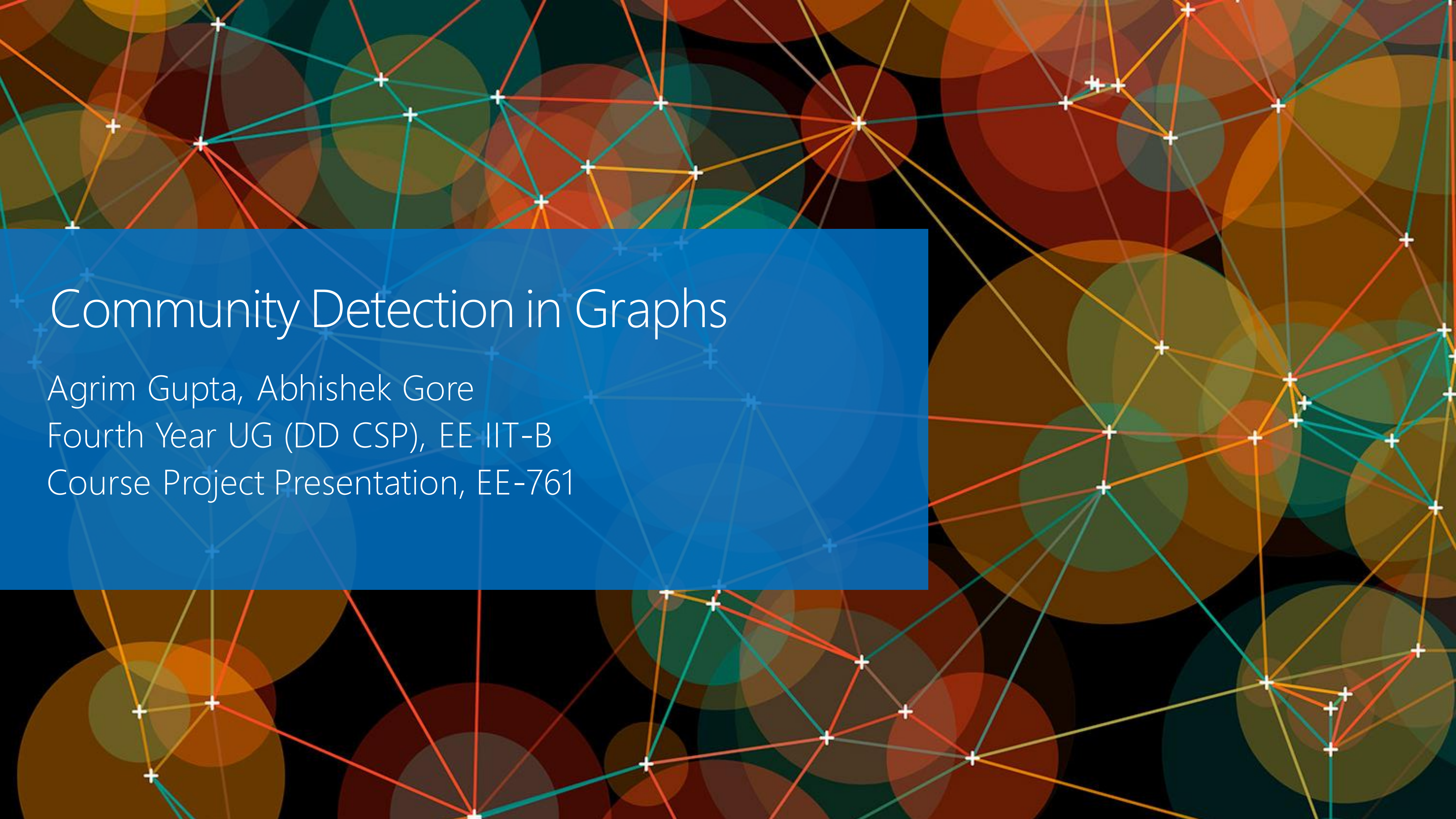


Community Detection in Graphs

Agrim Gupta, Abhishek Gore
Fourth Year UG (DD CSP), EE IIT-B

Course Project Presentation, EE-761

Overview of Presentation

- Motivation and Applications
- Setting of the algorithm: Stochastic Block Model and Side Information
- Intuition behind the algorithm
- Proof Outlines and Theorems used
- Avenues for research and Key Takeaways

Detection of Communities in Graphs

Motivation

Community detection, is the classic problem of finding subsets of nodes such that each subset has higher connectivity within itself, as compared to the average connectivity of the graph as a whole. These algorithms are very innately connected to problems in Social Network analysis, Sensor networks and Information Theory.

Graph Clustering has evolved as a capable tool for solving community detection problems. Its ability to scale out efficiently over multiple machines means that graph algorithms run seamlessly over clustered computing frameworks like Spark and Hadoop. Also, due to advent of NoSQL databases, like graph databases, it is bound to get more usage in near future.

Concentration Bounds are crucial in this field as they allow us to design algorithms with guarantees on accuracy. A number of concepts taught in the course are applied here to get useful bounds.

Applications

- Social Network Theory
- Military Intelligence
- Computational Biology

Setting of the algorithm: Stochastic Block Model (SBM)

In this model, the n nodes of a graph are partitioned into k groups, two nodes in the same group are connected by an edge with some probability p, and two nodes in different groups are connected by an edge with some probability q (<p)

- ### Why choosing SBM ?

