Community Detection Algorithms

Vedanta Pawar
Electrical and Computer Engineering
Cornell University
Ithaca, U.S.A.
vp273@cornell.edu

***Abstract*—Real networks ranging from biological networks to human social networks tend to display a strong community structure. Communities are important to study because they reveal important information like factions, interests or societal divisions. There are numerous algorithms out there that are used for community detection, having said this identifying a community is an ill-defined problem. There are no universal guidelines that define a community or a metric that helps in measuring the correctness of an algorithm. This paper explains some of the spectral and non-spectral community detection algorithms and tries to show some of the strength and weaknesses of these popular algorithms. Also, the performance of these algorithms was tested on artificial and real-world networks to assess their applicability.**

***Index Terms*—Community detection, network science, graphs**

I. Introduction

Networks are graphs that consist of vertices and edges and are represented by $G(V, E)$ where V represents vertices of a network and E represents the edges present in a network. Networks occur in wide range of places ranging from metabolic networks, gene networks in biology to the internet network and social networks like Facebook. The associated graphs are in general globally sparse but locally dense: there exist groups of vertices, called communities, highly connected between them but with few links to other vertices [1]. This kind of structure reveals important information of network like a network of political books purchased reveals the political ideologies of the buyers.

The concept of networks is hard to define and there is no universal definition that one must be looking for in the network. On top of that there are no guidelines as to how to assess the performance of clustering algorithms. Therefore this open definition allows for diverse range of algorithms using varied techniques to address the problem of community detection [2]. The problem appears to be subtle but detecting communities is generally computationally expensive. The goal of community detection algorithms is to find a group of nodes of interest. So if a small number of nodes in this group exhibit certain information then this information can be extrapolated to the rest of the nodes in that group. Community detection problem is similar to the network partitioning problems in graphs. Network partitioning problems is generally defined as partitioning a network in groups of constant size [3]. This problem is NP-Hard and efficient methods have been developed over the years. On the other hand community detection performs partitioning of the graph in similar nodes and not much information is usually available about the size of the communities or the number of communities.

In this paper we consider many spectral and non-spectral algorithms for performing community detection and their performance, accuracy, complexity and applicability is compared. Due to community detection being a very important yet open problem there is a rich literature out there that tackles this problem. This report being a semester long project does not provide an exhaustive comparison but only focuses on spectral and non-spectral methods.

The contents are organised in 3 main sections. Section II briefly describes the various community detection algorithms and the papers used to study them. Section III explains all these algorithms in a greater detail. Section IV describes the metrics and the benchmarks used to evaluate these algorithms.

II. Definitions and Literature Survey

As described earlier, there is no unique of community. The *classical* view of community is that the density of the edges in the cluster is higher than the density of the edges between the clusters. Therefore, communities are dense sub-graphs that are well separated from each other [2]. The *modern* view of community is slightly more general than the *classical* view. Where the *classical* view relies on counting of edges the *modern* view uses probability of the edges entering and leaving a cluster. Suppose we have sub-graphs of two sizes A and B of different sizes n_A and n_B, where n_A and n_B represent the number of nodes in sub-graph A and B and $n_B \gg n_A$. The network is generated by a model where the edge probability is p_{in} between nodes of the same group and p_{out} is the edge probability between nodes of two different groups, with $p_{in} > p_{out}$. Many modern benchmarks that are used to generate artificial communities rely on the modern view of a community.

This paper uses one such benchmark to generate artificial communities is the LFR benchmark by Lancichinetti et al. [4], more details about this benchmark are mentioned in section IV.

The various community detection algorithms used are divided in sub-classes as spectral algorithms and non-spectral algorithms. The spectral algorithms that are covered in this paper are spectral clustering based on Fiedler vector (covered in class) and two spectral methods for *modularity* maximization by Newman [6] [5]. The term *modularity* is explained in the dedicated section. These three methods use eigenvector

based community detection for determining the communities of the graph.

The non spectral methods considered are Girvan-Newman betweeness centrality method, that is, the number of shortest paths among all pairs of nodes in the network passing through that edge [7]. We also evaluate the performance of a near linear time algorithm called Label Propagation Algorithm based assignment of community labels at different labels at every time-step [3]. Last we also evaluate an algorithm based on random-walk called *Walktrap*, that uses a random walker to find out the underlying community structure of a network [1].

Another algorithm that we consider only for information purpose but we do not simulate it is the deep learning based algorithm that uses an auto-encoder for determining the latent-space representation of a network to determine the communities present [8].

