Graph Community Detection

Agrim Gupta (140020003), Abhishek Gore (140040011)

Abstract—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. Whereas, the search problem of the paper we studied aims to only find the nodes in a single community ("**the target**"), given other multiple communities. The algorithm presented utilizes suitable "**side information**", in form of small number of *labelled* nodes in target community. The algorithm presented is a variant of the method of moments that identifies nodes in the target more reliably using side information on target. In our experiments with the implementation, we found that we can actually beat a lower bound concerning with separation of communities generated by a Stochastic Block Model (SBM)
The report is based on "Searching for a Single Community in a Graph" by Avik Ray, Sujay Sanghavi & Sanjay Shakkottai

Index Terms—Graph Clustering, SBM model, Randomized Algorithm

1 Project Description & Setting

Community detection problems in graphs are inclined to partition the graph into small *denser* subgraphs, viz. communities, which show more intra-connectedness than inter-connectedness.

In the paper we studied for the project, a twist to the vanilla community detection was worked upon, by focusing to detect one **target** community with more accuracy and less computations,by utilising some side information in form of given "labelled" nodes for the target community. See Figure 1

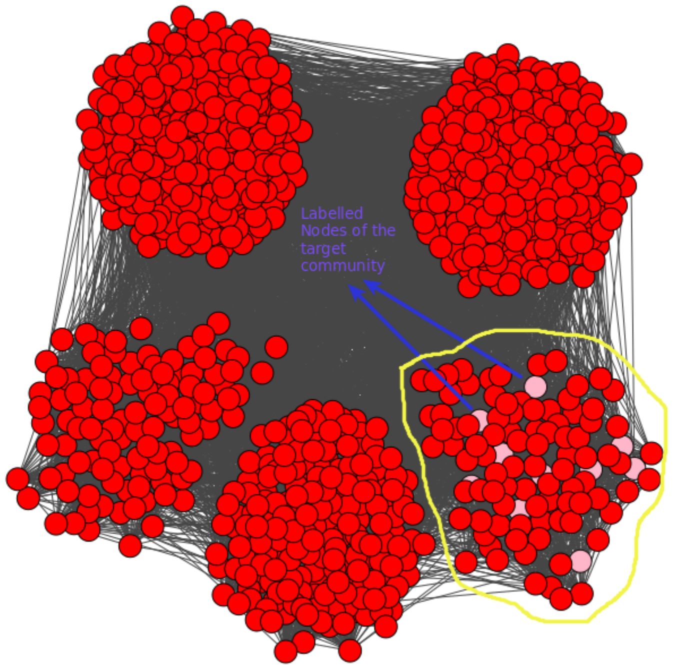

Figure 1. Labeled nodes & Target Community

The algorithm is analyzed over the standard **Stochastic Block Model**(SBM) setting. In *SBM(n,k,p,q)*, 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) Also, the side information is used to get **Biased Weights** of each node. The biased node weight of a node n is the number of labelled nodes reachable from n via atmost r+1 hops. Basically, the biased weights convey the side information about the target community in a mathematically useful way.
The algorithm requires the following constraints to be satisfied from the setting:

- **Biased Weight Constraint**: Let the set of communities be $\{V_i : i \in [k]\}$, where k is the number of communities. WLOG choose the target community as V_1. To honor the biased weight constraint, the node weights $w_j, j \in [n]$, where n is the number of nodes, must satisfy,

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

which is ensured by our scheme to generate biased weights
- **p-q Separation constraint**: This constraint comes from SBM itself. Intuitively, if p and q are close, the communities are 'homogeneous' and it will be tough for the algorithm to separate them. The exact lower bound on (p-q) is discussed in subsequent section. See Figure 2 for a visual cue on the same

2 MAIN RESULTS & IMPLICATIONS

The full algorithm can be found in Appendix 1. In this section, we give a high level overview of how the algorithm works, and various constraints and bounds linked with it.

Overview of the algorithm: The main aim of the algorithm is to estimate the expected value of the adjacency matrix X of the graph, as, the expected value of a column j will be:
$E[X_j]_i = \mu_{c_l}$ = $p\mathbb{1}$(i,j in same community) + $q\mathbb{1}$(i,j in different community) where $\mathbb{1}(.)$ is the indicator function, μ is the community membership vector of community c_j, j belongs to
Once we estimate μ_1, i.e. the community membership vector of the target community 1, all that needs to be done is thresholding to detect the members. Define the following

$$A := \frac{1}{n}\Sigma_j E[X_j]E[X_j]^T = \Sigma_{i=1}^k \alpha_i \mu_i \mu_i^T$$