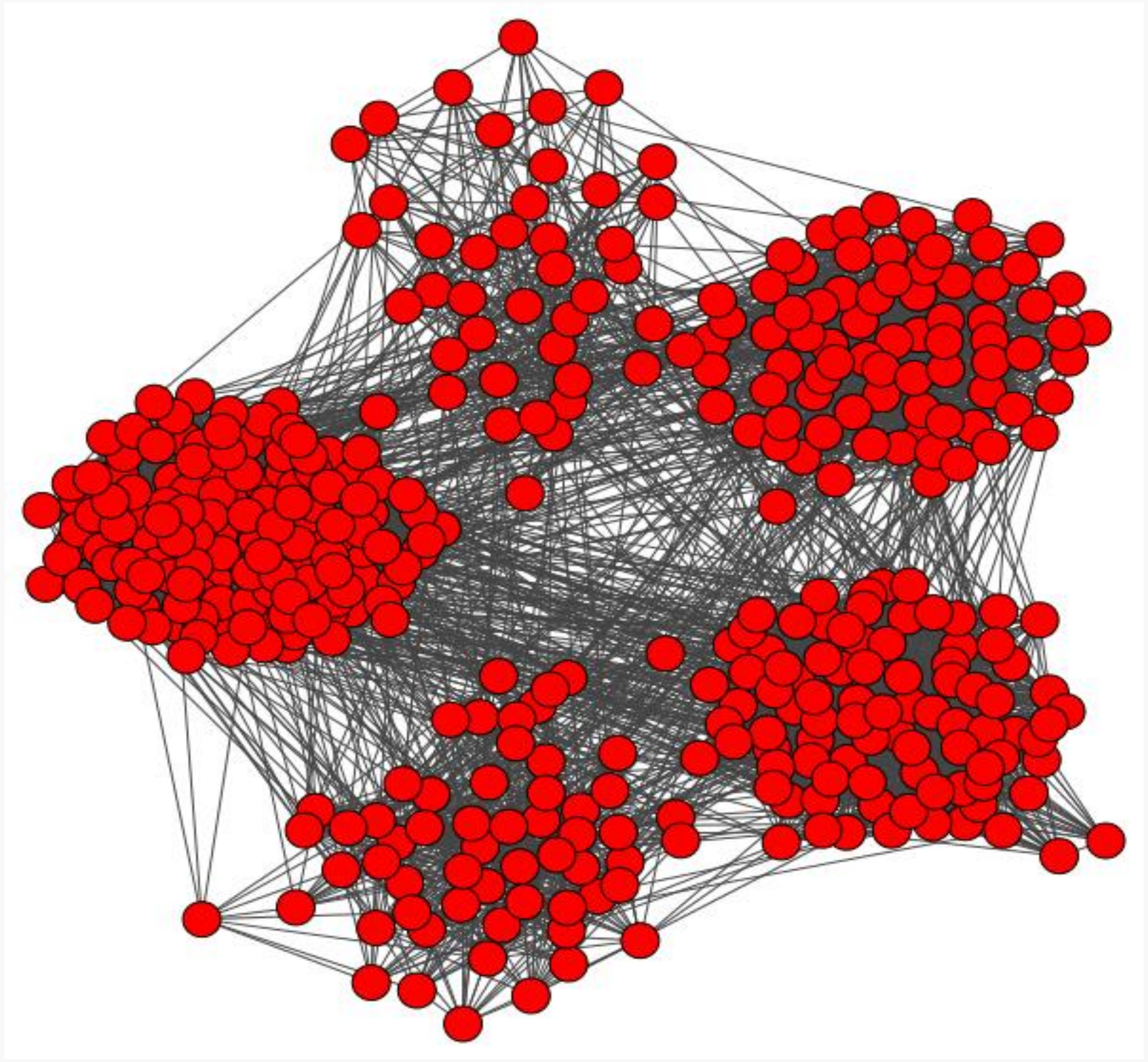
Models a 'community' well per se, and is easy to generate, generalize and work around with

- ### p-q separation condition

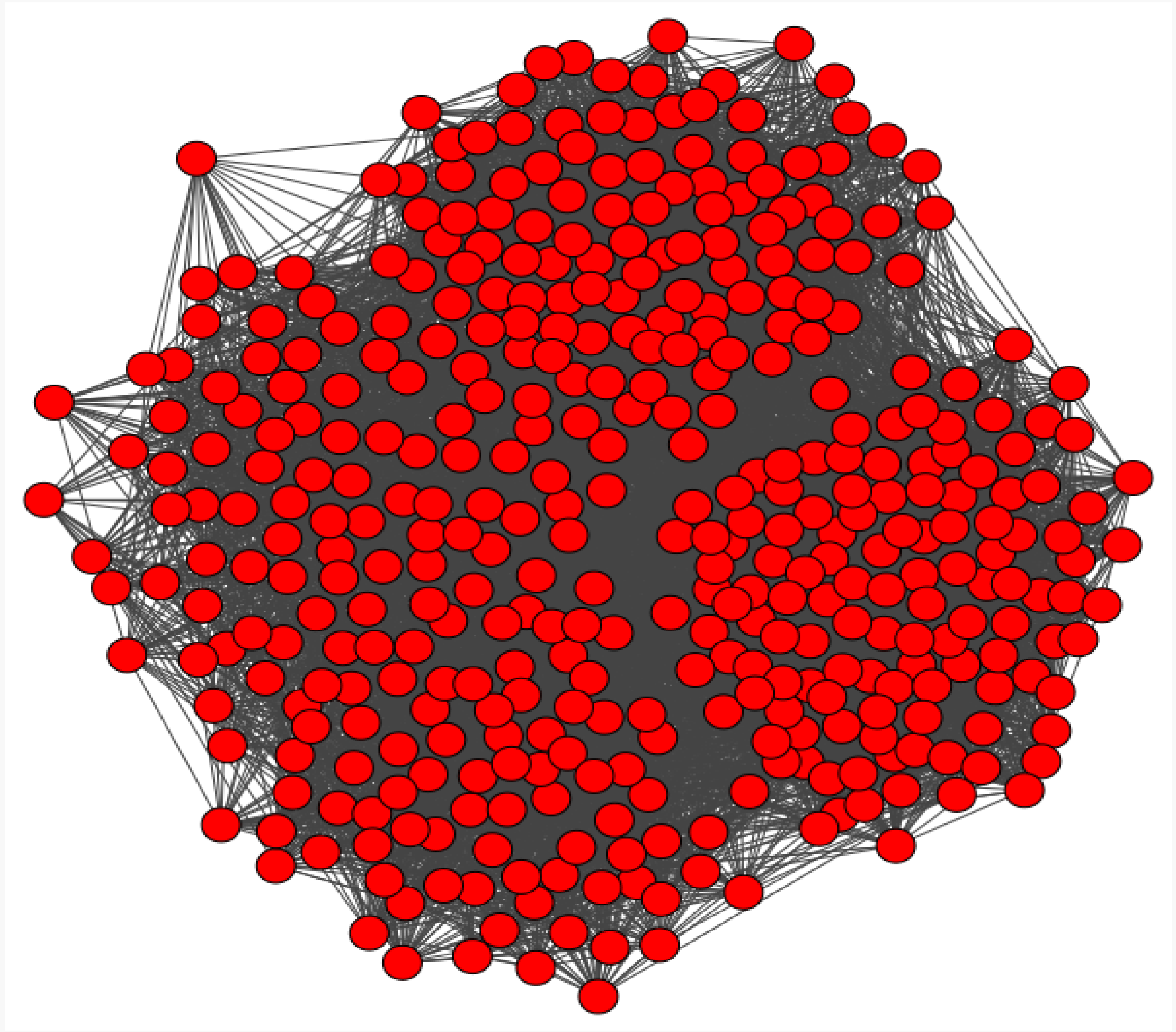
The model poses a very crucial condition for the algorithm to work with. Intuitively, if p and q are close, the communities are 'homogeneous' and it will be tough for the algorithm to separate them.

p-q separation, visually

SBM(p=0.3,q=0.01,n=400)



SBM(p=0.3,q=0.05,n=400)



