Community detection

Esteban Bautista-Ruiz, Lionel Tabourier

LIP6 – CNRS and Université Pierre et Marie Curie

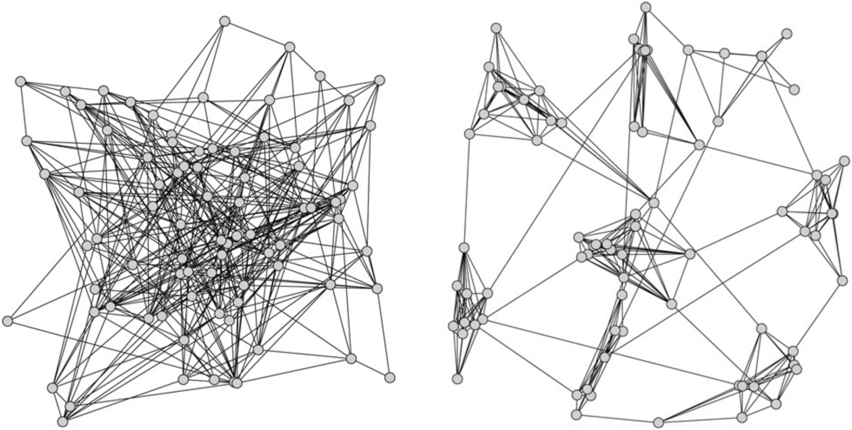
first_name.last_name@lip6.fr

October 26^{th} 2021

Outline

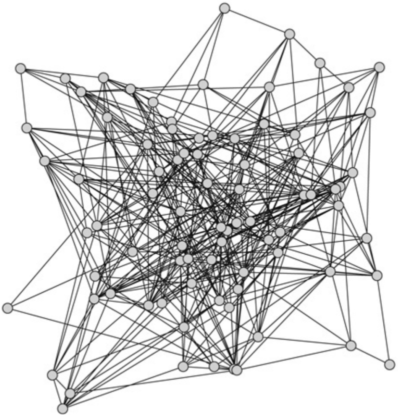
1. Introduction
   - Motivation
   - Community definitions
   - Measures for identifying communities
2. Global methods for finding communities
   - Label propagation algorithm
   - Louvain algorithm
3. Local methods for finding communities
   - Personalized PageRank
   - PageRank nibble

Motivation

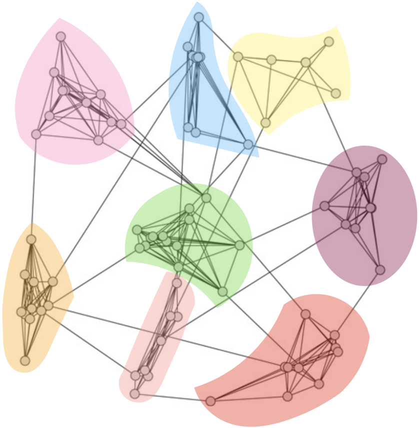


Difference?

Motivation



Random graph



Graph with communities

Applications

Community structure

Groups of nodes more densely connected between them than towards the rest of the network.

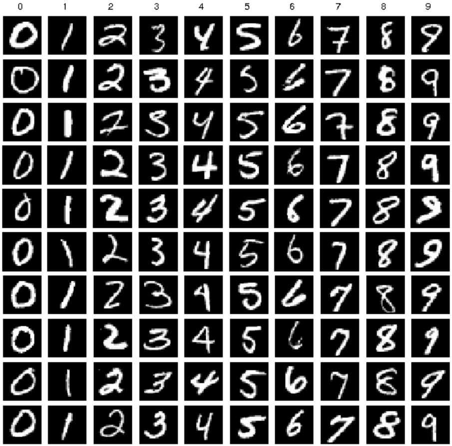
Goal

Automatically identify communities

Applications

- Recommendation systems
- Organize websites by topic
- Epidemic spreading
- Data clustering
- ...

