il single-link clustering si ferma non appena rimangono  $k$  componenti connesse. Questo significa che, in pratica, stiamo costruendo un MST e poi rimuoviamo gli  $k - 1$  archi più costosi per dividere il grafo in  $k$  cluster.

TEOREMA: Sia  $C^*$  il clustering  $C_1^*, \dots, C_k^*$  creato eliminando i  $k-1$  archi più pesanti di un MST. Allora,  $C^*$  è un  $K$ -clustering di spacing massimo.

DIM: La dimostrazione fa vedere che, comunque preso un altro clustering, ovvero un'altra  $K$  partizione diversa da quella prodotta dai  $k-1$  archi più pesanti da un qualunque MST, allora lo spacing di quest'altra soluzione non può essere migliore della soluzione  $C^*$  prodotta dal algoritmo greedy.

$C^* = C_1^*, \dots, C_k^*$ : soluzione ottima proposta dall'algoritmo greedy

$C = C_1, \dots, C_k$ : una qualunque altra soluzione

Obiettivo: Dimostrare che  $C_1, \dots, C_k$  ha come spacing (minima distanza tra una coppia di elementi) il valore  $d^*$

↳ peso del arco  $k-1$  che abbiamo rimosso

↳ arco di peso massimo

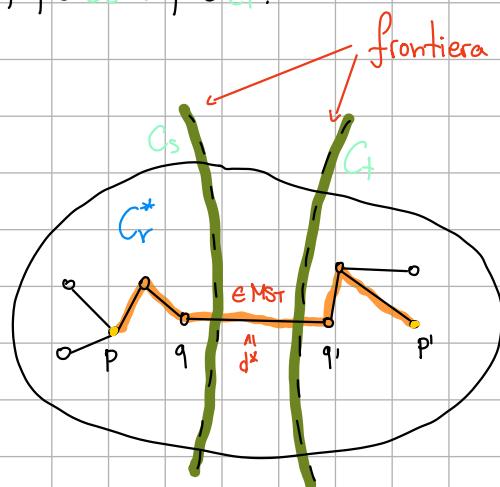
Ci sono due casi:

1)  $C = C^* \Rightarrow$  allora non bisogna dimostrare nulla in quanto  $\text{spacing}(C) = \text{spacing}(C^*)$

2)  $C \neq C^* \Rightarrow$  c'è almeno una differenza tra  $C$  e  $C^*$

↳ devono esistere 2 punti  $p$  e  $p'$  che sono nello stesso cluster in  $C^*$ , per comodità  $C_v^*$  ma sono in cluster diversi in  $C$ ,  $p \in C_s$  e  $p' \in C_t$ .

- Poiché  $p$  e  $p'$  ∈  $C_v^*$   $\Rightarrow$  allora dopo la rimozione dei  $k-1$  archi più pesanti  $p$  e  $p'$  appartengono ad una componente连通的 del MST, dunque per forza deve esistere un cammino da  $p$  a  $p'$  i cui archi fanno tutti parte del MST (perché giustamente questi archi non sono stati rimosso)



Non avendo rimosso tali archi, significa che il peso è  $\leq d^*$  (Per assurdo, se uno di quei archi avesse avuto un peso  $> d^*$  allora lo avremmo rimosso).

- Conclusione: Tale cammino attraversa la frontiera, in particolare deve esistere un arco  $(q, q)$  tale che  $q \in C_S$  e  $q \in C_+$  e il costo di tale arco abbiamo visto che è  $\leq d^*$ , ma allora lo spacing tra  $C_S$  e  $C_+$  è  $\leq d^*$ . Dunque la soluzione  $C$  non ha uno spacing più grande di  $d^*$  che è quello ottenuto dal algoritmo greedy.

Oss.: Nell'ambito dei BigData questo algoritmo dal punto di vista temporale non è efficiente in quanto per grafi molto densi ( $\Omega(n^2)$  archi) il tempo di esecuzione è  $\Omega(n^2)$ .

