Searching for a Single Community in a Graph

Avik Ray [*], Sujay Sanghavi [†], and Sanjay Shakkottai [‡]

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712

11^{th} August 2016

Abstract

In standard graph clustering/community detection, one is interested in partitioning the graph into more densely connected subsets of nodes. In contrast, the *search* problem of this paper aims to only find the nodes in a *single* such community, the target, out of the many communities that may exist. To do so , we are given suitable side information about the target; for example, a very small number of nodes from the target are labeled as such.

We consider a general yet simple notion of side information: all nodes are assumed to have random weights, with nodes in the target having higher weights on average. Given these weights and the graph, we develop a variant of the method of moments that identifies nodes in the target more reliably, and with lower computation, than generic community detection methods that do not use side information and partition the entire graph. Our empirical results show significant gains in runtime, and also gains in accuracy over other graph clustering algorithms.

1 Introduction

Community detection, or graph clustering, 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. Typically, when graphs represent similarity or affinity relationships between nodes, these subsets represent communities of similar nodes. Also typically, this problem has primarily been considered in the unsupervised setting, where the only input is the graph itself and the objective is to partition all or most of the nodes.

In this paper we look at a different, but related, community detection task, which we will refer to as the *search problem*. Our objective is to use the graph to find a single community of nodes – which we will call the *target community* – for which we have been given some relevant but quite noisy side information. We would like to do so more reliably, and with lower computation, than existing methods that do not use side information.

Our motivations are two-fold: *(i)* it is often the case that the network analyst is looking for nodes with a-priori specified characteristics, and *(ii)* it is rare that we are faced with a "pure" graph analysis problem; typically there is extra non-graphical side information that, if used properly, could make the inference task easier.

As an example setting, consider the case where we have some nodes from the target community explicitly marked as such, and our task is to recover the remaining nodes. This is a situation that frequently arises in military/intelligence settings, and also in analysis of regular consumer social networks, internet/web graphs etc. In military intelligence it can be useful to recover a single community which a known suspect is part of. Besides explicit node labels, side information could also come from meta-information one may have about the nodes; e.g. from text analysis if the graph is a web graph, or from browse/activity history of users in a social network. In recommendation system or targeted advertising it is useful to learn a community of users with a specific interest

[*]avik@utexas.edu
[†]sanghavi@mail.utexas.edu
[‡]shakkott@austin.utexas.edu

(e.g. sports) using the knowledge of how users interact with relevant contents (e.g. sports news and images). Our aim is to find a principled way to use such side information *and* the graph itself.

Our contributions are as follows.

(i) We develop a simple yet generic framework for how side-information is to be specified: each node is given a (possibly random) weight, with nodes in the target community having higher weight on average than nodes not in the target – we call these *biased weights*. This setting would thus split an overall data + graph analysis objective into two: the analyst needs to devise a (application-dependent) procedure to convert her side information into biased node weights; these are then used by our algorithm.

(ii) Given such biased weights, we develop a new spectral-like algorithm – specifically, a variant of the 2^{nd} order method of moments – to find the nodes in the target community. We call this *Community Search* below. In the following, we first provide the basic intuition behind it by considering the case where we have access to the population statistics of a graph coming from a stochastic block model, and then formally describe the algorithm.

(iii) Our main results characterize the effectiveness of this algorithm in finding the target community; we study this in the standard stochastic block model setting with many communities. Analytically, we show that it matches (potentially upto log factors) the analytical guarantees of the state of the art unsupervised community detection methods; empirically, we show that the method outperforms these methods even with very noisy side information (e.g. very small number of labeled nodes), and has significantly lower computational complexity.

(iv) We also specialize our results to the case where the side information is in the form of a small number of labeled nodes; for this case we show how one can effectively convert this to node weights, even for sparse graphs. Our experiments on a real world network further corroborate the practical applicability of this method.

1.1 Related work

While no other work has considered the problem of searching a single community in a graph, there has been a lot of research in three closely related fields; that of unsupervised and semi-supervised graph clustering, method of moments, and learning with side-information. Each of these threads have a rich history – here we cover the ones most relevant to this paper.

Unsupervised graph clustering: Graph clustering or community detection has been widely studied mainly in the unsupervised setting where nodes do not have any associated labels. There is a vast literature of graph clustering algorithms both in the setting where clusters are non-overlapping [1] and overlapping [2]. The most widely studied generative model for non-overlapping clusters in a graph is the planted partition or stochastic block model [3]. Assuming this model many algorithms have been proposed which provide statistical guarantees of recovery of all hidden clusters. These algorithms can be broadly divided into three categories (i) spectral clustering [4, 5, 6, 7, 8] (ii) convex optimization [9, 10, 11] and more recently (iii) tensor decomposition [12, 13].

Semi-supervised graph clustering: The graph clustering problem has also been explored in a semi-supervised settings, where some of the nodes and/or edges are explicitly labeled. Many optimization and kernel based algorithms have been proposed [14, 15] to solve this problem. The popular label propagation based clustering algorithms [16, 17] are also essentially semi-supervised graph clustering algorithms with labeled nodes.

Method of Moments: This is a classical parameter estimation technique, where the parameters to be estimated are described in terms of the moments from the true distribution. Empirical moments are now used to replace the true moments, leading to parameter estimates [18]. There has been much recent interest in these methods for many statistical learning problems. These include learning Gaussian mixture models [19, 20], LDA topic models [21], hidden Markov models [22] etc.

Others: There is a broader machine learning literature that incorporates the availability of extra side-information into existing models and algorithms. In the context of LDA topic models, side-information maybe available in the form of extra response variables for each document [23], or additional text review information of products [24]. In collaborative filtering, side-information can be of the form of item or user graph [25]. In overlap graph clustering, side-information maybe available in form of node attributes [26].

In this paper we consider the community search problem with side-information either in the form of biased node weights or a small set of labeled nodes. We note that early work leading to the results in this paper appears as a poster in [27].

2 Settings and Algorithm

Stochastic Block Model: Consider a graph $G = (V, E)$ with n nodes and k non-overlapping communities that partition the vertex set as $V = \cup_{i=1}^{k} V_i$. Let $\alpha_i = |V_i|/n$ be the fraction of nodes in the i-th community. In a stochastic block model the edge set E is generated as follows. Let $0 < q < p < 1$. Then for any two nodes in the same community $r, s \in V_i$ we have $P((r, s) \in E) = p$, and when r, s are in different community then $P((r, s) \in E) = q$. We define this as the (n, k, p, q) stochastic block model.

Target community and side information: In the search problem we are interested in the recovery of *one* target community, in this paper, without loss of generality, consider V_1 to be this target community. We are also provided with some side-information on this target community V_1. The side-information is in the form of *biased node weights.* Suppose for each node $j \in V$ we are given a biased weight $w_j > 0$. These weights are generated by a random process satisfying the condition that for any node $j \in V$ we have $E[w_j | j \in V_1] > E[w_j | j \in V_i]$ for all $i \neq 1$.

These biased weights may be computed using a set of *labeled nodes* from the target community $\mathcal{L} \subset V_1$ (see Section 3.2). These weights can also arise from other available sources of side-information (e.g. "likes" and "shares" of facebook posts). The main goal is to solve this search problem faster than the time required to recover all k communities, and without any loss in estimation accuracy.

2.1 Algorithm

In this section we describe our main algorithm called **Community Search**. Let X be the adjacency matrix of the graph G. Also define community membership vectors $\mu_1, \ldots, \mu_k$ where $\mu_i \in \mathbb{R}^n$, as follows. Let $\mu_{j,i}$ be the $j-$th coordinate of vector μ_i. Then,

$$\mu_{j,i} = \begin{cases} p & \text{if } j \in V_i \\ q & \text{otherwise} \end{cases}$$

Note that these μ_i-s are linearly independent and the community memberships of the nodes can be obtained from these membership vectors via thresholding. The main purpose of our algorithm is to estimate the membership vector of the first community μ_1 (which can then be used to recover nodes in V_1).

Intuition behind our method: To understand the core of our technique, let us suppose here – just for intuition – that we actually had access to the "average" adjacency matrix $E[X]$ (recall that X is the actual adjacency matrix of the stochastic block model), and let $E[X_j]$ be the average of the j^{th} column. Then it is easy to see that $E[X_j] = \mu_{c_j}$, where $c_j \in 1, \ldots, k$ is the community that node j belongs to. This means that the following holds for the matrix A defined below:

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

Similarly, let us now also suppose that we see the "average" node weights $\bar{w}_j = E[w_j]$ for every node j. Then, the following holds for the matrix B defined below:

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

where in the above, for each cluster i, we have defined ω_i be the averaged weights of all nodes in that cluster. By the bias condition, we have that $\omega_1 > \omega_i$ for all $i \neq 1$.

Note that both the A and B as defined above are symmetric positive definite rank-k matrices, with the column space of each spanned by the μ_i's. However note also that our desired vector μ_1 may not be an eigenvector of either A or B; indeed if the target community V_1 is small, it may be quite far from the leading eigenvector of either matrix.

The main idea is that we can still recover μ_1 by *"whitening" B using A*, a process we describe in the proto-algorithm below. The description also provides the (simple) reason why it works – in this idealized case where average X and w are available.

Proto-algorithm (and explanation):

1. Compute matrices A and B as described above,
2. Perform rank-k svd of A as $A = UDU^T$, and let $W := UD^{-1/2}$. Also note that,

$$W^T AW \;=\; I_k \;=\; \sum_{i=1}^{k} \tilde{\mu}_i \tilde{\mu}_i^T$$

   where we define $\tilde{\mu}_i := \sqrt{\alpha_i} W^T \mu_i$. Now, we see that the addition of k terms of the type $\tilde{\mu}_i \tilde{\mu}_i^T$ results in I_k; this can *only* happen if the corresponding $\tilde{\mu}_i$ are *orthonormal* vectors in $\mathbb{R}^k$. The vectors $\tilde{\mu}$ are thus "whitened" versions of the original μ vectors.

