Overlapping Communities in Social Networks

Jan Dreier, Philipp Kuinke, Rafael Przybylski, Felix Reidl, Peter Rossmanith, and Somnath Sikdar

Theoretical Computer Science
RWTH Aachen University, 52074 Aachen, Germany

Abstract

Complex networks can be typically broken down into groups or modules. Discovering this "community structure" is an important step in studying the large-scale structure of networks. Many algorithms have been proposed for community detection and benchmarks have been created to evaluate their performance. Typically algorithms for community detection either partition the graph (non-overlapping communities) or find node covers (overlapping communities).

In this paper, we propose a particularly simple semi-supervised learning algorithm for finding out communities. In essence, given the community information of a small number of "seed nodes", the method uses random walks from the seed nodes to uncover the community information of the whole network. The algorithm runs in time $O(k \cdot m \cdot \log n)$, where m is the number of edges; n the number of links; and k the number of communities in the network. In sparse networks with $m = O(n)$ and a constant number of communities, this running time is almost linear in the size of the network. Another important feature of our algorithm is that it can be used for either non-overlapping or overlapping communities.

We test our algorithm using the LFR benchmark created by Lancichinetti, Fortunato, and Radicchi [15] specifically for the purpose of evaluating such algorithms. Our algorithm can compete with the best of algorithms for both non-overlapping and overlapping communities as found in the comprehensive study of Lancichinetti and Fortunato [13].

1 Introduction

Many real-world graphs that model complex systems exhibit an organization into subgraphs, or *communities* that are more densely connected on the inside than between

each other. Social networks such as Facebook and LinkedIn divide into groups of friends or coworkers, or business partners; scientific collaboration networks divide themselves into research communities; the World Wide Web divides into groups of related webpages. The nature and number of communities provide a useful insight into the structure and organization of networks.

Discovering the community structure of networks is an important problem in network science and is the subject of intensive research [7, 17, 3, 20, 5, 19, 18, 1, 22, 21]. Existing community detection algorithms are distinguished by whether they find partitions of the node set (non-overlapping communities) or node covers (overlapping communities). Typically finding overlapping communities is a much harder problem and most of the earlier community detection algorithms focused on finding disjoint communities. A comparative analysis of several community detection algorithms (both non-overlapping and overlapping) was presented by Lancichinetti and Fortunato in [13]. In this paper we closely follow their test framework, also called the LFR-benchmark.

The notion of a community is a loose one and currently there is no well-accepted definition of this concept. A typical approach is to define an objective function on the partitions of the node set of the network in terms of two sets of edge densities: the density of the edges within a partite set (intra-community edges) and the density of edges across partitions (inter-community edges). The "correct" partition is the one that maximizes this function. Various community detection algorithms formalize this informal idea differently. One of the very first algorithms by Girvan and Newman [7] introduced a measure known as *modularity* which, given a partition of the nodes of the network, compares the fraction of inter-community edges with the edges that would be present had they been rewired randomly preserving the node degrees. Other authors such as Palla *et al.* [19] declare communities as node sets that formed by overlapping maximal cliques. Rosvall and Bergstrom [22] define the goodness of a partition in terms of the number of bits required to describe per step of an infinite random walk in the network, the intuition being that in a "correct" partition, a random walker is likely to spend more time within communities rather than between communities, thereby decreasing the description of the walk.

A severe restriction of many existing community detection algorithms is that they are too slow. Algorithms that optimize modularity typically take $O(n^2)$, even on sparse networks. The overlapping clique finding algorithm of Palla *et al.* [19] take exponential time in the worst case. In other cases, derivation of worst-case running time bounds are ignored.

Our contribution. Given that it is unlikely that users of community detection algorithms would unanimously settle on one definition of what constitutes a community, we feel that existing approaches ignore the *user perspective*. To this end, we chose

to design an algorithm that takes the network structure as well as user preferences into account. The user is expected to classify a small set of nodes of the network into communities (which may be 6–8% of the nodes of each community). Obviously this is possible only when the user has some information about the network, such as its semantics, which nodes are important and into which communities they are classified.

Such situations are actually quite common. The user might have data only on the leading scientific authors in a co-authorship network and would like to find out the research areas of the remaining members of the network. He may either be interested in a broad partition of the network into into its main fields or a fine grained decomposition into various subfields. By labeling the known authors accordingly, the user can specify which kind of partition he is interested in. Another example would be the detection of trends in a social network. Consider the case where one knows the political affiliations of some people and aims to discover political spectrum of the whole network, for example, to predict the outcome of an election.

Another scenario where this may be applicable is in recommendation systems. One might know the preferences of some of the users of an online retail merchant possibly because they purchase items much more frequently than others. One could then use this in the network whose nodes consist of users, with two nodes connected by an edge if they represent users that had purchased similar products in the past. The idea now would be to use the knowledge of the preferences of a few to predict the preferences of everyone in the network.

An important characteristic of algorithms surveyed in [13] is that the algorithms either find disjoint communities or overlapping ones. Most algorithms solve the easier problem of finding disjoint communities. The ones that are designed to find overlapping communities such as the overlapping clique finding algorithm of Palla *et al.* [19] do not seem to yield very good results (see [13]). Our algorithm naturally extends to the overlapping case. Of course, there is a higher price that has to be paid in that the number of nodes that need to be classified by the user typically is larger (5% to 10% of the nodes per community). The algorithm, however, does not need any major changes and we view this is as an aesthetically pleasing feature of our approach.

Thirdly, in many other approaches, the worst-case running time of the algorithms is neither stated nor analyzed. We show that our algorithm runs in time $O(k \cdot m \cdot \log n)$, where k is the number of communities to be discovered (which is supplied by the user), n and m are the number of nodes and edges in the network. In the case of sparse graphs and a constant number of communities, the running time is $O(n \cdot \log n)$. Given that even an $O(n^2)$ time algorithm is too computationally expensive on many real world graphs, a nearly linear time algorithm often is the only feasible solution.

Finally, we provide an extensive experimental evaluation of our algorithm on the LFR benchmark. In order to ensure a fair comparison with other algorithms reviewed

in [13], we choose all parameters of the benchmark as in the original paper.

This paper is organized as follows. In Section 2, we review some of the more influential algorithms in community detection. In Section 3, we describe our algorithm and analyze its running time. In Sections 4 and 5, we present our experimental results. Finally we conclude in Section 6 with possibilities of how our approach might be extended.

2 The Major Algorithms

In what follows, we briefly describe some common algorithms for community detection. We are particularly interested in the performance of these algorithms as reported in the study by Lancichinetti and Fortunato [13] on their LFR benchmark graphs.

The Girvan-Newman algorithm. One of the very first algorithms for detecting disjoint communities was invented by Girvan and Newman [7, 17]. Their algorithm takes a network and iteratively removes edges based on a metric called *edge betweenness*. The betweenness of an edge is defined as the number of shortest paths between vertex pairs that pass through that edge. After an edge is removed, betweenness scores are recalculated and an edge with maximal score is deleted. This procedure ends when the *modularity* of the resulting partition reaches a maximum. Modularity is a measure that estimates the quality of a partition by comparing the network with a so-called "null model" in which edges are rewired at random between the nodes of the network while each node keeps its original degree.

Formally, the *modularity* of a partition is defined as:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \delta(i,j), \tag{1}$$

where A_{ij} represent the entries of the adjacency matrix of the network; d_i is the degree of node i; m is the number of edges in the network; and $\delta(i,j) = 1$ if nodes i and j belong to the same set of the partition and 0 otherwise. The term $d_i d_j / 2m$ represents the expected number of edges between nodes i and j if we consider a random model in which each node i has d_i "stubs" and we are allowed to connect stubs at random to form edges. This is the null model against which the within-community edges of the partition is compared against. The worst-case complexity of the Newman-Girvan algorithm is dominated by the time taken to compute the betweenness scores and is $O(mn)$ for general graphs and $O(n^2)$ for sparse graphs [2].