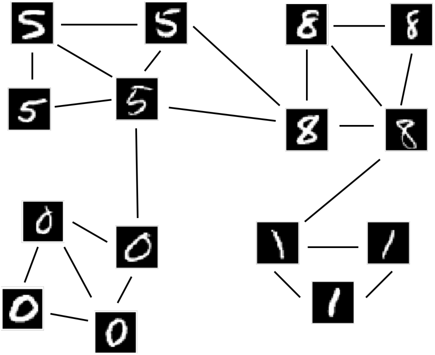
Community detection for data clustering



MNIST: images of handwritten digits

How to automatically organize images of same digits?

Community detection for data clustering



K nearest neighbors graph

Data instances linked to their K closest neighbors.

Edge weight proportional to similarity

- $w_{i,j} = \|x_i - x_j\|^{-1}$
- $w_{i,j} = exp\{-\frac{\|x_i - x_j\|^2}{\sigma}\}$
- $w_{i,j} = \langle x_i, x_j \rangle$

Community definitions

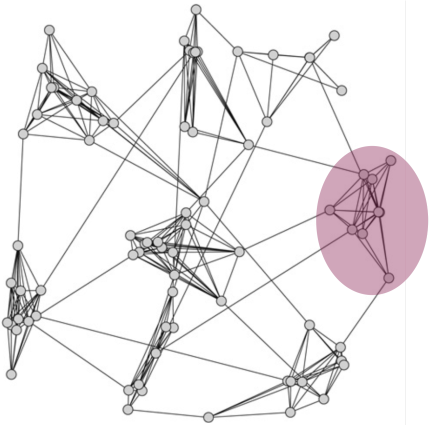
Desirable community properties

- Communities should be connected
  - At least one path between any two vertices of the community
  - Paths should only vertices of the community
- Community densities should be higher than the graph density

Community definitions

- Loosest definition: connected components ($\mathcal{O}(n + m)$ with BFS)
- Strictest definition: maximal cliques (NP-complete)
- Common definition: something in between (NP-hard)

How to objectively assess if a group of nodes is a community?



Three main approaches:

- Density-based metrics
- Modularity-based metrics
- Graph cut-based metrics

Density-based community detection

- **Graph** $G = (V, E)$: m links, n nodes
- **Group** $S \subseteq V$: subset of vertices
- **Degree** $d(u)$: split as $d(u) = d(u)_{in} + d(u)_{out}$ (links to S and S^c)

Rationale: Nodes in S should be more connected to S than to S^c, hence $d(u)_{in} >> d(u)_{out}$, for all $u \in S$.

Community detection task

Find the disjoint partitioning $V = S_1 \cup \cdots \cup S_k$ that maximizes the following quantity:

$$\sum_{i=1}^{k} \sum_{v \in S_i} d(v)_{in} - d(v)_{out}$$

• Necessary to constraint k, otherwise favors outliers.

Modularity-based community detection

Useful definitions

- Volume of S: $vol(S) = \sum_{u \in S} d(u)$
- Volume of G: $vol(G) = \sum_{u \in V} d(u)$

In a random graph with fixed degree distribution

- Probability for an edge endpoint to fall in S: $\frac{vol(S)}{vol(G)}$
- Probability for a link to be in S: $\frac{vol(S)^2}{vol(G)^2}$
- Expected number of links in S: $\frac{vol(G)}{2} \cdot \frac{vol(S)^2}{vol(G)^2} = \frac{vol(S)^2}{2vol(G)}$

Modularity-based community detection

Rationale: The actual number of links in S should be higher than the expected number of links in a comparable random graph. Hence:

$$\sum_{u \in S} \frac{d(u)_{in}}{2} > \frac{vol(S)^2}{2vol(G)}$$

Community detection task

Find the disjoint partitioning $V = S_1 \cup \cdots \cup S_k$ that maximizes the following modularity quantity:

$$Q = \sum_{i=1}^{k} \sum_{u \in S_i} \frac{d(u)_{in}}{vol(G)} - \frac{vol(S_i)^2}{vol(G)^2}$$

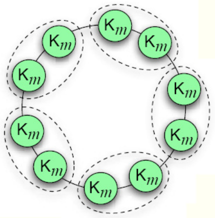
•$Q \in [-0.5, 1]$.