3. Next we compute the following matrix.

$$R \;:=\; W^T BW \;=\; \sum_{i=1}^{k} \omega_i \tilde{\mu}_i \tilde{\mu}_i^T$$

   Now, since $\tilde{\mu}_i$ are orthonormal, the above equation represents an eigenvalue decomposition of the $k \times k$ size matrix R, with eigenvectors $\tilde{\mu}_i$ and corresponding eigenvalues ω_i. Thus, $\tilde{\mu}_1$ – the whitened vector corresponding to the target community – is now the *leading eigenvector of R*, because $\omega_1 > \omega_i$.

4. Find $\tilde{\mu}_1$ by setting it to be the leading eigenvector of R. Finally we can recover μ_1 from $\tilde{\mu}_1$ in two steps. First compute $z := UD^{1/2}\tilde{\mu}_1 = \sqrt{\alpha_1}\mu_1$. Next compute vector

$$m_1 \;:=\; \frac{1}{n} \sum_{j=1}^{n} E[X_j] \;=\; \sum_{i=1}^{k} \alpha_i \mu_i$$

   We can recover $\sqrt{\alpha_1} = \tilde{\mu}_1^T W^T m_1$. Then simply divide the z defined above by this to find μ_1.

An issue: Although simple, it is not straight forward to convert this intuition to an algorithm because due to inter dependencies it becomes hard to estimate these A and B matrices. In particular note that in the actual problem we are given adjacency matrix X, and a natural impulse is to approximate A using the matrix

$$\frac{1}{n} \sum_{j} X_j X_j^T$$

Unfortunately, this is a good approximation to $E[XX^T]$, but $E[XX^T] \neq E[X]E[X]^T$ – and we require the latter. However we can get around these dependencies by first partitioning the graph. This is outlined in Algorithm 1 below.

For any two subsets $P, Q \subset [n]$ let $X_{P,Q}$ denote the submatrix of X corresponding to the rows and columns in set P and Q respectively. The input parameters to Algorithm 1 are the adjacency matrix X, number of communities k, the set of biased node weights $(w_1, \ldots, w_n)$, and a threshold τ. The output is the community estimate $\hat{V}_1$.

3 Main Result

In this section we present our main theoretical results. First we show that when the set of biased weights $(w_1, \ldots, w_n)$ satisfy certain mild sufficient conditions, then Algorithm 1 is guaranteed to recover the target community V_1. Later we show how such weights can be obtained even with a set of labeled nodes from the target community.

Algorithm 1 Community Search

Input: Adjacency matrix X, k, biased weights $(w_1, \ldots, w_n)$, threshold τ
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 $\hat{B} = \frac{1}{|P_4|} \sum_{j \in P_4} w_j X_{P_1,j} X_{P_2,j}^T$
5: $\hat{\mu}_{P_1}, \hat{\alpha}_1 \leftarrow \text{SearchSubroutine}(\hat{A}_1, \hat{A}_2, \hat{B}, \hat{m}_1, k)$
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 $\hat{\alpha}_1 \leftarrow a^2$

3.1 Recovery using biased weights

When side information is available in the form of biased weights w_j for each node $j \in V$, these weights need to be *informative* about the target community V_1 so that it could be recovered. Clearly *good* side-information will lead to a better performance of any search algorithm. We quantify this quality of information in the following set of assumption (condition (A1) and (A2)) on the biased weights. The third condition (A3) is a more fundamental condition that determines when the community structure itself is identifiable in a stochastic block model.

- *(A1) Average weight bias:* Under this condition the expected weight of a node in community V_1 is greater than the expected weight of a node in any other community V_i. Precisely the weights satisfy:

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

  This weight bias allows us to determine that community V_1 is being searched / preferred over the remaining communities. However we only require this to hold in expectation and the actual weights themselves may vary significantly. Clearly any algorithm which only uses the weight bias to determine community membership by simple thresholding will perform very poorly.

- *(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,

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

  This condition dictates that the maximum variation of the weights γ_2 is also small compared to the difference between the largest and second largest expected weights $\sigma_1(R) - \sigma_2(R)$. Since the weights are used primarily to construct the matrix B in Algorithm 1, this condition ensures that the matrix B can be estimated up to a tolerable error.

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

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

  This condition fundamentally determines when communities are identifiable in a stochastic block model and similar conditions are required for other community detection algorithms [12, 7, 9]. The more the gap $p-q$ easier it is to identify communities. Hence this condition gives a lower bound on $p-q$ which is required for community identifiability.

Theorem 1 shows that under the above assumptions on the biased weights Algorithm 1 can reconstruct community V_1 with high accuracy.

Theorem 1. *Consider a (n, k, p, q) stochastic block model satisfying condition (A3). Given biased weights $(w_1, \ldots, w_n)$ satisfying conditions (A1), (A2), then Algorithm 1 recovers community V_1 with fraction of error nodes $o(1)$ with high probability.*

Remark 1. *For a stochastic block model with equal community sizes n/k condition (A3) reduces too $\frac{(p-q)^2}{p\sqrt{p}} = \tilde{\Omega}\left(\frac{k}{\sqrt{n}}\right)$. When $p = \Theta(p-q)$ this has the same scaling as other community detection algorithms [9, 12, 7]. Therefore even in sparse graphs where $p, q = \Theta\left(\frac{\log n}{n}\right)$ or for small community sizes up to $\Omega(\sqrt{n})$ nodes Algorithm 1 can recover the community.*

In Theorem 1 the $o(1)$ fraction error can be easily converted to a zero error guarantee using an additional post-processing step. Instead of estimating community 1 nodes inside partition P_1 we can estimate those in partition P_2, first by observing for each node $j \in P_2$ the number of edges shared with the estimated set V_{P_1}, followed by thresholding. Since V_{P_1} estimates $V_1 \cap P_1$ up to only $o(\alpha_1 n)$ error nodes this does not cause any errors in thresholding, with high probability. This post-processing step is also independent of the previous steps in the algorithm since the edges between partitions P_1 and P_2 are not utilized in Algorithm 1. The following theorem formalizes this idea.

Theorem 2 (Exact recovery)**.** *In a (n, k, p, q) stochastic block model, under assumptions (A1)-(A3), Algorithm 1 with an additional degree thresholding step can recover community V_1 completely with high probability.*

We prove Theorems 1 and 2 in Appendix A.

3.2 Recovery using labeled nodes

Biased weights, as required in Theorem 1, can be obtained from a small set of labeled nodes $\mathcal{L}$ as follows:

- Choose a radius r
- Weight w_i is the number of edges between nodes in $\mathcal{L}$ and nodes at a distance of r hops from node i

Note that the weight can also be viewed as the number of neighbors of the set $\mathcal{L}$ which are at a distance r from node i. Larger choice of radius r means less variance in the weights, but also potentially less bias if it becomes too large. For example, $r = 1$ means only neighbors of labeled nodes get weights; this is very high bias but also high variance.

The theorem below provides the correct way to choose the radius r such that the weights w_i can be made to satisfy conditions (A1), (A2). This means that even with such weights computed via labeled nodes we can efficiently find community V_1 using Algorithm 1. Note that when $p \geq \frac{1}{\sqrt{n}}$ then with high probability the labeled nodes in $\mathcal{L}$ has neighbors with any other node $i \in V_1 \backslash \mathcal{L}$, hence the number of common neighbors between i and nodes $l \in \mathcal{L}$ can be taken as weights w_i which will satisfy conditions (A1), (A2). However this does not work for sparse graphs when $p < \frac{1}{\sqrt{n}}$. In the following theorem we show that even for $p = \Theta\left(\frac{\log n}{n^\epsilon}\right)$, $\frac{1}{2} \leq \epsilon \leq 1$ the weights chosen by the above procedure and a correct r will work.

Theorem 3. *Consider a (n, k, p, q) stochastic block model satisfying condition (A3) where $p = \Theta\left(\frac{\log n}{n^\epsilon}\right)$, $q = \Theta\left(\frac{\log n}{n^\epsilon}\right)$, $p - q = \Theta\left(\frac{\log n}{n^\epsilon}\right)$ and all equal sized communities. Given $L = \tilde{\Omega}(n^{\epsilon/2})$ labeled nodes, the biased weights computed with $r = \frac{2\log(n^\epsilon/L)}{\log np}$, satisfy conditions (A1), (A2) with high probability.*

We prove this in Appendix A.3. For simplicity in Theorem 3 we assume equal community sizes, however this can be extended to unequal but comparable communities sizes.

3.3 Parallel semi-supervised graph clustering

Our algorithm naturally provides a method for the standard semi-supervised graph clustering problem. This is the setting where we are given a small number of labeled nodes from every community, and we are interested in recovering all communities. In such a scenario we can apply the community search algorithm to search for each individual community using the labeled nodes in that target community. Moreover this search can be performed in parallel. Therefore Algorithm 1 can also be used as a parallel graph clustering algorithm. Note that the vector m_1 and matrices A_1, A_2 remain the same for individual searches, only matrix B should be computed separately for every target community. Section 4 shows some numerical results evaluating the performance of Algorithm 1 in this semi-supervised graph clustering setting.

3.4 Comparison

In this section we compare the theoretical performance of our algorithm with other unsupervised graph clustering algorithms.

For graphs with equal communities of size n/k, convex optimization based algorithms by [9, 10, 28] can achieve the performance bound $\frac{(p-q)}{\sqrt{p}} = \tilde{\Omega}\left(\frac{k}{\sqrt{n}}\right)$. In comparison our algorithm achieves a slightly higher bound $\frac{(p-q)^2}{p\sqrt{p}} = \tilde{\Omega}\left(\frac{k}{\sqrt{n}}\right)$. However when $p = \Theta(p-q)$[1] both bounds are equivalent (upto log factors) implying our algorithm can recover communities even in sparse graphs with $p, q = \Theta\left(\frac{\log n}{n}\right)$ and for growing number of communities $k = O(\sqrt{n})$. In terms of runtime our algorithm runs in $O(n^2 k)$ time faster than $\Omega(n^3)$ time required by convex optimization based algorithms.

The Community Search by Whitening algorithm is also faster than tensor decomposition based graph clustering algorithm by [12]. Note that the first step of this tensor algorithm is to compute a whitening matrix using rank–k svd, which is identical to the search algorithm. In the remaining steps, for the tensor algorithm, the bulk of the computation is a rank–k tensor decomposition requiring $O(k^5)$ computation, which is slower than rank–1 svd computed in $O(k^2)$ time by the search algorithm. This is corroborated by our experiments in Section 4.

