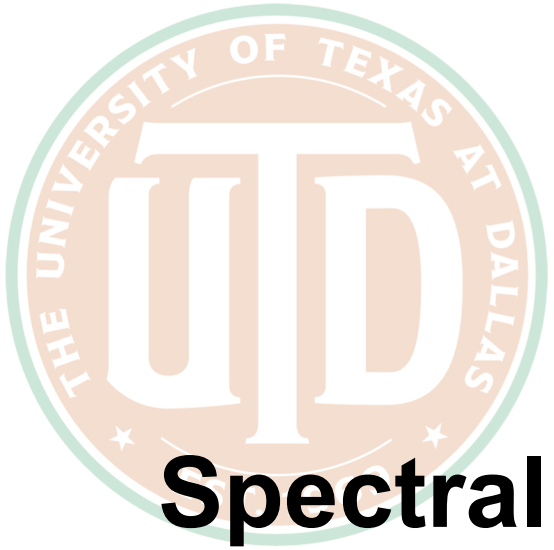


Spectral Clustering for Community Detection

[1] **A tutorial on spectral clustering,** *Ulrike von Luxburg, Stat Comput (2007)*

[2] **Covariate-assisted spectral clustering,** Binkiewicz, Norbert, Joshua T. Vogelstein, and Karl Rohe, *Biometrika* (2017)

[3] **Spectral clustering-based community detection using graph distance and node attributes,** Tang, F., Wang, C., Su, J., & Wang, Y. *Computational Statistics* (2020)

Presenter: Guihong Wan, Nov/2020

Outline

- **Introduction**
  - Problem setup and notations
  - Graph Laplacians and basic properties
- **Spectral clustering**
  - Algorithms
  - Evaluation
- **Spectral clustering-based community detection**
  - CASC
  - DCFI

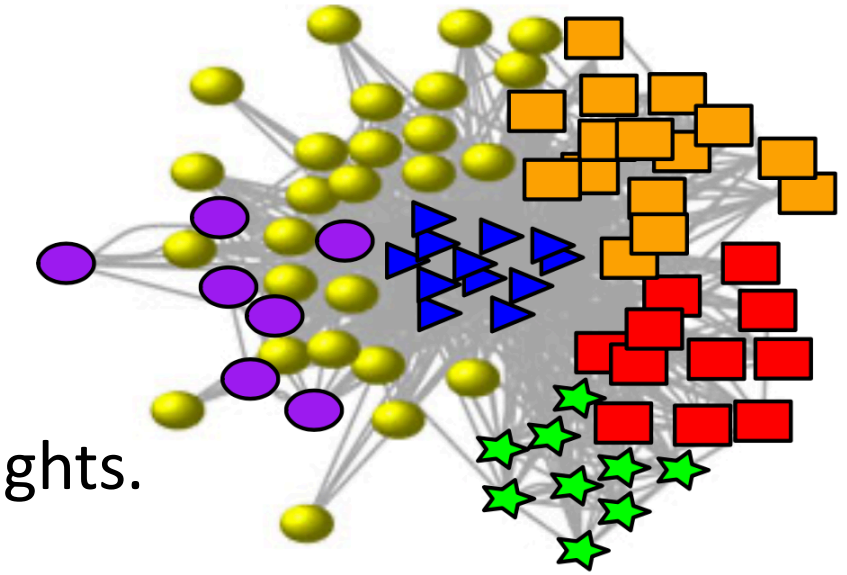
Introduction

Problem setup

Given an undirected, weighted graph G(V,E)

Goal : find a partition of nodes such that

1. edges between different groups have low weights;
2. edges within a group have high weights.



- One family of algorithms to solve this problem is spectral clustering.
- It has been used to cluster graph nodes using functions of the adjacency matrix.

Notations

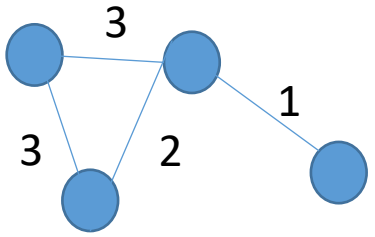
An undirected, weighted graph G(V,E)

- Nodes: $V = \{v_1, \dots, v_n\}$.
- Weighted adjacency matrix : W, edge between v_i and $\mathrm{v_j}$ $carries\ w_{ij} \geq 0$.
- Degree matrix: D with degrees d_i on the diagonal.
- The subset of nodes: $A \subset V$
- Consider two ways of measuring the size of a subset $A \subset V$:

|A| := the number of vertices in A

$vol(A) := \sum_{i \in A} d_i$

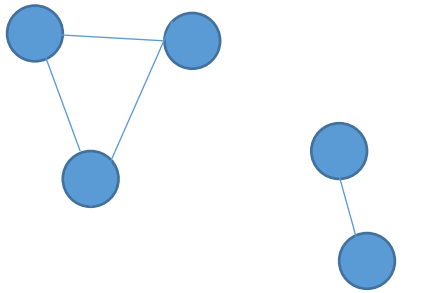
$vol(A)$ measures the size of A by the weights of its edges.