III. Algorithms

A. Spectral Algorithms

1) Newman's Leading eigenvector clustering: The first spectral we consider is that proposed by Newman based on the concept of maximizing the *modularity* [5]. *Modularity Q* is defined by Newman as:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta_{g_i g_j}$$

A_{ij} is the element of the Adjacency matrix **A** indexed by the vertices *i* and *j*. Here, k_i is the degree of the vertex *i* and *m* is the total number of the edges in the graph and $\delta_{g_i g_j}$ is the Kronecker delta, such that g_i represents which group that vertex *i* belongs to group. If *i* and *j* belong to the same community then the Kronecker delta is 1, otherwise 0.
This equation is further written by replacing the Kronecker delta with $\frac{1}{2}(s_i s_j + 1)$ which is 1 if *i* and *j* are in the same group and 0 otherwise :

$$Q = \frac{1}{4m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] (s_i s_j + 1) = \frac{1}{4m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] s_i s_j$$

Noting that $2m = \sum_i k_i = \sum_{ij} A_{ij}$. This can be conveniently written in matrix form as:

$$Q = \frac{1}{m} \mathbf{s}^T \mathbf{B} \mathbf{s}$$

where **s** is the column vector whose elements are the s_i and we have defined a real symmetric matrix **B** with elements:

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

B is called the modularity matrix [5]. The elements of each of its rows and columns sum to zero, so that it always has an eigenvector (1,1,1, . . .) with eigenvalue zero. Now we proceed by writing **s** as a linear combination of the normalized eigenvectors u_i of **B** so that $\mathbf{s} = \sum_{i=1}^{n} a_i \boldsymbol{u}_i$

$$Q = \frac{1}{4m} \sum_i a_i \mathbf{u}_i^T \mathbf{B} \sum_j a_j \mathbf{u}_j = \frac{1}{4m} \sum_{i=1}^{n} (\mathbf{u}_i^T \cdot \mathbf{s})^2 \beta_i$$

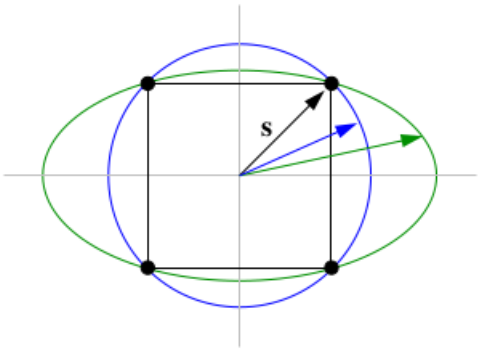

Fig. 1. Geometric representation of the relaxation method employed here. Source: Spectral methods for network community detection and graph partitioning, M. E. J. Newman [6]

where β_i is the eigenvalue of **B** corresponding to eigenvector u_i . In order to maximize the modularity we need to select the largest eigenvalue (leading eigenvalue) and also the signs of the elements in vector **s** must match the signs of the elements in the eigenvector **u** corresponding to the largest eigenvalue. We already know that the elements of **s** indicate the community which the node belongs to. This is the overall spectral based modularity maximization algorithm.

2) Newman's Second Largest Eigenvector clustering: We have seen from the previous algorithm that,

$$Q = \frac{1}{4m} B_{ij} (s_i s_j)$$

Maximizing *Q* is an NP-complete problem, therefore maximization of Q is produced by approximating heuristics. This problem is made easier by reducing the discreteness of s_i to take any real values. However, a minimum constraint must be imposed on s_i to restrict them from growing large. Most common constraints that are applied are that $\sum_i s_i^2 = n$, which limit the s_i to $-\sqrt{n} \leq s_i \leq \sqrt{n}$. In the language of spin models, this would be called a "spherical model". It can be seen as shown in figure 1

If we consider s_i to be the elements of a vector **s** then the allowed values of s_i are between ± 1, restricts the $\sum_i s_i^2 = n$ and is bounded by the hypersphere of $\sqrt{n}$. Note here that in figure 1 the sqaure denotes the hypercube belonging to the *unrelaxed* problem. This paper uses an hyperellipsoidal relaxations as compared to the spherical approach. So instead of a hypersphere touching all the corners of the hypecube the hyperellipsoid touches all the corners of the cube. So the constraint is a bit modified to $\sum_i a_i s_i^2 = \sum_i a_i$. So for $a_i = 1$ for all i then it can be seen that it is reduced to the hypersphere optimization problem. We set $a_i = k_i$ where k_i is the degree of the vertex *i*, therefore the contraint is reduced to,

$$\sum_i k_i s_i^2 = 2m$$

where *m* is the number of edges in the network. Now the original modularity can be solved using Lagrange optimization

as ,

$$\frac{\delta}{\delta s_l}\left[\sum_{ij} B_{ij}s_i s_j - \lambda \sum_i k_i s_i^2\right] = 0$$