Recently a quasi–linear time graph clustering algorithm was presented by [29] for the case when number of communities $k = O(1)$. In comparison our algorithm can be applied even when the number of communities scale as $k = O(\sqrt{n})$, and it requires much lesser knowledge of model parameters than the former.

4 Experiments

In this section we present our numerical results showing the performance of the Community Search algorithm on synthetic and real datasets. We compare our algorithm with the Spectral clustering algorithm by [5] and the Tensor decomposition based clustering algorithm by [12]. We generate synthetic datasets according to the stochastic block model (see Section 2) with $n = 1000$ nodes, $k \in \{5, 8\}$ communities, and different values of p and q. The real world network we consider is the US political blogosphere network first introduced in [30]. The Spectral clustering algorithm [5] requires clustering of the rows corresponding to bottom k eigenvectors of the normalized Laplacian. Although k-means may be used for this, it tends to converge to local minima resulting in poor performance. To prevent this we perform clustering of the rows via the hierarchical SLINK algorithm [31]. We refer to this Spectral+SLINK algorithm simply as Spectral clustering in the remaining section. Our

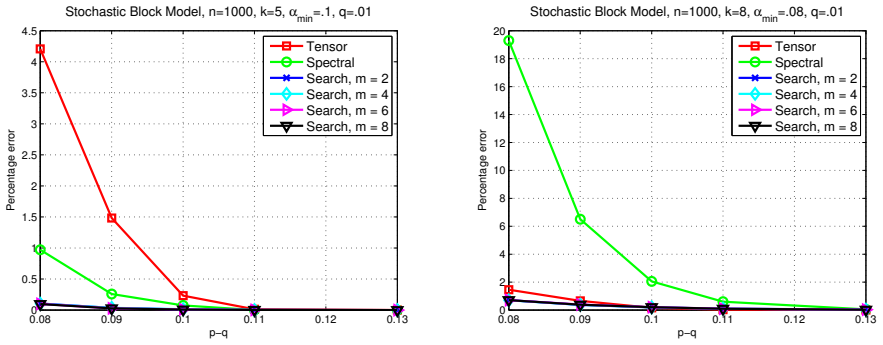
[1] This is the case in most real sparse networks when $p = \Theta(\log n/n)$, if not then it becomes impossible for any algorithm to recover communities in this regime as shown by [11].

algorithm implementations are all in Matlab. We consider two types of side information: *labeled nodes* and *synthetic weights.*

Labeled Nodes: As discussed earlier, this is a natural means of providing side information to the algorithm. A set of m labeled nodes are randomly chosen from the target community V_1. The corresponding weights are then computed as described in Section 3.2 with $r \in \{1, 2\}$.

Synthetic Weights: We synthetically generate three sets of weights, each of which are (on average) larger over the target community. These weights are generated as follows. A pair of weights (w_1, w_2) are first chosen to be one of $\{(5, 8), (5, 10), (5, 12)\}$. For each node in community V_1, we set the node's weight to be w_2 with probability 0.8, and w_1 other-wise. For all other nodes in $V \backslash V_1$, we swap the probabilities, i.e., we set a node's weight to be w_2 with probability 0.2 and w_1 other-wise. This process generates three possible values of the expected node weights in the target community, $\sigma_1(R) \in \{7.4, 9, 10.6\}$.

Performance Metrics: Note that our algorithm directly uses labeled nodes/biased node weights, and the graph to infer the target community. The baseline algorithms however first estimate all communities in the graph, then it computes the average node weight in each community, finally outputs as target the community which has the highest average node weight. The estimation error for the i-th community is given as $e_i = |\{j \in V : j \in V_i, j \notin \hat{V}_i \text{ or } j \in \hat{V}_i, j \notin V_i\}|$. We compute the error for searching each community and plot either the overall average, or average over a subset of clusters. Let $T_{\mathcal{A}_1}, T_{\mathcal{A}_2}$ be the runtimes of algorithms $\mathcal{A}_1, \mathcal{A}_2$ respectively. Then we define speedup s of algorithm $\mathcal{A}_1$ over algorithm $\mathcal{A}_2$ as $s = T_{\mathcal{A}_2}/T_{\mathcal{A}_1}$. $s > 1$ implies algorithm $\mathcal{A}_1$ is faster than $\mathcal{A}_2$.



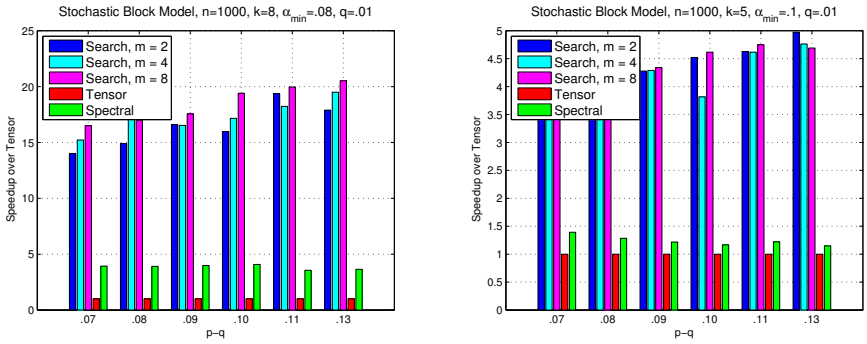
(a) (b)

Figure 1: Labeled Nodes: Comparing the average error performance of Community Search algorithm with Spectral clustering [5] and Tensor decomposition [12] algorithms in a stochastic block model with (a) $n = 1000$, $k = 5$, $\alpha_{min} = .1$, (b) $n = 1000$, $k = 8$, $\alpha_{min} = .08$. The algorithms use m labeled node from target cluster as side information and compute biased weights. The Community Search algorithm outperforms both Spectral clustering and Tensor decomposition.

4.1 Performance and Speedup with Labeled Nodes

First we compare the error performance of Community Search algorithm with Spectral clustering and Tensor decomposition algorithms in the setting where side-information is given in the form of m labeled nodes from the target community. We then compute biased weights w_j using the tree method of Section 3.2 with a radius $r = 2$. Note that this tree method may assign weights in violation of condition (A1) for small target communities, since for small target clusters the number of nodes in the tree from a large cluster may exceed those from the target community, in such cases Algorithm 1 cannot be expected to recover the communities. Therefore we consider a subset of larger communities which assign the correct weights satisfying condition (A1) and evaluate our algorithm over these communities. Figure 1 (a) plots the average error over 3 largest cluster in a stochastic block model (SBM) with $n = 1000$, and $k = 5$ unequal sized communities. The Community Search shows significantly less error than Tensor decomposition and Spectral clustering. In Figure 1 (b) we plot the average over 5 larger cluster in a SBM with $n = 1000$, $k = 8$ unequal communities. Again Community Search shows better error than Spectral clustering and comparable error to Tensor decomposition.

Figure 2 show the speedup performance of the Community Search and Spectral clustering algorithms over Tensor decomposition in this setting with labeled nodes. As indicated earlier, all three algorithms were implemented in Matlab. We observe that the Community Search has a much lower runtime than both Spectral clustering and Tensor decomposition.



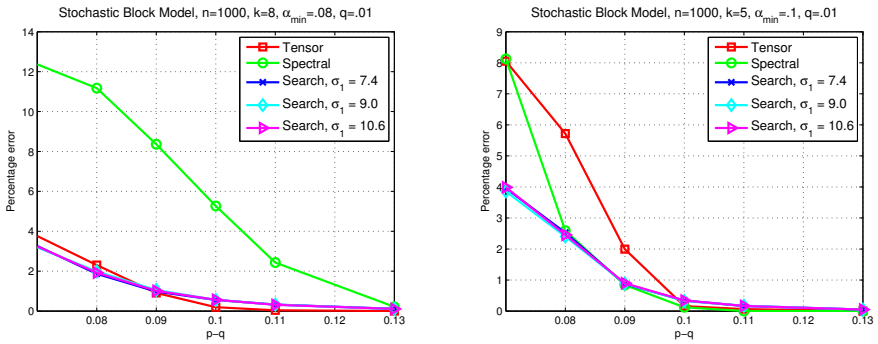
(a) (b)

Figure 2: Labeled Nodes: The average speedup performance of the Community Search and Spectral clustering [5] algorithms with respect to Tensor decomposition [12] in a stochastic block model with (a) $n = 1000$, $k = 8$, $\alpha_{min} = .08$, (b) $n = 1000$, $k = 5$, $\alpha_{min} = .1$, and labeled nodes as side-information. The Community Search algorithm is faster than both Spectral clustering and Tensor decomposition.

4.2 Performance and Speedup with Synthetic Weights

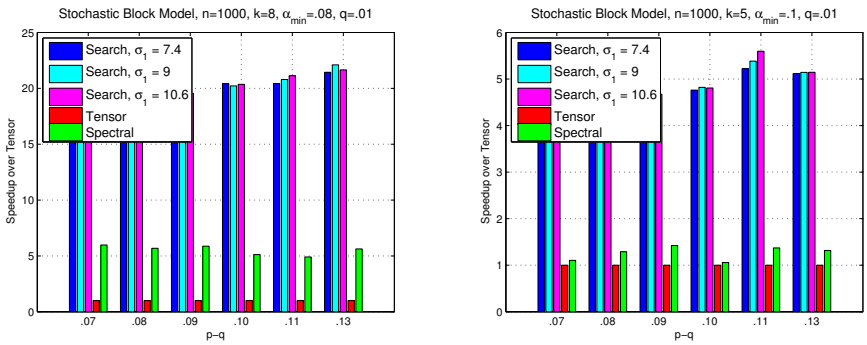
Next we compare the error and runtime performance of all three algorithms in a setting where side-information is available in the form of synthetically generated biased weights (as discussed earlier, three different choices of parameters). Figure 3 (a) plots the average error over all communities in a SBM with $n = 1000$, $k = 8$. The Community Search algorithm has a better performance over Spectral clustering and comparable performance with Tensor decomposition. In Figure 3 (b) plots the average error in a SBM with $n = 1000$, $k = 5$. In this case Community Search outperforms Tensor decomposition and has comparable performance to Spectral clustering.