6 + 5 + 6 + 1 = 18

Notations (cont.)

- A subset A_i is a connected component if
  1. it is connected
  2. if there are no connections between nodes in A_i and A_j.
- The sets $A_1, \ldots, A_k$ form a partition of the graph if
  1. $A_i \cap A_j = \emptyset$;
  2. $A_1 \cup \cdots \cup A_k = V$.



A_1 A_2

Graph Laplacians

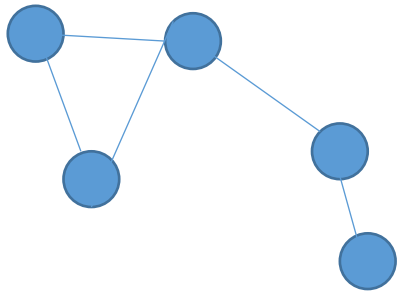
The main tools for spectral clustering are graph Laplacian matrices.

- Unnormalized graph Laplacian: L = D – W
  - the smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector 1.
  - L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \lambda_n$.
  - Number of connected components: the number of 0 eigenvalues.

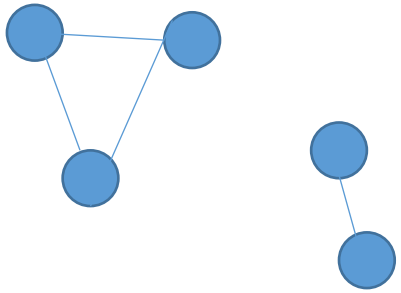
Graph Laplacians (cont.)

Number of connected components: the number of 0 eigenvalues.

(0, 1, 2, 3, 4)



(0, 0, 1.5, 2.7, 3.9)



Graph Laplacians (cont.)

Consider the case of k connected components

Assume that the nodes are ordered according to the connected components they belong to, then

- L has a block diagonal form;
- Each L_i is a graph Laplacian of the connected component.

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}$$

L has as many eigenvalues 0 as there are connected components, and the corresponding eigenvectors are the indicator of the connected components.

Graph Laplacians (cont.)

Normalized graph Laplacians:

$$\mathrm{L}_{sym} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$

$$L_{rw} := D^{-1} L$$

- λ is an eigenvalue of L_{rw} with eigenvector v iff v solve the generalized eigenproblem $Lv = \lambda Dv$;
- Number of connected components: number of 0 eigenvalues.

Spectral Clustering

Spectral Clustering Algorithms

Algorithm 1: using unnormalized Laplacian

- Compute the unnormalized Laplacian L.
- **Compute the first k eigenvectors $v_1, \ldots, v_k$ of L.**
- Let $V \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $v_1, \ldots, v_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of V.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

Three main steps:

1. Compute Laplacian
2. Compute V
3. Run kmeans on V

Spectral Clustering Algorithms (cont.)

Algorithm 2: using normalized Laplacian L_{rw}

- Compute the unnormalized Laplacian L.
- **Compute the first k eigenvectors $v_1, \ldots, v_k$ of the generalized eigenproblem $Lv = \lambda Dv$.**
- Let $V \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $v_1, \ldots, v_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of V.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

Spectral Clustering Algorithms (cont.)

Algorithm 3: using normalized Laplacian L_{sym}

- Compute the normalized Laplacian L_{sym}.
- **Compute the first k eigenvectors $v_1, \ldots, v_k$ of L_{sym}.**
- Let $V \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $v_1, \ldots, v_k$ as columns.
- **Form the matrix $U \in \mathbb{R}^{n \times k}$ from V by normalizing the row sums to have norm 1,** that is $u_{ij} = v_{ij} / (\sum_k v_{ik}^2)^{1/2}$.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

Spectral Clustering Algorithms (cont.)

- All three algorithms look rather similar, apart from the fact that they use different graph Laplacian L, L_{rw}, L_{sym}
- The main trick is to use y_i to represent each node.