Cosa succede se  $K$  non viene dato in input?  $K$  molto spesso non è un parametro dato in input, ma un problema che va risolto.

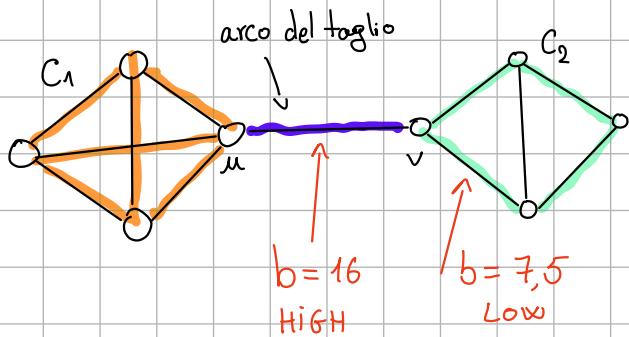
↳ Soluzione buona:

Provo tutti i  $K$ , ma qual è la soluzione migliore?

## COMMUNITY DETECTION

Dato un grafo non diretto e non pesato.

DEF: La edge betweenness è il numero di cammini minimi passanti per un arco.



Intuizione: Dato una comunità, (rete densa), su ogni arco all'interno della comunità non passano tanti cammini minimi perché dati 2 nodi molto probabilmente c'è l'arco diretto tra questi 2 nodi (il cammino minimo è a distanza 1).

Se invece si guarda l'arco del taglio tra due comunità  $C_1$  e  $C_2$ , tutti i cammini minimi da un qualunque nodo in  $C_1$  a un qualunque nodo in  $C_2$  devono passare per l'arco  $(u, v)$  e di conseguenza il numero di cammini minimi aumenta.

In generale: Se su un arco passano tanti cammini minimi del grafo, allora quel arco è un arco che fa parte di un taglio, connette 2 comunità distinte.

## Girvan-Newman Algorithm

Intuizione: Creare un clustering gerarchico basato sulla edge betweenness.

GN-Algo:

Input: Grafo non diretto e non pesato

Nota: Non abbiamo bisogno di  $k$ !!

While ci sono ancora archi

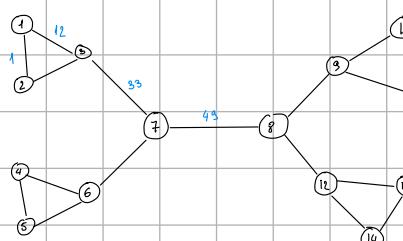
Calcolo lo betweenness di ciascun arco

Rimuovi gli archi con betweenness maggiore

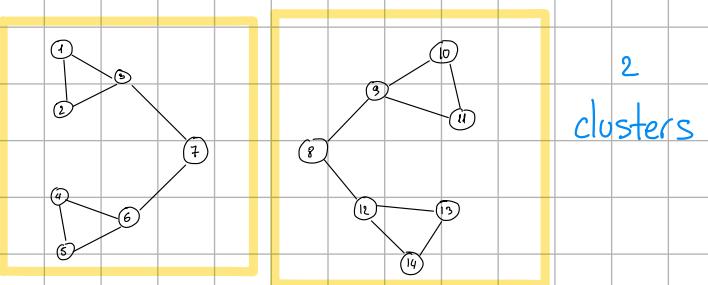
Le componenti fortemente connesse sono comunità

Return clustering gerarchico del grafo

→ Dopo ogni iterazione otteniamo un nuovo livello del clustering gerarchico.



1<sup>st</sup> STEP

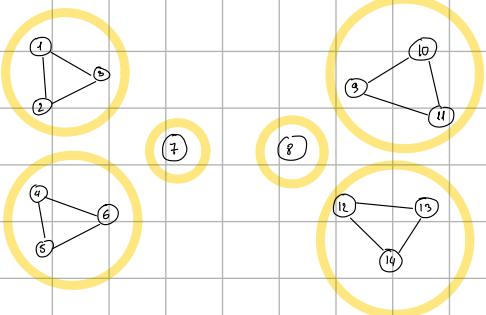


2 clusters

Rimuovendo l'arco con betweenness 49, otteniamo un clustering gerarchico di primo livello.