In Figure 4 we plot the average speedup of Community Search and Spectral clustering over Tensor decomposition. Again the Community Search algorithm is significantly faster than both Spectral clustering and Tensor decomposition. We also observe that the speedup increases with increasing $p - q$.



(a) (b)

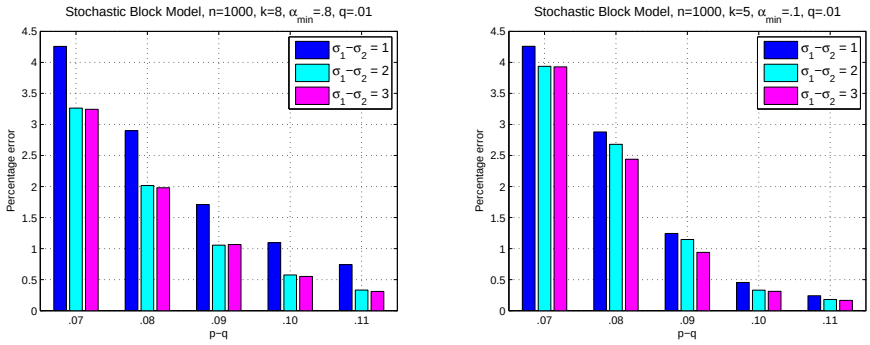
Figure 3: Synthetic Weights: Comparing the average error performance of Community Search algorithm with Spectral clustering [5] and Tensor decomposition [12] algorithms in a stochastic block model with (a) $n = 1000$, $k = 8$, $\alpha_{min} = .08$, (b) $n = 1000$, $k = 5$, $\alpha_{min} = .1$. The algorithms use synthetic weights as side information to search for the target community. Community Search algorithm shows a lower error.



(a) (b)

Figure 4: Synthetic Weights: The average speedup performance of the Community Search and Spectral clustering [5] algorithms with respect to Tensor decomposition [12] in a stochastic block model with (a) $n = 1000$, $k = 8$, $\alpha_{min} = .08$, (b) $n = 1000$, $k = 5$, $\alpha_{min} = .1$, and synthetic weights as side-information. The Community Search algorithm is faster than both Spectral clustering and Tensor decomposition.

To see how the quality of side-information effects the performance of our algorithm we plot the average error with increasing singular value gap $\sigma_1(R) - \sigma_2(R)$ (or the difference between the largest and second largest expected node weight) in Figure 5. In this experiment we fix the synthetic weights ($w_1 = 5, w_2 = 10$) and vary $\sigma_1(R) - \sigma_2(R)$ by changing the probabilities with which the weights appear in each community. Note that the singular value gap increases when one weight appears with greater chance than the other. Therefore $\sigma_1(R) - \sigma_2(R)$ can also be viewed as a measure of quality of side-information. As predicted from our analysis, we observe that the error improves with an increase in the gap $\sigma_1(R) - \sigma_2(R)$.

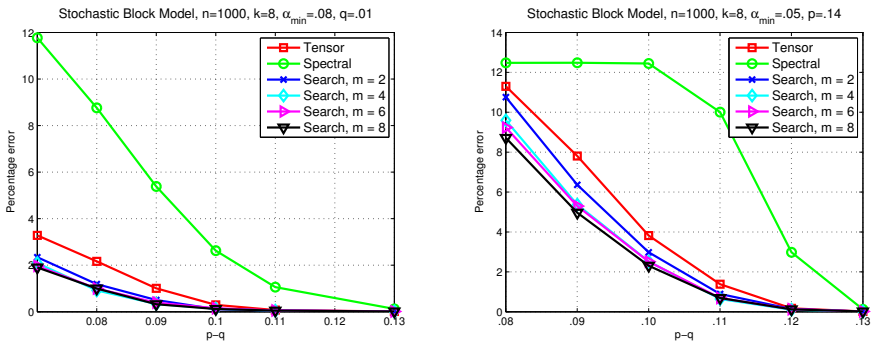


(a) (b)

Figure 5: Sensitivity: The average percentage error of Community Search Algorithm in a stochastic block model with (a) $n = 1000$, $k = 8$, $\alpha_{min} = .08$, (b) $n = 1000$, $k = 5$, $\alpha_{min} = .1$, with increasing singular value gap $\sigma_1(R) - \sigma_2(R)$. As shown in our analysis the performance improves with increase in the singular value gap.

4.3 Parallel Clustering

Finally we consider the semi-supervised graph clustering setting described in Section 3.3 where we are provided with m labeled nodes from each community, and we want to recover all communities. Recall that the Community Search algorithm can also be used as a semi-supervised parallel graph clustering algorithm. Figure 6 plots the cumulative error over all communities with increasing $p - q$ in a SBM with $n = 1000$, $k = 8$, and using different number of labeled nodes. The weights in this case are computed using the tree method and with radius $r = 1$. The Community Search algorithm outperforms both Spectral clustering and Tensor decomposition algorithms in both the experiments.



(a) (b)

Figure 6: Labeled Nodes + Parallel Clustering: Comparing the average error performance of Community Search algorithm with Spectral clustering [5] and Tensor decomposition [12] algorithms in a stochastic block model with $n = 1000$, $k = 8$, $\alpha_{min} = .08$, (a) $q = .01$, (b) $p = .14$. We consider the semi-supervised graph clustering setting when side-information is available in the form of $m \in \{2, 4, 6, 8\}$ labeled node from each community. The Community Search algorithm has a better performance over both Spectral clustering and Tensor decomposition.

	W ($m = 2$)	W ($m = 4$)	W ($m = 6$)	W ($m = 8$)	W ($m = 10$)	T	S
Mean	56	55.64	55.32	55.30	54.98	60	70
Best	55	54	54	53	53	60	70

Table 1: Average and best classification error (number of nodes that are misclassified in each estimated community compared to ground-truth) obtained by Community Search algorithm (W) with Spectral clustering (S) [5] and Tensor decomposition (T) [12] algorithms on US political blogosphere network [[30]]. The Community Search algorithm achieves the best classification error of 53 and better average error over the competing algorithms.

4.4 Results on real dataset

In this section we evaluate the performance of Community Search algorithm on a real world network. For this experiment we consider the *US political blogosphere network* first introduced by [30] where nodes correspond to political blogs classified as either liberal or conservative during 2004 US election, and edges represent hyperlinks between them. We consider the largest connected component of the network having 1222 nodes and $16,716$ edges. This dataset provides the ground-truth labels (liberal or conservative) for each node; these labels were manually generated by authors in [30] according to their content. The largest component has two communities of sizes 586 and 636 according to this ground-truth.

In this semi-supervised graph clustering setting, we randomly choose $m \in \{2, 4, 6, 8, 10\}$ labeled nodes from each ground-truth community as side information. Our performance metric is the classification error, namely, the number of nodes wrongly classified in each estimated community compared to the ground-truth communities[2] ($e = |\{j \in V : j \in V_1, j \notin \hat{V}_1 \text{ or } j \in \hat{V}_1, j \notin V_1\}|$).

For each m we observe the overall performance of the Community Search algorithm over 50 different random choices of labeled nodes. As before, we compare the performance with Tensor decomposition and Spectral clustering algorithms. For the Community Search algorithm we compute weights using the tree method with radius $r = 1$. In Table 1, we show the best and average classification error obtained by the clustering algorithms. With $r = 1$ the Community Search algorithm shows a better classification error than both Tensor and Spectral algorithms. In fact our algorithm achieves the best classification error of 53, which is better than other state–of–the–art algorithms [32],[33] which achieved errors in the range $58 - 60$ on this dataset.

[2]Since there are only two communities, the estimation error in the first community is equal to that in the second; thus, we can count any one of them.

5 Conclusion and Discussion

In this paper we defined the search problem in community detection, provided a simple generic framework for incorporating side information, and a corresponding algorithm to solve the problem. Our algorithm analytically matches the state of the art performance of existing algorithms that do not use side information, and empirically outperforms them on reliability and speed.

More generally, we believe that incorporating side information into graph analysis is a fertile and important area of research, as no real-world problem is a "pure" graph problem (i.e. where the only input is a graph) of the kind studied in e.g. the vast majority of clustering literature.

There are several possible future directions: *(A)* Understanding fundamental limits of community detection [34, 35] when there is non-trivial side information (e.g. $\Theta(\log n)$ of labeled nodes in a community). *(B)* Richer notions of side information, and corresponding problem definitions beyond search. *(C)* From a more practical viewpoint we show in our experimental results (Section 4) that even this simple form of side information can dramatically reduce the computation time for searching communities, and also improve error performance. As discussed in the previous section this work also provides a new method to parallelize graph clustering, an inherently difficult task. Adapting even faster algorithms, e.g. those based on belief-propagation, to this new semi-supervised setting is also an important prospect.

Acknowledgement

We would like to acknowledge support from NSF grants CNS-1320175, 0954059, ARO grants W911NF-15-1-0227, W911NF-14-1-0387, and the US DoT supported D- STOP Tier 1 University Transportation Center.

References

[1] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[2] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys (CSUR)*, 45(4):43, 2013.

[3] A. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001.

[4] F. McSherry. Spectral partitioning of random graphs. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 529–537. IEEE, 2001.

[5] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[6] K. Rohe, S. Chatterjee, and B. Yu. Spectral clustering and the high-dimensional stochastic blockmodel. *The Annals of Statistics*, 39(4):1878–1915, 2011.

[7] K. Chaudhuri, F. Chung, and A. Tsiatas. Spectral clustering of graphs with general degrees in the extended planted partition model. *Journal of Machine Learning Research*, 2012:1–23, 2012.

[8] S. Yun and A. Proutiere. Accurate community detection in the stochastic block model via spectral algorithms. *arXiv preprint arXiv:1412.7335*, 2014.

[9] Y. Chen, S. Sanghavi, and H. Xu. Clustering sparse graphs. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[10] N. Ailon, Y. Chen, and H. Xu. Breaking the small cluster barrier of graph clustering. *arXiv preprint arXiv:1302.4549*, 2013.