Performing the derivatives and rearranging, we find that,

$$\sum_i B_{ij}s_j = \lambda k_i s_i$$

Or in matrix notation as

$$\mathbf{Bs} = \lambda \mathbf{Ds}$$

where **D** is the diagonal matrix with elements equal to the vertex degrees $D_{ii} = k_i$. Since all rows of the modularity matrix **B** sum to zero, there is an eigenvector (1,1,1,1...) at the eigenvalue $\lambda = 0$. This tells us that if $\lambda = 0$ is the highest eigenvalue then the best modularity is achieved by not dividing the network at all—the calculation is telling us that there is no good division of the network into groups, so we should leave it undivided. By re-substituting the value of matrix **B** as $B_{ij} = A_{ij} - \frac{k_i k_j}{4m}$,

$$\mathbf{As} = \mathbf{D}(\lambda \mathbf{s} + \frac{k^T \mathbf{s}}{2m}\mathbf{1})$$

where k is the vector of elements k_i. This is further reduced to,

$$\mathbf{As} = \lambda \mathbf{Ds}$$

Defining $\mathbf{u} = \mathbf{D}^{1/2}\mathbf{s}$ and substituting in the equation above we get,

$$(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{u} = \lambda \mathbf{u}$$

We define $\mathbf{L} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. We find the second largest eigenvector of the symmetric matrix **L** and divide the network based on the signs of the elements of this eigenvector.

B. Girvan-Newman Algorithm

The Girvan–Newman algorithm detects communities by progressively removing edges from the original network. The connected components of the remaining network are the communities. Instead of trying to construct a measure that tells us which edges are the most central to communities, the Girvan–Newman algorithm focuses on edges that are most likely "between" communities [7]. The betweeness centrality *g(v)* of a node *v* is given by the expression [9]:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of paths from node *s* to *t* and $\sigma_{st}(v)$ is the total number of the paths from node *s* to *t* through *v*. Vertex betweenness is an indicator of highly central nodes in networks. For any node i, vertex betweenness is defined as the number of shortest paths between pairs of nodes that run through it. The Girvan–Newman algorithm extends this definition to the case of edges, defining the "edge betweenness" of an edge as the number of shortest paths between pairs of nodes that run along it. If a network contains communities or groups that are only loosely connected by a few inter-group edges, then all shortest paths between different communities must go along one of these few edges. Thus, the edges connecting communities will have high edge betweenness (at least one of them). By removing these edges, the groups are separated from one another and so the underlying community structure of the network is revealed. This is shown in figure 2 where the edges *I* and *II* have a high betweeness centrality score and removing these edges effectively separates the communities.

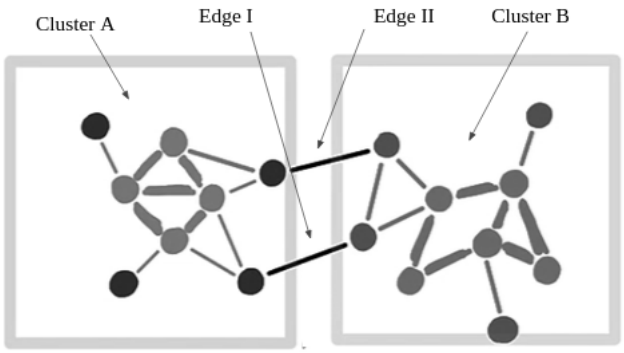

Fig. 2. This figure shows the important edges with high edge betweeness measure. Here *Edge I* and *Edge II* have a high edge betweeness as removing these edges separates the two clusters *A* and *B*

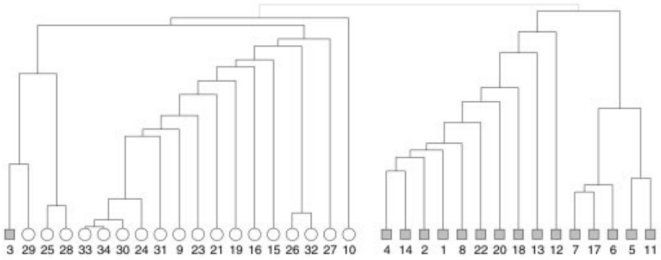

Fig. 3. Dendrogram structure produced on the repeated execution of the Girvan-Newman Algorithm on the Zachary's Karate Club dataset. Source: [7]

The Girvan-Newman algorithm can be summarised as:

1) The betweenness of all existing edges in the network is calculated first.
2) The edge(s) with the highest betweenness are removed.
3) The betweenness of all edges affected by the removal is recalculated.
4) Steps 2 and 3 are repeated until no edges remain.