Dopo ogni passo, ovviamente è bisogno di ricalcolare lo betweenness per ogni arco.

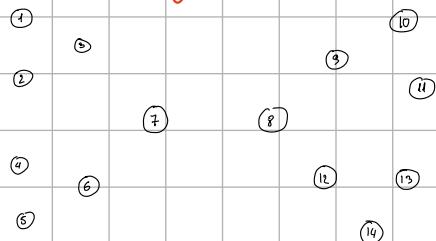
↓ 2<sup>nd</sup> Step



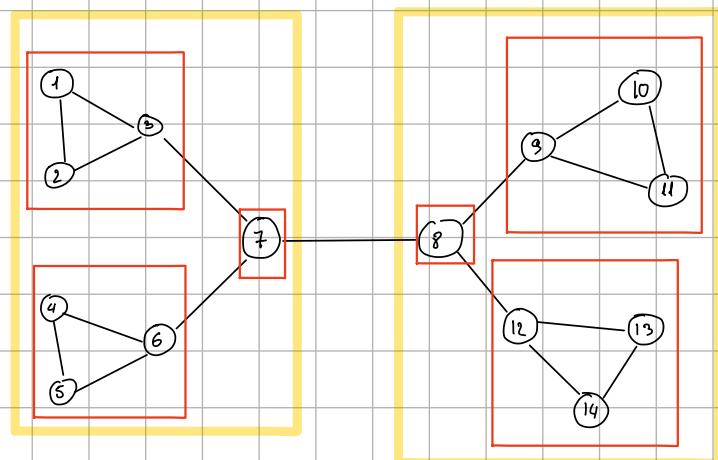
Clustering gerarchico di secondo livello.

6 clusters

|| 3<sup>rd</sup> step



Al terzo livello quello che succede poiché tutti gli archi hanno betweenness massima otteniamo  $n$  (14) clusters.



Dall' "alto" si può osservare una scomposizione in due livelli:

1) Come calcolare la betweenness

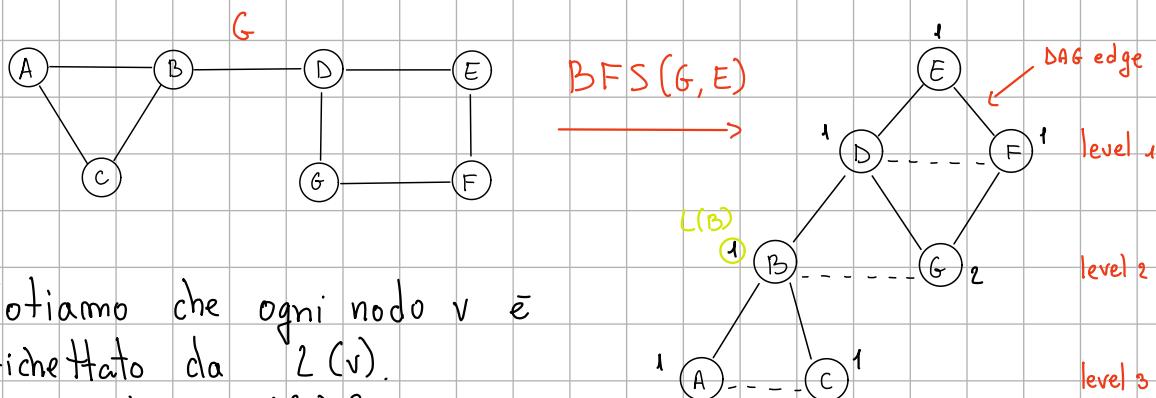
2) Come selezionare il numero di clusters

## COME CALCOLARE LA BETWEENNESS

FASE 1:  $\forall s \in V$ , calcola  $BFS(G, s) = \{L_1, L_2, \dots, L_d\}$  leggere nota  
 ↳ nodo sorgente struttura ad "albero" a livelli

Nel mentre che eseguo la BFS,  $\forall v \neq s$  calcolo  $L(v)$  che è il numero di shortest path che vanno da  $s$  a  $v$ .