Side Information and Target Community

Target Community

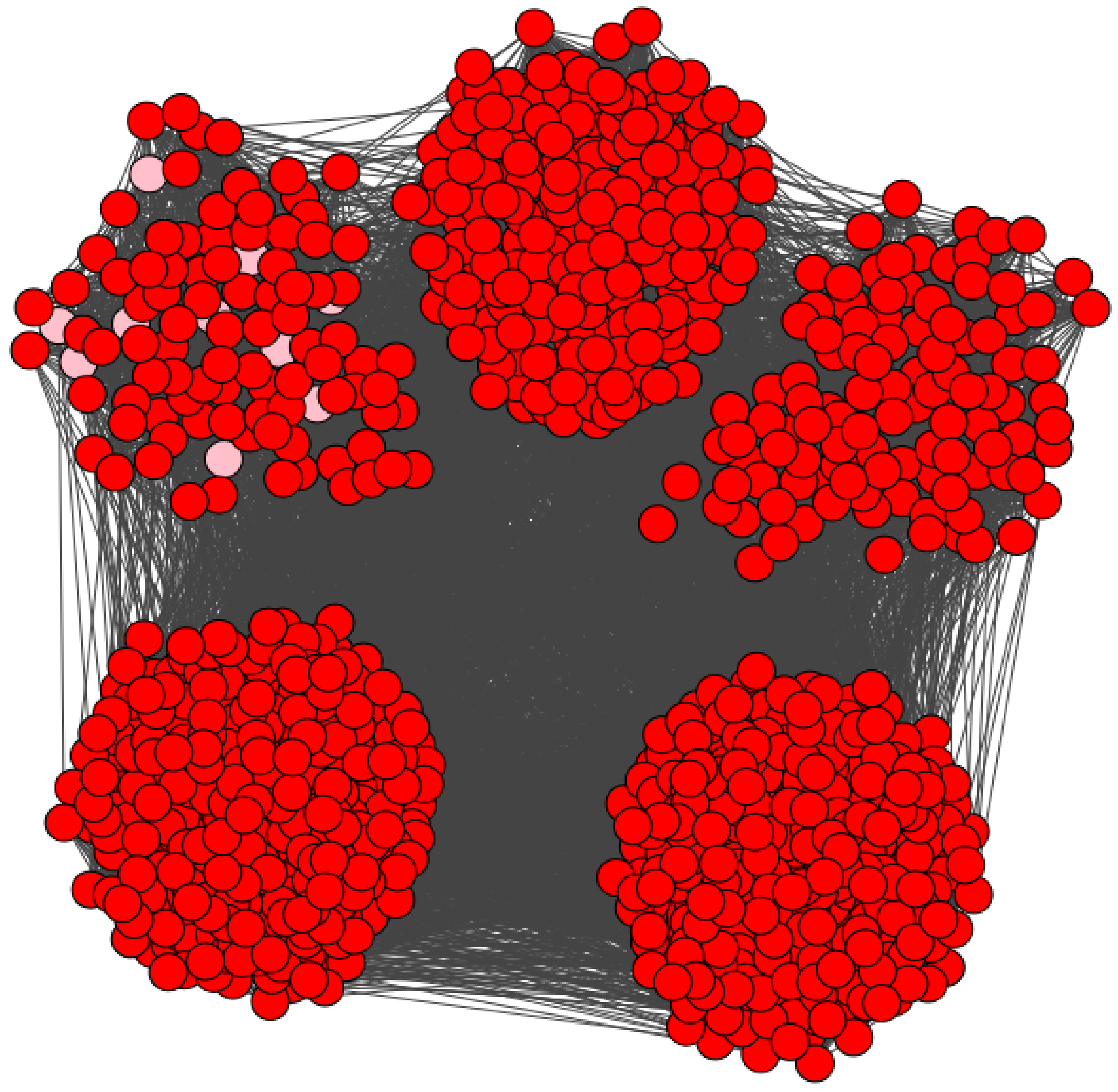
In the paper we try and recover one community from the ensemble of communities in the graph. The paper intends to detect one community faster than time it takes by other pre-existing algorithms to detect all the communities

Side Information

To help our cause, we have some side information on what the target community should compose of. The side-information is in the form of biased node weights.

Labelled Nodes and Biased Weight Calculations

We assume that we know some labelled nodes of the target community (pink). The biased node weights are calculated as the number of labelled nodes reachable from the node via at most r+1 hops



Summarising the Setting, Mathematically

1. p-q separation constraint

Intuitively, we can see the the quantity (p-q) will get lower bounded.

$$\frac{(p-q)^2}{p\sqrt{p}} \geq \frac{\alpha_{max}}{\alpha_{min}{}^2\sqrt{n}}$$

where, α_i denotes the fraction of nodes in community i, i.e. $\alpha_i = \frac{n_i}{n}$

2. Side Information constraint

Let the set of communities be V_i. WLOG, choose the target community as V_1. To honor the side information constraint, the node weights W_i must follow the below condition:

$$E[w_j | j \in V_1] > E[w_j | j \in V_i], \ \forall i \neq 1$$

3. Biased weight calculations

Let m be the number of labeled nodes

$$w_j = \sum_{i=1}^{m} \mathbb{1}(\text{i reachable within r+1 hops}) \ \forall j \in [1, 2, 3..., n]$$

The Algorithm

Algorithm 1 Community Search

```
Input: Adjacency matrix $X$, $k$, biased weights $(w_1, \dots, w_n)$, threshold $\tau$
Output: $\hat{V}_1$
1: Partition nodes into four sets $P_1, P_2, P_3, P_4$ at random
2: Compute matrices $\hat{A}_1 = \frac{1}{\sqrt{|P_3|}} X_{P_1,P_3}$, $\hat{A}_2 = \frac{1}{\sqrt{|P_3|}} X_{P_2,P_3}$
3: Compute vector $\hat{m}_1 = \frac{1}{|P_1|} \sum_{j \in P_1} X_{P_1,j}$
4: Compute matrix $\hat{B} = \frac{1}{|P_4|} \sum_{j \in P_4} w_j X_{P_1,j} X^T_{P_2,j}$
5: $\hat{\mu}_{P_1}, \hat{\alpha}_1 \leftarrow$ SearchSubroutine$(\hat{A}_1, \hat{A}_2, \hat{B}, \hat{m}_1, k)$
6: Compute $V_{P_1} = \{j \in P_1 : \hat{\mu}_{P_1,j} > \tau\}$
7: Repeat steps 2-5 with $P_i$'s rotated in order to estimate $\hat{\mu}_{P_2}, \hat{\mu}_{P_3}, \hat{\mu}_{P_4}$. Use them to compute $V_{P_2}, V_{P_3}, V_{P_4}$ as
   in step 6
8: Return community $\hat{V}_1 = V_{P_1} \cup V_{P_2} \cup V_{P_3} \cup V_{P_4}$
```