The greedy algorithm for modularity optimization by Clauset *et al.* [3]. This algorithm starts with each node being the sole member of a community of one, and

repeatedly joins two communities whose amalgamation produces the largest increase in modularity. The algorithm makes use of efficient data structures and has a running time of $O(m \log^2 n)$, which for sparse graphs works out to $O(n \log^2 n)$.

Fast Modularity Optimization by Blondel *et al.* The algorithm of Blondel *et al.* [1] consists of two phases which are repeated iteratively. It starts out by placing each node in its own community and then locally optimizing the modularity in the neighborhood of each node. In the second phase, a new network is built whose nodes are the communities found out in the first phase. That is, communities are replaced by "super-nodes"; the within-community edges are modeled by a (weighted) self-loop to the super-node; and the between-community edges are modeled by a single edge between the corresponding super-nodes, with the weight being the sum of the weights of the edges between these two communities. Once the second phase is complete, the entire procedure is repeated until the modularity does not increase any further. The algorithm is efficient due to the fact that one can quickly compute the change in modularity obtained by moving an isolated node into a community.

Lancichinetti and Fortunato opine that modularity-based methods in general have a rather poor performance, which worsens for larger networks. The algorithm due to Blondel *et al.* performs well probably due to the fact that the estimated modularity is not a good approximation of the real one [13].

The CFinder algorithm of Palla *et al.* One of the first algorithms that dealt with overlapping communities was proposed by Palla *et al.* [19]. They define a community to be a set of nodes that are the union of k-cliques such that any one clique can be reached from another via a series of adjacent k-cliques. Two k-cliques are defined to be adjacent if they share $k-1$ nodes.

The algorithm first finds out all maximal cliques in the graph, which takes exponential-time in the worst case. It then creates a symmetric clique-clique overlap matrix $\boldsymbol{C}$ which is a square matrix whose rows and columns are indexed by the set of maximal cliques in the graph and whose $(i,j)^{\text{th}}$ entry is the number of vertices that are in both the i^{th} and j^{th} clique. This matrix is then modified into a binary matrix by replacing all those entries with value less than $k-1$ by a 0 and the remaining entries by a 1. The final step is to find the connected components of the graph represented by this binary symmetric matrix which the algorithm reports as the communities of the network.

The authors report to have tested the algorithm on various networks including the protein-protein interaction network of *Saccharomyces cerevisiae*[1] with $k = 4$; the co-authorship network of the Los Alamos condensed matter archive (with $k = 6$).

[1] A species of yeast used in wine-making, baking, and brewing.

Lancichinetti and Fortunato report that CFinder did not perform particularly well on the LFR benchmark and that its performance is sensitive to the sizes of community (but not the network size). For networks will small communities it has a decent performance, but has a worse performance on those with larger communities.

Using random walks to model information flow. Rosvall and Bergman [22] approach the problem of finding communities from an information-theoretic angle. They transform the problem into one of finding an optimal description of an infinite random walk in the network. Given a fixed partition M of n nodes into k clusters, Rosvall and Bergman use a two-level code where each cluster is assigned a unique codeword and each node is assigned a codeword which is unique per community. One can now define the average number of bits per step that it takes to describe an infinite random walk on the network partitioned according to M. The intuition is that a random walker is statistically likely to spend more time within clusters than between clusters and therefore the "best" partition corresponds to the one which has the shortest possible description. An approximation of the best partition is found out using a combination of a greedy search heuristic followed by simulated annealing. Lancichinetti and Fortunato report that this algorithm (dubbed Infomap) was the best-performing among all other community detection algorithms on their benchmark.

3 The Algorithm

We assume that the complex networks that we deal with are modeled as connected, undirected graphs. The algorithm receives as input a network and a set of nodes such that there is at least one node from each community that we are aiming to discover. These nodes are called *seed nodes* and it is possible that a particular seed node belongs to multiple communities.

The affinity of a node in the network to a community is 1 if it belongs to it; if it does not belong to it, it has an affinity of 0. We allow intermediate affinity values and view these as specifying a *partial belonging*. The user specifies the affinities of the seed nodes for each of the communities. For all other nodes, called *non-seed nodes*, we want deduce the affinity to each community using the information given by the seed nodes' affinities and the network structure. The main idea is that non-seed nodes should adopt the affinities of seed nodes within their close proximity. We define a proximity measure based on random walks: Each random walk starts at a non-seed node, traverses through the graph, and ends as soon as it reaches a seed node. The affinity of a non-seed node u

for a given community is then the weighted sum of the affinities of the seed nodes for that community and reachable by a random walk starting at u, the weights being the probabilities that a random walk from u ends up at a certain seed node.

Each step of a random walk can be represented as the iterated product of a transition matrix $\boldsymbol{P}$. The result of the (infinite) walk itself can be expressed as $\lim_{k\to\infty} \boldsymbol{P}^k$. One of the contributions of this paper is to show how the calculation of these limits can be reduced to solving a symmetric, diagonally dominant system of linear equations (with different right-hand-sides per community), which can be done in $O(m \log n)$ time, where m is the number of edges in the graph. The fact that such systems can be solved in almost linear time was discovered by Spielman and Teng [23, 6, 24, 11, 12, 25]. If we assume that our networks are sparse in the sense that $m = O(n)$, the running time of our algorithm can be bounded by $O(n \log n)$.

3.1 Absorbing Markov Chains and Random Walks

We now provide a formal description of our model. The input is an undirected, connected graph $G = (V, E)$ with nodes $v_1, \ldots, v_n$, m edges and a nonempty set of s seed nodes. We also know that there are k (possibly overlapping) communities which we want to discover.

The community information of a node v is represented by a $1 \times k$ vector called the *affinity vector* of v, denoted by

$$\boldsymbol{B}(v) = (\alpha(v, 1), \ldots, \alpha(v, k))^{\mathsf{T}}.$$

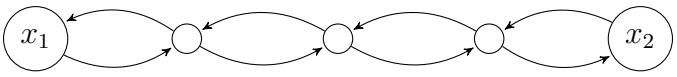
The entry $\alpha(v, l)$ of the affinity vector represents the affinity of node v to community l. It may be interpreted as the probability that a node belongs to this community. We point out that $\sum_{i=1}^{k} \alpha(v, l)$ need not be 1. An example of this situation is when v belongs to multiple communities with probability 1. The user-chosen affinity vectors of all seed nodes are part of the input. The objective is to derive the affinity vectors of all non-seed nodes.

Since we require the random walks to end as soon as they reach a seed node, we transform the undirected graph G into a directed graph G' as follows: replace each undirected edge $\{u, v\}$ by arcs (u, v) and (v, u); then for each seed node, remove its outarcs and add a self-loop. This procedure is illustrated in Figure 1.

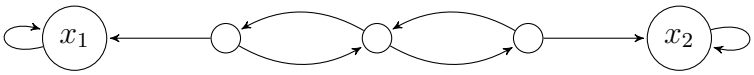
Random walks in this graph can be modelled by an $n \times n$ transition matrix $\boldsymbol{P}$, with

$$\boldsymbol{P}(i, j) = \begin{cases} \frac{1}{\deg_{G'}(v_i)} & \text{if } (v_i, v_j) \in E(G') \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $\deg_{G'}(v)$ is the degree of node v in the directed graph G'. The entry $\boldsymbol{P}(i, j)$ represents the transition-probability from node v_i to v_j. Additionally, $\boldsymbol{P}^r(i, j)$ may be



(a) Example graph with seed nodes x_1, x_2.



(b) Example graph with seed nodes x_1, x_2 after transformation.

Figure 1: Remove outgoing edges and add self-loop for all seed nodes in an example graph. A random walk reaching x_1 or x_2 will stay there forever.

.

interpreted as the probability that a random walk starting at node v_i will end up at node v_j after r steps.

