Two provably consistent divide and conquer clustering algorithms for large networks

Soumendu Sundar Mukherjee

Based on joint work with

Peter Bickel (UC Berkeley) and Purnamrita Sarkar (UT Austin)

IISA Conference 2017, Hyderabad

December 28, 2017

Community detection

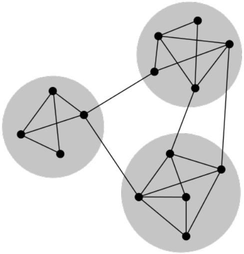


Image source: Wikipedia

Community detection

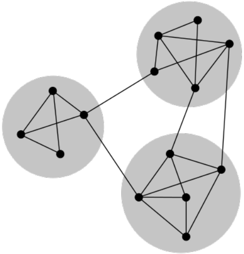


Image source: Wikipedia

Communities: Groups of nodes that are more densely connected within than across (or the other way round).

Community detection

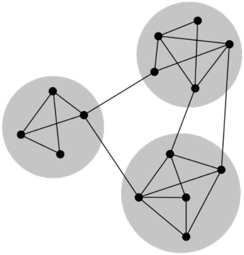


Image source: Wikipedia

Communities: Groups of nodes that are more densely connected within than across (or the other way round).

Examples:

- **political orientation** of blogs,
- **functionally similar** genes,
- facebook groups,
- groups according to **feeding mode** in **ecological networks** (who eats who) etc.

Community detection

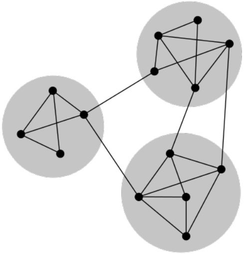


Image source: Wikipedia

- **Stochastic Block Model (SBM)**
  K : number of communities/blocks
  σ : latent block membership vector
  B : block connection probabilities
- $\mathbb{P}_{\sigma(i)=a,\sigma(j)=b}(A_{ij} = 1) = B_{ab}$.

Communities: Groups of nodes that are more densely connected within than across (or the other way round).

Examples:

- **political orientation** of blogs,
- **functionally similar** genes,
- facebook groups,
- groups according to **feeding mode** in **ecological networks** (who eats who) etc.

Community detection techniqies

- **Likelihood based methods**, **modularity optimization**, **semi-definite programming (SDP)**, **spectral clustering** etc.

Community detection techniqies

- **Likelihood based methods**, **modularity optimization**, **semi-definite programming (SDP)**, **spectral clustering** etc.
- Most of these can only be used on **small to moderate sized** graphs. For instance, SDPs are exactly solvable for only up to hundreds of nodes.

Community detection techniqies

- **Likelihood based methods**, **modularity optimization**, **semi-definite programming (SDP)**, **spectral clustering** etc.
- Most of these can only be used on **small to moderate sized** graphs. For instance, SDPs are exactly solvable for only up to hundreds of nodes.
- **Idea of divide and conquer:** take small subgraphs, apply clustering on them and finally put together the community structures.

Community detection techniqies

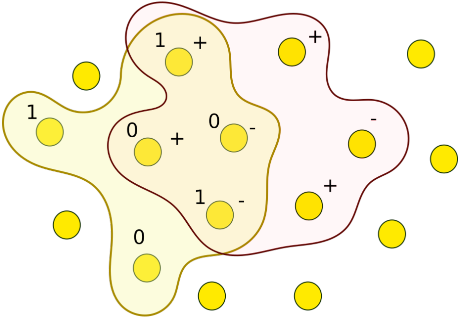
- **Likelihood based methods**, **modularity optimization**, **semi-definite programming (SDP)**, **spectral clustering** etc.
- Most of these can only be used on **small to moderate sized** graphs. For instance, SDPs are exactly solvable for only up to hundreds of nodes.
- **Idea of divide and conquer:** take small subgraphs, apply clustering on them and finally put together the community structures.
- Added advantage: **parallelizable**.

Divide and Conquer techniques for community detection

- Main issue: community structure is **"labeling" invariant** — how to properly align the (potentially conflicting) estimated community structures that we get from different subgraphs?

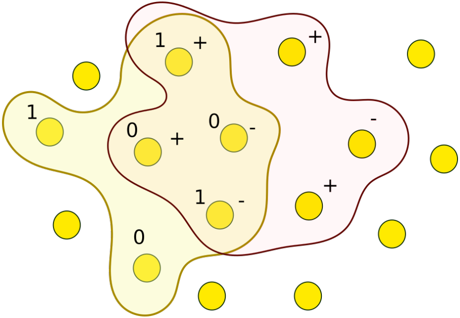
Divide and Conquer techniques for community detection

- Main issue: community structure is **"labeling" invariant** — how to properly align the (potentially conflicting) estimated community structures that we get from different subgraphs?