Algorithm 2 SearchSubroutine

```
Input: $\hat{A}_1, \hat{A}_2, \hat{B}, \hat{m}_1, k$
Output: $\hat{\mu}_1, \hat{\alpha}_1$
1: Compute rank $k$-svd of matrices $\hat{A}_1, \hat{A}_2 : \hat{A}_1 = U_1 D_1 V_1^T$, $\hat{A}_2 = U_2 D_2 V_2^T$
2: Compute matrices $W_1 = U_1 D_1^{-1}$, $W_2 = U_2 D_2^{-1}$
3: Let $u_1$ be the largest left singular vector of $W_1^T \hat{B} W_2$
4: Compute $z = U_1 D_1 u_1$
5: Compute $a = u_1^T W_1^T \hat{m}_1$
6: Return $\hat{\mu}_1 \leftarrow z/a$ and $\hat{\alpha}_1 \leftarrow a^2$
```

The Intuition behind algorithm: New Notations

- Since the graph is random, the adjacency matrix of graph behaves randomly as well
- Suppose, we knew the expectation of the adjacency matrix a priori. The expected value of a column j, $\mathbb{E}[X_j]$ would be then 'p' for the points belonging to the community j belongs to, and 'q' for all other points.

> i.e $\mathbb{E}[X_j]_i = p\mathbb{1}($i in same community as j$)+q\mathbb{1}($i in diff. community$)$

- Define *community membership vectors*, $\mu_1, \mu_2, \ldots . \mu_k$, with $\mu_i \in \mathbb{R}^n$ in the same spirit as above, with:

> $\mathbb{E}[X_j] = \mu_{c_j}$ where μ_{c_j}, where c_j is the community j belongs to

- Now, the whole problem boils down to estimating μ_1 from the graph. Let $\bar{w_j} := \mathbb{E}[w_j]$. Define A and B matrices as below.

$$A := \frac{1}{n}\sum_j E[X_j]E[X_j]^T = \sum_{i=1}^k \alpha_i \mu_i \mu_i^T \qquad B := \frac{1}{n}\sum_j \bar{w}_j E[X_j]E[X_j]^T = \sum_{i=1}^k \alpha_i \omega_i \mu_i \mu_i^T$$

The Intuition behind algorithm: Whitening

- The main idea now is that we can still recover μ_1 by "whitening" B using A (All this given that we know, or can estimate $\mathbb{E}[X]$ closely).
- Do rank k SVD of A, to get $A = UDU^T$, let $W := UD^{-\frac{1}{2}}$.

> Observe, $W^TAW = I_k = \sum_{i=1}^{k} \widetilde{\mu_i}\widetilde{\mu_i}^T$, where $\widetilde{\mu_i} := \sqrt{\alpha_i} W^T \mu_i$. Also, note that addition of k terms of type $\widetilde{\mu_i}\widetilde{\mu_i}^T$, results in I_k, which happens only if corresponding μ_i are *orthonormal* vectors in $\mathbb{R}^k$

- Hence we have obtained $\widetilde{\mu_i}$ which are whitened versions of μ_i. Now compute $R := W^TBW = \sum_{i=1}^{k} w_i \widetilde{\mu_i}\widetilde{\mu_i}^T$

> Now, since $\widetilde{\mu_i}$ are orthonormal, the above equation represents an eigenvalue decomposition of the k × k size matrix R, with eigenvectors $\widetilde{\mu_i}$ and corresponding eigenvalues w_i . Thus, $\widetilde{\mu_1}$ – the whitened vector corresponding to the target community – is now the leading eigenvector of R, because $w_1 > w_i, \forall i \neq 1$

- Find $\widetilde{\mu_1}$ by setting it to be the leading eigenvector of R. Finally recover μ_1 from $\widetilde{\mu_1}$ in the foll. two steps.
  - First compute $z := UD^{\frac{1}{2}} \widetilde{\mu_1} = \sqrt{\alpha_1}\mu_1$
  - Next compute $m_1 := \frac{1}{n} \sum_{j=1}^{n} \mathbb{E}[X_j] = \sum_{i=1}^{n} \alpha_i u_i$. Using this, recover $\sqrt{\alpha_1} = \widetilde{\mu_1}^T W^T m_1$
  - Divide the obtained z, by recovered $\sqrt{\alpha_1}$ to get μ_1 and hence the nodes in V_1

Transiting from Known to Estimated Matrices

- All the analyses done before assumed that we knew E[X] a priori, but this is not the case.
- However, if we can estimate the A and the B matrices defined by some means, we can override and still detect communities

Ok, but that being said, why were we partitioning X in the algo ?

- A natural choice of estimator of matrix A would be $\frac{1}{n}\sum_j X_j X_j^T$
- However, this is a good estimator of , $E[XX^T]$ and $E[XX^T] \neq E[X]E[X]^T$
- So, we partition X so as to gain independence at this point

Hence, apart from the two conditions discussed before, we will get another bound as a penalty of not knowing things a priori (plus a formal bound on p-q separability as well)

Estimated Matrices: Tools used

- Before Diving into probabilistic bounds, there are a set of Lemmas which bound the spectral norm difference of the matrices discussed in earlier sections.
  - Some theorems used were Weyl's inequality and Wedin's theorem
- Main tool used to bound the difference between estimated matrix and actual matrix is "Matrix Bernstein Inequality", which corresponds to the following:

Theorem (Matrix Bernstein). *Let* $\{A_j\}_{j=1}^n$ *be a sequence of i.i.d. real random* $d_1 \times d_2$ *matrices such that* $E[A_j] = 0$, $\|A_j\| \leq L$. *Define* $Z = \sum_{j=1}^n A_j$. *Let* $\sigma^2 = \max\{\|E[ZZ^T]\|, \|E[Z^T Z]\|\}$. *Then for all* $t \geq 0$,

$$P\left(\|Z\| \geq t\right) \leq (d_1 + d_2) \exp\left(\frac{-t^2/2}{\sigma^2 + Lt/3}\right)$$

  - Usage of Matrix Bernstein is motivated from the fact that from Weyl's inequality we know a bound on spectral norm of diff. between matrix and estimated matrix

Proof Motivation (1)

(A2) Weight concentration: Let $\alpha_{max} = \max_{i\in[k]} \alpha_i$ and $\alpha_{min} = \min_{i\in[k]} \alpha_i$. Define $\sigma_1(R) := E[w_j | j \in V_1]$, $\sigma_2(R) := \max_{i\neq 1} E[w_j | j \in V_i]$, $\gamma_2 := \max_{i\in[k], j\in V} |w_j - E[w_j | j \in V_i]|$, and $\xi(n) = o(\sqrt{\log n})$ be any slowly growing function. Then with high probability the maximum deviation of the weights are bounded as,

$$\frac{\gamma_2}{(\sigma_1(R) - \sigma_2(R))} = O\left(\min\left\{\frac{\alpha_{min}^4 (p-q)^4}{\alpha_{max}^4 p^4 \xi(n)}, \frac{\alpha_{min}^5 \sqrt{n}(p-q)^5}{\alpha_{max}^4 p^{4.5} \xi(n)} - 1\right\}\right)$$

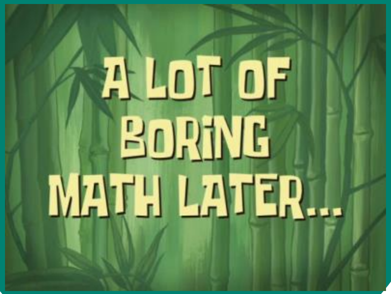
Get a concentration result on estimated A

Whitening matrix concentration

R matrix concentration

Choice of thresholding s.t.max o(αn) errors are made

$$\tau = \sqrt{\alpha_1}\frac{(p+q)}{2}$$



We get the above precondition on the algorithm!