Assume that the nodes of G' are labeled $u_1, \ldots, u_{n-s}, x_1, \ldots, x_s$, where $u_1, \ldots, u_{n-s}$ are the non-seed nodes and $x_1, \ldots, x_s$ are the seed nodes. We can now write the transition matrix $\boldsymbol{P}$ in the following canonical form:

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{R} \\ \mathbf{0}_{s\times(n-s)} & \boldsymbol{I}_{s\times s} \end{bmatrix}, \tag{3}$$

where $\boldsymbol{Q}$ is the $(n-s)\times(n-s)$ sub-matrix that represents the transition from non-seed nodes to non-seed nodes; $\boldsymbol{R}$ is the $(n-s)\times s$ sub-matrix that represents the transition from non-seed nodes to seed nodes. The $s \times s$ identity matrix $\boldsymbol{I}$ represents the fact that once a seed node is reached, one cannot transition away from it. Here $\mathbf{0}_{s\times(n-s)}$ represents an $s \times (n-s)$ matrix of zeros. Since each row of $\boldsymbol{P}$ sums up to 1 and all entries are positive, this matrix is stochastic.

It is well-known that such a stochastic matrix represents what is known as an *absorbing Markov chain* (see, for example, Chapter 11 of Grinstead and Snell [9]). A Markov chain is called absorbing if it satisfies two conditions: It must have at least one absorbing state i, where state i is defined to be absorbing if and only if $\boldsymbol{P}(i,i) = 1$ and $\boldsymbol{P}(i,j) = 0$ for all $j \neq i$. Secondly, it must be possible to transition from every state to some absorbing state in a finite number of steps. It follows directly from the construction of the graph G' and the fact that the original graph was connected, that random walks in G' define an absorbing Markov chain. Here, the absorbing states correspond to the set of seed nodes.

For any non-negative r, one can easily show that:

$$\boldsymbol{P}^r = \begin{bmatrix} \boldsymbol{Q}^r & \sum_{i=0}^{r-1} \boldsymbol{Q}^i \cdot \boldsymbol{R} \\ \mathbf{0}_{s\times(n-s)} & \boldsymbol{I}_{s\times s} \end{bmatrix}. \tag{4}$$

Since we are dealing with infinite random walks, we are interested in the following property of absorbing Markov chains.

Proposition 1. *Let $\boldsymbol{P}$ be the $n \times n$ transition matrix that defines an absorbing Markov chain and suppose that $\boldsymbol{P}$ is in the canonical form specified by equation (3). Then*

$$\lim_{r \to \infty} \boldsymbol{P}^r = \begin{bmatrix} \boldsymbol{0}_{(n-s)\times(n-s)} & (\boldsymbol{I} - \boldsymbol{Q})^{-1} \cdot \boldsymbol{R} \\ \boldsymbol{0}_{s\times(n-s)} & \boldsymbol{I}_{s\times s} \end{bmatrix}. \tag{5}$$

Intuitively, every random walk starting at a non-seed node eventually reaches some seed node where it is "absorbed." The probability that such an infinite random walk starting at non-seed node u_i ends up at the seed node x_j is entry (i, j) of the submatrix $\boldsymbol{X} := (\boldsymbol{I} - \boldsymbol{Q})^{-1} \cdot \boldsymbol{R}$.

Now we can finally define the affinity vectors of non-seed nodes. The affinity of non-seed node u_i to a community l is defined as:

$$\alpha(u_i, l) = \sum_{j=1}^{s} \boldsymbol{X}(i, j) \cdot \alpha(x_j, l). \tag{6}$$

The computational complexity of calculating these affinity values depends on how efficiently we can calculate the entries of $\boldsymbol{X}$, i.e., solve $(\boldsymbol{I} - \boldsymbol{Q})^{-1}$. In the next subsection, we show how to reduce this problem to that of solving a system of linear equations of a special type which takes time $O(m \cdot \log n)$, where m is the number of edges in G.

3.2 Symmetric Diagonally Dominant Linear Systems

An $n \times n$ matrix $\boldsymbol{A} = [a_{ij}]$ is *diagonally dominant* if

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \text{ for all } i = 1, \ldots, n.$$

A matrix is *symmetric diagonally dominant (SDD)* if, in addition to the above, it is symmetric. For more information about matrices and matrix computations, see the textbooks by Golub and Van Loan [8] and Horn and Johnson [10].

An example of a symmetric, diagonally dominant matrix is the graph Laplacian. Given an unweighted, undirected graph G, the *Laplacian* of G is defined to be

$$\boldsymbol{L}_G = \boldsymbol{D}_G - \boldsymbol{A}_G,$$

where $\boldsymbol{A}_G$ is the adjacency matrix of the graph G and $\boldsymbol{D}_G$ is the diagonal matrix of vertex degrees.

A symmetric, diagonally dominant (SDD) system of linear equations is a system of equations of the form:

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b},$$

where $\boldsymbol{A}$ is an SDD matrix, $\boldsymbol{x} = (x_1, \ldots, x_n)^\intercal$ is a vector of unknowns, and $\boldsymbol{b} = (b_1, \ldots, b_n)^\intercal$ is a vector of constants. There is near-linear time algorithm for solving such a system of linear equations and this result is crucial to the analysis of the running time of our algorithm.

The solution of $n \times n$ system of linear equations takes $O(n^3)$ time if one uses Gaussian elimination. Spielman and Teng made a seminal contribution in this direction and showed that SDD linear systems can be solved in nearly-linear time [23, 6, 24]. Spielman and Teng's algorithm (the ST-solver) iteratively produces a sequence of approximate solutions which converge to the actual solution of the system $\boldsymbol{Ax} = \boldsymbol{b}$. The performance of such an iterative system is measured in terms of the time taken to reduce an appropriately defined approximation error by a constant factor. The time complexity of the ST-solver was reported to be at least $O(m \log^{15} n)$ [12]. Koutis, Miller and Peng [11, 12] developed a simpler and faster algorithm for finding ε-approximate solutions to SDD systems in time $\tilde{O}(m \log n \log(1/\varepsilon))$, where the $\tilde{O}$ notation hides a factor that is at most $(\log \log n)^2$. A highly readable account of SDD systems is the monograph by Vishnoi [25]. We summarize the main result that we use as a black-box.

Proposition 2. [12, 25] *Given a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an SDD matrix, there exists an algorithm to compute $\tilde{\boldsymbol{x}}$ such that:*

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{A}} \leq \varepsilon \|\boldsymbol{x}\|_{\boldsymbol{A}},$$

where $\|\boldsymbol{y}\|_{\boldsymbol{A}} := \sqrt{\boldsymbol{y}^\intercal \boldsymbol{A} \boldsymbol{y}}$. The algorithm runs in time $\tilde{O}(m \cdot \log n \cdot \log(1/\varepsilon))$ time, where m is the number of non-zero entries in $\boldsymbol{A}$. The $\tilde{O}$ notation hides a factor of at most $(\log \log n)^2$.

We can use Proposition 2 to upper-bound the time taken to solve the linear systems, which are needed to calculate the affinity vectors defined in (6).

Theorem 1. *Given a graph G, let $\boldsymbol{P}$ be the $n \times n$ transition matrix defined by equation (2) in canonical form (see equation (3)). Then, one can compute the affinity vectors of all non-seed nodes in time $O(m \cdot \log n)$ per community, where m is the number of edges in the graph G.*

Proof. Recall that we ordered the nodes of G as $u_1, \ldots, u_{n-s}, x_1, \ldots, x_s$, where $u_1, \ldots, u_{n-s}$ denote the non-seed nodes and $x_1, \ldots, x_s$ denote seed nodes. Define $G_1 := G[u_1, \ldots, u_{n-s}]$, the subgraph induced by the non-seed nodes of G. Let $\boldsymbol{A}_1$ denote the adjacency matrix of the graph G_1; let $\boldsymbol{D}_1$ denote the $(n-s) \times (n-s)$ diagonal matrix satisfying

$\boldsymbol{D}_1(u_i, u_i) = \deg_G(u_i)$ for all $1 \leq i \leq n - s$. That is, the entries of $\boldsymbol{D}_1$ are not the degrees of the vertices in the induced subgraph G_1 but in the graph G. We can then express $\boldsymbol{I} - \boldsymbol{Q}$ as

$$\boldsymbol{I} - \boldsymbol{Q} = {\boldsymbol{D}_1}^{-1}(\boldsymbol{D}_1 - \boldsymbol{A}_1). \tag{7}$$