Known problem: resolution limit

Ring of cliques: α cliques of size β

$$Q_{single} = 1 - \frac{2}{\beta(\beta-1)+2} - \frac{1}{\alpha}$$

$$Q_{pairs} = 1 - \frac{1}{\beta(\beta-1)+2} - \frac{2}{\alpha}$$



Known problem: resolution limit

Ring of cliques: α cliques of size β

$$Q_{single} > Q_{pairs} \iff \beta(\beta - 1) + 2 > \alpha$$

Suppose 30 cliques of size 5 then:

- $\alpha = 30$ and $\beta(\beta - 1) + 2 = 22 \Rightarrow Q_{single} < Q_{pairs}$
- $Q_{single} = 0.876$, $Q_{pairs} = 0.888$

counter-intuitive

Tendency to favour large communities...
... may appear at any length scale

Graph cut-based community detection

- **Graph cut**: edges between S and S^c

$$\text{cut}(S, S^c) = \sum_{u \in S} \sum_{v \in S^c} w_{uv}$$

- **Conductance**: ratio of external and internal edges of S

$$h_S = \frac{\text{cut}(S, S^c)}{min(vol(S), vol(S^c))}$$

Graph cut-based community detection

Rationale: A community S should have more links internally than externally, hence a small conductance.

Community detection task

Find the disjoint partitioning $V = S_1 \cup \cdots \cup S_k$ that minimizes the graph conductance:

$$h_G = \frac{1}{k} \sum_{i=1}^{k} h_{S_i}$$

• $h_G \in [0, 1]$.

Outline

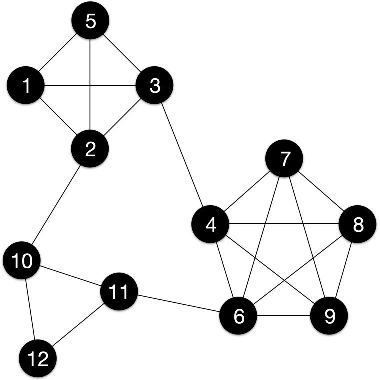
Label propagation algorithm

Near linear time algorithm to detect community structures in large-scale networks - *Raghavan et al.*

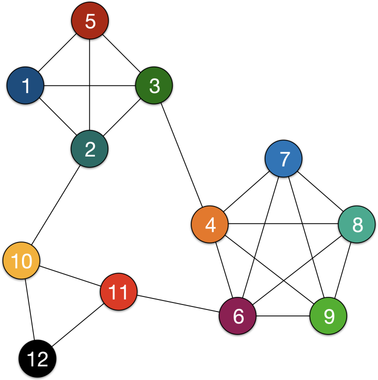
- **Step 1:** give a unique label to each node in the network
- **Step 2:** Arrange the nodes in the network in a random order
- **Step 3:** for each node in the network (in this random order) set its label to a label occurring with the highest frequency among its neighbours
- **Step 4:** go to 2 as long as there exists a node with a label that does not have the highest frequency among its neighbours.

Ties resolved randomly

Example



Initial network

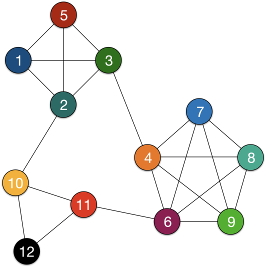


Step 1

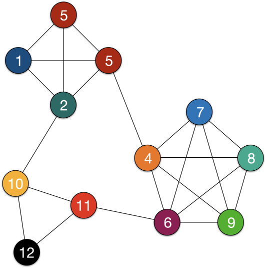
Example

Step 2: random order of vertices $[3, 8, 12, 2, 5, 9, 1, 7, 4, 10, 6, 11]$

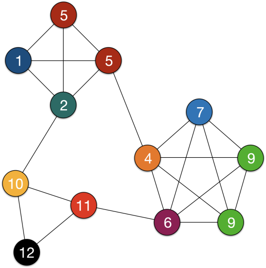
Step 3:



Init assignment



Processing node 3

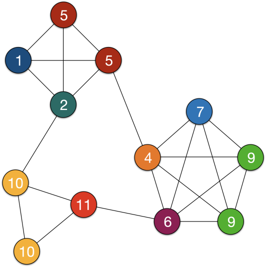


Processing node 8

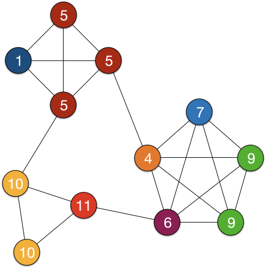
Example

Step 2: random order of vertices $[3, 8, 12, 2, 5, 9, 1, 7, 4, 10, 6, 11]$

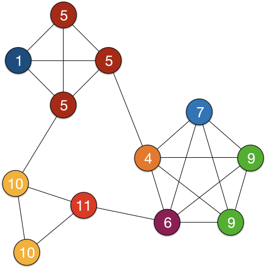
Step 3 (continuation):



Processing node 12



Processing node 2

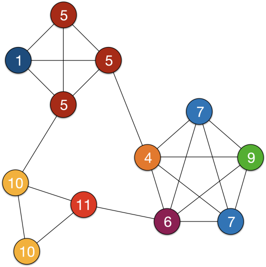


Processing node 5

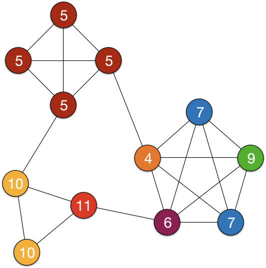
Example

Step 2: random order of vertices $[3, 8, 12, 2, 5, 9, 1, 7, 4, 10, 6, 11]$

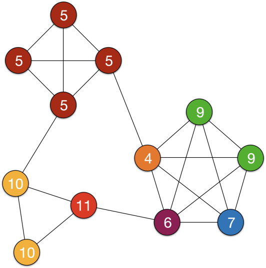
Step 3 (continuation):



Processing node 9



Processing node 1

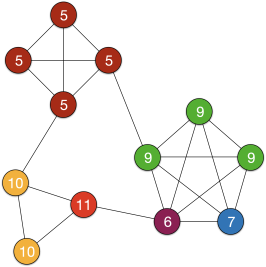


Processing node 7

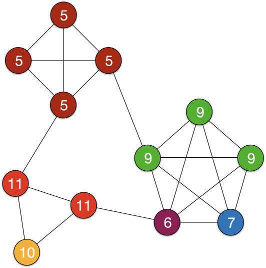
Example

Step 2: random order of vertices $[3, 8, 12, 2, 5, 9, 1, 7, 4, 10, 6, 11]$

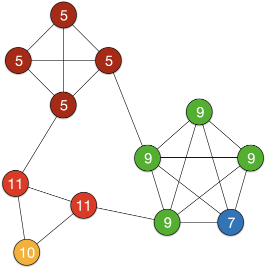
Step 3 (continuation):



Processing node 4



Processing node 10

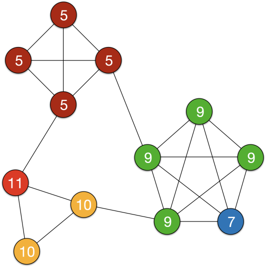


Processing node 6

Example

Step 2: random order of vertices $[3, 8, 12, 2, 5, 9, 1, 7, 4, 10, 6, 11]$

Step 3 (continuation):



Processing node 11

Not all nodes assigned to the majority class of the neighbors.

We repeat step 2 and step 3

Louvain algorithm

- **Step 1.** Initialization: node = community
- **Step 2.** Remove node u from its community
- **Step 3.** Insert node u in a neighboring community that maximizes Q
- **Step 4.** Iterate from step 1 until the partition does not evolve

Louvain algorithm

- **Step 1.** Initialization: node = community
- **Step 2.** Remove node *u* from its community
- **Step 3.** Insert node *u* in a neighboring community that maximizes Q
- **Step 4.** Iterate from step 1 until the partition does not evolve