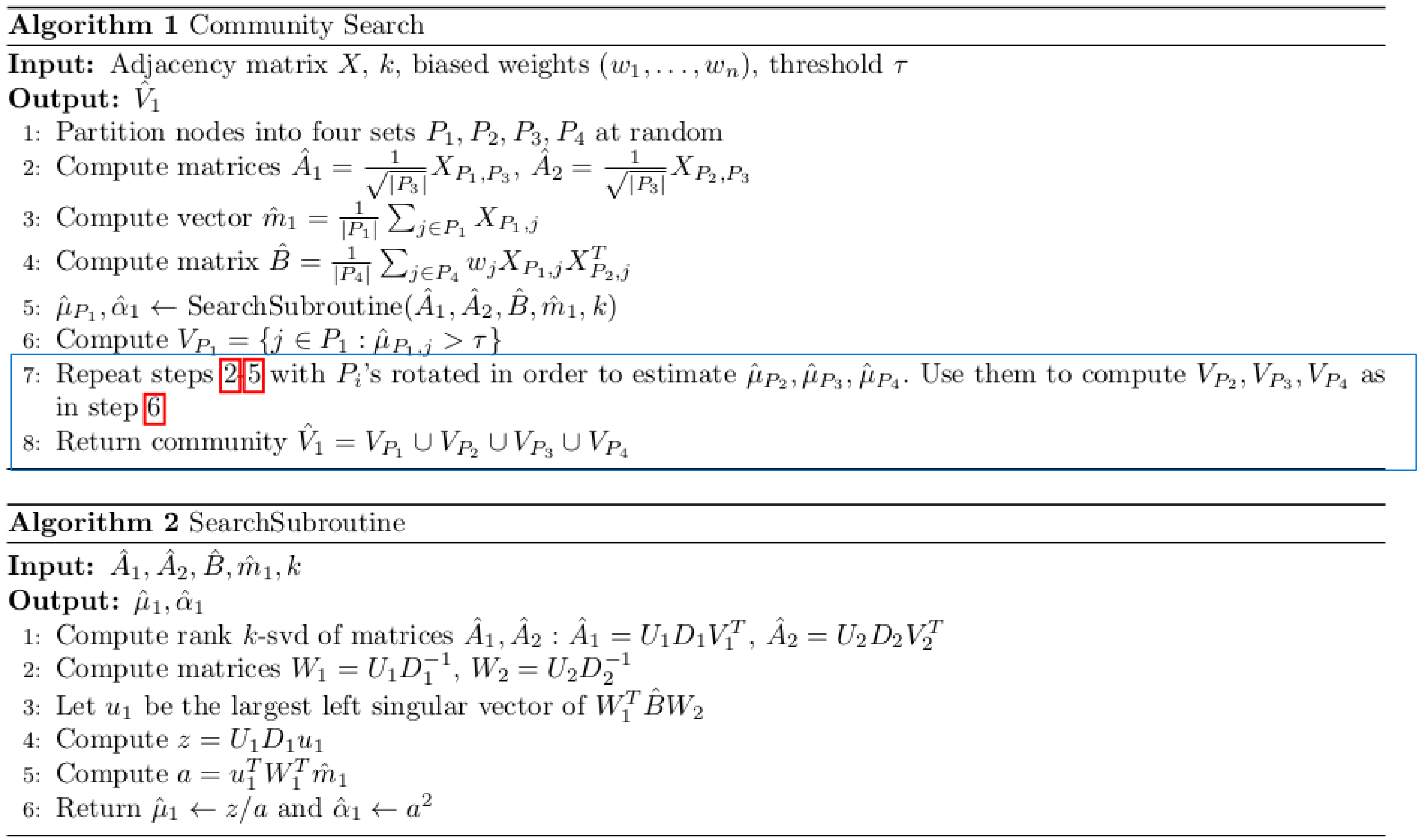
Intuitively, this precondition ensures that **Matrix B can be estimated up to a tolerable error**

We also get the formal bound required on (p-q) !

(A3) p, q separation: Let p, q, n satisfy

$$\frac{(p-q)^2}{p\sqrt{p}} = \tilde{\Omega}\left(\frac{\alpha_{max}}{\alpha_{min}^2 \sqrt{n}}\right)$$

Recall the algorithm once more



```
Algorithm 1 Community Search
Input: Adjacency matrix $X$, $k$, biased weights $(w_1, \dots, w_n)$, threshold $\tau$
Output: $\hat{V}_1$
 1: Partition nodes into four sets $P_1, P_2, P_3, P_4$ at random
 2: Compute matrices $\hat{A}_1 = \frac{1}{\sqrt{|P_3|}} X_{P_1,P_3}$, $\hat{A}_2 = \frac{1}{\sqrt{|P_3|}} X_{P_2,P_3}$
 3: Compute vector $\hat{m}_1 = \frac{1}{|P_1|} \sum_{j \in P_1} X_{P_1,j}$
 4: Compute matrix $\hat{B} = \frac{1}{|P_4|} \sum_{j \in P_4} w_j X_{P_1,j} X^T_{P_2,j}$
 5: $\hat{\mu}_{P_1}, \hat{\alpha}_1 \leftarrow$ SearchSubroutine$(\hat{A}_1, \hat{A}_2, \hat{B}, \hat{m}_1, k)$
 6: Compute $V_{P_1} = \{j \in P_1 : \hat{\mu}_{P_1,j} > \tau\}$
 7: Repeat steps $\boxed{2}$-$\boxed{5}$ with $P_i$'s rotated in order to estimate $\hat{\mu}_{P_2}, \hat{\mu}_{P_3}, \hat{\mu}_{P_4}$. Use them to compute $V_{P_2}, V_{P_3}, V_{P_4}$ as
    in step $\boxed{6}$
 8: Return community $\hat{V}_1 = V_{P_1} \cup V_{P_2} \cup V_{P_3} \cup V_{P_4}$
```

```
Algorithm 2 SearchSubroutine
Input: $\hat{A}_1, \hat{A}_2, \hat{B}, \hat{m}_1, k$
Output: $\hat{\mu}_1, \hat{\alpha}_1$
 1: Compute rank $k$-svd of matrices $\hat{A}_1, \hat{A}_2$ : $\hat{A}_1 = U_1 D_1 V_1^T$, $\hat{A}_2 = U_2 D_2 V_2^T$
 2: Compute matrices $W_1 = U_1 D_1^{-1}$, $W_2 = U_2 D_2^{-1}$
 3: Let $u_1$ be the largest left singular vector of $W_1^T \hat{B} W_2$
 4: Compute $z = U_1 D_1 u_1$
 5: Compute $a = u_1^T W_1^T \hat{m}_1$
 6: Return $\hat{\mu}_1 \leftarrow z/a$ and $\hat{\alpha}_1 \leftarrow a^2$
```