This is a hierarchical clustering algorithm that produces communities at every step of its execution. This produces a dendrogram structure as shown in figure 3

C. Label Propagation Algorithm

The main idea behind the label propagation algorithm is the following. Suppose that a node x has neighbors x_1, x_2, x_3,..., x_k and that each neighbor carries a label denoting the community to which they belong to. Then x determines its community based on the labels of its neighbors. Is is assumed that each node in the network chooses to join the community

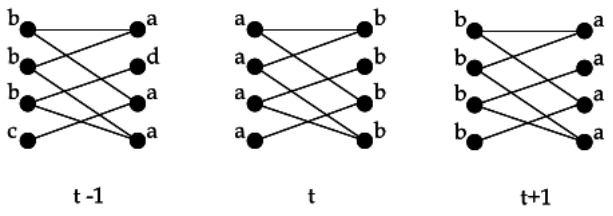

Fig. 4. In this case, due to the choices made by the nodes at step t, the labels on the nodes oscillate between a and b. Source: [3]

to which the maximum number of its neighbors belong to, with ties broken uniformly randomly. Every node is initialized with unique labels and let the labels propagate through the network. As the labels propagate, densely connected groups of nodes quickly reach a consensus on a unique label. When many such dense (consensus) groups are created throughout the network, they continue to expand outwards until it is possible to do so. At the end of the propagation process, nodes having the same labels are grouped together as one community [3].

This process is iterated and at each step, each node updates its label based on the labels of its neighbors. The update process can be synchronous or asynchronous. In synchronous update the node *x* at the t^{th} iteration updates its labels based on the labels of its neighbours at iteration $t-1$. The problem however is that subgraphs in the network that are bi-partite or nearly bi-partite in structure lead to oscillations of labels. This is shown in the figure 4 Hence, asynchronous updating of labels is used, where the labels of the current time step t are used. This process is continued till there are no nodes in the network whose labels are not changing. However, there could be nodes in the network that have an equal maximum number of neighbors in two or more communities. Hence the iterative process is repeated until every node in the network has a label to which the maximum number of its neighbors belong to.

The algorithm can be summarised as:

1) Initialize the labels at all nodes in the network. For a given node *x*, $C_x(0) = x$. Where C_x is the label of node *x* and the parentheses denotes the at time $t = 0$
2) Set $t = 1$
3) Arrange the nodes in the network in a random order and set it to X.
4) For each $x \in X$ chosen in that specific order, let $C_x(t) = f(C_{i1}(t), C_{i2}(t), ..., C_{im}(t), C_{i(m+1)}(t-1), ...C_{i(m+1)}(t-1))$ where $x_{i1}, ..., x_{im}$ are neighbors of x that have already been updated in the current iteration and $x_{i(m+1)}, ..., x_{ik}$ are neighbors that are not yet updated in the current iteration. f returns the label occurring with the highest frequency.
5) If every node has a label that the maximum number of their neighbors have, then stop the algorithm. Else, set $t = t + 1$ and go to (3)

The algorithm begins with each node carrying a unique label, the first few iterations result in various small pockets (dense regions) of nodes forming a consensus (acquiring the same label). These consensus groups then gain momentum and try to acquire more nodes to strengthen the group. However, when a consensus group reaches the border of another consensus group, they start to compete for members. The algorithm converges, and the final communities are identified, when a global consensus among groups is reached. Consequently there is no unique solution and more than one distinct partition of a network into groups satisfies the stop criterion.

D. Walktrap- Random walk based algorithm

Random walks on a graph tend to get "trapped" into densely connected parts corresponding to communities [1]. Here we are going to see some properties of random walks on graphs. Let us consider a discrete random walk process (or diffusion process) on the graph G. At each time step a walker is on a vertex and moves to a vertex chosen randomly and uniformly among its neighbors. The sequence of visited vertices is a Markov chain, the states of which are the vertices of the graph. At each step, the transition probability from vertex *i* to vertex *j* is $P_{ij} = \frac{A_{ij}}{d(i)}$, where $d(i)$ is the degree of vertex *i*. This defines the *transition matrix P* of random walk processes. The process is governed by the powers of the matrix P and the probability of going from the *i* to *j* through a random walk of length *t* is $(P^t)_{ij}$. First we will see two important properties of the random walk.