NOTA: Gli archi tra livelli sono chiamati archi DAG (Directed Acyclic Graph). Ciascun arco DAG appartiene ad almeno un cammino minimo dalla sorgente  $s$ . Se esiste un arco DAG  $(y, z)$ , dove il nodo  $y$  si trova ad un livello superiore rispetto a  $z$ , allora chiameremo  $y$  **padre** di  $z$  e  $z$  **figlio** di  $y$ , sebbene i padri non sono unici in un DAG come sarebbero in un albero.



Notiamo che ogni nodo  $v$  è etichettato da  $L(v)$ .

Come calcolare  $L(v)$ ?

Si inizia etichettando la root con 1. Poi, dall'alto verso il basso etichettiamo ciascun nodo  $y$  con la somma delle etichette dei padri.

FASE 2: L'ultima fase consiste nel calcolare per ogni arco  $e$ , la somma su tutti i nodi  $y$  della frazione dei percorsi più brevi dal nodo radice  $X$  a  $Y$  che passano attraverso  $e$ .

Si procede dal basso verso l'alto.

Vengono dunque assegnati i crediti  $c(v)$  ad ogni nodo  $v \neq s$ , dove

$$\left\{ \begin{array}{l} c^s(v) = 1 \text{ se } v \text{ è una foglia (1)} \\ \rightarrow \text{frazione} \end{array} \right.$$

$$\left\{ \begin{array}{l} c^s(v) = 1 + \sum_{\substack{w \text{ child of} \\ v}} c^s(w) \cdot f_v^s(w) \text{ (2) dove} \\ f_v(w) = \frac{L(v)}{\sum L(v')} \end{array} \right.$$

$\hookrightarrow$  per ogni figlio  
(livello inferiore)

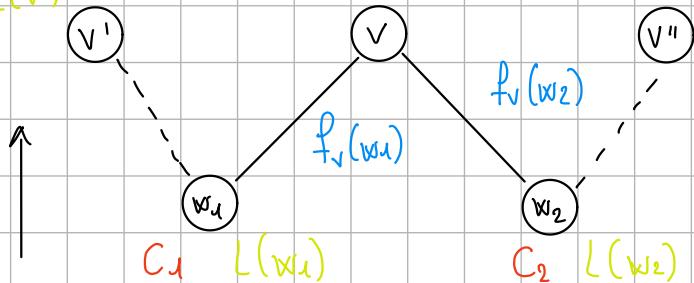
$v'$  parent of  $w$

$\hookrightarrow$  per ogni padre  
(livello superiore)

$$L(v) \quad c_v = ? \quad L(v)$$

$$L(v'')$$

Come si calcolano  
 $f_v(w_1)$  e  $f_v(w_2)$



$$f_v(w_1) = \frac{L(v)}{L(v) + L(v')}$$

FASE 3: Dalle equazioni 1 e 2, assegnamo a ciascun arco  $(w, v)$  il valore  $f_v^s(w)$  calcolato nella fase 2.