Note that $\boldsymbol{D}_1 - \boldsymbol{A}_1$ is a symmetric and diagonally dominant matrix. Let us suppose that $\boldsymbol{X}$ is an $(n - s) \times s$ matrix such that

$$\boldsymbol{X} = (\boldsymbol{I} - \boldsymbol{Q})^{-1} \cdot \boldsymbol{R}.$$

Fix a community l. Then the affinities of the non-seed nodes for community l may be written as:

$$\begin{aligned}
\begin{pmatrix} \alpha(u_1, l) \\ \vdots \\ \alpha(u_{n-s}, l) \end{pmatrix} &= \sum_{j=1}^{s} \alpha(x_j, l) \cdot \boldsymbol{X}_j \\
&= \sum_{j=1}^{s} \alpha(x_j, l)(\boldsymbol{I} - \boldsymbol{Q})^{-1} \cdot \boldsymbol{R}_j \\
&= (\boldsymbol{I} - \boldsymbol{Q})^{-1} \cdot \sum_{j=1}^{s} \alpha(x_j, l) \cdot \boldsymbol{R}_j,
\end{aligned} \tag{8}$$

where $\boldsymbol{X}_j$ and $\boldsymbol{R}_j$ denote the j^{th} columns of $\boldsymbol{X}$ and $\boldsymbol{R}$, respectively. Using equation (7), we may rewrite equation (8) as:

$${\boldsymbol{D}_1}^{-1}(\boldsymbol{D}_1 - \boldsymbol{A}_1) \cdot \begin{pmatrix} \alpha(u_1, l) \\ \vdots \\ \alpha(u_{n-s}, l) \end{pmatrix} = \sum_{j=1}^{s} \alpha(x_j, l) \cdot \boldsymbol{R}_j. \tag{9}$$

Finally, multiplying by $\boldsymbol{D}_1$ on both sides, we obtain

$$(\boldsymbol{D}_1 - \boldsymbol{A}_1) \cdot \boldsymbol{\alpha}_l = \boldsymbol{D}_1 \cdot \sum_{j=1}^{s} \alpha(x_j, l) \cdot \boldsymbol{R}_j, \tag{10}$$

where we used $\boldsymbol{\alpha}_l$ to denote the vector $(\alpha(u_1, l), \ldots, \alpha(u_{n-s}, l))^{\mathsf{T}}$.

Note that computing $\sum_{j=1}^{s} \alpha(x_j, l) \cdot \boldsymbol{R}_j$ takes time $O(\tilde{m})$, where $\tilde{m}$ denotes the number of non-zero entries[2] in $\boldsymbol{P}$. Computing the product of $\boldsymbol{D}_1$ and $\sum_{j=1}^{s} \alpha(x_j, l) \cdot \boldsymbol{R}_j$ takes time $O(\tilde{m})$ so that the right hand side of equation (10) can be computed in time

[2]This is almost the same as the number m of edges in G, but not quite, since while constructing $\boldsymbol{P}$ from the graph G, we add self-loops on seed nodes and delete edges between adjacent seed nodes, if any. However what is true is that $\tilde{m} \leq m + s \leq m + n$.

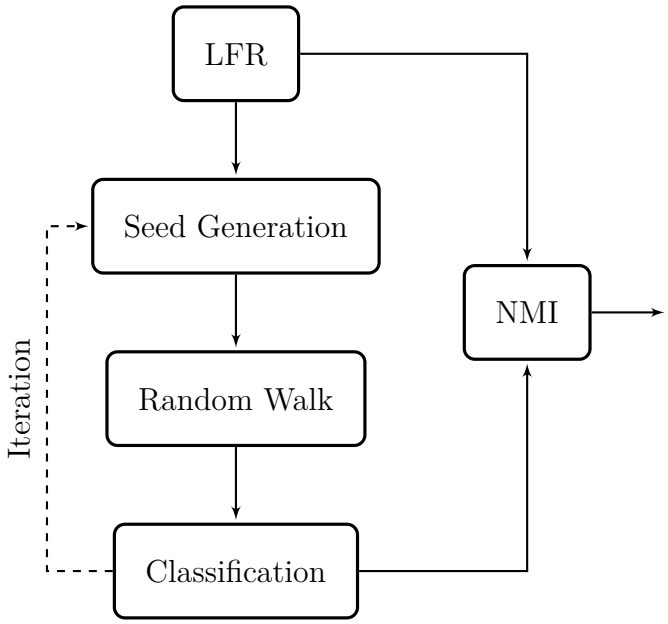


Figure 2: Pipeline

$O(\tilde{m})$. We now have a symmetric diagonally dominant system of linear equations which by Proposition 2 can be solved in time $O(\tilde{m} \cdot \log n)$. Therefore, the time taken to compute the affinity to a fixed community is $O(\tilde{m} \cdot \log n) = O(m \log n)$, which is what was claimed. Since we assume our networks to be sparse, $m = O(n)$, and the time taken is $O(n \cdot \log n)$ per community. □

4 Experimental Setup

Our experimental setup consists of five parts (see Figure 2) but the respective parts differ slightly depending on whether we test non-overlapping or overlapping communities. We use the LFR benchmark graph generator developed by Lancichinetti, Fortunato, and Radicchi [15, 13], which outputs graphs where the community information of each node is known. From each community in the graph thus generated, we pick a fixed number of seed nodes per community and give these as input to our algorithm. Once the algorithm outputs the affinities of all non-seed nodes, we classify them into communities and finally compare the output with the ground truth using normalized mutual information (NMI) as a metric [4]. We implemented our algorithm in C++ and Python and the code is available online.[3]

[3]At `https://github.com/somnath1077/CommunityDetection`

LFR. The LFR benchmark was designed by Lancichinetti, Fortunato and Radicci [15] generates random graphs with community structure. The intention was to establish a standard benchmark suite for community detection algorithms. Using this benchmark they did a comparative analysis of several well-known algorithms for community detection [13]. To the best of our knowledge, this study seems to be the first where standardized tests were carried out on such a range of community detection algorithms. Subsequently, there has been another comprehensive study on overlapping community detection algorithms [26] which also uses (among others) the LFR benchmark. As such, we chose this benchmark for our experiments and set the parameters in the same fashion as in [13].

We briefly describe the major parameters that the user has to supply for generating benchmark graphs in the LFR suite. The node degrees and the community sizes are distributed according to power law, with different exponents. An important parameter is the *mixing parameter* μ which is the fraction of neighbors of a node that do not belong to any community that the node belongs to, averaged over all nodes. The other parameters include maximum node degree, average node degree, minimum and maximum community sizes. For generating networks with overlapping communities, one can specify what fraction of nodes are present in multiple communities.

In what follows, we describe tests for non-overlapping and overlapping communities separately, since there are several small differences in out setup for these two cases.

4.1 Non-overlapping communities

The networks we test have either 1000 nodes or 5000 nodes. The average node degree was set at 20 and the maximum node degree set at 50. The parameter controlling the distribution of node degrees was set at 2 and that for the community size distribution was set at 1. Moreover, we distinguished between big and small communities: small communities have 10–50 nodes and large communities have 20–100 nodes. For each of the four combinations of network and community size, we generated graphs with the above parameters and with varying mixing parameters. For each of these graphs, we tested the community information output by our algorithm and compared it against the ground truth using the normalized mutual information as a metric. The plots in the next section show how the performance varies as the mixing parameter was changed. Each data point in these plots is the average over 100 iterations using the same parameters.

Seed node generation. To use our algorithm, we expect that users pick seed nodes from every community that they wish to identify in the network. We simulate this by picking a fixed fraction of nodes from each community as seed nodes. One of our assumptions is that the user knows the more important members of each community.

To replicate this phenomenon in our experiments, we picked a node as seed node with a probability that is proportional to its degree. That is, nodes with a higher degree were picked in preference to those with a lower degree. For those nodes which were picked as seed nodes, we set the affinity to a community to be 1 if and only if the node belongs to that community and 0 otherwise.