Three main steps:

1. Compute Laplacian;
2. Compute V ← first k eigenvectors;
3. Run k-means on rows of V.

$$V_{n\times k} = \begin{bmatrix} u_{11} & \cdots & u_{k1} \\ \vdots & \ddots & \vdots \\ u_{1n} & \cdots & u_{kn} \end{bmatrix}$$

y_1

First k eigenvectors

Spectral Clustering Algorithms (cont.)

$$\begin{pmatrix} 1 & -1/\sqrt{6} & -1/2 & 0 & 0 & 0 \\ -1/\sqrt{6} & 1 & -1/\sqrt{6} & -1/3 & 0 & 0 \\ -1/2 & -1/\sqrt{6} & 1 & 0 & 0 & 0 \\ 0 & -1/3 & 0 & 1 & -1/\sqrt{6} & -1/\sqrt{6} \\ 0 & 0 & 0 & -1/\sqrt{6} & 1 & -1/2 \\ 0 & 0 & 0 & -1/\sqrt{6} & -1/2 & 1 \end{pmatrix}$$

With $w_{ij} = 1$ for all drawn edges we have:

$$W = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

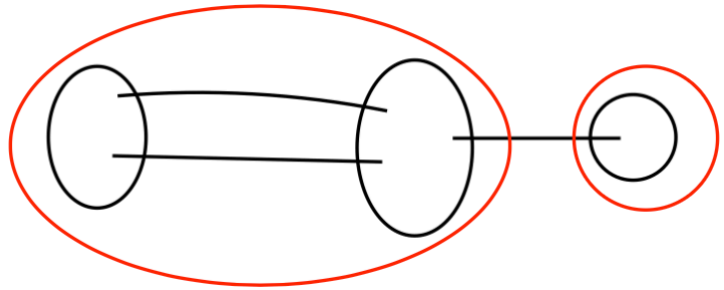
$$u_0 = (0.37796, 0.46291, 0.37796, 0.46291, 0.37796, 0.37796)$$
$$u_1 = (-0.44514, -0.32202, -0.44514, 0.32202, 0.44514, 0.44514)$$

y_1 y_2 y_3 y_4 y_5 y_6

Normalized cuts

Consider a partition of the graph nodes into two disjoint subsets: S^+ and S^- .
The weight of the cut that separates S^+ and S^- is given by:

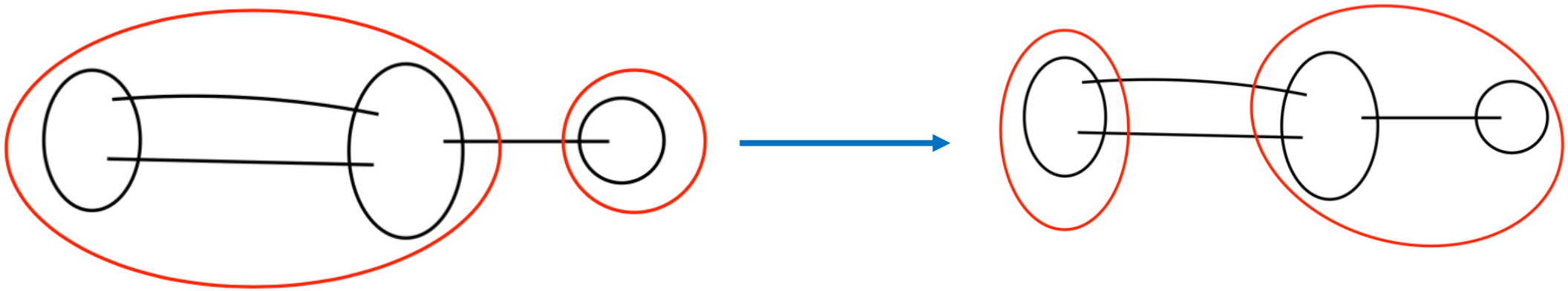
$$C(S^+, S^-) = \sum_{i \in S^+, j \in S^-} w_{ij}$$



A minimal cut will typically cut out a node in the outliers,
Intuitively we would like to compute a partition into two "large" subsets.

Normalized cuts (cont.)

A minimal cut will typically cut out a node in the outliers,
Intuitively we would like to compute a partition into two
"large" subsets.



Normalized cuts (cont.)

Recall: $vol(A) := \sum_{i \in A} d_i$

Intuitively, we would like to partition the graph into subsets of large volume.