Property 1: A random walk of length *t* as t goes towards infinity depends only on the degree of the vertex where the walk ends.

$$\forall i, \lim_{t\to\infty} P^t_{ij} = \frac{d(j)}{\sum_k d(k)}$$

Property 2: The probabilities of going from i to j and from j to i through a random walk of a fixed length t have a ratio that only depends on the degrees d(i) and d(j):

$$\forall i, \forall j, d(i)P^t_{ij} = d(j)P^t_{ji}$$

We have seen the basics of a random walk, now we will see a distance *r* to measure the distance vertices. In order to group the vertices in communities this distance *r* must be used. If the distance *r* is large then the two vertices are most likely to be present two different communities. The length *t* of the random walks must be sufficiently long to gather enough information about the topology of the graph. However *t* must not be too long, to avoid the effect predicted by Property 1. The information about vertex i encoded in p^t resides in the *n* probabilities $(P^t_{ik})_{1\leq k\leq n}$ which is the i^{th} row of the matrix *P*.

- If *i* and *j* are in the same communities then the probability of P^t_{ij} is surely high. But it is not a sufficient condition as the probability of the P^t_{ij} is influenced by the degree *d(j)*. Hence, it is not necessary that *i* and *j* are in the same community.
- Two vertices of a same community tend to "see" all the other vertices in the same way. Thus if *i* and *j* are in the same community, we will probably have $\forall k, P^t_{ik} \simeq P^t_{jk}$

Using this information we define the distance *r* between two vertices *i* and *j*. We define the probability P_{Cj} to go from community *C* to vertex *j* in *t* steps:

$$r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}}$$

Now we generalize our distance between vertices to a distance between communities in a straightforward way.

$$r_{C_1C_2} = \sqrt{\sum_{k=1}^{n} \frac{(P_{C_1k}^t - P_{C_2k}^t)^2}{d(k)}}$$

The algorithm is as follows:

1) We start from a partition $\mathbb{P}_0 = \{\{v\}, v \in V\}$ of the graph into n communities. We first compute the distance between all adjacent vertices. Partitions evolves by repeating the following operations. At each step k
2) Choose any two communities C_1 and C_2 in $\mathbb{P}_k$ according to the distance criteria that we mention later.
3) Merge these two communities into a new community C_3 and update $\mathbb{P}_{k+1}$
4) Update the distances between communities

The distance criteria must satisfy Ward's method. At each step *k*, we merge the two communities that minimize the mean σ_k of the squared distances between each vertex and its community.

$$\sigma_k = \frac{1}{n} \sum_{C \in \mathbb{P}_k} \sum_{i \in C} r_{iC}^2$$

This is the overall summary of the Walktrap algorithm.

IV. Validation

Validation means how clustering algorithms can precisely detect communities in benchmarks or in networks where the ground truth community structure is known. The latter case where the community structure being known for large networks is usually rare although research is being conducted in making communities from the metadata that is available. Here, we will see a artificial benchmark called the LFR benchmark and how it is used to generate networks with high degree of community structure.

LFR Benchmark: Lancichinetti–Fortunato–Radicchi benchmark is an algorithm that generates benchmark networks (artificial networks that resemble real-world networks). They have a=priori known communities and are used to compare different community detection methods.
The node degrees and the community sizes are distributed according to a power law, with different exponents. The benchmark assumes that both the degree and the community size have power law distributions with different exponents, α and γ respectively. **N** is the number of nodes and the average degree is $< k >$. There is a mixing parameter μ , which is the average fraction of neighboring nodes of a node that do not belong to any community that the benchmark node belongs to. This parameter controls the fraction of edges that are between communities. Thus, it reflects the amount of noise in the network. At the extremes, when $\mu = 0$ all links are within community links, if $\mu = 1$ all links are between nodes belonging to different communities. There many artificial benchmark generating algorithms like the *l-planted partition model* and the *Girvan-Neman benchmark* but we are only going to use the LFR benchmark because it is able to generate networks that closely resemble real-world networks with heterogeneous degree of the nodes and the community size.