Dunque allo fine della fase 3, abbiamo che:

a)  $\forall$  sorgente  $s \in V$ ,  $\forall$  arco  $e = (w, v)$ , il valore

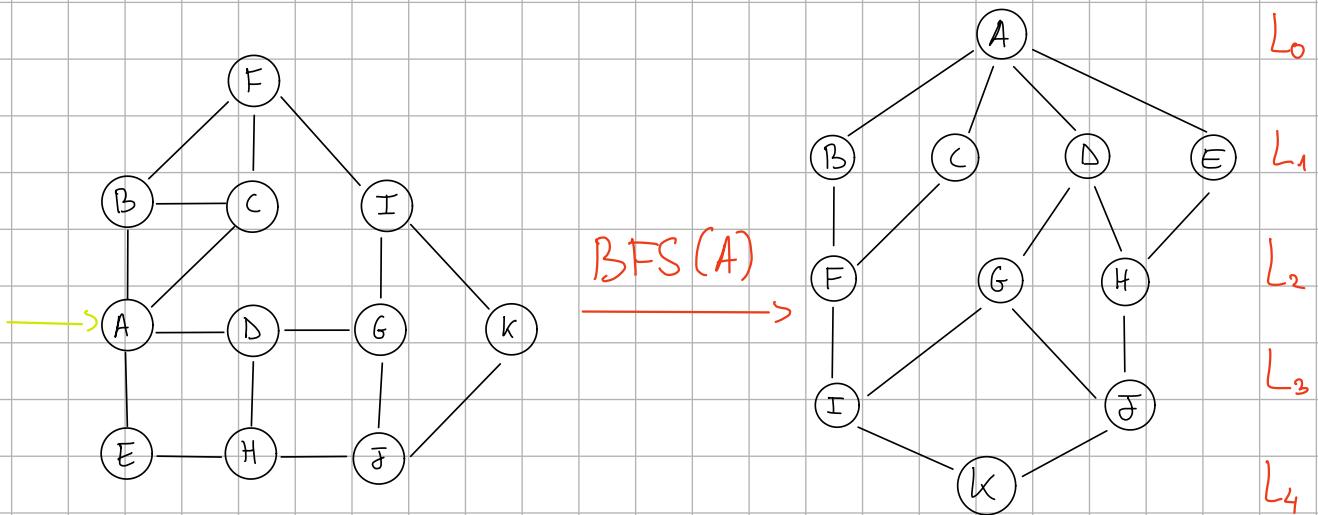
$$b^s(w, v) := f_v^s(w) \cdot c^s(v)$$

b) La betweenness di ogni arco  $(w, v)$  è

$$B(w, v) := \sum_{\substack{\text{sorgente} \\ s \in V}} b^s(w, v)$$

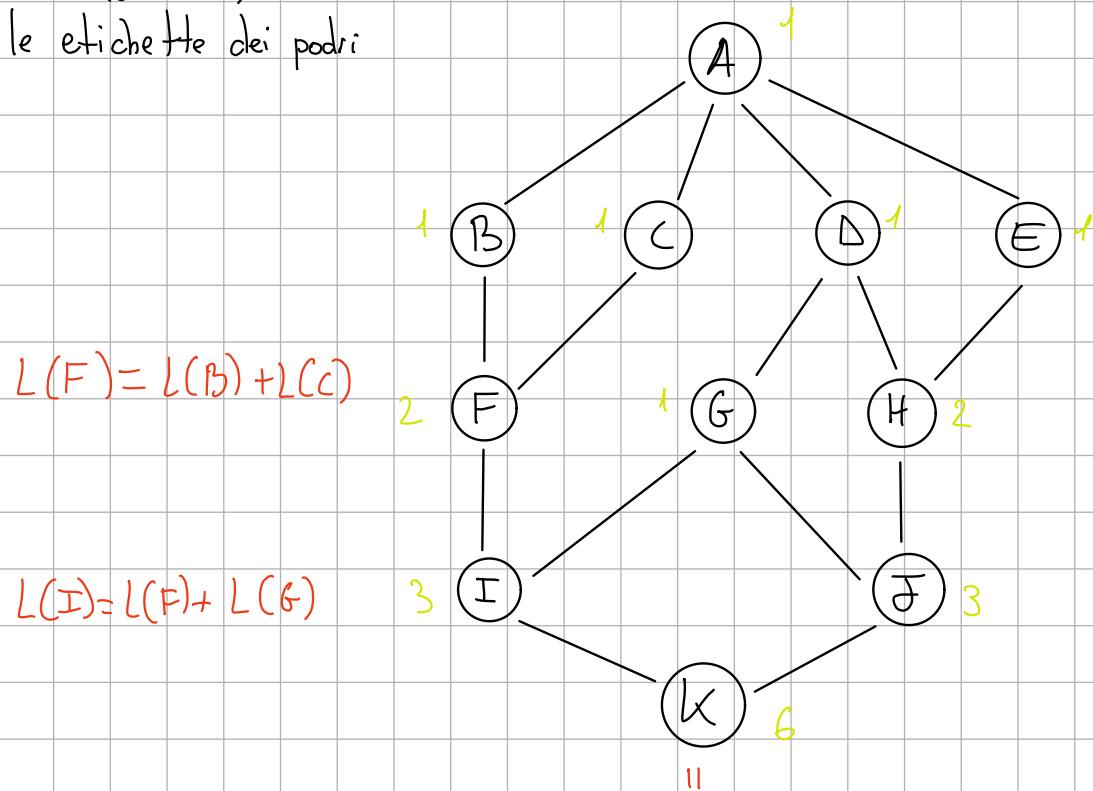
Nota: Se un arco  $e$  non appare in nessuna BFS, allora la sua betweenness è 0.