$$B := \frac{1}{n}\Sigma_j \bar{w_j} E[X_j]E[X_j]^T = \Sigma_{i=1}^k w_i \alpha_i \mu_i \mu_i^T$$

where $\bar{w_j} = E[w_j]$, k is the number of communities, and α_i is the fraction of nodes in community i, i.e. $\frac{n_i}{n}$, n_i being nodes in community i
The main idea is to recover μ_1 by **whitening** the matrix B using A. The complete procedure to do so is discussed in appendix 2. In a nutshell, what we want to do is to obtain orthonormalised eigenvectors of B, and once we get those, the leading eigenvector would give us μ_1, due to the constraint on the biased weights, i.e. weights of nodes in community 1 are greater than others.

That being said, we now require estimates for matrices A and B. A natural choice of an estimator for A will be $\frac{1}{n}\Sigma_j X_j X_j^T$, however this is a good estimator of $E[XX^T]$ and not $E[X]E[X^T]$. To solve this issue, we partition X, and take row indexes of one partition, column indexes corresponding to other partition, to get a submatrix for which we can estimate A using the above natural trick. Not knowing matrices A and B a priori imposes a penalty on the algorithm, which appears up as a concentration bound on weights, required to estimate B to a reasonable accuracy.

Partitioning also affects the threshold being used to detect the community members from the membership vector μ_1. Under the conditions discussed in the next section, the algorithm in appendix 1 detects the nodes belonging to target community V_1 lying in partition P_1, by choosing a threshold of $\tau = \sqrt{\alpha_1}\frac{p+q}{2}$, making at most $o(\alpha_1 n)$ erroneous nodes. However to detect the nodes lying in P_2, P_3 & P_4 with the same error performance, we need to set up a threshold of $\tau = v_0\frac{p+q}{2}$, where $v_0 = |V_1 \cap P_1|$. By choosing the minimum out of the two, we can choose one single threshold for the algorithm.

Preconditions required by the algorithm

- **p-q Separation Condition**: This condition fundamentally determines when communities are identifiable in a stochastic block model. As q increases, the distinction between communities get lost. See Figure 2

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

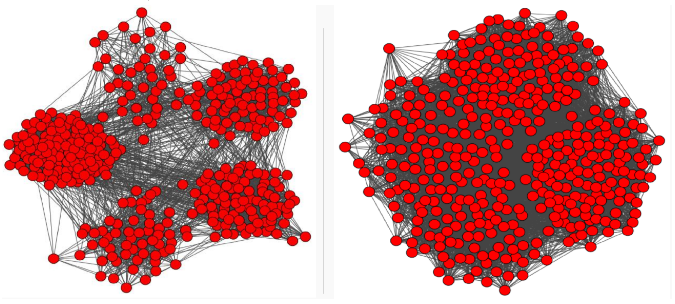

Figure 2. $SBM(p = 0.3, q = 0.01, n = 400)$ (left), $SBM(p = 0.3, q = 0.05, n = 400)$ (right)

- **Weight Concentration**
Define $\sigma_1(R) := E[w_j | j \in V_1]$, $\gamma_2 := max_{i\in[k], j\in V}|w_j - E[w_j|j \in V_i]|$, and $\xi(n) = o(\sqrt{log(n)})$ be a slowly growing function. Then with high probability the max. deviation of the weights are bounded as:

$$\frac{\gamma_2}{\sigma_1(R)} = O(min\{\frac{\alpha_{min}^4(p-q)^4}{\alpha_{max}^4 p^4 \xi(n)}, \frac{\alpha_{min}^5\sqrt{n}(p-q)^5}{\alpha_{max}^4 p^{4.5}\xi(n)} - 1\}) \quad (3)$$

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

Concluding the section on results, if the setting of algorithm satisfies Equation (1), Equation (2) & Equation (3), we can detect nodes in target community with at most o(n) erroneous nodes.

3 Summary of Concentration Inequalities & Tools used

Matrix Bernstein Inequality

Let $\{A_j\}_{j=1}^n$ be a sequence of i.i.d. real d_1*d_2 matrices s.t. $E[A_j] = 0$, $||A_j|| \leq L$. Define $Z = \sum_{j=1}^n A_j$. Let $\sigma^2 = max\{||E[ZZ^T]||, ||E[Z^T Z]||\}$.

$$P(||Z|| \geq t) \leq (d_1 + d_2)exp(\frac{-t^2/2}{\sigma^2 + Lt/3}), \forall t \geq 0$$