[11] E. Abbe, A. S. Bandeira, and G. Hall. Exact recovery in the stochastic block model. *arXiv preprint arXiv:1405.3267*, 2014.

[12] A. Anandkumar, R. Ge, D. Hsu, and S. M. Kakade. A tensor spectral approach to learning mixed membership community models, 2013. http://arxiv.org/abs/1302.2684.

[13] F. Huang, UN. Niranjan, M. Hakeem, and A. Anandkumar. Fast detection of overlapping communities via online tensor methods, 2013. http://arxiv.org/abs/1309.0787.

[14] X. Zhu. Semi-supervised learning literature survey, 2005. `http://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf`.

[15] B. Kulis, S. Basu, I. Dhillon, and R. Mooney. Semi-supervised graph clustering: a kernel approach. *Machine learning*, 74(1):1–22, 2009.

[16] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16(16):321–328, 2004.

[17] Y. Fujiwara and G. Irie. Efficient label propagation. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 784–792, 2014.

[18] Wikipedia: `https://en.wikipedia.org/wiki/Method_of_moments_(statistics)`.

[19] D. Hsu and S. M. Kakade. Learning mixtures of spherical gaussians: moment methods and spectral decompositions. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 11–20. ACM, 2013.

[20] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

[21] A. Anandkumar, Y. Liu, D. J. Hsu, D. P. Foster, and S. M. Kakade. A spectral algorithm for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 917–925, 2012.

[22] J. T. Chang. Full reconstruction of markov models on evolutionary trees: identifiability and consistency. *Mathematical biosciences*, 137(1):51–73, 1996.

[23] J. D. Mcauliffe and D. M. Blei. Supervised topic models. In *Advances in neural information processing systems*, pages 121–128, 2008.

[24] Y. Lu and C. Zhai. Opinion integration through semi-supervised topic modeling. In *Proceedings of the 17th International Conference on World Wide Web*, pages 121–130. ACM, 2008.

[25] N. Rao, H. F. Yu, P. Ravikumar, and I. Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *Advances in neural information processing systems*, 2015.

[26] J. Yang, J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In *Data Mining (ICDM), 2013 IEEE 13th international conference on*, pages 1151–1156. IEEE, 2013.

[27] A. Ray, S. Sanghavi, and S. Shakkottai. Searching for a single community in a graph. In *Proceedings of the ACM Sigmetrics 2016 (poster paper)*. ACM, 2016.

[28] N. Agarwal, A. S. Bandeira, K. Koiliaris, and A. Kolla. Multisection in the stochastic block model using semidefinite programming. *arXiv preprint arXiv:1507.02323*, 2015.

[29] E. Abbe and C. Sandon. Community detection in general stochastic block models: Fundamental limits and efficient algorithms for recovery. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 670–688. IEEE, 2015.

[30] L. A. Adamic and N. Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005.

[31] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[32] J. Jin. Fast community detection by score. *The Annals of Statistics*, 43(1):57–89, 2015.

[33] C. Gao, Z. Ma, A. Y. Zhang, and H. H. Zhou. Achieving optimal misclassification proportion in stochastic block model. *arXiv preprint arXiv:1505.03772*, 2015.

[34] E. Mossel, J. Neeman, and A. Sly. Reconstruction and estimation in the planted partition model. *Probability Theory and Related Fields*, pages 1–31, 2014.

[35] A. Montanari. Finding one community in a sparse graph. *arXiv preprint arXiv:1502.05680*, 2015.

[36] P. Wedin. Perturbation bounds in connection with singular value decomposition. *BIT Numerical Mathematics*, 12(1):99–111, 1972.

[37] Yi Yu, Tengyao Wang, and Richard J Samworth. A useful variant of the davis–kahan theorem for statisticians. *Biometrika*, 102(2):315–323, 2015.

[38] J. Tropp. An introduction to matrix concentration inequalities. *arXiv preprint arXiv:1501.01571*, 2015.

A Community Search: Proofs

In this section we provide the proof details of Theorem 1, Theorem 3 and the relevant Lemmas.

A.1 Community Search: Perturbation Analysis

Let the expectation of the estimates $\hat{m}_1$, $\hat{A}_1$, $\hat{A}_2$ and $\hat{B}$ be represented by m_1, A_1, A_2, B respectively. Let $n_i = |P_i|$ be the size of partition P_i. For a matrix M, $\|M\|$ denotes its spectral norm. Recall that,

$$\begin{aligned} A_1 &= \frac{1}{\sqrt{n_3}} E[X_{P_1,P_3}] \ , \ A_2 = \frac{1}{\sqrt{n_3}} E[X_{P_2,P_3}] \\ m_1 &= \sum_{i=1}^{k} \alpha_i \mu_{P_1,i} \ , \ B = \sum_{i=1}^{k} \alpha_i \omega_i \mu_{P_2,i}^T \end{aligned}$$

where $\omega_i = E[w_j | j \in V_i]$. Let the rank k-svd of A_1, A_2 be given by $A_1 = U_1 D_1 V_1^T$, $A_2 = U_2 D_2 V_2^T$, and for the estimates $\hat{A}_1 = \hat{U}_1 \hat{D}_1 \hat{V}_1^T$, $\hat{A}_2 = \hat{U}_2 \hat{D}_2 \hat{V}_2^T$.

Lemma 4. *Let* $\max\{\|\hat{A}_1 - A_1\|, \|\hat{A}_2 - A_2\|\} \le \epsilon_2$ *and* $\epsilon_2 < \min\{\sigma_k(A_1), \sigma_k(A_2)\}/12$. *Let* $\hat{W}_1 = \hat{U}_1 \hat{D}_1^{-1}, \hat{W}_2 = \hat{U}_2 \hat{D}_2^{-1}$ *be the whitening matrices. Then,*

$$\begin{aligned} \|I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{1/2}\| &\le \frac{6\epsilon_2}{\sigma_k(A_1)} \\ \|I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1/2}\| &\le \frac{12\epsilon_2}{\sigma_k(A_1)} \\ \|I_k - (\hat{W}_2^T A_2 A_2^T \hat{W}_2)^{-1/2}\| &\le \frac{12\epsilon_2}{\sigma_k(A_2)} \end{aligned}$$

Proof. We prove this along the lines in [19]. The matrix $\hat{W}_1$ whitens $\hat{A}_1 \hat{A}_1^T$ since,

$$\hat{W}_1^T \hat{A}_1 \hat{A}_1^T \hat{W}_1 = \hat{D}_1^{-1} \hat{U}_1^T \hat{A}_1 \hat{A}_1^T \hat{U}_1 \hat{D}_1^{-1} = I_k$$

Similarly $\hat{W}_2$ whitens $\hat{A}_2 \hat{A}_2^T$.
Also note $\epsilon_2 < \sigma_k(A_1)/2$, hence using Weyl's inequality $\sigma_k(\hat{A}_1) \ge \sigma_k(A_1)/2$. This implies

$$
\begin{aligned}
\|I_k - \hat{W}_1^T A_1 A_1^T \hat{W}_1\| &= \|\hat{W}_1^T(\hat{A}_1\hat{A}_1^T - A_1A_1^T)\hat{W}_1\| \\
&\leq \|\hat{W}_1^T\hat{A}_1(\hat{A}_1^T - A_1^T)\hat{W}_1\| + \|\hat{W}_1^T(\hat{A}_1 - A_1)A_1^T\hat{W}_1\| \\
&\leq \|\hat{W}_1^T\hat{A}_1\|\|\hat{A}_1^T - A_1^T\|\|\hat{W}_1\| + \|\hat{W}_1^T\|\|\hat{A}_1 - A_1\|\|A_1^T\hat{W}_1\| \\
&< \frac{2\epsilon_2}{\sigma_k(A_1)} + \frac{2\epsilon_2}{\sigma_k(A_1)}\left(\|\hat{A}_1^T\hat{W}_1\| + \|(A_1^T - \hat{A}_1^T)\hat{W}_1\|\right) \\
&< \frac{2\epsilon_2}{\sigma_k(A_1)} + \frac{2\epsilon_2}{\sigma_k(A_1)}\left(1 + \frac{2\epsilon_2}{\sigma_k(A_1)}\right) \\
&\leq \frac{6\epsilon_2}{\sigma_k(A_1)}
\end{aligned}
$$

Therefore all eigenvalues of the matrix $\hat{W}_1^T A_1 A_1^T \hat{W}_1$ lie in the interval $\left(1 - \frac{6\epsilon_2}{\sigma_k(A_1)}, 1 + \frac{6\epsilon_2}{\sigma_k(A_1)}\right)$. This implies the eigenvalues of $(\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{1/2}$ also lie in the same interval and that of $(\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1}$ lie in the interval $(1/(1 + 6\epsilon_2/\sigma_k(A_1)), 1/(1 - 6\epsilon_2/\sigma_k(A_1)))$. The first bound follows directly. To show the second bound we compute,

$$
\begin{aligned}
(I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1/2})(I_k + (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1/2}) &= I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1} \\
I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1/2} &= \left(I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1}\right) \times \\
&\quad (I_k + (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1/2})^{-1} \\
\|I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1/2}\| &\leq \|I_k - (\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1}\| \\
&\leq \frac{1}{1 - 6\epsilon_2/\sigma_k(A_1)} - 1 \\
&\leq \frac{12\epsilon_2}{\sigma_k(A_1)}
\end{aligned}
$$

Similarly we can show the second bound using $\hat{W}_2$ and A_2. □

Lemma 5. *Let $\|\hat{R} - R\| \leq \delta < \sigma_2(R)/2$. u_1 be a left singular vector of R corresponding to the largest singular value and $\hat{u}_1$ be that of $\hat{R}$. Then,*

$$
\|\hat{u}_1 - u_1\| \leq \frac{8\delta}{(\sigma_1(R) - \sigma_2(R))}
$$

Proof. The result follows from the generalized sin–θ theorem by [36]. In particular we use an useful version of it from [37] [Theorem 4]. We get,

$$
\begin{aligned}
\|\hat{u}_1 - u_1\| &\leq \frac{2^{3/2}(2\sigma_1(R) + \|\hat{R} - R\|)\|\hat{R} - R\|}{\sigma_1(R)^2 - \sigma_2(R)^2} \\
&\leq \frac{2^{3/2}(2\sigma_1(R) + 2\sigma_2(R))\|\hat{R} - R\|}{(\sigma_1(R) - \sigma_2(R))(\sigma_1(R) + \sigma_2(R))} \\
&\leq \frac{8\delta}{(\sigma_1(R) - \sigma_2(R))}
\end{aligned}
$$