Classification into communities. The input to the algorithm consists of the network, the set of seed nodes together with their affinities. Once the algorithm calculates the affinities of all non-seed nodes, we classify them into their respective communities. This is quite easy for non-overlapping communities where we simply assign each node to the community to which it has the highest affinity, breaking ties arbitrarily.

Iteration. We extended the algorithm to iteratively improve the goodness of the detected communities. The idea is that after running the algorithm once, there are certain nodes which can be classified into their communities with a high degree of certitude. We add these nodes to the seed node set of the respective community and iterate the procedure. To be precise, in the j^{th} round, let C_A^j be the set of nodes that were classified as community A and S_A^j be the seed nodes of community A. We create S_A^{j+1} as follows: For a fixed $\varepsilon > 0$, choose $\varepsilon \cdot |C_A^j|$ nodes of C_A^j that have the highest affinity to community A, and add them to S_A^j to obtain S_A^{j+1}. The factor ε declares by how much the set of seed nodes is allowed to grow in each iteration. Choosing $\varepsilon = 0.1$ gives good results. Repeating this procedure several times significantly improves the quality of the communities detected as measured by the NMI. Each iteration takes $O(k \cdot m \cdot \log n)$ time and hence the cost of running the iterative algorithm is the number of iterations times the cost of running it once.

4.2 Overlapping Communities.

The LFR benchmark suite can generate networks with an overlapping community structure. In addition to the parameters mentioned for the non-overlapping case, there is an additional parameter that controls what fraction of nodes of the network are in multiple communities. As in the non-overlapping case, we generated graphs with 1000 and 5000 nodes with the average node degree set at 20 and maximum node degree set at 50. We generated graphs with two types of community sizes: small communities with 10–50 nodes and large communities with 20–100 nodes. Moreover, as in [13], we chose two values for the mixing factor: 0.1 and 0.3 and we plot the quality of the community structure output by the algorithm (measured by the NMI) against the fraction of overlapping nodes in the network.

Seed Generation. As in the case for non-overlapping communities, we experimented with a non-iterative and an iterative version of our approach. For the non-iterative version, the percentage of seed nodes that we picked were 5, 10, 15 and 20% per community, with the probability of picking a node being proportional to its degree. For the iterative version, we used 2, 4, 6, 8 and 10% seed nodes per community.

Classification into communities. For the overlapping case, we cannot use the naive strategy of classifying a node to a community to which it has maximum affinity, since we do not even know the *number* of communities a node belongs to. We need a way to infer this information from a node's affinity vector.

For each node, we expect the algorithm to assign high affinities to the communities it belongs to and lower affinities to the communities it does not belong to. We tried assigning a node to all communities to which it has an affinity that exceeds a certain threshold. This, however, did not give good results. The following strategy worked better.

Sort the affinities of a node in descending order and let this sequence be $a_1, \ldots, a_k$. Calculate the differences $\Delta_1, \ldots, \Delta_{k-1}$ with $\Delta_{j-1} := a_{j-1} - a_j$; let $\Delta_{\max}$ denote the maximum difference and let i be the smallest index for which $\Delta_i = \Delta_{\max}$. We then associate the node with the communities to which it has the affinities $a_1, \ldots, a_i$. The intuition is that, while the node can have a varying affinity to the communities it belongs to, there is likely to be a sharp decrease in affinities for the communities that the node does not belong to. This is what is captured by computing the difference in affinities and then finding out where the first big drop in affinities occurs.

Iteration. For overlapping communities, we need to extend our strategy for iteratively improving the quality of the communities found. As in the non-overlapping case, after j rounds, we increase the size of the seed node set of community A by a factor ε by adding those nodes which were classified to be in community A and have the highest affinity to this community. Let v be a such a node. The classification strategy explained above might have classified v to be in multiple communities, say, $A_1, \ldots, A_l$. In this case, we assign v to be a seed node for communities $A, A_1, \ldots, A_l$. The running time is the number of iterations times the cost of running the algorithm once.

4.3 Normalized Mutual Information

This is an information-theoretic measure that allows us the compare the "distance" between two partitions of a finite set. Let V be a finite set with n elements and let $\mathcal{A}$ and $\mathcal{B}$ be two partitions of V. The probability that an element chosen uniformly at

random belongs to a partite set $A \in \mathcal{A}$ is n_A/n, where n_A is the number of elements in A. The Shannon entropy of the partition $\mathcal{A}$ is defined as:

$$H(\mathcal{A}) = -\sum_{A \in \mathcal{A}} \frac{n_A}{n} \log_2 \frac{n_A}{n}. \tag{11}$$

The mutual information of two random variables is a measure of their mutual dependence. For random variables X and Y with probability mass functions $p(x)$ and $p(y)$, respectively, and with a joint probability mass function $p(x, y)$, the *mutual information* $I(X, Y)$ is defined as:

$$I(X, Y) = \sum_{x \in \Omega(X)} \sum_{y \in \Omega(Y)} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}, \tag{12}$$

where $\Omega(X)$ is the event space of the random variable X. The mutual information of two partitions $\mathcal{A}$ and $\mathcal{B}$ of the node set of a graph is calculated by using the so-called "confusion matrix" $\boldsymbol{N}$ whose rows correspond to "real" communities and whose columns correspond to "found" communities. The entry $\boldsymbol{N}(A, B)$ is the number of nodes of community A in partition $\mathcal{A}$ that are classified into community B in partition $\mathcal{B}$. The mutual information is defined as:

$$I(\mathcal{A}, \mathcal{B}) = \sum_{A \in \mathcal{A}} \sum_{B \in \mathcal{B}} \frac{n_{A,B}}{n} \log \frac{n_{A,B}/n}{(n_A/n) \cdot (n_B/n)}. \tag{13}$$

Danon *et al.*[4] suggested to use a normalized variant of this measure. The normalized mutual information $I_N(\mathcal{A}, \mathcal{B})$ between partitions $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$$I_N(\mathcal{A}, \mathcal{B}) = \frac{2I(\mathcal{A}, \mathcal{B})}{H(\mathcal{A}) + H(\mathcal{B})}. \tag{14}$$

The normalized mutual information takes the value 1 when both partitions are identical. If both partitions are independent of each other, then $I_N(\mathcal{A}, \mathcal{B}) = 0$.

The classical notion of normalized mutual information measures the distance between two *partitions* and hence cannot be used for overlapping community detection. Lancichinetti, Fortunato, and Kertész [14] proposed a definition of the measure for evaluating the similarity of covers, where a *cover* of the node set of a graph is a collection of node subsets such that every node of the graph is in at least one set. Their definition of normalized mutual information is:

$$\mathrm{NMI}_{\mathrm{LFK}} := 1 - \frac{1}{2} \left(\frac{H(\mathcal{A}|\mathcal{B})}{H(\mathcal{A})} + \frac{H(\mathcal{B}|\mathcal{A})}{H(\mathcal{B})} \right). \tag{15}$$

This definition is not exactly an extension of normalized mutual information in that the values obtained by evaluating it on two partitions is different from what is given by normalized mutual information evaluated on the same pair of partitions. However in this paper we use this definition of NMI to evaluate the quality of the overlapping communities discovered by our algorithm.

We note that McDaid *et al.* [16] have extended the definition of normalized mutual information to covers and that for partitions, their definition corresponds to the usual definition of NMI.

5 Experimental Results

As in the last section, we first discuss our results for the non-overlapping case followed by the ones for the overlapping case.

5.1 Non-overlapping communities

Figures 3, 4, and 5 show the plots that we obtained for non-overlapping communities. Figure 3 shows tests for the non-iterative method of our algorithm with 5, 10, 15, and 20% seed nodes per community.

The first observation here is that anything less than 10% seed nodes per community do not give good results. With a seed node percentage of 10% or more and a mixing factor of at most 0.4 we achieve an NMI above 0.9 and can compete with *Infomap*, which was deemed to be one the best performing algorithms on the LFR benchmark [13]. Above a mixing factor of 0.4, our algorithm has a worse performance than *Infomap* which, curiously enough, achieves an NMI of around 1 till a mixing factor of around 0.6 after which its performance drops steeply. The drop in the performance of our algorithm begins earlier but is not as steep. See Figure 6 for the performance of Infomap and other algorithms that were studied in [13].