Final piece in the jigsaw: Union of Vp's

- Where are we placed now ?

Recall that we partitioned graph into 4 parts, $P_1, P_2 P_3, P_4$. Under conditions (1)-(3) discussed we can guarantee that the estimated community $\hat{V}_{P_1}$ has at most $o(\alpha_1 n)$ erroneous nodes. We now want to make a similar statement for $\hat{V}_{P_i}, i \in [2,3,4]$. Then, we can take unions of $\hat{V}_{P_i}$'s and recover all nodes belonging to V_1

- The analysis until thresholding step remains same, but the thresholding changes.

Define $\forall j \in P_2, d_j(\ddot{V}_{P_1})$ be the number of edges node j shares with nodes in $\ddot{V}_{P_1}$.

Let $v_1 = |\hat{V}_{P_1} \cap V_1|$ be the number of correctly detected nodes in $\hat{V}_{P_1}$, and $e_1 = |\hat{V}_{P_1} \cap V_1^c|$ be the number of erroneous nodes in $\hat{V}_{P_1}$.

From earlier discussions we have with high probability $v_1 = \Theta(\alpha_1 n)$, $e_1 = o(\alpha_1 n)$. Let $v_0 = |V_1 \cap P_1| = \Theta(\alpha_1 n)$. Note that $v_1 \geq v_0 - e_1$.

Now $\forall j \in V_1 \cap P_2$ using Chernoff bound we get with high probability:

$$d_j(\hat{V}_{P_1}) \geq v_1 p - \sqrt{v_1 plog(n)} + e_1 q - \sqrt{e_1 qlog(n)} \implies d_j(\hat{V}_{P_1}) \geq v_0 \frac{p+q}{2}$$

Similarly $\forall j \in V_1^c \cap P_2$ we get:

$$d_j(\hat{V}_{P_1}) \leq v_1 q - \sqrt{v_1 qlog(n)} + e_1 p - \sqrt{e_1 plog(n)} \implies d_j(\hat{V}_{P_1}) < v_0 \frac{p+q}{2}$$

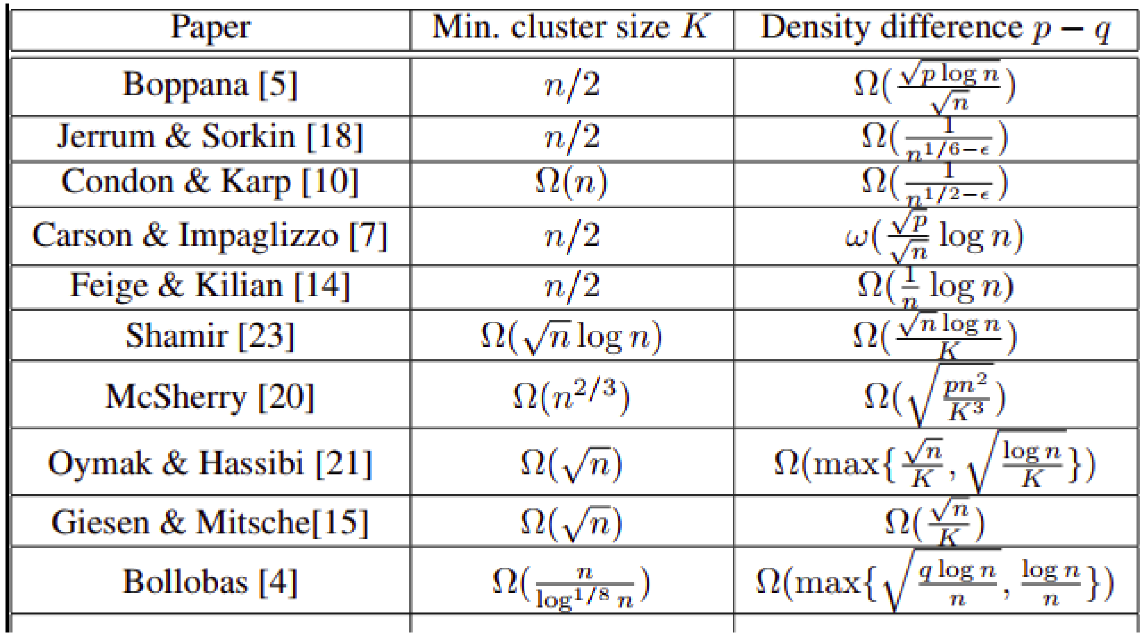
Therefore using a threshold $v_0 \frac{(p+q)}{2}$ we can correctly recover all nodes in $V_1 \cap P_2$. Observe that this threshold is independent of P_2, and hence the same analysis will hold for P_3 and P_4 as well. Thus, we can detect all the nodes in V_1 using a same threshold, i.e. $\tau = min(\sqrt{\alpha_1} \frac{(p+q)}{2}, v_0 \frac{(p+q)}{2})$.