Divide and Conquer techniques for community detection

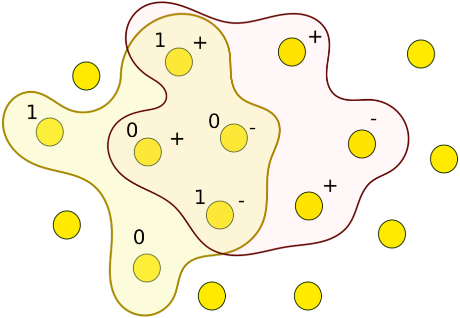
- Main issue: community structure is **"labeling" invariant** — how to properly align the (potentially conflicting) estimated community structures that we get from different subgraphs?



- We propose two such techniques: PACE and GALE.

Divide and Conquer techniques for community detection

- Main issue: community structure is **"labeling" invariant** — how to properly align the (potentially conflicting) estimated community structures that we get from different subgraphs?



- We propose two such techniques: PACE and GALE.
- Will talk about PACE only.

PACE: Piecewise Averaged Community Estimation

Let $\sigma : [n] \to [K]$ be the unknown community assignment. An equivalent description is given by the $n \times K$ matrix Z with

$$Z_{ij} = \delta_{i,\sigma(i)}.$$

PACE: Piecewise Averaged Community Estimation

Let $\sigma : [n] \to [K]$ be the unknown community assignment. An equivalent description is given by the $n \times K$ matrix Z with

$$Z_{ij} = \delta_{i,\sigma(i)}.$$

Our main idea is to concentrate on estimating **whether two individuals belong to the same community or not**, instead of estimating Z directly, i.e. we focus on

$$C = ZZ^\top,$$

the so-called clustering matrix. However, this is equivalent to estimating the community structure.

PACE: Piecewise Averaged Community Estimation

PACE has three main steps:

PACE: Piecewise Averaged Community Estimation

PACE has three main steps:

1. subgraph selection

PACE: Piecewise Averaged Community Estimation

PACE has three main steps:

1. subgraph selection
2. clustering on subgraphs

PACE: Piecewise Averaged Community Estimation

PACE has three main steps:

1. subgraph selection
2. clustering on subgraphs
3. patching up

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.
2. T: $\#$ of subgraphs to be sampled.

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.
2. T: $\#$ of subgraphs to be sampled.
3. Sampling scheme:

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.
2. T: # of subgraphs to be sampled.
3. Sampling scheme:
    1. **Random sampling:** select $m \geq m_\star$ nodes at random.

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.
2. T: $\#$ of subgraphs to be sampled.
3. Sampling scheme:
    1. **Random sampling:** select $m \geq m_\star$ nodes at random.
    2. **Network driven sampling:** Pick the h-hop neighborhood $N_h(r)$ of a random vertex r in G. Accept it if $|N_h(r)| \geq m_\star$.

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.
2. T: $\#$ of subgraphs to be sampled.
3. Sampling scheme:
   1. **Random sampling:** select $m \geq m_\star$ nodes at random.
   2. **Network driven sampling:** Pick the h-hop neighborhood $N_h(r)$ of a random vertex r in G. Accept it if $|N_h(r)| \geq m_\star$.

      Variants: ego/onion neighborhoods, choose root vertex with probability proportional to degree, etc.

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.
2. T: $\#$ of subgraphs to be sampled.
3. Sampling scheme:
    1. **Random sampling:** select $m \geq m_\star$ nodes at random.
    2. **Network driven sampling:** Pick the h-hop neighborhood $N_h(r)$ of a random vertex r in G. Accept it if $|N_h(r)| \geq m_\star$.

       Variants: ego/onion neighborhoods, choose root vertex with probability proportional to degree, etc.

Given A, choose T subsets $S_1, \ldots, S_T$ of the nodes by some sampling procedure.

PACE: subgraph selection

1. $m_\star$: minimum required subgraph size.
2. T: $\#$ of subgraphs to be sampled.
3. Sampling scheme:
    1. **Random sampling:** select $m \geq m_\star$ nodes at random.
    2. **Network driven sampling:** Pick the h-hop neighborhood $N_h(r)$ of a random vertex r in G. Accept it if $|N_h(r)| \geq m_\star$.

       Variants: ego/onion neighborhoods, choose root vertex with probability proportional to degree, etc.

Given A, choose T subsets $S_1, \ldots, S_T$ of the nodes by some sampling procedure.

Let A_{S_ℓ} : the adjacency matrix of the network induced by S_ℓ.