Figure 4 shows the results for the iterative approach of our algorithm in the non-overlapping case. When compared with the non-iterative approach, one can see that even after ten iterations there is a significant improvement in performance (See Figure 5). As can be seen, typically with 6% seed nodes per community we obtain acceptable performance (an NMI value of over 0.9 with the mixing factor of up to 0.5).

5.2 Overlapping communities

Figures 7 and 8 show our results for the overlapping case. In the study of Lancichinetti and Fortunato [13], only one algorithm (*Cfinder* [19]) for overlapping communities was

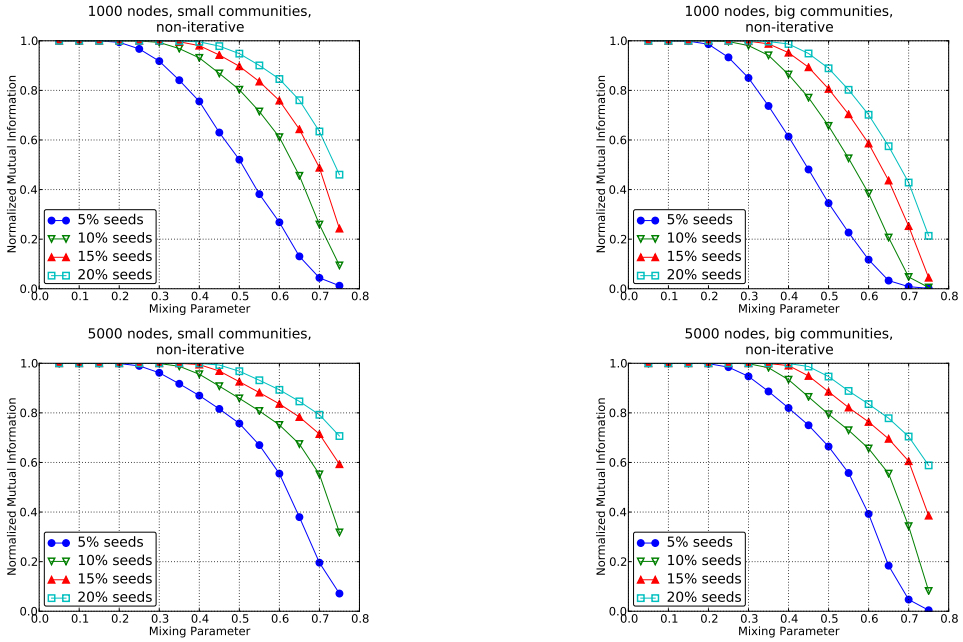


Figure 3: Non-iterative method for non-overlapping communities.

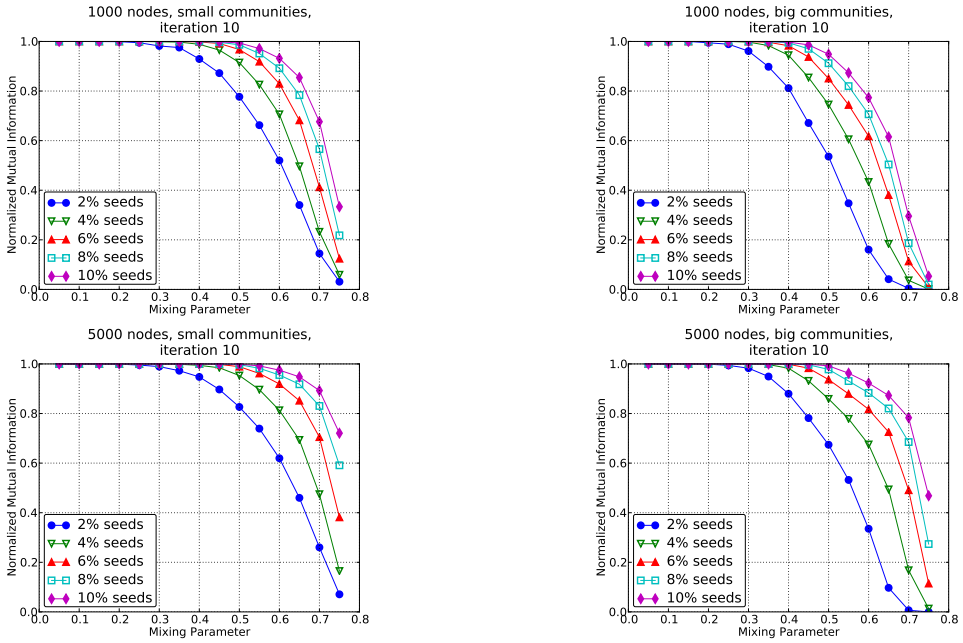


Figure 4: Iterative method for non-overlapping communities.

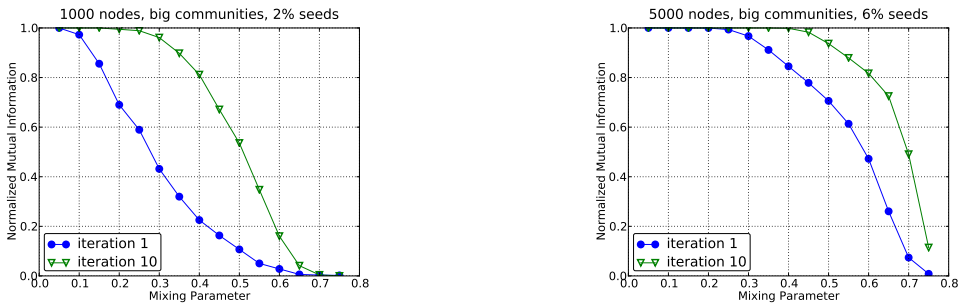


Figure 5: Comparison between the iterative and non-iterative method for non-overlapping communities.

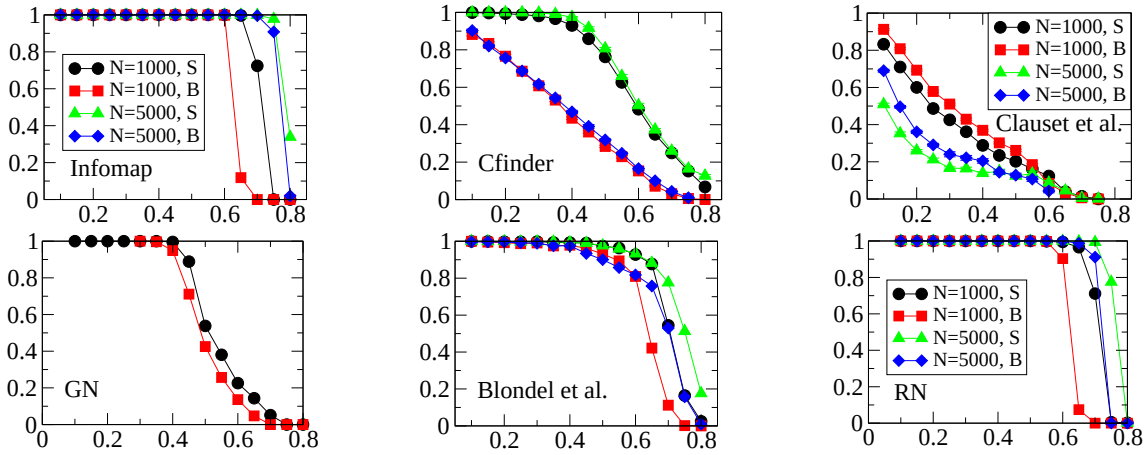


Figure 6: Plots for Infomap, CFinder, the algorithm of Clauset *et al*, Girvan-Newman (GN), Blondel *et al*, and the Pott's model approach by Ronhovde and Nussinov (RN) on the LFR benchmark for non-overlapping communities. As usual, the NMI-value (y-axis) is plotted against the mixing factor (x-axis). Tests were performed on graphs with 1000 and 5000 nodes with big (B) and small (S) communities. Reproduced from [13].

benchmarked (see Figure 12). The main difference with the non-overlapping case is that typically our algorithm needs a larger seed node percentage per community. This is not surprising since in the overlapping case, we would need seed nodes from the various overlaps as well as from the non-overlapping portions of communities to make a good-enough calculation of the affinities.