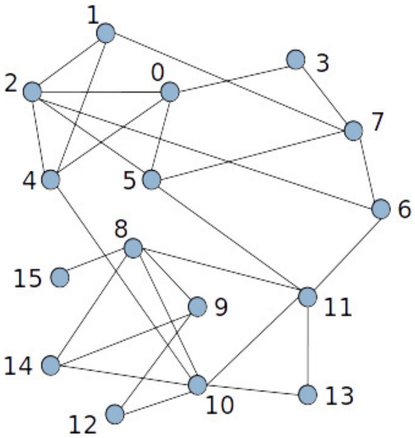
Can be trapped in bad local minima

Louvain algorithm

- **Step 1.** Initialization: node = community
- **Step 2.** Remove node *u* from its community
- **Step 3.** Insert node *u* in a neighboring community that maximizes Q
- **Step 4.** Iterate from step 1 until the partition does not evolve
- **Step 5.** Transform the communities into (hyper-)nodes and go back to step 1 with the new graph

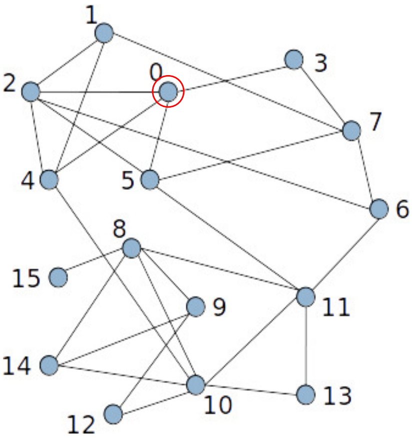
Leads to better local optima

Example



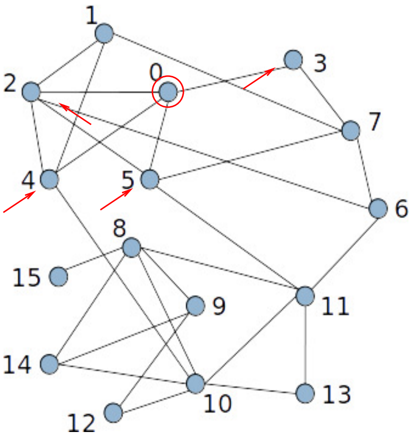
First passage, first iteration: isolated nodes

Example



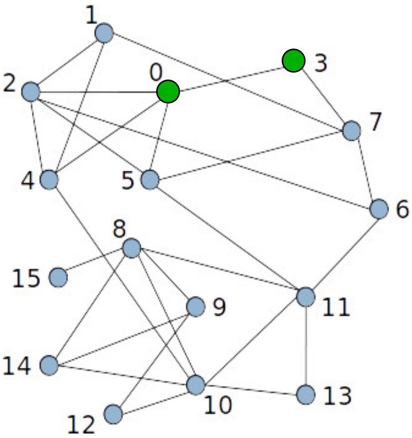
considering 0...

Example



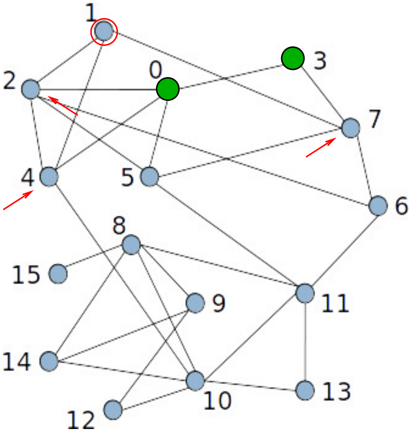
its neighboring communities are...

Example



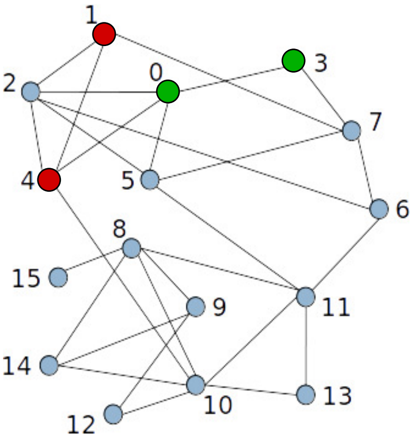
0 is put in C(3), best Q increase

Example



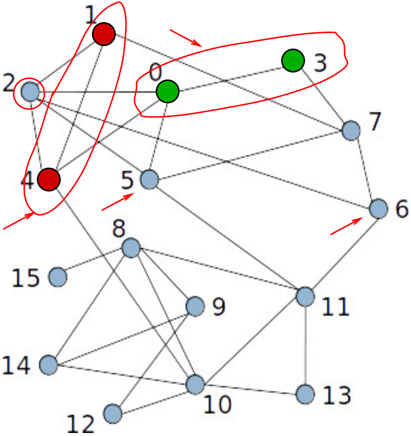
considering 1, its neighboring communities are...