Here are two possible functions that measure that:

$$N_h(S^+, S^-) = \min\{vol(S^+), vol(S^-)\},$$

$$N_b(S^+, S^-) = \frac{1}{\frac{1}{vol(S^+)} + \frac{1}{vol(S^-)}}.$$

Normalized cuts (cont.)

$$N_h(S^+, S^-) = \min\{vol(S^+), vol(S^-)\},$$

$$N_b(S^+, S^-) = \frac{1}{\frac{1}{vol(S^+)} + \frac{1}{vol(S^-)}}.$$

Using N_h: $h(S^+, S^-) = \frac{C(S^+, S^-)}{N_h(S^+, S^-)}$

Using N_b: $b(S^+, S^-) = \frac{C(S^+, S^-)}{N_b(S^+, S^-)}$

We would like to compute partition which minimizes h or b.
Both cases are known to be NP-hard.

Spectral clustering for community detection

Spectral clustering for community detection

If data matrix $X = (x_1, \dots, x_n)$ is given (not graph), we can construct graph using pairwise similarities, e.g.,

the Gaussian similarity function $s(x_i, x_j) = \exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$.

If graph $G(E, V)$ is given and is attributed (X), like social networks, then X is not used in previous algorithms.

Several algorithms are proposed to utilize both the graph structure W and the feature matrix X.

Covariate-assisted spectral clustering (CASC)

The key idea in CASC is to use the augmented Laplacian defined as: $\widehat{\mathrm{L}} = LL + \alpha XX^T$

The influence of the attributes on the communities is adjusted by the tuning parameter α.

Spectral clustering-based community detection using graph distance and node attributes (DCFI)

The key idea in DCFI is to **d**etect **c**ommunities by **f**using the two sources of **i**nformation.

Define distance matrix on X as:
$W_X = \{w_X(i,j), 0 \leq i,j \leq n\}$, where $w_X(i,j) = w_X(j,i)$ is the distance between x_i and x_j.

Rewrite adjacency matrix as: W_G.

DCFI (cont.)

Using W_G and W_X,

- define graph distance as:

$$S_G = \exp\{-\frac{W_G \odot W_G}{\sigma_1^2}\}$$

- define similarity matrix as:

$$S_X = \exp\{-\frac{W_X \odot W_X}{\sigma_2^2}\}$$

where $\odot$ is element-wise product operator,
σ_1 controls the rate of decay,
σ_2 is a parameter in the Gaussian similarity function determining the width of the neighborhoods.

DCFI (cont.)

Group nodes into K clusters with objective function: normalized cut,

- on graph structure: $$Ncut(G) = \sum_{k=1}^{K} \frac{\sum_{i \in C_k} \sum_{j \notin C_k} s_G(i, j)}{\sum_{i \in C_k} \sum_{j=1}^{n} s_G(i, j)}$$

- on attributes: $$Ncut(X) = \sum_{k=1}^{K} \frac{\sum_{i \in C_k} \sum_{j \notin C_k} s_X(i, j)}{\sum_{i \in C_k} \sum_{j=1}^{n} s_X(i, j)}$$

DCFI algorithm

1: Compute S_G by Eq. (2) and compute its Laplacian matrix L_G;
2: Select orthonormal eigenvectors corresponding to the K largest eigenvalues of L_G in absolute value and write $U_G = [U_G(1), \ldots, U_G(K)]$;
3: Normalize each row of $U_G = \{u_G(i,j); 1 \le i \le n, 1 \le j \le K\}$ to have unit norm and write $U_G^* = \{u_G^*(i,j)\}$ with $u_G^*(i,j) = u_G(i,j) / \sum_{j=1}^{K} u_G(i,j)$; ← U_G^*

initialize $\vec{\alpha}^{(0)}$;

for $t = 1, 2 \ldots$;

4: Compute the similarity matrix $S_X^{(t)}$ by Eqs. (3), (4), and compute $L_X^{(t)}$;
5: Select orthonormal eigenvectors $U_X^{(t)}(1), \ldots, U_X^{(t)}(K)$ corresponding to the K largest eigenvalues of $L_X^{(t)}$ and write $U_X^{(t)} = [U_X^{(t)}(1), \ldots, U_X^{(t)}(K)]$; ← U_X^*
6: Normalize each row of $U_X^{(t)} = \{u_X^{(t)}(i,j); 1 \le i \le n, 1 \le j \le K\}$ to have unit norm and write $U_X^{(t)*} = \{u_X^{(t)*}(i,j)\}$ with $u_X^{(t)*}(i,j) = u_X^{(t)}(i,j) / \sum_{j=1}^{K} u_X^{(t)}(i,j)$;
7: Perform K-means on $[U_G^*, U_X^{(t)*}] \in \mathbb{R}^{n \times 2K}$ to obtain $Z^{(t)}$; ← Concatenate U_G^* and U_X^*
8: Optimize $\vec{\alpha}^{(t)}$ by Eq. (7), given $Z^{(t)}$ in step 7;
9: Repeat steps 4–8 until the objective functions equations (5) and (6) converge.