Normalized Mutual Information (NMI): First we will consider two partitions of a network $\mathfrak{X}$ and $\mathfrak{Y}$ of a network G, such that $\mathfrak{X} = (X_1, X_2, ...X_{q_X})$ and $\mathfrak{Y} = (Y_1, Y_2, ...Y_{q_Y})$ with q_X and q_Y clusters respectively. Let *n* be the total number of vertices, let n_i^X and n_j^Y in clusters X_i and Y_j respectively and n_{ij} be the number of vertices shared by the clusters X_i and Y_j. The $q_X \times q_Y$ matrix $N_{\mathfrak{XY}}$ whose entries are the overlaps n_{ij} is called confusion matrix [2]. Similarity can be estimated by computing, given a partition, the additional amount of information that one needs to have to infer the other partition. If partitions are similar, little information is needed to go from one to the other. Such extra information can be used as a measure of dissimilarity. To evaluate the Shannon information content of a partition, we start from the community assignments $\{x_i\}$ and $\{y_i\}$, where x_i and y_i indicate the cluster labels of vertex *i* in partition $\mathfrak{X}$ and $\mathfrak{Y}$ respectively. The labels *x* and *y* are the values of two random variables *X* and *Y* , with joint distribution $P(x, y) = P(X = x, Y = y) = n_{xy}/n$. The mutual information $I(X, Y)$ of two random variables is $I(X, Y) = H(X)H(X|Y)$, where $H(X) = -\sum_x P(x) log P(x)$ is the Shannon entropy of X and $H(X|Y) = -\sum_{x,y} P(x, y) log P(x|y)$ is the conditional entropy of X given Y . The normalized mutual information (NMI), obtained by dividing the mutual information by the arithmetic average of the entropies of $\mathfrak{X}$ and $\mathfrak{Y}$ as :

$$I_{norm}(\mathfrak{X}, \mathfrak{Y}) = \frac{2I(X, Y)}{H(X) + H(Y)}$$

NMI equals 1 if and only if the partitions are identical, whereas it has an expected value of 0 if they are independent. NMI has been regularly used to compute the similarity of partitions in the literature of community detection. There are more sophisticated measures like *variation of information* but we will not be using it in this paper.

Modularity: Modularity Q was defined earlier in Section III. The modularity for a given division of a network is defined to be the fraction of edges within groups minus the expected fraction of such edges in a randomized null model of the network. As before the modularity Q is given by the expression:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta_{g_i g_j}$$

These algorithms were evaluated on numerous real-world and artificial generated benchmark networks. The metric used

for the comparison were described in the previous section as *NMI* and *modularity*. The network datasets used or generated are described below:
Zachary's Karate Club dataset: Consists of 34 nodes and 62 edges. There are tow known communities that karate club split into [10].

Bottlenose Dolphin Dataset: Consists of 62 nodes(dolphins) and 159 edges. It was collected of the coast of New Zealand. Two known communities are present [11].
Political Books Dataset: Consists of nodes 105 nodes and 441 edges. There are three clusters present conservative, liberal and neutral [12].
EU-Mail Dataset: The network was generated using email data from a large European research institution. Consists of 1005 nodes and 25571 edges. The communities were constructed using metadata and there are a total of 42 communities [13].
Artificial Benchmarks: Parameters for generating LFR graph are:
n (int) – Number of nodes in the created graph.
τ_1 – Power law exponent for the degree distribution of the created graph. This value must be strictly greater than one.
τ_2 – Power law exponent for the community size distribution in the created graph. This value must be strictly greater than one.
μ – Fraction of intra-community edges incident to each node. This value must be in the interval [0, 1].
average-degree – Desired average degree of nodes in the created graph. This value must be in the interval [0, n].
min-degree – Minimum degree of nodes in the created graph. This value must be in the interval [0, n].
max-degree (int) – Maximum degree of nodes in the created graph.
min-community – Minimum size of communities in the graph. If not specified, this is set to min-degree

1) *LFR-100* A network of 100 nodes and 3 communities was generated using the LFR benchmark algorithm, with n = 100, τ_1 = 2, τ_2 = 3, μ = 0.8, average-degree = 10, min-degree = 5, max-degree = 40, min-community= 3
2) *LFR-1000* A network of 1000 nodes and 50 communities was generated using the LFR benchmark algorithm, with n = 1000, τ_1 = 2, τ_2 = 3, μ = 0.8, average-degree = 30, min-degree = 25, max-degree = 40, min-community = 50