Example



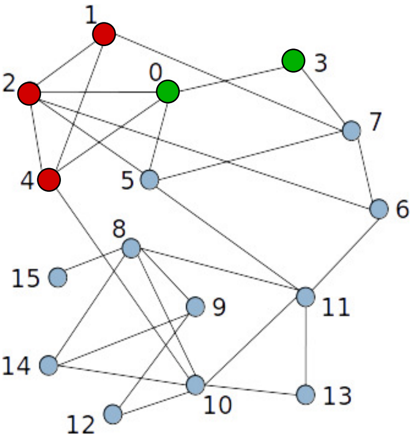
1 is put in C(4), best Q increase

Example



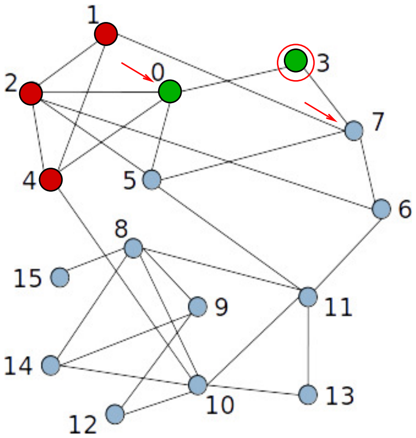
considering 2, its neighboring communities are...

Example



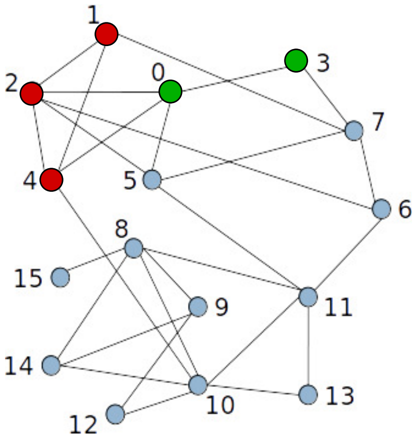
2 is put in C(1,4), best Q increase

Example



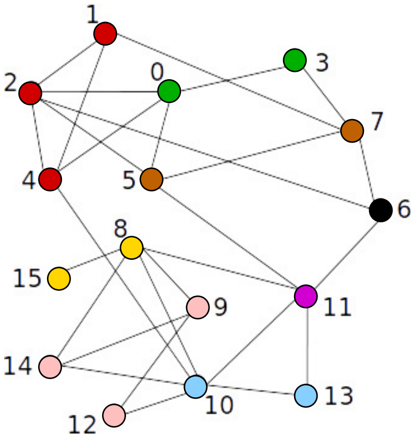
considering 3, its neighboring communities are...

Example



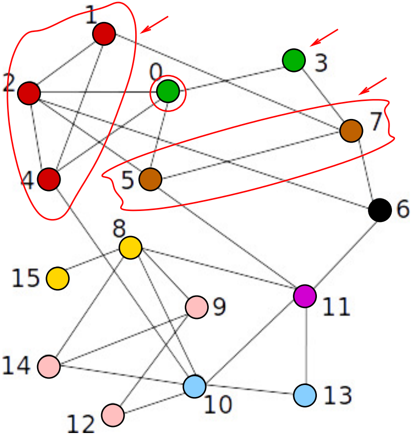
3 stays in the same community C(0,3), otherwise Q decreases

Example



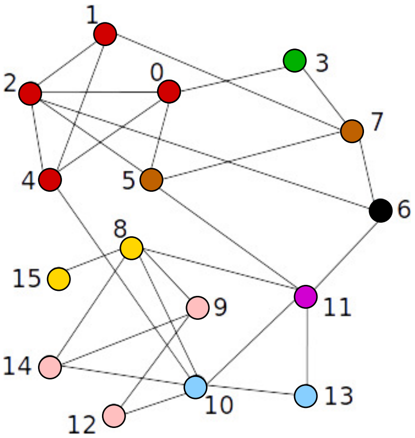
and so on...