Esempio: Vogliamo calcolare la betweenness dei archi partendo da A.



Adesso dobbiamo calcolare  $L(\cdot)$ , ovvero il numero di shortest paths da A a tutti gli altri modi.

Al nodo  $v \neq A$ , sommiamo le etichette dei nodi:



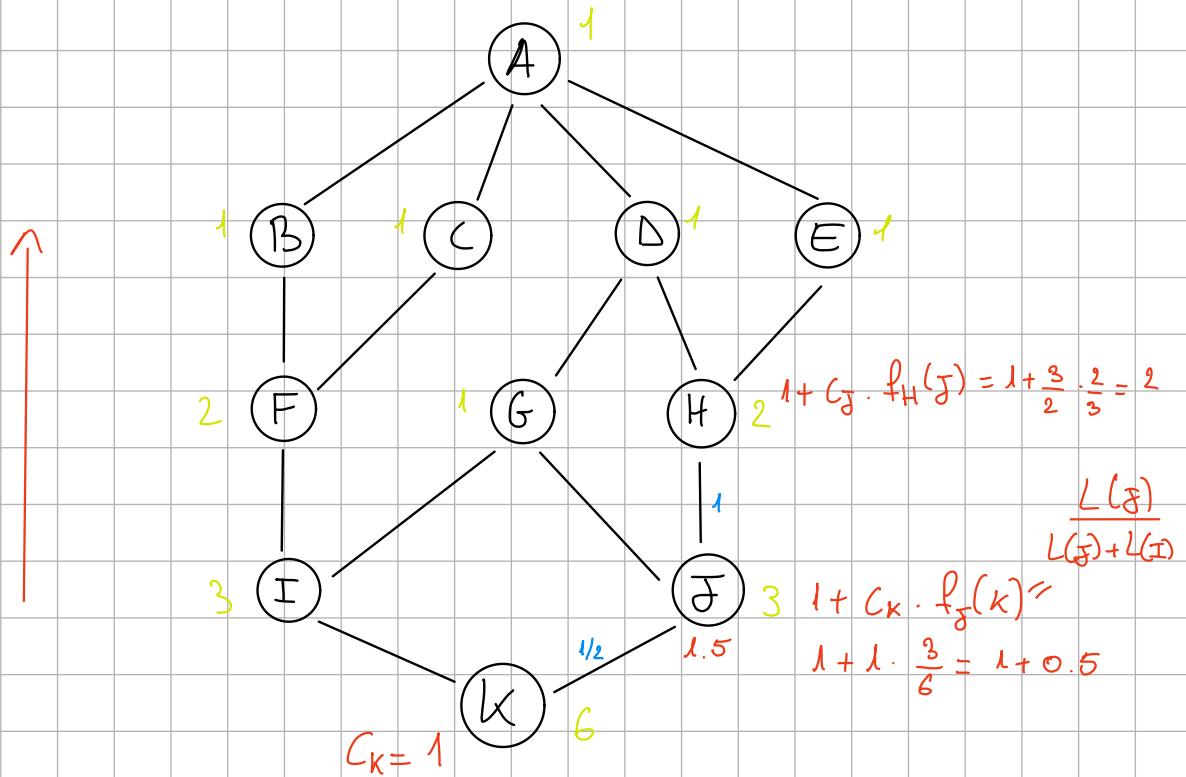
$$L(F) = L(B) + L(C)$$

$$L(I) = L(F) + L(G)$$

$$\begin{aligned}
 & \text{Shortest } A-K \text{ paths} \\
 & = \text{Shortest } A-J \text{ paths} \equiv L(K) = L(I) + L(J) \\
 & + \text{Shortest } A-I \text{ paths}
 \end{aligned}$$

Adesso calcoliamo lo betweenness ripercorrendo l'albero dal basso verso l'alto: Se ci sono percorsi multipli, conta la frazione.

- $L(\cdot)$
- $c(v)$



$$c(v) = 1 + \sum_{w \in \text{sons}(v)} c(w) \cdot f_v(w)$$

$$\hookrightarrow \frac{L(v)}{\sum_{v' \in p(w)} L(v')}$$

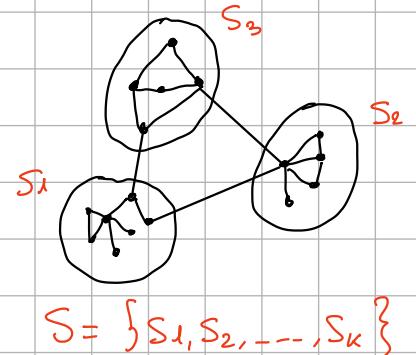
## COME SELEZIONARE il numero di CLUSTERS

Definiamo modularità  $Q$  una misura per capire quanto è buono il partitionamento di una rete.

Data una partitione della rete in clusters se  $S$

$$Q \propto \sum_{s \in S} [(\# \text{archi in } s) - (\# \text{atteso di archi in } s)]$$

si sa calcolare      ↓ ?  
 ↓  
 L'è proporzionale