The matrix bernstein inequality is used to bound the difference between the estimates matrix and the actual matrix in the algorithm. It is used to get concentration results on estimated A matrix and some intermediary matrices coming in the midst of proving the weight concentration result (i.e. Equation (3))

Chernoff bounds

Chernoff bound is used to prove the threshold $v_0\frac{p+q}{2}$ for detection of target community nodes in partitions P_2, P_3 & P_4. Let the estimated community from P_1 nodes be $\widehat{V}_{P_1}$, then from the preconditions, $\widehat{V}_{P_1}$ will have at most $o(\alpha_1 n)$ erroneous nodes. For any $j \in P_2$, define $d_j(\widehat{V}_{P_1})$ be the number of edges j shares with nodes in $\widehat{V}_{P_1}$. Let $v_0 = |\widehat{V}_{P_1}|$, $v_1 = |\widehat{V}_{P_1} \cap V_1|$ be the correctly detected nodes, and $e_1 = |\widehat{V}_{P_1} \cap V_1^C|$ be the erroneous nodes.

From SBM it is clear that any node $j \in V_1 \cap P_2$ will form an edge with v_1 nodes w.p. p, and with e_1 nodes w.p. q. Hence, using the distribution for nodes $j \in V_1 \cap P_2$, and using chernoff bound to get an upper bound on

$$P(d_j(\widehat{V}_{P_1}) < (v_1 p - \sqrt{v_1 p log(n)} + e_1 q - \sqrt{e_1 q log(n)})) \leq \delta$$

Simplifying & using $v_1 = \Theta(\alpha_1 n)$, $e_1 = o(\alpha_1 n)$

$$d_j(\widehat{V}_{P_1}) \geq (v_0\frac{p+q}{2} \approx (v_1 p - \sqrt{v_1 p log(n)} + e_1 q - \sqrt{e_1 q log(n)}\,))$$

we get, $P(d_j(\widehat{V}_{P_1}) \geq v_0\frac{p+q}{2}) \geq 1 - \delta$, i.e. with high probability, $d_j(\widehat{V}_{P_1}) \geq v_0\frac{p+q}{2}$, $\forall j \in V_1 \cap P_2$.

Following similar steps for $j \in V_1^C \cap P_2$, now any node j will form an edge with v_1 nodes w.p. q, and with e_1 nodes w.p. p. Using similar steps, with high probability $d_j(\widehat{V}_{P_1}) < v_0\frac{p+q}{2}$, $\forall j \in V_1^C \cap P_2$.

Hence, this motivates use of $v_0\frac{p+1}{2}$ as a threshold to detect nodes in P_2. Since the expression does not contain any term specific to P_2 same threshold can be readily applied to P_3 & P_4

Weyl's inequality and Wedin's theorem

These were used to prove a set of lemmas which bound the spectral norm difference of the matrices discussed in earlier sections.

4 Additional Insights Obtained

We implemented the algorithm using *igraph* package in python, and did some experiments with the parameters of the algorithm. We found out that the p-q separation condition given by Equation (2) can be further improved upon, by:

- Doing the algorithm over multiple rounds and averaging results allows us to get a reasonable estimate of the community and reduce variance, while taking linear time. This is linear in time complexity, but yields much better performance. It's just like using an ensemble of learners to reduce variance, a technique widely used in ML.

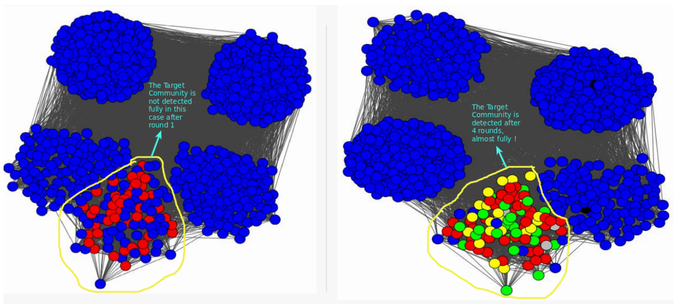
The Target Community is not detected fully in this case after round 1
The Target Community is detected after 4 rounds, almost fully !

Figure 3. Beating the p-q lower bound ! $SBM(p = 0.3, q = 0.01, n = 1000)$ Left pic: The Nodes detected after round 1 (i.e. the original algorithm) Right pic: The Nodes detected in subsequent rounds. Red:Round 1, Green:Round 2, Yellow: Round 3, Grey:Round 4

- Doing the algorithm over multiple rounds and using results of first run as input to the next. This method can help us get much better results as we drastically improve the quality of side-information we have, as we **boost up** the side information content. However, it is very sensitive to false positive errors in the first run, as it would detect those communities too in second run, if not taken care of. But, we can get around this by choosing a stricter threshold, which reduces false positives.