□

Lemma 6. *Assume $\|\hat{u}_1 - u_1\| \leq \eta_1$, $\|\hat{A}_1 - A_1\| \leq \epsilon_2$. Let $\hat{z} = \hat{U}_1\hat{D}_1\hat{u}_1$. z be given by the equation $u_1 = W_1^T z$, where $W_1 = \hat{W}_1(\hat{W}_1^T A_1 A_1^T \hat{W}_1)^{-1/2}$. Then,*

$$
\|\hat{z} - z\| \leq 2\sigma_1(A_1)\eta_1 + \frac{16\sigma_1(A_1)\epsilon_2}{\sigma_k(A_1)}
$$

Proof. First using Wedin's theorem [36] we get,

$$\|\hat{U}_1\hat{U}_1^T - U_1U_1^T\| \leq \frac{4\epsilon_2}{\sigma_k(A_1)} \tag{1}$$

We can bound $\hat{z} - z$ as follows.

$$\begin{aligned}
\|\hat{z} - z\| &= \|\hat{z} - U_1U_1^Tz\| \leq \|\hat{z} - \hat{U}_1\hat{U}_1^Tz\| + \|\hat{U}_1\hat{U}_1^T - U_1U_1^T\|\|z\| \\
&\overset{(a)}{\leq} \|\hat{z} - \hat{U}_1\hat{U}_1^Tz\| + \frac{4\epsilon_2\|z\|}{\sigma_k(A_1)} \\
&\overset{(b)}{=} \|\hat{z} - \hat{U}_1\hat{U}_1^Tz\| + \frac{4\epsilon_2\|U_1D_1u'\|}{\sigma_k(A_1)} \qquad (2) \\
&\leq \|\hat{z} - \hat{U}_1\hat{U}_1^Tz\| + \frac{4\sigma_1(A_1)\epsilon_2}{\sigma_k(A_1)} \qquad (3)
\end{aligned}$$

The step (a) uses equation 1, and step (b) uses the fact that the matrix $D_1^{-1}U_1^T$ also whitens $A_1A_1^T$, therefore z can also be expressed as $z = U_1D_1u'$ for some unit vector u'. Since $u_1 = W_1^Tz$, we can write also $\hat{D}_1(\hat{W}_1^TA_1A_1^T\hat{W}_1)^{1/2}u_1 = \hat{U}_1^Tz$. Now we bound the first term.

$$\begin{aligned}
\|\hat{z} - \hat{U}_1\hat{U}_1^Tz\| &= \|\hat{U}_1\hat{D}_1\hat{u}_1 - \hat{U}_1\hat{U}_1^Tz\| \\
&= \|\hat{U}_1\hat{D}_1(\hat{u}_1 - u_1) + \hat{U}_1\hat{D}_1u_1 - \hat{U}_1\hat{U}_1^Tz\| \\
&\leq \|\hat{D}_1\|\|\hat{u}_1 - u_1\| + \|\hat{U}_1\hat{D}_1u_1 - \hat{U}_1\hat{U}_1z\| \\
&\leq 2\sigma_1(A_1)\eta_1 + \|\hat{U}_1\hat{D}_1u_1 - \hat{U}_1\hat{D}_1(\hat{W}_1^TA_1A_1^T\hat{W}_1)^{1/2}u_1\| \\
&\leq 2\sigma_1(A_1)\eta_1 + \|\hat{U}_1\hat{D}_1(I_k - (\hat{W}_1^TA_1A_1^T\hat{W}_1)^{1/2})\|\|u_1\| \\
&\leq 2\sigma_1(A_1)\eta_1 + \|\hat{D}_1\|\|I_k - (\hat{W}_1^TA_1A_1^T\hat{W}_1)^{1/2}\| \\
&\leq 2\sigma_1(A_1)\eta_1 + \frac{12\sigma_1(A_1)\epsilon_2}{\sigma_k(A_1)} \qquad (4)
\end{aligned}$$

where the last inequality follows from Lemma 4. Combining the above with equation 3 we get,

$$\|\hat{z} - z\| \leq 2\sigma_1(A_1)\eta_1 + \frac{12\sigma_1(A_1)\epsilon_2}{\sigma_k(A_1)} + \frac{4\sigma_1(A_1)\epsilon_2}{\sigma_k(A_1)} = 2\sigma_1(A_1)\eta_1 + \frac{16\sigma_1(A_1)\epsilon_2}{\sigma_k(A_1)}$$

□

A.2 Community Search: Concentration

In this section using concentration bounds we compute the parameter range of p, q, k, α_i for which the Community Search algorithm can recover the particular community membership vector μ_1 with high probability. For ease of exposition for this section we assume the partitions P_1, P_2, P_3, P_4 are of equal size. Therefore $n_1 = n_2 = n_3 = n_4 = \frac{n}{4}$. However the results easily generalize to any random unbiased split. Now we restate the Matrix Bernstein inequality [38] and then use it to bound the perturbation of the estimates $\hat{A}_1, \hat{A}_2$.

Theorem 7 (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 $\sigma^2 = \max\{\|E[ZZ^T]\|, \|E[Z^TZ]\|\}$. Then for all $t \geq 0$,*

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

Lemma 8 (Concentration of $\hat{A}_1, \hat{A}_2$)**.** *Let $\hat{A}_1, \hat{A}_2$ be as given in Algorithm 1. Then,*

$$\begin{aligned}\|\hat{A}_1 - A_1\| &= O\left(\sqrt{p\log\frac{(n_1+n_3)}{\delta}}\right)\\ \|\hat{A}_2 - A_2\| &= O\left(\sqrt{p\log\frac{(n_2+n_3)}{\delta}}\right)\end{aligned}$$

with probability greater than $1-2\delta$.

Proof. Note that we can write $\hat{A}_1 = \frac{1}{\sqrt{n_3}}\sum_{j\in P_3} X_{P_1,j}e_j^T$, where e_j is the unit vector with 1 in the j-th coordinate. Then $Z = \hat{A}_1 - A_1 = \frac{1}{\sqrt{n_3}}\sum_{j\in P_3}(X_{P_1,j} - \mu_{P_1,c_j})e_j^T = \sum_{j\in P_3} Z_j$, c_j being the cluster of j-th node. Then,

$$\begin{aligned}\|E[ZZ^T]\| &= \|\sum_{j\in P_3} E[Z_jZ_i^T]\| \le \sum_{j\in P_3}\|E[Z_jZ_j^T]\|\\ &= \sum_{j\in P_3}\frac{1}{n_3}\|E[(X_{P_1,j}-\mu_{P_1,c_j})(X_{P_1,j}-\mu_{P_1,c_j})^T]\|\\ &\le \sum_{j\in P_3}\frac{1}{n_3}p(1-p) \le p\end{aligned}$$

Also,

$$\|E[Z^TZ]\| = \|\sum_{j\in P_3} E[Z_j^TZ_j]\| = \|\sum_{j\in P_3}\frac{1}{n_3}E[\|X_{P_1,j}-\mu_{P_1,c_j}\|^2 e_je_j^T]\| \le \frac{n_1p}{n_3}$$

Assuming $n_1 = n_3$ we have the variance term bounded by $\sigma^2 = \max\{\|E[ZZ^T]\|, \|E[ZZ^T]\|\} = p$. Now with high probability $\|Z_j\| = \frac{1}{n_3}\|(X_{P_1,j}-\mu_{P_1,c_j})e_j^T\| \le \sqrt{2p} := L$. Therefore by applying from Matrix-Bernstein inequality with probability greater than $1-\delta$,

$$\|\hat{A}_1 - A_1\| \le 2\sigma\sqrt{\log\frac{(n_1+n_3)}{\delta}} = O\left(\sqrt{p\log\frac{(n_1+n_3)}{\delta}}\right)$$

Similarly we find the second bound for $\|\hat{A}_2 - A_2\|$. □

Lemma 9 (Whitening matrix concentration)**.** *Assume that* $\max\{\|\hat{A}_1 - A_1\|, \|\hat{A}_2 - A_2\|\} < \epsilon_2$, *and* $\epsilon_2 < \min\{\sigma_k(A_1), \sigma_k(A_2)\}/4$. *Let* $\hat{W}_1 = \hat{U}_1\hat{D}_1^{-1}, \hat{W}_2 = \hat{U}_2\hat{D}_2^{-1}$ *be the whitening matrices. Define* $W_1 := \hat{W}_1(\hat{W}_1^TA_1A_1^T\hat{W}_1)^{-1/2}$ *and* $W_2 := \hat{W}_2(\hat{W}_2^TA_2A_2^T\hat{W}_2)^{-1/2}$. *Then,*

$$\begin{aligned}\|\hat{W}_1 - W_1\| &= O\left(\frac{\sqrt{p\log n_1}}{\alpha_{min}^2 n_1(p-q)^2}\right)\\ \|\hat{W}_2 - W_2\| &= O\left(\frac{\sqrt{p\log n_2}}{\alpha_{min}^2 n_2(p-q)^2}\right)\end{aligned}$$

Proof. First note that the matrix W_1 whitens the matrix $A_1A_1^T$ since,

$$W_1^TA_1A_1^TW_1 = (\hat{W}_1^TA_1A_1^T\hat{W}_1)^{-1/2}\hat{W}_1^TAA^T\hat{W}_1(\hat{W}_1^TA_1A_1^T\hat{W}_1)^{-1/2} = I_k$$

Similarly W_2 whitens matrix $A_2A_2^T$. We can bound the perturbation as follows.

$$\begin{aligned}\|\hat{W}_1 - W_1\| &= \|\hat{W}_1(I_k - (\hat{W}_1^TA_1A_1^T\hat{W})_1^{-1/2})\|\\ &\le \|\hat{W}_1\|\|I_k - (\hat{W}_1^TA_1A_1^T\hat{W}_1)^{-1/2}\|\\ &\le \frac{2}{\sigma_k(A)}\times\frac{12\epsilon_2}{\sigma_k(A_1)} = \frac{24\epsilon_2}{\sigma_k(A_1)^2}\end{aligned}$$

where the last inequality follows from Lemma 4. Now from Lemma 8 we have $\epsilon_2 = O(\sqrt{p \log n_1})$. Also observe that $\sigma_k(A_1) = \Omega(\alpha_{min}\sqrt{n_1}(p-q))$. Using these in the above bound we get

$$\|\hat{W}_1 - W_1\| = O\left(\frac{\sqrt{p \log n_1}}{\alpha_{min}^2 n_1 (p-q)^2}\right)$$

The second bound for $\|\hat{W}_2 - W_2\|$ follows. □

Lemma 10 (R matrix concentration)**.** *Let $\hat{R} = \hat{W}_1^T \hat{B} \hat{W}_2$ and $R = W_1^T B W_2$ then,*

$$\|\hat{R} - R\| = \tilde{O}\left(\max\left\{\frac{\alpha_{max}^2 p^{2.5}\gamma_1}{\alpha_{min}^3\sqrt{n}(p-q)^3}, \frac{\alpha_{max}^2 p^2 \gamma_2}{\alpha_{min}^2 (p-q)^2}\right\}\right)$$

where $\gamma_1 = \max_{j \in P_4} |\hat{w}_j|$, and $\gamma_2 = \max_{j \in P_4} |\hat{w}_j - \bar{w}_j|$.