Algorithm Working

Future Scope of Work

Running the algorithm multiple times to beat the third condition

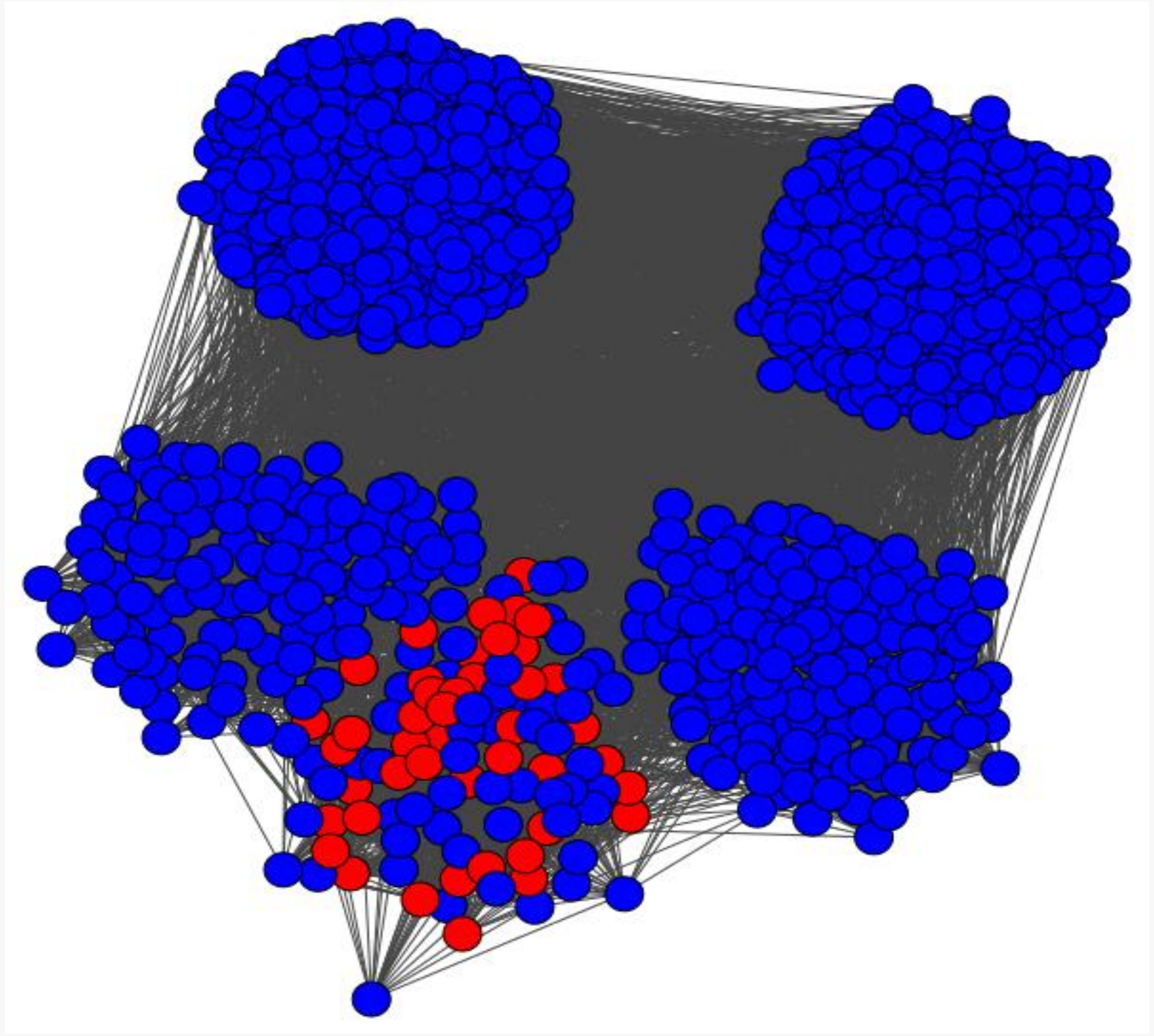
- Due to the (p-q) separability bound, in sparse graphs with lower p, the algorithm breaks down. However, we found that adopting a tighter threshold and running the algorithm multiple times, we can in fact detect the communities and beat the third condition. The motivation for the same is that since the inception of SBM based community detection algorithm, the (p-q) lower bound has got stricter, and it can even get more stricter
- Even better, we can feed the detected V1 nodes in the first run as labelled information to the second run, further tightening the second ("ugly") condition. But, this needs the type II error to be further minimized
- Can be done by adaptive thresholding ?
- Can be done by further improving our choice of threshold, by considering more strict bounds ?
- Can be achieved by assuming reasonable extra side information like bound on number of nodes in the community



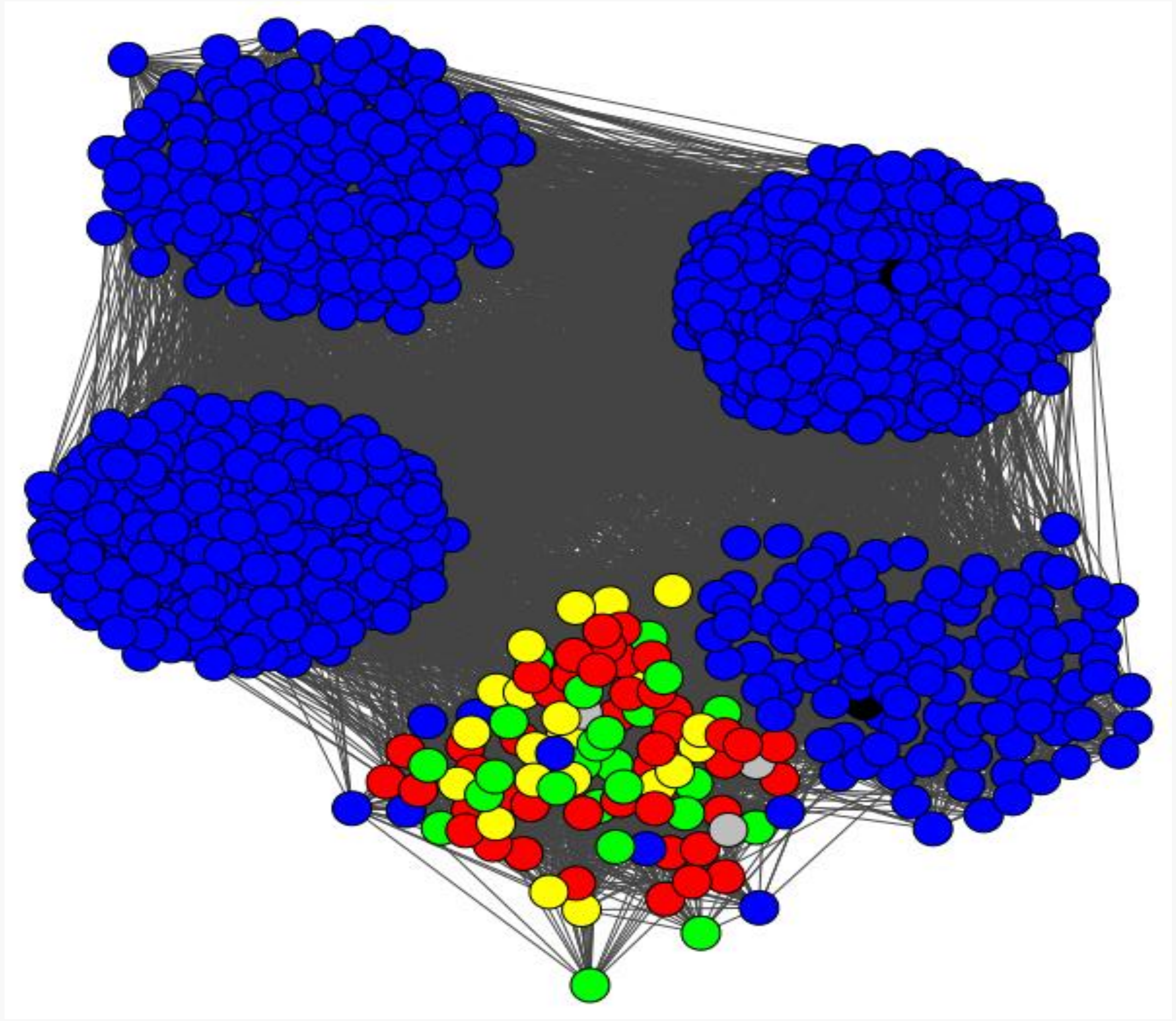
Paper	Min. cluster size K	Density difference $p-q$
Boppana [5]	$n/2$	$\Omega(\frac{\sqrt{p}\log n}{\sqrt{n}})$
Jerrum & Sorkin [18]	$n/2$	$\Omega(\frac{1}{n^{1/6-\epsilon}})$
Condon & Karp [10]	$\Omega(n)$	$\Omega(\frac{1}{n^{1/2-\epsilon}})$
Carson & Impaglizzo [7]	$n/2$	$\omega(\frac{\sqrt{p}}{\sqrt{n}}\log n)$
Feige & Kilian [14]	$n/2$	$\Omega(\frac{1}{n}\log n)$
Shamir [23]	$\Omega(\sqrt{n}\log n)$	$\Omega(\frac{\sqrt{n}\log n}{K})$
McSherry [20]	$\Omega(n^{2/3})$	$\Omega(\sqrt{\frac{pn^2}{K^3}})$
Oymak & Hassibi [21]	$\Omega(\sqrt{n})$	$\Omega(\max\{\frac{\sqrt{n}}{K}, \sqrt{\frac{\log n}{K}}\})$
Giesen & Mitsche[15]	$\Omega(\sqrt{n})$	$\Omega(\frac{\sqrt{n}}{K})$
Bollobas [4]	$\Omega(\frac{n}{\log^{1/8} n})$	$\Omega(\max\{\sqrt{\frac{q\log n}{n}}, \frac{\log n}{n}\})$