PACE: clustering on subgraphs

1. Perform clustering on A_{S_ℓ}, $l = 1, \ldots, T$ to get $\hat{Z}_{S_\ell}$.

PACE: clustering on subgraphs

1. Perform clustering on A_{S_ℓ}, $l = 1, \ldots, T$ to get $\hat{Z}_{S_\ell}$.
2. Set $\hat{C}_{S_\ell} = \hat{Z}_{S_\ell}\hat{Z}_{S_\ell}^\top$.

PACE: clustering on subgraphs

1. Perform clustering on A_{S_ℓ}, $l = 1, \ldots, T$ to get $\hat{Z}_{S_\ell}$.
2. Set $\hat{C}_{S_\ell} = \hat{Z}_{S_\ell} \hat{Z}_{S_\ell}^\top$.
3. Extend $\hat{C}_{S_\ell}$ to an $n \times n$ matrix $\hat{C}^{(\ell)}$:

$$\hat{C}^{(\ell)}_{ij} = \begin{cases} \hat{C}_{S_\ell, i, j} & \text{if } i, j \in S_\ell \\ 0 & \text{otherwise.} \end{cases}$$

PACE: clustering on subgraphs

1. Perform clustering on A_{S_ℓ}, $l = 1, \ldots, T$ to get $\hat{Z}_{S_\ell}$.
2. Set $\hat{C}_{S_\ell} = \hat{Z}_{S_\ell}\hat{Z}_{S_\ell}^\top$.
3. Extend $\hat{C}_{S_\ell}$ to an $n \times n$ matrix $\hat{C}^{(\ell)}$:

$$\hat{C}_{ij}^{(\ell)} = \begin{cases} \hat{C}_{S_\ell,i,j} & \text{if } i,j \in S_\ell \\ 0 & \text{otherwise.} \end{cases}$$

Can use any standard algorithm in the first step, e.g. profile likelihood (PL), mean field method (MF), spectral clustering (SC), semi-definite programming (SDP) etc.

PACE: patching up

1. Let $y_{ij}^{(\ell)} = \mathbf{1}_{\{i,j \in S_\ell\}}$. Set $N_{ij} = \sum_{l=1}^{T} y_{ij}^{(\ell)}$.

PACE: patching up

1. Let $y_{ij}^{(\ell)} = \mathbf{1}_{\{i,j \in S_\ell\}}$. Set $N_{ij} = \sum_{l=1}^{T} y_{ij}^{(\ell)}$.
2. Define the combined estimator $\hat{C} = \hat{C}_\tau$ by

$$\hat{C}_{ij} = \hat{C}_{\tau,ij} = \frac{\mathbf{1}_{\{N_{ij} \geq \tau\}} \sum_{\ell=1}^{T} y_{ij}^{(\ell)} \hat{C}_{ij}^{(\ell)}}{N_{ij}} = \frac{\mathbf{1}_{\{N_{ij} \geq \tau\}} \sum_{\ell=1}^{T} \hat{C}_{ij}^{(\ell)}}{N_{ij}}.$$

PACE: patching up

1. Let $y_{ij}^{(\ell)} = \mathbf{1}_{\{i,j \in S_\ell\}}$. Set $N_{ij} = \sum_{l=1}^{T} y_{ij}^{(\ell)}$.
2. Define the combined estimator $\hat{C} = \hat{C}_\tau$ by

$$\hat{C}_{ij} = \hat{C}_{\tau,ij} = \frac{\mathbf{1}_{\{N_{ij} \geq \tau\}} \sum_{\ell=1}^{T} y_{ij}^{(\ell)} \hat{C}_{ij}^{(\ell)}}{N_{ij}} = \frac{\mathbf{1}_{\{N_{ij} \geq \tau\}} \sum_{\ell=1}^{T} \hat{C}_{ij}^{(\ell)}}{N_{ij}}.$$

Here $1 \leq \tau \leq T$ is an integer tuning parameter. We will call $\hat{C}_\tau$ as Piecewise Averaged Community Estimator (PACE).

Simulations

Algorithm	ME(%)	Time (seconds)
SDP	0	1588
SDP + PACE + spectral	0	**288**
SDP + PACE + RPKmeans	9.1	**281**

Table: PACE with SDP. $n = 5000$, $d_n = 128$, $m = 500$, 4 equal sized clusters, $T = 100$, 20 parallel workers in Matlab.

Simulations

Algorithm	ME(%)	Time (seconds)
SDP	0	1588
SDP + PACE + spectral	0	**288**
SDP + PACE + RPKmeans	9.1	**281**

Table: PACE with SDP. $n = 5000$, $d_n = 128$, $m = 500$, 4 equal sized clusters, $T = 100$, 20 parallel workers in Matlab.