For graphs of both 1000 and 5000 nodes, our algorithm performs better than Cfinder up to an overlapping fraction of 0.4. We stress that Cfinder has an exponential worst-case running time and would be infeasible on larger graphs.

Figures 9 and 10 show the plots for the iterative method (with 10 iterations). A comparison of the non-iterative and iterative method is shown in Figure 11. Iteration

yields an improvement in performance, as measured by the NMI, but it is not as dramatic as in the non-overlapping case with the NMI increase being at most 10% at best. The percentage of seed nodes per community required in the iterative approach with a mixing factor of 0.3 is around 8%.

6 Concluding Remarks

Our algorithm seems to work very well with around 6% seed nodes for the non-overlapping case and around 8% seed nodes for the overlapping case. For the non-overlapping case, we can work with a mixing factor of up to 0.5, whereas in the overlapping case a mixing factor of 0.3 and with the overlapping fraction of around 20%. This of course suggests that our algorithm has a higher tolerance while detecting non-overlapping communities and needs either a "well-structured" network or a high seed node percentage for overlapping communities. None of this is really surprising. What is surprising is that such a simple algorithm manages to do so well at all.

An obvious question is whether it is possible to avoid the semi-supervised step completely, that is, avoid having the user to specify seed nodes for every community. One possibility is to initially use a clustering algorithm to obtain a first approximation of the communities in the network. The next step would be to pick seed nodes from among the communities thus found (without user intervention) and use our algorithm to obtain a refinement of the community structure.

A possible extension of our algorithm would be to allow the user to interactively specify the seed nodes. The user initially supplies a set of seed nodes and allows the algorithm to find communities. The user then checks the quality of the output and, if dissatisfied with the results, can prompt the algorithm to correctly classify some more nodes that it had incorrectly classified in the current round. In effect, at the end of each round, the user supplies an additional set of seed nodes until the communities found out by the algorithm are accurate enough for the user. Such a tool might be useful for visualization.

We wish to point out that while the running time of our algorithm is $O(k \cdot m \cdot \log n)$, we do not know of any commercial solvers for SDD systems that run in $O(m \cdot \log n)$ time. Since we use the Cholesky factorization method from the C++ Eigen Library, it is unlikely that our implementation would be able to handle very large networks. Recall that in Cholesky factorization, the matrix of coefficients $\boldsymbol{A}$ is decomposed as $\boldsymbol{LDL}^{\intercal}$, where $\boldsymbol{L}$ is lower triangular and $\boldsymbol{D}$ is diagonal, all of which takes $n^3/3$ operations making it prohibitively expensive for large networks (see, for instance [8]). This is not a serious disadvantage since we expect that in the near future we would have commercial SDD solvers implementing the Speilman-Teng algorithm. It would then be interesting to see

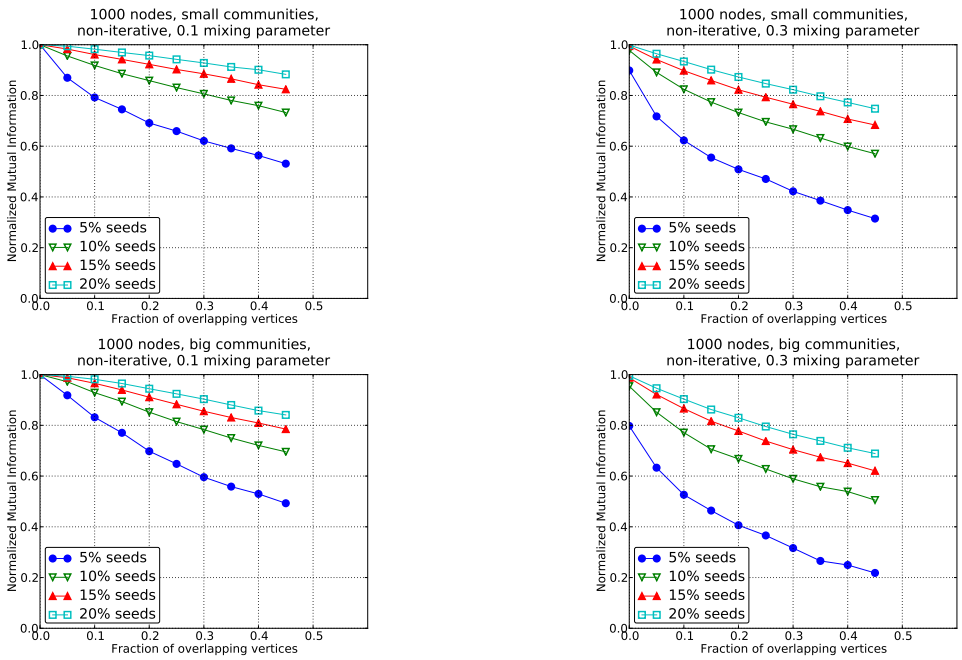


Figure 7: Non-iterative method for overlapping communities on 1000 nodes.

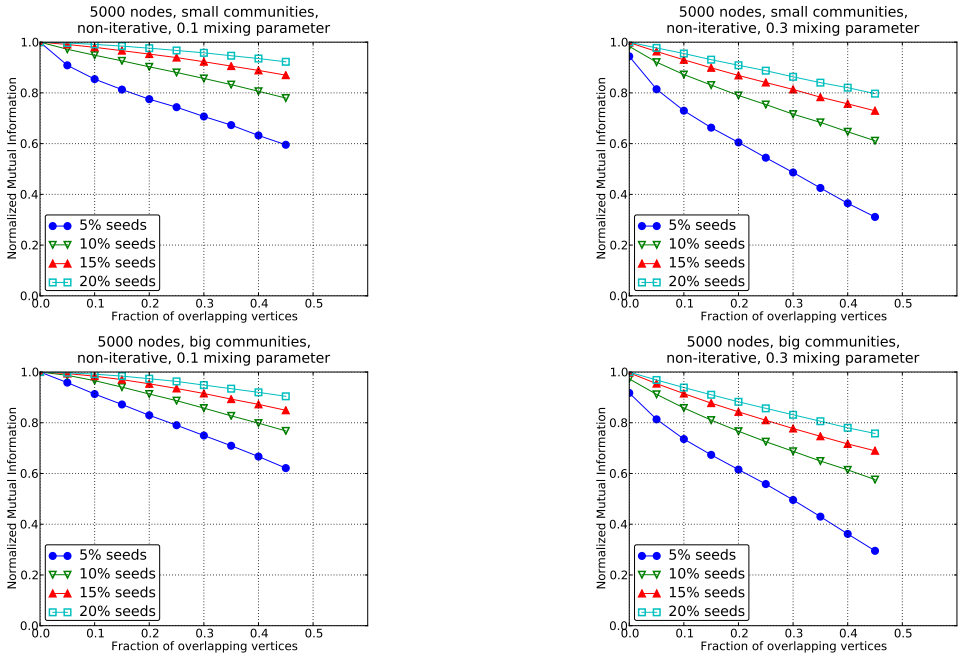


Figure 8: Non-iterative method for overlapping communities on 5000 nodes.