Proof. Let $c_j \in [k]$ denote the community for the j-th node. We can upper bound the estimation error in R matrix as follows.

$$\begin{aligned}
\|\hat{R} - R\| &= \|\hat{W}_1^T \hat{B} \hat{W}_2 - W_1^T B W_2\| = \frac{1}{n_4} \| \sum_{j \in P_4} (\hat{w}_j \hat{W}_1^T X_{P_1,j} X_{P_2,j}^T \hat{W}_2 - \bar{w}_j W_1^T \mu_{P_1,c_j} \mu_{P_2,c_j}^T W_2)\| \\
&\leq T_1 + T_2 + T_3 + T_4 + T_5
\end{aligned}$$

where,

$$\begin{aligned}
T_1 &= \|\frac{1}{n_4} \sum_{j \in P_4} \hat{w}_j \hat{W}_1^T (X_{P_1,j} - \mu_{P_1,c_j}) X_{P_2,j}^T \hat{W}_2\| = \|\hat{W}_1^T (\hat{A}_1 - A_1) diag(\hat{w}_1, ..., \hat{w}_{n_4}) \hat{A}_2^T \hat{W}_2\| \\
T_2 &= \|\frac{1}{n_4} \sum_{j \in P_4} \hat{w}_j \hat{W}_1^T \mu_{P_1,c_j} (X_{P_2,j} - \mu_{P_2,c_j})^T \hat{W}_2\| = \|\hat{W}_1^T A_1 diag(\hat{w}_1, ..., \hat{w}_{n_4}) (\hat{A}_2 - A_2)^T \hat{W}_2\| \\
T_3 &= \|\frac{1}{n_4} \sum_{j \in P_4} \hat{w}_j (\hat{W}_1 - W_1)^T \mu_{P_1,c_j} \mu_{P_2,c_j}^T \hat{W}_2\| = \|(\hat{W}_1 - W_1)^T A_1 diag(\hat{w}_1, ..., \hat{w}_{n_4}) A_2^T \hat{W}_2\| \\
T_4 &= \|\frac{1}{n_4} \sum_{j \in P_4} \hat{w}_j W_1 \mu_{P_1,c_j} \mu_{P_2,c_j}^T (\hat{W}_2 - W_2)\| = \|W_1^T A_1 diag(\hat{w}_1, ..., \hat{w}_{n_4}) A_2^T (\hat{W}_2 - W_2)\| \\
T_5 &= \|\frac{1}{n_4} \sum_{j \in P_4} (\hat{w}_j - \bar{w}_j) W_1 \mu_{P_1,c_j} \mu_{P_2,c_j}^T W_2\| = \|W_1^T A_1 diag(\hat{w}_1 - \bar{w}_1, ..., \hat{w}_{n_4} - \bar{w}_{n_4}) A_2^T W_2\|
\end{aligned}$$

Let $\gamma_1 = \max_{j \in P_4} |\hat{w}_j|$, and $\gamma_2 = \max_{j \in P_4} |\hat{w}_j - \bar{w}_j|$. Then using Lemmas 8 and 9 we get $T_1 = T_2 = \tilde{O}\left(\frac{\alpha_{max} p^{1.5}\gamma_1}{\alpha_{min}^2\sqrt{n}(p-q)^2}\right)$, $T_3 = T_4 = \tilde{O}\left(\frac{\alpha_{max}^2 p^{2.5}\gamma_1}{\alpha_{min}^3\sqrt{n}(p-q)^3}\right)$, and $T_5 = \tilde{O}\left(\frac{\alpha_{max}^2 p^2 \gamma_2}{\alpha_{min}^2 (p-q)^2}\right)$. The dominating term is given by the maximum of T_3, T_4 and T_5. □

Lemma 11 (Thresholding)**.** *Let $e = \hat{z} - z$, threshold $\tau = \sqrt{\alpha_1}(p+q)/2$. Let $E = \{i \in V : i \in \hat{V}_1, i \notin V_1$ or $i \in V_1, i \notin \hat{V}_1\}$ be the set of erroneous nodes after thresholding. If $\|e\| = o\left(\alpha_1\sqrt{n}(p-q)\right)$ then $|E| = o(\alpha_1 n)$.*

Proof. We prove this along similar lines in [12]. Note that $z = \sqrt{\alpha_1}\mu_1$ is a vector with all coordinates either $\sqrt{\alpha_1}p$ or $\sqrt{\alpha_1}q$. Since the threshold is $\tau = \sqrt{\alpha_1}(p+q)/2$, this implies error in any node $i \in E$ is caused when the magnitude error in the corresponding coordinate $\hat{z}_i$ is at least $\sqrt{\alpha_1}(p-q)/2$. Let e_i denote the magnitude error in the i-th coordinate. Then,

$$\begin{aligned}
\|e\|^2 &= \sum_{i \in E} e_i^2 + \sum_{i \in V \setminus E} e_i^2 \\
\|e\|^2 &\geq \sum_{i \in E} e_i^2 \geq |E| \alpha_1 (p-q)^2/4
\end{aligned}$$

Now since $\|e\|$ is upper bounded as $\|e\| = o\left(\alpha_1\sqrt{n}(p-q)\right)$ then it follows that the number of error is bounded by $|E| = o(\alpha_1 n)$. □

Proof of Theorem 1:

Proof. We observe that the vector z in Algorithm 2 is simply a scalar multiple of the partial community membership vector estimate $\hat{\mu}_{P_1}$. Therefore we can also recover community 1 subset V_{P_1} by directly thresholding z using a threshold $\tau' = \sqrt{\alpha_1}(p+q)/2$. In other words the threshold τ required in Algorithm 1 is simply τ'/a, a as defined in Algorithm 2. Therefore it is sufficient show that the estimated vector $\hat{z}$ is close enough to the true vector z and use Lemma 11 to guarantee we can estimate the community with only $o(1)$ fraction error.

In Lemma 10 note that the maximum weight $\gamma_1 \leq \sigma_1(R)+\gamma_2$. Then using conditions (A2) and (A3) in Lemma 10 we get with high probability,

$$\|\hat{R} - R\| \leq C_1(\sigma_1(R) - \sigma_2(R))\frac{\alpha_{min}^2(p-q)^2}{\alpha_{max}^2 p^2 \xi(n)} \tag{5}$$

for some constant C_1. Therefore from Lemma 5 and equation 5 we get,

$$\|\hat{u} - u\| \leq \frac{8\|\hat{R} - R\|}{(\sigma_1(R) - \sigma_2(R))} =\leq 8C_1\frac{\alpha_{min}^2(p-q)^2}{\alpha_{max}^2 p^2 \xi(n)} := \eta_1 \tag{6}$$

From Lemma 8 we get $\epsilon_2 = O(\sqrt{p\log n})$. Also note that $\|z\| = O(\sqrt{\alpha_1 n}p)$, $\sigma_1(A_1) = O(\alpha_{max}\sqrt{n}p)$, and $\sigma_k(A_1) = \Omega(\alpha_{min}\sqrt{n}(p-q))$. Now using the above bound η_1 in Lemma 6 we bound $\|\hat{z} - z\|$ as follows.

$$\begin{aligned}
\|\hat{z} - z\| &\leq 2\sigma_1(A_1)\eta_1 + \frac{16\sigma_1(A_1)\epsilon_2}{\sigma_k(A_1)} \\
&\leq 2\alpha_{max}\sqrt{n}p \times 8C_1\frac{\alpha_{min}^2(p-q)^2}{\alpha_{max}^2 p^2\xi(n)} + 16C_2\frac{\alpha_{max}\sqrt{n}p\sqrt{p\log n}}{\alpha_{min}\sqrt{n}(p-q)} \\
&= 16C_1\frac{\alpha_{min}^2\sqrt{n}(p-q)^2}{\alpha_{max}p\xi(n)} + 16C_2\frac{\alpha_{max}p^{1.5}\sqrt{\log n}}{\alpha_{min}(p-q)} \\
&\overset{(a)}{\leq} C_3\alpha_{min}\frac{\sqrt{n}(p-q)}{\sqrt{\xi(n)}} \\
&= o\left(\alpha_1\sqrt{n}(p-q)\right)
\end{aligned} \tag{7}$$

where C_1, C_2, C_3 are constants. Step (a) follows from condition (A3). Now applying Lemma 11 for the partition P_1 it follows that by thresholding $\hat{z}$ using threshold $\tau = \sqrt{\alpha_1}(p+q)/2$ the number of erroneous nodes in $\hat{V}_{P_1}$ is bounded as $|E| = o(\alpha_1 n)$. Similarly with high probability this holds for partitions P_2, P_3 and P_4 as well. Therefore we can recover community V_1 with $o(1)$ fraction error with high probability. □

Proof of Theorem 2:

Proof. Under conditions (A1)-(A3) using Theorem 1 we can guarantee that the estimated community $\hat{V}_{P_1}$ has at most $o(\alpha_1 n)$ erroneous nodes. Now consider the following degree thresholding step. For any $j \in P_2$ $d_j(\hat{V}_{P_1})$ be the number of edges j share with nodes in $\hat{V}_{P_1}$. Define $\hat{V}_{P_2} := \{j \in P_2 : d_j(\hat{V}_{P_1}) \geq \tau''\}$, for $\tau'' = |V_1 \cap P_1|(p+q)/2$. Then we claim that $\hat{V}_{P_2} = V_1 \cap P_2$ with high probability.