Dato un grafo  $G$  di  $m$  nodi e  $m$  archi, costruire il grafo  $G'$ .

Inoltre impongo una distribuzione sui gradi.

L'è multigrafo

Calcolare il numero atteso di archi in un grafo in cui ho  $m$  archi complessivi e dati due nodi  $i$  e  $j$   $\deg(i) = k_i$  e  $\deg(j) = k_j$ .

Il numero atteso di archi fra 2 nodi  $i$  e  $j$  è:  $k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$

$\Rightarrow$  Il numero atteso di archi nel multigrafo random  $G'$  è:

L'è per non prendere lo stesso arco 2 volte.

$$\begin{aligned} \frac{1}{2} \sum_{i \in N} \sum_{j \in N} \frac{k_i k_j}{2m} &= \frac{1}{2} \cdot \frac{1}{2m} \sum_{i \in N} k_i \sum_{j \in N} k_j \\ &= \frac{1}{4m} 2m \cdot 2m = m \end{aligned}$$

NOTA:  $\sum_{i \in N} k_i = 2m$

$$Q \propto \sum_{s \in S} [(\# \text{archi in } s) - (\# \text{atteso di archi in } s)]$$

L'è se esiste l'arco in  $S$

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

L'è fattore di normalizzazione

La modularità è un valore compreso tra  $[-1, 1]$ .

È positivo quando il numero di archi dentro un gruppo è superiore rispetto al numero che ci aspettiamo.

Se  $Q > 0.3 - 0.7$ , allora significa che abbiamo individuato una buona struttura.