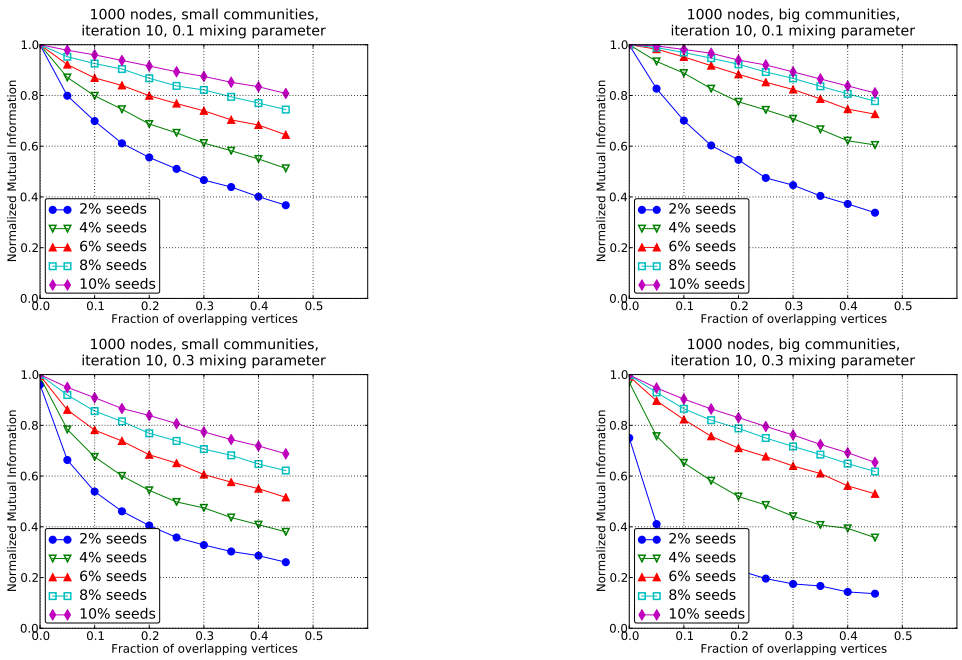


Figure 9: Iterative method for overlapping communities on 1000 nodes.

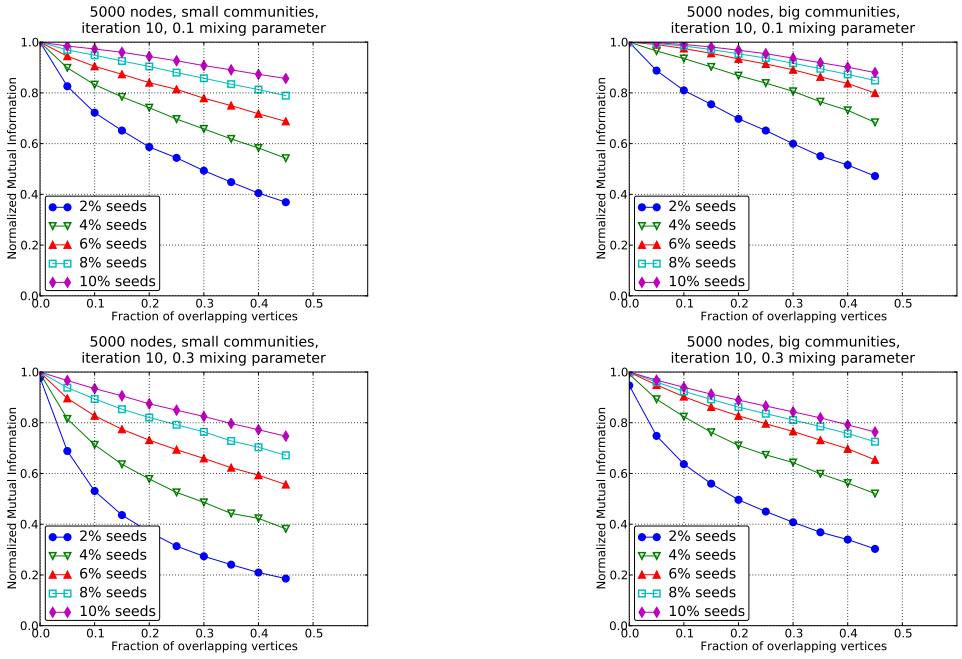


Figure 10: Iterative method for overlapping communities on 5000 nodes.

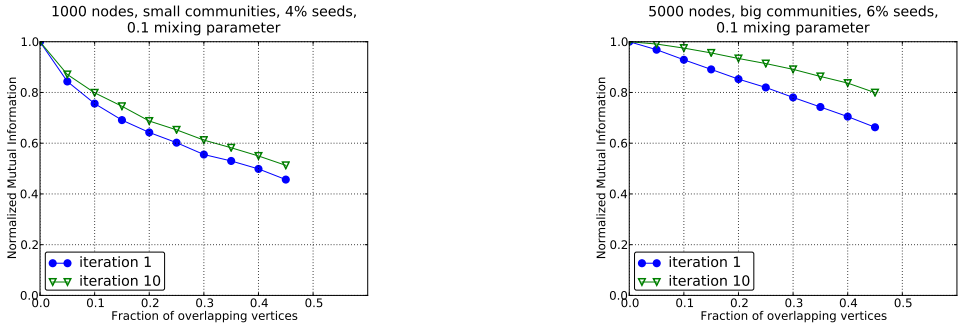


Figure 11: Comparison between the iterative and non-iterative method for overlapping communities.

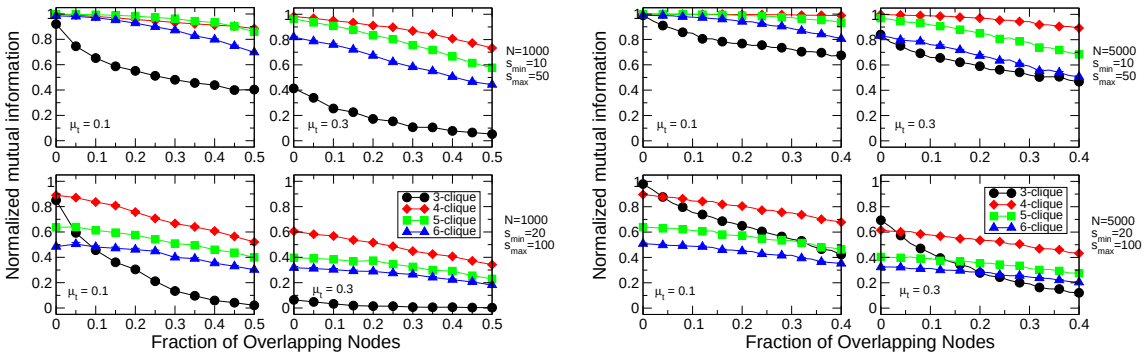


Figure 12: Plots for CFinder on the LFR benchmark on graphs with 1000 and 5000 nodes with overlapping communities. Reproduced from [13].

the size range of real networks our algorithm can handle.

References

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), 2008.

[2] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[3] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70, 2004.

[4] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09), 2005.

[5] L. Donetti and M. A. M. noz. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2004(10), 2004.

[6] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 494–503. ACM, 2005.

[7] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 4th edition, 2013.

[9] C. M. Grinstead and J. L. Snell. *Introduction to Probability*. American Mathematical Society, 1998.

[10] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2013.

[11] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, pages 235–244. IEEE Computer Society, 2010.

[12] I. Koutis, G. L. Miller, and R. Peng. A nearly $O(m \log n)$-time solver for SDD linear systems. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE Computer Society, 2011.

[13] A. Lancichinetti and S. Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80, 2009.

[14] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3), 2009.

[15] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78, 2008.

[16] A. F. McDaid, D. Greene, and N. J. Hurley. Normalized mutual information to evaluate overlapping community finding algorithms. `http://arxiv.org/abs/1110.2515`, 2011.

[17] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69, 2004.

[18] M. E. J. Newman and E. A. Leicht. Mixture models and exploratory analysis in networks. *Proceedings of the National Academy of Sciences*, 104(23), 2007.

[19] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[20] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, 2004.

[21] P. Ronhovde and Z. Nussinov. Multiresolution community detection for megascale networks by information-based replica correlations. *Physical Review E*, 80, 2009.

[22] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[23] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90. ACM, 2004.

[24] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. Available at: `http://arxiv.org/abs/cs/0607105`. Submitted to SIMAX, 2008.

[25] N. K. Vishnoi. *$Lx = b$. Laplacian Solvers and their Algorithmic Applications.* Now Publishers, 2013.

[26] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys*, 45(4):43:1–43:35, 2013.