Note that the edges between P_1 and P_2 are not used in Algorithm 1. Let $v_1 = |\hat{V}_{P_1} \cap V_1|$ be the number of correct nodes, and $e_1 = |\hat{V}_{P_1} \cap V_1^c|$ be the number of erroneous nodes in $\hat{V}_{P_1}$. Theorem 1 asserts 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 for any $j \in V_1 \cap P_2$ using Chernoff bound we get with high probability

$$\begin{aligned}
d_j(\hat{V}_{P_1}) &\geq v_1 p - \sqrt{v_1 p\log n} + e_1 q - \sqrt{e_1 q\log n} \\
&\geq v_0 p - e_1 p - \sqrt{v_1 p\log n} + e_1 q - \sqrt{e_1 q\log n} \\
&= v_0\frac{(p+q)}{2} + \left[v_0\frac{(p-q)}{2} - e_1(p-q) - \sqrt{v_1 p\log n} - \sqrt{e_1 q\log n}\right] \\
&\geq v_0\frac{(p+q)}{2}
\end{aligned}$$

where the last step follows since $e_1 = o(\alpha_1 n)$ and using condition (A3). Similarly for any node $j \in V_1^c \cap P_2$ using Chernoff bound we can get with high probability

$$\begin{aligned} d_j(\hat{V}_{P_1}) &\leq v_1 q + \sqrt{v_1 q \log n} + e_1 p + \sqrt{e_1 p \log n} \\ &\leq v_0 q + \sqrt{v_1 q \log n} + e_1 p + \sqrt{e_1 p \log n} \\ &= v_0 \frac{(p+q)}{2} - \left[v_0 \frac{(p-q)}{2} - \sqrt{v_1 q \log n} - e_1 p - \sqrt{e_1 p \log n} \right] \\ &< v_0 \frac{(p+q)}{2} \end{aligned}$$

again using the fact $e_1 = o(\alpha_1 n)$ and using condition (A3). Therefore using a threshold $v_0 \frac{(p+q)}{2}$ we can correctly recover all nodes in $V_1 \cap P_2$. Now by rotating the partitions and repeatedly applying Algorithm 1 + degree thresholding we can correctly recover community V_1 with high probability. This concludes the proof. □

A.3 Recovery via Labeled Nodes

Proof of Theorem 3:

Proof. First note that edges between partitions P_1 and P_2 are not used in Algorithm 1. Therefore the weights can be computed using these edges so that they are independent of the remaining algorithm. For this proof we assume $\mathcal{L}$ to be the set of labeled nodes in partition P_2. For each node $i \in P_1$ we consider a tree of radius r in this partition with i as root, then count the number of edges from the leaves of this tree and labeled node set $\mathcal{L}$ in partition P_2. Let $T_i(r)$ be the subgraph of all nodes at a distance less than or equal to r from node i. When $p = \Theta\left(\frac{\log n}{n^\epsilon}\right)$, $q = \Theta\left(\frac{\log n}{n^\epsilon}\right)$ applying Chernoff bound it is easy to see as long as $|T_i(r)| = o\left(\frac{n^\epsilon}{\log n}\right)$ then with high probability $T_i(r)$ is a tree. With $L = \Omega(n^{\epsilon/2}\sqrt{\log n})$, and for $r \leq \frac{2\log(n^\epsilon/L)}{\log np} = \frac{2\epsilon \log n - 2\log L}{(1-\epsilon)\log n + O(\log\log n)}$ this holds. Now starting from a node in community i let $f_j^i(t)$ be the number of nodes in community j at a distance t from the root node. Since $f_j^i(t) < |T_i(r)|$ this implies $f_j^i(t) = o(\alpha n)$. Now consider the following set of k recursive equations.

$$\bar{f}_j^i(t) = \alpha n p \bar{f}_j^i(t-1) + \sum_{l \neq j} \alpha n q \bar{f}_l^i(t-1) \tag{8}$$

for $j \in [k]$. Since the number of nodes in community l at distance t which are neighbors of $f_j^i(t-1)$ nodes in community j at distance $t-1$ are binomially distributed with probability p if $l = j$ or probability q otherwise; we can use Chernoff bound to see that with high probability the actual number of nodes $f_j^i(t)$ can be expressed as

$$f_j^i(t) = \bar{f}_j^i(t) + o(\bar{f}_j^i(t))$$

Now the initial condition in the recursive equation (8) is given by $\bar{f}_j^i(0) = 1$ when $j = i$, $\bar{f}_j^i(0) = 0$ otherwise. We will prove the theorem in three steps.

Claim 1: *$\bar{f}_i^i(t) > \bar{f}_j^i(t)$ for all t and $j \neq i$*

We prove this by induction. From the initial conditions $\bar{f}_i^i(0) = 1 > 0 = \bar{f}_j^i(0)$, so the claim holds for $t = 0$. Assume it holds for $t-1$. Then for all $j \neq i$,

$$\begin{aligned} \bar{f}_i^i(t) &= \alpha n p \bar{f}_i^i(t-1) + \alpha n q \bar{f}_j^i(t-1) + \sum_{l \neq i,j} \alpha n q \bar{f}_l^i(t-1) \\ \bar{f}_j^i(t) &= \alpha n q \bar{f}_i^i(t-1) + \alpha n p \bar{f}_j^i(t-1) + \sum_{l \neq i,j} \alpha n q \bar{f}_l^i(t-1) \end{aligned}$$

Then,

$$\bar{f}_i^i(t) - \bar{f}_j^i(t) = \alpha n (p-q)(\bar{f}_i^i(t-1) - \bar{f}_j^i(t-1)) > 0 \tag{9}$$

which follows from the induction hypothesis and since $p > q$. This asserts that the claim is true.

Claim 2: *$E[w_l | l \in V_1] > E[w_l | l \in V_i]$ for all $i \neq 1$*

Wlog we prove this for $i = 2$. Note that condition (A3) implies $\frac{p-q}{\sqrt{p}} = \tilde{\Omega}(k/\sqrt{n})$, or $\alpha n(p-q) = \tilde{\Omega}(\sqrt{np})$. Since $p-q = \Theta\left(\frac{\log n}{n^\epsilon}\right)$ we have in equation (9) $\alpha n(p-q) > 1$, therefore the gap $\bar{f}_i^i(t) - \bar{f}_j^i(t)$ is of the same order of $\bar{f}_i^i(t), \bar{f}_j^i(t)$. For $r = \frac{2\log(n^\epsilon/L)}{\log np}$ the expected weights are given by,

$$\begin{aligned} E[w_l|l \in V_1] &= Lpf_1^1(r) + Lqf_2^1(r) + \sum_{j\neq 1,2} Lqf_j^1(r) \\ E[w_l|l \in V_2] &= Lpf_1^2(r) + Lqf_2^2(r) + \sum_{j\neq 1,2} Lqf_j^2(r) \end{aligned}$$

Subtracting the above equations,

$$\begin{aligned} E[w_l|l \in V_1] - E[w_l|l \in V_2] &\overset{(a)}{=} Lpf_1^1(r) + Lqf_2^1(r) - Lpf_1^2(r) - Lqf_2^2(r) \\ &= Lp\bar{f}_1^1(r) + Lq\bar{f}_2^1(r) - Lp\bar{f}_1^2(r) - Lq\bar{f}_2^2(r) \\ &\quad -o(Lp(\bar{f}_1^1(r) + \bar{f}_2^2(r))) \\ &\overset{(b)}{\geq} L(p-q)(\bar{f}_1^1(r) - \bar{f}_2^1(r)) - o(Lp\bar{f}_1^1(r)) \\ &\overset{(c)}{>} 0 \end{aligned}$$

Steps (a), (b) follow from symmetry since $\bar{f}_1^1(r) = \bar{f}_2^2(r)$, and $\bar{f}_i^j(r)$ are all equal for $i \neq j$. Step (c) uses Claim 1. Hence the proof.

Claim 3: *w_i satisfy condition (A2)*

Again for $r = \frac{2\log(n^\epsilon/L)}{\log np}$, and $p - q = \Theta\left(\frac{\log n}{n^\epsilon}\right)$ we have $f_1^1(r) - f_2^1(r) = \Theta(n^\epsilon/L)$. Then,

$$\begin{aligned} \sigma_1(R) - \sigma_2(R) &= E[w_i|i \in V_1] - E[w_i|i \in V_2] \\ &= \Theta(L(p-q)n^\epsilon/L) = \Theta(\log n) \end{aligned}$$

Also using Chernoff bound with high probability $\gamma_2 = O(\sqrt{\log n})$. Now for $p = \Theta\left(\frac{\log n}{n^\epsilon}\right), q = \Theta\left(\frac{\log n}{n^\epsilon}\right)$, $p - q = \Theta\left(\frac{\log n}{n^\epsilon}\right)$, under condition (A3) with $k = \Theta\left(n^{(1-\epsilon)/2}\right)$, (A2) requires γ_2 to satisfy the condition

$$\begin{aligned} \gamma_2 &= O\left((\sigma_1(R) - \sigma_2(R)) \min\left\{\frac{1}{\xi(n)}, \frac{\sqrt{\log n}}{\xi(n)} - 1\right\}\right) \\ &= O\left(\min\left\{\frac{\log n}{\xi(n)}, \frac{(\log n)^{1.5}}{\xi(n)} - \log n\right\}\right) \end{aligned}$$

This is satisfied since $\xi(n) = o(\sqrt{\log n})$ and $\gamma_2 = O(\sqrt{\log n})$. Hence condition (A2) holds with high probability. □