V. Results

Figure 5 shows the performance of various algorithms as measured against NMI. The higher the NMI the better is the performance. We can see that for a small network like Zachary's karate club with 34 nodes the performance of all the algorithms is very high. Infact the NMI for Newman's second largest eigenvector method, Fiedler vector of the laplacian matrix (taught in class) and the Girvan-Newman algorithm only mis-classify a single node. As we move to the political

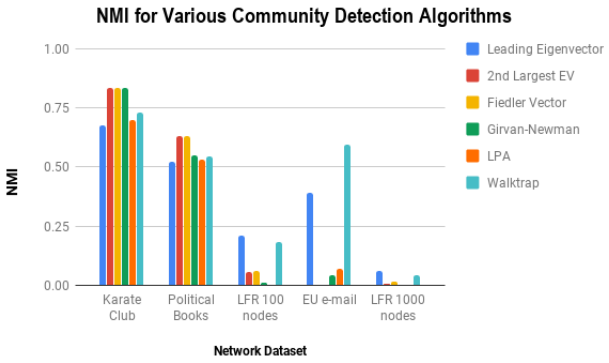

Fig. 5. NMI for various community detection algorithms measured on different datasets

books network which is larger than the previous network, the performance of these algorithms decreases. This is expected because there are more nodes and more communities and there is a higher probability for the nodes to be incorrectly classified. The performance for the EU-mail dataset is further worse for Newman second largest eigenvector algorithm and the Fiedler vector method because these algorithms are made to detect only two communities. Newman's paper does not suggest any solution for extending the algorithm to more than two communities and calls it an "open problem". The Fiedler vector method can be recursively applied but there is no significant improvement in the results. It is important to note that the *Walktrap* algorithm performs very well on this large dataset and the authors of the paper mention its accuracy for very large sparse networks, which most real-world networks are. It is also important to note here that NMI calculations require the ground-truth community labels and many real-world datasets do not have their community affiliation but is generated using meta-data algorithms. Hence, we prefer using artificial benchmarks.
The performance of these algorithms on the LFR benchmark are poor in genral because the LFR was generated for $\mu = 0.8$ which is a very high mixing parameter. Only Newman's leading eigenvector method and Walktrap perform reasonably well on the LFR benchmark for both LFR-100 and LFR-1000. This can be explained visually in the figure 6. It becomes more and more difficult to detect communities and the boundaries between communities become more obscure as the mixing parameter increases.

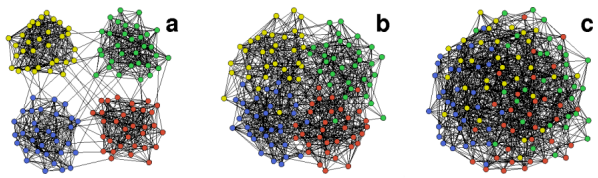

Fig. 6. As the mixing parameter μ increases it becomes difficult for algorithms to identify communities. Source: [2]

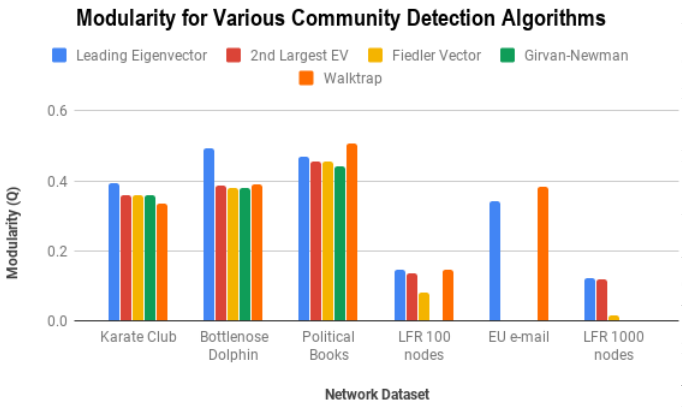

Fig. 7. Modularity for various community detection algorithms measured on different datasets

We also use the modularity metric to evaluate the performance of these algorithms as there is no universal definition of a community, there are not specific guidelines as to which metric to use. The performance of these algorithms as measured against modularity seem to be similar to that as seen in NMI. The figure 7 shows the modularity for various algorithms.