DCFI (cont.)

Feature selection: integrate a general attribute weighting mechanism by rewriting the similarity measure between x_i and x_j:

$$s_X(i,j) = \exp\left\{-\frac{\sum_{l=1}^{m} \alpha_l (w_X^{(l)}(i,j))^2}{\sigma_2^2}\right\}$$

α_l: represents the weight of the l^{th} feature,

$w_X^{(l)}(i,j)$: denotes the distance between x_i and x_j along the l^{th} feature.

DCFI Experimental results

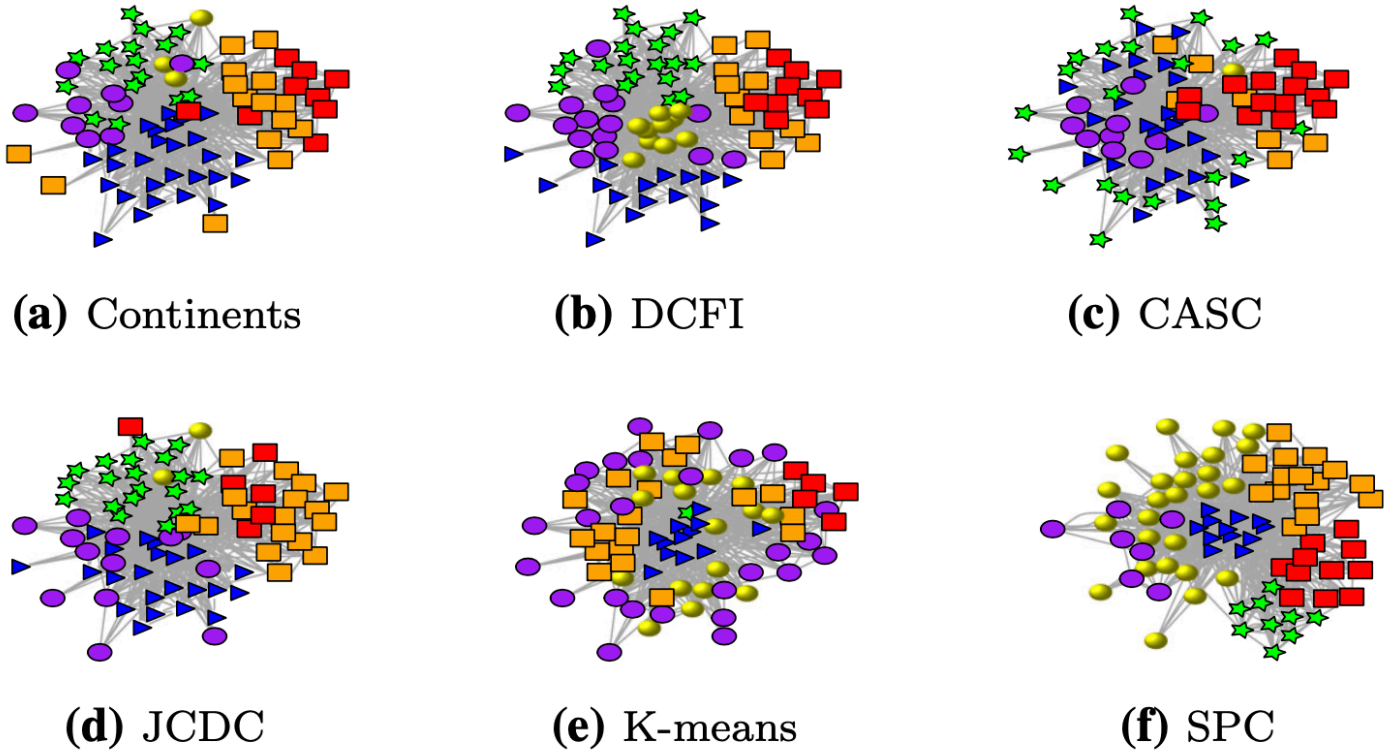


Fig. 9 **a** The world trade network with nodes colored by continent; red is N. America, green is Asia, blue is Europe, purple is Africa and yellow is S. America. **b–f** Estimated results by different methods; colors are mated to **a** in the best way possible (colour figure online)

Thank you!