Example



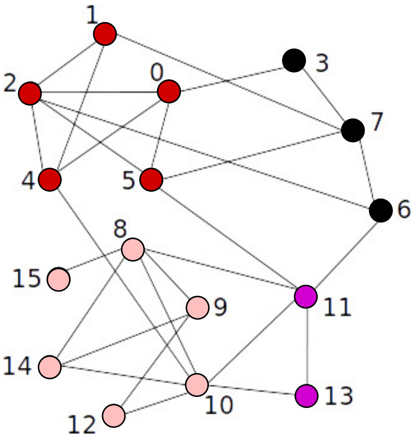
First passage, second iteration: considering 0...

Example



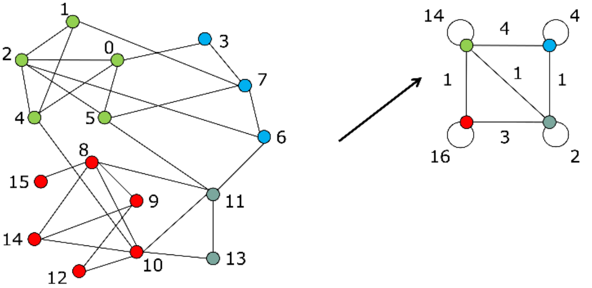
0 is put in C(1,2,4), best Q increase

Example



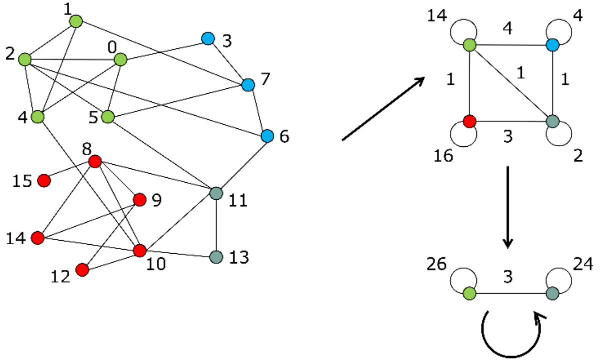
after 4 iterations, no change anymore

Example



Second passage

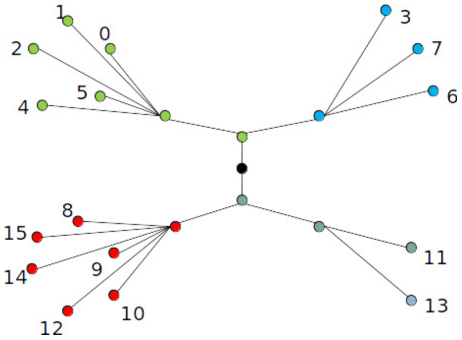
Example



Third passage

Example

Outcome: non-binary dendrogram



Evaluating and comparing algorithms

How to evaluate the quality of the algorithm's output?

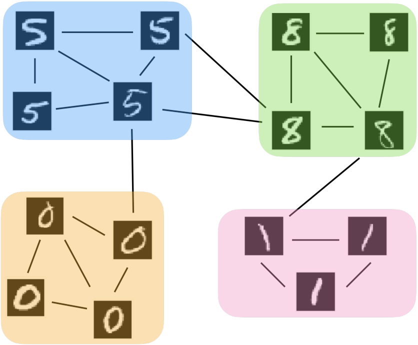
If no extra information is available

Measure modularity, density, conductance, etc

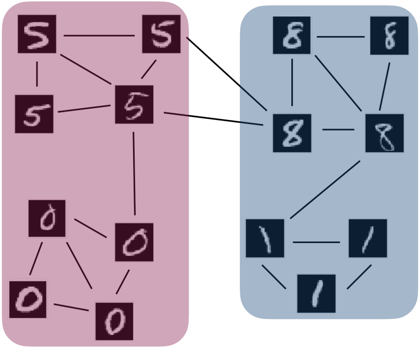
If a dataset with ground truth communities is available

Measure normalized mutual information

Normalized mutual information



Ground truth communities



Algorithm assignment

Normalized mutual information

Normalized mutual information

Score to evaluate a community assignment when true communities are known:

$$NMI(T, C) = \frac{2\ \mathcal{I}(T, C)}{H(T) + H(C)}$$

- T : ground truth labels
- C : algorithm labels
- H : Community entropies: log of samples per label.
- $\mathcal{I}(T, C)$: Mutual information (log of correlation between gt labels and algo labels).