We found the python *igraph* library easy and fun to work with. Visit the Github Repository for the codes written for the project.

5 Appendix-1: Algorithm

Input: Adjacency matrix X, number of communities k, biased weights $w_i i \in [n]$ & threshold τ
Output: $\widehat{V_1}$

1) Partition nodes into 4 sets P_1, P_2, P_3 & P_4 at random
2) Compute matrices $\widehat{A}_1 = \frac{1}{\sqrt{|P_3|}} X_{P_1,P_3}$ & $\widehat{A}_2 = \frac{1}{\sqrt{|P_3|}} X_{P_2,P_3}$
3) Compute vector $\widehat{m}_1 = \frac{1}{|P_1|} \Sigma_{j \in P_1} X_{P_1,j}$
4) Compute matrix $\widehat{B} = \frac{1}{|P_4|} \Sigma_{j \in P_4} w_j X_{P_1,j} X_{P_2,j}^T$
5) $\widehat{\mu}_{P_1}, \widehat{\alpha}_1 \leftarrow$ SearchSubroutine($\widehat{A}_1, \widehat{A}_2, \widehat{B}, \widehat{m}_1, k$)
6) Compute $V_{P_1} = \{j \in P_1 : \widehat{\mu}_{P_1,j} > \tau\}$
7) Repeat steps 2-5 with P_i's rotated in order to estimate $\widehat{\mu}_{P_2}, \widehat{\mu}_{P_3}, \widehat{\mu}_{P_4}$. Use them to compute $V_{P_2}, V_{P_3}, V_{P_4}$ as in step 6.
8) Return community $\widehat{V_1} = V_{P_1} \cup V_{P_2} \cup V_{P_3} \cup V_{P_4}$

5.1 SearchSubroutine

Input: $\widehat{A}_1, \widehat{A}_2, \widehat{B}, \widehat{m}_1, k$ **Output**: $\widehat{\mu}_{P_1}, \widehat{\alpha}_1$

1) Compute rank k-svd of matrices $\widehat{A}_1$ & $\widehat{A}_2$: $\widehat{A}_1 = U_1 D_1 V_1^T$ & $\widehat{A}_2 = U_2 D_2 V_2^T$
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 $W_1^T \widehat{B} W_2$
4) Compute $z = U_1 D_1 u_1$
5) Compute $a = u_1^T W_1^T \widehat{m}_1$
6) Return $\widehat{\mu}_{P_1} \leftarrow z/a$ & $\widehat{\alpha}_1 \leftarrow a^2$

6 Appendix-2: Whitening of estimated matrix B

1) Do rank k SVD of A, to get $A = UDU^T$, let $W := UD^{-\frac{1}{2}}$.
   - Observe, $W^T A W = I_k = \sum_{i=1}^k \tilde{\mu_i} \tilde{\mu_i}^T$, where $\tilde{\mu_i} := \sqrt{\alpha_i} W^T \mu_i$.
   - Also, note that addition of k terms of type $\tilde{\mu_i} \tilde{\mu_i}^T$, results in I_k, which happens only if corresponding μ_i are *orthonormal* vectors in $\mathbb{R}^k$
2) Hence we have obtained $\tilde{\mu_i}$ which are whitened versions of μ_i.
   - Now compute $R := W^T B W = \sum_{i=1}^k w_i \tilde{\mu_i} \tilde{\mu_i}^T$
   - Now, since $\tilde{\mu_i}$ are orthonormal, the above equation represents an eigenvalue decomposition of the k*k size matrix R, with eigenvectors $\tilde{\mu_i}$ and corresponding eigenvalues w_i .
   - Thus, $\tilde{\mu_1}$ – the whitened vector corresponding to the target community – is now the leading eigenvector of R, because $w_1 > w_i, \forall i \neq 1$
3) Find $\tilde{\mu_1}$ by setting it to be the leading eigenvector of R. Finally recover μ_1 from $\tilde{\mu_1}$ in the following steps.
   - First compute $z := UD^{\frac{1}{2}} \tilde{\mu_1} = \sqrt{\alpha_1} \mu_1$
   - Next compute $m_1 := \frac{1}{n} \sum_{j=1}^n E[X_j] = \sum_{i=1}^n \alpha_i u_i$ Using this, recover $\sqrt{\alpha_1} = \tilde{\mu_1}^T W^T m_1$
   - Divide the obtained z, by recovered $\sqrt{\alpha_1}$ to get μ_1 and hence the nodes in V_1