Beating condition 3 using successive rounds

After Round 1: SBM(0.3,0.01,1000)

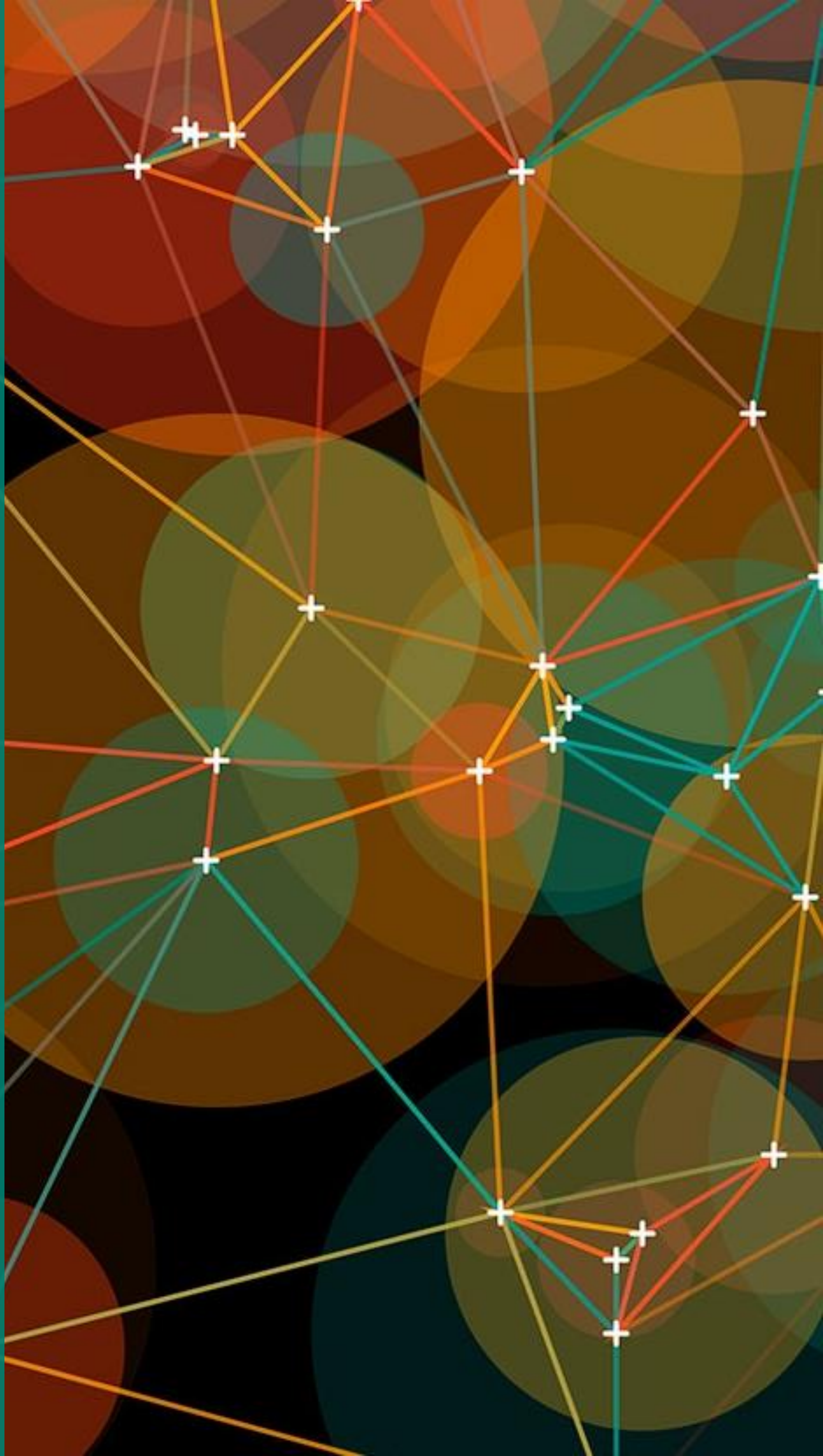


After Round 5: SBM(0.3,0.01,1000)



Key Takeaways !

- Random Graph generation using Stochastic Block model
- Whitening of matrix, forming smaller independent random matrices by partitioning them
- p-q Separability conditions
- Matrix Bernstein inequality
- Future scope in further improving the bound on p-q, and using stricter threshold to remove type-2 errors, enabling us to relabel nodes



Thank You !

- Based on Paper on "Searching For A Single Community in a Graph" by Avik Ray, Sujay Sanghavi and Sanjay Shakkottai
- Graphs generated using igraph package in python