All these algorithms have their strengths and weakness and we will be analysing them in detail here:

Leading Eigenvector method: Newman's leading eigenvector method works well for most dataset from small to medium size. Many papers talk about spectral methods failing at sparse networks we were not able to prove this because of the lack of computing engines. But there is drop in accuracy for EU-network(1000 nodes) and much further drop can be expected for even larger and sparser networks. The runtime of this algorithm is $O(n^3)$ for calculating the eigenvectors and since the network is repeatedly divided and communities found for it should still be $O(n^3)$ as the number of communities is usually $\ll n$. This can be reduced further by using techniques that can calculate eigenvectors more efficiently one such method is the Jacobi one-sided EVD (eigenvector decomposition) using Brent-Luk elimination. We had studied this in our Parallel Computing course (ECE 5720) but I did not use it here because I felt it was out of scope for this project.

Newman's Second largest eigenvector method: This algorithm is very accurate for small networks and even more accurate than the previous spectral algorithm. Major drawback is it is not useful for more than two communities and Newman does not suggest any solution for the same. Additional disadvantage is that for large networks its performance decreases drastically this is primarily because larger networks rarely have only two communities. The runtime of this algorithm is $O(n^3)$, although I expect this algorithm to be much faster than the previous one because it is not repeatedly applied to detect communities.

Fiedler vector method: This was done in class and its advantage is that it works well for small to moderate size networks and is reasonably fast. As the network grows and becomes sparse the accuracy decreases further. Since this algorithm relies on calculating eigenvectors its time complexity is $O(n^3)$ and as I mentioned earlier this can be improved using sophisticated algorithms.

Girvan-Newman Algorithm: This algorithm works well for small datasets and is very accurate. This algorithm is similar to the ratio-cut metric algorithm. Girvan-Newman has time-complexity of $O(nm^2)$(here n is number of nodes and m is number of edges) and hence its performance was not evaluated for large sparse networks. Girvan-Newman was a benchmark for some years and is widely used for comparing many algorithms on standard networks.

Label Propagation Method: Label propagation is an extremely fast and almost linear time algorithm which makes it very versatile and useful. An important drawback of this algorithm is that it produces different results on every execution because the ties in label majority are decided randomly. The accuracy as it is with other algorithms as well tends to decrease with the increase in the network size. The runtime of this algorithm is $O(m+n)$.

Walktrap: Random walk based walktrap is the best algorithm considered so far amongst all the algorithms. It works well moderately well small networks but it out-performs other algorithms for large networks. It is relatively slower than the Label Propagation algorithm and has a runtime of $O(mn(H+t)))$ where H is the height of the dendrogram, t is the length of the walk and m, n have their usual meanings. A disadvantage of this algorithm is that there is a tuning parameter t that needs to be changed.

VI. Conclusion

In this project we have considered various community detection algorithms analysed their strengths and weaknesses. There is an ongoing research in this field as modern networks keep evolving for example there is research ongoing in the area of dynamic clustering because modern day networks are evolving over time and communities evolve and nodes change their affiliation. Also, there is a need for making community detection more computationally easier without hampering accuracy. Also, the performance of these algorithms changes from network to network, most of these algorithms are constructed to be generic in nature and can be applied to any network. This is has the advantage of algorithms being easy to implement from one network to another. On the other hand such algorithms do not capture the essence of the network, an example would be that human social networks are different from biological networks or genetic networks. Therefore, more application specific algorithms need to be considered that can make use of the information of the network at hand.

Acknowledgment

I have made extensive use of networkx and iGraph Python packages which made the development of large graphs easier.

REFERENCES

[1] Pascal Pons and Matthieu Latapy, Computing communities in large networks using random walks

[2] Santo Fortunato and Darko Hric, Community detection in networks: A user guide

[3] Raghavan, U. N., R. Albert, and S. Kumara, 2007, Phys. Rev. E 76(3), 036106.

[4] Lancichinetti, A., S. Fortunato, and F. Radicchi, 2008, Phys.Rev. E 78(4), 046110

[5] M. E. J. Newman, Modularity and community structure in networks.

[6] M. E. J. Newman, Spectral methods for network community detection and graph partitioning

[7] M. Girvan and M. Newman, Proceedings of National Academy of Sciences 99, 7821 (2002).

[8] Liang Yang et al. ,Modularity Based Community Detection with Deep Learning

[9] https://en.wikipedia.org/wiki/Betweenness_centrality

[10] Zachary, W. W., 1977, J. Anthropol. Res. 33, 452.

[11] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. Behavioral Ecology and Sociobiology, 54:396–405, 2003

[12] V. Krebs, unpublished, http:www.orgnet.com

[13] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. ”Local Higher-order Graph Clustering.” In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017.