Algorithm	ME(%)	Time (seconds)
RSC	39.6	87
RSC + PACE + spectral	**3.4**	**21**
RSC + PACE + RPKmeans	**34.2**	**14**

Table: PACE with RSC. $n = 5000$, $d_n = 7$, cluster proportions $\boldsymbol{\pi} = (0.2, 0.8)$, $T = 100$, 20 parallel workers in Matlab, 3-hop neighbourhoods were used as subgraphs

Consistency

Measure of distance between two clustering matrices $C = ZZ^T$ and $C' = Z'Z'^T$:

$$\tilde{\delta}(C, C') := \frac{1}{n^2}\|C - C'\|_F^2.$$

Consistency

Measure of distance between two clustering matrices $C = ZZ^T$ and $C' = Z'Z'^T$:

$$\tilde{\delta}(C, C') := \frac{1}{n^2}\|C - C'\|_F^2.$$

Theorem (informal)

Suppose $\tilde{\delta}_S$ denotes the misclustering error on a randomly selected subgraph S. Then

$$\mathbb{E}\tilde{\delta}_{\text{PACE}} \preceq \mathbb{E}\tilde{\delta}_S + \textit{ coverage error.}$$

Coverage error is $O(\exp(-O(Tm^2/n^2)))$ for random m-subgraphs.

A concrete application

Adjacency spectral clustering (ASP) and **semi-definite programming (SDP)** on a simple two parameter SBM

$$B = \alpha_n[(1-\lambda)I + \lambda 1 1^\top].$$

A concrete application

Adjacency spectral clustering (ASP) and **semi-definite programming (SDP)** on a simple two parameter SBM

$$B = \alpha_n[(1-\lambda)I + \lambda 11^\top].$$

For simplicity, let λ and K be fixed constants. Let $d_n =$ expected average degree $\asymp n\alpha_n$.

A concrete application

Adjacency spectral clustering (ASP) and **semi-definite programming (SDP)** on a simple two parameter SBM

$$B = \alpha_n[(1-\lambda)I + \lambda 11^\top].$$

For simplicity, let λ and K be fixed constants. Let $d_n =$ expected average degree $\asymp n\alpha_n$.

Using results of Lei and Rinaldo (2015), Guédon and Vershynin (2015) can show

1. one needs subgraphs of size $m \gg n d_n^{-1}$, and $T \gg \frac{n^2}{m^2}$ for consistency.
2. Complexity becomes $O_\star(\frac{n^3}{d_n})$, from $O(n^3)$.
3. Also a significant boost comes from parallelizability.

Thank You!

Recovering $\hat{Z}$ from $\hat{C}$

- Note that

$$\|C_{i\star} - C_{j\star}\| = \begin{cases} 0 & \text{if } \sigma(i) = \sigma(j) \\ \sqrt{|\sigma(i)| + |\sigma(j)|} & \text{otherwise.} \end{cases}$$

Recovering $\hat{Z}$ from $\hat{C}$

- Note that

$$\|C_{i\star} - C_{j\star}\| = \begin{cases} 0 & \text{if } \sigma(i) = \sigma(j) \\ \sqrt{|\sigma(i)| + |\sigma(j)|} & \text{otherwise.} \end{cases}$$

- The latter quantity is $\Omega(\sqrt{n/K})$, if the communities are balanced.

Recovering $\hat{Z}$ from $\hat{C}$

- Note that

$$\|C_{i\star} - C_{j\star}\| = \begin{cases} 0 & \text{if } \sigma(i) = \sigma(j) \\ \sqrt{|\sigma(i)| + |\sigma(j)|} & \text{otherwise.} \end{cases}$$

- The latter quantity is $\Omega(\sqrt{n/K})$, if the communities are balanced.
- Thus we expect that any distance based clustering algorithm working on rows of $\hat{C}$ will do well.

Recovering $\hat{Z}$ from $\hat{C}$

- Note that

$$\|C_{i\star} - C_{j\star}\| = \begin{cases} 0 & \text{if } \sigma(i) = \sigma(j) \\ \sqrt{|\sigma(i)| + |\sigma(j)|} & \text{otherwise.} \end{cases}$$

- The latter quantity is $\Omega(\sqrt{n/K})$, if the communities are balanced.
- Thus we expect that any distance based clustering algorithm working on rows of $\hat{C}$ will do well.
- Computing distance for n-dimensional vectors is expensive, so we do a random projection of the rows to $\Omega(\log n)$ dimensions (**Johnson-Lindenstrauss**) first, and then do clustering of those projected rows.