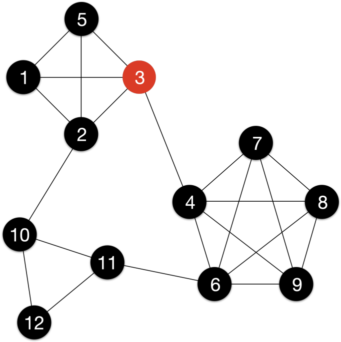
NMI score between 0 (no mutual information) and 1 (perfect correlation)

Full details in : `https://course.ccs.neu.edu/cs6140sp15/7_locality_cluster/Assignment-6/NMI.pdf`

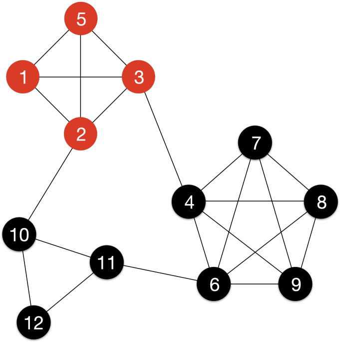
Outline

Local community detection

How to identify the community of a seed node?

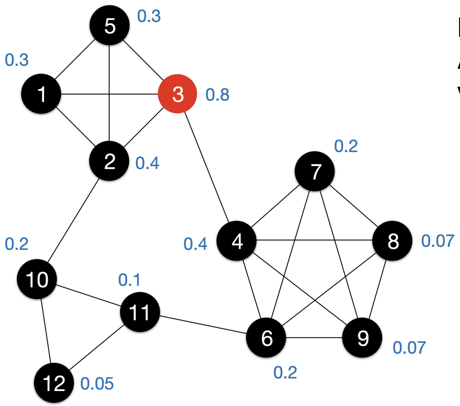


Seed node



Algorithm output

Personalized PageRank



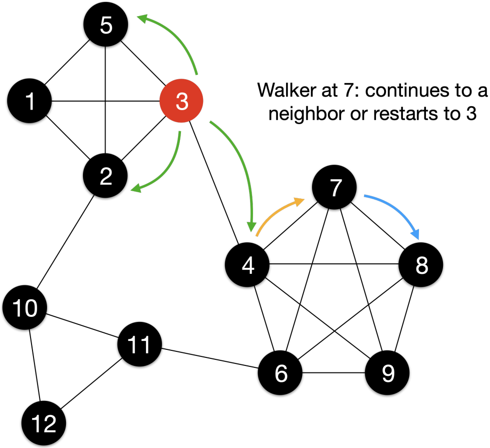
Personalized PageRank:
Algorithm to rank the importance of vertices with respect to a seed.

- Proposed in seminal paper by Brin and Page, 1999 (http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf)
- Basis of Google's search engine

Personalized PageRank Algorithm

- **Step 1.** Choose seed node
- **Step 2.** Start a random walker from the seed node
- **Step 3.** After each jump, continue the walk with probability α or restart it with probability $1 - \alpha$.
- **Step 4.** After each jump, assign the fraction of visits that the walker has done to u as the PageRank score of node u.
- **Step 5.** Repeat 3 and 4 until convergence of the scores.

Personalized PageRank Algorithm



PageRank score: probability of finding the walker at a node

PageRank nibble

Rationale: It should be hard for a random walker that starts within a community to leave the community.

- **Step 1.** Compute personalized PageRank
- **Step 2.** Order the vertices of the graph from the one of largest PageRank score to the lowest
- **Step 3.** Take the first k vertices of this new ordering as a test community and measure its conductance
- **Step 4.** Repeat step 3 for all $k \in [1, n]$.
- **Step 5.** From the tested communities, return the one with smallest conductance